DIGITAL COMMONS
@ UNIVERSITY OF SOUTH FLORIDA

University of South Florida

Digital Commons @ University of South Florida

USF Tampa Graduate Theses and Dissertations

USF Graduate Theses and Dissertations

March 2021

Strategies in Botnet Detection and Privacy Preserving Machine Learning

Di Zhuang
University of South Florida

Follow this and additional works at: https://digitalcommons.usf.edu/etd

Part of the Computer Engineering Commons, Computer Sciences Commons, and the Electrical and Computer Engineering Commons

Scholar Commons Citation

Zhuang, Di, "Strategies in Botnet Detection and Privacy Preserving Machine Learning" (2021). *USF Tampa Graduate Theses and Dissertations.*
https://digitalcommons.usf.edu/etd/8895

This Dissertation is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact digitalcommons@usf.edu.

Strategies in Botnet Detection and Privacy Preserving Machine Learning

by

Di Zhuang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering
Department of Electrical Engineering
College of Engineering
University of South Florida

Major Professor: Ji-En Morris Chang, Ph.D.
Zhuo Lu, Ph.D.
Xinming Ou, Ph.D.
Kaiqi Xiong, Ph.D.
Yasin Yilmaz, Ph.D.

Date of Approval:
March 16, 2021

Keywords: Network Security, Social Network Analysis, Differential Privacy, Data Mining, Deep Learning

Copyright © 2021, Di Zhuang

Dedication

This dissertation is dedicated to my parents Xiuqing Zhuang and Xiaoping Ma for their unconditional support in all my endeavors. I also dedicate this dissertation to my fiancee Xiaomin Zhao for her encouragements and inspirations.

Acknowledgments

I would like to take this opportunity to express my heartfelt thanks to those who have supported, guided, inspired, and motivated me during my years as a Ph.D. student.

First of all, I would like to thank my major advisor Dr. J. Morris Chang. I am more than grateful that he can give me the opportunity to join his lab as a Ph.D. student at Iowa State University and University of South Florida, and provide all kinds of valuable academic and professional training including developing competitive research proposals, presentation skills, and paper writing during my Ph.D. study. It is his guidance and patience that makes me go through those hard times in research and Ph.D. student life.

The dissertation would not be possible without my committee: Dr. Zhuo Lu, Dr. Xinming Ou, Dr. Kaiqi Xiong, and Dr. Yasin Yilmaz, and the chairperson of my Ph.D. dissertation defense, Dr. Attila A. Yavuz. The knowledge that I have gained from them during coursework or discussions helped enrich my dissertation.

Last but not the least, I would like to express my deepest gratitude to my family and friends. This dissertation would not have been possible without their warm love, continued patience, and endless support.

Table of Contents

List of Tables

List of Figures

Abstract

Peer-to-peer (P2P) botnets have become one of the major threats in network security for serving as the infrastructure that responsible for various of cyber-crimes. Though a few existing work claimed to detect traditional botnets effectively, the problem of detecting P2P botnets involves more challenges. In this dissertation, we present two P2P botnet detection systems, PeerHunter and Enhanced PeerHunter. PeerHunter starts from a P2P hosts detection component. Then, it uses mutual contacts as the main feature to cluster bots into communities. Finally, it uses community behavior analysis to detect potential botnet communities and further identify bot candidates. Enhanced PeerHunter is an extension of PeerHunter, aiming to use network-flow level community behaviors to detect waiting stage P2P botnets, even in the scenario that P2P bots and legitimate P2P applications are running on the same set of hosts. Through extensive experiments with real and simulated network traces, both PeerHunter and Enhanced PeerHunter can achieve very high detection rate and low false positives.

The major component of our P2P botnet detection is a community detection algorithm. Community detection is of great importance for online social network analysis. The volume, variety and velocity of data generated by today's online social networks are advancing the way researchers analyze those networks. For instance, real-world networks, such as Facebook, LinkedIn and Twitter, are inherently growing rapidly and expanding aggressively over time. However, most of the studies so far have been focusing on detecting communities on the static networks. It is computationally expensive to directly employ a well-studied static algorithm repeatedly on the network snapshots of the dynamic networks. We propose DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities of dynamic networks as effective as repeatedly applying static algorithms but in a more efficient way. In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real-world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

In the big data era, many real-world applications, e.g., botnet detection, community detection, image recognition, require to collect a large amount of data from individuals, which involves more privacy concerns. The collected data could be repurposed in different ways, so it could be reused for entirely different purposes by different data users, which were not envisioned at the data collection stage by the data publisher but might jeopardize someone else's privacy. To provide strong privacy guarantees for the collected data and to give the data users greater flexibility in conducting the required data analysis, it is of great importance to enable privacy-enhancing technologies in such analysis. In this dissertation, we present several privacy-enhancing technologies for data mining and machine learning applications, utilizing the concept of dimensionality reduction and differential privacy, including (i) a privacy-preserving facial recognition approach utilizing dimensionality reduction techniques; (ii) a perturbation-based utility-aware privacy-preserving data releasing framework; and (iii) a locally differentially private distributed deep learning framework via knowledge distillation.

Chapter 1: General Introduction

Since the last decades, botnets have become one of the major threats in network security for serving as the fundamental infrastructure for various cyber-crimes, such as distributed denial-of-service (DDoS), email spam, click fraud, etc. For instance, recent botnet attacks including those carried out by WhiskeyAlfa (responsible for Sony Pictures Entertainment attack) and WannaCry (responsible for ransoming healthcare facilities in Europe) showed the scale and scope of damage that botnets can cause.

A botnet is a set of compromised machines controlled by botmasters through command and control (C&C) channels. Botnets may have different communication architectures. Traditional botnets are known to use centralized architectures, which have potential single point of failure. Peer-to-peer (P2P) network is modeled as a distributed architecture, where even if a certain number of peers do not function properly, the whole network is not compromised. Most of the recent botnets (e.g., Storm, Waledac and ZeroAccess) attempted to use P2P architectures, and P2P botnets were proved to be highly resilient even after a certain number of bots being identified or taken down [2]. Therefore, detecting P2P botnets effectively is rather important for securing cyberspace.

However, designing an effective P2P botnets detection systems is rather challenging. First, botnets tend to act stealthily [1] and spend most of their time in the waiting stage before performing any malicious activities [3]. Approaches using malicious activities would have small window of opportunities to detect such botnets. Second, botnets tend to encrypt the C&C channels, causing deep-packet-inspection (DPI) based methods ineffective. Third, the role of a single bot can be changed dynamically depending on the current structure of a botnet [4] (e.g., P2P bot can shift its functionality to act as a botmaster when the prior botmaster has been taken down). Hence, it is difficult to characterize a botnet just by looking at a single bot.

In this report, the main focus is to design and build an effective and efficient P2P botnet detection system relying on the community behavior analysis of the network traffic generated by the suspicious machines. We consider a botnet community as a group of compromised machines that communicate with each other or connect to the same set of botmasters through the same C&C channel, are controlled by the same attacker, and aim to perform similar malicious activities. In the "waiting stage", no malicious activities could be observed. As discussed in [4], the dynamic change of communication behaviors of P2P botnets makes it extremely hard to identify a single bot. Nonetheless, bots belonging to the same P2P botnet always operate together as a community and share the same set of community behaviors.

In chapter 2, we present a novel community behavior analysis based P2P botnet detection system, PeerHunter, which operates under several challenges: (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) no bot-blacklist or "seeds" are available; (d) none statistical traffic patterns known in advance; and (e) do not require to monitor individual host. We propose three types of community behaviors that can be utilized to detect P2P botnets effectively. In the experimental evaluation, we propose a network traces sampling and mixing method to make the experiments as unbiased and challenging as possible. Experiments and analysis have been conducted to show the effectiveness and scalability of our system. With the best parameter settings, our system can achieved 100% detection rate with none false positives.

In chapter 3, we present Enhanced PeerHunter, an extension of PeerHunter [5], aiming to use network-flow level community behaviors to detect waiting stage P2P botnets, even in the scenario that P2P bots and legitimate P2P applications are running on the same set of hosts. We propose two evasion attacks (i.e., passive and active mimicking legitimate P2P application attacks), where we assume the adversaries know our techniques in advance and attempt to evade our system via instructing P2P bots to mimic the behavior of legitimate P2P applications. The experiment results showed that our system is robust to both attacks. We experimented our system using a wide range of parameter settings. With the best parameter settings, our system achieved 100% detection rate with zero false positive.

The major component of our P2P botnet detection is a community detection algorithm. Detecting community structure is of great challenge, and most of the recent studies are proposed to detect communities in the static networks, such as spectral clustering [6], label propagation [7], modularity optimization [8], and k-clique communities [9]. However, real-world networks, especially the botnet network and most of the online social networks, are not static. Most popular online social networks (e.g., Facebook, LinkedIn and Twitter) are de facto growing rapidly and expanding aggressively in terms of either the size or the complexity over time. For instance, in Facebook network, the updating of its community structure could be simply caused by new users joining in, old users leaving, or certain users connecting (i.e., friend) or disconnecting (i.e., unfriend) with the other users. Facebook announced that it had 1.52 billion daily active users in the fourth quarter of 2018 [10], which shows a 9% increase over the same period of the previous year, and 4 million likes generated every minute as of January 2019 [11]. Hence, it is rather important and impending to enable community detection in such dynamic networks. As such, another focus of this report is designing an effective and efficient algorithm to detect communities in (general) dynamic networks.

In chapter 4, we present DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect non-overlapped communities of dynamic networks. DynaMo is an adaptive and incremental algorithm designed for maximizing the modularity gain while updating the community structure of dynamic networks. To update the community structures efficiently, we model the dynamic network as a sequence of incremental network changes. For each incremental network change, we design an operation to maximize the modularity. In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) [12] and 5 dynamic algorithms (i.e., QCA [13], Batch [14], GreMod [15], LBTR-LR [16] and LBTR-SVM [16]). Extensive experiments have been conducted on 6 large-scale real-world networks and 10,000 synthetic networks. Our results show that DynaMo consistently outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

Furthermore, designing a promising botnet detection system usually relies on collecting the vast amount of network related data, which involves more privacy concerns. The collected data could be repurposed in different ways, so it could be reused for entirely different purposes by different data users, which were not envisioned at the data collection stage by the data publisher but might jeopardize someone else's privacy. For instance, the collected network data being used for botnet detection, can also be used to detect or infer the benign applications (e.g. P2P applications, websites) that the data owners are using or running on their hosts, which could be considered as a privacy leakage. To provide strong privacy guarantees for the collected data and to give the data users greater flexibility in conducting the required data analysis, it is of great importance to enable privacy-enhancing technologies in such analysis. As such, in the second half of this report, we focus on designing privacy-enhancing technologies for data mining and machine learning, utilizing the concept of compressive privacy [17, 18] and differential privacy [19].

In chapter 5, we present a privacy-preserving facial recognition approach utilizing dimensionality reduction techniques. These techniques can efficiently transform the raw data from the data owner to a new set of data before they are given to the data users. Without revealing the raw data, the transformation is irreversible. We have implemented our approach in a system called FRiPAL, Face Recognition in Privacy Abstraction Layer, which is a privacy-preserving face recognition service design. RapidGather [20] proposed an architecture of Privacy-Enhanced Android (PE-Android) which is an extension of the current Android OS with new privacy features. One of the most important components in PE-Android is the Privacy Abstraction Layer (PAL), which is defined as a wrapper of the low level PE-Android services that allows the developers to develop privacy preserving applications in their traditional way. FRiPAL has been integrated into RapidGather as a privacy-preserving face recognition service for image data. Extensive experiments have been conducted on three different public datasets to evaluate FRiPAL in terms of accuracy, privacy and efficiency. The accuracy results show that our system maintains the utility for face recognition. The privacy results illustrate that our system protects the privacy which motivates the data owners to submit photos. The efficiency results demonstrate that our system is efficient for practical usage.

In chapter 6, we present a perturbation-based utility-aware privacy-preserving data releasing framework. Given certain specific utility/privacy targets (i.e., the inference problems and the corresponding domain knowledge and public domain data), our approach precisely transforms the original data into privatized data that can be successfully utilized for certain intended purpose (learning to succeed), without jeopardizing certain predefined privacy (training to fail). In the experiments, we have tested our frame on three public datasets: Human Activity Recognition, Census Income and Bank Marketing datasets. The experiment results demonstrate that (a) our approach is a more general utility-aware dimensionality reduction approach compared with DCA [21] and MDR [18]; (b) given certain predefined privacy target, our fine-grained data perturbation approach can reduce the accuracy of the corresponding inference attack to the level of random guessing.

In chapter 7, we present a privacy-preserving distributed deep learning framework, LDP-DL, via local differential privacy [22] and knowledge distillation [23]. Our approach adopts the same "teacher-student" paradigm as described in PATE [24], where each data owner learns a teacher model using its own (local) private dataset, and the data user aims to learn a student model to mimic the output of the ensemble of the teacher models using the unlabelled public data. Knowledge distillation [23] has been applied on the ensemble of the teacher models to enable faster and more accurate knowledge transferring to the student model, and leverage the advantage of having multiple data owners (teacher models). To ensure privacy, our approach employs local differential privacy on the data owners' side, i.e., the query results of each teacher model, which does not require any trusted aggregator (compared to [24]). Since more queries to the teacher models tends to result in more privacy leakage (i.e., cost more privacy budget), we also design an active query sampling approach that could actively select a subset of the unlabelled public dataset for the data user to query from the data owners. In the experimental evaluation, a comprehensive comparison has been made among our proposed approach (i.e., LDP-DL), DP-SGD [25], PATE [24] and DP-FL [26], using three popular deep learning benchmark datasets (i.e., CIFAR10 [27], MNIST [28] and FashionMNIST [29]). The experimental results show that our LDP-DL framework consistently outperforms the other competitors in terms of privacy budget and model accuracy.

Chapter 8 concludes the dissertation's contributions.

Chapter 2: P2P Botnet Detection through Host-level Community Behavior Analysis

Peer-to-peer (P2P) botnets have become one of the major threats in network security for serving as the infrastructure that responsible for various of cyber-crimes. Though a few existing work claimed to detect traditional botnets effectively, the problem of detecting P2P botnets involves more challenges. In this chapter, we present PeerHunter, a community behavior analysis based method, which is capable of detecting botnets that communicate via a P2P structure. PeerHunter starts from a P2P hosts detection component. Then, it uses mutual contacts as the main feature to cluster bots into communities. Finally, it uses community behavior analysis to detect potential botnet communities and further identify bot candidates. Through extensive experiments with real and simulated network traces, PeerHunter can achieve very high detection rate and low false positives. [1]

2.1 Introduction

A botnet is a set of compromised machines controlled by botmaster through a command and control (C&C) channel. Botnets may have different communication architectures. Classical botnets were known to use a centralized architecture, which has a single point of failure. Peer-to-peer (P2P) network happens to be modeled as a distributed architecture, where even though a certain number of peers fail to function properly, the whole network is not compromised. In this case, the most of recent botnets (e.g. Storm, Waledac, ZeroAccess, Sality and Kelihos) attempt to build on P2P network, and P2P botnets have proven to be highly resilient even after a certain number of bots being identified or taken-down [2]. P2P botnets provide a fundamental infrastructure for various cyber-crimes [30], such as distributed denial-of-service (DDoS), email spam, click fraud, etc. Therefore, detecting P2P botnets effectively is rather important for cyber security.

[1] This chapter was published in IEEE Conference on Dependable and Secure Computing 2017 [5]. Copyright permission is included in Appendix A.

However, designing an effective P2P botnets detection system is extremely hard, due to several challenges. First, botnets tend to act stealthily [1] and spend most of their time in the waiting stage before actually performing any malicious activities [31, 3]. Second, botnets tend to encrypt C&C channels, which makes deep-packet-inspection (DPI) based methods fail to work. Third, botnets can randomize their communication patterns dynamically without jeopardizing any primary functions [32, 33, 34], which makes statistical traffic signatures based methods unable to work.

In this chapter, we present PeerHunter, a novel community behavior analysis based P2P botnet detection system, which could address all the challenges above. We consider a botnet community as a group of compromised machines that communicate with each other or connect to the botmaster through the same C&C channel, are controlled by the same attacker, and aim to perform similar malicious activities. Due to the dynamic changes of communication behaviors of P2P botnets [4], it would be extremely hard to identify a single bot. However, bots within the same P2P botnet always work together as a community, thus, have distinct community behaviors to be identified. PeerHunter begins with a general P2P hosts detection component. Then, it builds a mutual contact graph (MCG) of the detected P2P hosts. Afterwards, it applies a community detection method on the MCG, which uses mutual contacts [35] as the main feature of P2P botnets to cluster bots within the same botnet together, and separate bots and legitimate hosts or different types of bots into different communities. Finally, it uses *destination diversity* and *mutual contacts* as the natural features to capture the "P2P behavior" and "botnet behavior" respectively of each P2P botnet community, and further identify all the P2P botnets.

Specifically, PeerHunter is capable of detecting P2P bots with the following challenges and assumptions: (a) botnets are in their waiting stage, which means there is no clear malicious activity can be observed [31]; (b) the C&C channel has been encrypted, so that no deep-packet-inspection (DPI) can be deployed; (c) no bot-blacklist or "seeds" information [35] are available; (d) none statistical traffic patterns [4] known in advance; and (e) could be deployed at network boundary (e.g. gateway), thus, do not require to monitor individual host.

In the experiments, we mixed a real network dataset from a public traffic archive [36] with several P2P botnet datasets and legitimate P2P network datasets [37]. To make the experimental evaluation as unbiased and challenging as possible, we propose a network traces sampling and mixing method to generate synthetic data. We tested our system with 24 synthetic experimental datasets that each contains 10,000 internal hosts. We implemented our P2P hosts detection component using a Map-Reduce framework, which could dramatically reduce the number of hosts subject to analysis by 99.03% and retained all the P2P hosts in our experiments. The Map-Reduce design and implementation of our system could be deployed on popular cloud-computing platforms (e.g. amazon EC2), which ensures the scalability of our system to deal with a big data. With the best parameter settings, our system achieved 100% detection rate with none false positives.

The rest of this chapter is organized as follows: Section 2.2 presents the related works. Section 2.3 explains the motivation and details of the features applied in our system. Section 2.4 describes the system design and implementation details about PeerHunter. Section 2.5 presents the experimental evaluation of PeerHunter. Section 2.6 makes the conclusion.

2.2 Related Work

A few methods attempt to detect P2P botnets have been proposed [38, 3, 37, 1, 35, 33, 39, 34, 40, 41, 4, 42, 43]. Host-level methods have been proposed [42]. However, in host-level methods, all the hosts are required to be monitored individually, which is impractical in real network environments. Network-level methods can be roughly divided into (a) traffic signature based methods, and (b) group/community behavior based methods.

Traffic signature based methods [3, 37, 1, 33, 34, 40, 41, 39] rely on a variety of statistical traffic signatures. For instance, Entelecheia [3] uses traffic signatures to identify a group of P2P bots in a super-flow graph. PeerRush [37] is a signature based P2P traffic categorization system, which can distinguish traffic from different P2P applications, including P2P botnet. Nevertheless, these methods suffer from botnets that have dynamic statistical traffic patterns. Traffic size statistical features can be randomized or modified, since they are only based on the communication protocol design of a botnet. Traffic temporal statistical features can also act dynamically without jeopardizing any primary functions of a botnet.

Group or community behavior based methods [4, 35] consider the behavior patterns of a group of bots within the same P2P botnet community. For instance, Coskun et al. [35] developed a P2P botnets detection approach that start from building a mutual contact graph of the whole

Table 2.1: PeerHunter notations and descriptions.

Notations	Descriptions
MNF	the management network flows
AVGDD	the average # of distinct /16 MNF dstIP prefixes
AVGDDR	the average destination diversity ratio
AVGMC	the average # of mutual contacts between a pair of hosts
AVGMCR	the average mutual contact ratio

Table 2.2: PeerHunter measurements of features.

Trace	AVGDD	AVGDDR	AVGMC	AVGMCR
eMule	8,349	17.6%	3,380	3.7%
FrostWire	11,420	15.2%	7,134	4.5%
uTorrent	17,160	8.7%	13,888	3.5%
Vuze	12,983	10.1%	18,850	7.9%
Storm	7,760	25.1%	14,684	30.2%
Waledac	6,038	46.0%	7,099	37.0%
Sality	9,803	9.5%	72,495	53.2%
Kelihos	305	97.4%	310	98.2%
ZeroAccess	246	96.9%	254	100.0%

network, then attempt to utilize “seeds” (known bots) to identify the rest of bots within the same botnet. However, most of the time, it is hard to have a “seed” in advance. Yan et al. [4] proposed a group-level behavior analysis based P2P botnets detection method. However, they only considered to use statistical traffic features to cluster P2P hosts, which is subject to P2P botnets that have dynamic or randomized traffic patterns. Besides, their method cannot cope with unknown P2P botnets, which is the common case in botnet detection [30], because of relying on supervised classification methods (e.g. SVM).

2.3 Background and Motivation

To demonstrate the features discussed in this section, we conducted some preliminary experiments using dataset shown in Table 3.3 and Table 3.4.Table 3.1 shows the notations and descriptions, and Table 3.2 shows the measurements of features.

2.3.1 P2P Network Characteristics

Due to the decentralized nature of P2P network, a P2P host usually communicates with peers that distributed in a large range of distinct physical networks, which results in the desti-

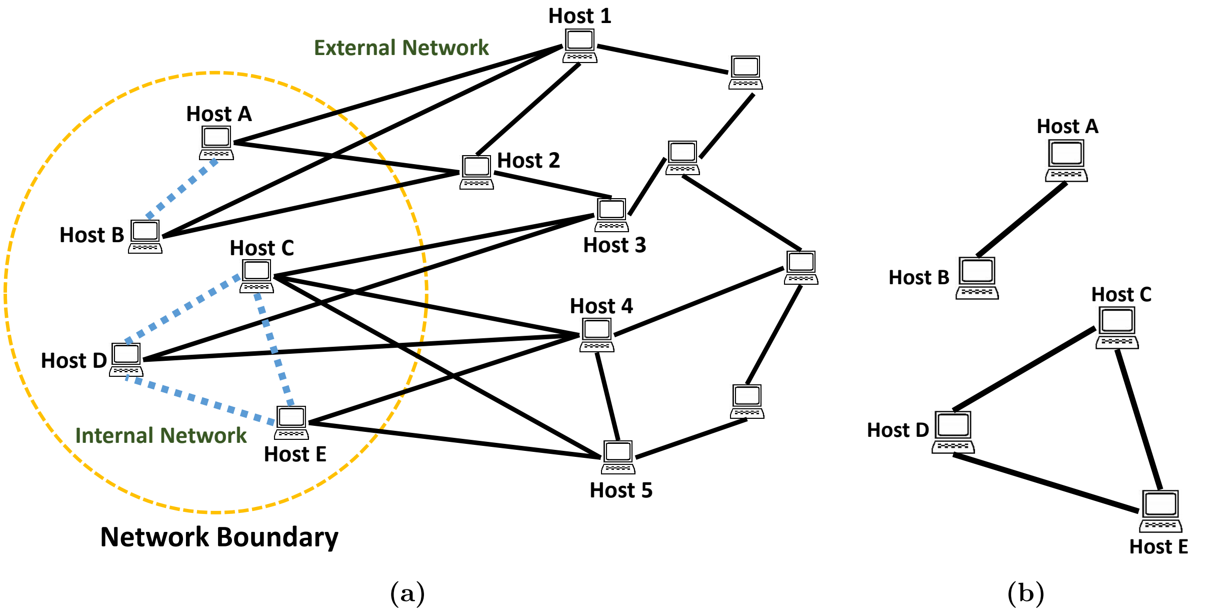


Figure 2.1: Illustration of network (a) and its mutual contact graph (b).

nation diversity (DD) characteristic [37] of P2P management network flow (MNFs). MNF is the network flow for maintaining the function and structure of the P2P network. The P2P network flow mentioned in this section and the rest only refers to P2P MNF.

We use DD as our main feature to detect P2P network flows and further identify P2P hosts. In addition, we use the number of distinct /16 IP prefixes of each host's network flows, rather than BGP prefix used in [1] to approximate DD feature of each P2P host/network flow. /16 IP prefix is a good approximation of network boundaries. For instance, it is very likely that two IP addresses with different /16 IP prefixes belong to two distinct physical networks. This is also supported by Table 3.2, which shows the network flows in a P2P network spreading across a large number of distinct physical networks according to the number of /16 IP prefixes.

2.3.2 Mutual Contacts

The mutual contacts (MC) between a pair of hosts is a set of shared contacts between the corresponding pair of hosts. Consider the network illustrated in Figure 3.1a which contains an internal network (Host A, B, C, D and E) and an external network (Host 1, 2, 3, 4 and 5). A link between a pair of hosts means they have at least one connection. In Figure 3.1a, Host 1, 2 are the mutual contacts shared by Host A, B.

Mutual contacts is the natural characteristic of P2P botnet. Compared with legitimate hosts, a pair of bots within the same P2P botnet has a much higher probability to share a mutual contact [35]. Because bots within the same P2P botnet tend to receive or search for the same C&C messages from the same set of botmasters (peers) [44]. Moreover, in order to prevent peers from churning in a P2P botnet, botmaster has to check each bot periodically, which results in a convergence of contacts among peers within the same botnet [1]. However, since bots from different botnets are controlled by different botmasters, they won't share many mutual contacts. Legitimate host pairs may have a small set of mutual contacts, since nearly all hosts communicate with several extremely popular servers, such as google.com, facebook.com [35]. Furthermore, the host pairs running the same P2P applications may also result in a decent ratio of mutual contacts, if they are accessing the same resource from the same set of peers by coincidence. However, in reality, legitimate P2P hosts with different purposes will not search for the same set of peers. Thus, we can use mutual contacts as a feature to cluster the bots within the same botnet.

The basic idea is to build a mutual contacts graph (MCG) as shown in Figure 3.1, where Host A, B are linked together in Figure 3.1b, since they have mutual contacts Host 1, 2 in Figure 3.1a. Similarly, Host C, D, E are linked to each other in Figure 3.1b, since every pair of them share at least one mutual contacts in Figure 3.1a. More details about MCG is discussed in Section 2.4.2.

2.3.3 Community Behavior Analysis

We consider three types of community behaviors: (a) flow statistical feature, (b) numerical community feature and (c) structural community feature.

2.3.3.1 Flow Statistical Feature

Botnet detection methods using flow statistical features, have been widely discussed [34, 45, 46, 33]. We use the statistical features of P2P MNFs, which are usually generated through the same P2P protocol for a specific P2P application, and some of the statistical patterns of P2P MNFs fully depend on protocols. However, the other network flows, such as data-transfer flows, are usually situation-dependant, which vary a lot even in the same P2P network. In this work, we use the ingoing and outgoing bytes-per-packets (BPP) of network flows in one P2P network as the community flow statistical feature.

2.3.3.2 Numerical Community Feature

We consider two types of features: average destination diversity ratio (AVGDDR) and average mutual contacts ratio (AVGMCR).

The average destination diversity ratio (AVGDDR) captures the “P2P behavior” of P2P botnet communities. The destination diversity (DD) of a P2P host is the number of distinct /16 IP prefixes of each host’s network flows. The destination diversity ratio (DDR) of each host is its DD divided by the total number of distinct destination IPs of its network flows.

Due to the decentralized nature of P2P networks, P2P network flows tend to have higher DDR than non-P2P network flows. Furthermore, network flows from P2P botnet communities usually have higher average DDR (AVGDDR) than network flows from legitimate network communities. Network flows from bots within the same botnet tend to have similar DDR, since those bots are usually controlled by machines, rather than humans. However, the destinations of legitimate P2P network flows are usually user-dependant, which result in their DDR varying greatly from user to user. Besides, our botnet community detection method aims to cluster bots within the same botnets together, rather than clustering the same legitimate P2P hosts together. Legitimate communities might contain both P2P hosts and non-P2P hosts, leading to lower AVGDDR than botnet communities.

Table 3.2 shows the number of distinct destination IP /16 prefixes in MNFs of each type of P2P host, where both legitimate hosts and bots spread across a large number of distinct networks. However, most of the botnets communities have higher AVGDDR than legitimate communities, except Sality. We could combine the next feature to identify Sality.

The average mutual contacts ratio (AVGMCR) captures the “botnet behavior” of P2P botnet communities. The mutual contacts ratio (MCR) between a pair of hosts is the number of mutual contacts between them, divided by the number of total distinct contacts of them. This idea is based on three observations: (a) P2P botnet communities are usually formed by at least two bots, otherwise they cannot act as a group, (b) MCR between a pair of bots within the same botnet is much higher than that between a pair of legitimate hosts or bots from different botnets, and (c) each pair of bots within the same botnet has similar MCR. Thus, we consider the average

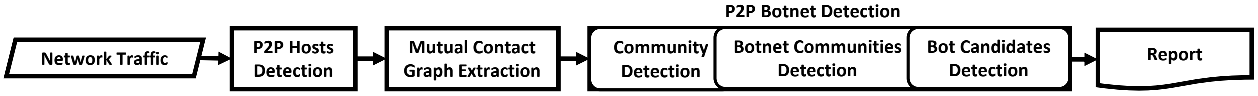


Figure 2.2: PeerHunter system overview.

MCR (AVGMCR) among all pairs of hosts within one network community as another numerical community feature.

Table 3.2 shows the average number of mutual contacts between a pair of hosts within the same community, where both botnets and certain legitimate network communities have a considerable number of mutual contacts. That is because those legitimate communities have much more contacts than botnets. However, botnets has much higher AVGMCR.

2.3.3.3 Structural Community Feature

This captures the structural characteristics of a botnet. The basic idea is that, every pair of bots within the same botnet tends to have a considerable number or ratio of mutual contacts. Therefore, if we consider each hosts as a vertex and link an edge between a pair of hosts if they have a certain amount or ratio of mutual contacts, the bots within the same botnet tend to form certain complete graphes (cliques). On the contrary, the contacts of different legitimate hosts usually tend to diverge into different physical networks. Thus, the probability that legitimate communities form certain cliques is relatively low. Then, we can consider P2P botnets detection as a clique detection problem, which detects cliques from a given network with certain requirements. However, since clique detection problem is NP-complete, we cannot just apply such method to detect botnets. Therefore, we use all three botnet community behaviors.

2.4 System Design

PeerHunter has three components, that work synergistically to (a) detect P2P hosts, (b) construct mutual contact graph, and (c) detect bots. Figure 6.2 illustrates the framework of PeerHunter.

2.4.1 P2P Hosts Detection

This component is responsible for detecting hosts engaged in P2P communications. The input is a 5-tuple network flow [ip_{src}, ip_{dst}, $proto$, bpp_{out}, bpp_{in}], where ip_{src} is source IP, ip_{dst} is destination IP, $proto$ is tcp or udp, and bpp_{out} and bpp_{in} are outgoing and ingoing BPP of network flows. First, we cluster all network flows $F = \{f_1, f_2, \ldots, f_k\}$ based on the 4-tuple [ip_{src}, $proto$, bpp_{out}, bpp_{in}] into flow clusters $FC = \{FC_1, FC_2, \ldots, FC_m\}$. Then, we calculate the number of distinct /16 prefixes of ip_{dst} (destination diversity) associated with each flow cluster, $dd_i = DD(FC_i)$. If dd_i is greater than a pre-defined threshold Θ_{dd}, we consider FC_i as a P2P MNF cluster, and the corresponding source hosts as P2P hosts.

Algorithm 1: P2P Hosts Detection

```
Function Map([$ip_{src}, ip_{dst}, proto, bpp_{out}, bpp_{in}$]):
    $Key \leftarrow [ip_{src}, proto, bpp_{out}, bpp_{in}]$;
    $Value \leftarrow ip_{dst}$;
    return $(Key, Value)$;
Function Reduce($Key$, $Value[\,]$):
    $k \leftarrow Key$;
    $dd_k = \emptyset$;
    for $v \in Value[\,]$ do
        $dd_k \leftarrow dd_k \cup \{v\}$;
    if $|dd_k| \geq \Theta_{dd}$ then
        for $v \in Value[\,]$ do
            return $(k, v)$;
```

As shown in Algorithm 1, we design this component using a MapReduce framework [47]. For a mapper, the input is a set of 5-tuple network flows, and the output is a set of key-value pairs. For a reducer, the input is the set of key-values pairs. Then, the reducer aggregates all values with the same key to calculate the DD of each flow cluster, and finally output the detected P2P MNFs based on Θ_{dd}.

2.4.2 Mutual Contact Graph Extraction

This component is responsible for extracting mutual contact graph (MCG) through mutual contacts. The input is a list of detected P2P hosts, H=$\{h_1, h_2, \ldots, h_{|H|}\}$, and their corresponding P2P MNFs, F=$\{f_1^1, f_2^1, \ldots, f_{n_1}^1, f_1^2, f_2^2, \ldots, f_{n_2}^2, \ldots, f_1^{|H|}, f_2^{|H|}, \ldots, f_{n_{|H|}}^{|H|}\}$, where f_i^j is flow i from

h_j. The output is a MCG, $G_{mc} = (V, E)$, where each vertex $v_i \in V$ contains a DDR score ddr_i of h_i's MNFs, and each edge $e_{ij} \in E$ contains a nonnegative MCR weight mcr_{ij} between h_i and h_j. Algorithm 2 shows the main steps in this component.

First, for each host h_i, we generate a contact set C_i, that contains all the destination IPs in its MNFs. Each host h_i also contains a flow statistical pattern set S_i, which contains all [*proto*, bpp_{out}, bpp_{in}] 3-tuple in its MNFs. Let $DD(C_i)$ be the set of distinct /16 prefixes of all the IPs in C_i. Then, ddr_i and mcr_{ij} can be calculated as below.

$$ddr_i = \frac{\|DD(C_i)\|}{\|C_i\|} \qquad mcr_{ij} = \frac{C_i \cap C_j}{C_i \cup C_j} \tag{2.1}$$

Furthermore, as discussed in Section 2.3.3.1, MNFs from different hosts within the same network communities should have similar statistical patterns. Thus, for each pair of input hosts, say h_i and h_j, we calculate the intersection between S_i and S_j. If $S_i \cap S_j = \emptyset$, then there is no edge between h_i and h_j in MCG. Otherwise, they share at least one MNF statistical pattern, and we calculate mcr_{ij} as shown in (3.1). Let Θ_{mcr} be a pre-defined threshold. Then, if $mcr_{ij} > \theta_{mcr}$, there is an edge between h_i and h_j, with weight mcr_{ij}. Otherwise, there is no edge between h_i and h_j ($mcr_{ij} = 0$).

2.4.3 P2P Botnet Detection

This component is responsible for detecting P2P bots from given MCG. First, we cluster bots into communities. Then, we detect botnet communities using numerical community behavior analysis. In the end, we perform structural community behavior analysis to further identify or verify each bot candidates. Algorithm 3 shows the main steps.

2.4.3.1 Community Detection

In a MCG $G_{mc} = (V, E)$, $\forall\ e_{ij} \in E$, we have $mcr_{ij} \in [0, 1]$, where $mcr_{ij} = 1$ means all contacts of h_i and h_j are mutual contacts and $mcr_{ij} = 0$ means there is no mutual contacts between h_i and h_j. Furthermore, bots within the same botnet tend to have a large number/ratio of mutual contacts. Then, the bots clustering problem can be considered as a classical community detection problem. Various community detection methods have been discussed in [48]. In this work, we utilize

Louvain method, a modularity-based community detection algorithm [12], due to (a) its definition of a good community detection result (high density of weighted edges inside communities and low density of weighted edges between communities) is perfect-suited for our P2P botnet community detection problem; (b) it outperforms many other modularity methods in terms of computation time [12]; and (c) it can handle large network data sets (e.g. the analysis of a typical network of 2 million nodes only takes 2 minutes [12]).

Given $G_{mc} = (V, E)$ as input, Louvain method outputs a set of communities Com=$\{com_1$, com_2, …, $com_{|Com|}\}$, where com_i=(V_{com_i}, E_{com_i}). V_{com_i} is a set of hosts in com_i. E_{com_i} is a set of edges, where $\forall$ $e_{jk} \in E_{com_i}$, we have $e_{jk} \in E$ and $v_j, v_k \in V_{com_i}$.

Algorithm 2: Mutual Contact Graph Extraction

```
Input: $H$, $F$, $\Theta_{mcr}$.
Output: $G_{mc} = (V, E)$.
$E = \emptyset$, $V = \emptyset$;
for $h_i \in H$ do
    $C_i = \emptyset$;
    $S_i = \emptyset$;
for $f_i^j \in F$ do
    $C_j \leftarrow C_j \cup \{ip_{dst}\}$;
    $S_j \leftarrow S_j \cup \{[proto, bpp_{out}, bpp_{in}]\}$;
for $h_i \in H$ do
    $ddr_i \leftarrow \frac{\|DD(C_i)\|}{\|C_i\|}$;
    $vertex\ v_i \leftarrow < ddr_i >$;
    $V \leftarrow V \cup \{v_i\}$;
for $\forall\ h_i, h_j \in H$ $and$ $i < j$ do
    if $S_i \cap S_j \neq \emptyset$ then
        $mcr_{ij} \leftarrow \frac{C_i \cap C_j}{C_i \cup C_j}$;
        if $mcr_{ij} > \Theta_{mcr}$ then
            $edge\ e_{ij} \leftarrow < mcr_{ij} >$;
            $E \leftarrow E \cup \{e_{ij}\}$;
return $G_{mc} = (V, E)$;
```

2.4.3.2 Botnet Communities Detection

Given a set of communities Com, for each community $com_i \in Com$, we start from calculating $avgddr_i$ and $avgmcr_i$, as shown below.

$$avgddr_i = \frac{\sum_{v_j \in V_{com_i}} ddr_j}{\|V_{com_i}\|} \tag{2.2}$$

$$avgmcr_i = \frac{2 \times \sum_{\forall e_{jk} \in E_{com_i}} mcr_{jk}}{\|V_{com_i}\| \times (\|V_{com_i}\| - 1)} \tag{2.3}$$

We define two thresholds Θ_{avgddr} and Θ_{avgmcr}. Then, $\forall$ $com_i \in Com$, if $avgddr_i \geq \Theta_{avgddr}$ and $avgmcr_i \geq \Theta_{avgmcr}$, we consider com_i as a botnet community.

2.4.3.3 Bot Candidates Detection

Recall from Section 2.3.3.3, the MCG of a botnet usually has a structure of one or several cliques. Therefore, we utilize a maximum clique detection method *CliqueDetection* to further identify or verify each bot candidates from botnet communities. Each time it tries to detect one or several maximum cliques on the given MCG of botnet communities. If maximum clique (at least contains 3 vertices) has been found, we consider the hosts in that clique as bot candidates, remove those hosts from the original MCG, and run the maximum clique detection algorithm on the remaining MCG, until no more qualified maximum cliques to be found, then return the set of bot candidates.

Algorithm 3: P2P Botnet Detection

```
Input: $G_{mc}$, $\Theta_{avgddr}$, $\Theta_{avgmcr}$.
Output: $S_{bot}$.
$S_{botnetCom} = \emptyset$, $S_{bot} = \emptyset$;
$Com \leftarrow$ Louvain($G_{mc}$);
for $com_i \in Com$ do
    $avgddr_i \leftarrow \frac{\sum_{v_j \in V_{com_i}} ddr_j}{\|V_{com_i}\|}$;
    $avgmcr_i \leftarrow \frac{2 \times \sum_{\forall e_{jk} \in E_{com_i}} mcr_{jk}}{\|V_{com_i}\| \times (\|V_{com_i}\| - 1)}$;
    if $avgddr_i \geq \Theta_{avgddr}$ and $avgmcr_i \geq \Theta_{avgmcr}$ then
        $S_{botnetCom} \leftarrow S_{botnetCom} \cup \{com_i\}$;
for $com_i \in S_{botnetCom}$ do
    $S_{bot} \leftarrow$ CliqueDetection($com_i$);
return $S_{bot}$
```

Table 2.3: PeerHunter traces of ordinary P2P networks (24 hrs).

Trace	# of hosts	# of flows	# of dstIP	Size
eMule	16	4,181,845	725,367	42.1G
FrostWire	16	4,479,969	922,000	11.9G
uTorrent	14	10,774,924	2,326,626	57.1G
Vuze	14	7,577,039	1,208,372	20.3G

Table 2.4: PeerHunter traces of P2P botnets (24 hrs).

Trace	# of bots	# of flows	# of dstIP	Size
Storm	13	8,603,399	145,967	5.1G
Waledac	3	1,109,508	29,972	1.1G
Sality	5	5,599,440	177,594	1.5G
Kelihos	8	122,182	944	343.9M
ZeroAccess	8	709,299	277	75.2M

Table 2.5: PeerHunter traces of background network.

Date	Dur	# of hosts	# of flows	Size
2014/12/10	24 hrs	48,607,304	407,523,221	788.7G

2.5 Experimental Evaluation

2.5.1 Experiment Setup

2.5.1.1 Experiment Environment

The experiments are conducted on one single PC with an 8 core Intel i7-4770 Processor, 32GB RAM, 400GB SSD and 4TB HHD, and on the 64-bit Ubuntu 14.04 LTS operating system.

2.5.1.2 Data Collection and Analysis Tool

The dataset contains three categories: (a) ordinary P2P network traces, (b) P2P botnets network traces, and (c) background network traces. In practice, all the network traces could be collected at a network boundary (e.g. firewall, gateway, etc.).

We used the dataset obtained from the University of Georgia [37] as our ordinary P2P network traces (D_1), which collected the network traces of 4 different popular P2P applications for several weeks. There are 16 eMule hosts, 16 FrostWire hosts, 14 uTorrent hosts and 14 Vuze hosts, and we randomly selected 24 hours network traces of each host. More details about D_1 are shown in Table 3.3.

Table 2.6: Summaries of PeerHunter experimental datasets (EDs).

Descriptions	Values
the # of EDs	24
the # of bots in each ED	37
the # of ordinary P2P hosts in each ED	60
the # of internal hosts in each ED	10,000
the AVG # of external hosts in each ED	6,607,714
the AVG # of flows in each ED	91,240,099
the duration of each ED	24 hr

Part of our botnet network traces (D_2) is also from the University of Georgia dataset [37], which contains 24 hours network traces of 13 hosts infected with Storm and 3 hosts infected with Waledac. We also collected 24 hours network traces of another three infamous P2P botnets, Sality, Kelihos and ZeroAccess. These network traces were collected from the hosts intentionally infected by Kelihos, ZeroAccess, and Sality binary samples obtained from [49]. Furthermore, all malicious activities have been blocked with the same settings as shown in [37]. We collected the network traces of 8 Kelihos bots, 8 ZeroAccess bots and 5 Sality bots. More details about D_2 are shown in Table 3.4.

We used the dataset downloaded from the MAWI Working Group Traffic Archive [36] as background network traces (D_3), as shown in Table 3.5. This dataset contains 24 hours anonymized network traces at the transit link of WIDE (150Mbps) to the upstream ISP on 2014/12/10 (sample point F). This network traces contains approximate 407,523,221 flows and 48,607,304 unique IPs. 79.3% flows are TCP flows and the rest are UDP flows. We utilize ARGUS [50] to process and cluster network traces into the 5-tuple format tcp/udp flows.

2.5.1.3 Experimental Dataset Generation

To evaluate our approach, we generate 24 experimental datasets by mixing the network traces from D_1 and D_2 into different sub-datasets of D_3. Table 3.7 illustrates the summaries of experimental datasets (EDs). Each experimental datasets contains 10,000 internal hosts sampled from D_3, where the network traces of 37 randomly selected hosts are mixed with D_2, and the network traces of another 60 randomly selected hosts are mixed with D_1. To make the experimental evaluation as unbiased and challenging as possible, below we propose two criterions.

First, we need to maintain a bipartite network structure. Our system aims to deploy at a network boundary (e.g. firewall, gateway, etc.), where the network forms a bipartite structure, and only network flows within the connections between internal hosts and external hosts could be captured. Then, the network in each experimental dataset should maintain a bipartite network structure, where any pair of internal hosts should not have any communications to each other.

Then, we need to keep the connectedness of mutual contacts graph. The easiest way to obtain a list of background hosts is to sample the hosts randomly from D_3, with the respect of bipartite structure. However, since D_3 contains an extremely large number of hosts, simply sampling hosts randomly will result in that most of the sampled background hosts do not have a mutual contact with the other background hosts, which is much easier for PeerHunter to identify botnet communities. Because less number of mutual contacts among legitimate hosts means more disconnected legitimate communities in the corresponding MCG, which is in favor of Louvain method to detect strongly connected botnet communities. Therefore, we need to sample a list of internal hosts in a way that every internal host should have at least one mutual contact with at least one another internal host.

To follow the criterions described above without making our evaluation tasks any easier, we propose the following experimental dataset generation procedure:

• Utilize a two-coloring approach to sample the network traces of 10,000 background hosts from D_3 without jeopardize the bipartite network structure and the connectedness of mutual contacts graph: (a) initialize two counters, C_{black} and C_{white}, to count the number of hosts colored in black and white respectively; (b) coloring a random host h_i as black, and C_{black} plus one; (c) coloring all contacts of h_i as white, and increase C_{white} by the number of hosts colored as white in this round; (d) for each new colored host, color its contacts with the opposite color, and adjust the counters repeatedly, until we have $C_{black} \geq 10,000$ and $C_{white} \geq 10,000$; (e) select the colored host set with exactly 10,000 hosts as the internal hosts, the hosts in the other colored host set will be the external hosts; and (f) extract the network traces of the 10,000 internal hosts from D_3. Then, it forms a bipartite graph, where each colored host set forms a bipartite component, and each host shares at least one mutual contacts with some other hosts from its own bipartite component.

• To maintain a bipartite network structure of botnets and ordinary P2P network traces, we eliminate all communications among bots in D_2 and legitimate P2P hosts in D_1.

Table 2.7: PeerHunter detection rate and false positive rate for different θ_{dd}.

θ_{dd}	**DR**	**FP**	θ_{dd}	**DR**	**FP**
2-10	97/97	$\geq$ 450/9,903	500-1,000	81/97	0
15	97/97	$\geq$ 8/9,903	5,000	60/97	0
20-25	97/97	$\leq$ 1/9,903	10,000	18/97	0
30-185	97/97	0	12,500	5/97	0
200	89/97	0	13,500	0	0

- To mix D_1 and D_2 with D_3, each time we randomly select 97 internal hosts from one sub-datasets sampled from D_3, map those IPs to 37 bots' IP in D_2 and 60 legitimate P2P hosts' IP in D_1, and merge the corresponding network traces.

To evaluate our system, 24 synthetic experimental datasets have been created by running this procedure repeatedly.

2.5.2 Evaluation on P2P Host Detection

We evaluate the P2P host detection with different parameter settings. This component uses a pre-defined threshold θ_{dd} (Section 2.4.1) to detect P2P hosts. We applied this component on all 24 experimental datasets, and Table 3.8 shows the experimental results with different θ_{dd}, ranging from 2 to 13500. If θ_{dd} is set too small, non-P2P hosts are likely to be detected as P2P hosts, which results in many false positives. For instance, when $2 \leq \theta_{dd} \leq 10$, there are, on average more than 450 non-P2P hosts have been falsely identified as P2P hosts. In contrast, if θ_{dd} is set too large, all P2P hosts will be removed, which results in false negatives. For instance, when $\theta_{dd} = 5000$, there are, on average 37 P2P hosts have been falsely discarded, and when $\theta_{dd} \geq 12000$, nearly all hosts are removed. When $20 \leq \theta_{dd} \leq 185$, it detects all P2P hosts with a very small number of false positives ($\leq 1/9903$), which demonstrates that our P2P hosts detection component is stable and effective over a large range of θ_{dd} settings. The effectiveness of θ_{dd} is also subject to the time window of the collected data. In our experiment, we used 24 hrs network traces. The destination diversity (DD) of P2P hosts tends to grow over time. Then, θ_{dd} will be effective in a even larger range, if the time window increase.

Table 2.8: PeerHunter community detection results for different Θ_{mcr}.

Θ_{mcr}	**FLCR**	**FBCR**	**FBSR**
0-0.25	0	0	0
0.5	0	0	2.8
1.0	0	0	6.4

2.5.3 Evaluation on Community Detection

We evaluate the performance of community detection with different parameter settings. We applied this component on the remain network flows (24 experimental datasets) after the P2P host detection (with $\theta_{dd} = 50$). For each experimental dataset, this component generates a MCG $G_{mc} = (V, E)$ with a pre-defined threshold Θ_{mcr}, where each edge $e_{ij} \in E$ contains a weight $mcr_{ij} \in [0, 1]$. Then, we applied Louvain method (with default resolution 1.0) on the MCG for community detection. The choice of Θ_{mcr} has an influence on the community detection results.

We evaluated the community detection performance in terms of (a) the ability to separate bots and legitimate hosts, (b) the ability to separate bots from different botnets, and (c) the ability to cluster bots within the same botnet. Let $falsely$*-clustered hosts* denote the number of legitimate hosts that have been clustered with bots into the same community, *cross-community bots* denote the number of bots of different types that have been clustered into the same community, and *split-communities botnets* denote the number of detected communities that contain bots, subtract the number of ground truth botnets (e.g. 5 in our experiments). Then, we propose three evaluation criterions: (a) False Legitimate Cluster Rate (FLCR), which is $falsely$*-clustered hosts* divided by the total number of legitimate hosts during community detection; (b) False Bot Cluster Rate (FBCR), which is *cross-community bots* divided by the total number of bots during community detection; (c) False Botnet Split Rate (FBSR), which is *split-communities botnets* divided by the total number of ground truth botnets.

Table 3.9 shows the results with different Θ_{mcr}, ranging from 0 to 1. If Θ_{mcr} is set too small, there will be more non-zero weight edges, which might result in less but larger communities. In contrast, if Θ_{mcr} is set too large, most of the vertices will be isolated, which results in more but small communities. As shown in Table 3.9, when $\Theta_{mcr} \leq 0.25$, FBSR also remains 0, which means no botnets have been falsely split into different communities. However, as Θ_{mcr} increasing from 0.5

Table 2.9: DR and FPR for different θ_{avgddr} and θ_{avgmcr} of PeerHunter.

