# ENHANCING CYBERSECURITY WITH ENCRYPTED TRAFFIC FINGERPRINTING

by

Khaled Mohammed Al-Naami

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Latifur Khan, Co-Chair

Dr. Kevin W. Hamlen, Co-Chair

Dr. Bhavani Thuraisingham

Dr. Zhiqiang Lin

Dr. Farokh B. Bastani

*“And Allah has brought you forth from the wombs of your mothers not knowing a thing,*

*and He made for you hearing and vision and intellect that perhaps you would be grateful.”*

*The Qur’an, An-Nahl 16:78*

*To The Creator, The Almighty, The All Knowing, The Omniscient.*

*To Allah.*

# ENHANCING CYBERSECURITY WITH ENCRYPTED TRAFFIC FINGERPRINTING

by

KHALED MOHAMMED AL-NAAMI, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2017

# ACKNOWLEDGMENTS

All Praise is due to Allah. We praise Him and we seek help from Him. First and foremost, I thank Him for giving me the ability to fulfill my wish to earn the Doctoral degree.

I would like to express my warmest appreciation to Dr. Latifur Khan for being a great advisor and for his support and patience. You have set an example of excellence as a mentor and role model. I also wish to extend my warmest appreciation to my wonderful co-advisor Dr. Kevin W. Hamlen, for his support and constant enthusiasm and encouragement. Besides my advisors, I would like to thank my dissertation committee, Dr. Bhavani Thuraisingham, whose kindness and help are greatly appreciated, and Dr. Zhiqiang Lin for his support and useful feedback, and Dr. Farokh B. Bastani for his encouragement, support, and useful comments.

I am very grateful to my parents. My father, you have always wanted to celebrate this moment with me. May Allah bless you in your grave. Although you passed away a few months ago, you have been always in my heart while persevering to achieve your dream of me getting my Ph.D. Well, here I am doing it. My mother, the passion and support you have provided me over the years were the greatest gifts anyone has ever given me. Thank you.

I would like to extend my deepest love and appreciation to my wife, Amal, and my children, Maryam, Zahraa, and Hamza, whose sacrifice during this journey is invaluable.

I also need to thank my family at large and my siblings, Fahmi, Iqbal, Raidan, Ammar, and Ibrahim for their support. To all my friends and labmates in the school and everywhere, thank you for being there and contributing to this dissertation. I am very thankful to each one of you.

This research was supported in part by AFOSR awards #FA9550-09-1-0468, #FA9550-12-1-0077, and #FA9550-14-1-0173, NSF awards #1054629 and CNS 1229652, and NSA award

#H98230-15-1-0271. Any opinions, recommendations, or conclusions expressed are those of the author and not necessarily of the AFOSR, NSF, or NSA.

October 2017

# ENHANCING CYBERSECURITY WITH ENCRYPTED TRAFFIC FINGERPRINTING

Khaled Mohammed Al-Naami, PhD
The University of Texas at Dallas, 2017

Supervising Professors: Dr. Latifur Khan, Co-Chair
Dr. Kevin W. Hamlen, Co-Chair

Recently, network traffic analysis and cyber deception have been increasingly used in various applications to protect people, information, and systems from major cyber threats. Network traffic fingerprinting is a traffic analysis attack which threatens web navigation privacy. It is a set of techniques used to discover patterns from a sequence of network packets generated while a user accesses different websites. Internet users (such as online activists or journalists) may wish to hide their identity and online activity to protect their privacy. Typically, an anonymity network is utilized for this purpose. These anonymity networks such as Tor (The Onion Router) provide layers of data encryption which poses a challenge to the traffic analysis techniques.

Traffic fingerprinting studies have employed various traffic analysis and statistical techniques over anonymity networks. Most studies use a similar set of features including packet size, packet direction, total count of packets, and other summaries of different packets. Moreover, various defense mechanisms have been proposed to counteract these feature selection processes, thereby reducing prediction accuracy.

In this dissertation, we address the aforementioned challenges and present a novel method to extract characteristics from encrypted traffic by utilizing data dependencies that occur over sequential transmissions of network packets. In addition, we explore the temporal nature of encrypted traffic and introduce an adaptive model that considers changes in data content over time. We not only consider traditional learning techniques for prediction, but also use semantic vector space models (VSMs) of language where each word (packet) is represented as a real-valued vector. We also introduce a novel defense algorithm to counter the traffic fingerprinting attack. The defense uses sampling and mathematical optimization techniques to morph packet sequences and destroy traffic flow dependency patterns.

Cyber deception has been shown to be a key ingredient in cyber warfare. Cyber security deception is the methodology followed by an organization to lure the adversary into a controlled and transparent environment for the purpose of protecting the organization, disinforming the attacker, and discovering zero-day threats. We extend our traffic fingerprinting work to the cyber deception domain and leverage recent advances in software deception to enhance Intrusion Detection Systems by feeding back attack traces into machine learning classifiers. We present a feature-rich attack classification approach to extract security-relevant network- and system-level characteristics from production servers hosting enterprise web applications.

# TABLE OF CONTENTS

# LIST OF FIGURES

## LIST OF TABLES

# CHAPTER 1

# INTRODUCTION [1] [2] [3] [4]

With a tremendous growth in the number of Internet users over the past decade, network *traffic analysis* has gained significant interest in both academia and industry. Applications such as personalized marketing [72] and traffic engineering [115, 114] have spurred the demand for tracking online activities of users [80]. For example, by tracking the websites accessed by a particular user, related products may be advertised. Unfortunately, online users have fallen victim to adversaries who use such tracking mechanisms for malicious activities by passively monitoring network traffic. As a result, encryption technologies such as SSL/TLS are used extensively to hide data in network traffic from unauthorized access.

In addition to data encryption, end-node network identifiers (e.g., IP addresses) may also be hidden from external adversaries using technologies such as Tor [51], to anonymize the user.

[1] This chapter contains material previously published as: K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K.W. Hamlen, and B. Thuraisingham. “Adaptive encrypted traffic fingerprinting with bi-directional dependence.” In Proceedings of the 32nd Annual Computer Security Applications Conference, pp. 177-188. ACM, 2016. DOI: https://doi.org/10.1145/2991079.2991123. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[2] This chapter contains material previously published as: ©2015 IEEE. Portions Reprinted, with permission, from K. Al-Naami, G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan and B. Thuraisingham. “P2V: Effective Website Fingerprinting Using Vector Space Representations.” IEEE Symposium Series on Computational Intelligence, December 2015. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[3] Some of the work presented in this chapter was performed in collaboration with L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Lead author Al-Naami conducted the majority of the research, including most of the design, the full implementation, and the full evaluation.

[4] Some of the work presented in this chapter was performed in collaboration with F. Araujo, A. Gbadebo, A. Mustafa, L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Al-Naami led the machine learning half of the research, including feature generation, classification design, implementation, and experiments.

Recent studies [35, 137] on traffic analysis have focused on identifying characteristic patterns in network traffic that reveal the behavior of an end-node, thereby de-anonymizing the network. Essentially, pattern recognition techniques are employed over features extracted from encrypted network traffic passively captured at the user's end. This behavior identification process of an end-node (i.e., either a service accessed by the user, or an application at the user's end involved in the network traffic) is called *Traffic Fingerprinting*.

## 1.1 Network Traffic Fingerprinting

We focus on the following two applications (illustrated in Figure 1.1) whose primary goal is to perform traffic fingerprinting to identify an end-node generating encrypted traffic. Here, a man-in-the-middle (i.e., network administrator, ISP, government agency, etc.) captures encrypted network traffic passively at the user's end.

**Website Fingerprinting**. This application involves identifying the webpage (end-node) accessed by a user who actively hides online activities using an anonymity network such as Tor. Knowledge of the user's online activities may be useful in applications such as targeted advertisements, tracking terrorist activities, checking for DRM violations, etc. On the contrary, it violates the user's online privacy. Destination IP addresses obtained from encrypted traffic in this setting cannot be used for webpage identification since they would be encapsulated by the encryption scheme. Fingerprinting over such encrypted data for identification of webpage (or website) is widely known as *Website Fingerprinting* [92]. We denote this as WFIN.