		θ_{avgddr}				
θ_{avgmcr}	-	0-0.0625	0.125	0.25	0.5	1
0-0.03125	DR	37/37	32/37	32/37	16/37	0/37
	FP	60/60	32/60	0/60	0/60	0/60
0.0625-0.25	DR	37/37	32/37	32/37	16/37	0/37
	FP	0/60	0/60	0/60	0/60	0/60
0.5	DR	21/37	16/37	16/37	16/37	0/37
	FP	0/60	0/60	0/60	0/60	0/60
1	DR	0/37	0/37	0/37	0/37	0/37
	FP	0/60	0/60	0/60	0/60	0/60

to 1, FBSR is also increasing, which means bots within the same botnets have been clustered into different communities. This reflects that most of the MCG edge weighs between bots are less than 0.5. If $\Theta_{mcr} \geq 0.5$, bots even within the same botnets will be isolated. FLCR and FBCR are always 0 no matter how Θ_{mcr} has been changed. FLCR is 0 means that all bots are successfully separated from legitimate hosts. FBCR is 0 means none of the communities contains more than one type of bots. This results demonstrate that our system is very effective and robust in separating bots and legitimate hosts, and separating different types of bots.

2.5.4 Evaluation on Botnet Detection

We evaluate the botnet detection component with different parameter settings. We applied this component on the remain network flows (24 experimental datasets) after previous two components (with $\theta_{dd} = 50$ and $\Theta_{mcr} = 0.03125$). Table 3.10 shows the results with different $\theta_{avgddr} \in [0, 1]$ and $\theta_{avgmcr} \in [0, 1]$. The results support our idea that the AVGDDR of legitimate host communities is lower than most of the P2P botnets. For instance, the AVGDDR of all (60/60) legitimate host communities are less than 0.25, but the AVGDDR of 32 out of 37 botnets are higher than 0.25. The missing ones turned out to be 5 Sality bots, which could be detected by AVGMCR. As shown in Table 3.10, legitimate P2P hosts have lower AVGMCR than P2P bots (e.g. $\theta_{avgmcr} = 0.0625$). This experimental results demonstrate that our botnet detection component is effective (detection rate equals to 100 % with zero false positives) and stable over a large range of θ_{avgddr} (e.g. $[0, 0.0625]$) and θ_{avgmcr} (e.g. $[0.0625, 0.25]$).

Table 2.10: Number of hosts identified by each component of PeerHunter.

-	**Before P2P detection**	**After P2P detection**
# of hosts	10,000	97
-	**After Community detection**	**After Bot detection**
# of hosts	97	37

Table 2.11: PeerHunter execution time.

-	**Processing Time**
P2P Host Detection	15 minutes
MCG Extraction	5 minutes
Community Detection	18 milliseconds
Bot Detection	11 milliseconds
Total	20 minutes

2.5.5 Evaluation on PeerHunter

We evaluate our system according to effectiveness and scalability. Effectiveness is to evaluate the capability of our systems to detect P2P botnets, and scalability is to evaluate the practicality of our systems to deal with the real world big data. We applied PeerHunter on 24 experimental synthetic datasets, with θ_{dd}=50, Θ_{mcr}=0.03125, θ_{avgddr}=0.0625 and θ_{avgmcr}=0.25, and all results are averaged over 24 datasets.

We use detection rate and false positive rate to measure the effectiveness. As shown in Table 3.11, our system identified all 97 P2P hosts from 10,000 hosts, and detected all 37 bots from those 97 P2P hosts, with zero false positives. It is clear that PeerHunter is effective and accurate in detecting P2P botnets.

Our system has a scalable design based on efficient detection algorithm and distributed / parallelized computation. Out of three components in our system, the P2P botnet detection component (community detection and botnet detection as shown in Table 3.11) has a negligible processing time compared with the other two components. This is due to previous two components are designed to reduce a huge amount of the hosts subject to analysis (e.g. 99.03% in our experiments). The P2P host detection component has linear time complexity, since it scans all the input flows only once to compute the flow clusters and further identify P2P flows. However, since it is the very first component to process the input data, which could be large, it still costs the highest processing time (as shown in Table 3.11). To accommodate the growth of a real world input data (big data), we designed and implemented the P2P host detection component using a Map-Reduce framework, which could be deployed in distributed fashion on scalable cloud-computing platforms (e.g. amazon EC2). The MCG extraction component requires pairwise comparison to calculate edges weights. Let n be the number of hosts subject to analysis and m be the maximum number of distinct contacts of a host. We implemented the comparison between each pair of hosts parallelly to handle the growth of n. If we denote k as the number of threads running parallelly, the time complexity of MCG extraction is $O(\frac{n^2m}{k})$. For a given ISP network, m grows over time. Since our system uses a fixed time window (24 hours), for a given ISP network, m tends to be stable and would not cause a scalability issue. Besides, since the percentage of P2P hosts of an ISP network is relatively small (e.g. 3% [1]), and an ISP network usually has less than 65,536 (/16 subnet) hosts, n would be negligible compared with m. Furthermore, even if n and m are both big numbers, our system could use an as large as possible k to adapt the scale of n and m. In a nutshell, PeerHunter is scalable to handle the real world big data.

2.6 Conclusion

In this work, we present a novel community behavior analysis based P2P botnet detection system, PeerHunter, which operates under several challenges: (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) no bot-blacklist or “seeds” are available; (d) none statistical traffic patterns known in advance; and (e) do not require to monitor individual host. We propose three types of community behaviors that can be utilized to detect P2P botnets effectively. In the experimental evaluation, we propose a network traces sampling and mixing method to make the experiments as unbiased and challenging as possible. Experiments and analysis have been conducted to show the effectiveness and scalability of our system. With the best parameter settings, our system can achieved 100% detection rate with none false positives.

Chapter 3: P2P Botnet Detection through Flow-level Community Behavior Analysis

Peer-to-peer (P2P) botnets have become one of the major threats in network security for serving as the fundamental infrastructure for various cyber-crimes. More challenges are involved in the problem of detecting P2P botnets, despite a few work claimed to detect centralized botnets effectively. We present Enhanced PeerHunter, a network-flow level community behavior analysis based system, to detect P2P botnets. Our system starts from a P2P network flow detection component. Then, it uses "mutual contacts" to cluster bots into communities. Finally, it uses network-flow level community behavior analysis to detect potential botnets. In the experimental evaluation, we propose two evasion attacks, where we assume the adversaries know our techniques in advance and attempt to evade our system by making the P2P bots mimic the behavior of legitimate P2P applications. Our results showed that Enhanced PeerHunter can obtain high detection rate with few false positives, and high robustness against the proposed attacks. [2]

3.1 Introduction

A botnet is a set of compromised machines controlled by botmasters through command and control (C&C) channels. Botnets may have different communication architectures. Traditional botnets are known to use centralized architectures, which have potential single point of failure. Peer-to-peer (P2P) network is modeled as a distributed architecture, where even if a certain number of peers do not function properly, the whole network is not compromised. Most of the recent botnets (e.g., Storm, Waledac and ZeroAccess) attempted to use P2P architectures, and P2P botnets were proved to be highly resilient even after a certain number of bots being identified or taken down [2]. P2P botnets provide a fundamental infrastructure for various cyber-crimes, such as distributed denial-of-service (DDoS), email spam, click fraud, etc. For instance, recent botnet attacks including those carried out by WhiskeyAlfa (responsible for Sony Pictures Entertainment

[2] This chapter was published in IEEE Transactions on Information Forensics and Security (Volume: 14, Issue: 6, June 2019) [51]. Copyright permission is included in Appendix A.

attack) and WannaCry (responsible for ransoming healthcare facilities in Europe) showed the scale and scope of damage that P2P botnets can cause. As such, detecting P2P botnets effectively is rather important for securing cyberspace.

Designing an effective P2P botnets detection systems is very challenging. First, botnets tend to act stealthily [1] and spend most of their time in the waiting stage before performing any malicious activities [3]. Approaches using malicious activities would have small window of opportunities to detect such botnets. Second, botnets tend to encrypt the C&C channels, causing deep-packet-inspection (DPI) based methods ineffective. Third, the role of a single bot can be changed dynamically depending on the current structure of a botnet [4] (e.g., P2P bot can shift its functionality to act as a botmaster when the prior botmaster has been taken down). Hence, it is difficult to characterize a botnet just by looking at a single bot.

In this work, we present Enhanced PeerHunter, an extension of PeerHunter [5], aiming to use network-flow level community behaviors to detect waiting stage P2P botnets, even in the scenario that P2P bots and legitimate P2P applications are running on the same set of hosts. We consider a botnet community as a group of compromised machines that communicate with each other or connect to the same set of botmasters through the same C&C channel, are controlled by the same attacker, and aim to perform similar malicious activities. In the "waiting stage", no malicious activities could be observed. As discussed in [4], the dynamic change of communication behaviors of P2P botnets makes it extremely hard to identify a single bot. Nonetheless, bots belonging to the same P2P botnet always operate together as a community and share the same set of community behaviors. Our system starts from a P2P network flow detection component, and builds a network-flow level mutual contacts graph (MCG) depending on the mutual contacts characteristics [35] between each pair of the P2P network flows. Afterwards, it employs a community detection component to cluster the same type of bots into the same community, and separate bots and legitimate applications or different types of bots into different communities. Finally, our system uses the *destination diversity* (the "P2P behavior") and the *mutual contacts* (the "botnet behavior") as the natural behaviors to detect P2P botnet communities.

In the experiments, we mixed a background network dataset [36] with 5 P2P botnets datasets and 4 legitimate P2P applications datasets [37]. To make our experimental evaluation as unbiased and challenging as possible, we propose a network traces sampling and mixing method to generate synthetic experimental datasets. To be specific, we evaluated our system with 100 synthetic experimental datasets that each contains 10,000 internal hosts. We implemented our P2P network flow detection component using MapReduce framework, which dramatically reduced the number of hosts subject to analysis by 99.03% and retained most of the P2P hosts. Also, the MapReduce design and implementation of our system could be deployed on cloud-computing platforms (e.g., Amazon EC2), which ensures the scalability of our system (i.e., processing an average of 97 million network flows in about 20 minutes). To summarize, our work has the following contributions:

- We present a novel, effective and efficient network-flow level community behavior analysis based system, Enhanced PeerHunter, which is capable of detecting P2P botnets when (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) the botnet traffic are overlapped with legitimate P2P traffic on the same host; and (d) none statistical traffic patterns are known in advance (unsupervised).

- We experimented our system using a wide range of parameter settings. With the best parameter settings, our system achieved 100% detection rate with zero false positive.

- We propose two evasion attacks (i.e., passive and active mimicking legitimate P2P application attacks), where we assume the adversaries know our techniques in advance and attempt to evade our system via instructing P2P bots to mimic the behavior of legitimate P2P applications. The experiment results showed that our system is robust to both attacks.

- We compared Enhanced PeerHunter with PeerHunter [5] (i.e., our previous work) and Zhang *et al.* [1]. Extensive experiments were conducted to show that (a) our system outperforms Zhang *et al.* [1] in terms of the detection rate of different botnets, the overall precision, recall and false positives, and (b) our system is more robust to MMKL attacks compared with PeerHunter [5] and Zhang *et al.* [1].

The rest of this chapter is organized as follows: Section 3.2 presents the related work. Section 3.3 explains the motivation and details of the features applied in our system. Section 3.4 describes the system design and implementation details. Section 3.5 presents the experimental evaluation. Section 3.6 discusses the evasions and possible solutions, deployment and the potential extensions of our system. Section 3.7 concludes.

3.2 Related Work

To date, a few methods attempting to detect P2P botnets were proposed [38, 35, 52, 3, 4, 37, 1, 5, 53, 54, 55, 56]. From the data perspective, recent approaches can be divided into two categories [56]: payload-based and flow-based. Payload-based systems [38, 57, 58] use payload content and header information of network packets to detect botnets. For instance, BotHunter [38] is a well-known packet inspecting bot detection system that relies on a modified Snort [59] (i.e., a rule-based intrusion detection system that requires the access to the full payload) to detect potential malicious activities and further identify infected hosts. Lu *et al.* [57] proposed to use decision tree models trained on the n-gram features extracted from the network traffic payload to detect botnets. Wang *et al.* [58] proposed to use lexical features of HTTP header (TCP payload) to discover malicious behaviors of Android botnets.

Flow-based systems [35, 52, 60, 3, 4, 37, 1, 5, 53, 54, 55, 56, 61] use header information of network packets (i.e., network flow characteristics) to capture botnets behaviors. Compared with payload-based systems, flow-based systems use less information from the network packets. Since recent botnets tend to use encryption to hide their payload information from the detection systems, most of the packet-based systems that applying deep packet inspection (DPI) on the payload information (e.g., BotHunter [38]) will be foiled. Zhang *et al.* [34] proposed to add a high-entropy flow detector into BotHunter to detect bots, when part of the packets payloads of botnets' network flows are encrypted. Their assumption is that the presence of high-entropy flows (detected from the encrypted packets payloads) together with existing botnets events (detected from the non-encrypted packets payloads by BotHunter) could identify botnets using encrypted network traffic. However, if all the packets payloads are encrypted [56], it will be hard for their approach to perform. The flow-based detection systems have advantage over the packet-based systems that applying deep packet inspection (DPI) on the payload information (e.g., BotHunter [38]) given that they can be applied to encrypted traffic. Some flow-based systems applied one or several different supervised machine learning algorithms on a set of well extracted network flow features to model the botnets behaviors. For instance, Jianguo *et al.* [62] applied three supervised machine learning algorithms (i.e., SVM, Logistic Regression and Neural Network) on network flow features extracted from Netmate and Tranalyzer to detect botnets. They obtained very high performance metrics, while employing a fully labelled dataset. Khanchi *et al.* [61] proposed an approach using genetic programming and ML on data streams to detect botnets flows. However, since most of the supervised ML-based approaches usually generate models that are focusing on specific types of botnets (existing in the training data), those approaches will not be effective to detect botnets not appeared in the training data (unknown botnets).

Some flow-based systems utilized a combination of different heuristics to model P2P botnets behaviors. For instance, Botgrep [52] proposed to detect P2P botnets through localizing structured communication graphs, where they found that the communication graph of P2P applications have fast convergence time of random walks to a stationary distribution. However, their method can only identify structured communication subgraphs, rather than ensure those subgraphs containing P2P botnets. Entelecheia [3] proposed to use a synergistic graph-mining approach on a super-flow graph built from network flow features (i.e., volume per hour, duration per flow) to identify a group of P2P bots, where they claimed that P2P botnet network flow tend to have low volume and long duration. Group or community behavior based methods [4, 35, 5, 53] considered the behavior patterns of a group of bots within the same P2P botnet community. Coskun *et al.* [35] developed a P2P botnets detection approach that started from building a mutual contacts graph of the whole network, then attempted to use "seeds" (known bots) to identify the rest of botnets. However, it is impractical to have a "seed" in advance. Similar to the idea of using mutual contacts graph, Ma *et al.* [63] proposed to use the coexistence of domain cache-footprints distributed in networks that participate in the outsourcing service (i.e., coexistence graph) to detect malicious domains. Yan *et al.* [4] proposed a group-level behavior analysis based P2P botnets detection method, where they started from clustering P2P hosts into groups, and then used supervised machine learning methods (e.g., SVM) to identify bots through a set of group-level behavior features. Since their approach relied on supervised classification methods (e.g., SVM) which required to train the model of each botnet on fully labelled dataset in advance, it would be hard for their method to detect unknown botnets. Chen *et al.* [64] applied three unsupervised machine learning algorithms (i.e., self-organising map, local outlier factor and k-NN outlier) to build a normal behavior profile to detect botnet. They obtained a very high detection rate (91.3%), but with inherited high false positive rates due to the nature of the unsupervised ML algorithms employed. PeerHunter [5], our previous work, proposed to use the host level community behavior analysis to detect P2P botnets, which did not consider the scenario that P2P bots and legitimate P2P applications could run on the same set of hosts. Zhang *et al.* [1] proposed a scalable botnet detection system capable of detecting stealthy P2P botnets (i.e., in the waiting stage), where no knowledge of existing malicious behavior was required in advance. They also claimed to work in the scenario that the botnet traffic are overlapped with the legitimate P2P traffic on the same host. However, their experimental dataset was slightly biased and less challenging. For example, in their dataset, the number of bots was twice as many as the number of legitimate P2P hosts, which was much easier for bots to form clusters than legitimate P2P hosts.

In this work, we present Enhanced PeerHunter, a network-level flow-based system that relies on community behavior analysis to detect P2P botnets. We compared Enhanced PeerHunter with PeerHunter [5] and Zhang *et al.* [1] on a more challenging and comprehensive experimental datasets, and showed that our system outperforms both systems in terms of detection rate, false positives and the performance under the proposed mimicking legitimate P2P application attacks.

3.3 Background and Motivation

In this section, we investigate the characteristics being used to detect P2P network traffic, and introduce the concept of "mutual contacts", which motivated us to formulate the P2P botnet detection problem as a network community detection problem. Also, we explore the P2P botnet community behaviors being used to identify botnets communities. To demonstrate the features discussed in this section, we conducted some preliminary experiments using the dataset shown in Table 3.3 and Table 3.4. Table 3.1 shows the notations and descriptions, and Table 3.2 shows the measurements of features.

3.3.1 P2P Network Characteristics

Due to the nature of P2P networks, P2P hosts usually communicate with their peers through IP addresses directly, without any queries from DNS services [65], namely, non-DNS connections (NoDNS). Also, peer churn is another typical behavior in P2P networks [66], which results in a significant number of failed connections in P2P network flow. Furthermore, due to the decentralized nature of P2P network, a P2P host usually communicates with peers distributed in a large range

Table 3.1: Enhanced PeerHunter notations and descriptions.

Notations	Descriptions
MNF	the management network flow
AVGDD	the average # of distinct /16 MNF dstIP prefixes
AVGDDR	the average destination diversity ratio
AVGMC	the average # of mutual contacts between a pair of hosts
AVGMCR	the average mutual contacts ratio
Θ_{dd}	the threshold of destination diversity
Θ_{mcr}	the threshold of mutual contacts ratio
Θ_{avgddr}	the threshold of AVGDDR
Θ_{avgmcr}	the threshold of AVGMCR
BSI	Bot Separation Index
BAI	Bot Aggregation Index
BLSI	Bot-Legitimate Separation Index

Table 3.2: Enhanced PeerHunter measurements of features.

Trace	AVGDD	AVGDDR	AVGMC	AVGMCR
eMule	8,349	17.6%	3,380	3.7%
FrostWire	11,420	15.2%	7,134	4.5%
uTorrent	17,160	8.7%	13,888	3.5%
Vuze	12,983	10.1%	18,850	7.9%
Storm	7,760	25.1%	14,684	30.2%
Waledac	6,038	46.0%	7,099	37.0%
Sality	9,803	9.5%	72,495	53.2%
Kelihos	305	97.4%	310	98.2%
ZeroAccess	246	96.9%	254	100.0%

of physical networks, which results in destination diversity (DD) [37] of P2P management network flow (MNF). To be clearer, P2P host generate two types of network flow: (1) management network flow, which maintains the function and structure of the P2P network, and (2) other network flow, such as data-transfer flow, which does not necessarily have the P2P network characteristics. The P2P network flow mentioned in this section and the rest all refers to P2P MNF.

Zhang *et al.* [1] proposed to remove a decent number of non-P2P network flow using NoDNS, and then performed a fine-grained P2P hosts detection using DD. Based on their experiment results, DD plays a much more important role in detecting P2P hosts than NoDNS. Therefore, in this work, we decided to only use DD to simplify and speed up the P2P network flow detection procedure. In addition, we used the number of distinct /16 IP prefixes of each host's network flow, rather than BGP prefix used in [1] to approximate DD, since /16 IP prefix is a good approximation of network

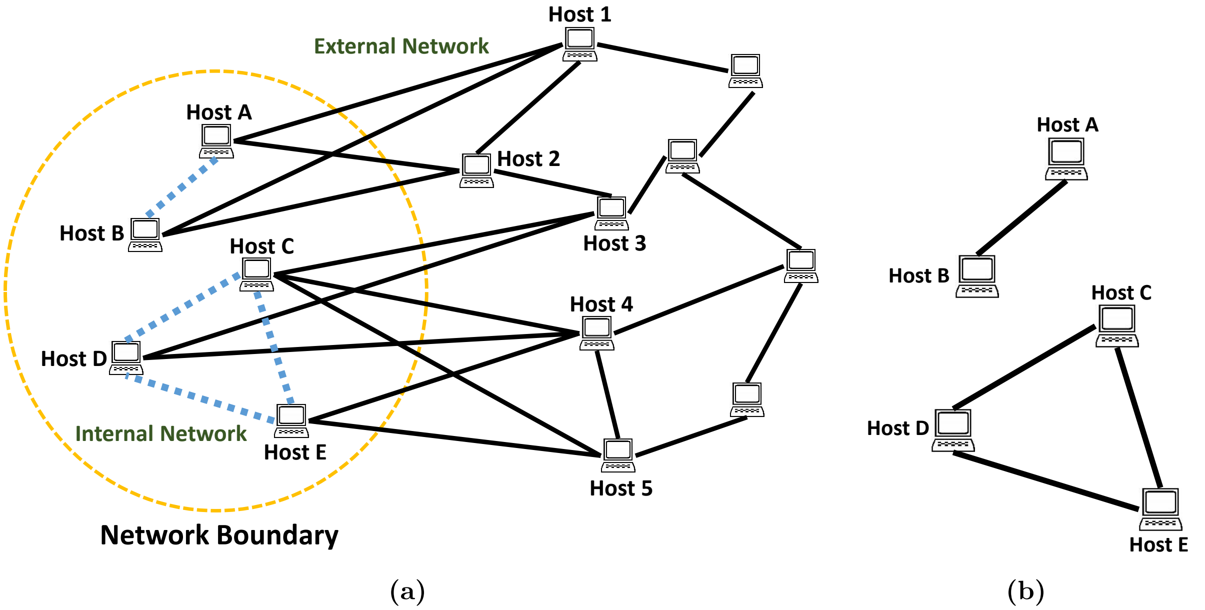


Figure 3.1: Illustration of network (a) and its mutual contacts graph (b).

boundaries. For instance, it is very likely that two IP addresses with different /16 IP prefixes belong to two distinct physical networks. This is also supported by Table 3.2, which shows the network flow in a P2P network spreading across many distinct physical networks according to the number of /16 IP prefixes.

3.3.2 Mutual Contacts

The mutual contacts (MC) between a pair of hosts is a set of shared contacts between them [35]. Consider the network illustrated in Figure 3.1a which contains an internal network (A, B, C, D and E) and an external network (1, 2, 3, 4 and 5). A link between a pair of hosts means communication between them. In Figure 3.1a, 1, 2 are the mutual contacts shared by A, B.

Mutual contacts are the natural characteristic of P2P botnet. Compared with legitimate hosts, a pair of bots within the same P2P botnet has higher probability to share mutual contacts [35]. Because bots within the same P2P botnet tend to receive the same C&C messages from the same set of botmasters [44]. Moreover, in order to prevent bots (peers) from churning, the botmaster must check each bot periodically, which results in a convergence of contacts among peers within the same botnet [1]. However, since bots from different botnets are controlled by different botmasters, they will not share many mutual contacts. A pair of Legitimate hosts may have a small set of mutual contacts, since nearly all hosts communicate with some popular servers, such as google.com, facebook.com [35]. Furthermore, the host pairs running the same P2P applications may also result in a decent ratio of mutual contacts, if they communicate with the same set of peers by coincidence. However, in practice, legitimate P2P hosts with different purposes will not search for the same set of peers. As such, we can use mutual contacts to cluster the bots within the same botnet, and separate P2P botnets from legitimate P2P applications.

The basic idea of using mutual contacts is to build a mutual contacts graph (MCG) as shown in Figure 3.1, a host level MCG, where A, B are linked together in Figure 3.1b, since they have mutual contacts 1, 2 in Figure 3.1a. Similarly, C, D, E are linked to each other in Figure 3.1b, since every pair of them share at least one mutual contacts in Figure 3.1a. More details about network-flow level MCG is discussed in Section 3.4.2.

3.3.3 Community Behavior Analysis

Due to the dynamic changes of a single bot's communication behavior [4], it would be extremely hard to identify a single bot. However, bots within the same P2P botnet always work together as a community, thus, should have distinguishable community behaviors. We consider three types of community behaviors: (a) flow statistical feature, (b) numerical community feature and (c) structural community feature.

3.3.3.1 Flow Statistical Feature

Botnet detection methods using flow statistical features, have been widely discussed [4, 3, 1, 5]. For the MNFs of a specific P2P application, most of its statistical patterns depend on its P2P network protocol. However, the statistical patterns of other network flows, such as data-transfer flow, are usually situation-dependent, which vary a lot even in the same P2P network. In this work, we use the ingoing and outgoing bytes-per-packets (BPP) of MNFs in one P2P network as its community flow statistical feature.

3.3.3.2 Numerical Community Feature

We consider two numerical community features: average destination diversity ratio (AVGDDR) and average mutual contacts ratio (AVGMCR).

The average destination diversity ratio (AVGDDR) captures the “P2P behavior” of P2P botnets. The destination diversity (DD) of a P2P host is the number of distinct /16 IP prefixes of its network flows’ destination IPs. The destination diversity ratio (DDR) of each host is its DD divided by the total number of distinct destination IPs of its network flows. Due to the decentralized nature of P2P networks, P2P network flow tend to have higher DDR than non-P2P network flow. Furthermore, network flow from P2P botnets usually have higher AVGDDR than network flow from legitimate networks. Network flow from bots within the same botnet tend to have similar DDR, since those bots are usually controlled by machines, rather than humans. However, the destinations of legitimate P2P network flow are usually user-dependent, which result in their DDR varying greatly from user to user. Besides, our approach aims to cluster bots within the same botnets together, rather than attempting to cluster the legitimate hosts. Therefore, legitimate communities might contain both P2P hosts and non-P2P hosts, leading to lower AVGDDR. As shown in Table 3.2, both legitimate hosts and bots spread across a wide range of distinct networks. However, most of the botnets have higher AVGDDR than legitimate applications, except Sality.

The average mutual contacts ratio (AVGMCR) captures the “botnet behavior” of P2P botnets. The mutual contacts ratio (MCR) between a pair of hosts is the number of mutual contacts between them, divided by the number of total distinct contacts of them. This is based on three observations: (a) P2P botnets are usually formed by at least two bots, otherwise they cannot act as a group, (b) the MCR of a pair of bots within the same botnet is much higher than the MCR of a pair of legitimate applications or a pair of bots from different botnets, and (c) each pair of bots within the same botnet has similar MCR. Thus, we define AVGMCR as the average MCR among all pairs of hosts within one network community. As shown in Table 3.2 both botnets and certain legitimate network communities have a considerable number of mutual contacts. That is because those legitimate communities have much more “base” contacts than botnets. However, botnets have much higher AVGMCR.

3.3.3.3 Structural Community Feature

This captures the structural characteristics of a botnet. As discussed above, every pair of bots within the same botnet tends to have a considerable number or ratio of mutual contacts. If we consider each host as a vertex and link an edge between a pair of hosts when they have mutual

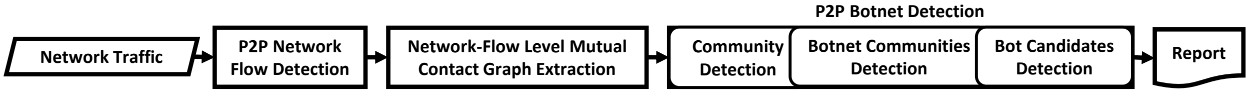

Network Traffic
P2P Network Flow Detection
Network-Flow Level Mutual Contact Graph Extraction
P2P Botnet Detection
Community Detection
Botnet Communities Detection
Bot Candidates Detection
Report

Figure 3.2: Enhanced PeerHunter system overview.

contacts, the bots within the same botnet tend to form cliques. On the contrary, the contacts of different legitimate hosts usually diverge into different physical networks. Thus, the probability that legitimate communities form certain cliques is relatively low. Then, we can consider P2P botnets detection as a clique detection problem, which detects cliques from a given network with certain requirements. However, since clique detection problem is NP-complete, we cannot directly apply such method to detect botnets, without any preprocessing. We propose to combine all three botnet community behaviors, and use the previous two community behaviors as the "preprocessing" of the clique detection problem.

3.4 System Design

Enhanced PeerHunter has three components, as shown in Figure 6.2, that work synergistically to (a) detect P2P network flow, (b) construct the network-flow level mutual contacts graph, and (c) detect P2P botnets.

3.4.1 P2P Network Flow Detection

This component aims to detect network flow that engage in P2P communications using the features described in Section 3.3.1. The input is a set of 5-tuple network flow [ip_{src}, ip_{dst}, $proto$, bpp_{out}, bpp_{in}], where ip_{src} is the source IP, ip_{dst} is the destination IP, $proto$ is either tcp or udp, and bpp_{out} and bpp_{in} are outgoing and ingoing bytes-per-packets (BPP) statistics. First, we group all network flows $F = \{f_1, f_2, \ldots, f_k\}$ into flow clusters $FC = \{FC_1, FC_2, \ldots, FC_m\}$ using the 4-tuple [ip_{src}, $proto$, bpp_{out}, bpp_{in}]. Then, we calculate the number of distinct /16 prefixes of ip_{dst} (destination diversity) associated with each flow cluster, $dd_i = DD(FC_i)$. If dd_i is greater than a pre-defined threshold Θ_{dd}, we consider FC_i as a P2P MNF cluster, and its source hosts as P2P hosts. We retain all the network flows within the P2P MNF clusters for the next component, and eliminate all the other network flows.

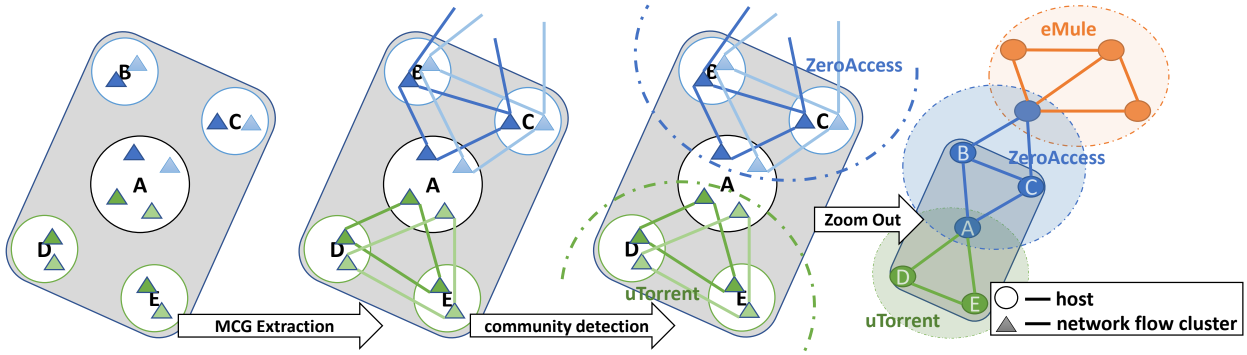


Figure 3.3: Mutual contacts graph extraction and community detection. Each triangle represents a network flow cluster, and the same color triangles represent the same type of network flow clusters. The areas separated by the dash-dot line with different color represents different communities.

Algorithm 4: P2P Network Flow Detection

```
Function Map([$ip_{src}, ip_{dst}, proto, bpp_{out}, bpp_{in}$]):
    $Key \leftarrow [ip_{src}, proto, bpp_{out}, bpp_{in}]$;
    $Value \leftarrow ip_{dst}$;
    return $(Key, Value)$;
Function Reduce($Key$, $Value$[ ]):
    $k \leftarrow Key$;
    $dd_k = \emptyset$;
    for $v \in Value[\,]$ do
        $dd_k \leftarrow dd_k \cup \{v\}$;
    if $|dd_k| \geq \Theta_{dd}$ then
        for $v \in Value[\,]$ do
            return $(k, v)$;
```

As shown in Algorithm 4, we designed this component using a MapReduce framework [47]. For a mapper, the input is a set of 5-tuple network flow, and the output is a set of key-value pairs, where the key is the 4-tuple [ip_{src}, $proto$, bpp_{out}, bpp_{in}], and the value is its corresponding ip_{dst}. For a reducer, the input is the set of key-values pairs that outputs by the mapper. Then, the reducer aggregates all values with the same key to calculate the DD of each flow cluster, and finally output the detected P2P MNF based on Θ_{dd}.

3.4.2 Network-Flow Level Mutual Contacts Graph Extraction

This component aims to extract mutual contacts graph (MCG) using the network-flow level mutual contacts. We call a pair of P2P network flow clusters are the same type, if they have the same 3-tuple [$proto$, bpp_{out}, bpp_{in}]. As illustrated in Figure 3.3, each host might contain one type or several different types of P2P network flow clusters generated by either P2P botnets or legitimate P2P applications running on it. If a pair of the same type of P2P network flow clusters generated by different hosts, have at least one (network-flow level) mutual contacts, we create an edge between them in the corresponding network-flow level MCG.

Algorithm 5: Network-Flow Level MCG Extraction

```
Input: $FC$, $F$, $\Theta_{mcr}$.
Output: $G_{mc} = (V, E)$.
$E = \emptyset$, $V = \emptyset$;
for $FC_i \in FC$ do
    $C_i = \emptyset$;
    $S_i = \emptyset$;
for $f_i^j \in F$ do
    $C_j \leftarrow C_j \cup \{ip_{dst}\}$;
    $S_j \leftarrow S_j \cup \{[proto, bpp_{out}, bpp_{in}]\}$;
for $FC_i \in FC$ do
    $ddr_i \leftarrow \frac{\|DD(C_i)\|}{\|C_i\|}$;
    $vertex\ v_i \leftarrow< ddr_i >$;
    $V \leftarrow V \cup \{v_i\}$;
for $\forall\ FC_i, FC_j \in FC\ and\ i < j$ do
    if $S_i \cap S_j \neq \emptyset$ then
        $mcr_{ij} \leftarrow \frac{\|C_i \cap C_j\|}{\|C_i \cup C_j\|}$;
        if $mcr_{ij} > \Theta_{mcr}$ then
            $edge\ e_{ij} \leftarrow< mcr_{ij} >$;
            $E \leftarrow E \cup \{e_{ij}\}$;
return $G_{mc} = (V, E)$;
```

To be specific, the input is a set of P2P network flow clusters FC={FC_1, FC_2, $\ldots$, FC_m}, and their corresponding P2P network flows, F={f_1^1, f_2^1, $\ldots$, $f_{n_1}^1$, f_1^2, f_2^2, $\ldots$, $f_{n_2}^2$, $\ldots$, $f_1^{|FC|}$, $f_2^{|FC|}$, $\ldots$, $f_{n_{|FC|}}^{|FC|}$}, where f_i^j denotes the flow i of FC_j. The output is a MCG, $G_{mc} = (V, E)$, where each vertex $v_i \in V$ represents network flow cluster FC_i and has a DDR score ddr_i, and each edge $e_{ij} \in E$

represents the existence of mutual contacts between FC_i and FC_j and has a nonnegative MCR weight mcr_{ij}. Algorithm 5 shows the detailed steps.

First, for each P2P network flow cluster FC_i, we generate a contact set C_i, that contains all the destination IPs of its network flows. Each P2P network flow cluster FC_i also contains a flow statistical pattern set S_i, which contains all the 3-tuple [$proto$, bpp_{out}, bpp_{in}] of its network flows. Let $DD(C_i)$ be the set of distinct /16 prefixes of all the IPs in C_i. Then, ddr_i and mcr_{ij} can be calculated as follows.

$$ddr_i = \frac{\|DD(C_i)\|}{\|C_i\|} \qquad mcr_{ij} = \frac{\|C_i \cap C_j\|}{\|C_i \cup C_j\|} \tag{3.1}$$

Furthermore, as discussed in Section 3.3.3.1, the network flows from different hosts (or network flow clusters) within the same network communities (generated by the same type of P2P botnet or legitimate P2P application) should have similar statistical patterns. Thus, for each pair of input P2P network flow clusters, say FC_i and FC_j, we calculate the intersection between S_i and S_j. If $S_i \cap S_j = \emptyset$, then there should be no edge between FC_i and FC_j in MCG. Otherwise, they share at least one network flow statistical pattern, and we calculate mcr_{ij} as shown in equation (3.1). Let Θ_{mcr} be a pre-defined threshold. Then, if $mcr_{ij} > \Theta_{mcr}$, there is an edge between FC_i and FC_j, with weight mcr_{ij}. Otherwise, there is no edge between FC_i and FC_j (i.e., $mcr_{ij} = 0$).

3.4.3 P2P Botnet Detection

This component aims to detect P2P bots from given MCG. First, we cluster the bots and the other hosts into their own communities using a community detection method. Afterwards, we detect botnet communities using numerical community behavior analysis. Finally, we use structural community behavior analysis to further identify or verify each bot candidate. Algorithm 6 shows the detailed steps.

3.4.3.1 Community Detection

Given MCG $G_{mc} = (V, E)$, $\forall\ e_{ij} \in E$, we have $mcr_{ij} \in [0.0, 1.0]$, where $mcr_{ij} = 1.0$ means all contacts of FC_i and FC_j are mutual contacts and $mcr_{ij} = 0.0$ means there is no mutual contact between FC_i and FC_j. Furthermore, the same type of P2P network flow clusters that generated by different bots within the same botnet tend to have a higher ratio of mutual contacts. As such, the P2P bots clustering problem can be considered as a network community detection problem. As shown in Figure 3.3, each host might be running P2P bots or legitimate P2P applications or both, and each P2P bot or each legitimate P2P application generates different types of network flow clusters. Our community detection aims to cluster the same type of P2P network flow clusters generated by different bots into the same network flow cluster community. As such, each network flow cluster should only belong to a single network flow cluster community, but each host might belong to different host communities. Also, each botnet might contain several different network flow cluster communities. Once one network flow cluster community has been detected as belonging to a botnet, we consider the corresponding hosts as bots.

We used Louvain method, a modularity-based community detection algorithm [12], due to (a) its definition of a good community detection result (high density of weighted edges within communities and low density of weighted edges between communities) is perfect-suited for our P2P botnet community detection problem; (b) it outperforms many other modularity methods in terms of computation time [12]; and (c) it can handle large network data sets (e.g., the analysis of a typical network of 2 million nodes takes 2 minutes [12]).

Given $G_{mc} = (V,\ E)$ as input, Louvain method outputs a set of network flow cluster communities $Com = \{com_1,\ com_2,\ \ldots,\ com_{|Com|}\}$, where $com_i = (V_{com_i},\ E_{com_i})$. V_{com_i} is a set of network flow clusters in com_i. E_{com_i} is a set of edges, where $\forall\ e_{jk} \in E_{com_i}$, we have $e_{jk} \in E$ and $v_j, v_k \in V_{com_i}$.

3.4.3.2 Botnet Communities Detection

Given a set of communities Com, for each community $com_i \in Com$, we calculate its $avgddr_i$ and $avgmcr_i$ as follows.

$$avgddr_i = \frac{\sum_{v_j \in V_{com_i}} ddr_j}{\|V_{com_i}\|} \tag{3.2}$$

$$avgmcr_i = \frac{2 \times \sum_{\forall e_{jk} \in E_{com_i}} mcr_{jk}}{\|V_{com_i}\| \times (\|V_{com_i}\| - 1)} \tag{3.3}$$

We define two thresholds Θ_{avgddr} and Θ_{avgmcr}. $\forall\ com_i \in Com$, if $avgddr_i \geq \Theta_{avgddr}$ and $avgmcr_i \geq \Theta_{avgmcr}$, we consider com_i as a botnet network flow cluster community.

Table 3.3: Enhanced PeerHunter traces of legitimate P2P networks (24 hours).

Trace	# of hosts	# of flow	# of dstIP	Size
eMule	16	4,181,845	725,367	42.1G
FrostWire	16	4,479,969	922,000	11.9G
uTorrent	14	10,774,924	2,326,626	57.1G
Vuze	14	7,577,039	1,208,372	20.3G

3.4.3.3 Bot Candidates Detection

Recall from Section 3.3.3.3, the MCG of botnet communities usually have a structure of one or several cliques. Therefore, we used a maximum clique detection method *CliqueDetection* to verify each bot network flow cluster from botnet network flow cluster communities, and further identify bot candidates. Each time it tries to detect one or several maximum cliques on the given botnet (network flow cluster) communities. If the maximum clique (at least containing 3 vertices) has been found, we consider the network flow clusters in that clique as bot network flow cluster, and run the maximum clique detection algorithm on the remaining parts, until no more qualified maximum cliques to be found. Afterwards, we report the corresponding source hosts of the identified bot network flow clusters as the bot candidates.

Algorithm 6: P2P Botnet Detection

```
Input: $G_{mc}$, $\Theta_{avgddr}$, $\Theta_{avgmcr}$.
Output: $S_{bot}$.
$S_{botFCCom} = \emptyset$, $S_{botFC} = \emptyset$, $S_{bot} = \emptyset$;
$Com \leftarrow$ Louvain($G_{mc}$);
for $com_i \in Com$ do
    $avgddr_i \leftarrow \frac{\sum_{v_j \in V_{com_i}} ddr_j}{\|V_{com_i}\|}$;
    $avgmcr_i \leftarrow \frac{2 \times \sum_{\forall e_{jk} \in E_{com_i}} mcr_{jk}}{\|V_{com_i}\| \times (\|V_{com_i}\| - 1)}$;
    if $avgddr_i \geq \Theta_{avgddr}$ and $avgmcr_i \geq \Theta_{avgmcr}$ then
        $S_{botFCCom} \leftarrow S_{botFCCom} \cup \{com_i\}$;
for $com_i \in S_{botFCCom}$ do
    $S_{botFC} \leftarrow$ CliqueDetection($com_i$);
    for $FC_i \in S_{botFC}$ do
        for $f_j^i \in FC_i$ do
            $S_{bot} \leftarrow S_{bot} \cup \{ip_{src}\}$;
return $S_{bot}$;
```

Table 3.4: Enhanced PeerHunter traces of P2P botnets (24 hours).

Trace	# of bots	# of flow	# of dstIP	Size
Storm	13	8,603,399	145,967	5.1G
Waledac	3	1,109,508	29,972	1.1G
Sality	5	5,599,440	177,594	1.5G
Kelihos	8	122,182	944	343.9M
ZeroAccess	8	709,299	277	75.2M

Table 3.5: Enhanced PeerHunter traces of background network.

Date	Dur	# of hosts	# of flow	Size
2014/12/10	24 hours	48,607,304	407,523,221	788.7G

3.5 Experimental Evaluation

3.5.1 Experiment Setup

3.5.1.1 Experiment Environment

All the experiments were conducted on a PC with an 8 core Intel i7-4770 Processor, 32GB RAM, running 64-bit Ubuntu 16.04 LTS operating system. Our system was implemented using Java with JDK 8.

3.5.1.2 Data Collection and Analysis Tool

We used three main datasets: (a) 24 hours network traces of 4 popular legitimate P2P applications, (b) 24 hours network traces of 5 P2P botnets, and (c) 24 hours network traces from a Trans-Pacific backbone line between the United States and Japan as the background network traces (non-P2P & manually verified P2P).

Our legitimate P2P network traces D_{p2p} were obtained from the University of Georgia [37], which collected the network traces of 4 popular P2P applications for several weeks. We obtained the network traces of 16 eMule hosts, 16 FrostWire hosts, 14 uTorrent hosts and 14 Vuze hosts by randomly selecting a set of continuous 24 hours network traces of each host (as shown in Table 3.3).

Part of our botnets network traces (D_{bot}) were from the University of Georgia [37], containing 24 hours network traces of 13 Storm hosts and 3 Waledac hosts. We also collected 24 hours network traces of another three P2P botnets, Sality, Kelihos and ZeroAccess. These network traces were all collected from the hosts manually infected by the binary samples of Kelihos, ZeroAccess, and Sality obtained from [49]. Our data collection was operated in a controlled environment, where all malicious activities were blocked. The same data collection settings were used in several previous works [4, 37, 1]. We collected the network traces of 8 Kelihos bots, 8 ZeroAccess bots and 5 Sality bots (as shown in Table 3.4).

We used a dataset from the MAWI Working Group Traffic Archive [36] as our background network traces (D^b_{non} and D^b_{p2p}), containing 24 hours anonymized and payload-free network traces at the transit link of WIDE (150Mbps) to the upstream ISP on 2014/12/10 (as shown in Table 3.5). The dataset contains approximate 407,523,221 flows and 48,607,304 unique IPs. 79.3% flows are TCP flows and the rest are UDP flows.

We investigated the background network traces, and made our best effort to separate the P2P traffic (D^b_{p2p}) from the non-P2P traffic (D^b_{p2p}). Since the WIDE dataset was anonymized and payload-free, it prevented us from using payload analysis to thoroughly check if P2P traffic, especially P2P Botnet traffic existing there. Instead, we used port analysis to manually detect P2P traffic within the background dataset. This is based on the simple concept that many P2P applications have default ports on which they function (see [67] for a list of default network ports of popular P2P applications). We manually examined all the network flows of each host in the background network traces. If a host involved in more than five flows using any of the default P2P port values in either source port or destination port, we considered the host as a P2P host. After this procedure, we identified 667 P2P hosts.

One thing worth to be noticed is that despite the whole background network traces lasting for 24 hours, not all these P2P hosts were active for the entire 24 hours. P2P hosts that did not have enough active time, may not produce sufficient network flows for our system to work (as discussed in Section 3.5.2). To ensure a fair and rigorous evaluation, we estimated the active time of each P2P host. We divided the 24 hours background network traces into 96 15-minute blocks. If a P2P host had any network flow fell in a block, we considered it was active in that block. We used the number of blocks where a P2P host was active to estimate the active time of each P2P host. Table 3.6 reflects the active time distribution of these P2P hosts. As shown in Table 3.6, even though there were 667 P2P hosts in total, only 4 of them had been active for the entire 24 hours and 26 of them had been active for no less than 5 hours.

Table 3.6: Active time of P2P hosts within the background network trace. (P_i is the set of P2P hosts have no less than $i \times 15$ minutes active time.)

-	# of hosts	-	# of hosts	-	# of hosts
P_1	667	P_8	66	P_{32}	21
P_2	325	P_{14}	38	P_{48}	13
P_4	180	P_{20}	26	P_{96}	4

Table 3.7: Summaries of Enhanced PeerHunter experimental datasets (EDs).

Descriptions	Values
the # of EDs	100
the # of bots (D_{bot}) in each ED	37
the # of legitimate P2P hosts (D_{p2p}) in each ED	60
the # of P2P hosts (D^b_{p2p}) in each ED	667
the # of internal hosts in each ED	10,000
the AVG # of external hosts in each ED	8,642,618
the AVG # of flow in each ED	97,640,210
the duration of each ED	24 hr

We used ARGUS [50] to process and cluster network traces into the 5-tuple format tcp/udp flow.

3.5.1.3 Experimental Dataset Generation

As illustrated in Figure 3.1a, we consider a scenario that an organization has a set of internal hosts communicating with a set of external hosts (outside of the organization), and our system is deployed at the boundary of the organization. Since our original datasets did not maintain a internal-external network structure while collecting them, we generated synthetic experimental datasets by mixing network traces from the original datasets. We considered a case that contains 10,000 internal hosts. For each synthetic experimental dataset, the 667 P2P hosts in D^b_{p2p} were considered as the internal hosts. Another 9,333 internal hosts were sampled from D^b_{non}, where the traffic of 37 randomly selected hosts were mixed with the traffic of 37 P2P bots in D_{bot}, and the traffic of another 60 randomly selected hosts were mixed with the traffic of 60 P2P hosts in D_{p2p}. To make the experimental evaluation as unbiased and challenging as possible, we propose to sample the internal hosts and generate the synthetic experimental datasets under the following two criterions.

First, we need to maintain a bipartite network structure. Our system aims to deploy at a network boundary (e.g., firewall, gateway, etc.), where the network forms a bipartite structure, and only network flow within the connections between internal hosts and external hosts could be captured. Then, the network in each experimental dataset should maintain a bipartite network structure, where any pair of internal hosts should not have any communications to each other.

Second, we need to keep the connectedness of mutual contacts graph. The easiest way to obtain a list of background hosts is to sample the hosts randomly from D^b_{non}, with the respect of bipartite structure. However, since D^b_{non} contains an extremely large number of hosts, simply sampling hosts randomly will result in that most of the sampled background hosts do not have a mutual contact with the other background hosts, which is much easier for our system to identify botnet communities. Because less number of mutual contacts among legitimate hosts means more disconnected legitimate communities in the corresponding MCG, which happens to be in favor of Louvain method to detect strongly connected botnet communities. Therefore, we need to sample a list of internal hosts in a way that every internal host should have at least one mutual contact with at least one another internal host.

To follow the criterions described above without making our evaluation tasks any easier, we propose the following synthetic experimental dataset generation procedure:

• Use a two-coloring approach to sample the network traces from D^b_{non} without jeopardize the bipartite network structure and the connectedness of mutual contacts graph: (a) initialize two counters, C_{black} and C_{white}, to count the number of hosts colored in black and white respectively; (b) coloring a random host h_i as black, and C_{black} plus one; (c) coloring all contacts of h_i as white, and increase C_{white} by the number of hosts colored as white in this round; (d) for each new colored host, color its contacts with the opposite color, and adjust the counters repeatedly, until we have $C_{black} \geq 9,333$ and $C_{white} \geq 9,333$; (e) select the colored host set with exactly 9,333 hosts as the internal hosts, the hosts in the other set will be the external hosts; and (f) extract the network traces of the 9,333 internal hosts from D^b_{non}. Then, it forms a bipartite graph, where each colored host set forms a bipartite component, and each host shares at least one mutual contacts with some other hosts from its own bipartite component.

• To maintain a bipartite network structure of botnets and legitimate P2P hosts, we eliminate all communications among bots in D_{bot}, and P2P hosts in D_{p2p} and D^b_{p2p}.

Table 3.8: Enhanced PeerHunter detection rate and false positive rate. (P_i is the set of P2P hosts within the background network traces that have no less than $i \times 15$ minutes active time. All the hosts of 4 legitimate P2P applications and 5 P2P botnets have 24 hours active time.)