**App Fingerprinting**. Unlike websites, smartphone apps access the Internet by connecting to remote services that provide necessary data for their operation. Examples of such services include advertisements, 3rd party libraries, and other API-based services. Applications, such as ad relevance, network bandwidth management, and app recommendations, may require

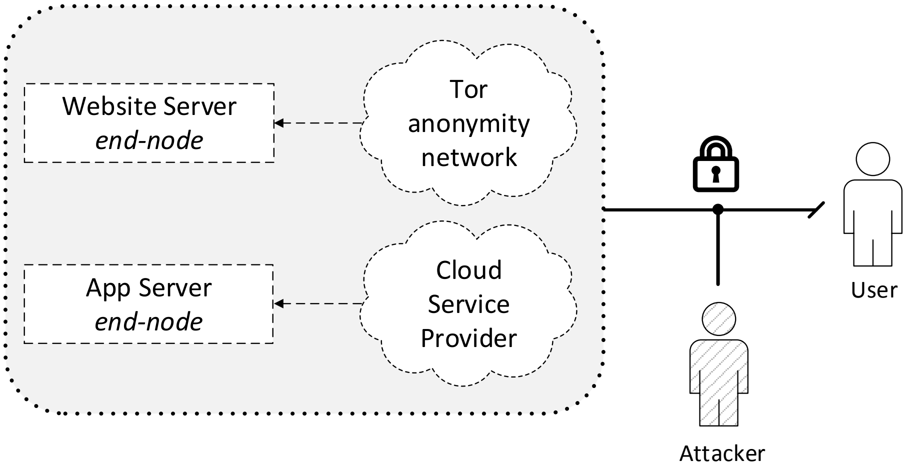


Figure 1.1: Illustration of website and app fingerprinting

the knowledge of apps running on a particular device in order to improve user experience. On the other hand, an adversarial view of such knowledge may lead to initiation of targeted attacks [142] involving known vulnerabilities in apps. While apps do not hide the destination IP addresses, they may access multiple overlapping destinations. For example, two apps may access the same 3rd-party library while utilizing the service in a distinct manner. For a man-in-the-middle observing network traffic, identifying the two apps on the same device is hard when relying only on the IP addresses. However, the apps may have distinct network traffic patterns useful for discrimination. We call the identification of apps on a device, using their encrypted network traffic patterns, *App Fingerprinting*, denoted by AFIN.

A fundamental challenge in performing traffic fingerprinting over encrypted data is the identification of characteristic features, which are often used in machine learning classifiers. In particular, encrypted traffic consists of network packets that carry application data along with other control messages depending on the communication protocol. In general, a protocol such as TCP limits the size of each packet. Moreover, each packet incurs a finite transmission time depending on the network path followed from its source to its destination. When a man-in-the-middle passively captures a sequence of packets flowing at the user's end, the packet size, time-stamp, and direction can be observed to form a set of features. As the goal of fingerprinting is to determine end-node patterns, one must consider a sequence of network packets in the captured traffic generated during a communication session involving the end-node under investigation. We call this sequence of packets a *trace*.

Over time, the captured network traffic may contain multiple traces associated with a set of end-nodes with different sessions initiated by the same user. In this setting, feature extraction is performed over each trace by combining features of each of its packets in a suitable manner [92, 70, 35, 137, 34, 106]. Most existing techniques combine features by assuming independence between subsequent transmissions [34, 106]. Therefore, relationship between packets in a TCP session, occurring consecutively in opposite directions (*viz.*, uplinks from user to server, or downlinks from server to user), are ignored. A relationship between these packets may exist due to control messages resulting from the current data transmission.

Another major challenge in traffic fingerprinting is the changes of behavioral patterns in network traffic over time, due to changes in the end-node content. While traffic fingerprinting can be seen as a continuous process with a man-in-the-middle observing network traffic perpetually, a classification model trained initially captures patterns in network traffic available at that particular time. However, traffic patterns may evolve over time, changing their distinguishing characteristics. Since these changes are not reflected in the classifier, its performance degrades while classifying newer data. A recent study in WFIN observed this temporal behavior [75]. Yet, this remains an open challenge.