$\boldsymbol{\Theta_{dd}}$	**Detection Rate**											**False Positive Rate**
	Bot	**P2P**	$\boldsymbol{P_1}$	$\boldsymbol{P_2}$	$\boldsymbol{P_4}$	$\boldsymbol{P_8}$	$\boldsymbol{P_{14}}$	$\boldsymbol{P_{20}}$	$\boldsymbol{P_{32}}$	$\boldsymbol{P_{48}}$	$\boldsymbol{P_{96}}$	
2	37/37	60/60	667/667	325/325	180/180	66/66	38/38	26/26	21/21	13/13	4/4	1,052/9,236
5	37/37	60/60	364/667	242/325	180/180	66/66	38/38	26/26	21/21	13/13	4/4	110/9,236
10	37/37	60/60	156/667	133/325	106/180	66/66	38/38	26/26	21/21	13/13	4/4	44/9,236
30	37/37	60/60	36/667	36/325	36/180	33/66	30/38	26/26	21/21	13/13	4/4	4/9,236
50-180	37/37	60/60	15/667	15/325	15/180	15/66	15/38	15/26	15/21	13/13	4/4	0/9,236
185	37/37	60/60	6/667	6/325	6/180	6/66	6/38	6/26	6/21	6/13	4/4	0/9,236
200	29/37	60/60	4/667	4/325	4/180	4/66	4/38	4/26	4/21	4/13	2/4	0/9,236
500-1,000	21/37	60/60	1/667	1/325	1/180	1/66	1/38	1/26	1/21	1/13	1/4	0/9,236
5,000	13/37	45/60	0/667	0/325	0/180	0/66	0/38	0/26	0/21	0/13	0/4	0/9,236
10,000	0/37	18/60	0/667	0/325	0/180	0/66	0/38	0/26	0/21	0/13	0/4	0/9,236
12,500	0/37	5/60	0/667	0/325	0/180	0/66	0/38	0/26	0/21	0/13	0/4	0/9,236
13,500	0/37	0/60	0/667	0/325	0/180	0/66	0/38	0/26	0/21	0/13	0/4	0/9,236

Table 3.9: Enhanced PeerHunter community detection results for different Θ_{mcr}.

Θ_{mcr}	**BSI**	**BAI**	**BLSI**
[0.00, 0.15)	1.00 ± 0.00	0.85 ± 0.00	1.00 ± 0.00
[0.15, 0.40)	1.00 ± 0.00	0.83 ± 0.02	1.00 ± 0.00
[0.40, 1.00)	1.00 ± 0.00	$\leq 0.62 \pm 0.05$	1.00 ± 0.00

- To mix D_{bot} and D_{p2p} with D^b_{non}, each time we randomly select 97 internal hosts out of 9,333 background hosts, map the 97 hosts IPs to 37 bots IPs (D_{bot}) and 60 legitimate P2P hosts IPs (D_{p2p}), and merge the corresponding network traces.

To evaluate our system, 100 synthetic experimental datasets were generated by running this procedure. Table 3.7 illustrates the summaries of the experimental datasets (EDs).

3.5.2 Evaluation on P2P Network Flow Detection

We evaluated the P2P network flow detection with different Θ_{dd}. We applied this component on all 100 EDs, and Table 3.8 shows the average detection rate and false positives with different Θ_{dd}, ranging from 2 to 13,500. If Θ_{dd} is set too small, non-P2P hosts are likely to be detected as P2P hosts, which results in many false positives. For instance, when $2 \leq \Theta_{dd} \leq 5$, at least 110 non-P2P hosts were falsely identified as P2P hosts. If Θ_{dd} is set too large, all P2P hosts will be removed, which results in false negatives. For instance, when $\Theta_{dd} = 10,000$, most of the P2P hosts were falsely discarded, and only 18 P2P hosts were detected.

On the other hand, the effectiveness of Θ_{dd} is also subject to the active time of P2P hosts. Since if a P2P host has less active time, it tends to generate less number of P2P network flows to show enough destination diversity, so that it will not be distinguished from non-P2P network flows by our system. For instance, since all the bots and P2P hosts in D_{bot} and D_{p2p} had 24 hours active time, our system can distinguish them well from the non-P2P network flows. However, not all the P2P hosts in D^b_{p2p} were active for the entire 24 hours. As shown in Table 3.8, when the active time of a P2P host was less than 5 hours (not belonging to P_{20}, the set of hosts have no less than 20×15 minutes active time), it was hard for our system to detect P2P network flows from non-P2P network flows ($\Theta_{dd} < 30$). Hence, when considering P2P hosts that had no less than 12 hours active time (P_{48}), and setting $30 \leq \Theta_{dd} \leq 180$, our system detected all P2P hosts with a small number of false positives ($\leq 4/9,236$), which demonstrated that our P2P network flow detection component is stable and effective over a large range of Θ_{dd} settings.

3.5.3 Evaluation on Community Detection

We evaluated the performance of community detection with different Θ_{mcr}. We applied this component on the remaining network flows (100 EDs) of the previous component (with $\Theta_{dd} = 30$).

For each ED, our system generated a MCG $G_{mc} = (V, E)$ with a pre-defined threshold Θ_{mcr}, where each edge $e_{ij} \in E$ contained a weight $mcr_{ij} \in [0.0, 1.0]$. Afterwards, we applied Louvain method (with default resolution 1.0) on the MCG for community detection. The choice of Θ_{mcr} would have an influence on the community detection results.

We evaluated the community detection performance in terms of (a) the ability to cluster a pair of bots belonging to the same botnet, (b) the ability to separate a pair of bots coming from different botnets, and (c) the ability to separate bots and legitimate applications. As such, we propose three criterions to evaluate the community detection performance below.

Given a set of bots belonging to n botnets $X = \{X_1, X_2, \ldots, X_n\}$ (the ground truth), and the community detection results, m communities $Y = \{Y_1, Y_2, \ldots, Y_m\}$, define *Bot Separation Index* (BSI) and *Bot Aggregation Index* (BAI) as BSI $= a/(a + c)$ and BAI $= a/(a + b)$, where a is the number of pairs of bots that are in the same botnet in X, and in the same community in Y; b is the number of pairs of bots that are in the same botnet in X, and in different communities in Y; c is the number of pairs of bots that are in different botnets in X, and in the same community in Y. BSI denotes the degree of that bots coming from different botnets being separated into different communities. BAI denotes the degree of that bots coming from the same botnet being clustered into the same community. Both BSI and BAI are between 0.0 and 1.0, and the higher the better. "BSI equals to 1.0" means all different types of bots are well separated, and "BAI equals to 1.0" means all the same types of bots are well clustered.

Given p bots and q legitimate applications, define *Bot-Legitimate Separation Index* (BLSI) as BLSI $= d/(p \times q)$, where d is the number of pairs of a bot and a legitimate application being separated into different communities via our method. BLSI indicates the ability of our method to separate bots and legitimate applications. BLSI is between 0.0 and 1.0, and the higher the better. "BLSI equals to 1.0" means all pairs of one bot and one legitimate application are well separated.

Table 3.9 shows the community detection results with different Θ_{mcr}, ranging from 0.0 to 1.0. If Θ_{mcr} is set too small, there will be more non-zero weight edges, which might result in less but larger communities. On the other hand, if Θ_{mcr} is set too large, most of the vertices will be isolated, which results in more but smaller communities. For instance, as Θ_{mcr} increasing, BSI decreased. When $\Theta_{mcr} \leq 0.4$, BSI was around 0.8 to 0.85, meaning one or more botnets have been split into different communities. It turned out to be our algorithm separates the Storm botnet

(13 bots) into two communities, one containing 10 bots and another containing 3 bots. Changing Θ_{mcr} does not affect BSI and BLSI. BSI=1.0 means our system separates different types of bots into different communities. BLSI=1.0 means our system separates bots and legitimate P2P applications into different communities. The result demonstrated that our system is very effective and robust in separating bots and legitimate hosts, and separating different types of bots. Since larger Θ_{mcr} will result in less edges in the MCG, which could reduce the execution time of community detection, we used $\Theta_{mcr} = 0.1$ as our system parameter.

3.5.4 Evaluation on Botnet Detection

We evaluated the botnet detection component with different parameter settings. We applied this component on the remaining network flows (100 EDs) of the previous component (with $\Theta_{dd} = 30$ and $\Theta_{mcr} = 0.1$). We assumed that all the host in the background trace (D^b and D^b_{p2p}) were not malicious, and would be reported as false positives if being detected.

Table 3.10 shows the P2P botnet detection results which supports our idea that the AVGDDR of legitimate P2P network flow cluster communities is lower than most of the P2P botnets network flow cluster communities. For instance, the AVGDDR of all (60/60) legitimate P2P network flow cluster communities were higher than 0.6, and the AVGDDR of 32 out of 37 botnets were higher than 0.8. The other 5 turned out to be 5 Sality bots, which could be detected by AVGMCR. Also, the legitimate P2P network flow clusters have lower AVGMCR than P2P bots (i.e., $\Theta_{avgmcr} \in [0.15, 0.35]$). For most of the botnets (i.e., ZeroAccess, Waledac, Kelihos and Sality), our system is effective (100% detection rate with zero false positive) and stable over a large range of Θ_{avgddr} (i.e., $[0.0, 0.6]$) and Θ_{avgmcr} (i.e., $[0.15, 0.8]$). Storm has a relative small AVGMCR, hence the effective parameters narrowed down to $\Theta_{avgddr} \in [0.0, 0.6]$ and $\Theta_{avgmcr} \in [0.15, 0.35]$.

3.5.5 Evaluation on Enhanced PeerHunter

3.5.5.1 Analyzing the System Effectiveness

We applied Enhanced PeerHunter on 100 EDs, with Θ_{dd}=30, Θ_{mcr}=0.1, Θ_{avgddr}=0.6 and Θ_{avgmcr}=0.15, and all the results were averaged over 100 EDs. Using Θ_{avgddr}=0.6 and Θ_{avgmcr}=0.15 was based on our empirical study (shown in Table 3.10). As illustrated in Table 3.11, our system

Table 3.10: Botnet detection results for different Θ_{avgddr} and Θ_{avgmcr}. (ZeroA.: the detection rate of ZeroAccess; FP: the number of false positives.)

		Θ_{avgddr}				
Θ_{avgmcr}	-	0.0	0.2	0.4	0.6	0.8
0.0	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	100%
	Storm	100%	100%	100%	100%	100%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	28.9%	29.1%	29.3%	38.1%	34.8%
	Recall	100%	100%	100%	100%	86.5%
	FP	91	90	89	60	60
	F-score	44.8%	45.1%	45.4%	55.2%	49.6%
0.05	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	100%
	Storm	100%	100%	100%	100%	100%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	33.9%	34.2%	34.9%	47.4%	43.8%
	Recall	100%	100%	100%	100%	86.5%
	FP	72	71	69	41	41
	F-score	50.7%	51%	51.7%	64.3%	58.2%
0.1	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	100%
	Storm	100%	100%	100%	100%	100%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	56.0%	56.9%	56.9%	100%	100%
	Recall	100%	100%	81%	100%	86.5%
	FP	29	28	28	0	0
	F-score	71.8%	72.5%	72.5%	100%	92.8%
0.15-0.35	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	100%
	Storm	100%	100%	100%	100%	100%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	100%	100%	100%	100%	100%
	Recall	100%	100%	100%	100%	86.5%
	FP	0	0	0	0	0
	F-score	100%	100%	100%	100%	92.8%
0.4	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	100%
	Storm	84.6%	84.6%	84.6%	84.6%	76.9%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	100%	100%	100%	100%	100%
	Recall	94.6%	94.6%	94.6%	94.6%	78.4%
	FP	0	0	0	0	0
	F-score	97.2%	97.2%	97.2%	97.2%	87.9%
0.6-0.8	ZeroA.	100%	100%	100%	100%	100%
	Waledac	100%	100%	100%	100%	0%
	Storm	0%	0%	0%	0%	0%
	Kelihos	100%	100%	100%	100%	100%
	Sality	100%	100%	100%	100%	0%
	Precision	100%	100%	100%	100%	100%
	Recall	64.9%	64.9%	64.9%	64.9%	43.2%
	FP	0	0	0	0	0
	F-score	78.7%	78.7%	78.7%	78.7%	60.4%

Table 3.11: The number of hosts identified by each component.

-	**Before P2P detection**	**After P2P detection**
# of hosts	10,000	97
-	**After Community detection**	**After Bot detection**
# of hosts	97	37

Table 3.12: Enhanced PeerHunter execution time.

-	**Processing Time**
P2P Host Detection	15 minutes
MCG Extraction	5 minutes
Community Detection	5 minutes
Bot Detection	10 seconds
Total	20 minutes

identified all 97 P2P hosts from 10,000 hosts, and detected all 37 bots from those 97 P2P hosts, with zero false positive, which demonstrated that Enhanced PeerHunter is effective and accurate in detecting P2P botnets.

3.5.5.2 Analyzing the System Scalability

The system scalability is to evaluate the practicality of our systems to deal with the real world big data. First, we applied Enhanced PeerHunter on 100 EDs of 10,000 internal hosts to analyze the processing time of each component. Our system has a scalable design based on efficient detection algorithm and distributed/parallelized computation. As shown in Table 3.11, community detection and botnet detection had negligible processing time compared with P2P network flow detection and MCG extraction, since our first two steps (i.e., P2P network flow detection and MCG extraction) were designed to reduce a huge amount of the hosts subject to analysis (i.e., 99.03% in our experiments). The P2P network flow detection component has linear time complexity, since it scans all the input flows only once to get the flow clusters and further detect P2P flow clusters. However, since it is the very first component to process the input data (data could be large), it still costs the highest processing time (i.e., 15 minutes). To accommodate the growth of a real-world input data, we designed and implemented the P2P network flow detection component using a MapReduce framework, which could be deployed in distributed fashion on scalable cloud computing platforms (e.g., amazon EC2). The MCG extraction component requires pairwise comparison to calculate edges weights. Let n be the number of P2P network flow clusters subject to analysis and m be the maximum number of distinct contacts of a P2P network flow cluster. We implemented

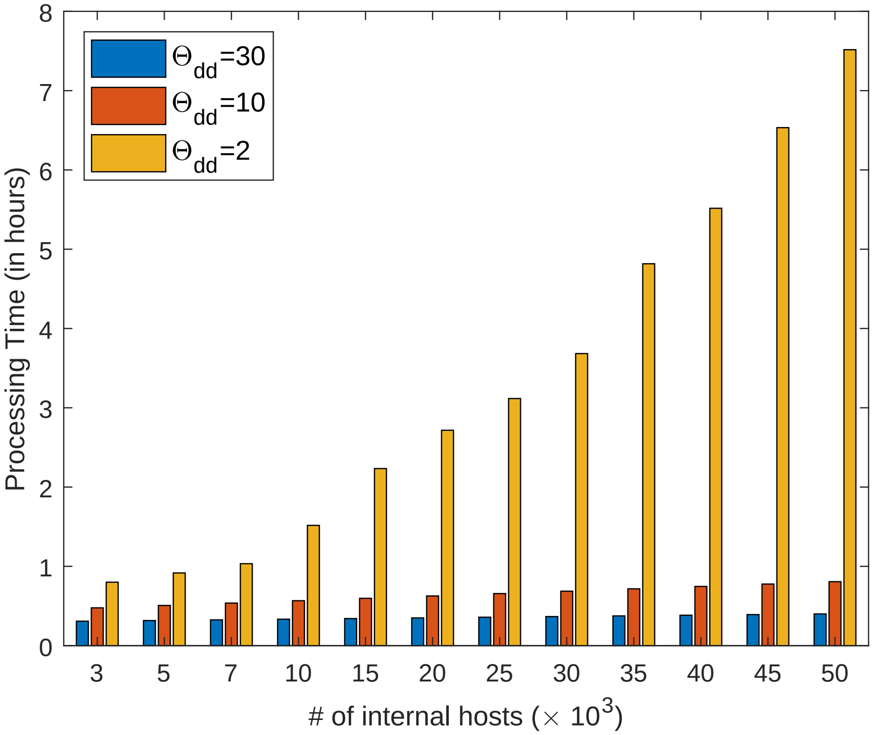


Figure 3.4: Enhanced PeerHunter processing time.

the comparison between each pair of hosts parallelly to handle the growth of n. If we denote k as the number of threads running parallelly, the time complexity of MCG extraction is $O(\frac{n^2m}{k})$. For a given ISP network, m grows over time. Since our system uses a fixed time window (24 hours), for a given ISP network, m tends to be stable and would not cause a scalability issue. Besides, since the percentage of P2P hosts of an ISP network is relatively small (i.e., 3% [1]), an ISP network usually has less than 65,536 (/16 subnet) hosts, and most P2P hosts generate less than 150 P2P network flow clusters (our empirical study), n would be negligible compared with m. Moreover, since the waiting stage bots always act stealthily and only make necessary communications, m also will not be large. We also tested our system using different sizes (i.e., different number of internal hosts) of EDs. For each size, we generated 10 EDs, and recorded the average processing time of our system with different Θ_{dd}. As shown in Figure 3.4, compared with the size of datasets, Θ_{dd} has more influence on the system scalability. Because in our P2P network flow detection component, Θ_{dd} has an impact on n (the number of P2P network flow clusters subject to analysis), and larger Θ_{dd} leads to smaller n, thus less processing time. For instance, when $\Theta_{dd} = 10$ or 30, the increase of processing time, caused by increasing the size of data, was much less than when $\Theta_{dd} = 2$. Therefore,

Table 3.13: Comparison of the community detection results under attack.

-	PeerHunter [5]		Enhanced PeerHunter	
	No Attack	PMMKL	No Attack	PMMKL
BSI	1.00 ± 0.00	0.73 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
BAI	1.00 ± 0.00	0.81 ± 0.01	0.85 ± 0.00	0.85 ± 0.00
BLSI	1.00 ± 0.00	0.78 ± 0.01	1.00 ± 0.00	1.00 ± 0.00

our system is very scalable on different sizes of data with an appropriate Θ_{dd} (e.g., 10 or 30). Also, by tuning Θ_{dd}, our system has the potential to deal with different size of datasets in a reasonable time. To summarize, Enhanced PeerHunter is scalable to handle the real world network data.

3.5.5.3 Analyzing the Effectiveness of System Parameters

Although we had analyzed the effectiveness of Θ_{dd}, Θ_{avgddr} and Θ_{avgmcr} within the corresponding components, the effectiveness of combinations among different values of Θ_{dd}, Θ_{avgddr} and Θ_{avgmcr} has not been studied. As shown in Figure 3.5, we used precision, recall and false positives to evaluate the effectiveness of different parameter combinations. As discussed in Section 3.5.2, Θ_{dd} is used to detect P2P network flow clusters. Larger Θ_{dd} tends to result in more false negatives (lower recall), and smaller Θ_{dd} tends to result in more false positives (lower precision). For instance, changing Θ_{dd} from 30 or 50 to 10 resulted in 47 or 42 more false positives ($\Theta_{avgddr} = 0.15$) as shown in Figure 3.5c and Figure 3.5f, respectively. When $\Theta_{dd} \in \{30, 50\}$, $\Theta_{avgddr} \in [0.15, 0.35]$ and $\Theta_{avgddr} \in [0.2, 0.6]$, our system yielded 100% detection rate with zero false positive. Even when $\Theta_{dd} = 10$, our system can still work effectively with $\Theta_{avgddr} \in [0.25, 0.35]$ and $\Theta_{avgddr} \in [0.2, 0.6]$. This demonstrated our system can work effectively over several different parameter combinations.

3.5.5.4 Analyzing the "True" False Positives

In this section, we discuss about some interesting findings about the false positives resulted from setting $\Theta_{dd} = 10$. As discussed in Section 3.3.3.2, Θ_{avgddr} is used to capture the "P2P behavior" of network flows, and Θ_{avgmcr} is used to capture the "botnet behavior" of network flows. Hence, if we use a larger Θ_{avgddr} (i.e., 0.6) and a smaller Θ_{avgmcr} (i.e., 0.0), most of the false positives should be legitimate P2P host. For instance, in Figure 3.5f, when $\Theta_{dd} = 10$, $\Theta_{avgddr} = 0.6$ and $\Theta_{avgmcr} = 0.0$, 115 out of 118 false positives were P2P hosts (60 from D_{p2p} and 55 from D^{b}_{p2p}).

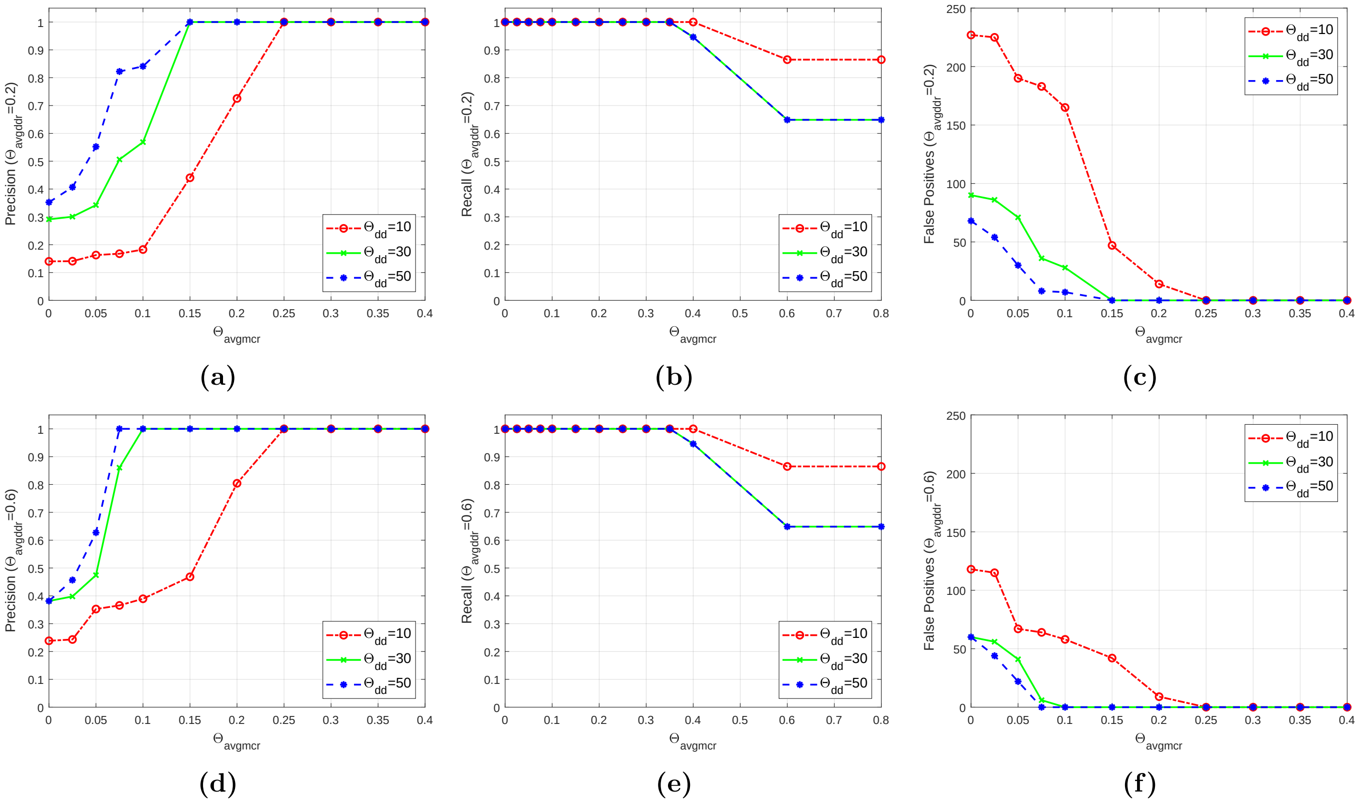
On the other hand, we assume that if we use a smaller Θ_{avgddr} (i.e., 0.2) and a larger Θ_{avgmcr} (i.e., 0.15), some of the false positives might come from the other types of botnets. As shown in Figure 3.5c, when $\Theta_{dd} = 10$, $\Theta_{avgddr} = 0.2$ and $\Theta_{avgmcr} = 0.15$, 9 out of 47 false positives were not our known legitimate P2P hosts. We investigated these false positives, with their anonymized and payload-free network traces. It turned out that, 4 out of the 9 false positives (i.e., "180.217.2.181", "180.217.2.246 ", "180.217.2.248" and "180.217.2.177") were listed in the Barracuda Reputation Block List (BRBL) [68], a highly accurate list of the IP addresses known to send spam. Hence, we are convinced that those false positives were infected with virus or botnets. These interesting "true" false positives findings demonstrated that our system has the potential to detect other unknown botnets.

3.5.6 Mimicking Legitimate P2P Application Attacks (MMKL)

Our work is focusing on detecting P2P botnets from legitimate P2P applications. If the adversaries (e.g., botmasters) know our techniques in advance, they might attempt to evade our system via instructing P2P bots to mimic the behavior of legitimate P2P applications. Inspired by [1], in this section, we propose two evasion attacks. All the parameters used in experiments of this Section were the same as in Section3.5.5.

3.5.6.1 Passive MMKL (PMMKL)

In this attack, the botmaster can instruct the bots to passively generate network traffic together with legitimate P2P applications running on the same machine at the same time. As such, the botnet traffic will be overlapped with the legitimate P2P traffic. Since during most of the time, P2P botnets will be acting stealthily, the legitimate P2P traffic will dominate the host level behavior. Hence, the attack could effectively evade the host level group behavior based methods [4, 5]. Also, the attack does not require the botnets to generate more or new types of network flows, and just need to monitor the legitimate P2P application activities, which can evade certain anomaly-based methods. Since our detection algorithm is based on network-flow level mutual contacts graph, which could differentiate the network flows coming from different P2P applications, it is capable of detecting P2P bots while the bots traffic and the legitimate P2P traffic are overlapped on the same host.



(a) (b) (c)

(d) (e) (f)

Figure 3.5: Precision, recall and false positives.

Table 3.14: Botnet detection results under no attack and PMMKL attack. (* detection rate)

-	PeerHunter [5]		Enhanced PeerHunter		Zhang *et al.* [1] ($\Theta_{bot} = 0.6$)		Zhang *et al.* [1] ($\Theta_{bot} = 0.8$)	
	No Attack	PMMKL	No Attack	PMMKL	No Attack	PMMKL	No Attack	PMMKL
ZeroAccess*	100%	0%	100%	100%	100%	82.5%	100%	90%
Waledac*	100%	0%	100%	100%	100%	0%	100%	60%
Storm*	100%	37.5%	100%	100%	97.8%	61.5%	100%	95.4%
Kelihos*	100%	0%	100%	100%	85.5%	45%	85.5%	77.5%
Sality*	100%	79.2%	100%	100%	89.6%	80%	96.8%	88%
Precision	100%	99.1%	100%	100%	100%	100%	60.8%	62.7%
Recall	100%	23.9%	100%	100%	94.7%	60%	96.4%	86.5%
FP	0/9,963	39/9,926	0/9,963	0/9,926	0/9,963	0/9,926	23/9,963	19/9,926
F-score	100%	38.5%	100%	100%	97.3%	75%	74.6%	72.7%

Table 3.15: Effort needed to completely evade Enhanced PeerHunter under AMMKL.

-	**# of P2P flow clusters**	**# of peers per flow cluster**	**# of peers per host**	γ	**extra # of peers needed**
ZeroAccess	3	686	2,058	220%	4,528
Waledac	171	244	41,724	180%	75,104
Storm	67	740	49,580	80%	39,664
Kelihos	15	252	3,780	200%	7,560
Sality	1,158	918	1,063,044	80%	850,436

To simulate this attack on each ED, we randomly selected 37 hosts out of the 60 legitimate P2P application hosts, and randomly mapped their IPs to 37 bots' IPs. By doing this, the traffic of each bot were overlapped with the traffic of one legitimate P2P host. And we made a comparison between Enhanced PeerHunter and PeerHunter [5] under this attack, where PeerHunter [5] was using one of its best parameter setting Θ_{dd}=50, Θ_{mcr}=0.05, Θ_{avgddr}=0.06 and Θ_{avgmcr}=0.2. As shown in Table 3.13, all three community detection indices (i.e., BSI, BAI and BLSI) decreased around 20% while running PeerHunter under this attack. However, PMMKL had no effects on Enhanced PeerHunter's community detection performance. As shown in Table 3.14, PMMKL completely failed PeerHunter in detecting ZeroAccess, Waledac and Kelihos, and dramatically reduced the detection rate of Storm and Sality. On the contrary, PMMKL had no affects on Enhanced PeerHunter's P2P botnet detection performance.

To summarize, compared with our previous work, Enhanced PeerHunter can detect P2P botnets effectively even if bots are running on the same host as legitimate P2P applications.

3.5.6.2 Active MMKL (AMMKL)

In this attack, the botmaster can instruct the bots to mimic the behaviors of legitimate P2P applications actively. For instance, each bot can actively communicate with an extra set of randomly selected peers to decrease the rate of mutual contacts between a pair of bots. Compared with PMMKL, in AMMKL, bots do not need to monitor and wait until some legitimate P2P application running to work. However, communicating with much more extra but unnecessary peers will lead the botnets to act less stealthy and less efficient, and enable certain anomaly-based methods (e.g., high volumes of network traffic) to detect them.

To simulate this attack on each ED, after the P2P network flow detection procedure, for each botnet network flow cluster that communicates with n peers, we inserted certain network flows communicating with an extra of $\gamma * n$ randomly selected peers. As shown in Figure 3.6, our community detection component is robust to AMMKL, since both BAI and BLSI were unchanged and only BSI dropped a little bit when γ increased. When combining both attacks, both BSI and BAI dropped a lot, and BLSI dropped from 1.0 to around 0.88, as γ increasing from 0.0 to 3.0. This is because when combining both attacks, as γ increasing, the community detection component tends to cluster different types of bots into the same community and separate the same type of bots into different communities. The good news is, it can still well separate bots and legitimate P2P hosts into different communities. In summary, even though combining both attacks makes it harder for our method to separate different or aggregate the same type of bots, Enhanced PeerHunter is still robust in separating P2P bots from other hosts in the community detection process.

As shown in Figure 3.7c, both scenarios (i.e., AMMKL and combining both attacks) did not introduce new false positives (i.e., precisions equals to 1.0). Compared with conducting AMMKL, combining both attacks has more influences on the dropping of detection rate. Figure 3.7a and Figure 3.7b illustrate the detection rate of each botnet under two different scenarios, where the detection rate of different botnets started to drop around different γ. Table 3.15 shows the analysis of all 5 botnets. Take Storm for instance, to affect the detection of Storm, each P2P network flow cluster of Storm needs to communicate with at least an extra 40% of its current peers, and in order to completely evade our system, γ needs to be increased to 80%. Consider the fact that each Storm host generates an average of 67 P2P network flow clusters in 24 hours, and each network flow cluster communicates to an average of 740 peers. As such, to completely evade our system, each Storm host must communicate with at least an extra of $67 \times 740 \times 80\% \approx 39,664$ peers. In this case, it makes the P2P botnet less stealthy, less efficient and more exposed to trigger anomaly-based P2P botnet detection approaches [69]. In conclusion, although our system could not completely mitigate AMMKL, conducting AMMKL makes the botnets less stealthy, less efficient and more exposed, which still shows a winning of our system against P2P botnets.

3.5.7 Comparison to Zhang *et al.* [1]

We compared our system to one of the state of art P2P botnet detection system Zhang *et al.* [1]. They proposed a scalable botnet detection system capable of detecting stealthy P2P botnets (i.e., in the waiting stage), where no knowledge of existing malicious behavior is required in advance. The system first applies a two-step flow clustering approach to create the fingerprints of hosts that have engaged in P2P activities. Afterwards, it applies two layers of filtering to detect potential P2P bots: a coarse-grained filtering to detect "persistent" P2P hosts that have longer active time of P2P behaviors, and a fine-grained filtering that applies hierarchical clustering to group pairs of P2P hosts that have less distance between their fingerprints. Our system shares many similarities with Zhang *et al.* [1]. For instance, both systems are (a) using network flow-based approach, (b) using unsupervised approach (i.e., no knowledge of existing malicious behaviors are required and have the potential to detect unknown botnets), (c) claiming to work while the botnet traffic are overlapped with the legitimate P2P traffic on the same set of hosts, (d) designed to have the built-in scalability, and (e) deployed at the network boundary (e.g., gateway), thus could be evaluated on the same datasets.

The main differences between our system and Zhang *et al.* [1] are listed as follows. First, two systems are using different network flow features. Zhang *et al.* [1] uses the absolute number of bytes and packets of each flow; Enhanced PeerHunter uses the bytes-per-packet rate of each flow. Second, two systems are using different approach to cluster network flows (i.e., at different granularity). Zhang *et al.* [1] uses a two-step distance-based clustering (i.e., k-means, BIRCH) to cluster network flows of similar feature values; Enhanced PeerHunter clusters the network flows that have exactly the same feature values. Third, two systems apply the botnet detection step at different levels (i.e., host-level or network-flow-level). Zhang *et al.* [1] uses the distance between each pair of hosts to detect bots; Enhanced PeerHunter uses the distance between each pair of network flows to detect botnet network flow communities and then further identify the corresponding bots. Last but not least, two systems are using different heuristics to detect botnets. Zhang *et al.* [1] uses an threshold on the height of the hierarchical clustering dendrogram to detect bot clusters, which is very sensitive to the experimental datasets (as shown in Table 3.14); Enhanced PeerHunter uses network-flow level community behavior analysis (i.e., AVGDDR and AVGMCR) to identify botnet (network flow) communities, which is more robust to the proposed attacks and can also be extended to other/new community behaviors.

We implemented a prototype system of Zhang *et al.* [1], since Zhang *et al.* [1] did not have a publicly available implementation. Most of our implementations followed the description as in [1], other than the system parallelization, which has no impact on the system effectiveness evaluation. The experimental datasets used in both works are also different. For instance, we evaluated our system on 100 synthetic experimental datasets (of different background traffic and different topology, as described in Section 3.5.1.3) and took the average results; Zhang *et al.* [1] was evaluated on single customized dataset. Furthermore, even though both datasets use the same 24 hours time window, our datasets have much more internal hosts (i.e., 10,000 vs. 953), higher legitimate P2P hosts to P2P bots ratio (i.e., 727:37 vs. 8:16), and more types of botnets (i.e., 5 vs. 2). To summarize, our experimental datasets is more challenging and comprehensive.

We applied our implemented Zhang *et al.* [1] on the same experimental datasets as Enhanced PeerHunter under two circumstances (i.e., No Attack and PMMKL). We followed the same settings for most of the system parameters as described in [1], such as $\Theta_{BGP} = 50$, $\Theta_{p2p} = 0.5$, $K = 4,000$, $\lambda = 0.5$. Since the default value of Θ_{bot} (i.e., 0.95) used by the original paper, did not perform well on our dataset, we evaluated Zhang *et al.* [1] using two other different well selected values of Θ_{bot} (i.e., 0.6 and 0.8) that shows better results.

From the experimental results (Table 3.14), we achieved several observations as follows. First, Zhang *et al.* [1] is more sensitive to the experimental dataset. For instance, Zhang *et al.* [1] was reported to achieve 100% detection rate and 0.2% false positive rate on their own datasets (using $\Theta_{bot} = 0.95$), while could not achieve similar results on our datasets using either the default parameter ($\Theta_{bot} = 0.95$) or the well selected parameter ($\Theta_{bot} = 0.6$ or $\Theta_{bot} = 0.8$). Second, as discussed in Section 3.5.5, our system is more stable and effective over a large range of system parameters (Θ_{avgddr} and Θ_{avgmcr}), while Zhang *et al.* [1] is more sensitive to its system parameter (Θ_{bot}). For instance, Zhang *et al.* [1] had higher precision (lower false positives) and lower recall (higher false negatives) while using $\Theta_{bot} = 0.6$ comparing with using $\Theta_{bot} = 0.8$. Third, our system outperforms Zhang *et al.* [1] in terms of the detection rate of different botnets, the overall precision, recall and false positives. For instance, our system achieved 100% detection rate with zero false positives under different circumstances, while Zhang *et al.* [1] failed to detect all the bots under both well selected parameters. At last, our system is more robust to PMMKL attack. For instance, PMMKL attack had no impact on the effectiveness of our system, while decreasing the F-score of Zhang *et al.* [1] from 97.3% to 75% ($\Theta_{bot} = 0.6$) or from 74.6% to 72.7% ($\Theta_{bot} = 0.8$).

3.6 Discussion

3.6.1 Evasions and Possible Solutions

To avoid being detected by Enhanced PeerHunter, the botmaster could use a combination of the following three approaches: (a) adding randomized paddings or junk packets to influence the bytes-per-packet characteristics for network flow clustering, (b) reducing the number or rate of destination diversity, or (c) reducing the number or rate of mutual contacts. To deal with the randomized spatial-communication behavior, we could adopt more time-communication features, such as packet/flow duration and inter-packet delays, or apply more generalized features, such as the distribution, mean or standard deviation of bytes-per-packet. The other two evasion approaches would be the victory of our system. On one hand, to reduce the number or rate of destination diversity, a bot has to limit its communication to the network of certain locations, which degrades the P2P botnet into a centralized fashion. On the other hand, reducing the number of mutual contacts means there will be less bots targeting on the same set of victims, and less bots playing the role as botmasters, which will jeopardize the effectiveness and the decentralized structure of a P2P botnet. Also, as shown in Section 3.5.6.2, reducing the rate of mutual contacts while maintaining the same number of mutual contacts (i.e., by conducting AMMKL) will make the botnets less stealthy, less efficient and more exposed to the other detection systems (e.g., anomaly-based botnet detection using high volumes of network traffic).

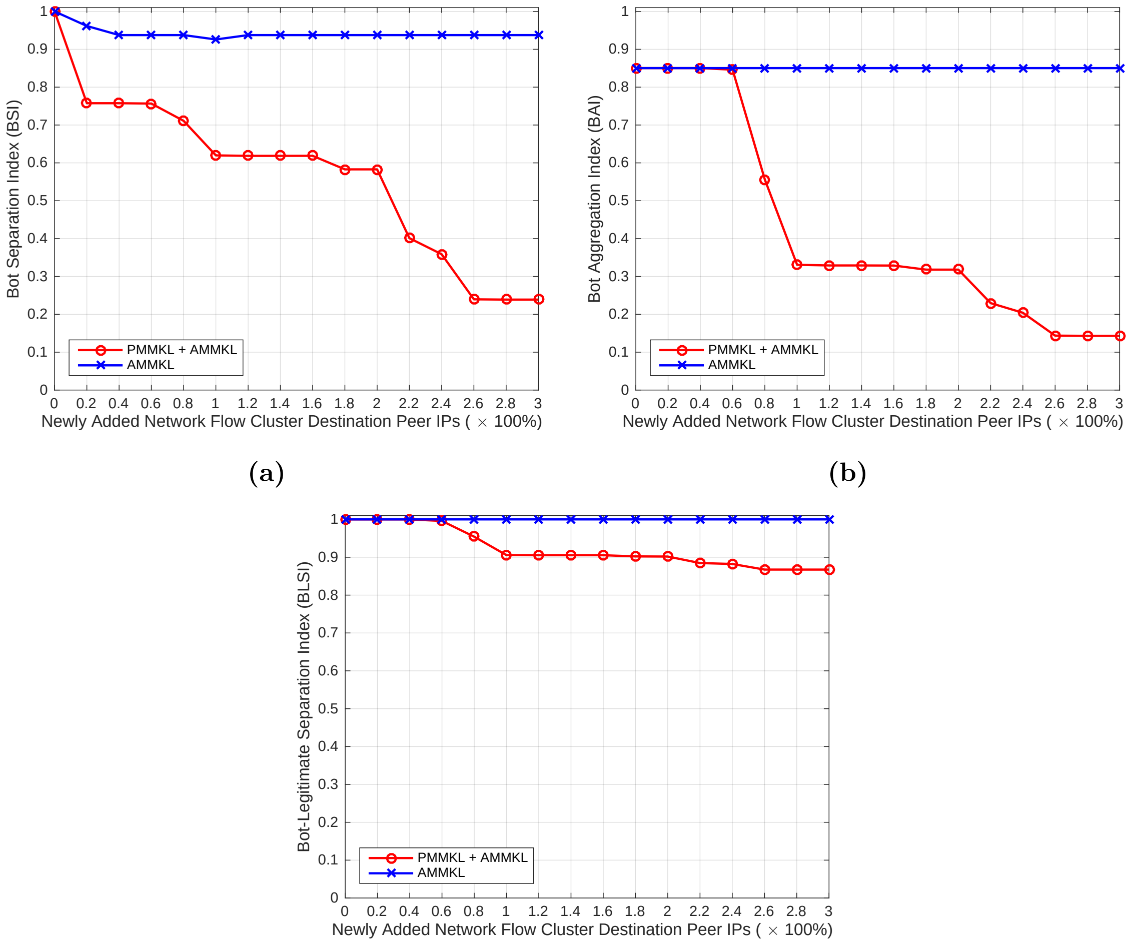
3.6.2 The Deployment of Enhanced PeerHunter

In the previous sections, we simply assumed that our system is deployed at the boundary of a single organization. In this section, we discuss about the deployment of Enhanced PeerHunter in three more realistic scenarios.

- If The number of bots within an organization is too small, it would be challenging to build the MCG of botnet communities (i.e., the number of bots belonging to the same botnet is less than 3). In this case, we can deploy multiple Enhanced PeerHunter systems at the boundaries of multiple organizations, and correlate the network flows collected by those multiple Enhanced PeerHunter systems to build an appropriate size of MCG to detect botnet communities.

- If the number of bots within an organization is too large, the mutual contacts of certain bots might be within the organization internal network, hence invisible to the single system monitoring at the network boundary. In this case, we can deploy multiple Enhanced PeerHunter systems within the organization, that divide the organization network into several appropriate size of sub-internal networks. Each system is responsible for one sub-internal network.

- If the botmaster knows the system deployment location, the botmaster could assign the location of bots or control the communications of the bots based on the knowledge of the system



(a)

(b)

(c)

Figure 3.6: Community detection results under attacks. (a) Bot Separation Index (BSI). (b) Bot Aggregation Index (BAI). (c) Bot-Legitimate Separation Index (BLSI).

deployment location to evade our system. For instance, the botmaster could assign bots into different sub-internal networks, and instruct most of the bots communicate with the others within the same sub-internal network. In this case, we could use the concept and idea of Moving Target Defense (MTD) [70] to develop a strategy that makes it more difficult for botmasters to learn the deployment locations of our systems, by dynamically changing the settings or deployments of our systems.

3.6.3 Extend Enhanced PeerHunter to Detect Other Botnets

Although Enhanced PeerHunter is designed to detect P2P botnets, our idea of using mutual contacts graph has the potential to detect not only unknown botnets, but also the other types of

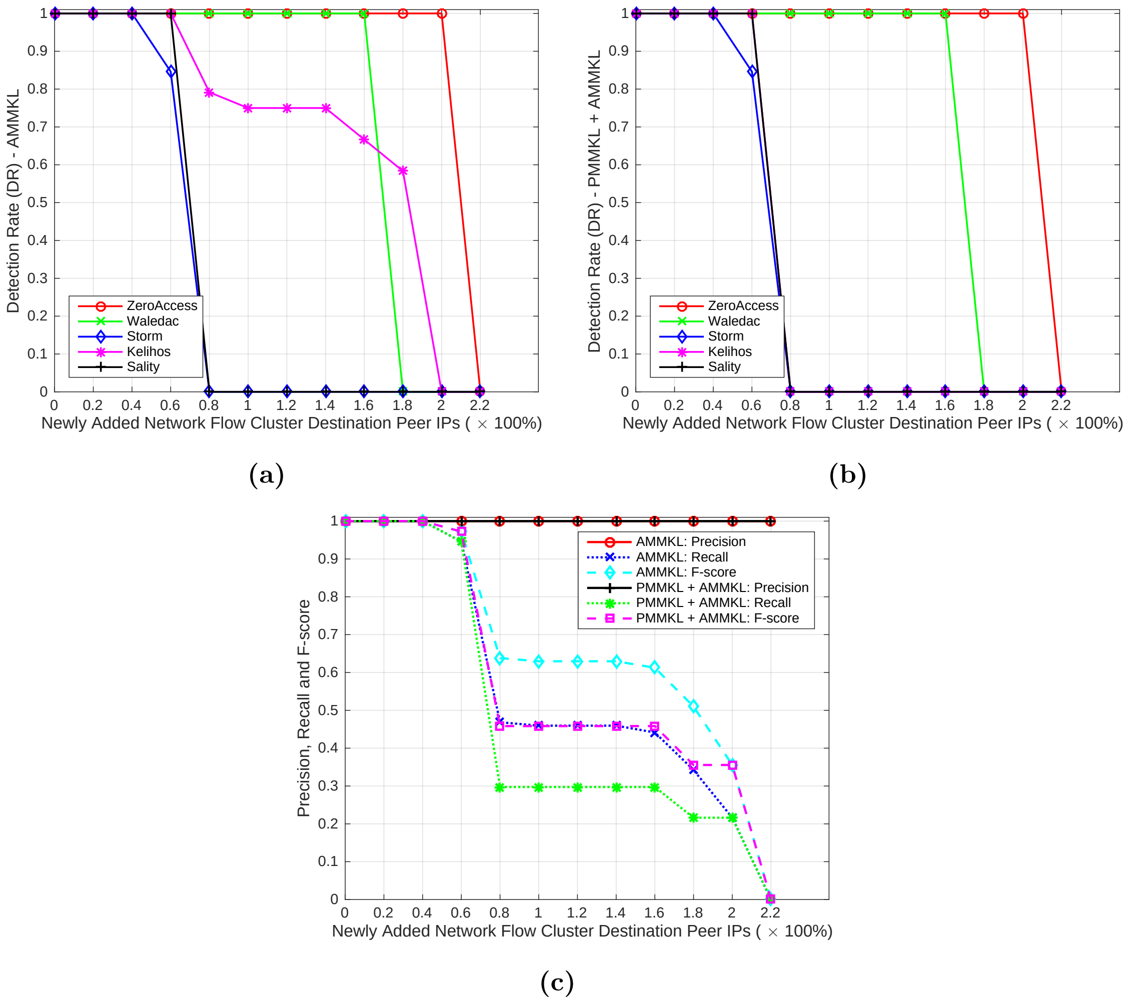


Figure 3.7: The P2P botnet detection results of Enhanced PeerHunter. (a) P2P botnet detection rate when conducting AMMKL. (b) P2P botnet detection rate when combining PMMKL and AMMKL. (c) Precision, recall and F-score, when conducting AMMKL, and when combining PMMKL and AMMKL.

botnets (e.g., centralized botnets, such as IRC botnets [60], mobile botnets [71]). Since bots are usually controlled by machines, rather than humans, bots from the same botnets tend to communicate with a similar set of peers or attacking targets. For instance, bots from the same IRC botnets tend to contact a similar set of C&C servers, while bots from the same mobile botnets tend to contact a similar set of satellite servers. Hence, we argue that Enhanced PeerHunter could be easily extended to detect the other types of botnets.

3.7 Conclusion

We present a novel community behavior analysis based P2P botnet detection system, Enhanced PeerHunter, which operates under several challenges: (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) the botnet traffic are overlapped with legitimate P2P traffic on the same host; (d) no bot-blacklist or "seeds" are available; (e) none statistical traffic patterns known in advance; and (f) does not require to monitor individual host. We propose three types of community behaviors (i.e., flow statistical features, numerical community features and structural community features) that can be used to detect P2P botnets effectively. In the experimental evaluation, we propose a network traces sampling and mixing method to make the experiments as unbiased and challenging as possible. Experiments and analysis were conducted to show the effectiveness and scalability of our system. With the best parameter settings, our system achieved 100% detection rate with none false positives. We also propose two mimicking legitimate P2P application attacks (i.e., PMMKL and AMMKL). The experiment results showed that our system is robust to PMMKL, and will make the botnets less stealthy, less efficient and more exposed while conducting AMMKL.

Chapter 4: Dynamic Community Detection by Incrementally Maximizing Modularity

Community detection is of great importance for online social network analysis. The volume, variety and velocity of data generated by today's online social networks are advancing the way researchers analyze those networks. For instance, real-world networks, such as Facebook, LinkedIn and Twitter, are inherently growing rapidly and expanding aggressively over time. However, most of the studies so far have been focusing on detecting communities on the static networks. It is computationally expensive to directly employ a well-studied static algorithm repeatedly on the network snapshots of the dynamic networks. We propose DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities of dynamic networks as effective as repeatedly applying static algorithms but in a more efficient way. DynaMo is an adaptive and incremental algorithm, which is designed for incrementally maximizing the modularity gain while updating the community structure of dynamic networks. In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real-world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm. [3]

4.1 Introduction

With the advance of online social network analysis, more and more real-world systems, such as social networks [73], collaboration relationships [74], recommendation systems [75] and intrusion detection system [5, 51], are represented and analyzed as networks, where the vertices represent certain objects and the edges represent the relationships or connections between the objects. Most social networks have been shown to present certain community structures [76], where vertices are densely connected within communities and sparsely connected between communities. Community

[3] This chapter was published in IEEE Transactions on Knowledge and Data Engineering (2019) [72]. Copyright permission is included in Appendix A.

detection is one of the most important and fundamental problem in the field of graph mining, network science and social network analysis.

Detecting community structure is of great challenge, and most of the recent studies are proposed to detect communities in the static networks, such as spectral clustering [6], label propagation [7], modularity optimization [8], and k-clique communities [9]. However, real-world networks, especially most of the online social networks, are not static. Most popular online social networks (e.g., Facebook, LinkedIn and Twitter) are de facto growing rapidly and expanding aggressively in terms of either the size or the complexity over time. For instance, in Facebook network, the updating of its community structure could be simply caused by new users joining in, old users leaving, or certain users connecting (i.e., friend) or disconnecting (i.e., unfriend) with the other users. Facebook announced that it had 1.52 billion daily active users in the fourth quarter of 2018 [10], which shows a 9% increase over the same period of the previous year, and 4 million likes generated every minute as of January 2019 [11]. Hence, it is rather important and impending to enable community detection in such dynamic networks.

Designing an effective and efficient algorithm to detect communities in dynamic networks is highly difficult. First, an efficient algorithm should update the communities adaptively and incrementally depending on the changes of the dynamic networks, and avoid redundant and repetitive computations. Second, it is hard to design a dynamic algorithm that performs as effective as the static algorithms by only observing the historical community structures and the incremental changes of the dynamic networks. Third, it is still quite open about how to categorize the incremental changes of dynamic networks, and how to assess the influence of different types of the incremental changes on the community structure updates, which is rather important to design an effective and efficient dynamic algorithm.

A few algorithms have been proposed to detect communities in dynamic networks [77, 78, 13, 15, 79, 16, 80, 14, 81]. An intuitive way to detect communities in dynamic networks is to slice the network into small snapshots based on the timestamps, and directly employ well-studied static algorithms repeatedly on each network snapshot. However, these algorithms [77, 78] usually are computational expensive, since they compute the current community structures completely independent from the historical information (i.e., the previous community structures), especially when the dynamic network changes rapidly and the time interval between two consecutive network snapshots are extremely small. Another way to update the communities is using not only the current network changes but also the previous community structures. These algorithms [13, 15, 79, 16, 80, 14, 81] adaptively and incrementally detect communities in dynamic networks, without re-executing any static algorithms on each entire network snapshot. Those algorithms are usually more efficient than repeatedly applying static algorithms on network snapshots. However, most of those algorithms are still not practical enough to be directly used to analyze the real-world networks. For instance, some algorithms [13, 16] only considers vertices/edges additions, while vertices/edges deletions happen quite often in online social networks (e.g., "unfriend" in Facebook). Some algorithms [13, 15, 16, 14] only consider unweighted networks, which are not applicable for weighted networks. Furthermore, some algorithms [82, 83, 81] need certain prior information about the community structures (e.g., the number of communities, the ratio of vertices in overlapped communities) or need certain predefined parameters which are not available in practice.

We present DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect non-overlapped communities of dynamic networks. DynaMo is an adaptive and incremental algorithm designed for maximizing the modularity gain while updating the community structure of dynamic networks. To update the community structures efficiently, we model the dynamic network as a sequence of incremental network changes. We propose 6 types of incremental network changes: (a) intra-community edge addition/weight increase, (b) cross-community edge addition/weight increase, (c) intra-community edge deletion/weight decrease, (d) cross-community edge deletion/weight decrease, (e) vertex addition, and (f) vertex deletion. For each incremental network change, we design an operation to maximize the modularity.

In the experimental evaluation, a comprehensive comparison has been made among DynaMo, Louvain (static) [12] and 5 dynamic algorithms (i.e., QCA [13], Batch [14], GreMod [15], LBTR-LR [16] and LBTR-SVM [16]). Extensive experiments have been conducted on 6 large-scale real-world networks and 10,000 synthetic networks. Our results show that DynaMo consistently outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm. To summarize, our work has the following contributions:

- We present a novel, effective and efficient modularity-based dynamic community detection algorithm, DynaMo, capable of detecting non-overlapped communities in real-world dynamic networks.

- We present the theoretical analysis to show why/how DynaMo could maximize the modularity, while avoiding certain redundant and repetitive computations.

- A comprehensive comparison among our algorithm and the state-of-the-art algorithms has been conducted (Section 4.5). For the sake of reproducibility and convenience of future studies about dynamic community detection, we have released our prototype implementation of DynaMo, the experiment datasets and a collection of the implementations of the other state-of-the-art algorithms. [4]

The rest of this chapter is organized as follows: Section 4.2 presents the related work. Section 4.3 presents the notations, the concept of dynamic networks and the definition of modularity, and introduces a baseline static community detection algorithm (i.e., Louvain algorithm). Section 4.4 describes our algorithm design and theoretical analysis. Section 4.5 presents the experimental evaluation. Section 4.6 concludes.

4.2 Related Work

To date, a few dynamic community detection approaches were proposed [84, 77, 78, 82, 83, 85, 86, 87, 81, 88, 13, 14, 15, 16, 89]. Rossetti et al. [90], a comprehensive survey on dynamic community detection, divide most of the algorithms into three categories (i.e., instant-optimal, temporal trade-off and cross-time) in terms of their ability to solve the community instability and temporal smoothing issue. However, some approaches in the literature are not belonging to any of these categories. For instance, some approaches, such as our proposed algorithm, do not consider the community instability and temporal smoothing issue, but still aim to detect communities in dynamic networks effectively and efficiently. The concept of "cross-time" approaches is also beyond the scope of this chapter. After incorporating the taxonomy of [90], we consider three categories of related approaches: instant-optimal, temporal trade-off and incremental approaches (to replace the "cross-time" in [90]).

The instant-optimal approaches [84, 77, 78] have two steps: (i) static algorithms are applied on each network snapshot independently to detect static communities, (ii) communities detected on each network snapshot are matched with communities detected on the previous one. Greene

[4] https://github.com/nogrady/dynamo

et al. [84] proposed a general model for tracking communities in dynamic networks via solving a classic cluster matching problem on the communities independently detected on consecutive network snapshots. Such approaches take advantage of existing static algorithms. However, repeatedly applying static algorithms on all network snapshots of the dynamic networks is computationally expensive.

The temporal trade-off approaches [82, 83, 85, 86, 87, 81, 88] incorporate the community detection and tracking via considering the community structures of the current and historical network snapshots at the same time. Those approaches aim to maintain the evolution of the community structures of the dynamic networks, where the community structure (e.g., the number of communities, the size of communities) of the current network snapshot should be similar to that of the previous one. Tang et al. [85] propose a temporally regularized clustering algorithm to identify evolving groups in dynamic networks, where they use a metric that attempts to optimize two objectives: the quality of the current community structure and the similarity between the current and the previous community structures. However, most of those approaches, such as [82, 83, 88], require determining the number of communities to be detected/tracked in advance, which is rather impractical for the real-world dynamic networks where the number of communities changes over time.

The incremental approaches [13, 14, 15, 16, 89] adaptively update the community structures fully based on the network changes happened during the current snapshot and the community structure of the previous snapshot. For instance, GreMod [15] is a rule-based incremental algorithm that performs the predetermined operations on different types of the edge addition changes of the dynamic network. QCA [13] is another rule-based adaptive algorithm that updates the community structures according to the predefined rules of different types of the incremental changes (i.e., vertices/edges addition/deletion) on the dynamic network. QCA is also one of the most efficient dynamic community detection algorithms in the literature. However, since the rule-based algorithms, such as GreMod [15] and QCA [13], considers each network change as an independent event, they are less efficient when abundant (i.e., a batch of) network changes appear in the same network snapshot. Chong et al. [14] propose a batch-based incremental modularity optimization algorithm that updates the community structures by initializing all of the new and changed vertices of the current network snapshot (i.e., the batch) as singleton communities and using Louvain algorithm to further update the community structures. However, since their initialization approach, that generates the intermediate community structure of a batch of network changes, is rather coarse, it is less efficient to apply Louvain algorithm on those intermediate community structures. LBTR [16] is a learning-based framework that uses machine learning classifiers and historical community structure information to predict certain vertices' new community assignments after each round of network changes. In those learning-based algorithms, once the models are being trained, the testing phase could be very efficient. However, since the supervised nature of the learning-based algorithms, it would be extremely hard to generalize the trained models. For instance, the models trained on one type of dynamic networks (e.g., social network) might be less effective to another type of dynamic networks (e.g., collaboration network). Furthermore, even for the same dynamic network, the network patterns change over time. Thus, the models have to be updated periodically, which would be rather illogical, since the network usually changes rapidly and updating models is also time consuming.

Our proposed approach, DynaMo, is an adaptive and incremental algorithm. Compared with rule-based algorithms [15, 13], our approach is capable of processing a set of network changes as a batch, and redesigns the "rules" by considering more extreme cases (Section 4.4.3). Compared with batch-based algorithms [14], our approach has a more fine-grained initialization phase (Section 4.4.3), which could reduce the computation time dramatically. Compared with learning-based algorithms [16], our approach is more generalized to real-world networks. In Section 4.5, we compare DynaMo with Louvain algorithm and 5 other dynamic algorithms on 6 real-world networks and 10,000 synthetic networks, showing that DynaMo consistently outperforms all the other 5 dynamic algorithms in terms of effectiveness, and much more efficient than Louvain algorithm.

4.3 Preliminaries

In this section, we introduce 1) the notations; 2) the dynamic network model; 3) modularity, to quantify the quality of a community structure; and 4) Louvain algorithm, a modularity-based static community detection approach.

4.3.1 Notations

Let $G = (V, E)$ be an undirected weighted network, where V is a set of vertices ($n = |V|$), E is a set of undirected weighted edges ($m = |E|$), and there could be more than one edge between a pair of vertices. Let C denote a set of disjoint communities associated with G, A_{ij} denote the sum of the weights of all the edges between vertices i and j, k_i denote the sum of the weights of all the edges linked to vertex i, and c_i denote the assigned community of vertex i.

4.3.2 Dynamic Network

Let $G^{(t)}$ denote the snapshot of a network at time t, and $\triangle G^{(t)} = (\triangle V^{(t)}, \triangle E^{(t)})$ denote the incremental change from $G^{(t)}$ to $G^{(t+1)}$ (i.e., $G^{(t+1)} = G^{(t)} \cup \triangle G^{(t)}$), where $\triangle V^{(t)}$ and $\triangle E^{(t)}$ are the sets of vertices and edges being changed during time period $(t, t+1]$. A dynamic network G is a sequence of its network snapshots changing over time: $G = \{G^{(0)}, G^{(1)}, \ldots, G^{(t)}\}$.

4.3.3 Modularity

Modularity [91] is a widely used criteria to evaluate the quality of given network community structure. Community structures with high modularity have denser connections among vertices in the same communities but sparser connections among vertices from different communities. Given network $G = (V, E)$, its modularity is defined as follows:

$$Q = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{k_i k_j}{2m}] \delta_{ij} = \frac{1}{2m} \sum_{c \in C}^{c} (\alpha_c - \frac{\beta_c^2}{2m}) \tag{4.1}$$

where $\alpha_c = \sum_{i,j \in c} A_{ij}$, $\beta_c = \sum_{i \in c} k_i$ and δ_{ij} equals to 1, if i, j belong to the same community, otherwise equals to 0.

4.3.4 Louvain Method for Community Detection

Since the modularity optimization problem is known to be NP-hard, various heuristic approaches are proposed [92, 93, 8]. Most of the algorithms have been superseded by Louvain algorithm [12], which attempts to maximize the modularity using a greedy optimization approach composed of three steps: (i) Initialization, where each vertex forms a singleton community. (ii)

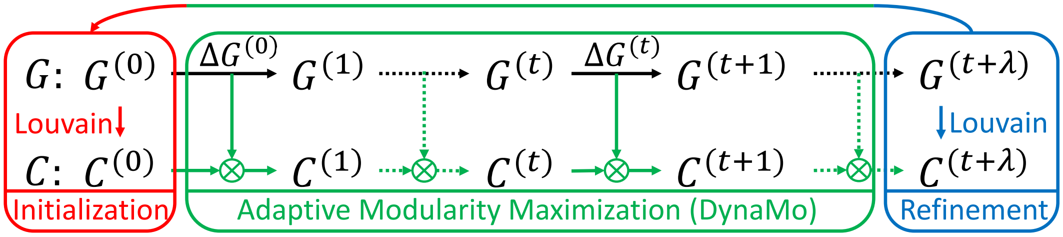


Figure 4.1: The overview of DynaMo.

Local Modularity Optimization, where each vertex moves from its own community to its neighbor's community to maximize the local modularity gain. If there is no positive modularity gain, keep the vertex in its original community. Repeat this step over all vertices multiple times until the modularity gain is negligible. (iii) Network Compression, where vertices belonging to the same community are aggregated as super vertices and a new network is built with the super vertices.

Louvain algorithm repeats the last two steps, until the modularity improvements is negligible. Although the actual computational complexity of Louvain depends on the input network, it has an average-case time complexity of $O(m)$ with most of the computational effort spending on the optimization of the first level network (i.e., before creating the super vertices).

4.4 Dynamic Community Detection

4.4.1 Problem Statement

Given a dynamic network $G = \{G^{(0)}, G^{(1)}, \ldots, G^{(t)}\}$, where $G^{(0)}$ is the initial network snapshot, let $C = \{C^{(0)}, C^{(1)}, \ldots, C^{(t)}\}$ denote the list of community structures of the corresponding network snapshots. As illustrated in Figure 4.1, we aim to design an adaptive and incremental algorithm to detect $C^{(t+1)}$, given $G^{(t)}$, $C^{(t)}$ and $\triangle G^{(t)}$.

4.4.2 Methodology Overview

As shown in Figure 4.1, our approach has three components:

4.4.2.1 Initialization

Use well-studied static algorithms (i.e., Louvain [12]) to compute $C^{(0)}$, which generates a comparatively accurate community structure of $G^{(0)}$.

4.4.2.2 Adaptive Modularity Maximization (DynaMo)

Given $G^{(t)}$, $C^{(t)}$ and $\triangle G^{(t)}$, update the community structure of $G^{(t+1)}$ from $C^{(t)}$ to $C^{(t+1)}$ while maximizing the modularity gain, using predesigned strategies that fully depend on $\triangle G^{(t)}$ and $C^{(t)}$. This is the core component of our framework that relies on fine-grained and theoretical-verified strategies (Section 4.4.3) to maximize the modularity gain while maintaining the efficiency.

4.4.2.3 Refinement

Once the obtained modularity of $C^{(t+\lambda)}$ is less than a predefined threshold, use $G^{(t+\lambda)}$ as the new initial network snapshot to restart our algorithm from the initialization step. This component prevents our frame from being trapped in the suboptimal solutions.

4.4.3 The DynaMo Algorithm

DynaMo is an adaptive and incremental algorithm aiming to maximize the community structure modularity gain based on the incremental changes of a dynamic network. We propose a two-step approach: (i) initialize an intermediate community structure, depending on the incremental network changes and the previous network community structure, and (ii) repeat the last two steps of Louvain algorithm (Section 4.3.4) on the intermediate community structure until the modularity gain is negligible.

Our algorithm benefits community detection in dynamic networks in three folds. First, in the initialization step, we categorize the incremental changes into 6 types. For each type of the incremental change, we design a strategy to initialize its corresponding intermediate community structure. Most of the strategies are theoretically verified to incrementally maximize the modularity, while avoiding redundant and repetitive computations. Second, compared with the original initialization step of Louvain algorithm, our initialization step takes advantage of the historical information, thus reduces most of the unnecessary computations happened at Louvain's first level network optimization, where Louvain spends most of its computational effort (Section 4.3.4). Hence, DynaMo would be much more efficient than Louvain algorithm while detecting communities in dynamic networks. Third, in the initialization, our algorithm could process a set of incremental changes as a batch, which makes the computational complexity of our algorithm less sensitive to

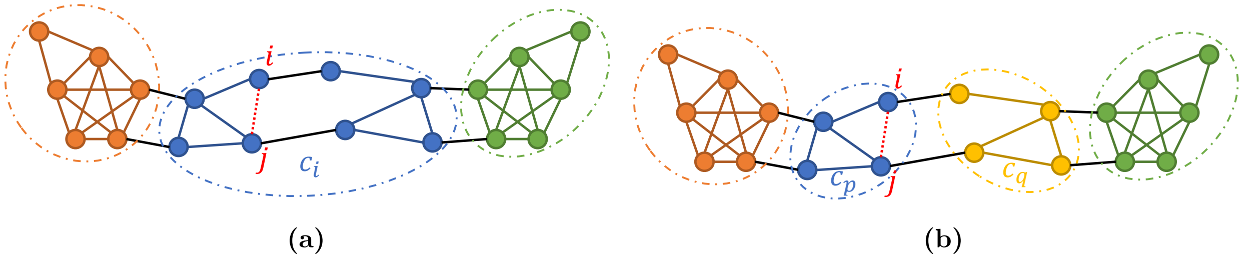


Figure 4.2: Change of communities by adding an intra-community edge. (a) unchanged, (b) splitting to smaller communities.

the amount of incremental changes and the frequency of network changes. So, DynaMo can detect communities while the network changing rapidly.

In this section, 6 different types of the incremental changes have been defined, where the initialization strategy of each type is also designed accordingly. Eight propositions are proposed and proved to provide the theoretical guarantees of our strategies towards maximizing the modularity.

4.4.3.1 Edge Addition/Weight Increase (EA/WI)

In this scenario, an edge (i, j, w_{ij}) between two existing vertices i and j has been changed to $(i, j, w_{ij} + \triangle w)$, where $w_{ij} \geq 0$ and $\triangle w > 0$. Edge addition is a special case of edge weight increase, where $w_{ij} = 0$. Depending on the edge property, we define two sub-scenarios:

The first scenario is called Intra-Community EA/WI (ICEA/WI), where ertices i and j belong to the same community (i.e., $c_i = c_j$). According to Proposition 1, ICEA/WI will never split i and j into different communities. And according to Remark 1, sometimes ICEA/WI will split c_i into multiple communities, while keeping i and j in the same community. Proposition 2 also provides us a convenient tool to decide when c_i should be bi-split into two smaller communities (i.e., c_p and c_q). However, this approach requires checking all the bi-split combinations of c_i, which is time consuming, especially when c_i is huge. In this case, we propose to initialize i and j as a two-vertices community, and all the other vertices in c_i as singleton communities.

Proposition 1. *Adding an edge or increasing the edge weight between vertices i and j, that belong to the same community ($c_i = c_j$), will not split i and j into different communities.*

Proof. Let $Q_1^{(t+1)}$ denote the new modularity value if the community structure keeps unchanged (i.e., $c_i = c_j$), and $Q_2^{(t+1)}$ denote the new modularity value if i and j are split into different communities (i.e., $c'_i \subseteq c_i$ and $c'_j = c_i \backslash c'_i$).

$$\begin{aligned} Q_1^{(t+1)} =& \frac{1}{2m+2\triangle w}\Big(\alpha_{c_i} + 2\triangle w - \frac{(\beta_{c_i}+2\triangle w)^2}{2m+2\triangle w} \\ &+ \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big) \end{aligned} \tag{4.2}$$

$$\begin{aligned} Q_2^{(t+1)} =& \frac{1}{2m+2\triangle w}\Big(\alpha_{c'_i} - \frac{(\beta_{c'_i}+\triangle w)^2}{2m+2\triangle w} + \alpha_{c'_j} \\ &- \frac{(\beta_{c'_j}+\triangle w)^2}{2m+2\triangle w} + \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big) \end{aligned} \tag{4.3}$$

where $\beta_{c_i} = \beta_{c'_i} + \beta_{c'_j}$.

Let $Q_1^{(t)}$ denote the modularity value of the "optimal" community structure of network snapshot $G^{(t)}$, and $Q_2^{(t)}$ denote its modularity value while i and j were split into different communities as in the calculation of $Q_2^{(t+1)}$.

$$Q_1^{(t)} = \frac{1}{2m}\Big(\alpha_{c_i} - \frac{\beta_{c_i}^2}{2m} + \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m}\big)\Big) \tag{4.4}$$

$$\begin{aligned} Q_2^{(t)} =& \frac{1}{2m}\Big(\alpha_{c'_i} - \frac{\beta_{c'_i}^2}{2m} + \alpha_{c'_j} - \frac{\beta_{c'_j}^2}{2m} \\ &+ \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m}\big)\Big) \end{aligned} \tag{4.5}$$

Since $c_i = c_j$ is the "optimal" community structure of $G^{(t)}$, we have:

$$\begin{aligned} & Q_1^{(t)} - Q_2^{(t)} \geq 0 \\ & \Leftrightarrow \frac{1}{2m}\Big(\alpha_{c_i} - \alpha_{c'_i} - \alpha_{c'_j} - \frac{\beta_{c'_i}\beta_{c'_j}}{m}\Big) \geq 0 \\ & \Leftrightarrow \alpha_{c_i} - \alpha_{c'_i} - \alpha_{c'_j} - \frac{\beta_{c'_i}\beta_{c'_j}}{m} \geq 0 \\ & \Rightarrow \alpha_{c_i} - \alpha_{c'_i} - \alpha_{c'_j} - \frac{\beta_{c'_i}\beta_{c'_j}}{m+\triangle w} \geq 0 \end{aligned} \tag{4.6}$$

By comparing $Q_1^{(t+1)}$ and $Q_2^{(t+1)}$, we get the difference of the modularity gain between the "unchanged" and "split" operations as follows:

$$\begin{aligned}
&\left(Q_1^{(t+1)} - Q_1^{(t)}\right) - \left(Q_2^{(t+1)} - Q_1^{(t)}\right) = Q_1^{(t+1)} - Q_2^{(t+1)} \\
&= \frac{1}{2m + 2\triangle w}\Big(\alpha_{c_i} - \alpha_{c'_i} - \alpha_{c'_j} - \frac{\beta_{c'_i}\beta_{c'_j}}{m + \triangle w}\Big) \\
&+ \frac{1}{2m + 2\triangle w}\Big(\frac{\triangle w(2m - \beta_{c'_i} - \beta_{c'_j}) + (\triangle w)^2}{m + \triangle w}\Big)
\end{aligned} \tag{4.7}$$

Since $\beta_{c'_i} + \beta_{c'_j} = \beta_{c_i} \leq 2m$, $\triangle w > 0$ and equation (4.6), we have $Q_1^{(t+1)} - Q_2^{(t+1)} > 0$ and the conclusion follows. □

Remark 1. *Although our Proposition 1 shows that ICEA/WI between i and j, where $c_i = c_j$, will not split them into different communities, sometimes splitting c_i into smaller communities in other ways (i.e., keeping i and j in the same community after the splitting) might maximize the modularity. For instance, as shown in Figure 4.2, assume all the edge weights are 1.0, and the red dash line between i and j is a newly added intra-community edge. Before adding the new edge, the modularity of community structure in Figure 4.2a (i.e.,* 0.561*) is higher than that in Figure 4.2b (i.e.,* 0.558*). However, after adding the new edge, the modularity of community structure in Figure 4.2a (unchanged, i.e.,* 0.564*) becomes lower than that in Figure 4.2b (split, i.e.,* 0.568*). In this case, although an intra-community edge has been added, splitting c_i into c_p and c_q provides higher modularity. Our algorithm carefully considers these "counterintuitive" cases, which is different from QCA [13, 94], thus, leading our algorithm to be more effective (Section* 4.5.5*).*

Proposition 2. *(ICEA/WI Community Bi-split) After ICEA/WI between vertices i and j, where $c_i = c_j$, if a bi-split of c_i (i.e., $c_p \subseteq c_i$ and $c_q = c_i \backslash c_p$) does not exist such that $\triangle w > \frac{m\alpha_1 - \beta_{c_p}\beta_{c_q}}{2\beta_{c_q} - \alpha_1}$, where $\alpha_1 = \alpha_{c_i} - \alpha_{c_p} - \alpha_{c_q}$, any other bi-split of c_i will not improve the modularity gain comparing with keeping the community structure unchanged.*

Proof. By Proposition 1, i and j should belong to the same community after ICEA/WI happened between i and j, where $c_i = c_j$. Without loss of generality, we assume i and j belong to community

c_p, even after certain bi-split operation. Therefore,

$$\begin{aligned}
&\triangle w > \frac{m\alpha_1 - \beta_{c_p}\beta_{c_q}}{2\beta_{c_q} - \alpha_1} \Leftrightarrow \\
&\Bigg(\frac{1}{2m+2\triangle w}\Big(\alpha_{c_i} + 2\triangle w - \frac{(\beta_{c_i}+2\triangle w)^2}{2m+2\triangle w} \\
&+ \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big)\Bigg) \\
&- \Bigg(\frac{1}{2m+2\triangle w}\Big(\alpha_{c_p} + 2\triangle w - \frac{(\beta_{c_p}+2\triangle w)^2}{2m+2\triangle w} + \alpha_{c_q} \\
&- \frac{\beta_{c_q}^2}{2m+2\triangle w} + \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big)\Bigg) < 0
\end{aligned} \tag{4.8}$$

where $\beta_{c_i} = \beta_{c_p} + \beta_{c_q}$, and $\{c_p, c_q\}$ is a bi-split of c_i. □

The second scenario is called Cross-Community EA/WI (CCEA/WI), where vertices i and j are from two different communities (i.e., $c_i \neq c_j$). CCEA/WI between i and j leads to three possible operations: (a) keeping the community structure unchanged; (b) merging c_i and c_j into one community; and (c) splitting $c_k = c_i \cup c_j$ into other smaller communities. According to Proposition 3, if $\triangle w$ is large enough, merging c_i and c_j into one community (e.g., c_k) provides higher modularity gain than keeping the community structure unchanged. However, if $\triangle w$ is too large (as shown in Proposition 4), CCEA/WI is equivalent to a two-step process: (a) CCEA/WI between i and j ($c_i \neq c_j$), that results in merging c_i and c_j into one community c_k (Proposition 3); (b) ICEA/WI between i and j ($c_i = c_j = c_k$). Proposition 4 provides a bi-split condition. However, Proposition 4 also requires checking all bi-split combinations of c_k. Hence, to deal with CCEA/WI, we propose: (a) if $\triangle w \leq \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \sqrt{(2m-\alpha_2-\beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big)$, where $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, we keep the community structure unchanged; (b) otherwise, we employ the same initialization approach proposed to deal with ICEA/WI on $c_k = c_i \cup c_j$, where we consider ICEA/WI has happened between vertices i and j, where $c_i = c_j = c_k$.

Proposition 3. *(CCEA/WI Community Merge) After CCEA/WI between i and j, where $c_i \neq c_j$, if and only if $\triangle w > \frac{1}{2}\big(\alpha_2+\beta_2-2m+\sqrt{(2m-\alpha_2-\beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big)$, where $\alpha_2 = \alpha_{c_i}+\alpha_{c_j}-\alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, merging c_i and c_j into c_k (i.e., $c_k = c_i \cup c_j$) has higher modularity gain than keeping the community structure unchanged.*

Proof. Let $Q_1^{(t+1)}$ denote the modularity value if the community structure keeps unchanged, and $Q_2^{(t+1)}$ denote the modularity value if c_i and c_j are merged into c_k (i.e., $c_k = c_i \cup c_j$). Then, we have:

$$\begin{aligned} Q_1^{(t+1)} = &\frac{1}{2m+2\triangle w}\Big(\alpha_{c_i} - \frac{(\beta_{c_i}+\triangle w)^2}{2m+2\triangle w} + \alpha_{c_j} \\ &- \frac{(\beta_{c_j}+\triangle w)^2}{2m+2\triangle w} + \sum_{c\in C}^{c\neq c_i,c_j} \big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big) \end{aligned} \tag{4.9}$$

$$\begin{aligned} Q_2^{(t+1)} = &\frac{1}{2m+2\triangle w}\Big(\alpha_{c_k} + 2\triangle w - \frac{(\beta_{c_k}+2\triangle w)^2}{2m+2\triangle w} \\ &+ \sum_{c\in C}^{c\neq c_i,c_j} \big(\alpha_c - \frac{\beta_c^2}{2m+2\triangle w}\big)\Big) \end{aligned} \tag{4.10}$$

where $\beta_{c_k} = \beta_{c_i} + \beta_{c_j}$. Therefore, we have the follows:

$$\begin{aligned} &\triangle w > \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \\ &\sqrt{(2m-\alpha_2-\beta_2)^2 + 4(m\alpha_1 + \beta_{c_i}\beta_{c_j})}\big) \\ &\Leftrightarrow \alpha_1 - 2\triangle w + \frac{(\beta_{c_i}+\triangle w)(\beta_{c_j}+\triangle w)}{m+\triangle w} < 0 \\ &\Leftrightarrow Q_1^{(t+1)} < Q_2^{(t+1)} \end{aligned} \tag{4.11}$$

Hence, the conclusion follows. □

Proposition 4. *(CCEA/WI Community Bi-split) After CCEA/WI between i and j, where $c_i \neq c_j$, $c_k = c_i \cup c_j$, and $\{c_p, c_q\}$ is another bi-split of c_k (i.e., $c_p \subseteq c_k$ and $c_q = c_k \backslash c_p$), if and only if $\triangle w > \frac{1}{2}\big(\alpha_2 + \beta_2 - 2m + \sqrt{(2m-\alpha_2-\beta_2)^2 + 4(m\alpha_2 + \beta_{c_i}\beta_{c_j})}\big) + \frac{m\alpha_1 - \beta_{c_p}\beta_{c_q}}{2\beta_{c_q}-\alpha_1}$, where $\alpha_1 = \alpha_{c_i} - \alpha_{c_p} - \alpha_{c_q}$, $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$ and $\beta_2 = \beta_{c_i} + \beta_{c_j}$, splitting c_k into c_p and c_q has higher modularity gain than either keeping the community structure unchanged or merging c_i and c_j into c_k.*

The proof could be easily derived from Proposition 2 and Proposition 3.

4.4.3.2 Edge Deletion/Weight Decrease (ED/WD)

In this scenario, an edge (i, j, w_{ij}) between two existing vertices i and j has been changed to $(i, j, w_{ij} - \triangle w)$, where $w_{ij} \geq \triangle w > 0$. Edge deletion is a special case of edge weight decrease, where $w_{ij} = \triangle w$. Depending on the edge property, we define two sub-scenarios:

The first scenario is called Intra-Community ED/WD (ICED/WD), where vertices i and j belong to the same community (i.e., $c_i = c_j$). According to Proposition 5, if i or j has one degree, decreasing the edge weight between i and j will keep the community structure unchanged. Also, intuitively, if i or j has one degree, deleting the edge between i and j will result in the same community structure plus one or two singleton communities (i.e., the vertex of one degree becomes singleton community). Except for the case above (i.e., i or j has one degree), ICED/WD between i and j leads to three other possible operations: (a) keeping the community structure unchanged, if c_i is still densely connected; (b) splitting c_i into multiple smaller communities, if c_i becomes sparsely connected; and (c) merging c_i with some of its neighbor communities (i.e., the opposite situation of Remark 1). Since the analytical approach is complex and time consuming, we propose to initiate all vertices within the communities, that adjacent to i or j (including c_i), as singleton communities.

Proposition 5. *For any pair of vertices i, j that belong to the same community (i.e., $c_i = c_j$), if i or j has only one neighbor vertex (j or i), decreasing the edge weight between i and j, does not split i and j into different communities.*

Proof. Suppose the edge weight between vertices i and j has been decreased, where $c_i = c_j$. Let $Q_1^{(t+1)}$ be the modularity value if the community structure keeps unchanged, and $Q_2^{(t+1)}$ be the (best case) modularity value if i and j are split into smaller communities (i.e., $c_i' \subseteq c_i$ and $c_j' = c_i \backslash c_i'$).

$$\begin{aligned} Q_1^{(t+1)} = & \frac{1}{2m - 2\triangle w}\Big(\alpha_{c_i} - 2\triangle w - \frac{(\beta_{c_i} - 2\triangle w)^2}{2m - 2\triangle w} \\ & + \sum_{c \in C}^{c \neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m - 2\triangle w}\big)\Big) \end{aligned} \tag{4.12}$$

$$\begin{aligned} Q_2^{(t+1)} = \frac{1}{2m-2\triangle w}\Big(&\alpha_{c'_i} - \frac{(\beta_{c'_i}-\triangle w)^2}{2m-2\triangle w} + \alpha_{c'_j} \\ &- \frac{(\beta_{c'_j}-\triangle w)^2}{2m-2\triangle w} + \sum_{c\in C}^{c\neq c_i}\big(\alpha_c - \frac{\beta_c^2}{2m-2\triangle w}\big)\Big) \end{aligned} \tag{4.13}$$

where $\beta_{c_i} = \beta_{c'_i} + \beta_{c'_j}$.

$$\begin{aligned} &Q_1^{(t+1)} - Q_2^{(t+1)} \\ &= \frac{1}{2m-2\triangle w}\Big(\alpha_{c_i} - 2\triangle w - \alpha_{c'_i} - \alpha_{c'_j} \\ &- \frac{(\beta_{c'_i}-\triangle w)(\beta_{c'_j}-\triangle w)}{m-\triangle w}\Big) \\ &= \frac{(w'_{ij}-\triangle w)\Big((2m-\alpha_{c_i}) + (w'_{ij}-\triangle w)\Big) - \alpha_{c'_i}\alpha_{c'_j}}{2(m-\triangle w)^2} \end{aligned} \tag{4.14}$$

where $w'_{ij} = \frac{\alpha_{c_i}-\alpha_{c'_i}-\alpha_{c'_j}}{2}$, $\beta_{c'_i} = \alpha_{c'_i} + w'_{ij}$, $\beta_{c'_j} = \alpha_{c'_j} + w'_{ij}$.

If i or j has only one neighbor vertex (j or i), then,

$$\begin{aligned} &Q_1^{(t+1)} - Q_2^{(t+1)} \\ &= \frac{(w'_{ij}-\triangle w)\Big((2m-\alpha_{c_i}) + (w'_{ij}-\triangle w)\Big)}{2(m-\triangle w)^2} > 0 \end{aligned} \tag{4.15}$$

where $w'_{ij} > \triangle w$, $2m > \alpha_{c_i}$. The conclusion follows. □

The second scenario is called Cross-Community ED/WD (CCED/WD), where vertices i and j are from two different communities (i.e., $c_i \neq c_j$). By Proposition 6, CCED/WD strengthens the community structure, thus, keeping the community structure unchanged.

Proposition 6. *If vertices i and j are from different communities ($c_i \neq c_j$), deleting an edge or decreasing the edge weight between i and j, will increase the modularity gain coming from c_i and c_j.*

Proof. Let $Q_i^{(t+1)}$ and $Q_i^{(t)}$ be the modularities of c_i before and after the CCED/WD scenario. Then, we have:

$$\begin{aligned}\triangle Q = Q_i^{(t+1)} + Q_j^{(t+1)} - Q_i^{(t)} - Q_j^{(t)} &= \frac{\triangle w(\alpha_{c_i} + \alpha_{c_j})}{2m(m + \triangle w)} \\ + \frac{1}{4}\big(\frac{\beta_{c_i}}{m} - \frac{\beta_{c_i} - \triangle w}{m - \triangle w}\big)\big(\frac{\beta_{c_i}}{m} + \frac{\beta_{c_i} - \triangle w}{m - \triangle w}\big) \\ + \frac{1}{4}\big(\frac{\beta_{c_j}}{m} - \frac{\beta_{c_j} - \triangle w}{m - \triangle w}\big)\big(\frac{\beta_{c_j}}{m} + \frac{\beta_{c_j} - \triangle w}{m - \triangle w}\big)\end{aligned} \tag{4.16}$$

Let $k = \min\{\big(\frac{\beta_{c_i}}{m} + \frac{\beta_{c_i} - \triangle w}{m - \triangle w}\big), \big(\frac{\beta_{c_j}}{m} + \frac{\beta_{c_j} - \triangle w}{m - \triangle w}\big)\}$. Thus,

$$\begin{aligned}\triangle Q &> \frac{k}{4}\big(\frac{\beta_{c_i} + \beta_{c_j}}{m} - \frac{\beta_{c_i} + \beta_{c_j} - 2\triangle w}{m - \triangle w}\big) \\ &= \frac{k\triangle w(2m - \beta_{c_i} - \beta_{c_j})}{4m(m - \triangle w)} > 0\end{aligned} \tag{4.17}$$

where $2m > \beta_{c_i} + \beta_{c_j}$, $m > \triangle w$. The conclusion follows. □

4.4.3.3 Vertex Addition (VA)

In this scenario, a new vertex i and its associated edges are added. On one hand, if i has no associated edge, we make it as a singleton community and keep the rest community structure unchanged. On the other hand, if i has one or more associated edges, some interesting cases would happen. For instance, if all of i's associated edges are connected to the same community, i.e., c_j, by Proposition 7, we should merge i into c_j and treat all of i's associated edges as ICEA/WI. A more complicated case occurs when i's associated edges are connected to different communities. In this case, by Proposition 8, we could merge i into community c_j that has the highest $\triangle w_{ij}$. However, other than simply determining which community i should merge into, we should also consider which set of vertices could together with i to form a new community, or which community could be split into smaller communities, to further maximize the modularity. To cope with all the cases, where i has one or more associated edges, we propose to initialize i and j as a two-vertices community, where edge e_{ij} has the highest weight among all of i's associated edges (randomly selecting a vertex j if there are ties), and initialize all the other vertices within i's adjacent communities as singleton communities.

Proposition 7. *If a new vertex i has been added and all of its associated edges are connected to the same community, i.e., c_j, merging i into c_j has higher modularity gain than keeping i as a singleton community.*

Proof. Let $Q_1^{(t+1)}$ denote the modularity value while merging i into community c_j, $Q_2^{(t+1)}$ denote the modularity value while keeping i as a singleton community, and $\triangle w > 0$ denote the sum of the weights of all of i's associated edges. Then, we have:

$$\begin{aligned} Q_1^{(t+1)} &= \frac{1}{2m + 2\triangle w}\Big(\alpha_{c_j} + 2\triangle w - \frac{(\beta_{c_j} + 2\triangle w)^2}{2m + 2\triangle w} \\ &\quad + \sum_{c \in C}^{c \neq c_j}\big(\alpha_c - \frac{\beta_c^2}{2m + 2\triangle w}\big)\Big) \end{aligned} \tag{4.18}$$

$$\begin{aligned} Q_2^{(t+1)} &= \frac{1}{2m + 2\triangle w}\Big(\alpha_{c_j} - \frac{(\beta_{c_j} + \triangle w)^2}{2m + 2\triangle w} \\ &\quad - \frac{(\triangle w)^2}{2m + 2\triangle w} + \sum_{c \in C}^{c \neq c_j}\big(\alpha_c - \frac{\beta_c^2}{2m + 2\triangle w}\big)\Big) \end{aligned} \tag{4.19}$$

$$Q_1^{(t+1)} - Q_2^{(t+1)} = \frac{\triangle w(2m - \beta_{c_j}) + (\triangle w)^2}{2(m + \triangle w)^2} > 0 \tag{4.20}$$

where $2m \geq \beta_{c_j}$. Hence, the conclusion follows. □

Proposition 8. *Suppose a new vertex i has been added and its associated edges are connected to different communities. Let $\triangle w_{ij}$ denote the sum of the edge weights of vertex i's associated edges that are connected to community c_j. Given two communities c_p and c_q, if $\triangle w_{ip} > \triangle w_{iq}$, merging i into c_p has more modularity gain than merging i into c_q.*

Proof. Let $Q_1^{(t+1)}$ and $Q_2^{(t+1)}$ denote the modularity values while merging i into community c_p and community c_q, respectively. Suppose $\triangle w_{ip} > \triangle w_{iq}$, and $\triangle w$ denoting the sum of the weights of all of i's associated edges. Then, we have:

$$\begin{aligned}
Q_1^{(t+1)} - Q_2^{(t+1)} &= \frac{1}{2m + 2\triangle w}\Big(\alpha_{c_p} + 2\triangle w_{ip} \\
&- \frac{(\beta_{c_p} + 2\triangle w_{ip})^2}{2m + 2\triangle w} + \alpha_{c_q} - \frac{(\beta_{c_q} + \triangle w_{iq})^2}{2m + 2\triangle w}\Big) \\
&- \frac{1}{2m + 2\triangle w}\Big(\alpha_{c_p} - \frac{(\beta_{c_p} + \triangle w_{ip})^2}{2m + 2\triangle w} + \alpha_{c_q} \\
&+ 2\triangle w_{iq} - \frac{(\beta_{c_q} + 2\triangle w_{iq})^2}{2m + 2\triangle w}\Big) \\
&= \frac{(4m - 2\beta_{c_p})\triangle w_{ip} - (4m - 2\beta_{c_q})\triangle w_{iq}}{(2m + 2\triangle w)^2} \\
&+ \frac{(4\triangle w - 3\triangle w_{ip})\triangle w_{ip} - (4\triangle w - 3\triangle w_{iq})\triangle w_{iq}}{(2m + 2\triangle w)^2} \\
&\geq \frac{(k_1 + k_2)(\triangle w_{ip} - \triangle w_{iq})}{(2m + 2\triangle w)^2}
\end{aligned} \tag{4.21}$$

where $k_1 = \min\{(4m - 2\beta_{c_p}), (4m - 2\beta_{c_q})\}$ and $k_2 = \min\{(4\triangle w - 3\triangle w_{ip}), (4\triangle w - 3\triangle w_{iq})\}$.

Since $2m > \beta_{c_p}$, $2m > \beta_{c_q}$, $\triangle w > \triangle w_{ip}$, $\triangle w > \triangle w_{iq}$ and $\triangle w_{ip} > \triangle w_{iq}$, we have $k_1 > 0$, $k_2 > 0$, and $Q_1^{(t+1)} - Q_2^{(t+1)} > 0$. Hence, the conclusion follows. □

4.4.3.4 Vertex Deletion (VD)

In this scenario, an old vertex i and its associated edges are deleted. On one hand, if i has no associated edge, deleting i has no influence on the rest of the network, and hence, we should keep the community structure unchanged. On the other hand, if i has too many associated edges, deleting i might cause its community and its neighbor communities being broken into smaller communities and potentially being merged into other communities. To handle this case, we propose to initialize all the vertices within c_i and i's neighbor communities as singleton communities.

4.4.4 Implementation and Analysis

4.4.4.1 Implementation

Algorithm 7 presents the DynaMo Initialization, where we implement the operation of each type of incremental network change to initialize the intermediate community structure towards maximizing the modularity. The input contains the current network $G^{(t+1)}$, the previous network $G^{(t)}$ and the previous community structure $C^{(t)}$. The output contains two set of communities, $\triangle C_1$

Algorithm 7: DynaMo Initialization (Init)

```
Input: $V^{(t+1)}$, $E^{(t+1)}$, $V^{(t)}$, $E^{(t)}$, $C^{(t)}$.
Output: $\triangle C_1$, $\triangle C_2$.
$\triangle E \leftarrow A\ set\ of\ edges\ changed\ from\ E^{(t)}\ to\ E^{(t+1)}$;
$\triangle V_{add} \leftarrow V^{(t+1)} \backslash V^{(t)}$; $\triangle V_{del} \leftarrow V^{(t)} \backslash V^{(t+1)}$;
$\triangle C_1 \leftarrow \emptyset$; $\triangle C_2 \leftarrow \emptyset$;
for $e_{ij} \in \triangle E$ do
    for $k \in \{i, j\}$ do
        if $k \in \triangle V_{del}$ then
            $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_k\}$;
            for $e_{kl} \in E^{(t)}$ do
                $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
        if $k \in \triangle V_{add}$ then
            $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_k\}$;
            $w_{max} = 0$; $c \leftarrow \emptyset$;
            for $e_{kl} \in E^{(t+1)}$ do
                $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
                if $w_{kl} > w_{max}$ then
                    $w_{max} = w_{kl}$; $c \leftarrow \{k, l\}$;
            $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
    if $i, j \notin \triangle V_{del} \cup \triangle V_{add}$ then
        if $e_{ij} \notin E^{(t+1)}$ or $w_{ij}^{t} > w_{ij}^{t+1}$ then
            if $c_i = c_j$ then
                $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i\}$;
                for $k \in \{i, j\}$ do
                    for $e_{kl} \in E^{(t)}$ do
                        $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_l\}$;
        if $e_{ij} \notin E^{(t)}$ or $w_{ij}^{t} < w_{ij}^{t+1}$ then
            if $c_i = c_j$ then
                $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i\}$; $c \leftarrow \{i, j\}$;
                $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
            else
                $\triangle w = w_{ij}^{t+1} - w_{ij}^{t}$; $c_k = c_i \cup c_j$;
                $\alpha_2 = \alpha_{c_i} + \alpha_{c_j} - \alpha_{c_k}$; $\beta_2 = \beta_{c_i} + \beta_{c_j}$;
                $\delta_1 = 2m - \alpha_2 - \beta_2$; $\delta_2 = m\alpha_2 + \beta_{c_i}\beta_{c_j}$;
                if $2\triangle w + \delta_1 > \sqrt{\delta_1^2 + 4\delta_2}$ then
                    $\triangle C_1 \leftarrow \triangle C_1 \cup \{c_i, c_j\}$; $c \leftarrow \{i, j\}$;
                    $\triangle C_2 \leftarrow \triangle C_2 \cup \{c\}$;
return $\triangle C_1$, $\triangle C_2$.
```

and $\triangle C_2$, that will be modified to initialize the intermediate community structure at the beginning of the second phase. $\triangle C_1$ contains a set of communities in $C^{(t)}$ to be separated into singleton communities, and $\triangle C_2$ contains a set of two-vertices communities to be created. Algorithm 8 presents the second phase, where the last two steps of Louvain algorithm is applied on the initialized intermediate community structure of $G^{(t+1)}$.

Algorithm 8: DynaMo

```
Input: $G^{(t+1)}$, $G^{(t)}$, $C^{(t)}$.
Output: $C^{(t+1)}$.
$\triangle C_1$, $\triangle C_2 \leftarrow$ Init($V^{(t+1)}$, $E^{(t+1)}$, $V^{(t)}$, $E^{(t)}$, $C^{(t)}$);
$C^{(t+1)} \leftarrow C^{(t)}$;
for $c_i \in \triangle C_1$ do
    $C^{(t+1)} \leftarrow C^{(t+1)} \backslash \{c_i\}$;
    for $k \in c_i$ do
        Create singleton community: $c_k \leftarrow \{k\}$;
        $C^{(t+1)} \leftarrow C^{(t+1)} \cup \{c_k\}$;
for $c = \{i, j\} \in \triangle C_2$ do
    Create two-vertices community: $c_k \leftarrow \{i, j\}$;
    $C^{(t+1)} \leftarrow (C^{(t+1)} \backslash \{c_i, c_j\}) \cup \{c_k\}$;
$C^{(t+1)} \leftarrow$ Louvain($C^{(t+1)}$, $G^{(t+1)}$);
return $C^{(t+1)}$.
```

Most of the operations in Algorithm 7 are theoretically guaranteed by our propositions described in Section 4.4.3 to maximize the modularity, while some of the operations are heuristically designed for the sake of the efficiency. For instance, according to Proposition 1, Remark 1 and Proposition 2, given ICEA/WI between vertices i and j, we initialize i and j as a two-vertices community to incrementally maximize the modularity, and initialize all the other vertices in c_i as singleton communities to take all the influenced vertices into consideration carefully while maintaining the algorithm efficiency (lines 26-28). According to Proposition 3 and Proposition 4, we use a designed threshold condition (lines 30-33) to determine the operation of given CCEA/WI. If the condition is true, we use the same operation of ICEA/WI to tackle CCEA/WI (lines 33-35). Otherwise, we keep the community structure unchanged to incrementally maximize the modularity. According to Proposition 5 and the analysis in Section 4.4.3.2, given ICED/WD, we initialize all the potentially influenced vertices as singleton communities to maintain a trade-off between the effectiveness and efficiency (lines 18-24). According to Proposition 6, given CCED/WD, we keep the community structure unchanged to maximize the local modularity gain. According to Proposition 7 and Proposition 8, given new vertex i and its associated edges, we initialize i and its most closely connected neighbor vertex as a two-vertices community (lines 12, 15-17), and initialize all the potentially influenced vertices as singleton communities (lines 10-16). After deleting vertex i, we heuristically initialize all the vertices within c_i and i's neighbor communities as singleton communities (lines 6-9). To summarize, initializing $\triangle C_2$ aims to incrementally maximize the modularity with certain theoretical guarantees, and initializing $\triangle C_1$ aims to heuristically maximize the modularity (by Algorithm 8) while maintaining the algorithm efficiency.

4.4.4.2 Time Complexity Analysis

The computation of our algorithm tackling one network snapshot comes from two parts: (a) the initialization, and (b) the last two steps of Louvain algorithm. In the initialization, different network changes trigger different operations, thus resulting in different computation time. For instance, if one network change is ICEA/WI (i.e., e_{ij}, $c_i = c_j$), our algorithm (line 26-28) will add c_i into $\triangle C_1$, and add $c = \{i, j\}$ into $\triangle C_2$. The time complexity of both operations are $O(1)$, thus, the time complexity to deal with single change of ICEA/WI is $O(1)$. Similarly, the time complexities to deal with single change of CCEA/WI (line 29-35) and CCED/WD (no operation needed) are also $O(1)$. To deal with single change of ICED/WD, VA or VD, our algorithm runs through the set of neighbor vertices of the changed edge, and thus, result in $O(\frac{|E|}{|V|})$ time complexity. Furthermore, as shown in Algorithm 7, each network snapshot usually has multiple network changes. Since the number of network changes is proportional to $\triangle E$, the overall time complexity of the initialization is $O(|\triangle E|)$ or $O(|\triangle E| \cdot \frac{|E|}{|V|})$.

The time complexity of the original Louvain algorithm is $O(|E|)$. However, compared with the Louvain algorithm initialization, our algorithm considers the historical information and designs an initialization phase to reduce the number of edges left for the second phase analysis as much as possible. Thus, the time complexity of the second phase of our algorithm is $O(|E|^*)$, where $|E|^* \ll |E|$. Hence, the overall best case time complexity of our algorithm is $O(|\triangle E| + |E|^*)$, and the worst case is $O(|\triangle E| \cdot \frac{|E|}{|V|} + |E|^*)$.

4.5 Experimental Evaluation

4.5.1 Experiment Environment

All the experiments were conducted on a PC with an Intel Xeon Gold 6148 Processor, 128GB RAM, running 64-bit Ubuntu 18.04 LTS operating system. All the algorithms and experiments are implemented using Java with JDK 8.

Table 4.1: Description of the real-world dynamic networks. [Notations: $|V|$ ($|E|$): # of unique vertices (edges); $\mathbb{E}[|\triangle V|]$ ($\mathbb{E}[|\triangle E|]$): avg. # of vertices (edges) changed per network snapshots; # of snapshots: total number of consecutive network snapshots; time-interval: period of time between two consecutive network snapshots; time-span: total time spanning of each network dataset].

networks	$\mathbf{|V|}$	$\mathbb{E}[|\triangle \mathbf{V}|]$	**vertex-type**	$\mathbf{|E|}$	$\mathbb{E}[|\triangle \mathbf{E}|]$	**edge-type**	**# of snapshots**	**time-interval**	**time-span**
Cit-HepPh	30,501	6,460	author	346,742	11,127	co-citation	31	4 months	124 months
Cit-HepTh	7,577	1,253	author	51,089	2,042	co-citation	25	5 months	125 months
DBLP	1,411,321	122,731	author	5,928,285	191,233	co-authorship	31	2 years	62 years
Facebook	59,302	12,765	user	592,406	20,943	friendship	28	1 month	28 months
Flickr	780,079	93,253	user	4,407,259	168,977	follow	24	3 days	72 days
YouTube	3,160,656	91,954	user	7,211,498	175,303	subscription	33	5 days	165 days

Table 4.2: Comparison of the time complexities of the competing algorithms. [Notations: $n = |V|$ ($m = |E|$): # of unique vertices (edges); $\upsilon = |\triangle V|$ ($\epsilon = |\triangle E|$): # of vertices (edges) changed; m_b^* (m_d^*): # of unique vertices (edges) after the initialization phase of Batch (DynaMo), and $m_b^* \ll m$ ($m_d^* \ll m$); T_{LR} (T_{SVM}): the time complexity of using logistic regression (support vector machine) in LBTR].

algorithms	**the best case**	**the worst case**
Louvain [12]	$O(m)$	$O(m)$
Batch [14]	$O((\upsilon + \epsilon) \cdot \frac{m}{n} + m_b^*)$	$O((\upsilon + \epsilon) \cdot \frac{m}{n} + m_b^*)$
DynaMo	$O(\epsilon + m_d^*)$	$O(\epsilon \cdot \frac{m}{n} + m_d^*)$
QCA [13]	$O(\epsilon)$	$O(\epsilon \cdot m)$
GreMod [15]	$O(\epsilon)$	$O(\epsilon \cdot n)$
LBTR-LR [16]	$O(\upsilon \cdot T_{LR})$	$O(\upsilon \cdot T_{LR})$
LBTR-SVM [16]	$O(\upsilon \cdot T_{SVM})$	$O(\upsilon \cdot T_{SVM})$

4.5.2 Baseline Approaches

We compare DynaMo with Louvain (Section 4.3.4), and 5 dynamic algorithms: (i) Batch [14]: a batch-based incremental modularity optimization algorithm; (ii) GreMod [15]: a rule-based incremental algorithm that performs predetermined operations on edge additions; (iii) QCA [13]: a rule-based incremental algorithm that updates the community structures according to predefined rules of vertex/edge additions/deletions; (iv) LBTR [16]: a learning-based algorithm that uses classifiers to update community assignments. We use Support Vector Machine (SVM) and Logistic Regression (LR) as the classifiers, namely LBTR-SVM and LBTR-LR.