In this dissertation, we introduce BIND (fingerprinting with BI-directioNal Dependence), a new set of features from encrypted network traffic, that incorporates feature relationships between consecutive sets of packets in opposite directions. These features are used in conjunction with other independent features to enrich discriminating factors of end-nodes during pattern recognition. Furthermore, we propose a technique for adapting the classifier to temporal changes in data patterns while fingerprinting over a long period of time. Our approach continuously monitors the classifier performance on the training data. When the accuracy drops below a predefined threshold, we replace the classifier with another one trained on the latest data. We call this AdaBind (ADAptive fingerprinting with BI-directioNal Dependence).

## 1.2 Vector Space Models

In semantic vector space models (VSMs) of language, each word is represented as a real-valued vector. These vectors can be utilized as features in a variety of natural language processing and machine learning tasks [112, 99, 100]. The constructed word vectors exhibit interesting semantic and syntactic regularities. For example, in word vector space, the sentence "king to queen is as man to woman" was proven to be represented as king $-$ queen $=$ man $-$ woman [100]. There are many proposed methods for VSMs. They take a text corpus as input and give us word vectors. Initially, a vocabulary or dictionary is built from the training data and then based on the followed approach, the vectors are learned for each word. Mikolov et al. [98] introduced a word to vector algorithm that focuses on representations of words learned by neural networks. Most recently, Pennington et al. [112] proposed "GloVe", a global vector log-bilinear regression model that uses global matrix factorization and local context window methods.

Existing website fingerprinting studies focus on collecting packets from the user's network and extract statistical features which are used by machine learning techniques to predict the destination of web pages. In this dissertation, we propose the packet to vector ($P2V$) approach. We model the website fingerprinting attack using the Global Vector space representation (GloVe) [112] as one of the most recent word vectors methods. We construct a corpus from network packets and represent these packets as real-valued vectors. We show how global log-bilinear regression models are appropriate to improve the website fingerprinting attack. We demonstrate how the suggested model outperforms previous website fingerprinting works.

The intuition behind P2V is as follows. Communication between client and server is stacked over the TCP protocol. Based on connection measures like network congestion, the TCP protocol uses flow control mechanisms such as window size and scaling, acknowledgement and sequence numbers, and others to ensure a certain understanding between client and server. Accordingly, the number of bytes (packet lengths) and hence time of transmitted packets in each direction are decided based on this fact. This means each packet flow affects the subsequent packet flow. This mechanism continues until communicating parties flag to finalize the connection. We view this understanding between client and server as a language or dialogue between two parties.

Moreover, the GloVe model leverages statistical information by training on elements in a word-word co-occurrence matrix. In this matrix, based on a sliding context window, each element $X_{ij}$ tabulates the number of times word $i$ occurs in the context of word $j$. As described above, in TCP, each packet flow affects the next packet flow. This means there is a dependence between consecutive packet flows in a TCP connection. Hence, we build a packet-packet co-occurrence matrix which gives us meaningful counts for each trace (or website download).

Previous website fingerprinting studies ignored the TCP flow control packets (like the ACK packets) as they decrease accuracy and do not provide distinguishing statistical benefits between websites. In this work, it is the first time that the TCP flow control packets are utilized. We show how the ACK packets are essential to build the corpus as they enrich the vocabulary. Compared to NLP, the ACK packets may act as filter or stop words in English. Although some applications tend to eliminate these words as they are considered noise, there are other domains such as Author Attribution where these words are considered important as they provide distinguishing writing styles [23]. In website fingerprinting, the ACK packets are important as they provide accurate fingerprints for each website. In addition, as mentioned earlier, the GloVe model depends mainly on the co-occurrences of words (context-counting). It does not neglect stop words from the corpus. Instead, it assigns different weights for frequent co-occurrences.