4.5.3 Experiment Datasets

We conduct our experiments on two categories of networks: real-world networks (ground-truth is unknown), and synthetic networks (ground-truth is known).

4.5.3.1 Real-world Dynamic Networks

As shown in Table 4.1, six real-world networks are used in our experiments. (i) Cit-HepPh (Cit-HepTh) [95] contains the citation network of high-energy physics phenomenology (theory) papers from 1993 to 2003. (ii) DBLP [96] contains a co-authorship network of computer science papers ranging from 1954 to 2015, where each author is represented as a vertex and co-authors are linked by an edge. (iii) Facebook [97] contains the user friendship establishment information from about 52% of Facebook users in New Orleans area, spanning from September 26th, 2006 to January 22nd, 2009. In this network, each vertex represents a Facebook user, and each edge represents an user-to-user friendship establishment link that contains a timestamp representing the time of friendship establishment. (iv) Flickr [98] was obtained on January 9th, 2007, and contains over 1.8 million users and 22 million links, and each link has a timestamp that represents the time of the following link establishment. We select a sub-network, where all the user-to-user following links were established from March 6th, 2007 to May 15th, 2007. (v) YouTube [99] was obtained on January 15th, 2007 and consists of over 1.1 million users and 4.9 million links, and each link has a timestamp that represents the time of the subscribing link establishment. We select a sub-network, where all the user-to-user subscribing links were established from February 2nd, 2007 to July 23rd, 2007.

4.5.3.2 Synthetic Dynamic Networks

We use RDyn [100], a benchmark model focusing on community changes in dynamic networks, to generate synthetic networks and their ground-truth communities. It allows us to specify different parameters, such as the number of vertices (N), the number of time points (T), the maximum number of community change events (e.g., splitting or merging) per time point (M), etc.. We use various combinations of N, T and M to generate synthetic networks, where $N \in \{200, 400, 600, 800, 1000\}$, $T \in \{25, 50, 75, 100, 125\}$, $M \in \{1, 2, 3, 4\}$ and all the other parameters set by default values. For each parameter combination (out of 100 combinations in total), we randomly generate 100 synthetic networks, resulting in 10,000 synthetic networks in total.

4.5.4 Experimental Procedure

For each real-world network, we apply Louvain algorithm on its initial snapshot to obtain its initial community structure (Section 4.4.2). For each synthetic network, we use the ground-truth communities of its initial snapshot as its initial community structure. For the rest of snapshots of real-world and synthetic networks, the dynamic algorithms only use the initial community structure and the network changes between two consecutive snapshots to update the new community structures, while the static algorithm will be applied on the whole network of each snapshot. All experiments are performed for 200 times to obtain the average results.

4.5.5 Effectiveness Analysis

4.5.5.1 Effectiveness Metrics

We evaluate the effectiveness of the community detection algorithms using three metrics: modularity, Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI). Modularity (Section 4.3.3) is designed to measure the strength of dividing a network into communities, and does not require the ground-truth information. Hence, we use modularity to evaluate the results of the real-world networks. NMI and ARI are designed to measure the similarities between the community structure obtained from the experiments and that of the ground-truth, which are used to evaluate the results of the synthetic networks.

Let C_t denote the ground-truth community division, and C_r denote the experiment result. NMI is defined as follows:

$$NMI(C_t, C_r) = \frac{2 \times I(C_t; C_r)}{[H(C_t) + H(C_r)]} \tag{4.22}$$

where $H(C_r)$ is the entropy of C_r, and $I(C_t; C_r)$ is the mutual information between C_t and C_r. NMI ranges from 0 to 1. NMI closing to 1 indicates C_r is similar to C_t, while closing to 0 means C_r is random compared with C_t.

Let a be the number of pairs of vertices in the same community in both C_t and C_r, b be the number of pairs of vertices in the same community in C_t and in different communities in C_r, c be the number of pairs of vertices in different communities in C_t and in the same community in C_r, d be the number of pairs of vertices in different communities in both C_t and C_r. ARI is defined as follows:

$$ARI(C_t, C_r) = \frac{2(ad - bc)}{b^2 + c^2 + 2ad + (a+d)(b+c)} \tag{4.23}$$

where its upper bound is 1, and the higher, the better.

4.5.5.2 Experimental Results

Figure 4.3 shows the modularity results of 7 algorithms running on 6 real-world networks, respectively. We observe that DynaMo consistently outperforms all the other dynamic algorithms in terms of modularity. Compared with the runner-up algorithm (Batch), DynaMo obtains 2.6%, 2.2%, 4.3%, 2.1%, 1.1% and 2.2% higher modularity averaged over all the time points, and 3.2%, 4.4%, 17.3%, 2.4%, 1.2% and 4.7% higher modularity on the last time point of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively. Compared with Louvain, DynaMo achieves nearly identical performance, with only 0.49%, 0.38%, 0.06%, 0.7%, 0.5% and 0.5% lower modularity averaged over all the time points, and only 0.52%, 0.74%, 0.27%, 0.46%, 0.5% and 1.7%

lower modularity on the last time point of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively.

Figure 4.4 shows the NMI results (mean and standard deviation) of 6 dynamic algorithms running on 10,000 synthetic networks. We observe that DynaMo obtains the highest NMI value among all the dynamic algorithms regardless of any RDyn parameters. DynaMo outperforms the runner-up algorithm (QCA) by 69.1%, 66.4% and 70.3% on average with the increase of the number of vertices, the maximum number of events per time point, and the number of time points, respectively, which is statistically significant according to the two-sample t-test with 95% confidence interval. The NMI standard deviation of DynaMo is also lower than that of QCA, demonstrating the consistency of DynaMo in detecting communities of various dynamic networks. Furthermore, as the maximum number of events per time point and the number of time points increase, DynaMo has the minimum NMI value loss among all the dynamic algorithms, indicating DynaMo is more robust and consistent while detecting communities of dynamic networks that last longer and have more events per time point.

Figure 4.5 shows the ARI results (mean and standard deviation) of 6 dynamic algorithms running on 10,000 synthetic networks, which share similar patterns as the NMI results. DynaMo outperforms the runner-up algorithm (QCA) by 211.1%, 224.5% and 257.6% on average with the increase of the number of vertices, the maximum number of events per time point, and the number of time points, respectively, which is statistically significant according to the two-sample t-test with 99% confidence interval. As the number of vertices increases, the ARI standard deviation of DynaMo dramatically decreases, while as the maximum number of events per time point and the number of time points increase, the standard deviation of DynaMo slightly increases. However, even considering the standard deviation difference, DynaMo still significantly outperforms all the other dynamic algorithms.

4.5.6 Efficiency Analysis

4.5.6.1 Time Complexity Analysis

Table 4.2 shows the theoretical time complexities of all the competing algorithms. DynaMo, QCA and GreMod have different time complexities while running in different scenarios (i.e., best/worst case). As discussed in Section 4.4.4.2, DynaMo has the best case time complexity when the network changes are ICEA/WI, CCEA/WI or CCED/WD, and otherwise, has the worst case time complexity. Similarly, QCA and GreMod have the best case time complexity if the network changes are ICEA or CCED, and otherwise, have the worst case time complexity. For the other algorithms, the time complexities of the best and the worst cases are identical. Below show the details about our analysis.

• Compared with Louvain [12], DynaMo has less time complexity, when the impact of the network changes of a given network snapshot on its community structure updating is small enough to ensure $m_d^* \ll m$. First, the evolutionary nature of the real-world dynamic networks assumes two consecutive network snapshots of the same network should have similar community structures. Therefore, each snapshot of a dynamic network should only result in a small part of its community structure being updated (i.e., $m_d^* \ll m$). Also, from our empirical studies, the assumption of $m_d^* \ll m$ always holds. Hence, DynaMo should be more efficient than Louvain for most of the time.

• Compared with Batch [14], DynaMo has less initialization time complexity (i.e., $O(\epsilon \cdot \frac{m}{n}) < O((v + \epsilon) \cdot \frac{m}{n})$), and different second phase time complexities (i.e., m_d^* vs. m_b^*).

• Compared with QCA [13] and GreMod [15], who update the community structure according to certain predefined rule of each network change and one at a time (i.e., not in a batch fashion), DynaMo is more efficient if each network snapshot has more network changes, since DynaMo is capable of handling a batch of network changes.

• Compared with LBTR [16], who uses machine learning models to decide if a vertex needs to revise its community, DynaMo is more consistent and practical when dealing with different real-world networks. Since the characteristics of an dynamic network keep changing over time, LBTR has to keep updating the machine learning models to adapt the new characteristics. In such case, we have to take the training time into account. Also, the time complexity of LBTR highly depends on the machine learning algorithm used for the classification problem (e.g., $O(T_{SVM}) > O(T_{LR})$).

4.5.6.2 Empirical Result Studies

Since the theoretical time complexities always depend on the ideal scenarios or extreme cases, it is necessary to conduct empirical studies using real-world networks. To ensure the comparison is as unbiased as possible, all the algorithms are implemented using Java and running on the same environment. Figure 4.6 shows the cumulative elapsed time results, and below show the details about our observations.

• Compared with Louvain [12], DynaMo obtains over 2x, 2x, 4x, 3x, 4x and 3x speed up on the series of network snapshots of Cit-HepPh, Cit-HepTh, DBLP, Facebook, Flickr and YouTube, respectively.

• Compared with Batch [14], DynaMo obtains over 3x, 5x, 2x, 7x and 5x speed up on the series of network snapshots of Cit-HepPh, Cit-HepTh, DBLP, Facebook and Flickr, respectively. DynaMo spends nearly the same amount of time as Batch on YouTube network.

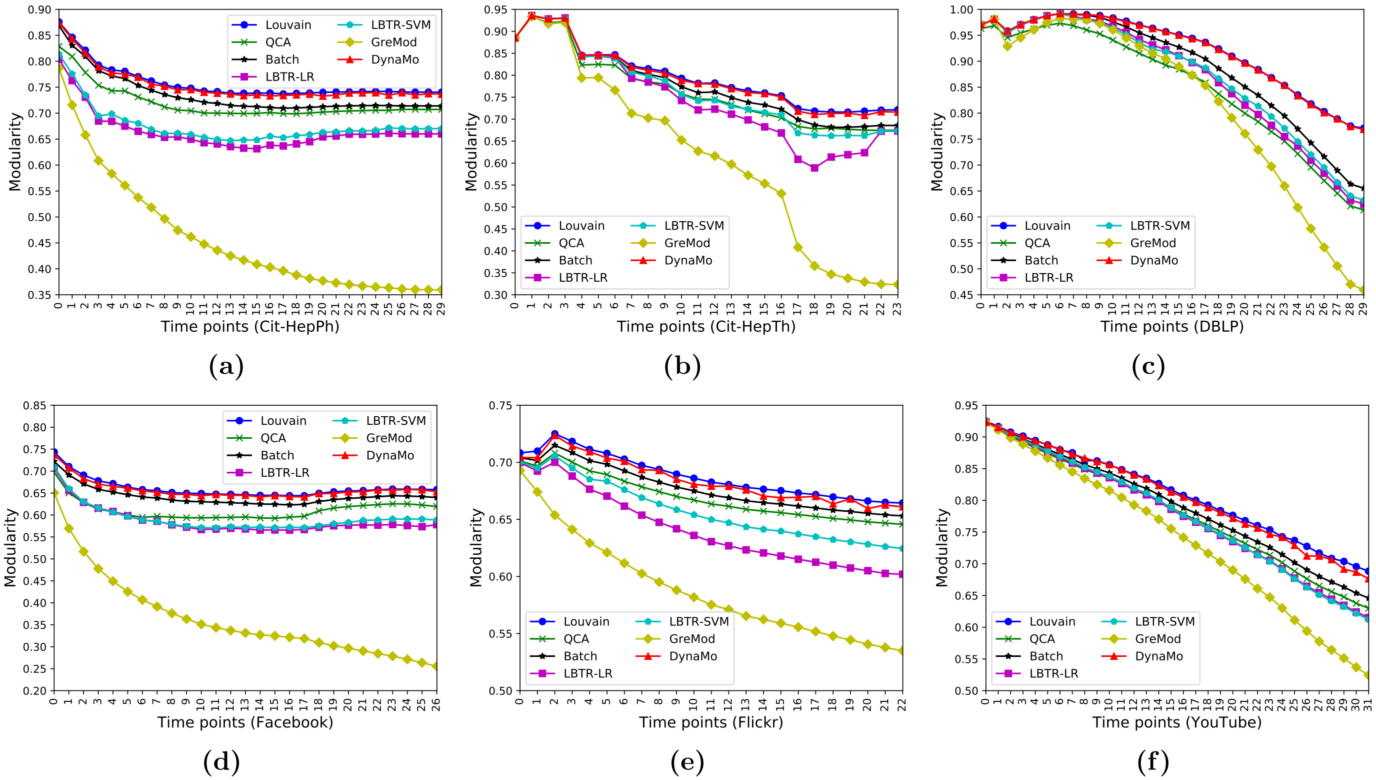
• Compared with QCA [13], DynaMo obtains over 2x, 2x, 4x and 5x speed up on the series of network snapshots of Cit-HepTh, Facebook, Flickr and YouTube, respectively. DynaMo is as efficient as QCA on DBLP network, and spends slightly more time on Cit-HepPh network than QCA.

• Compared with GreMod [15], DynaMo spends more time on most of the networks, and only performs better on the Flickr and YouTube network.

• Compared with LBTR [16], DynaMo is much more efficient than LBTR-SVM, and spends slightly more time than LBTR-LR on certain networks.

4.5.7 Summary of the Experimental Evaluation

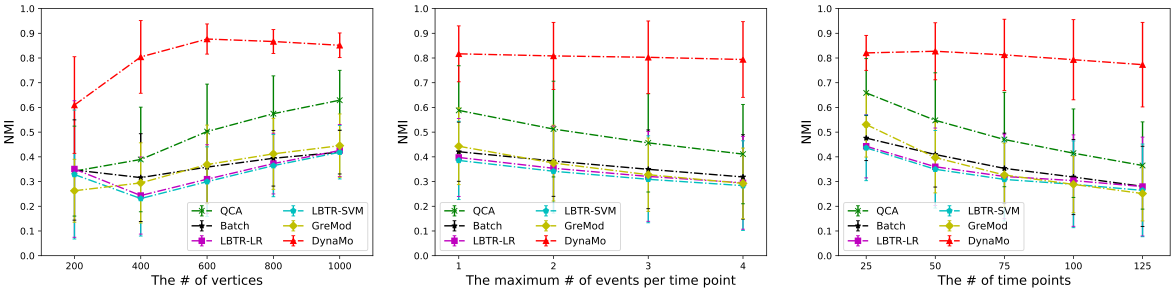
DynaMo consistently outperforms all the other 5 dynamic algorithms on 6 real-world networks and 10,000 synthetic networks in terms of the effectiveness (i.e., modularity, NMI and ARI) of detecting communities. DynaMo has almost identical performance as Louvain in terms of the effectiveness, with only 0.27% to 1.7% lower modularity on certain networks. DynaMo also performs comparably well in terms of the efficiency. For instance, in terms of the cumulative elapsed time results, DynaMo outperforms Louvain, Batch and LBTR-SVM, and obtains similar performance as QCA and LBTR-LR. Even though GreMod acts more efficient than DynaMo, DynaMo is much more effective than GreMod (e.g., GreMod has the worst effectiveness performance running on nearly all datasets). In conclusion, DynaMo significantly outperformed the state-of-the-art dynamic algorithms in terms of effectiveness, and demonstrated much more efficient than the state-of-the-art static algorithm, Louvain algorithm, in detecting communities of dynamic networks, while also maintaining similar efficiency as the best set of competing dynamic algorithms.



(a) (b) (c)

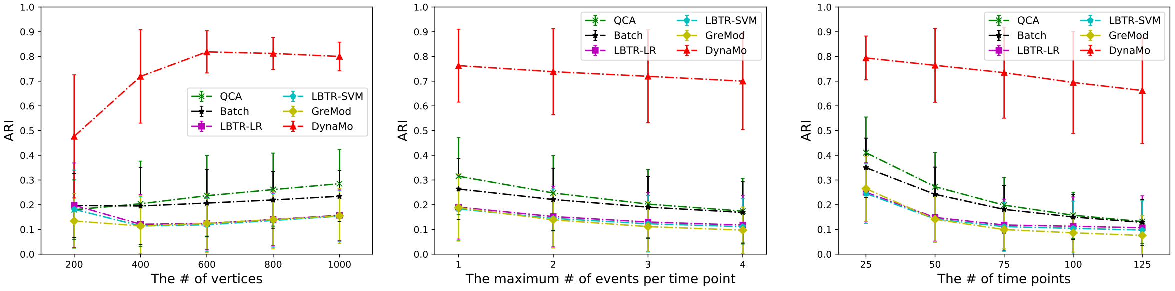
(d) (e) (f)

Figure 4.3: The modularity results of real-world networks. (a) Cit-HepPh. (b) Cit-HepTh. (c) DBLP. (d) Facebook. (e) Flickr. (f) YouTube.



(a) **(b)** **(c)**

Figure 4.4: The NMI results of synthetic networks. (a) The # of vertices. (b) The maximum # of events per time point. (c) The # of time points.



(a) **(b)** **(c)**

Figure 4.5: The ARI results of synthetic networks. (a) The # of vertices. (b) The maximum # of events per time point. (c) The # of time points.

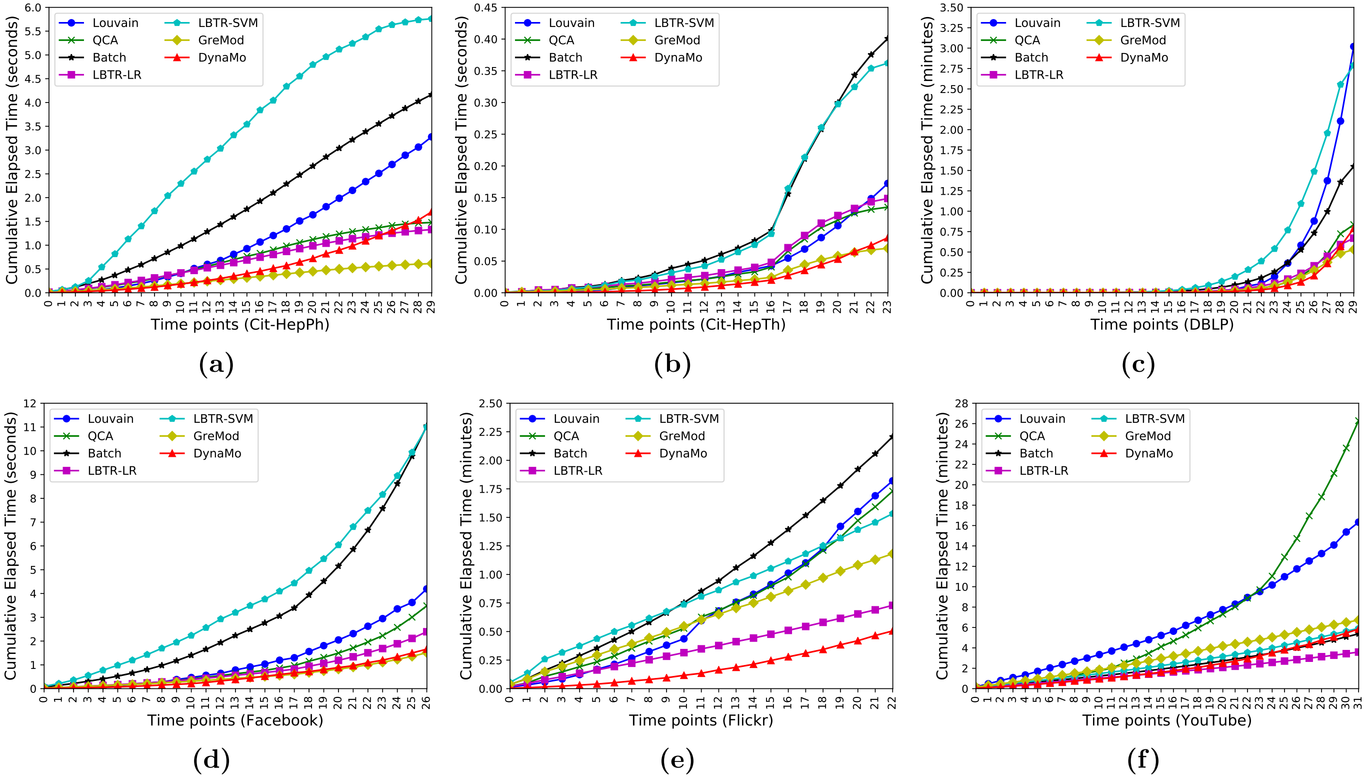


Figure 4.6: The cumulative elapsed time results of real world networks. (a) Cit-HepPh. (b) Cit-HepTh. (c) DBLP. (d) Facebook. (e) Flickr. (f) YouTube.

4.6 Conclusion

In this chapter, we presented DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities in dynamic networks. We also present the theoretical guarantees to show why/how our operations could maximize the modularity, while avoiding redundant and repetitive computations. In the experimental evaluation, a comprehensive comparison has been made among our algorithm, Louvain algorithm and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

Chapter 5: FRiPAL: Face Recognition in Privacy Abstraction Layer

Data-driven mobile applications are becoming increasingly popular in civilian and law enforcement. RapidGather, for instance, is an smartphone application that collects data from individual, and spreads rapid emergency responses. Image data is widely used in such applications, and machine learning methods could be utilized to analyze the image data. However, people would hesitate to share the data without protecting their privacy. In this chapter, we propose to utilize dimensionality reduction techniques for privacy-preserving machine learning in face recognition for the image data. To demonstrate the proposed approach, we implement a client server system, FRiPAL. With extensive experiments, we show that FRiPAL is efficient, and could preserve the privacy of data owners while maintaining the utility for data users. [5]

5.1 Introduction

Modern data-driven applications are becoming increasingly popular in civilian and law enforcement. Such applications collect data from the smartphones, analyze the data at back-end systems, and help people to make decisions. RapidGather [20], for instance, is a data-driven emergency response application that collects different types of data from smartphones, and spreads rapid emergency response to citizens and authorities. However, smartphone users might hesitate to share their data, if RapidGather could not protect the data privacy properly.

Gathering and analyzing photos rapidly is of great importance in emergency events. For instance, in the Boston Marathon bombing scenario (a potential RapidGather use case), even if information transmission immediately through Internet, social media and news report, it still took several days for authorities to gather photos from smartphone users who were in that area, and pore through thousands of photos to identify the suspects. We come up with a privacy-preserving mechanism that could motivate the data owners to share their photos with the authorities, and the

[5] This chapter was published in IEEE Conference on Dependable and Secure Computing 2017 [101]. Copyright permission is included in Appendix A.

authorities could query photos from the crowd around the scene rapidly, and recognize the wanted suspects effectively.

In this chapter, we propose a privacy-preserving machine learning framework for face recognition for the image data in the RapidGather application. Machine learning is an important tool to model the appearance of faces and to classify them. Eigenface [102] and Fisherface [103] have been utilized for a long time for face recognition. However, those traditional methods do not consider privacy issues. As the demand for privacy increasing, privacy-preserving machine learning becomes an emerging area. To date, a few approaches rely on cryptographic protocols (e.g. homomorphic [104] or commutative encryption [105]) or data perturbation (e.g. random projection [106]) techniques have been proposed. However, the cryptography-based approaches suffer from low efficiency, due to high computation and communication cost. The data perturbation based approaches suffer from low accuracy, due to the loss of useful information. Therefore, instead of utilizing cryptography or data perturbation techniques, we propose to utilize dimensionality reduction techniques for privacy-preserving machine learning. These techniques can efficiently transform the raw data from the data owner to a new set of data before they are given to the data users. Without revealing the raw data, the transformation is irreversible.

We have implemented our methods in a system called FRiPAL, Face Recognition in Privacy Abstraction Layer, which is a privacy-preserving face recognition service design. RapidGather proposed [20] an architecture of Privacy-Enhanced Android (PE-Android) which is an extension of the current Android OS with new privacy features. One of the most important components in PE-Android is the Privacy Abstraction Layer (PAL), which is defined as a wrapper of the low level PE-Android services that allows the developers to develop privacy preserving applications in their traditional way. FRiPAL has been integrated into RapidGather as a privacy-preserving face recognition service for image data.

The contributions of this chapter are as follows:

- We propose to utilize dimensionality reduction techniques for privacy-preserving machine learning in face recognition. We demonstrate the proposed approach with three dimensionality reduction methods, including Principal Component Analysis (PCA) [107], Linear Discriminant Analysis (LDA) [103] and Discriminant Component Analysis (DCA) [21].

• We design and implement a privacy-preserving face recognition client server system, FRiPAL, which could preserve the privacy of data owners while maintaining the utility for data users.

• Extensive experiments have been conducted on three different public datasets to evaluate FRiPAL in terms of accuracy, privacy and efficiency. The accuracy results show that our system maintains the utility for face recognition. The privacy results illustrate that our system protects the privacy which motivates the data owners to submit photos. The efficiency results demonstrate that our system is efficient for practical usage.

The rest of this chapter is organized as follows. Section 5.2 presents the preliminaries of dimensionality reduction methods. Section 5.3 describes the privacy-preserving face recognition problem and our proposed solution. Section 5.4 describes the system design of FRiPAL. Section 5.5 presents the experimental evaluation. Section 5.6 presents the related works. Section 5.7 presents the conclusion and future work.

5.2 Preliminaries

5.2.1 Privacy-preserving by Dimensionality Reduction

In machine learning, dimensionality reduction is a tool to transform the feature vector from a high dimension space to a low dimension space. It has been used to deal with: (a) over-fitting problems when the number of features far exceed the number of training samples, (b) performance degradation due to suboptimal search, and (c) high computational cost and power consumption resulting from high dimensional feature space. However, in this chapter, we investigate the privacy preserving usage of dimensionality reduction.

Dimensionality reduction is more resilient to reconstruction attacks [106]. Since the mapping from the original feature vectors to a low dimensional subspace is a many-to-one mapping, it is impossible to determine the privately held features from the reduced feature vectors without knowing any of the original feature vectors, as there are infinite possible feature vectors which could lead to identical reduced feature vector. Therefore, by utilizing dimensionality reduction, the data privacy is preserved since this transformation is irreversible. In this chapter, we utilize three dimensionality reduction methods, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Discriminant Component Analysis (DCA).

5.2.2 Principal Component Analysis (PCA)

Consider a training data set consisting of n m-dimensional vectors: $\boldsymbol{X}$=$\{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} \in \mathbb{R}^m$. Below shows the general steps of PCA:

1. Compute the m-dimensional mean vector $\boldsymbol{\mu}$ of the whole data set:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x_i} \tag{5.1}$$

2. Compute the scatter matrix $\bar{\boldsymbol{S}}$ (alternatively, the covariance matrix) of the whole data set:

$$\bar{\boldsymbol{S}} = \sum_{i=1}^{n}(\boldsymbol{x_i} - \boldsymbol{\mu})(\boldsymbol{x_i} - \boldsymbol{\mu})^T \tag{5.2}$$

3. Compute the eigenvectors $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_m}\}$ and corresponding eigenvalues $\{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ of scatter matrix $\bar{\boldsymbol{S}}$ through spectral decomposition, e.g. eigen decomposition.

4. Sort the eigenvectors by non-increasing eigenvalues and choose d eigenvectors with the largest eigenvalues to form a projection matrix $\boldsymbol{W_{pca}} \in \mathbb{R}^{m\times d}$, where each column is an eigenvector.

5. Transform each sample onto the new subspace:

$$\boldsymbol{x_i'} = \boldsymbol{W_{pca}^T} \times \boldsymbol{x_i} \tag{5.3}$$

   where $\boldsymbol{x_i} \in \mathbb{R}^m$, and $\boldsymbol{x_i'} \in \mathbb{R}^d$.

$\boldsymbol{W_{pca}}$ is the PCA projection matrix. The parameter d determines the dimension of the subspace of the transformed data, and the signal power retained after dimensionality reduction. For instance, suppose the original feature vectors have full signal power $\sum_{i=1}^{m}\lambda_i$, the transformed data has signal power $\sum_{i=1}^{d}\lambda_i$, and signal power $\sum_{i=d+1}^{m}\lambda_i$ has been irreversibly lost. We consider more privacy is preserved, as more signal power losing. Therefore, d could be utilized to control the level of privacy.

5.2.3 Linear Discriminant Analysis (LDA)

Consider a k-class training data set consisting of n m-dimensional vectors $\boldsymbol{X}$=$\{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} \in \mathbb{R}^m$. Each training sample $\boldsymbol{x_i}$ associates with a class label y_i indicating its belonging to one of the k classes $C_1, C_2, \ldots, C_k$. Each class C_j contains n_j training samples in this data set. Below shows the general steps of LDA:

1. Compute the total mean vector $\boldsymbol{\mu} \in \mathbb{R}^m$, and the class mean vector $\boldsymbol{\mu_j} \in \mathbb{R}^m$, $j = 1, 2, \ldots, k$:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x_i} \qquad \boldsymbol{\mu_j} = \frac{1}{n_j}\sum_{y_i \in C_j}\boldsymbol{x_i} \tag{5.4}$$

2. Compute the between-class scatter matrix $\boldsymbol{S_B}$ and the within-class scatter matrix $\boldsymbol{S_W}$:

$$\boldsymbol{S_B} = \sum_{j=1}^{k} n_j(\boldsymbol{\mu_j} - \boldsymbol{\mu})(\boldsymbol{\mu_j} - \boldsymbol{\mu})^T \tag{5.5}$$

$$\boldsymbol{S_W} = \sum_{j=1}^{k}\sum_{y_i \in C_j}(\boldsymbol{x_i} - \boldsymbol{\mu_j})(\boldsymbol{x_i} - \boldsymbol{\mu_j})^T \tag{5.6}$$

3. Compute the eigenvectors $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_m}\}$ and corresponding eigenvalues $\{\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_m\}$ of scatter matrix $\boldsymbol{S_W^{-1}}\boldsymbol{S_B}$ through spectral decomposition, e.g. eigen decomposition.

4. Sort the eigenvectors by non-increasing eigenvalues and choose d eigenvectors with the largest eigenvalues to form a projection matrix $\boldsymbol{W_{lda}} \in \mathbb{R}^{m \times d}$, where each column is an eigenvector.

5. Transform each sample onto the new subspace:

$$\boldsymbol{x_i'} = \boldsymbol{W_{lda}^T} \times \boldsymbol{x_i} \tag{5.7}$$

   where $\boldsymbol{x_i} \in \mathbb{R}^m$, and $\boldsymbol{x_i'} \in \mathbb{R}^d$.

$\boldsymbol{W_{lda}}$ is the LDA projection matrix. Unlike PCA, LDA can reduce the m dimensionality at most to $k - 1$, because of the k-discriminant constraint of LDA.

5.2.4 Discriminant Componenet Analysis (DCA)

The training data set is the same as described in Section 5.2.3. Below shows the general steps of DCA:

1.–2. The same as LDA's 1.–2.

3. Compute the regulated between-class scatter matrix $\boldsymbol{S'_B}$, the regulated within-class scatter matrix $\boldsymbol{S'_W}$, and the regulated total scatter matrix $\boldsymbol{\bar{S}'}$:

$$\boldsymbol{S'_B} = \boldsymbol{S_B} + \rho' \boldsymbol{I} \quad \boldsymbol{S'_W} = \boldsymbol{S_W} + \rho \boldsymbol{I} \tag{5.8}$$

$$\boldsymbol{\bar{S}'} = \boldsymbol{S'_B} + \boldsymbol{S'_W} = \boldsymbol{\bar{S}} + (\rho + \rho')\boldsymbol{I} \tag{5.9}$$

where ρ' and ρ are ridge parameters, and $\boldsymbol{\bar{S}} = \boldsymbol{S_B} + \boldsymbol{S_W}$.

4. Compute the eigenvectors $\{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_m}\}$ and corresponding eigenvalues $\{\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_m\}$ of scatter matrix $\boldsymbol{S_W^{-1}}\boldsymbol{\bar{S}'}$ through spectral decomposition, e.g. eigen decomposition.

5. Sort the eigenvectors by non-increasing eigenvalues and choose d eigenvectors with the largest eigenvalues to form a projection matrix $\boldsymbol{W_{dca}} \in \mathbb{R}^{m \times d}$, where each column is an eigenvector.

6. Transform each sample onto the new subspace:

$$\boldsymbol{x'_i} = \boldsymbol{W_{dca}^T} \times \boldsymbol{x_i} \tag{5.10}$$

where $\boldsymbol{x_i} \in \mathbb{R}^m$, and $\boldsymbol{x'_i} \in \mathbb{R}^d$.

$\boldsymbol{W_{dca}}$ is the DCA projection matrix. Similar to LDA, DCA could at most reduce the m dimensionality to $k - 1$. There are works [21] discussing about the influence of parameters ρ and ρ' on the performance of DCA. In this chapter, we set $\rho = 0.02$ and $\rho' = 0.1$.

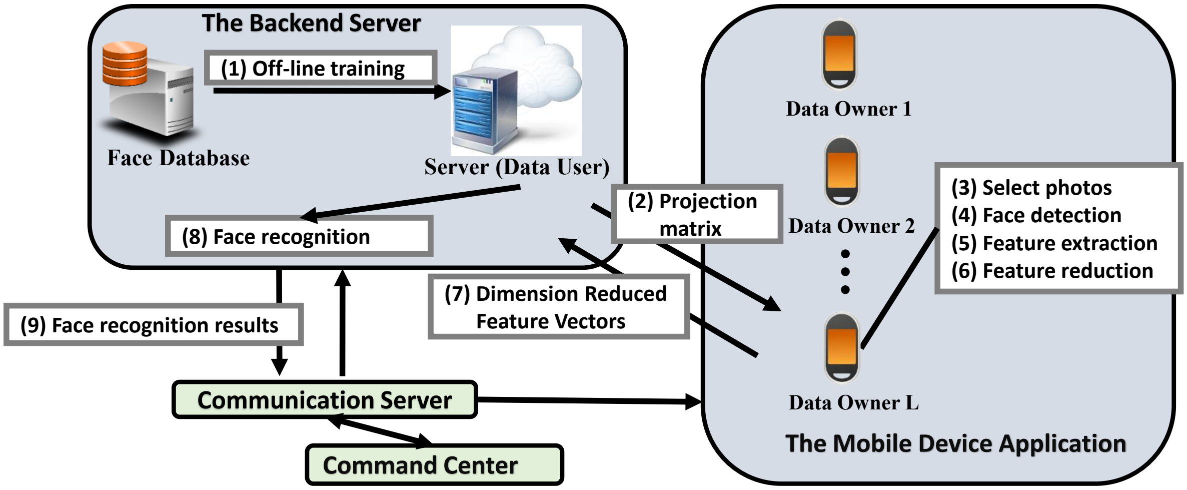


Figure 5.1: FRiPAL Framework.

5.3 Privacy-preserving Face Recognition

5.3.1 Problem Description

We consider a two-party problem, where the data user (e.g. authorities) has a centralized face database (e.g. suspects), and the data owner (e.g. smartphone users) owns face images for testing. The goal of privacy-preserving face recognition is to allow the data user to determine if a face from the data owner is contained in his database, without compromising the privacy of data owner. Our privacy preserving face recognition framework contains three steps: feature extraction, privacy-preserving dimensionality reduction and classification. Below describes the details of the methods we utilized for each step to test our framework. However, in practice, the specific methods utilized in each step could also be replaced by other corresponding (more advanced) methods.

5.3.2 Feature Extraction

In this step, we transform the face image into feature vector (FV). Two feature extraction methods are utilized, respectively.

5.3.2.1 Pixel Feature

Pixel feature is a vector of all pixel values of a grayscale image. For instance, a 100×100 grayscale face image has a FV of $10,000$ length.

5.3.2.2 Gabor Feature

Gabor feature is utilized for edge detection and texture representation of images. For a face image, a set of Gabor filters are applied, and the downsized magnitude results forms its Gabor feature. For instance, applying 40 Gabor filters on a 100×100 face image, and downsizing each reuslt to 30×30, results in a FV of 36,000 length.

5.3.3 Privacy-preserving Dimensionality Reduction

To preserve the privacy of testing data, PCA, LDA and DCA are utilized to transform FV to dimension reduced feature vector (DRFV), respectively. Below shows the three dimensionality reduction methods in detail.

In the case of PCA, suppose the projection matrix is $\boldsymbol{W_{pca}}$, each testing phase begins with selecting a parameter d. Then, project FV on the $m \times d$ matrix $\boldsymbol{W_{pca}}$ to get $DRFV$.

In our case, the dimension of FV of a image is usually much larger than the number of training data. For LDA, this would result in (a) the within class scatter matrix $\boldsymbol{S_w}$ becoming singular, and (b) the overfitting of the transformed data. To overcome this issue, a two step dimensionality reduction method applies. In the training, the m-dimensional training data is projected to a r-dimensional subspace using a $m \times r$ PCA projection matrix $\boldsymbol{W_{pca}}$, where $r < n-k$, n is the number of training data, and k is the number of unique classes. Then, the r-dimensional data is projected into a $k-1$ dimensional subspace using a $r \times (k-1)$ LDA projection matrix $\boldsymbol{W_{lda}}$. In the testing, each FV is projected on a $m \times (k-1)$ matrix $\boldsymbol{W_{pca}} \cdot \boldsymbol{W_{lda}}$ to get $DRFV$.

Comparing with LDA, adding the ridge parameters in DCA makes the within class scatter matrix $\boldsymbol{S_w}$ non-singular. However, applying two step dimensionality reduction could also relax the overfitting issue and improve the efficiency. Therefore, we apply PCA before DCA in the same way as described for LDA.

PCA is the common step for all three dimensionality reduction methods. As discussed in Section 5.2.2, the number of Principal Components (PCs) in PCA, namely parameter d, has an impact on the balance between privacy and utility. For instance, in our problem, the data user can determine the range of acceptable number of PCs they want to keep for them to have good utility of the data. The data owner can choose to share their data at any of the dimensions in that range, or not at all. Therefore, we could use the number of PCA PCs as the privacy policy level. In this chapter, we use the privacy policy level and the number of PCs interchangeably.

5.3.4 Classification

Support Vector Machine (SVM) [108] is utilized as our classification method. Each subject is trained multiple models, where each model subjects to a feature type, a dimensionality reduction method and one privacy policy level. Each model specifies a threshold of the probability $\theta \in [0, 1]$.

In the testing phase, given a $DRFV$ of a face image, each two-class SVM model outputs a binary result and a probability calculated from the SVM decision value. If the probability is larger than a model's θ, we consider that the testing face belongs to the corresponding subject. The testing face might be recognized as multiple subjects, we return the subject whose model outputs the highest probability as the final result. If the testing face is not recognized as any subject, we consider it is not in the database.

5.4 FRiPAL System Design

FRiPAL is a privacy-preserving face recognition framework, which enables rapid photo collection and face recognition, while ensuring the data owner control over the data privacy. FRiPAL supports two feature extraction methods (Pixel Feature and Gabor Feature), and three dimensionality reduction methods (PCA, LDA and DCA). Figure 5.1 shows the main components of FRiPAL: back-end server, mobile application, communication server, and command center.

Figure 5.1 also shows the work flow of FRiPAL. The back-end server begins with an offline training to prepare the classification models, mean vectors, projection matrices and data scale parameters. In each use case,

1. The mobile application updates the mean vectors and the projection matrices from the back-end server.

2. The end user selects photos for face detection. Then, the mobile application performs feature extraction and dimensionality reduction on the selected faces.

3. The mobile application sends DRFVs to the back-end server.

4. The back-end server performs face recognition and sends results to the command center.

Below describes the design and implementation details about each components.

5.4.1 Back-end Server

This component provides the server side support of FRiPAL, including 1) information synchronization, 2) privacy-preserving face recognition, and 3) results update.

5.4.1.1 Information Synchronization

This process synchronizes the mean vector and the projection matrix with the mobile application. In each use case, mobile application sends the client-side version number to the back-end server. The back-end server updates the newest projection matrix and mean vectors to the mobile application if any update is available.

5.4.1.2 Privacy-preserving Face Recognition

In each use case, the server receives DRFV, the feature type, the dimensionality reduction method, and privacy policy level from the mobile application. Then, each DRFV is tested against all the subjects' models with corresponding settings (Section 5.3.4).

5.4.1.3 Results Update

If a wanted subject (e.g. suspect) is recognized in a given photo, the back-end server sends the face recognition results to the command center.

5.4.2 *Mobile Application*

The mobile application is developed upon Android API level 23, including 1) face detection, 2) feature reduction, and 3) DRFV upload.

5.4.2.1 Face Detection

The Haar Feature-based Cascade Classifiers [109] are adopted for face detection, which is implemented using OpenCV 3.0.0 library. The library contains a pre-trained classifier and API calls to do the face detection over an grayscale image.

5.4.2.2 Feature Reduction

Given a detected grayscale face, the application first resizes it to a predefined width and height (e.g. 100 × 100). Then, the specified type of feature is extracted. The same dimensionality reduction procedure (Section 5.3.3) applies on the FV to generate DRFV. As mentioned in Section 5.3.3, the end user is allowed to specify the preferred privacy level (the number of PCs).

5.4.2.3 DRFV Upload

The application uploads the DRFV, along with the feature type, the dimensionality reduction method and selected privacy policy level to the back-end server.

5.4.3 Communication Server and Command Center

The communication server is built upon RabbitMQ [110], which is an open source message broker software that supports the AMQP [111]. The mobile application works as a message producer, which creates and publishes messages to the communication server. The back-end server works as the message consumer that handles the message routed through the communication server.

The command center displays the face recognition results and provides an interface for the agents and authorities to analyze and make further decision.

5.5 Experimental Evaluation

In this section, we evaluate FRiPAL through extensive experiments, in terms of accuracy, privacy and efficiency.

Table 5.1: The summaries of experimental datasets.

	Number of Subjects (tr. / te.)	Number of Photos (tr. / te.)
Yale	28 / 28	5600 / 840
Gatech	0 / 50	0 / 714
Caltech	0 / 11	0 / 126
Total	28 / 89	5600 / 1680

5.5.1 Experiment Setup

5.5.1.1 Environment

The back-end server and commend center have been deployed on the same commodity computer with an 8 core Intel i7-4770 Processor, 32GB RAM, 400GB SSD, running 64-bit Ubuntu 14.04 LTS operating system. The communication server is a Ubuntu 14.04 LTS virtual machine, with 1 Processor, 8GB RAM and 20GB SSD, running on the same commodity computer. The mobile application has been deployed on two Android devices, a Nexus 5X and a Nexus 6P, respectively. The communication between the mobile devices and the commodity computer is through a wireless access point (MWR102 USB Powered Travel Router).

5.5.1.2 Dataset

The experiment dataset consists of data from three public datasets: the Caltech Faces 1999 (Caltech) [112], the Gatech Face Database (Gatech) [113], and the Yale Face Database B (Yale) [114]. More details for training data (tr.) and testing data (te.) are shown in Table 5.1. The training data contains 28 subjects all from Yale, each has 200 photos. Thus, 5600 photos in total.

In order to make the experimental evaluation as unbiased and practical as possible, we generate the testing data from three different public datasets, which contains subjects both from and distinct from the training data. Specifically, the testing data consists of 28 subjects from Yale, each contains 30 photos; 50 subjects from Gatech, 714 photos in total; and 11 subjects from Caltech, 126 photos in total. Thus, 1680 photos in total.

5.5.1.3 Off-line Training

As discussed in Section 5.3, we trained multiple models for each subject. Each training model subjects to a feature type, a dimensionality reduction method and one privacy policy level. In this experiment, we utilized two feature types, Pixel Feature and Gabor Feature and three dimensionality reduction methods, PCA, LDA and DCA, for the training of six types of models, namely, P-PCA, P-LDA, P-DCA, G-PCA, G-LDA and G-DCA. Furthermore, for each dimensionality reduction method, 18 different privacy policy levels (the number of PCA PCs) are selected, namely, 200, 190, 180, 170, 160, 150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40, 30. For each subject in the training data, we utilize 200 (all) of his/her photos as the positive samples and 200 randomly selected photos of other subjects as the negative samples. For each subject, $2 \times 3 \times 18$ two-class SVM models are trained in total.

5.5.2 Accuracy

We use the ROC curve to select the threshold of each model (Section 5.3.4), which is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. In our experiments, we use 101 threshold settings, namely, $\{0, 0.01, 0.02, \ldots, 0.99, 1\}$. Figure 5.2 illustrates six models of subject yaleB11 in Yale, where the x-axis is FPR and the y-axis is TPR. For each model, we consider the point of intersection of the ROC curves and $y = 1 - x$ as the threshold. For instance, in Figure 5.2, the intersection point of P-PCA is around $(0.005, 0.91)$, and the corresponding threshold is 0.71, which means when the threshold is 0.71, the TPR is around 0.91 and the FPR is around 0.005.

We use F1-Score as the measure of accuracy. Figure 5.3 shows the accuracy results of six methods. The x-axis shows 18 privacy policy levels. The y-axis shows the corresponding F1-Score results. Each result is the average over the results of all (28) subjects' models. It is clear that for three dimensionality reduction methods, as the number of PCs increasing, the accuracy increases gradually. It should be noted that the fluctuation in F1-Score is mostly due to the SVM parameter selection. The Gabor feature achieves an overall higher accuracy than the Pixel feature. The lowest accuracy is around 78% when we adopt P-PCA.

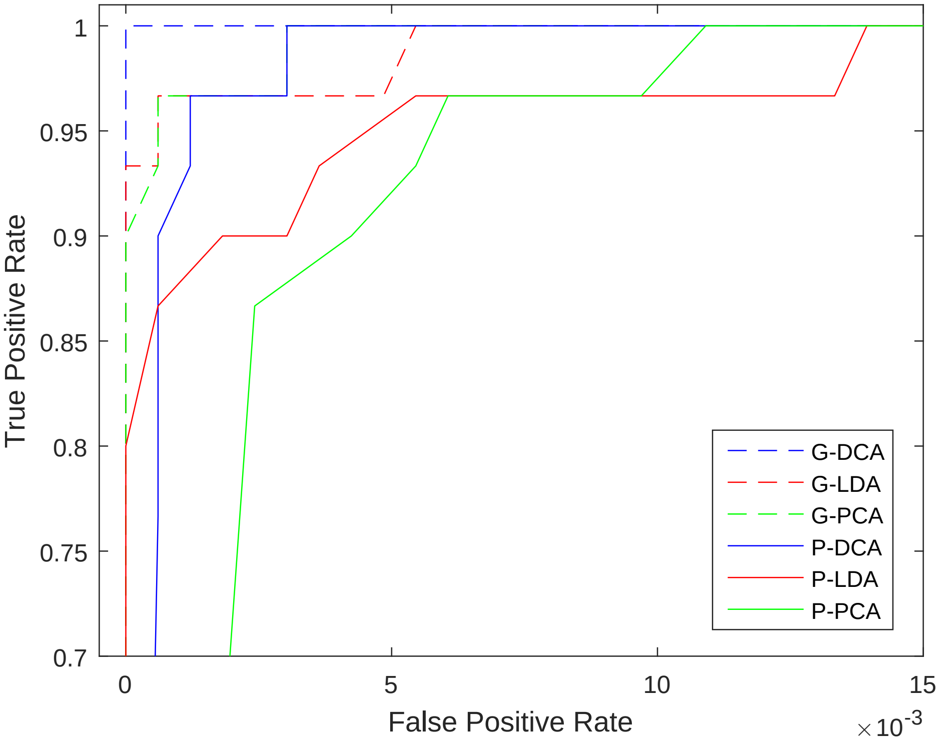


Figure 5.2: The ROC curves of six models (P-PCA, P-LDA, P-DCA, G-PCA, G-LDA and G-DCA, with 200 PCA PCs) of subject yaleB11 in Yale.

Considering the results mentioned above, for PCA, LDA and DCA, the pattern that the accuracy is positive correlated to the number of PCA PCs, is consistent.

5.5.3 Privacy

We utilized two metrics as the measure of privacy, namely Relative Error (RE) and Histogram Similarity (HS). The privacy metrics is conducted on the feature domain. We measure the difference between the original and the reconstructed FV, rather than the original image and reconstructed image. However, it is worth noting that the Pixel feature actually equals to the original image.

The RE is defined as equation (5.11), where N is the number of testing samples and m is the number of features. x_{ij} is the original value of jth feature in ith testing sample, and $\tilde{x}_{ij}$ is the corresponding value of the reconstructed data. Higher RE means more difference, thus, in our setting, more privacy is protected.

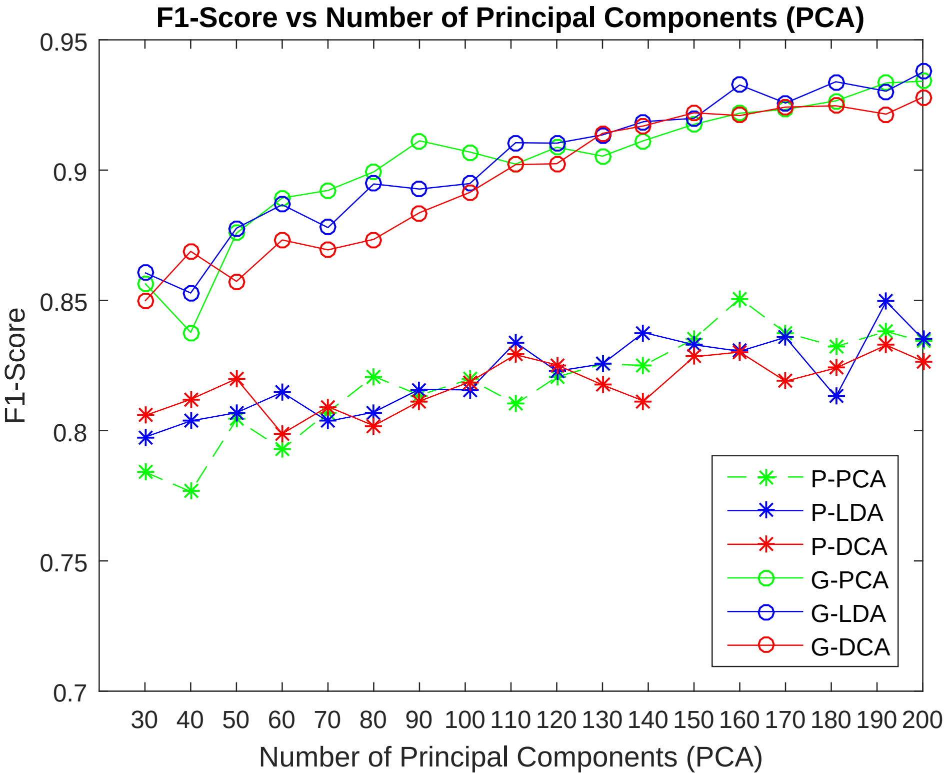


Figure 5.3: F1-score.

$$RE = \frac{1}{N \times m} \sum_{i=1}^{N} \sum_{j=1}^{m} |\frac{x_{ij} - \tilde{x}_{ij}}{x_{ij}}| \tag{5.11}$$

We utilize HS as the measure of image data's privacy. Since we only use grayscale images, the domain of feature value is $[0, 255]$, which is suitable for generating histograms. Let H and $\tilde{H}$ be the histograms of the original data and reconstructed data respectively. Then, the HS between H and $\tilde{H}$ is defined as equation (5.12), where M is the color dictionary (256) of grayscale images, and $S = \max(h_i - \tilde{h}_i)$, $i = 0, 1, \ldots, M-1$. The value of HS is in $[0, 1]$. Lower HS means less similarity, thus, in our setting, more privacy is protected.

$$HS(H, \tilde{H}) = \frac{1}{M} \sum_{i=1}^{M} \big(1 - \frac{|h_i - \tilde{h}_i|}{S}\big) \tag{5.12}$$

The histogram of an image describes its color distribution. Two distinct images may have a similar total color distribution, but it is rare that they have all the same partial color distributions. Therefore, we divide the image into grids. Then, we apply HS on each grid of the original data and

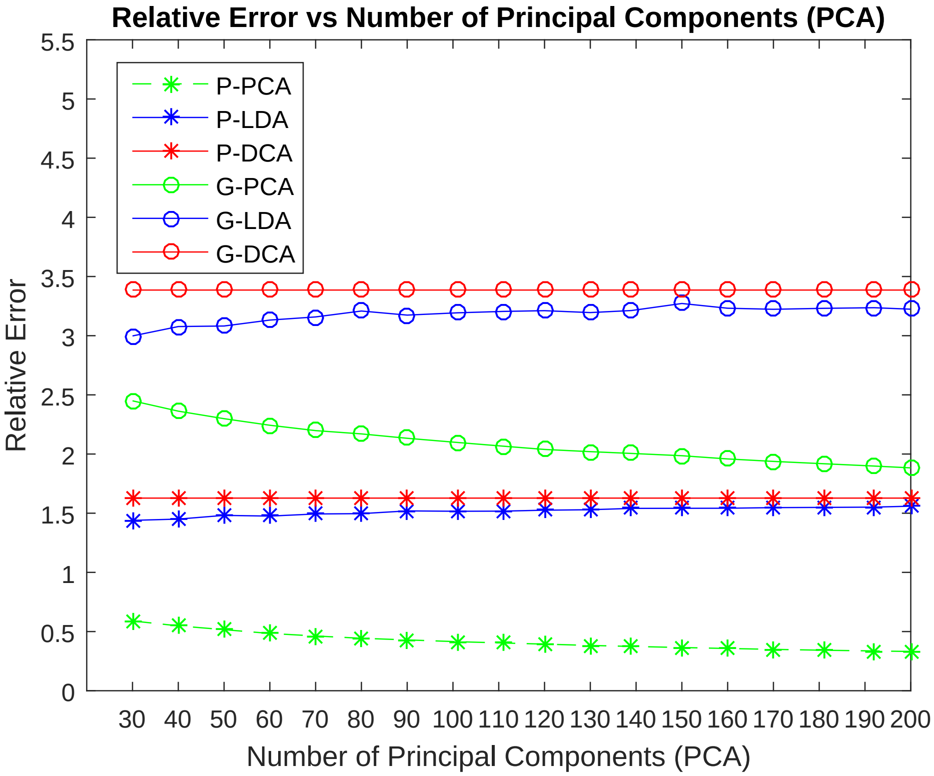


Figure 5.4: Relative error.

the corresponding grid of the reconstructed data. Finally, we calculate the average HS among all the grids. In this experiments, we divide each image into 40 grids.

Figure 5.4 shows the RE of PCA, LDA and DCA, with two feature types. For all privacy policy levels listed, the RE of DCA is always the largest one, the RE of PCA is always the smallest one, and the RE of LDA is always slight less than DCA. This pattern implies that DCA and LDA are more effective than PCA in terms of privacy preserving. Furthermore, as the privacy policy level increasing, the RE of PCA decreases, while the RE of DCA stays around the same value, and the RE of LDA slightly increases. This pattern shows that DCA and LDA are more consistent than PCA in terms of privacy preserving.

Figure 5.5 shows the HS results, which presents the same patterns as the RE results. In addition, the maximum HS of LDA and DCA are less than the minimum HS of P-PCA (30 PCs).

Considering the results mentioned above, in terms of privacy preserving, DCA and LDA performs better and more consistent than PCA. From Figure 5.3, it could be seen that PCA, DCA and LDA has similarity accuracy results, with the same feature types. Therefore, the data owners

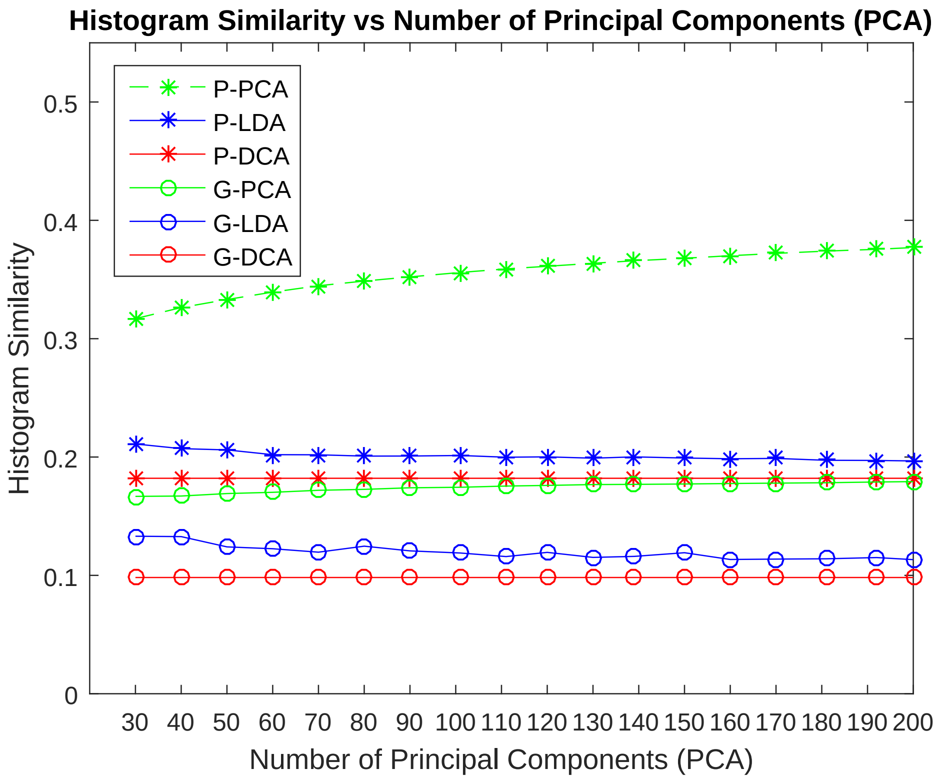


Figure 5.5: Histogram similarity.

could choose to use DCA and LDA, which gives better and more consistent privacy protection, and could use the number of PCs to control the utility.

5.5.4 Efficiency

We designed three experiments to show the system performance. The first experiment measures the time cost of updating projection matrix from the back-end server to the mobile application. The second experiment measures the time cost of the privacy preserving face recognition process, which is the core task of our proposed system. This process covers the face detection and feature reduction on the mobile device, the face recognition on the server side, and the data transmission between the two parties. The third experiment measures the time cost of uploading image to the back-end server. We select 10 images of each subject from the Yale testing data, which results in 280 images in total, and put them on the mobile device. We measure the performance with different privacy policy levels on P-PCA, P-LDA, P-DCA, G-PCA, G-LDA and G-DCA, respectively. In the second and third experiment, 10 images is grouped and processed together.

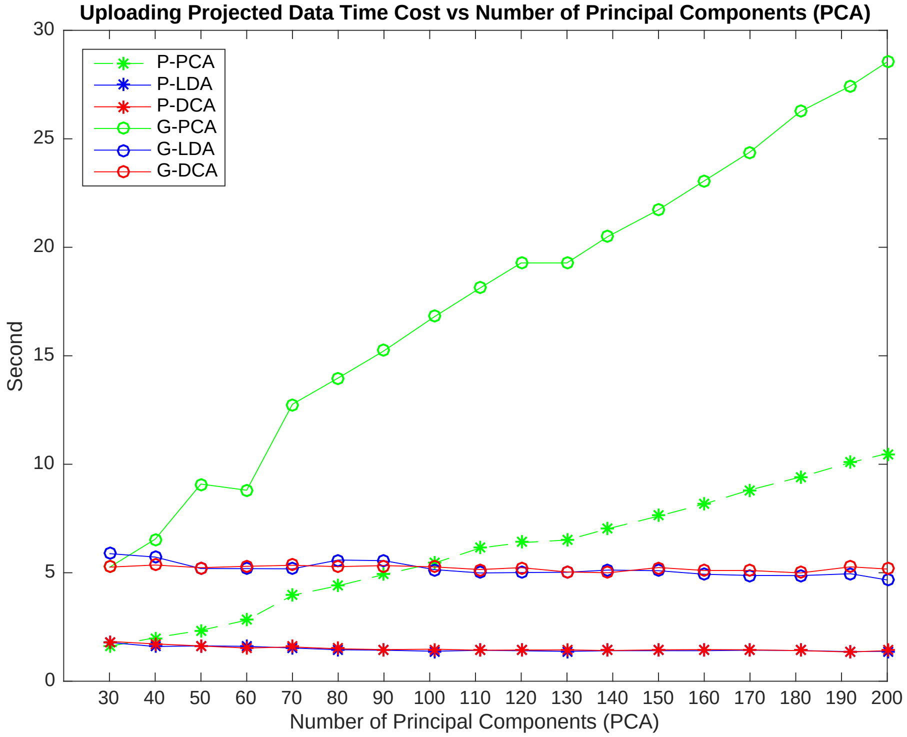


Figure 5.6: Performance evaluation results on Nexus 6P.

Table 5.2 shows the size of different projection matrices. Table 5.3 summarizes the results of the first and second experiments. For instance, for G-PCA, with 200 PCs, when running on Nexus 6P, it takes 27 seconds to update the corresponding projection matrix (the size is 57.6 MB), and it takes 28.5 seconds to accomplish the privacy preserving face recognition of 10 images. Figure 5.6 shows the performance of the second experiment on Nexus 6P. It can be seen that, since DCA and LDA reduced the feature vector to an identical number, their performance are invariant against the change of privacy policy levels. For the PCA method, the more PCs, the more time it takes for processing. The experiment on Nexus 5X gives a similar result.

Considering all the experiment results above, FRiPAL is efficient, and through DCA and LDA, FRiPAL could preserve the privacy of data owners while maintaining the utility for data users.

Table 5.2: The size of different projection matrices.

	Projection Matrix (privacy policy level)				
	40	80	120	160	200
P-PCA	3.2 MB	6.4 MB	9.6 MB	12.8 MB	16 MB
P-LDA	2.2 MB	2.2 MB	2.2 MB	2.2 MB	2.2 MB
P-DCA	2.2 MB	2.2 MB	2.2 MB	2.2 MB	2.2 MB
G-PCA	11.5 MB	23.0 MB	34.6 MB	46.1 MB	57.6 MB
G-LDA	7.8 MB	7.8 MB	7.8 MB	7.8 MB	7.8 MB
G-DCA	7.8 MB	7.8 MB	7.8 MB	7.8 MB	7.8 MB

Table 5.3: The performance of UpdateProjectionMatrix and UploadProjectedData.

	UpdateProjectionMatrix (second)				UploadProjectedData (second / 10 images)			
	Nexus 5X		Nexus 6P		Nexus 5X		Nexus 6P	
	30	200	30	200	30	200	30	200
P-PCA	1.305	10.647	2.058	7.566	1.331	10.797	1.608	10.493
P-DCA	1.325	1.294	1.020	1.208	1.634	1.197	1.833	1.41
P-LDA	1.266	1.277	1.133	1.024	1.472	1.212	1.786	1.378
G-PCA	3.878	30.881	3.924	27.062	6.251	29.957	5.248	28.543
G-DCA	5.135	4.476	4.030	3.653	7.198	5.365	5.259	5.170
G-LDA	4.344	4.116	3.499	3.861	6.982	5.2	5.88	4.667

5.6 Related Works

5.6.1 Data Transformation for Privacy-preserving

Data perturbation is an important technique for protecting the data privacy. [115] proposes to perturb the individual data with additive or multiplicative noise that is generated from certain distributions (e.g., Gaussian). [116] proposes to transforms the original whole data set by applying a random rotation matrix. However, both approaches suffer from a decreasing of the accuracy. Moreover, the first approach cannot resistant against the attack of noise filtering out [117], while the perturbed data obtained by the second approach can be restored by another rotation matrix [118]. Our proposed method maintains the utility, while is more resistant to the reconstruction attack.

5.6.2 Privacy-preserving Face Recognition

Erkin et al. [119] has proposed the first privacy preserving face recognition. They consider a two party problem, data user owns a database of face images, and data owner wants to know whether the face image he owns is in data user's database. The data owner does not want to reveal the image nor the recognition result, while the data user does not want to leak the privacy of his face image database. To resolve the problem, an additively homomorphic public key encryption scheme has been used to securely calculate the distance between the data owner's data and the data in the database. Their work suffers from a heavy computation and communication cost by involving the homomorphic encryption, and it cannot be applied to other machine learning methods directly. Sadeghi et al. [120] has proposed a hybrid solution using homomorphic encryption and garbled circuits, which improves the previous work by shifting most of the computation and communication cost to the pre-computation phase. Rather than encryption, we propose a more efficient and general method by using the dimensionality reduction methods.

5.7 Conclusion

This chapter explores the usage of dimensionality reduction techniques on privacy preserving face recognition. To demonstrate the proposed approach, we implement an efficient privacy preserving face recognition client server system, FRiPAL, using three dimensionality reduction methods, PCA, LDA and DCA with two types of features. The system performance is evaluated on two Android devices, Nexus 5X and Nexus 6P. The results confirm the efficiency of your system for real life usage. Through the extensive experiments, all three methods have similar accuracy results when using the same feature type. RE and HS is utilized to illustrate the privacy preserving performance. As the privacy policy level increasing, the privacy preserving of PCA degrades, while DCA and LDA has a more consistent and better results than PCA. Therefore, DCA and LDA maintain the utility while provide abetter privacy preserving. In the future, we will work on applying our methods on the multiple data owner and multiple data user scenario.

Chapter 6: Utility-aware Privacy-preserving Data Releasing

In the big data era, more and more cloud-based data-driven applications are developed that leverage individual data to provide certain valuable services (the utilities). On the other hand, since the same set of individual data could be utilized to infer the individual's certain sensitive information, it creates new channels to snoop the individual's privacy. Hence it is of great importance to develop techniques that enable the data owners to release privatized data, that can still be utilized for certain premised intended purpose. Existing data releasing approaches, however, are either privacy-emphasized (no consideration on utility) or utility-driven (no guarantees on privacy). In this work, we propose a two-step perturbation-based utility-aware privacy-preserving data releasing framework. First, certain predefined privacy and utility problems are learned from the public domain data (background knowledge). Later, our approach leverages the learned knowledge to precisely perturb the data owners' data into privatized data that can be successfully utilized for certain intended purpose (learning to succeed), without jeopardizing certain predefined privacy (training to fail). Extensive experiments have been conducted on Human Activity Recognition, Census Income and Bank Marketing datasets to demonstrate the effectiveness and practicality of our framework.

6.1 Introduction

As the advent and advance of cloud computing and data science in this big data era, more and more cloud-based data-driven applications are developed by different service providers (the data users, such as Facebook, LinkedIn and Google). Most of these applications leverage the vast amount of data collected from each individual (the data owner) to offer certain valuable service back to the corresponding individual or for the other political and commercial purposes, such as friend recommendation, human activity recognition, health monitoring, targeted advertising and election prediction. However, the same set of data could be repurposed in different ways to infer certain sensitive personal information, which would jeopardize the individual's privacy.

In the recent Facebook data leak scandal (April, 2018) [121], about 87 million Facebook users' data were collected by a Facebook quiz app (a cloud-based data-driven application) and then paired with information taken from their social media profile (including their gender, age, relationship status, location and "likes") without any privacy-preserving operations being taken other than anonymization. Thus, the data user or the other adversaries that have the access to the data can still infer certain sensitive information of each individual from his/her data, such as identity, sexual orientation and marital status. The unprecedented data leak scandal raised the alarm of privacy concerns among cloud-based data-driven applications which could became a big obstacle that impedes the individuals from releasing their data to the service providers to receive valuable service (the utilities).

A similar situation could happen in the patient-hospital scenario as shown in Fig. 6.1. Patient Alice (the data owner) would like to release her data to hospital Bob (the data user) with the premise of using it for disease A diagnosis. However, people like Eve (could be Bob), who works in the same hospital and has the access to Alice's data, could use the same data to infer certain irrelevant sensitive information about Alice, such as her disease B diagnosis. In this case, some individuals (e.g., Facebook users or Alice) would like to release their data to receive good utilities, while on the premise that the service providers are prevented from inferring certain sensitive information from their data (e.g., identity, sexual orientation and marital status). Therefore, it is of vital importance to develop a utility-aware privacy-preserving data releasing framework for cloud-based data-driven applications, which enables the released data to be utilized for certain premised intended purpose (utility target), without jeopardizing the corresponding data owner's certain privacy target.

Designing such general utility-aware privacy-preserving data releasing framework is rather challenging. To date, a few related approaches have been proposed [122, 123, 124, 125, 25, 21, 18, 17, 101]. However, these approaches cannot fulfil all the privacy requirements needed in the cloud-based data-driven application scenario. For example, approaches that relied on additive or multiplicative random noise perturbation [123] and k-anonymity [122] cannot handle the curse of dimensionality. Differential privacy machine learning approaches [124, 125, 25] have been proposed

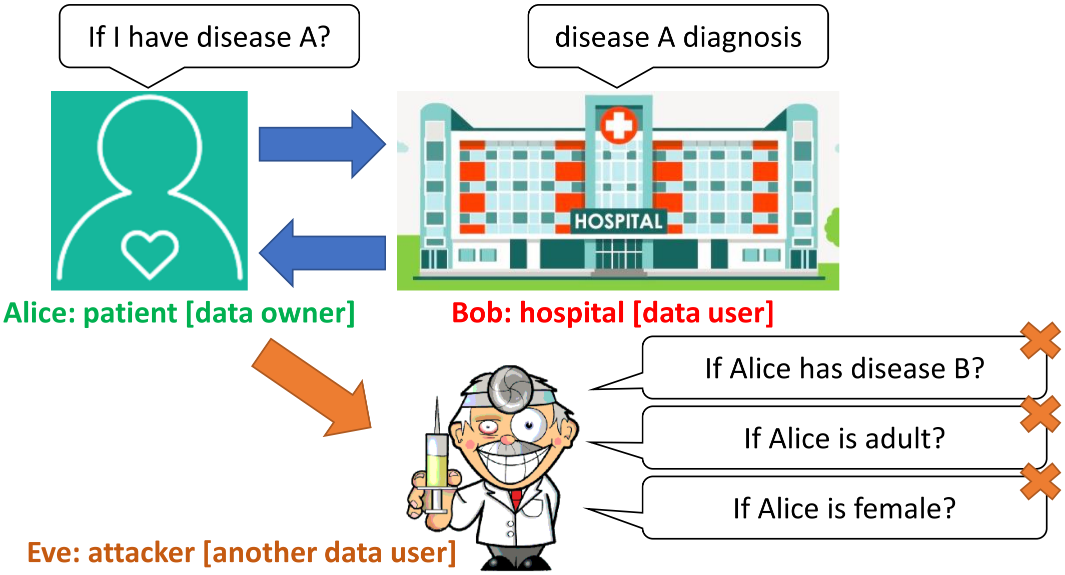


Figure 6.1: An example in patient-hospital scenario.

to publish machine learning models while preserving the training data privacy. In this chapter, however, we consider the scenario that the machine learning models have been trained in advance by the cloud-based service providers (the data users). The data to be protected would appear as the testing data, which is beyond the scope of those approaches. Besides, [126] has shown that some record-level differential privacy approaches applied to collaborative learning scenario are ineffective in dealing with inference attacks. Dimensionality reduction based approaches [21, 18, 17, 101, 127] have been proposed to preserve privacy while maintaining most of the utility. However, despite of their good experimental performance on several public datasets, those approaches didn't introduce any uncertainty to hide the sensitive information, which failed to show the needed guarantees on the privacy targets mathematically.

To address the challenges mentioned above, in this chapter, we devote to solve a two-party exemplar problem. The data user (i.e., the cloud-based service provider) use his/her domain knowledge and public domain data to train a model to provide certain service in advance. The data owner would like to receive the service via sharing his/her own data as the testing data to the data user. The data owner predefines several privacy targets (sensitive information) that he/she would like to prevent the data user from inferring from his/her data. By "predefines", we assume that the data owner knows what the malicious inference and the corresponding domain knowledge and public domain data will be utilized by the malicious data users.

In this chapter, a two-step perturbation-based utility-aware privacy-preserving data releasing framework is proposed to tackle this problem. Given certain specific utility/privacy targets (i.e., the inference problems and the corresponding domain knowledge and public domain data), our approach precisely transforms the original data into privatized data that can be successfully utilized for certain intended purpose (learning to succeed), without jeopardizing certain predefined privacy (training to fail). The first step is a coarse-grained data perturbation method, Joint Utility/Privacy Analysis (JUPA). JUPA is an subspace-optimized projection method, which combines the advantages from both DCA [21] (utility driven projection) and MDR [18] (privacy emphasized projection), and tries to find a subspace projection that could optimize for both utility and privacy targets with the knowledge learned from the public datasets. The second step is a fine-grained data perturbation method inspired by the "label changing" problems (e.g., adversarial image perturbation [128, 129, 130, 131, 132, 133, 134]) in the computer vision area, where in order to change the image's class membership, very small perturbations are added to the image that remain quasi-imperceptible to a human vision system. For instance, [130] proposed a Maximum Mean Discrepancy [135] (MMD) statistic test related approach to make semantic change to the appearance of given images. We propose to use a MMD-like loss function to leverage the knowledge (i.e., the discriminant distance among the classes in each privacy target) learned from the public domain dataset and precisely perturb each coarse-grain-perturbed data to a fine-grain-perturbed data that belongs to a randomly selected privacy target class (the data owner's secret parameter).

In the experiments, we have tested our frame on three public datasets: Human Activity Recognition, Census Income and Bank Marketing datasets. The experiment results demonstrate that (a) JUPA is a more general utility-aware dimensionality reduction approach compared with DCA [21] and MDR [18]; (b) given certain predefined privacy target, our fine-grained data perturbation approach can reduce the accuracy of the corresponding inference attack to the level of random guessing.

The rest of chapter is organized as follows: Section 6.2 presents the related works. Section 6.3 presents the preliminaries about dimensionality reduction and maximum mean discrepancy. Section 6.4 describes our proposed utility-aware privacy-preserving data releasing framework. Section 6.5 presents the experimental evaluation. Section 6.6 presents the conclusion and future work.

6.2 Related Work

A few privacy-preserving data releasing approaches have been proposed, including solutions based on cryptography [136, 137, 138, 139], differentially private synthetic data generation [140, 141, 142, 143], and dimensionality reduction [106, 21, 18, 17, 101, 127]. Most of the cryptography-based approaches are designed for specific applications/algorithms. For instance, [137] developed a privacy-preserving ridge regression system that utilized additive homomorphic encryption and Garbled circuits to train a ridge regression model with the encrypted data statistic shares submitted by multiple data owners. [138] proposed to use cryptographic building blocks to enable testing new samples while protecting both the ML model and the submitted samples, in three popular classification protocols: hyperplane decision, Naïve Bayes, and decision trees. Although cryptography-based approaches prevent the adversaries from performing inference attack on the encrypted data/model, they are not flexible enough to work for general data releasing purpose.

Differential privacy (DP) [19] is one of the most popular standard for quantifying individual privacy. DP aims to protect the privacy of individuals via adding randomness to the aggregate information. Differentially private synthetic data generation approaches utilize those differentially private aggregate information to generate synthetic data. For instance, [141] considers to use differential privacy component analysis for data releasing. "Plausible Deniability" [142], has been proposed and achieved by applying a privacy test after generating the synthetic data. The generative model proposed in [142] is a probabilistic model which captures the joint distribution of features based on correlation-based feature selection (CFS) [144]. [140] proposed an algorithm which combines the multiplicative weights approach and exponential mechanism for differentially private data release. [143] proposed a micro-aggregation [145] based differential private data releasing approach which reduces the noise required by differential privacy based on k-anonymity. Although DP-based approaches provide strong guarantees on individuals' privacy, they does not take any utility targets into account in designing their privacy-preserving data releasing mechanisms.

The dimensionality reduction approaches provide a promising way to irreversibly transform the original data, and publish the transformed data for general usage. [106] proposed to use random projection matrix to project the original data to a lower dimensional space. However, the random projection method mainly focuses on the privacy targets without considering the utility targets, which downgrades its utility performance. A few dimensionality reduction based privacy-preserving approaches focusing on maintaining the utility have been proposed [21, 18, 17, 101, 127]. For instance, [21] proposed to use Discriminant Component Analysis (DCA), a supervised version of Principle Component Analysis (PCA), to project the data into a lower dimensional space that maximizes the discriminant power for specific targets. However, since DCA mainly focuses on the utility target, it might maintain the utility while somewhat preserve the privacy because of the information loss through the dimensionality reduction. However, DCA could not control or adjust the projection matrix in terms of the privacy target. [18] proposed Multi-class Discriminant Ratio (MDR), which projects the data based on a pair of classification targets, (a) a privacy-insensitive and (b) a privacy-sensitive target. RUCA [127], improves the MDR to provide more flexibility to adjust the trade-off or tuning between utility and privacy. However. these approaches do not introduce any uncertainty/randomness to hide the sensitive information, which failed to show the needed guarantees on the privacy targets mathematically.

6.3 Preliminaries

6.3.1 Dimensionality Reduction via Eigenvalue Decomposition

An important component of our framework is supervised dimensionality reduction technique (i.e., it relies on data labels). Consider a dataset with N training samples $\{x_1, x_2, \ldots, x_N\}$, where each sample has M features ($x_i \in \mathbb{R}^M$). Since the same dataset could be utilized in different classification problems, each classification problem c has a unique set of labels L_i^c associated with the corresponding training samples x_i. Without loss of generality, we assume the dataset could be utilized for a single utility target U and a single privacy target P. Then, each training sample x_i has two labels $L_i^u \in \{1, 2, \ldots, L^u\}$ and $L_i^p \in \{1, 2, \ldots, L^p\}$. L^u and L^p are the numbers of classes of the utility target and the privacy target, respectively.

Based on Fisher's linear discriminant analysis [146, 147], given a classification problem, the within-class scatter matrix of its training samples contains most of the "noise information", while the between-class scatter matrix of its training samples contains most of the "signal information".

We define the within-class scatter matrix and the between-class scatter matrix for the utility target as follows:

$$S_{W_U} = \sum_{l=1}^{L^u} \left(\sum_{i=1}^{N_l^u} x_i x_i^T - N_l^u \mu_l \mu_l^T \right) \tag{6.1}$$

$$S_{B_U} = \sum_{l=1}^{L^u} N_l^u \mu_l \mu_l^T - N \mu \mu^T \tag{6.2}$$

where $\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$, μ_l is the mean vector of all training samples belonging to class l, N_l^u is the number of training samples belonging to class l of the utility target.

Similarly, for the privacy target the within-class scatter matrix and the between-class scatter matrix define as:

$$S_{W_P} = \sum_{l=1}^{L^p} \left(\sum_{i=1}^{N_l^p} x_i x_i^T - N_l^p \mu_l \mu_l^T \right) \tag{6.3}$$

$$S_{B_P} = \sum_{l=1}^{L^p} N_l^p \mu_l \mu_l^T - N \mu \mu^T \tag{6.4}$$

Let W be an $K \times M$ projection matrix, in which $K < M$. Given testing sample x, $\hat{x} = x^T \cdot W$ is its subspace projection. Our framework combines the advantages of two eigenvalue decomposition based dimensionality reduction techniques: DCA [21] (utility driven projection) and MDR [18] (privacy emphasized projection).

6.3.1.1 Discriminant Component Analysis (DCA)

DCA [21] involves searching for the projection matrix $W \in R^{M \times K}$:

$$DCA = \frac{det(W^T S_{B_U} W)}{det(W^T (\bar{S} + \rho I) W)} \tag{6.5}$$

where $det(\cdot)$ is the determinant operator, ρI is a small regularization term added for numerical stability, and $\bar{S} = S_{W_U} + S_{B_U} = \sum_{i=1}^{N} x_i x_i^T - N \mu \mu^T$.

The optimal solution to this problem can be derived from the first K principal generalized eigenvectors of the matrix pencil $(S_{B_U}, \bar{S} + \rho I)$.

6.3.1.2 Multi-class Discriminant Ratio (MDR)

MDR [18] considers both the utility target and the privacy target, which is defined as:

$$MDR = \frac{det(W^T(S_{B_U})W)}{det(W^T(S_{B_P} + \rho I)W)} \tag{6.6}$$

where ρI is a small regularization term added for numerical stability.

The optimal solution to MDR can be derived from the first K principal generalized eigenvectors of the matrix pencil $(S_{B_U}, S_{B_P} + \rho I)$.

6.3.2 Maximum Mean Discrepancy (MMD)

The Maximum Mean Discrepancy [135] (MMD) statistic has been proposed to test whether two distributions p and q are different based on the samples drawn from each of them. In this work, our fine-grained data perturbation utilized a MMD-like loss function inspired by a kernel-MMD solution [148]. Let p and q be two distributions defined on a domain $\mathcal{X}$. Given observations $X := \{x_1, x_2, \ldots, x_m\}$ and $Y := \{y_1, y_2, \ldots, y_n\}$, drawn i.i.d. from p and q respectively, the kernel-MMD solution [148] is defined as:

$$\begin{aligned} MMD[\mathcal{F}, X, Y] =& \frac{1}{m}\sum_{i=1}^{m}\phi(x_i) - \frac{1}{n}\sum_{i=1}^{n}\phi(y_i) \\ =& \Big[\frac{1}{m^2}\sum_{i,j=1}^{m}k(x_i, x_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n}k(x_i, y_j) \\ &+ \frac{1}{n^2}\sum_{i,j=1}^{n}k(y_i, y_j)\Big]^{\frac{1}{2}} \end{aligned} \tag{6.7}$$

where $\mathcal{F}$ is a unit ball in a universal RKHS $\mathcal{H}$, defined on the compact metric space $\mathcal{X}$, with associated kernel $k(\cdot, \cdot)$, and $\phi(x) = k(x, \cdot)$. $MMD[\mathcal{F}, X, Y] \approx 0$, if and only if $p = q$.

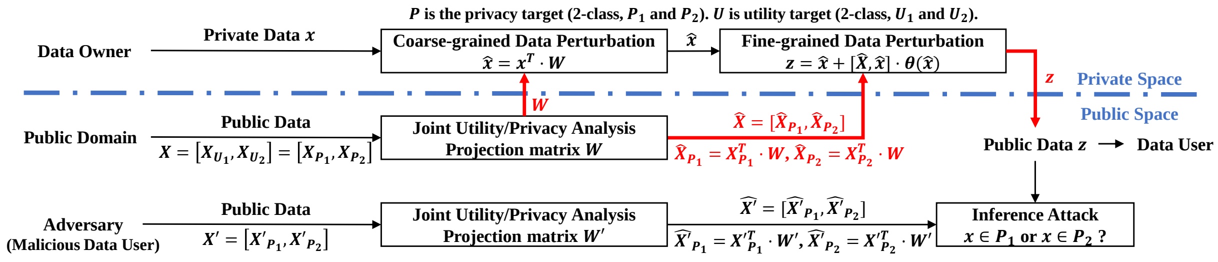

P is the privacy target (2-class, P_1 and P_2). U is utility target (2-class, U_1 and U_2).
Data Owner
Private Data x
Coarse-grained Data Perturbation
$\hat{x} = x^T \cdot W$
$\hat{x}$
Fine-grained Data Perturbation
$z = \hat{x} + [\hat{X}, \hat{x}] \cdot \theta(\hat{x})$
z
Private Space
Public Space
W
Public Domain
Public Data
$X = [X_{U_1}, X_{U_2}] = [X_{P_1}, X_{P_2}]$
Joint Utility/Privacy Analysis
Projection matrix W
$\hat{X} = [\hat{X}_{P_1}, \hat{X}_{P_2}]$
$\hat{X}_{P_1} = X_{P_1}^T \cdot W, \hat{X}_{P_2} = X_{P_2}^T \cdot W$
Public Data z → Data User
Adversary
(Malicious Data User)
Public Data
$X' = [X'_{P_1}, X'_{P_2}]$
Joint Utility/Privacy Analysis
Projection matrix W'
$\widehat{X'} = [\widehat{X'}_{P_1}, \widehat{X'}_{P_2}]$
$\widehat{X'}_{P_1} = X'^T_{P_1} \cdot W', \widehat{X'}_{P_2} = X'^T_{P_2} \cdot W'$
Inference Attack
$x \in P_1$ or $x \in P_2$?

Figure 6.2: A utility-aware privacy-preserving data releasing framework.

6.4 Utility-aware Privacy-preserving Data Releasing Framework

6.4.1 Framework Overview

6.4.1.1 Problem Statement

As illustrated in Fig. 6.2, our framework involves two parties: the data owner(s) and the data user(s). The data user uses public data (background knowledge) to train a machine learning model (i.e., classification model) in advance to provide certain service (the utility targets). The data owner would like to release her private data to the data user for the purpose of the utility targets, and prevent the malicious data user from inferring certain predefined sensitive information (the privacy targets). Assume the data owner and the data user have access to similar set of public data (background knowledge) utilized for both utility and privacy targets, but the data owner does't know the data user's machine learning model. Our goal is to perturb the data owner's private data x into perturbed data z with the knowledge of predefined utility and privacy targets, such that the perturbed data z could be utilized successfully for the intended purposes (i.e., the utility target achieves similar accuracies using either x or z), without jeopardizing the data owner's privacy (i.e., the privacy target get no better accuracy than random guessing while using z). To achieve this goal, we propose a two-step data perturbation framework (Fig. 6.2). More details about the two steps are described in Section 6.4.2 and Section 6.4.3.

6.4.1.2 Threat Model

The adversaries in our framework are the malicious data users, who have the access to the public data that could be utilized as the training data for certain predefined privacy target. The adversaries would like to infer the knowledge (i.e., class) of the privacy target (i.e., classification problems) associated with the data owner's private data based on the corresponding perturbed data and public data (background knowledge). For instance, as shown in Fig. 6.2, we shall assume that the predefined privacy target is a two-class (i.e., $\{P_1, P_2\}$) classification problem (utility targets are independent from the privacy task). Let $X = [X_{P_1}, X_{P_2}]$ be the public training samples for the privacy target, where X_{P_i} $(i \in \{1, 2\})$ presents the samples associated with class i. Let x be the data owner's private data, where $s \in \{P_1, P_2\}$ is its original (privacy target) class and $t \in \{P_1, P_2\}$ is its expected (privacy target) class after our privacy-preserving data releasing operation. The data owner could publish z (i.e., the perturbed version of x) using our framework F: $z \leftarrow F(x, t, X_{P_1}, X_{P_2})$. The adversary has to use his/her approach $A(z, X_{P_1}, X_{P_2})$ to guess/infer the original (privacy task) class s.

6.4.2 Coarse-grained Data Perturbation

In this section, we introduce a general dimensionality reduction method Joint Utility/Privacy Analysis (JUPA). JUPA combines the advantages from both DCA [21] (utility driven projection) and MDR [18] (privacy emphasized projection), and tries to find a subspace projection that could optimize for both utility and privacy targets with the knowledge learned from the public datasets. Our problem settings are exactly the same as described in Section 6.3.1. For simplicity, we shall start from a single utility/privacy target scenario. JUPA tries to find a projection matrix W that maximize the following function:

$$JUPA = \frac{det(W^T(S_{B_U} + \rho_1' S_{W_P})W)}{det(W^T(S_{W_U} + \rho_1 S_{B_P} + \rho_0 I)W)} \tag{6.8}$$

where $det(\cdot)$ is the determinant operator, ρ_0 is regularization parameter added for numerical stability, and ρ_1, ρ_1' are privacy-utility adjustment parameters.

The optimal solution to JUPA can be derived from the first K principal generalized eigenvectors of the matrix pencil $(S_{B_U} + \rho_1' S_{W_P}, S_{W_U} + \rho_1 S_{B_P} + \rho_0 I)$. After getting the projection matrix W, we perturb the data owner's private data x and the training data matrix X as $\hat{x} = x^T W$ and $\hat{X} = X^T W$.

Additionally, JUPA can be generalized to multiple utility/privacy targets by including multiple corresponding scatter matrices:

$$JUPA = \frac{det(W^T(\sum_{i=1}^{N_u} S_{B_{U_i}} + \sum_{i=1}^{N_p} \rho_i' S_{W_{P_i}})W)}{det(W^T(\sum_{i=1}^{N_u} S_{W_{U_i}} + \sum_{i=1}^{N_p} \rho_i S_{B_{P_i}} + \rho_0 I)W)} \tag{6.9}$$

6.4.2.1 Utility vs. "Somewhat Privacy"

JUPA maintains a trade-off between the utility and "somewhat privacy". "somewhat privacy" means our coarse-grained perturbation approach optimizes towards privacy, but could not provide privacy guarantee (as in Section 6.4.3). On one hand, JUPA optimizes a subspace projection that maximizes the "signal to noise" ratio of the utility targets. On the other hand, JUPA optimizes towards two "mappings" for privacy targets: a "many-to-one" mapping, after which data belonging to the same privacy class are near each other (tuned by ρ_1'); and a "one-to-many" mapping, after which data belonging to different privacy classes are far from each other (tuned by ρ_1). By adjusting ρ_1 and ρ_1', JUPA could be tuned between DCA [21], MDR [18] and RUCA [127]. For instance, if $\rho_1 = \rho_1' = 0$, this projection method becomes DCA; if ρ_1 is very large and $\rho_1' = 0$, it becomes MDR as the term S_{B_P} dominates $(S_{W_U} + \rho_1 S_{B_P} + \rho_0 I)$; and if $\rho_1' = 0$ it becomes RUCA. Higher value of ρ_1 and ρ_1' means more emphasis on the privacy targets.

6.4.3 Fine-grained Data Perturbation

In this section, we introduce a perturbation approach that gradually change the privacy target classification label of a given data owner's coarse-grained perturbed data $\hat{x}$ from its source (original) label s to a randomly selected target label t, via adding precisely calculated noise. For simplicity, we shall assume a single 2-class ($\{P_1, P_2\}$) privacy target scenario. Except for the data owner's coarse-grained perturbed data $\hat{x}$, another input for this approach is the coarse-grained perturbed training data matrix $\hat{X} = [\hat{X}_{P_1}, \hat{X}_{P_2}]$, where $\hat{X}$ will be split into two parts: $\hat{X}^G = [\hat{X}^G_{P_1}, \hat{X}^G_{P_2}]$ and $\hat{X}^V = [\hat{X}^V_{P_1}, \hat{X}^V_{P_2}]$. $\hat{X}^G$ is the "ground truth" training data matrix being used to gradually "train" the fine-grained perturbed data. $\hat{X}^V$ is the "verification" training data matrix being used to verify the current label of the input data $\hat{x}$ and intermediate perturbed data.

Given coarse-perturbed private data $\hat{x}$, we start from randomly selecting a target label $t \in \{P_1, P_2\}$ for $\hat{x}$, and use the following function to decide its current (source) label $s \in \{P_1, P_2\}$:

$$\begin{aligned}
s =& label(\hat{x}) =_{\{l:l\in\{P_1,P_2\}\}} \big(\frac{1}{|\hat{X}_l^V|}\sum_{\hat{x}_i\in\hat{X}_l^V}\phi(\hat{x}_i)-\phi(\hat{x})\big)^2 \\
=&_{\{l:l\in\{P_1,P_2\}\}}\frac{1}{|\hat{X}_l^V|^2}\sum_{\hat{x}_i,\hat{x}_j\in\hat{X}_l^V}k(\hat{x}_i,\hat{x}_j) \\
&-\frac{2}{|\hat{X}_l^V|}\sum_{\hat{x}_i\in\hat{X}_l^V}k(\hat{x}_i,\hat{x})+k(\hat{x},\hat{x}) \\
=&_{\{l:l\in\{P_1,P_2\}\}}\frac{1}{|\hat{X}_l^V|^2}\sum_{\hat{x}_i,\hat{x}_j\in\hat{X}_l^V}k(\hat{x}_i,\hat{x}_j) \\
&-\frac{2}{|\hat{X}_l^V|}\sum_{\hat{x}_i\in\hat{X}_l^V}k(\hat{x}_i,\hat{x})
\end{aligned} \tag{6.10}$$

Our approach perturbs $\hat{x}$ in an iterative fashion. Let z_i be the ith ($i = 1, 2, \dots$) intermediate perturbed data. Then, our iterative data sanitization function is defined as:

$$\begin{aligned}
z_0 =& \hat{x} \\
z_i =& z_{i-1} + \theta(z_{i-1}) \quad (i = 1,\ 2,\ \dots)
\end{aligned} \tag{6.11}$$

where $\theta(z_i)$ is the noise vector being added to z_i. The starting noise vector $\theta(z_0)$ could be initiated as a zero vector or a random vector.

In order to compute $\theta(z_i)$ in each iteration, inspired by the kernel-MMD solution [148] described in Section 6.3.2, we define a loss function as:

$$\begin{aligned}
L(\theta(z_i)) =& \big(\frac{1}{n_t}\sum_{\hat{x}_i\in\hat{X}_t^G}\phi(\hat{x}_i)-\phi(z_i)\big)^2 \\
&-\big(\frac{1}{n_s}\sum_{\hat{x}_i\in\hat{X}_s^G}\phi(\hat{x}_i)-\phi(z_i)\big)^2+\frac{\lambda}{2}\|\theta(z_i)\|_2^2 \\
=& \frac{1}{n_t^2}\sum_{\hat{x}_i,\hat{x}_j\in\hat{X}_t^G}k(\hat{x}_i,\hat{x}_j)-\frac{1}{n_s^2}\sum_{\hat{x}_i,\hat{x}_j\in\hat{X}_s^G}k(\hat{x}_i,\hat{x}_j) \\
&+\frac{2}{n_s}\sum_{\hat{x}_i\in\hat{X}_s^G}k(\hat{x}_i,z_i)-\frac{2}{n_t}\sum_{\hat{x}_i\in\hat{X}_t^G}k(\hat{x}_i,z_i) \\
&+\frac{\lambda}{2}\|\theta(z_i)\|_2^2
\end{aligned} \tag{6.12}$$

A large negative value of $L(\theta(z_i))$ indicates z_i belongs to the target class, and a large positive value of $L(\theta(z_i))$ indicates z_i belongs to the source class. Therefore, the value of $\theta(z_i)$ is obtain by minimizing the loss function $L(\theta(z_i))$ gradually, until $label(z_i)$ is t. To solve this optimization problem, we use a gradient descent approach:

$$\begin{aligned}\bigtriangledown_{\theta(z_i)} L(\theta(z_i)) =& \frac{1}{n_s} \sum_{\hat{x}_i \in \hat{X}_s^G} k(\hat{x}_i, z_i) \frac{\hat{x}_i - z_i}{\sigma^2} \\ & - \frac{1}{n_t} \sum_{\hat{x}_i \in \hat{X}_t^G} k(\hat{x}_i, z_i) \frac{\hat{x}_i - z_i}{\sigma^2} \\ & + \lambda \cdot \theta(z_i)\end{aligned} \tag{6.13}$$

$$\theta(z_i) = \theta(z_i) - \alpha \bigtriangledown_{\theta(z_i)} L(\theta(z_i)) \tag{6.14}$$

where we use RBF kernel $k(x_i, x_j) = e^{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}}$ as an example, and α is the learning rate. Finding the most appropriate kernel function is beyond the scope of this chapter, and there are a few papers discussing about kernel selection [149, 150]. In the experimental evaluation, we use the kernel function that gives the highest cross-validation accuracy on the training data.

6.4.3.1 Privacy Guarantee

Considering the “two-class” scenario described in Section 6.4.1, we assume the adversaries’ approach $A(z, X_{P_1}, X_{P_2})$ would be certain kernel-based classification models trained by public available dataset $[X_{P_1}, X_{P_2}]$. Inspired by semantic security [151], we give our definition of a privacy-preserving data releasing framework as below.

6.4.3.2 Privacy-preserving Data Release

We define F as a privacy-preserving data release, if given predefined privacy target and certain adversaries’ approach A, the advantage $Adv[A, F] = |Pr(s = P_1) - Pr(s = P_2)|$ is negligible. (It is straightforward to generate this definition to multi-class scenarios.)

Our proposed framework is a privacy-preserving data releasing framework.

Proof. Given predefined privacy target, certain appropriate kernel function and public available dataset $[X_{P_1}, X_{P_2}]$, our framework precisely perturbs the private data x towards a perturbed data z associated with a randomly selected privacy target label t. Then, given z and $[X_{P_1}, X_{P_2}]$, we have $Pr(s = P_1) = Pr(s = P_2) = \frac{1}{2}$. Therefore, $Adv[A, F] = |Pr(s = P_1) - Pr(s = P_2)| = 0$ is negligible. □

6.5 Experimental Evaluation

6.5.1 Experiment Datasets

We have tested our proposed frame with three public datasets: Human Activity Recognition (HAR) [152], Census Income (Census) [153] and Bank Marketing (Bank) [154]. Each dataset has been split into three subsets: training samples (for perturbation approaches), testing samples (data owner's private data), and adversary training samples (for inference attacks).

Human Activity Recognition (HAR) [152] dataset contains smartphone sensor data (i.e., accelerometer data) of 30 subjects' daily activities, where each sample has 561 features and two labels: activities of daily living (ADL) and identity (ID). In our experiments, we consider ADL as the utility target and ID as the privacy target. Specifically, ADL has 6 types of labels: "Walking", "Walking Upstairs", "Walking Downstairs", "Sitting", "Standing" and "Laying". On the other hand, ID has 30 types of labels, since 30 subjects have contributed to this dataset. The original dataset is unbalanced. For instance, some subjects contribute more data than the others and some ADLs happen more often than the others. As such, for each different ADL-ID combination ($6 \times 30 = 180$ combinations in total), we randomly sampled 20 samples from the original dataset, resulting in 3,600 samples. The numbers of training, testing and adversary training samples are 1,440, 720 and 1,440, respectively. We kept the number of samples in all ADL-ID combinations equal in all sets.

Census Income (Census) [153] dataset has been used to predict whether someone's income exceeds $50K/yr based on census data. We identify two labels of this dataset: "income", where the data user tries to classify if someone's income is "high" (higher than $50K/yr) or "low" (lower or equal to $50K/yr); and "gender" (i.e., male/female) which was one feature in the original dataset. Since based on the application, either "income" or "gender" can be served as utility or privacy targets, we experimented for both cases. Firstly, we removed the samples with missing features. Secondly, we turned all categorical features into numerical features using binary encoding, which resulted in 51 features. Lastly, we randomly sampled 750 samples for each income-gender combination ($2 \times 2 = 4$ combinations in total) from the original dataset, resulting in 3,000 samples. The numbers of training, testing and adversary training samples are 1,200, 600 and 1,200, respectively. As with the *HAR* dataset, we kept the number of samples in all income-gender combinations equal in all sets.

Bank Marketing (Bank) [154] dataset is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The original classification goal is to predict if the client will subscribe a term deposit (marketing purpose). As such, we used the marketing purpose ("marketing") as the utility target, which is a "yes" or "no" binary classification problem. We used marital status ("marital") as the privacy target, which was one feature in the original dataset. Since very few samples have "unknown" marital status, we removed those samples. Thus, "marital" has 3 types of labels: "divorced", "married" and "single". As with the *Census*, we turned all categorical features into numerical features using binary encoding, resulting in 31 features. We randomly sampled 410 samples for each marketing-marital combination ($2 \times 3 = 6$ combinations in total) from the original dataset, resulting in 2,460 samples. The numbers of training, testing and adversary training samples are 984, 492 and 984, respectively. We also kept the number of samples in all marketing-marital combinations equal in all sets.