## 1.3 Cyber Deception

For many organizations, identifying unseen cyber attacks before reaching vulnerable web servers (i.e., unpatched) has become a crucial necessity. Persistent and elaborate attacks against corporations and government organizations have become increasingly prevalent in recent years, posing an unprecedented threat to cyber security and the economy. In the year of 2015, and exceeding the double of previous year's rate, a new vulnerability was discovered *every week* and more than 75% of all legitimate web sites had unpatched vulnerabilities with almost 20% of them giving attackers full control over the vulnerable systems [129]. Unfortunately, the cost of data breaches caused by software exploits is expected to exceed $2.1 trillion in 2019 [77].

*Intrusion detection* [50] is widely known as an important means of mitigating such threats. This is based on the observation that most of the discovered damaging attacks often share similar characteristics and traits. Examples of such traits may include steps attackers follow to alter system configurations, open back doors, execute commands and files, and steal collected information from compromised devices. When an intruder sends the initial infection, such malicious activities often leave telltale traces that can be detected even when the exploited vulnerabilities are not known to defenders. Therefore, the challenge is to gather, characterize, and filter these attack trails from target applications, connected devices, and network traffic and develop a defense mechanism to accurately and effectively leverage such traces to disrupt and block ongoing threats and prevent any future attempted exploits. Particularly, machine learning-based intrusion detection systems send alerts to admins when detecting any deviations from normal behavior in the collected data [133].

Despite its great promise, machine learning-based intrusion detection systems capabilities have been drastically hampered by many challenges that arise in cyber security domain. One of the challenges is the scarcity of current, realistic, publicly available cyber attack datasets. Another challenge is the difficulty of efficiently and accurately labeling such datasets that are often big and complex. This *data drought* challenge has frustrated thorough, comprehensive, and timely training of machine learning-based intrusion detection systems (IDSes). The consequence is raising the false alarm rates and increasing their tendency to attacker adversarial evasion techniques [29, 38, 63, 108, 124].

Toward mitigating this data drought issue, we propose a novel deception-based methodology that enhances IDSes web data for more accurate, efficient, and more timely evolution of IDSes to emerging attacks and attacker evasions. Deceptiveness has long been identified as an important factor to effective cyber warfare [144]. However, its applications in IDSes have been introduced in isolated environments where deception is secluded and separate from the actual streams in which intrusions must be captured. For instance, the use of *honeypots* to collect malicious activities is a typical application [135]. Unfortunately, these methods have limited training values as they train IDSes to detect only malicious activities against non-production-server honeypots, or attacks carried out by unskilled adversaries who are unable to avoid honeypots.

To overcome these limitations, this work introduces a novel approach that leverages recent software deception techniques in which deceptive attack responses are integrated into live production server software through the use of *honey-patching* [21, 20]. Deployed into live production servers, honey-patching systems respond to malicious activities by redirecting the attacker's connections to a perfect isolated decoy environment while providing equivalent security to traditional patching systems. The purpose of isolation is to enhance IDSes web data streams by transparently monitoring and disinforming deceived attackers. These *deception-based* collected streams alleviate the data drought issue by providing machine learning-based intrusion detection systems with relevant, current, and feature-rich data to detect and prevent sophisticated attacks.

We show the effectiveness of this new intrusion detection system through the design and implementation of DeepDig (DEcEPtion DIGging), a framework for a novel deception-based IDSes. We evaluate our approach and show how extra information, collected via honey-patching deceptive systems and fed back into the classifier, improve the accuracy of IDSes tremendously. We believe the approach offers exceptional promises for future machine learning-based intrusion detection systems through generating automatically-labeled and rich web attack data streams.

## 1.4 Contributions of the dissertation

In this dissertation, we propose solutions for different challenges facing traffic analysis and cyber security.

### 1.4.1 Traffic Fingerprinting Feature Engine

- We propose a new feature extraction method, called BIND, for fingerprinting encrypted traffic to identify an end-node. In particular, we consider relationships among sequences of packets in opposite directions.

- We propose a method, called AdaBind, in which the machine learning classifier adapts to the changes in behavioral patterns that occur when fingerprinting over a long period of time. We continuously monitor classifier performance, and re-train it in an online fashion.