6.5.2 Experiment Setups

We evaluate the performance of our proposed two perturbation approaches step-by-step, in terms of utility and privacy. In all the experiments, we utilized RBF-kernel SVM to train the machine learning classifiers for both the utility and privacy targets. The utility classifier is to provide certain premised valuable service, while the privacy classifier is to perform the adversaries' inference attack. All the experiments were performed 15 iterations. At each iteration, a 10-fold cross-validation grid search was performed to find the best set of parameters for training utility and privacy classifiers. As discussed in the last section (Section 6.5.1), we evaluate our frame using three datasets and four scenarios. Given a scenario and its dataset, the evaluation metric is the accuracy of its utility/privacy classifiers. Higher accuracy of the utility classifier means providing better utility. Lower accuracy of the privacy classifier means less privacy leakage. The baseline (i.e., lowest accuracy, no privacy leakage) of the privacy classifier should be equal to the probability of random-guess, of which the prediction is drawn i.i.d. from a uniform distribution.

For the coarse-grained perturbation, we compare our proposed JUPA with a full-dimensional baseline method and four existing dimensionality reduction methods, including Random Projection, PCA, DCA and MDR. Moreover, We evaluate JUPA with regularization parameter $\rho_0 = 0.001$, and different combinations of privacy-utility adjustment parameters $\rho_1 = 1$, 10^2, 10^4, $\rho_1' = 1$, 10^2, 10^4. For the fine-grained perturbation, we set $\lambda = 0.001$, $\alpha = 0.1$, and use a zero vector to initiate the starting noise vector $\theta(z_0)$.

6.5.3 Experiment Results

Table 6.1, Table 6.2, Table 6.3 and Table 6.4 shows the experimental results of four scenarios (three datasets), and the following are the main observations and conclusions drawn from experimental results.

Considering the coarse-grained perturbation approach alone, JUPA outperforms the other DR methods in terms of the utility and "somewhat privacy". Compared with PCA and random projection, DCA, MDR and JUPA provide better balance between the utility and "somewhat privacy" performance, since PCA and random projection are not leveraging any help from the "label" information. For instance, in Table 6.1, after applying random projection (coarse-grained), the accuracy of ID (privacy) dropped from 62.78% to 13.75% (providing one of the best privacy performance), and the accuracy of ADL (utility) dropped from 97.22% to 60.28% (giving one of the worst utility performance). On the contrary, after applying PCA (coarse-grained), the accuracy of either utility or privacy does not drop much (providing less "somewhat privacy"). Compared with DCA and MDR, JUPA provides better utility and "somewhat privacy" performance under certain privacy parameters. For instance, in Table 6.1, when $\rho_1 = 1$ and $\rho_1' = 1$, compared with other DR methods, JUPA (coarse-grained) provides the highest accuracy (96.11%) of ADL (utility), and also the second lowest accuracy (only higher than random projection) (21.11%) of ADL (utility). Results in the other scenarios are inline with this observation.

JUPA provides the flexibility for finding a favorable trade-off or tuning between utility and privacy by tuning the privacy parameters. Based on our results, by increasing $\rho_1 = 1$ or $\rho_1' = 1$,

Table 6.1: The mean accuracy percentage results of HAR dataset. ($K = 5$, ADL is the utility target, and ID is the privacy target.)

Projection Method	ADL		ID	
	Coarse	Fine	Coarse	Fine
Full-Dimensional	97.22	66.94	62.78	3.33
Random Projection	60.28	57.36	13.75	3.33
PCA	84.72	73.33	30.28	3.75
DCA	94.58	93.75	23.61	3.33
MDR	91.67	88.75	22.92	4.58
JUPA ($\rho_1 = 1$, $\rho_1' = 1$)	96.11	94.31	21.11	3.75
JUPA ($\rho_1 = 1$, $\rho_1' = 10^2$)	95.83	93.47	20.28	3.61
JUPA ($\rho_1 = 1$, $\rho_1' = 10^4$)	95.56	93.47	19.72	3.33
JUPA ($\rho_1 = 10^2$, $\rho_1' = 1$)	94.44	93.33	20.00	3.33
JUPA ($\rho_1 = 10^2$, $\rho_1' = 10^2$)	94.17	92.64	17.78	3.33
JUPA ($\rho_1 = 10^2$, $\rho_1' = 10^4$)	93.75	92.36	16.67	3.33
JUPA ($\rho_1 = 10^4$, $\rho_1' = 1$)	92.50	88.19	13.61	3.33
JUPA ($\rho_1 = 10^4$, $\rho_1' = 10^2$)	89.58	86.39	12.50	3.33
JUPA ($\rho_1 = 10^4$, $\rho_1' = 10^4$)	87.50	86.11	12.08	3.33

JUPA weights more emphasis on preserving privacy (providing accuracy) with small amount of accuracy drop on the utility. For instance, in Table 6.1, adjusting JUPA from $\rho_1 = 1$, $\rho_1' = 1$ to $\rho_1 = 10^4$, $\rho_1' = 10^4$, results in a 42.78% drop of the ID (privacy) accuracy (from 21.11% to 12.08%), while only resulting in a 8.96% drop of the ADL (utility) accuracy (from 96.11% to 87.50%).

Our fine-grained perturbation approach provides the privacy guarantee. For instance, in all the scenarios, after applying the fine-grained perturbation, the accuracies of privacy targets are all converging to or near to the probability of random-guess.

In our framework, combining JUPA with the fine-grained perturbation outperforms the other options in terms of the utility. For instance, in Table 6.1, compared with other DR methods, DCA, MDR and JUPA (fine-grained) provide relative higher accuracy of ADL (utility) ($\leq 86.11\%$), and when $\rho_1 = 1$ and $\rho_1' = 1$, JUPA provides the best utility accuracy (94.31%). Results in the other scenarios are inline with this observation. The reason is that even though the fine-grained perturbation could provide guarantee for privacy, applying supervised DR methods (DCA, MDR and JUPA) reserves more utility information and need less noise added to the coarse-grained perturbed data to achieve the privacy guarantee.

Table 6.2: The mean accuracy percentage results of census dataset (income). ($K = 1$, income is the utility target, and gender is the privacy target.)

Projection Method	income		gender	
	Coarse	Fine	Coarse	Fine
Full-Dimensional	84.50	69.76	87.33	50.00
Random Projection	58.33	50.50	59.17	50.00
PCA	73.33	70.33	81.67	50.00
DCA	80.00	73.50	56.00	50.00
MDR	76.67	68.33	58.00	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 1$)	82.50	75.33	55.50	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 10^2$)	80.00	75.16	54.67	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 10^4$)	78.33	74.33	54.67	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 1$)	79.17	74.66	55.00	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^2$)	77.50	74.00	54.50	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^4$)	76.67	73.83	54.17	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 1$)	76.00	73.67	53.17	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^2$)	75.00	73.50	52.67	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^4$)	72.00	66.83	51.17	50.00

Table 6.3: The mean accuracy percentage results of census dataset (gender). ($K = 1$, gender is the utility target, and income is the privacy target.)

Projection Method	gender		income	
	Coarse	Fine	Coarse	Fine
Full-Dimensional	87.33	73.50	84.50	50.00
Random Projection	59.17	59.17	58.33	50.00
PCA	81.67	70.33	73.33	50.00
DCA	87.50	80.50	53.17	50.00
MDR	86.67	77.83	56.00	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 1$)	88.00	82.5	57.17	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 10^2$)	87.67	82.17	55.67	50.00
JUPA ($\rho_1 = 1$, $\rho'_1 = 10^4$)	87.50	82.17	55.50	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 1$)	87.67	81.33	55.67	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^2$)	86.67	81.17	54.67	50.00
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^4$)	86.00	80.17	54.67	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 1$)	87.00	80.33	54.33	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^2$)	86.67	79.67	53.50	50.00
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^4$)	85.67	78.67	52.67	50.00

6.6 Conclusion

In this chapter, we proposed a two-step perturbation-based utility-aware privacy-preserving data releasing framework. In the first step, we proposed JUPA, a supervised DR method, which outperforms several existing DR methods in terms of providing utility and "somewhat privacy",

Table 6.4: The mean accuracy percentage results of bank marketing dataset. ($K = 1$, marketing is the utility target, and marital is the privacy target.)

Projection Method	marketing		marital	
	Coarse	Fine	Coarse	Fine
Full-Dimensional	86.38	69.11	45.73	34.15
Random Projection	60.57	54.88	39.23	33.33
PCA	71.14	70.73	41.06	33.33
DCA	84.76	78.66	38.01	33.33
MDR	71.75	67.48	36.79	33.33
JUPA ($\rho_1 = 1.0$, $\rho'_1 = 1.0$)	86.38	81.30	39.63	33.33
JUPA ($\rho_1 = 1.0$, $\rho'_1 = 10^2$)	86.18	79.67	38.82	33.33
JUPA ($\rho_1 = 1.0$, $\rho'_1 = 10^4$)	85.37	78.66	38.41	33.33
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 1.0$)	86.18	76.22	38.01	33.33
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^2$)	85.98	75.61	37.60	33.33
JUPA ($\rho_1 = 10^2$, $\rho'_1 = 10^4$)	85.98	75.41	36.18	33.33
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 1.0$)	85.37	75.20	36.99	33.33
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^2$)	84.35	75.00	36.59	33.33
JUPA ($\rho_1 = 10^4$, $\rho'_1 = 10^4$)	83.13	74.59	35.77	33.33

and provides the flexibility for finding a favorable trade-off or tuning between utility and privacy by tuning the privacy parameters. In the second step, we proposed a fine-grained perturbation approach, which guarantees to provide the protection against inference attacks on certain predefined privacy targets. In the experimental evaluation, we deployed our frame in four scenarios using three public dataset. The experiment results are inline with our expectations and demonstrating the effectiveness and practicality of our framework. Future work will include and extension of JUPA to support non-linear sub-space projections, and an optimized kernel selection method for our fine-grained perturbation approach.

Chapter 7: Locally Differentially Private Distributed Deep Learning

Deep learning often requires a large amount of data. In certain real-world applications, e.g., healthcare applications, the data collected by a single organization (e.g., hospital) is often limited, and the majority of massive and diverse data is often segregated across multiple organizations. As such, it motivates the researchers to conduct distributed deep learning, where the data user (e.g., researcher/organization) would like to build DL models using the data segregated across multiple different data owners (e.g., organizations). However, this could lead to severe privacy concerns due to the sensitive nature of the data, thus the data owners would be hesitant and reluctant to participate. In this chapter, we propose LDP-DL, a privacy-preserving distributed deep learning framework via local differential privacy and knowledge distillation. Our approach use a "teacher-student" paradigm, where each data owner learns a teacher model using its own (local) private dataset, and the data user aims to learn a student model to mimic the output of the ensemble of the teacher models using the unlabelled public data. To ensure privacy, our approach employs local differential privacy on the data owners' side, i.e., the query results of each teacher model. We also design an active query sampling approach that could actively select a subset of the unlabelled public dataset for the data user to query from the data owners, thus save the privacy budget. In the experimental evaluation, a comprehensive comparison has been made among our proposed approach (i.e., LDP-DL), DP-SGD, PATE and DP-FL, using three popular deep learning benchmark datasets (i.e., CIFAR10, MNIST and FashionMNIST). The experimental results show that LDP-DL consistently outperforms the other competitors in terms of privacy budget and model accuracy.

7.1 Introduction

Deep learning (DL) has been shown to achieve extraordinary results in a variety of real-world applications, such as skin lesion analysis [155], active authentication [156], facial recognition

[157, 101], botnet detection [5, 158] and community detection [72]. In traditional DL environment, training data is held by a single organization in a centralized fashion, that executes the DL algorithms. In general, a DL model would be more accurate and robust if it has been trained with more massive and more diverse data. However, in certain real-world applications, e.g., healthcare applications, the data collected by a single organization (e.g., hospital) is often limited, and the majority of massive and diverse data is often segregated across multiple organizations. As such, it motivates the researchers to conduct DL in a distributed fashion, where the data user (e.g., researcher/organization) would like to build DL models using the data segregated across multiple different data owners (e.g., organizations). However, the data owners would be hesitant and reluctant to participate in the data user's distributed deep learning, if the data user's protocol cannot resolve the data owners' important privacy concerns of their data. For instance, it has been shown that the private information could be inferred during the learning process [159], and the membership of certain training data could be traced back from the resulting trained model [160]. Hence, it is imperative to design an effective privacy-preserving distributed deep learning approach.

Designing an effective and efficient privacy-preserving distributed deep learning approach is highly challenging. To date, a few approaches [24, 161, 26] have been proposed for privacy-preserving (distributed) deep learning. Papernot et al. [24] proposes PATE, a "teacher-student" paradigm for privacy-preserving deep learning, where each data owner learns a teacher model using its own (local) private dataset, and the data user aims to learn a student model using the unlabelled public data (but no direct access to the data owners' private data) to mimic the output of the ensemble of the teacher models, i.e., the student learns to make predictions that is the same as the most number of teachers. To ensure privacy, PATE [24] assumes a trusted aggregator to provide a differentially private query interface, where the data user could query the ensemble of the teacher models (from the data owners) using the unlabelled public data to obtain the labels for the training of the student model. However, a fully trusted aggregator barely exists in most of the real-world distributed deep learning scenarios. Chase et al. [161] proposes a private collaborative neural network learning approach, that combines secure multi-party computation (MPC), differential privacy (DP) and secret sharing. Since the MPC protocol is implemented via a garbled circuit whose size is subject to the number of parameters (i.e., the size of the gradient) of the neural network, it tends to be less efficient and not scalable while training larger neural networks. Also, in [161], using secret sharing requires at least two non-colluding honest data users which might not be practical.

To address the challenges mentioned above, in this chapter, we propose LDP-DL, a privacy-preserving distributed deep learning framework via local differential privacy [22] and knowledge distillation [23]. Our approach adopts the same "teacher-student" paradigm as described in PATE [24], where each data owner learns a teacher model using its own (local) private dataset, and the data user aims to learn a student model to mimic the output of the ensemble of the teacher models using the unlabelled public data. Knowledge distillation [23] has been applied on the ensemble of the teacher models to enable faster and more accurate knowledge transferring to the student model, and leverage the advantage of having multiple data owners (teacher models). To ensure privacy, our approach employs local differential privacy on the data owners' side, i.e., the query results of each teacher model, which does not require any trusted aggregator (compared to [24]). Since more queries to the teacher models tends to result in more privacy leakage (i.e., cost more privacy budget), we also design an active query sampling approach that could actively select a subset of the unlabelled public dataset for the data user to query from the data owners. In the experimental evaluation, a comprehensive comparison has been made among our proposed approach (i.e., LDP-DL), DP-SGD [25], PATE [24] and DP-FL [26], using three popular deep learning benchmark datasets (i.e., CIFAR10 [27], MNIST [28] and FashionMNIST [29]). The experimental results show that our LDP-DL framework consistently outperforms the other competitors in terms of privacy budget and model accuracy.

To summarize, our work has the following contributions:

• We present a novel, effective and efficient privacy-preserving distributed deep learning framework using local differential privacy and knowledge distillation.

• We present an active sampling approach to efficiently reduce the total number of queries from the data user to each data owners, so that to reduce the total cost of privacy budget.

• A comprehensive experimental evaluation among our approach, DP-SGD [25], PATE [24] and DP-FL [26] has been conducted on three benchmark dataset (i.e., CIFAR10 [27], MNIST [28] and FashionMNIST [29]). For the sake of reproducibility and convenience of future studies about privacy-preserving distributed deep learning, we have released our prototype implementation of

LDP-DL, information regarding the experiment datasets and the code of our comparison experiments. [6]

The rest of this chapter is organized as follows: Section 7.2 presents the preliminaries including local differential privacy and knowledge distillation. Section 7.3 presents the problem statement and notations of privacy-preserving distributed deep learning, and describes our proposed framework. Section 7.4 presents the experimental evaluation. Section 7.5 presents the related literature review. Section 7.6 concludes.

7.2 Preliminaries

7.2.1 Local Differential Privacy

Differential Privacy (DP) [162, 163] aims to protect the privacy of individuals while releasing aggregated information about the database, which prevents membership inference attacks [160] by adding randomness to the algorithm outcome. Two databases D and D' are neighbors if they differ in only one entry. The formal definition of (ϵ, δ)-Differential Privacy [162, 164] is given as follows:

A randomized mechanism A is (ϵ, δ)-differentially private if for every two neighboring databases D, D' and for any subset $S \subseteq Range(A)$:

$$Pr[A(D) \in S] \leq e^{\epsilon} \cdot Pr[A(D') \in S] + \delta \tag{7.1}$$

where $Pr[\cdot]$ denotes the the probability of an event, $Range(A)$ denotes the set of all possible outputs of algorithm A. Smaller values of ϵ, δ indicates closer between $Pr[A(D) \in S]$ and $Pr[A(D') \in S]$, thus stronger privacy protection gains. When $\delta = 0$ the mechanism A satisfies ϵ-DP, which provides stronger privacy guarantee than (ϵ, δ)- DP, where $\delta > 0$.

Local Differential Privacy (LDP) [22] is the local setting of DP, which does not require any trusted aggregator. In LDP, individuals (i.e., data owners) send their data to the data aggregator after privatizing data by perturbation. Hence, these techniques provide plausible deniability for individuals (i.e., data owners). Data aggregator collects all perturbed values and makes an estimation of statistics such as the frequency of each value in the population. The formal definition is given as follows: ϵ-Local Differential Privacy [165, 166]: A randomized mechanism A satisfies

[6] https://github.com/nogrady/LDP-DL

ϵ-LDP if for any input v_1, v_2 and for any subset $S \subseteq Range(A)$:

$$Pr[A(v_1) \in S] \leq e^{\epsilon} \cdot Pr[A(v_1) \in S] \tag{7.2}$$

Compared with DP, LDP provides more protection to the data owners. Other than sending the private data directly to a trusted aggregator, the data owners could perturb their private data with the mechanism that satisfies ϵ-LDP, and then release the perturbed data. As such, LDP provides a stronger privacy protection, since the aggregator (i.e., data user) only receives the perturbed data and the true values of the private data never leave the hands of the data owners.

7.2.2 Knowledge Distillation

Knowledge Distillation (KD) [167, 23, 168] was originally designed for deep neural network (DNN) compression and knowledge transfer. KD usually considers a "teacher-student" paradigm, where the teacher model is a DNN (or an ensemble of a set of DNNs) that performs well on a given dataset, and the student model is another neural network that may or may not have the same architecture as the teacher model, but aims to mimic the performance of the teacher model(s) using another public dataset. Hinton, et al. [23] proposes an end-to-end knowledge distillation framework with a loss function, namely Distillation Loss, where the output of the teacher model is used as the soft target (i.e., soft label) for the student model, and the overall loss function is presented as below:

$$\begin{aligned} \mathcal{L}(x;\Theta) = \alpha \cdot \mathcal{H}(y, \sigma(z_s; T=1)) \\ + \beta \cdot \mathcal{H}(\sigma(z_t; T=\tau), \sigma(z_s; T=\tau)) \end{aligned} \tag{7.3}$$

where y is the true label of data x, z_s is the output of the student model, z_t is the output of the teacher model, $\sigma(z_s; T=1))$ is the softened label of z_s at temperature $T=1$, and $\sigma(\cdot; T=\tau))$ is the softened label at temperature $T=\tau$, and usually $\tau > 1$.

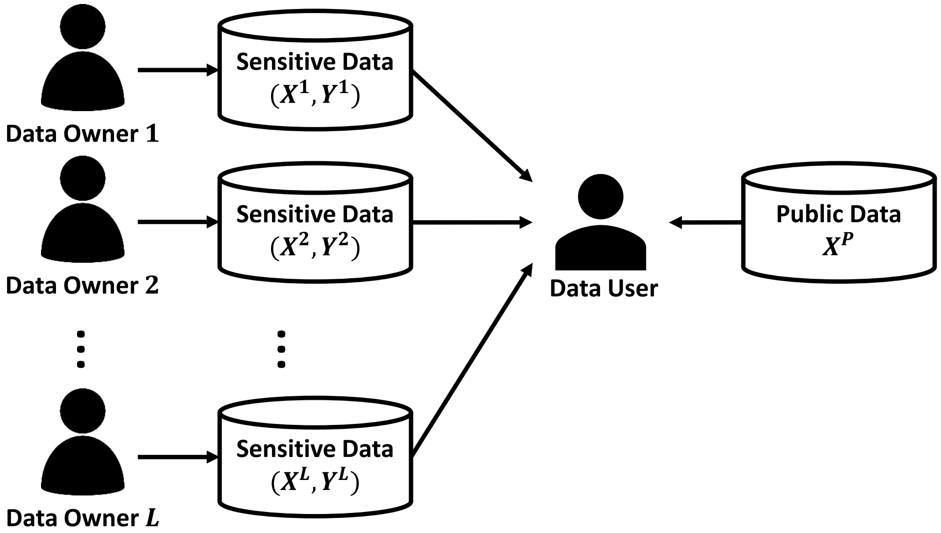


Figure 7.1: Problem overview.

7.3 Methodology

7.3.1 Problem Statement

In this work, we aim to develop a privacy-preserving distributed deep learning framework. As shown in Fig. 7.1, we consider the following problem: Given L data owners, each data owner l holds a set of private samples (X^l, Y^l), where $X^l = \{x_1^l, x_2^l, \ldots, x_{n^l}^l\}$, $Y^l = \{y_1^l, y_2^l, \ldots, y_{n^l}^l\}$, $x_i^l \in \mathbb{R}^d$, and $y_i^l \in \{1, 2, \ldots, k\}$ is the label associated with sample x_i^l, $i = 1, 2, \ldots, n^l$; the untrustworthy data user would like to learn a DNN model with the help of all the data owners, and a public dataset X^P that comes from the same distribution (i.e., the same problem) as the data owners' private datasets, but does not have the label information. In our problem setting, each data owner has two privacy requirements: (i) the value of the individual private data should not be shared to the data user, and (ii) any inference of the individual private data should be prevented from using the intermediate communication messages and the data user's DNN model.

7.3.2 Threat Model

In our problem, we assume (i) the data user is untrustworthy, and (ii) the data owners are honest-but-curious, where each data owner follows the protocol honestly, but try to use the protocol transcripts to extract new information. We assume the value of the individual private data is what

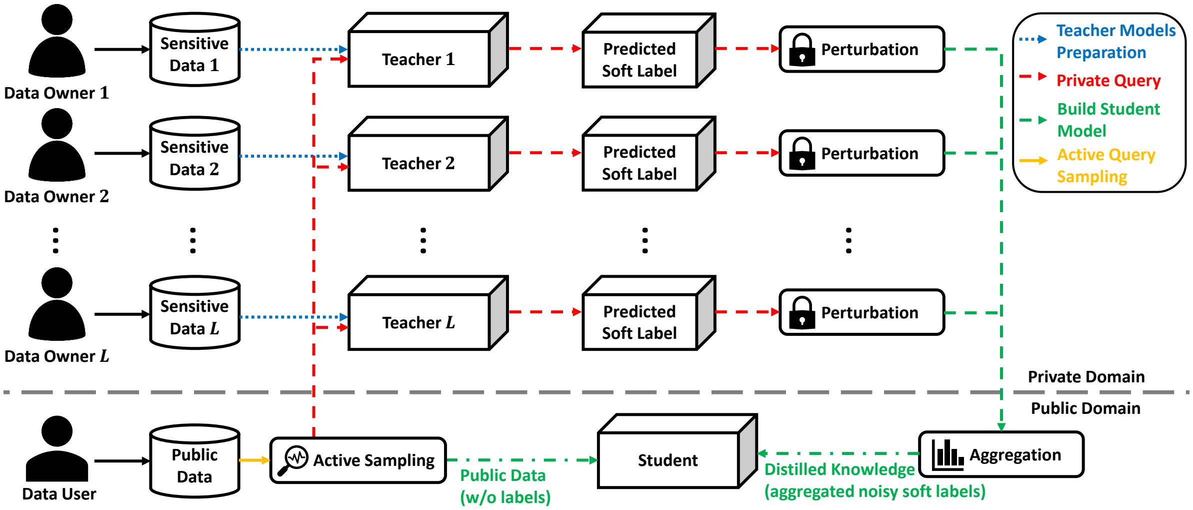


Figure 7.2: The overview of LDP-DL framework.

the adversaries would like to acquire during the whole protocol. Hence, the adversaries could be the data user, the participating data owners or an outside attacker that has the access to the intermediate communication messages or the data user's DNN model. We also assume that the adversaries may have arbitrary background knowledge and might collude with each other. Our work aims to protect the privacy of each data owner's individual private data while providing the reasonable utility to the data user's DNN model. Since we assume that the data user is untrustworthy, it is of the data user's own interest to correctly execute the algorithm or not. However, while using our proposed framework, if the untrustworthy data user behave dishonestly, it will not compromise data owner's privacy, but will only hurt the utility of the data user's DNN model. Furthermore, since the data owners are assumed to be honest-but-curious, the poisoning [169], backdoor [170, 171] or trojans [172] attacks (e.g., the data owner actively and maliciously modify their inputs to influence the performance of the data user's DNN model) are beyond the scope of this work.

7.3.3 Privacy-preserving Distributed Deep Learning

Our proposed privacy-preserving distributed algorithm framework, as shown in Fig. 7.2, consists of four stages that work synergistically between the data owners and the data user. Alg.9 shows the pseudo-code of our algorithm. Firstly, each data owner trains a teacher model using his/her own private dataset (i.e., lines 1-2), and the data user initialize the student models with

Algorithm 9: Locally Differentially Private Distributed Deep Learning (LDP-DL)

```
   Input: $\{(X^1, Y^1), (X^2, Y^2), \dots, (X^L, Y^L)\}$; $X^P$; active query sampling size (per iteration)
          $S$; the number of queries available for each selected public sample $N_Q$; distillation
          learning batch size $S_{db}$.
   Output: Student Model $M_S$.
1  for $l \in \{1, 2, \dots, L\}$ do
2      Data owner $l$ trains a teacher model $M_{T_l}$ using his/her private dataset $(X^l, Y^l)$;
3  Initialize the student model $M_S$ by the data user;
4  while $|X^P| \geq S$ and the accuracy of $M_S$ is not acceptable do
5      $X^Q \longleftarrow ActiveQuerySampling(M_S, X^P, S)$;
6      $X^P \longleftarrow X^P - X^Q$;
7      $(X^{Q'}, Y^{Q'}) = \emptyset$;
8      for $x \in X^Q$ do
9          $z_t \longleftarrow (0, 0, \dots, 0)^{1 \times k}$;
10         Randomly select $N_Q$ data owners from $\{1, 2, \dots, L\}$ to form a subset $O_Q$;
11         for $l \in O_Q$ do
12             $z_t^l \longleftarrow PrivacySanitize(M_{T_l}(x))$;
13             $z_t \longleftarrow z_t + z_t^l$;
14         $z_t \longleftarrow z_t / N_Q$;
15         $(X^{Q'}, Y^{Q'}) \longleftarrow (X^{Q'}, Y^{Q'}) \cup \{(x, z_t)\}$;
16     while $|(X^{Q'}, Y^{Q'})| \geq S_{db}$ do
17         Sample a batch $(X_i^{Q'}, Y_i^{Q'})$ of size $S_{db}$ from $(X^{Q'}, Y^{Q'})$;
18         Compute distill loss $\mathcal{L}_{distill}^i$ over $(X_i^{Q'}, Y_i^{Q'})$ by Eq.;
19         Backpropagate by $\mathcal{L}_{distill}^i$ to update $M_S$;
20 return $M_S$.
```

random or pretrained (i.e., ImageNet [173]) parameters (i.e., line 3). The student model and all the teacher models do not have to use the same DNN architecture. Secondly, in each iteration, the data user efficiently selects a subset of the available public dataset (i.e., lines 5-6), that could better improve the performance of the current student model in the upcoming training, using our well-designed active query sampling approach. The active query sampling component aims to reduce the total number of queries to each teacher model, thus saving the privacy budgets. Thirdly, the data user uses the selected subset of the public data (no labels) to query each of the teacher model from its corresponding selected data owner to obtain the "knoweledge" (data's soft label), and all the query results are sanitised by our local differential privacy techniques before being sent back to the data user (i.e., lines 7-15). Since the data owner might select a huge amount of query samples, it is not realistic to use all the selected samples to query all the data owners, which cost much on the privacy budge and the communication, but might not help a lot for the utility (per our experimental results). Therefore, we predefined a parameter N_Q to control/specify the upper bound of the available data owners for each selected public sample to query (lines 10-11). Last but not least, the data user aggregates the received sanitised query results (i.e., the distilled knowledge) of each data, and leverage the knowledge distillation techniques (using the subset of the public data, and the distilled knowledge) to update/train the student model. The details of the most important three components in our framework (i.e, private query from teacher models, build student model via knowledge transfer and active query sampling) are described in the subsequent sections.

Algorithm 10: Piecewise Mechanism for One-Dimensional Numerical Data (PM-ONE) [22]

```
Input: tuple $z_i \in [-1, 1]$; privacy budget $\epsilon$.
Output: perturbed tuple $z'_i \in [-\Delta, \Delta]$.
$\Delta \longleftarrow \frac{e^{\epsilon/2}+1}{e^{\epsilon/2}-1}$;
$L(z_i) \longleftarrow \frac{\Delta+1}{2} \cdot z_i - \frac{\Delta-1}{2}$;
$R(z_i) \longleftarrow L(z_i) + \Delta - 1$;
Sample value $v$ uniformly at random from $[0, 1]$;
if $v < \frac{e^{\epsilon/2}}{e^{\epsilon/2}+1}$ then
    Sample $z'_i$ uniformly at random from $[L(z_i), R(z_i)]$;
else
    Sample $z'_i$ uniformly at random from $[-\Delta, L(z_i)] \cup [R(z_i), \Delta]$;
return $z'_i$.
```

Algorithm 11: Piecewise Mechanism for Multidimensional Numerical Data (PM) [22]

```
Input: tuple $z \in [-1, 1]^k$; privacy budget $\epsilon$.
Output: perturbed tuple $z' \in [-k \cdot \Delta, k \cdot \Delta]^k$.
$z' \longleftarrow < 0, 0, \ldots, 0 >$;
$m \longleftarrow max\{1, min\{k, \lfloor \frac{\epsilon}{2.5} \rfloor\}\}$;
Sample $m$ values uniformly without replacement from $\{1, 2, \ldots, k\}$;
for each sampled attribute $j$ do
    $z'_j = \frac{k}{m} \cdot PM\text{-}ONE(z_j, \frac{\epsilon}{m})$;
return $z'$.
```

7.3.4 Private Query from Teacher Models

In our proposed algorithm, upon receiving the query data from the data user, each data owner evaluates it using his/her own teacher model and gets the data's soft label. Each data owner perturbs the query data's soft label (using LDP techniques) and then sends the perturbed value to the data user to transfer the distilled knowledge. The data user then aggregates the perturbed query results of each data to obtain the aggregated noisy soft label (i.e., averaged over the query results sent by all the selected data owners) of each data. As such, we could formulate this as a locally differentially private mean estimation problem, where we would like to protect the data owners' private data from the inference attacks given the perturbed query results, and ensure the aggregated noisy soft label as close as the real value. As described in Section 7.2.1, while applying LDP, the adversaries could not distinguish the true value from a perturbed value with a high confidence (adjusted by the parameter ϵ). To protect the privacy of the data owners' private data, the randomization method which satisfies ϵ-LDP is adopted. On the other hand, the performance of the aggregation of the perturbed data could be maintained with an error bound [166, 174], which provides us a way to control the utility of the distilled knowledge. Furthermore, since all the soft labels are multidimensional numerical values, different from the hard labels which are categorical values, we can not directly adopt the encoding-based LDP techniques [166].

To achieve our goal, we adopt the Piecewise Mechanism (PM) [22] that is designed to perturbed the multidimensional numerical values, and has an asymptotic optimal error bound for the mean estimation problem. Alg. 10 shows the PM for one-dimensional numerical data (i.e., PM-ONE). To simplify our explanation, in this section, the value to be perturbed (i.e., the soft labels of k classes) is denoted as $z \in \mathbb{R}^k$, $z = [z_1, z_2, \ldots, z_k]$. PM-ONE (Alg. 10) takes a one-dimentional numerical data $z_i \in [-1, 1]$ as the input, and returns its perturbed value $z'_i \in [-\Delta, \Delta]$, where $\Delta \leftarrow \frac{e^{\epsilon/2}+1}{e^{\epsilon/2}-1}$ is small and thus z'_i has relatively high probability (i.e., $\frac{e^{\epsilon/2}}{e^{\epsilon/2}+1}$) to be close to z_i. As shown in [22], while applying PM-ONE $|\frac{1}{n}\sum_{i=1}^{n} z'_i - \frac{1}{n}\sum_{i=1}^{n} z_i| = O(\frac{\sqrt{log(1/\beta)}}{\epsilon\sqrt{n}})$ with at least $1-\beta$ probability for the task of mean estimation, which is an asymptotically optimal error bound.

Alg. 11 shows the PM for multidimensional numerical data (i.e., PM), where for each data of k dimensions, it randomly selects m (i.e., $m < k$) attributes to perturb. Alg. 11 is designed to reduce the amount of the noise in the task of mean estimation for multidimensional numerical data. While using Alg. 10 to perturb k attributes, each attribute evenly shares a privacy budget of $\frac{\epsilon}{k}$, and the total amount of noise in the mean estimation is $O(\frac{k\sqrt{log(k)}}{\epsilon\sqrt{n}})$, which is super-liner to k. However, it has been shown [22] that while using Alg. 11, $E[max_{j\in[1,k]}|\frac{1}{n}\sum_{i=1}^{n} z'_{i,j} - \frac{1}{n}\sum_{i=1}^{n} z_{i,j}|] =$

$O(\frac{\sqrt{klog(k/\beta)}}{\epsilon\sqrt{n}})$ with at least $1-\beta$ probability for the task of mean estimation, which is still an asymptotically optimal error bound.

7.3.4.1 Privacy Budget Analysis of LDP-DL

As described in Section 7.3.1, in our proposed framework, there are $|X^P|$ public data and L data owners in total. If each public data could query the teacher model for at most N_Q times (Section 7.3.3), each teacher model will be queried for at most $r = \frac{|X^P|\cdot N_Q}{L}$ times by average. Suppose for each private query, the perturbed query result satisfies ϵ_i-LDP. According to the composition property of LDP [175], to meet the requirement of ϵ-LDP for each data owner's private data, we need to satisfy $\sum_i^r \epsilon_i \leq \epsilon$. Since each data owner would participate in the private query for at most r times by average, it requires $\epsilon_i \leq \frac{\epsilon}{r}$. Then, the noise of each query result becomes $O(\frac{r\sqrt{klog(k)}}{\epsilon})$, which is linear to r. Since each public data would be queried for at most N_Q times, the noise of the mean estimation of each public data's soft label (i.e., distilled knowledge) would be $O(\frac{r\sqrt{klog(k)}}{\epsilon\sqrt{N_Q}}) = O(\frac{|X^P|\cdot\sqrt{N_Q}\cdot\sqrt{klog(k)}}{\epsilon\cdot L})$. Since ϵ is the privacy budget that should be controlled by the data owner's preference, and N_Q has the direct influence on the precision of each query result's mean estimation that should be decided on the data user's empirical study, to reduce the overall noise, it is better to increase the number of participant data owners (i.e., L) or decrease the size of the set of public data (i.e., $|X^P|$) utilized for private query (as described in Section 7.3.6).

7.3.5 Build Student Model via Knowledge Transfer

While the data user receiving all the query results from the data owners, our proposed framework uses the knowledge distillation technique to transfer the knowledge learned from the queried teacher models to the student model. Our usage of knowledge distillation is slightly different from the it convectional usage (as described in Section 7.2.2), where (i) we only focus on the knowledge transfer perspective of KD, but not the model compression, thus the student model and all the teacher models could use different and arbitrary DNN architectures; and (ii) in our case, the public dataset does not have the true label information, thus cannot directly use equation (7.3). Hence, in our framework, the student model is trained to minimize the gap between its own predicted soft label and the aggregated soft label from the teacher models, i.e., the knowledge

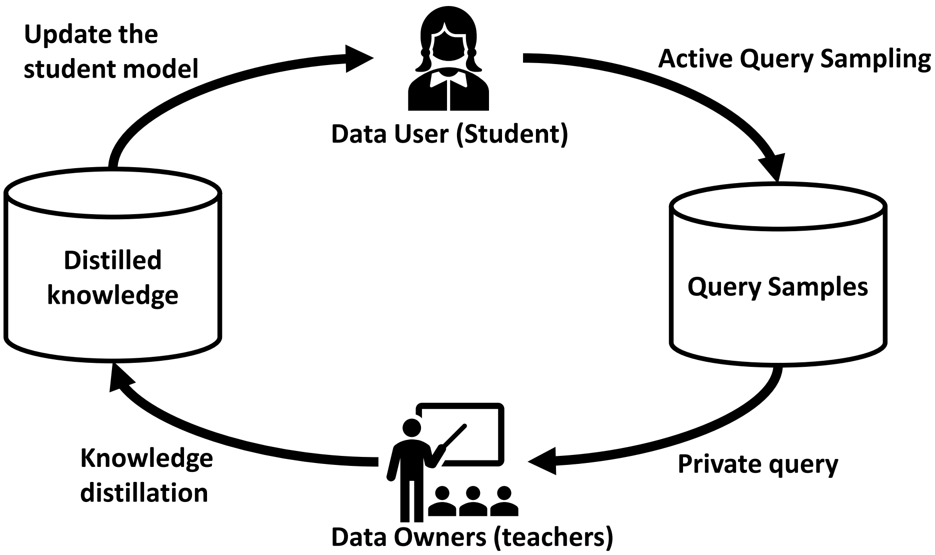


Figure 7.3: Active query sampling.

distillation loss:

$$\begin{aligned}\mathcal{L}(x;\Theta) &= \alpha \cdot \mathcal{H}(\sigma(z_t;T=1),\sigma(z_s;T=1)) \\ &+ \beta \cdot \mathcal{H}(\sigma(z_t;T=\tau),\sigma(z_s;T=\tau))\end{aligned} \tag{7.4}$$

where z_s is the soft label predicted by the student model, z_t is the aggregated soft label from the teacher models, T is the temperature parameter, and $\sigma(z;T) = softmax(z/T)$. The temperature parameter is usually set to 1. While $T > 1$, the probabilities of the classes whose normal values are near zero would be increased. To better distill the knowledge to the student, two temperature values are adopted in our KD loss (i.e., $T = 1$ and $T = \tau > 1$).

7.3.6 Active Query Sampling

As analyzed in Section 7.3.4.1, one direction to reduce the overall noise of the soft label estimation (thus, enhance the overall performance) is to decrease the size of the set of public data (i.e., $|X^P|$) utilized for private query. In this section, we present an active query sampling approach that could actively and adaptively choose samples from the public dataset batch-by-batch to query the teacher models. As shown in Fig. 7.3, we adopt a "least confidence" strategy [176], where in iteration, we attempt to select a set of query samples from the public dataset that the student model shows the “least confidence” at. To be specific, our active query sampling follows the procedure described below:

1. Select an initial subset of S unlabeled public data X^Q uniformly at random from X^P. Update $X^P \leftarrow X^P - X^Q$.

2. Use X^Q to query the teacher models, and use the distilled knowledge to train the student initial student model M_S.

3. For each available public data $x_i \in X_P$, evaluate it on the student model M_S. Let P_{ij} denote the probability of x_i belonging to class $j \in \{1, 2, \ldots, k\}$ predicted by M_S. Let $P_i = \{P_{i1}, P_{i2}, \ldots, P_{ik}\}$, and suppose $\sum_{l=1}^{k} P_{il} = 1$. Let P_i^* be the largest value (posterior probability) in P_i. Then, repeat the procedure below for S times to select S query samples:

$$\begin{aligned} x_i &\leftarrow X^P \\ P_i &\leftarrow M_S(x_i) \\ X^Q &\leftarrow X^Q \cup \underset{x_i}{argmin} \frac{1}{m-1} \sum_{l=1}^{k} (P_i^* - P_{il}) \\ X^P &\leftarrow X^P - X^Q \end{aligned} \tag{7.5}$$

   Then, use X^Q to query the teacher models, and use the distilled knowledge to train the student initial student model M_S.

4. Repeat 3., until the student model meet the performance requirement or no more public data available (i.e., $X_P = \emptyset$).

7.4 Experimental Evaluation

In this section, we evaluate the effectiveness of our proposal method, LDP-DL, on three popular image benchmark datasets (i.e., CIFAR-10 [27], MNIST [28] and Fashion-MNIST [29]) with three basic LDP mechanisms (i.e., Piecewise mechanism [22], Duchi’s mechanism [165] and Laplace mechanism [177]). We also evaluate the performance of our proposed Active Query Sampling (AQS) of our approach. Then, we compare LDP-DL with three state-of-the-art approaches, i.e., DP-SGD [25], PATE [24] and DP-FL [26].

7.4.1 Experiment Environment

All the experiments were conducted on a PC with an Intel Core i9-7980XE processor, 128GB RAM, a Nvidia GeForce GTX 1080Ti graphic card, running 64-bit Ubuntu 18.04 LTS operating system. All the experiments are implemented using Python 3.7.

7.4.2 Experiment Datasets

Three popular benchmark image datasets are utilized to conduct our experimental evaluation:

• CIFAR-10 [27] is a widely used benchmark dataset to evaluate deep learning algorithms. This dataset is a subset of the 80 million tiny images dataset. It contains 60,000 32 x 32 color photographs of objects in 10 different classes, such as frogs, birds, cats, ships, etc. For each class, there are 6,000 images in total, where the testing set includes exactly 1,000 images that randomly selected from each class, and the training set contains the remaining 5,000 images in a random order.

• MNIST [28] is a collection of handwritten digits that is commonly used in the field of image processing and machine learning. This dataset is created by "re-mixing" samples from the NIST dataset. It contains 70,000 28 x 28 grayscale images in 10 different classes, i.e., 10 digits, from 0 to 9. The handwritten digits have been size-normalized and centered in each images. The 70,000 samples have been separated to 60,000 training samples and 10,000 testing samples.

• Fashion-MNIST [29] is a collection of Zalando's article images, which is created as a drop-in (more challenging) replacement for MNIST to better represent modern computer vision tasks. It contains 70,000 28 x 28 gray-scale images in 10 different classes. Each class is a kind of cloth, such as T-shirt, dress, trouser, sneaker, etc. There are 60,000 training samples and 10,000 testing samples.

7.4.3 Experimental Setup

In our experiments, we assume the data owner's teacher models are using ResNet50, and the data user's student model is using ResNet18. For each experiment dataset, we assume each data owner has 4,000 private samples to train his/her teacher model (i.e., ResNet50). The data user would query 200 public samples in each iteration of the Active Query Sampling (AQS) process, and queries 5 iterations in total to train his/her student model (i.e., ResNet18). Each teacher and student model has been trained for 20 epochs with a batch size of 32.

As discussed in Section 7.3.4.1, there are three major parameters that we would like to tune and evaluate in our approach, such as the privacy budget (ϵ), the number of queries of each public sample (N_Q) and the total number of participated data owners (L). We use various combinations of ϵ, N_Q and L to evaluate our approach, where $\epsilon \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, $N_Q \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, and $L \in \{1,000,\ 2,000,\ 3,000,\ 4,000,\ 5,000,\ 6,000,\ 7,000,\ 8,000,\ 9,000,\ 10,000\}$.

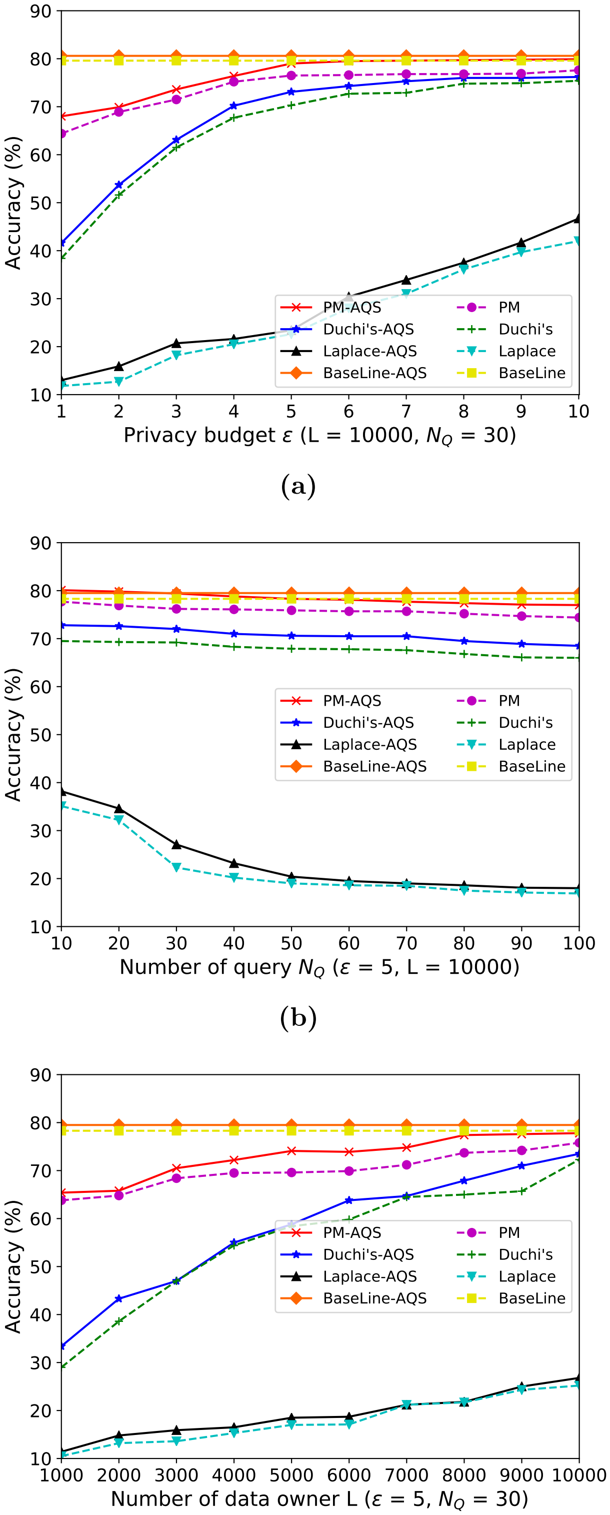
Then, we evaluate the performance of our approach while with and without the Active Query Sampling (AQS) process. Also, we evaluate the performance of our approach's private query using three basic LDP mechanisms, including Piecewise mechanism [22], Duchi's mechanism [165] and Laplace mechanism [177]. All the experiments have been repeated for 10 times and we take the average as the reported results.

7.4.4 Effectiveness Analysis

In this section, we evaluate the effectiveness of our approach on three benchmark image datasets with different parameters and basic LDP mechanisms (as shown in Fig. 7.4, Fig. 7.5 and Fig. 7.6). We can observe that Piecewise mechanism always performs better than Duchi's mechanism and Laplace mechanism in our framework. Moreover, the results with our AQS process consistently outperforms the ones without our AQS process, which demonstrates that our proposed AQS could dramatically save the privacy budget and prevent privacy budget exploding from happening in privacy-preserving distributed deep learning training.

7.4.4.1 Effectiveness Analysis of Different Parameters

The performance of our proposed LDP-DL framework is affected by multiple parameters (ϵ, N_Q and L). To evaluate the effectiveness of single parameter, as shown in Fig. 7.4, Fig. 7.5 and Fig. 7.6, the other parameter are set as constant values. From Fig. 7.4, Fig. 7.5 and Fig. 7.6, we observe that:

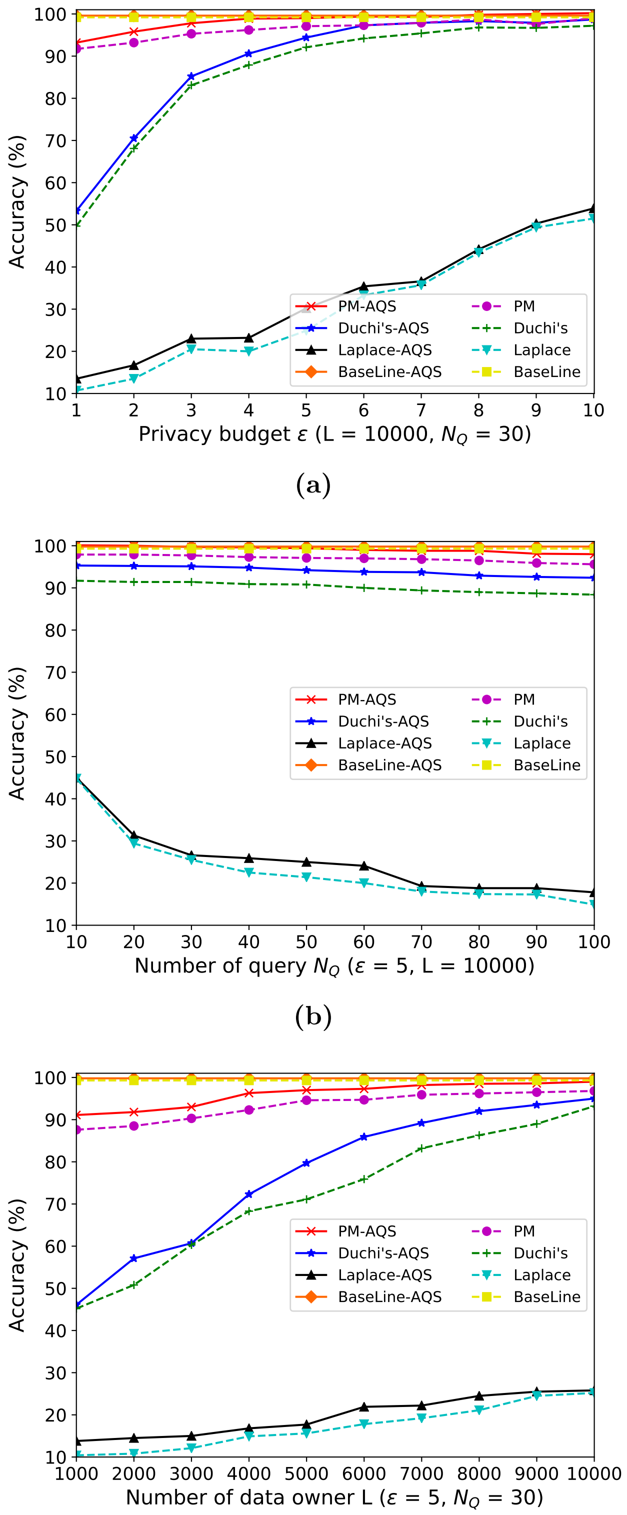


(a)

(b)

(c)

Figure 7.4: LDP-DL experimental results on CIFAR10 dataset.