- We evaluate the proposed methods over two applications, namely website fingerprinting (Wfin) and app fingerprinting (Afin). We perform Afin over encrypted traffic, which has not been explored in existing studies. Moreover, we use a variety of datasets for both Wfin and Afin while employing defense mechanisms to show the effectiveness of the proposed approaches especially in the open-world settings.

### 1.4.2 Feature Transformation using Vector Space Models

- We propose a packet to vector (P2V) model for the website fingerprinting attack. We build a corpus from network packets and represent these packets as real-valued vectors.

- Unlike previous website fingerprinting works, we consider the TCP ACK packets as essential elements to build the corpus as they enrich the vocabulary and hence increase the accuracy of the website fingerprinting attack.

- We show how P2V can remarkably increase the accuracy of website fingerprinting when compared to previous approaches.

- We also show that our P2V technique is more immune and resilient to website fingerprinting countermeasures (defenses) than previous classifiers.

### 1.4.3 Traffic Fingerprinting Defense

- We introduce a novel traffic fingerprinting defense, called BiMorphing, to thwart the fingerprinting cyber attack. Specifically, BiMorphing considers dependence between consecutive sequences of packets in opposite directions.

- We propose a new defense algorithm that leverages dependency sampling and zero latency traffic transmission.

- We show how this defense achieves minimum bandwidth overhead through the use of mathematical optimization techniques.

- We implement and evaluate our approach against a Tor dataset in both closed-world and open-world scenarios and show how the proposed methodology outperforms the state-of-the-art defense.

### 1.4.4 Enhancing Intrusion Detection with Cyber Deception

- We propose a new intrusion detection system that leverages advances in deception-based techniques for attack labeling and feature extraction process.

- We show how multi-dimensional data (i.e., network and system events) collected at decoys can support richer feature sets for attack characterization, and therefore better, more accurate detection of malicious activities, which is resistant against attacker evasion strategies.

- To quench data drought, we present a framework for generating realistic web data for both benign and malicious traffic.

- We implement and evaluate our approaches on large-scale network- and system-level events generated by a test bed built atop production web software, including the Apache web server.

## 1.5 Outline of the dissertation

The rest of the dissertation is organized as follows. Chapter 2 discusses related work in traffic analysis and cyber security. We present relevant background information and related studies in Wfin and Afin. We then discuss related work in Vector Space Models. Finally, we present a literature review about traditional intrusion detection systems and cyber deception.

Chapter 3 presents the novel feature extraction approach in details. It first discusses BIND and AdaBind. This is followed by the empirical evaluation including datasets, experiments, and results to show the effectiveness of the introduced approach.

Chapter 4 describes the P2V feature transformation approach. The model is then evaluated where we compare our results with previous traditional traffic fingerprinting studies.

Chapter 5 introduces the new defense mechanism (BiMorphing) to prevent the fingerprinting attack. We start by presenting the BiMorphing general methodology. Next, we discuss the new sampling technique that integrates an optimization method and a zero delay approach to achieve minimum bandwidth overhead and zero latency. We then show the effectiveness of the introduced defense empirically in the closed-world and open-world settings against known attacks and defenses.

Chapter 6 outlines our cyber deception approach and presents an overview of DeepDig, followed by a more detailed architecture description. Next, we show how our approach can support accurate characterization of attacks through decoy data. Finally, we summarize the implementation, followed by evaluation methodology and results.

Chapter 7 concludes the dissertation with possible avenues of future work and Appendix A presents other research publications performed during the course of the Doctoral study.

# CHAPTER 2

# LITERATURE SURVEY [1] [2] [3] [4]

In this chapter, we present relevant existing studies in traffic analysis and intrusion detection systems and distinguish our work from prior studies.

## 2.1 Network Traffic Fingerprinting

We start by giving a background about website fingerprinting (Wfin), apps fingerprinting (Afin), and defenses.

### 2.1.1 Website Fingerprinting

The online activity of a user accessing websites can be hidden using anonymity networks such as Tor [51]. Tor provides a low latency encrypted connectivity to the Internet, while anonymizing the connections via a process called pipeline randomization. A circuit of three relay nodes is formed within the Tor network, composed of an entry node, an exit node, and

[1] This chapter contains material previously published as: K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K.W. Hamlen, and B. Thuraisingham. "Adaptive encrypted traffic fingerprinting with bi-directional dependence." In Proceedings of the 32nd Annual Computer Security Applications Conference, pp. 177-188. ACM, 2016. DOI: https://doi.org/10.1145/2991079.2991123. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[2] This chapter contains material previously published as: ©2015 IEEE. Portions Reprinted, with permission, from K. Al-Naami, G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan and B. Thuraisingham. "P2V: Effective Website Fingerprinting Using Vector Space Representations." IEEE Symposium Series on Computational Intelligence, December 2015. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[3] Some of the work presented in this chapter was performed in collaboration with L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Lead author Al-Naami conducted the majority of the research, including most of the design, the full implementation, and the full evaluation.

[4] Some of the work presented in this chapter was performed in collaboration with F. Araujo, A. Gbadebo, A. Mustafa, L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Al-Naami led the machine learning half of the research, including feature generation, classification design, implementation, and experiments.

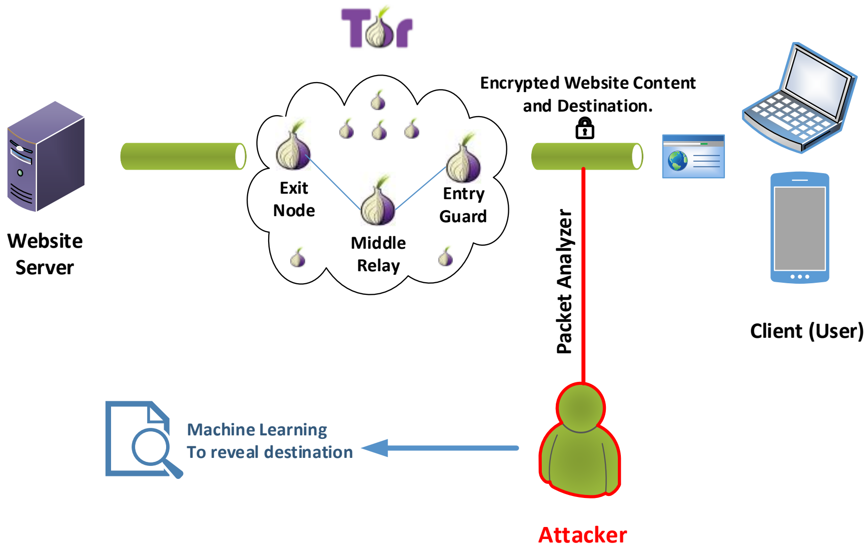


Figure 2.1: An example of Tor. A client or user connects to the Internet (server) using Tor network. The three Tor nodes are shown. The website fingerprinting attack occurs between the user and the Tor entry guard.

a randomly selected relay node. Circuit connections are reestablished approximately after every 10 minutes of usage [18]. Figure 2.1 depicts the communication over Tor.

Fingerprinting under this setting is hard due to the decoupling of user request with end-node (i.e., web server) response. Nevertheless, this challenging problem of WFIN has gained popularity in the research community with numerous studies [92, 70, 35, 137, 34, 106] proposing techniques to perform fingerprinting, and also to defend against it. The inductive assumption is that each website has a unique pattern in which data is transmitted from its server to the user's browser. Moreover, each website content is unique. Using this assumption, the website fingerprinting scenario, generally perceived as an attack against user's privacy, employs a statistical model to predict the website name associated with a given trace. Whereas, a defense mechanism explores methodologies to reduce the effectiveness of such models capable of performing an attack.