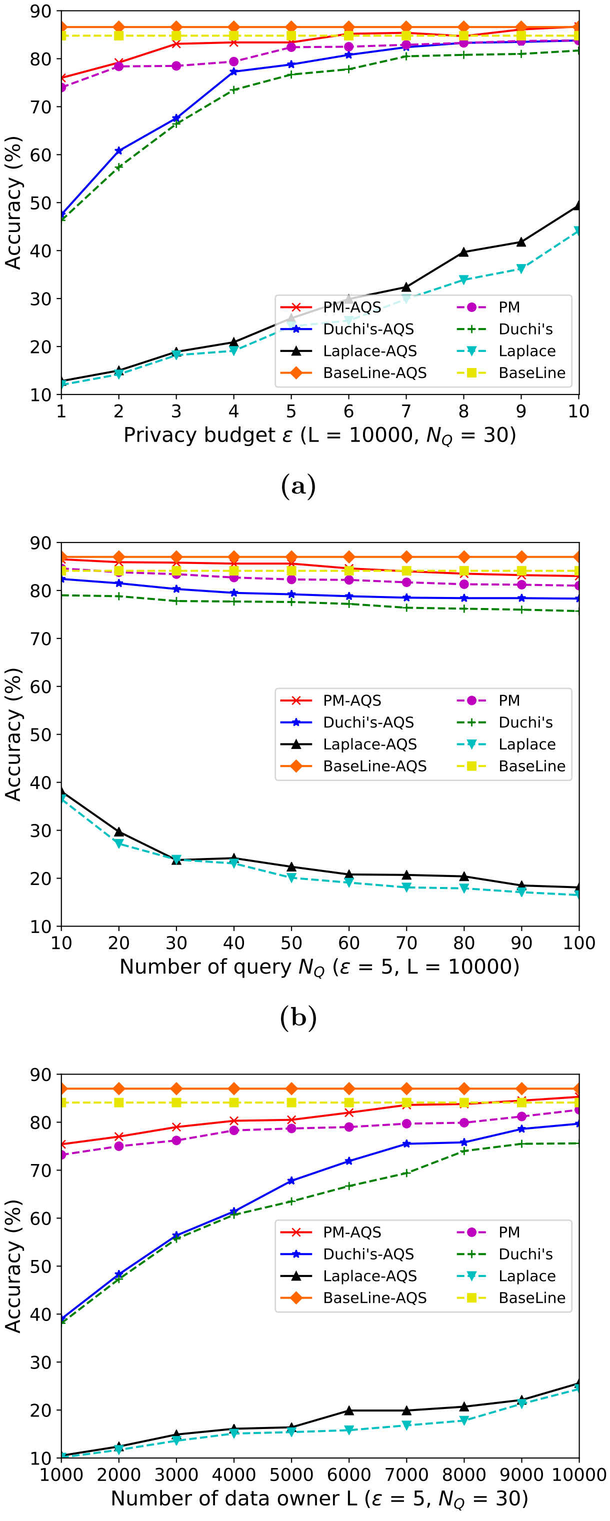


(a)

(b)

(c)

Figure 7.5: LDP-DL experimental results on MNIST dataset.



(a)

(b)

(c)

Figure 7.6: LDP-DL experimental results on FashionMNIST dataset.

- The total privacy budget (ϵ) controls the noise scale of the private queries from each data owner's teacher model. We investigate ϵ from 1 to 10. Fig. 7.4a, Fig. 7.5a and Fig. 7.6a show the impact of the privacy budget on the results of our approach. As the privacy budget ϵ increasing, the accuracy increases, since less noise would be added to the data owners' perturbed distilled knowledge. In LDP mechanisms, greater ϵ results in smaller-scaled noise, and vice versa. While querying from multiple data owners' teacher models, the aggregation of the query results with smaller-scaled noise gives more information towards the data user's student model. Namely, the aggregated value is more close to the actual value, which benefits the training of the data user's student model. As such, a greater ϵ would result in a better accuracy performance.

- The number of queries (N_Q) controls the number of data owners' teacher models to be queried for each unlabelled public data. Fig. 7.4b, Fig. 7.5b and Fig. 7.6b illustrate the results under different number of queries of each public data. As the number of queries increasing, the accuracy decreases, which is inline with our analysis in Section 7.3.4.1. Because increasing the number of queries of each public data will actually result in more noise being added to each query result while maintaining the same total privacy budget. Specifically speaking, as shown in our results, as N_Q increasing, the accuracy of the data user's student model only slightly declines while utilizing either Piecewise mechanism or Duchi's mechanism. However, for the Laplace mechanism, the accuracy significantly decreases while N_Q increasing. Because different LDP mechanisms would result in different noise scale in terms of N_Q. Compared with Laplace mechanism, Using Piecewise mechanism and Duchi's mechanism would decrease the influence of N_Q on the performance of our LDP-DL framework.

- The number of data owners (L) indicates the number of data owners participated in our LDP-DL framework. As analyzed in section 7.3.4.1, increasing the number of participated data owners can reduce the overall noise of the aggregated information. Fig. 7.4c, Fig. 7.5c and Fig. 7.6c show the influence of the number of participated data owners on the performance of our approach. As more data owners participating in our framework, the accuracy tends to increase. Since the total privacy budget is controlled by each data owner's preference, to obtain an appropriate performance of the data user's student model, the number of participated data owners of our LDP-DL framework should be set to a sufficient value.

Table 7.1: In comparison with existing approaches.

Datasets	CIFAR10 [27]		MNIST [28]		FashionMNIST [29]	
Approaches	Accuracy	Privacy Budget	Accuracy	Privacy Budget	Accuracy	Privacy Budget
LDP-DL	77.5%	5	98.1%	5	83.4%	5
	79.7%	8	98.8%	8	85.7%	8
DP-SGD [25]	73.0%	8	97.00%	8	-	-
PATE [24]	73.6%	5	97.7%	5	81.5%	5
	76.0%	8	98.2%	8	84.7%	8
DP-FL [26]	75.9%	5	96.4%	5	82.6%	5
	78.7%	8	97.2%	8	83.6%	8

7.4.5 In Comparison with Existing Approaches

In this section, we have compared our LDP-DL framework with 3 state-of-the-art approaches: DP-SGD [25], PATE [24] and DP-FL [26].

• Differentially Private Stochastic Gradient Descent (DP-SGD) algorithm [25] trains the deep neural network with differential privacy under a centralized setting. It utilizes the Gaussian mechanism on random subset of examples to produce average noisy gradient for model optimization. This approach does not have available code from public resource. Therefore, we directly refer the results presented in the original paper.

• Private Aggregation of Teacher Ensembles (PATE) [24] proposed a distributed teacher-student framework. The privacy guarantee comes from the perturbation on teachers voting aggregation. The ensemble decision based on the noisy voting provides the label of student model's training data. The student model is trained on semi-supervised learning with GANs. PATE [24] are evaluated on the code provided by the paper authors.

• Differentially Private Federated Learning (DL-FL) [26] describes a federated optimization algorithm under private manner. Instead of directly averaging the distributed client models updates, an alter approach that use random sampling and Gaussian mechanism on sum of clients updates is introduced to approximate the averaging. The curator collects the noisy updates to optimize the center server model. Since the original paper aims at protecting the privacy at the client's level but not at the sample's level, DP-FL [26] are evaluated on the code published by the author with some minor changes to enable privacy preservation at the sample's level.

Table 7.1 shows the results of different approaches under the same privacy budgets. The results of LDP-DL are evaluated under parameters $L = 10,000$, $N_Q = 30$. For the other approaches, we strictly follow the settings mentioned in the corresponding papers and keep the common parameters (such as the number of data owners (clients), privacy budgets) at the same level. Under the same level of privacy budgets, we can observe that LDP-DL consistently outperforms the other competitors. The improvement can be derived from the knowledge distillation and active query sampling. Knowledge distillation leverage richer information while transferring the knowledge from the teacher models to a student model. Meanwhile, active query sampling efficiently reduces the total number of queries from the data user (i.e., the student model) to the data owners (i.e., the teacher models). As such, the total cost of privacy budget is been reduced dramatically.

7.5 Related Work

Local Differential Privacy (LDP) has been proposed [165] to remove the trusted curator of the centralized differential privacy. LDP also provides the data owner more controls on the information left hands in a more strict and realistic privacy manner. LDP for statistical information collection and estimation have been well studied in the past decades [178, 179, 166, 180, 181, 182, 183, 184].

Recently, more works propose to apply DP or LDP in data mining and machine learning applications, such as clustering [185], Bayesian inference [186], frequent itemset mining [187] and probability distribution estimation [188, 189, 190, 187]. However, only a few recent works aim to use LDP in deep learning. For instance, Abadi et al. [25] propose to train the deep neural network via stochastic gradient descent with differential privacy under a centralized setting. However, it not only requires an impractical trusted third party to serve as the trusted curator but also has the privacy budget exploding issue (i.e., causing impractical huge privacy budget to train a meaningful deep learning model).

Papernot et al. [24] proposes PATE, a "teacher-student" paradigm for privacy-preserving deep learning, where each data owner learns a teacher model using its own (local) private dataset, and the data user aims to learn a student model using the unlabelled public data (but no direct access to the data owners' private data) to mimic the output of the ensemble of the teacher models, i.e., the student learns to make predictions that is the same as the most number of teachers. To ensure privacy, PATE [24] assumes a trusted aggregator to provide a differentially private query interface, where the data user could query the ensemble of the teacher models (from the data owners) using the unlabelled public data to obtain the labels for the training of the student model. However, a fully trusted aggregator barely exists in most of the real-world distributed deep learning scenarios. Chase et al. [161] proposes a private collaborative neural network learning approach, that combines secure multi-party computation (MPC), differential privacy (DP) and secret sharing. Since the MPC protocol is implemented via a garbled circuit whose size is subject to the number of parameters (i.e., the size of the gradient) of the neural network, it tends to be less efficient and not scalable while training larger neural networks. Also, in [161], using secret sharing requires at least two non-colluding honest data users which might not be practical. In [26], the authors present a federated optimization algorithm under private manner. Instead of directly averaging the distributed client models updates, an alter approach that use random sampling and Gaussian mechanism on sum of clients updates is introduced to approximate the averaging. The curator collects the noisy updates to optimize the center server model. However, this approach only focus on training small deep learning models (i.e., only training one or very few number of iterations) and easier datasets (i.e., not testing on any image datasets).

Our approach aims to solve the challenges left by the previous approaches. The difference could be summarized in three folds: (i) our approach aims to enable training large deep neural networks (e.g., ResNet) on popular benchmark image datasets (e.g., CIFAR-10); (ii) our approach designs a proactive mechanism (i.e., the active query sampling) to reduce the overall privacy budget efficiently to prevent privacy budget exploding while training large deep neural networks; (iii) our approach is not based on federated learning, thus does not have to satisfy the requirements of performing federated learning (e.g., clients being online around the same time period).

7.6 Conclusion

In this chapter, we proposed LDP-DL, a novel, effective and efficient privacy-preserving distributed deep learning framework using local differential privacy and knowledge distillation. We also present an active sampling approach to efficiently reduce the total number of queries from the data user to each data owners, so that to reduce the total cost of privacy budget. In the experimental evaluation, a comprehensive comparison has been made among our algorithm and three state-of-the-art privacy-preserving deep learning approaches. Extensive experiments have been conducted on three benchmark image datasets. Our results show that LDP-DL consistently outperforms the other competitors in terms of privacy budget and model accuracy.

Chapter 8: General Conclusion

In this dissertation, I have devoted into three major projects: (i) Peer-to-peer botnet detection, (ii) community detection in dynamic networks, and (iii) privacy-enhancing technologies for data mining and machine learning. In the first project, I design and implement two two P2P botnet detection systems, PeerHunter and Enhanced PeerHunter. PeerHunter operates under several challenges: (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) no bot-blacklist or “seeds” are available; (d) none statistical traffic patterns known in advance; and (e) do not require to monitor individual host. Enhanced PeerHunter is an extension of PeerHunter that can detect P2P bots, even in the scenario that the botnet traffic are overlapped with legitimate P2P traffic on the same host.

In the project of community detection in dynamic networks, I desngn and implement DynaMo, a novel modularity-based dynamic community detection algorithm, aiming to detect communities in dynamic networks. We also present the theoretical guarantees to show why/how our operations could maximize the modularity, while avoiding redundant and repetitive computations. In the experimental evaluation, a comprehensive comparison has been made among our algorithm, Louvain algorithm and 5 other dynamic algorithms. Extensive experiments have been conducted on 6 real world networks and 10,000 synthetic networks. Our results show that DynaMo outperforms all the other 5 dynamic algorithms in terms of the effectiveness, and is 2 to 5 times (by average) faster than Louvain algorithm.

In the last part of this dissertation, I design and developed three privacy-preserving frameworks, utilizing the concept of dimensionality reduction and differential privacy, including (i) a privacy-preserving facial recognition approach utilizing dimensionality reduction techniques; (ii) a perturbation-based utility-aware privacy-preserving data releasing framework, JUPA; and (iii) a locally differentially private distributed deep learning framework via knowledge distillation. I implement an efficient privacy-preserving facial recognition client server system, FRiPAL, using three dimensionality reduction methods, PCA, LDA and DCA with two types of features. The system performance is evaluated on two Android devices, Nexus 5X and Nexus 6P. The results confirm the efficiency of your system for real life usage. Extensive experiments have been conducted to demonstrate that (i) JUPA outperforms several existing DR methods in terms of providing utility and "somewhat privacy", and provides the flexibility for finding a favorable trade-off or tuning between utility and privacy by tuning the privacy parameters; and (ii) LDP-DL consistently outperforms the other competitors in terms of privacy budget and model accuracy.

References

[1] Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz. Building a scalable system for stealthy p2p-botnet detection. *Information Forensics and Security, IEEE Transactions on*, 9(1):27–38, 2014.

[2] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J Dietrich, and Herbert Bos. Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 97–111. IEEE, 2013.

[3] Huy Hang, Xuetao Wei, Michalis Faloutsos, and Tina Eliassi-Rad. Entelecheia: Detecting p2p botnets in their waiting stage. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[4] Qiben Yan, Yao Zheng, Tingting Jiang, Wenjing Lou, and Y Thomas Hou. Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 316–324. IEEE, 2015.

[5] Di Zhuang and J Morris Chang. Peerhunter: Detecting peer-to-peer botnets through community behavior analysis. In *Dependable and Secure Computing, 2017 IEEE Conference on*, pages 493–500. IEEE, 2017.

[6] Yilin Zhang and Karl Rohe. Understanding regularized spectral clustering via graph conductance. In *Advances in Neural Information Processing Systems*, pages 10654–10663, 2018.

[7] Jintao Meng, Dongmei Fu, and Tao Yang. Semi-supervised soft label propagation based on mass function for community detection. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 1163–1170. IEEE, 2018.

[8] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[9] Fei Hao, Liang Wang, Yifei Sun, and Doo-Soon Park. Detecting (k, r)-clique communities from social networks. In *Advanced Multimedia and Ubiquitous Engineering*, pages 583–590. Springer, 2018.

[10] Despite all its blunders, facebook continues to attract new users, 2019.

[11] 53 incredible facebook statistics and facts, 2019.

[12] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[13] Nam P Nguyen, Thang N Dinh, Ying Xuan, and My T Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *2011 Proceedings IEEE INFOCOM*, pages 2282–2290. IEEE, 2011.

[14] Wen Haw Chong and Loo Nin Teow. An incremental batch technique for community detection. In *Proceedings of the 16th International Conference on Information Fusion*, pages 750–757. IEEE, 2013.

[15] Jiaxing Shang, Lianchen Liu, Feng Xie, Zhen Chen, Jiajia Miao, Xuelin Fang, and Cheng Wu. A real-time detecting algorithm for tracking community structure of dynamic networks. *arXiv preprint arXiv:1407.2683*, 2014.

[16] Jiaxing Shang, Lianchen Liu, Xin Li, Feng Xie, and Cheng Wu. Targeted revision: A learning-based approach for incremental community detection in dynamic networks. *Physica A: Statistical Mechanics and its Applications*, 443:70–85, 2016.

[17] Sun-Yuan Kung. Compressive privacy: From information/estimation theory to machine learning [lecture notes]. *IEEE Signal Processing Magazine*, 34(1):94–112, 2017.

[18] Konstantinos Diamantaras and Sun-Yuan Kung. Data privacy protection by kernel subspace projection and generalized eigenvalue decomposition. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*, pages 1–6. IEEE, 2016.

[19] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[20] D. McCarthy M. Lepinski, D. Levin and R. Watro. Privacy-enhanced android for smart cities applications. In *EAI International Conference on Smart Urban Mobility Services*, 2015.

[21] Sun-Yuan Kung. Discriminant component analysis for privacy protection and visualization of big data. *Multimedia Tools and Applications*, pages 1–36, 2015.

[22] Ning Wang, Xiaokui Xiao, Yin Yang, Jun Zhao, Siu Cheung Hui, Hyejin Shin, Junbum Shin, and Ge Yu. Collecting and analyzing multidimensional data with local differential privacy. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 638–649. IEEE, 2019.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[25] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[26] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[27] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[30] Ping Wang, Baber Aslam, and Cliff C Zou. Peer-to-peer botnets. In *Handbook of Information and Communication Security*, pages 335–350. Springer, 2010.

[31] Xinyuan Wang and Daniel Ramsbrock. The botnet problem. *Computer and Information Security Handbook*, pages 119–132, 2009.

[32] Elizabeth Stinson and John C Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. *2nd USENIX Workshop on Offensive Technologies (WOOT)*, 8:1–9, 2008.

[33] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *USENIX Security Symposium*, volume 5, pages 139–154, 2008.

[34] Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 121–132. IEEE, 2011.

[35] Baris Coskun, Sven Dietrich, and Nasir Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, pages 131–140. ACM, 2010.

[36] Mawi working group traffic archive. http://mawi.wide.ad.jp/mawi/samplepoint-F/2015/201503101400.html. [Online; Accessed: 2015-03-15].

[37] Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi, and Kang Li. Peerrush: Mining for unwanted p2p traffic. *Journal of Information Security and Applications*, 19(3):194–208, 2014.

[38] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix Security Symposium*, volume 7, pages 1–16, 2007.

[39] John Felix, Charles Joseph, and Ali A Ghorbani. Group behavior metrics for p2p botnet detection. In *Information and Communications Security*, pages 93–104. Springer, 2012.

[40] Junjie Zhang, Xiapu Luo, Roberto Perdisci, Guofei Gu, Wenke Lee, and Nick Feamster. Boosting the scalability of botnet detection using adaptive traffic sampling. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 124–134. ACM, 2011.

[41] Ting-Fang Yen and Michael K Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 241–252. IEEE, 2010.

[42] B Soniya and M Wilscy. Fuzzy inference system based on entropy of traffic for bot detection on an endpoint host. In *Data Science & Engineering (ICDSE), 2014 International Conference on*, pages 112–117. IEEE, 2014.

[43] Liyun Li, Suhas Mathur, and Baris Coskun. Gangs of the internet: Towards automatic discovery of peer-to-peer communities. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 64–72. IEEE, 2013.

[44] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix C Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. *First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 8(1):1–9, 2008.

[45] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*, pages 174–180. IEEE, 2011.

[46] Carl Livadas, Robert Walsh, David Lapsley, and W Timothy Strayer. Usilng machine learning technliques to identify botnet traffic. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 967–974. IEEE, 2006.

[47] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[48] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[49] Malware sample sources for researchers. https://zeltser.com/malware-sample-sources/. [Online; Accessed: 2015-03-15].

[50] Argus: Auditing network activity. http://qosient.com/argus/. [Online; Accessed: 2015-09-30].

[51] Di Zhuang and J Morris Chang. Enhanced peerhunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis. *IEEE Transactions on Information Forensics and Security*, 14(6):1485–1500, 2019.

[52] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *USENIX Security Symposium*, pages 95–110, 2010.

[53] Jing Wang and Ioannis Ch Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, 4(2):392–404, 2017.

[54] Sridhar Venkatesan, Massimiliano Albanese, Ankit Shah, Rajesh Ganesan, and Sushil Jajodia. Detecting stealthy botnets in a resource-constrained environment using reinforcement learning. In *Proceedings of the 4th ACM Workshop on Moving Target Defense*, pages 75–85, 2017.

[55] Shankar Karuppayah, Leon Böck, Tim Grube, Selvakumar Manickam, Max Mühlhäuser, and Mathias Fischer. Sensorbuster: On identifying sensor nodes in p2p botnets. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, page 34. ACM, 2017.

[56] Fariba Haddadi and A Nur Zincir-Heywood. Botnet behaviour analysis: How would a data analytics-based system with minimum a priori information perform? *International Journal of Network Management*, 27(4):e1977, 2017.

[57] Wei Lu, Goaletsa Rammidi, and Ali A Ghorbani. Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, 34(3):502–514, 2011.

[58] Shanshan Wang, Qiben Yan, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti. Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security*, 13(5):1096–1109, 2018.

[59] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[60] Xiaobo Ma, Xiaohong Guan, Jing Tao, Qinghua Zheng, Yun Guo, Lu Liu, and Shuang Zhao. A novel irc botnet detection method based on packet size sequence. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[61] Sara Khanchi, Ali Vahdat, Malcolm I Heywood, and A Nur Zincir-Heywood. On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm and evolutionary computation*, 39:123–140, 2018.

[62] Jiang Jianguo, Biao Qi, Shi Zhixin, Yan Wang, and Bin Lv. Botnet detection method analysis on the effect of feature extraction. In *Trustcom, 2016 IEEE*, pages 1882–1888. IEEE, 2016.

[63] Xiaobo Ma, Junjie Zhang, Jing Tao, Jianfeng Li, Jue Tian, and Xiaohong Guan. Dnsradar: Outsourcing malicious domain detection based on distributed cache-footprints. *IEEE Transactions on Information Forensics and Security*, 9(11):1906–1921, 2014.

[64] Weikeng Chen, Xiao Luo, and A Nur Zincir-Heywood. Exploring a service-based normal behaviour profiling system for botnet detection. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 947–952. IEEE, 2017.

[65] Hung-Shen Wu, Nen-Fu Huang, and Guan-Hao Lin. Identifying the use of data/voice/video-based p2p traffic by dns-query behavior. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009.

[66] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[67] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, et al. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134. ACM, 2004.

[68] Barracuda reputation block list (brbl), 2018.

[69] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, pages 268–273. IEEE, 2009.

[70] Massimiliano Albanese, Sushil Jajodia, and Sridhar Venkatesan. Defending from stealthy botnets using moving target defenses. *IEEE Security & Privacy*, 16(1):92–97, 2018.

[71] Shuang Zhao, Patrick PC Lee, John Lui, Xiaohong Guan, Xiaobo Ma, and Jing Tao. Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device messaging service. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 119–128. ACM, 2012.

[72] Di Zhuang, Morris J Chang, and Mingchen Li. Dynamo: Dynamic community detection by incrementally maximizing modularity. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[73] Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory social network analysis with Pajek: Revised and expanded edition for updated software*, volume 46. Cambridge University Press, 2018.

[74] Liang Zhou, Dan Wu, Zhenjiang Dong, and Xuelong Li. When collaboration hugs intelligence: Content delivery over ultra-dense networks. *IEEE Communications Magazine*, 55(12):91–95, 2017.

[75] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983. ACM, 2018.

[76] Kun He, Yingru Li, Sucheta Soundarajan, and John E Hopcroft. Hidden community detection in social networks. *Information Sciences*, 425:92–106, 2018.

[77] Zeineb Dhouioui and Jalel Akaichi. Tracking dynamic community evolution in social networks. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 764–770. IEEE, 2014.

[78] Nagehan İlhan and Şule Gündüz Öğüdücü. Predicting community evolution based on time series modeling. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1509–1516. ACM, 2015.

[79] Yu Xin, Zhi-Qiang Xie, and Jing Yang. An adaptive random walk sampling method on dynamic community detection. *Expert Systems with Applications*, 58:10–19, 2016.

[80] Prerna Agarwal, Richa Verma, Ayush Agarwal, and Tanmoy Chakraborty. Dyperm: Maximizing permanence for dynamic community detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 437–449. Springer, 2018.

[81] Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, and Fosca Giannotti. Tiles: an online algorithm for community discovery in dynamic social networks. *Machine Learning*, 106(8):1213–1241, 2017.

[82] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L Tseng. Analyzing communities and their evolutions in dynamic social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(2):8, 2009.

[83] Yu-Ru Lin, Jimeng Sun, Hari Sundaram, Aisling Kelliher, Paul Castro, and Ravi Konuru. Community discovery via metagraph factorization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(3):17, 2011.

[84] Derek Greene, Donal Doyle, and Padraig Cunningham. Tracking the evolution of communities in dynamic social networks. In *2010 international conference on advances in social networks analysis and mining*, pages 176–183. IEEE, 2010.

[85] Lei Tang, Huan Liu, and Jianping Zhang. Identifying evolving groups in dynamic multimode networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(1):72–85, 2012.

[86] Chonghui Guo, Jiajia Wang, and Zhen Zhang. Evolutionary community structure discovery in dynamic weighted networks. *Physica A: Statistical Mechanics and its Applications*, 413:565–576, 2014.

[87] Anita Zakrzewska and David A Bader. A dynamic algorithm for local community detection in graphs. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 559–564. ACM, 2015.

[88] Xiaoke Ma and Di Dong. Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks. *IEEE transactions on knowledge and data engineering*, 29(5):1045–1058, 2017.

[89] Mário Cordeiro, Rui Portocarrero Sarmento, and João Gama. Dynamic community detection in evolving networks using locality modularity optimization. *Social Network Analysis and Mining*, 6(1):15, 2016.

[90] Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):35, 2018.

[91] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[92] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[93] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[94] Nam P Nguyen, Thang N Dinh, Yilin Shen, and My T Thai. Dynamic social community detection and its applications. *PloS one*, 9(4):e91431, 2014.

[95] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[96] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Soc., 2013.

[97] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[98] Alan Mislove, Hema Swetha Koppula, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the flickr social network. In *Proceedings of the first workshop on Online social networks*, pages 25–30. ACM, 2008.

[99] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[100] Giulio Rossetti. Rdyn: graph benchmark handling community dynamics. *Journal of Complex Networks*, 5(6):893–912, 2017.

[101] Di Zhuang, Sen Wang, and J Morris Chang. Fripal: Face recognition in privacy abstraction layer. In *2017 IEEE Conference on Dependable and Secure Computing*, pages 441–448. IEEE, 2017.

[102] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[103] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[104] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Cryptographically private support vector machines. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–624. ACM, 2006.

[105] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.

[106] Kun Liu, Hillol Kargupta, and Jessica Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 18(1):92–106, 2006.

[107] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[108] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[109] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[110] Rabbitmq - rabbit message queue, March 2016.

[111] Amqp: Advanced message queuing protocol, March 2016.

[112] California institute of technology. faces 1999 (front), March 2016.

[113] Ara Nefian and Monson H Hayes III. *A hidden Markov model-based approach for face detection and recognition.* PhD thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, 1999.

[114] Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):643–660, 2001.

[115] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM Sigmod Record*, volume 29, pages 439–450. ACM, 2000.

[116] Keke Chen and Ling Liu. Privacy preserving data classification with rotation perturbation. In *Data Mining, Fifth IEEE International Conference on*, pages 4–pp. IEEE, 2005.

[117] Zhengli Huang, Wenliang Du, and Biao Chen. Deriving private information from randomized data. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 37–48. ACM, 2005.

[118] Kun Liu, Chris Giannella, and Hillol Kargupta. An attacker's view of distance preserving maps for privacy preserving data mining. In *Knowledge Discovery in Databases: PKDD 2006*, pages 297–308. Springer, 2006.

[119] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, pages 235–253. Springer, 2009.

[120] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology–ICISC 2009*, pages 229–244. Springer, 2009.

[121] Facebook Cambridge Analytica data scandal. Facebook - Cambridge Analytica data scandal. http://www.bbc.com/news/technology-43649018, 2018. [Online; accessed 5-April-2018].

[122] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[123] J Kim and W Winkler. Multiplicative noise for masking continuous data. *Statistics*, page 01, 2003.

[124] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: regression analysis under differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1364–1375, 2012.

[125] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[126] Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618. ACM, 2017.

[127] Mert Al, Shibiao Wan, and Sun-Yuan Kung. Ratio utility and cost analysis for privacy preserving subspace projection. *arXiv preprint arXiv:1702.07976*, 2017.

[128] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[129] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[130] Jacob R Gardner, Paul Upchurch, Matt J Kusner, Yixuan Li, Kilian Q Weinberger, Kavita Bala, and John E Hopcroft. Deep manifold traversal: Changing labels with convolutional features. *arXiv preprint arXiv:1511.06421*, 2015.

[131] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[132] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[133] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[134] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[135] Robert Fortet and E Mourier. Convergence de la répartition empirique vers la répartition théorique. In *Annales scientifiques de l'École Normale Supérieure*, volume 70, pages 267–285. Elsevier, 1953.

[136] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE transactions on information forensics and security*, 7(3):1053–1066, 2012.

[137] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[138] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[139] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[140] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems*, pages 2339–2347, 2012.

[141] Xiaoqian Jiang, Zhanglong Ji, Shuang Wang, Noman Mohammed, Samuel Cheng, and Lucila Ohno-Machado. Differential-private data publishing through component analysis. *Transactions on data privacy*, 6(1):19, 2013.

[142] Vincent Bindschaedler, Reza Shokri, and Carl A Gunter. Plausible deniability for privacy-preserving data synthesis. *Proceedings of the VLDB Endowment*, 10(5):481–492, 2017.

[143] Jordi Soria-Comas and Josep Domingo-Ferrer. Differentially private data sets based on microaggregation and record perturbation. In *Modeling Decisions for Artificial Intelligence*, pages 119–131. Springer, 2017.

[144] Mark Andrew Hall. Correlation-based feature selection for machine learning. 1999.

[145] Josep Domingo-Ferrer and Vicenç Torra. Ordinal, continuous and heterogeneous k-anonymity through microaggregation. *Data Mining and Knowledge Discovery*, 11(2):195–212, 2005.

[146] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[147] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop.*, pages 41–48. IEEE, 1999.

[148] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007.

[149] Tony Jebara. Multi-task feature and kernel selection for svms. In *Proceedings of the twenty-first international conference on Machine learning*, page 55. ACM, 2004.

[150] Seung-Jean Kim, Alessandro Magnani, and Stephen Boyd. Optimal kernel selection in kernel fisher discriminant analysis. In *Proceedings of the 23rd international conference on Machine learning*, pages 465–472. ACM, 2006.

[151] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications.* Cambridge university press, 2009.

[152] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013.

[153] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *KDD*, volume 96, pages 202–207. Citeseer, 1996.

[154] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[155] Fábio Perez, Sandra Avila, and Eduardo Valle. Solo or ensemble? choosing a cnn architecture for melanoma classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[156] Pei-Yuan Wu, Chi-Chen Fang, Jien Morris Chang, and Sun-Yuan Kung. Cost-effective kernel ridge regression implementation for keystroke-based active authentication system. *IEEE transactions on cybernetics*, 47(11):3916–3927, 2016.

[157] Hung Nguyen, Di Zhuang, Pei-Yuan Wu, and Morris Chang. Autogan-based dimension reduction for privacy preservation. *Neurocomputing*, 2019.

[158] Di Zhuang and J Morris Chang. Enhanced peerhunter: Detecting peer-to-peer botnets through network-flow level community behavior analysis. *IEEE Transactions on Information Forensics and Security*, 14(6):1485–1500, 2018.

[159] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753, May 2019.

[160] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[161] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E Lauter, and Peter Rindal. Private collaborative neural network learning. *IACR Cryptology ePrint Archive*, 2017:762, 2017.

[162] Dwork Cynthia. Differential privacy. *Automata, languages and programming*, pages 1–12, 2006.

[163] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[164] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze gauss: optimal bounds for privacy-preserving principal component analysis. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 11–20, 2014.

[165] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.

[166] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 729–745, 2017.

[167] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[168] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[169] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[170] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[171] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[172] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017.

[173] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[174] John C Duchi, Michael I Jordan, and Martin J Wainwright. Minimax optimal procedures for locally private estimation. *Journal of the American Statistical Association*, 113(521):182–201, 2018.

[175] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103. IEEE, 2007.

[176] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[177] Yin Yang, Zhenjie Zhang, Gerome Miklau, Marianne Winslett, and Xiaokui Xiao. Differential privacy in data publication and analysis. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 601–606, 2012.

[178] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[179] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. *arXiv preprint arXiv:1407.1338*, 2014.

[180] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 127–135, 2015.

[181] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[182] Graham Cormode, Somesh Jha, Tejas Kulkarni, Ninghui Li, Divesh Srivastava, and Tianhao Wang. Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1655–1658, 2018.

[183] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. *arXiv preprint arXiv:1712.01524*, 2017.

[184] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[185] Kobbi Nissim and Uri Stemmer. Clustering algorithms for the centralized and local models. In *Algorithmic Learning Theory*, pages 619–653. PMLR, 2018.

[186] Emre Yilmaz, Mohammad Al-Rubaie, and J Morris Chang. Locally differentially private naive bayes classification. *arXiv preprint arXiv:1905.01039*, 2019.

[187] Tianhao Wang, Ninghui Li, and Somesh Jha. Locally differentially private frequent itemset mining. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 127–143. IEEE, 2018.

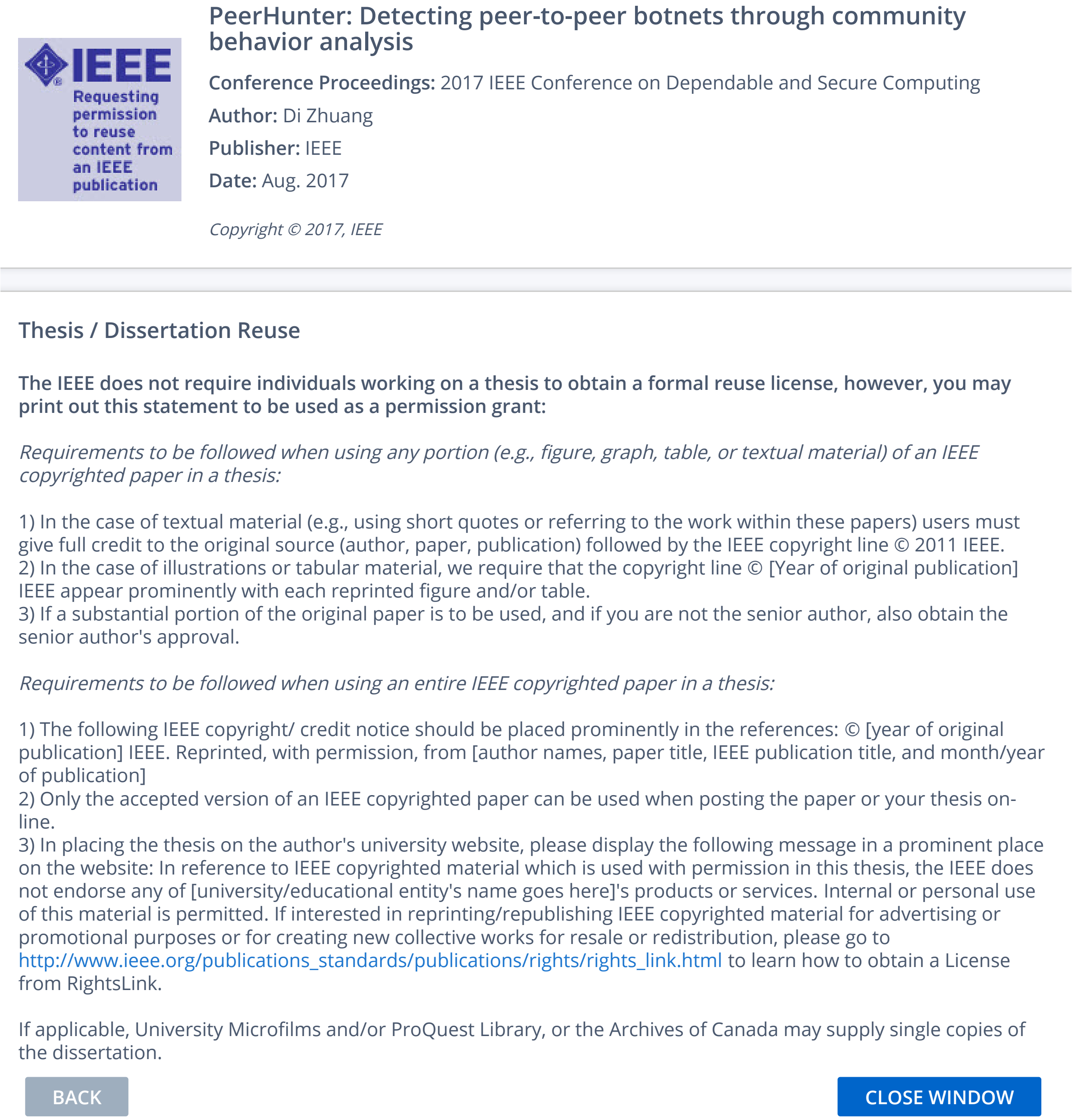
[188] Peter Kairouz, Keith Bonawitz, and Daniel Ramage. Discrete distribution estimation under local privacy. In *International Conference on Machine Learning*, pages 2436–2444. PMLR, 2016.

[189] Takao Murakami, Hideitsu Hino, and Jun Sakuma. Toward distribution estimation under local differential privacy with small samples. *Proceedings on Privacy Enhancing Technologies*, 2018(3):84–104, 2018.

[190] Min Ye and Alexander Barg. Optimal schemes for discrete distribution estimation under locally differential privacy. *IEEE Transactions on Information Theory*, 64(8):5662–5676, 2018.

Appendix A: Copyright Permissions

The permission below is for the use of the content in Chapter 2.


IEEE
Requesting permission to reuse content from an IEEE publication

PeerHunter: Detecting peer-to-peer botnets through community behavior analysis

Conference Proceedings: 2017 IEEE Conference on Dependable and Secure Computing

Author: Di Zhuang

Publisher: IEEE

Date: Aug. 2017

Copyright © 2017, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

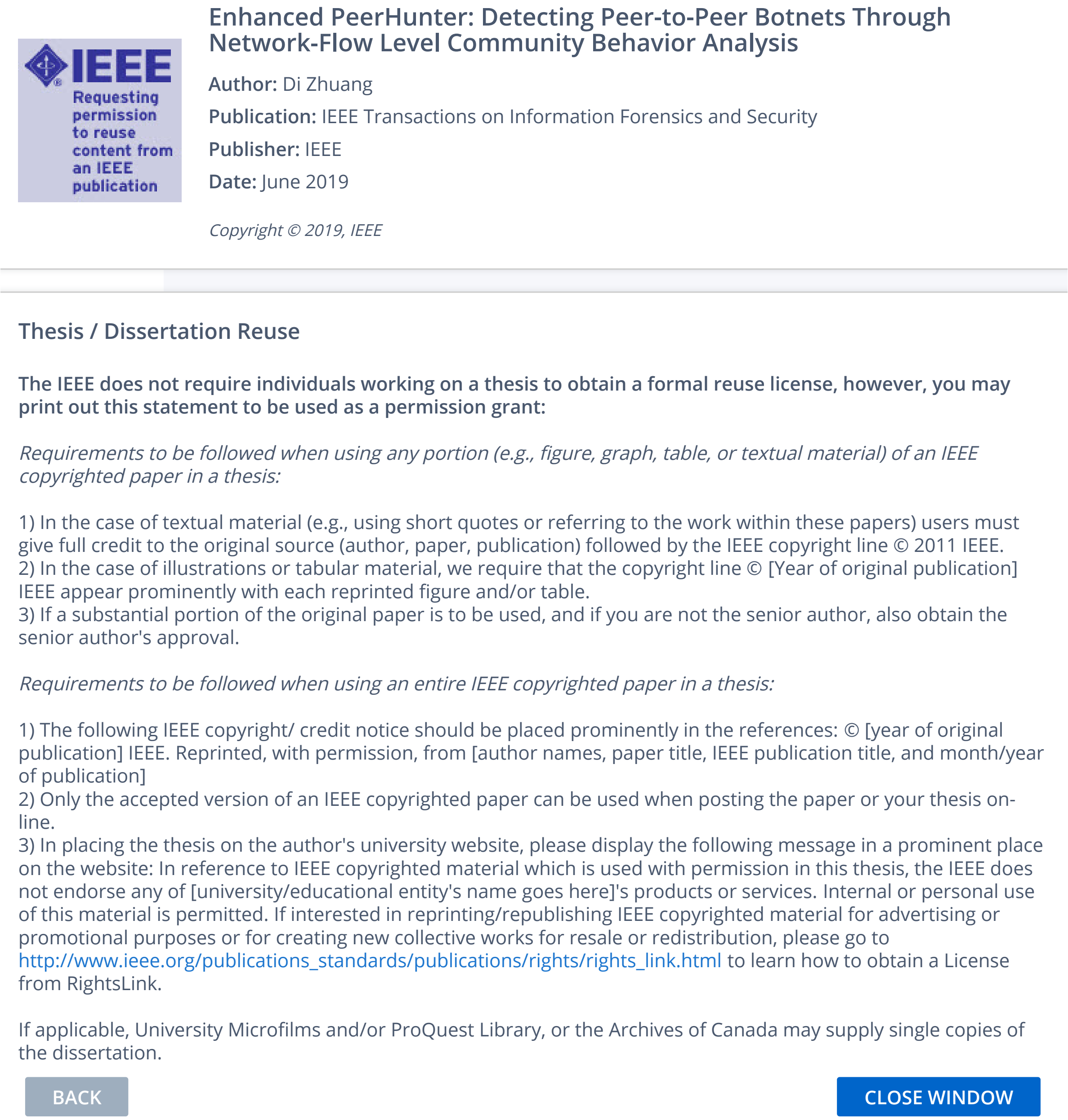
1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The permission below is for the use of the content in Chapter 3.


IEEE
Requesting permission to reuse content from an IEEE publication
BACK
CLOSE WINDOW

Enhanced PeerHunter: Detecting Peer-to-Peer Botnets Through Network-Flow Level Community Behavior Analysis

Author: Di Zhuang

Publication: IEEE Transactions on Information Forensics and Security

Publisher: IEEE

Date: June 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

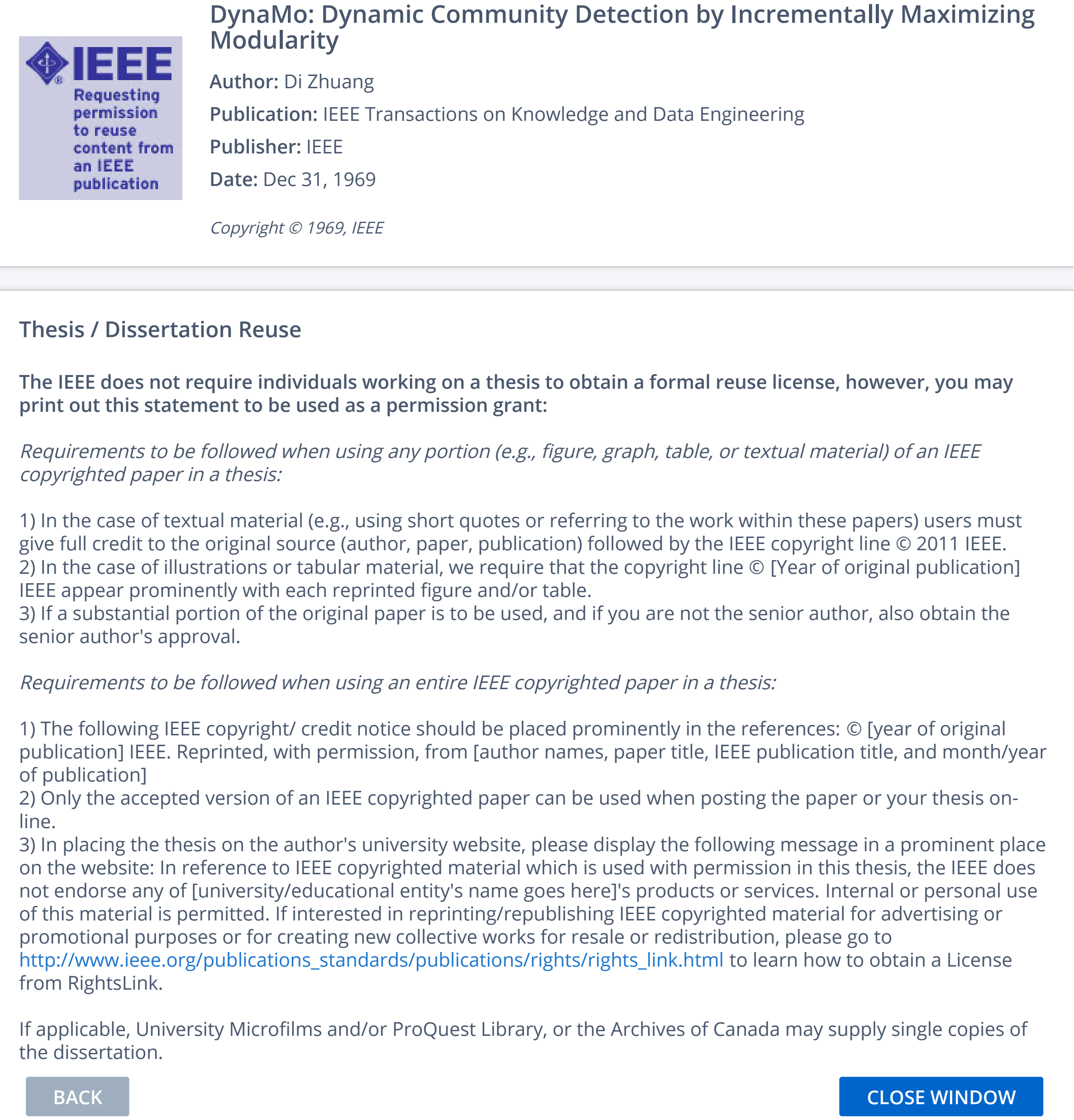
1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The permission below is for the use of the content in Chapter 4.


IEEE
Requesting permission to reuse content from an IEEE publication

DynaMo: Dynamic Community Detection by Incrementally Maximizing Modularity

Author: Di Zhuang

Publication: IEEE Transactions on Knowledge and Data Engineering

Publisher: IEEE

Date: Dec 31, 1969

Copyright © 1969, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

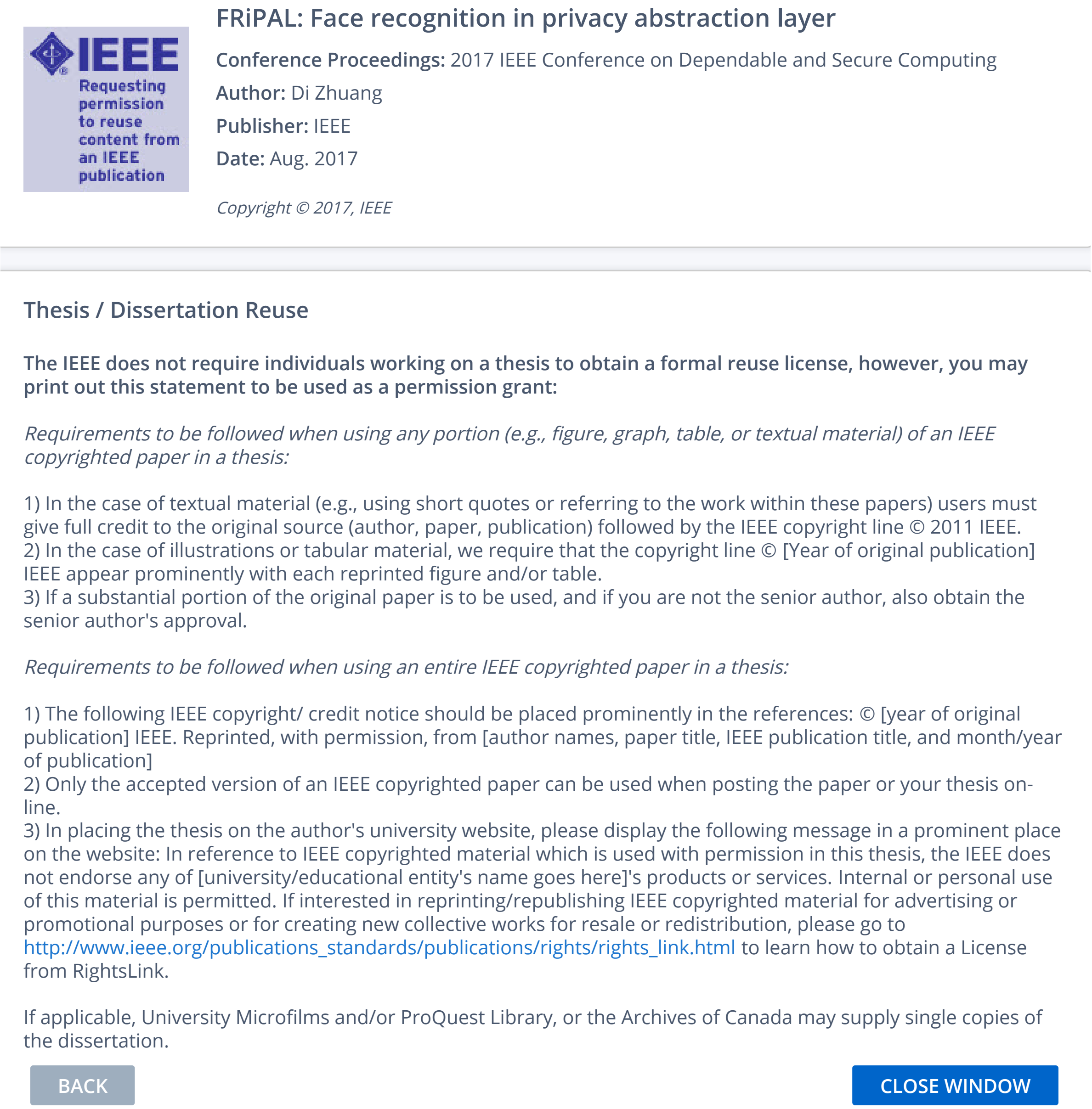
1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis online.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

The permission below is for the use of the content in Chapter 5.



IEEE
Requesting permission to reuse content from an IEEE publication

FRiPAL: Face recognition in privacy abstraction layer

Conference Proceedings: 2017 IEEE Conference on Dependable and Secure Computing

Author: Di Zhuang

Publisher: IEEE

Date: Aug. 2017

Copyright © 2017, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.