**Attack**.

The primary form of attack is to train a classifier using traces collected from different websites, where each trace is represented as a set of independent features. Information present in network packets associated with each trace is summarized to form a histogram feature vector, where the features include packet length (size) and direction (as used in [92]). In addition, Panchenko et al. [107] introduced a set of features extracted from a combination of packets known as *Size Markers or Bursts*. A burst is a sequence of consecutive packets transmitted along the same direction (uplink or downlink). Features such as burst sizes are computed by summing the length of each packet within a burst. These, along with other features such as unique packet sizes, HTML markers, and percentage of incoming and outgoing packets, form the feature vector for a trace. Dyer et al. [57] also used bandwidth and website upload time as features.

A recent work by Panchenko et al. [106] proposes a sampling process on aggregated features of packets to generate overall trace features. Importantly, Cai et al. [35] obtained high classification accuracy by selecting features that involve packet ordering, where the cumulative sum of packet sizes at a given time in each direction is considered. This feature set was also confirmed to provide improved classification accuracy in [137]. It indicates that features capturing relationships among packets in a trace are effective in distinguishing different websites (or end-nodes). In this dissertation, we focus on extracting such capability from traces in a novel fashion by capturing relationships between consecutive bursts in opposite directions.

While these features are used to train a classifier, e.g. Naïve Bayes [57] and Support Vector Machine (SVM) [107], studies have identified two major settings under which website fingerprinting can be performed. First, the user is assumed to access only a small set of known websites. This restriction simplifies the training process since the attacker can train a model in a supervised manner by considering traces only from those websites. This form of classification is known as *closed-world*. However, such a constraint is not valid in general as a user can have unrestricted access to a large number of websites. In this case, training a classifier by collecting trace samples from all websites to perform multi-class classification is unrealistic. Therefore, an adversary is assumed to *monitor* access to a small set of websites called the *monitored set*. The objective is to predict whether a user accesses one of these monitored websites or not. This binary classification setting is called *open-world*. Wang et al. [137] propose a feature weighting algorithm to train a $k$-Nearest Neighbor ($k$-NN) classifier in the open-world setting. They utilize a subset of traces from the monitored websites to learn feature weights which are used to improve classification. In this dissertation, we evaluate our proposed feature extraction approach on both these settings. Particularly for the open-world case, we utilize the feature weighting method proposed in [137] to perform a comparative study of feature extraction techniques.

A study by Juarez et al. [75] observes and evaluates various assumptions made in previous studies regarding Wfin. These include page load parsing by an adversary, background noise, sequential browsing behavior of a user, and replicability due to staleness in training data with time, among others. While recent studies [139, 64] have addressed each of these issues by relaxing appropriate assumptions, the issue of replicability still remains an open challenge. Wang et al. [139] attempt to address the issue of staleness in training data over time within their $k$-NN model [137] specific to open-world. They score the training data consisting of traces based on model performance of 20 nearest neighbors. However, this methodology cannot be generalized, i.e., it is not applicable if one uses a classifier other than $k$-NN. Moreover, it is also not applicable to the closed-world setting. In this dissertation, we introduce a generic method to update the classifier model for replicability of Wfin and Afin over long periods of time.

**Defense**.

The topic of website fingerprinting defense has been an active area of research. Several defenses have been proposed to resist website fingerprinting attacks. All of the defenses aim to obfuscate the pattern of the packets of the loaded website. These defenses vary from padding packets with extra bytes to morphing the website packet length distribution and make it appear to come from another target distribution (i.e., a different website) [57]. In packet padding, each packet size in the trace is increased to a certain value depending on the padding method used.

Padding techniques come with the overhead of appending large number of bytes to packets. Therefore, several other smart padding defenses have been introduced. We describe three of the most effective website fingerprinting defenses that we consider when evaluating our approach later when we discuss results.

- **Pad to MTU**. MTU (Maximum Transmission Unit) determines the maximum size of each packet in a communication between two ends. In a Pad to MTU defense [57], each packet is padded to the maximum size (MTU). This technique prevents the attacker from extracting detailed packet lengths distribution information which help machine learning classifiers to identify webpages. There is a tradeoff, however, when using this defense as it comes with a high cost of appending bytes to every packet of size less than MTU.

- **Direct Target Sampling (DTS)**. DTS was proposed by [143]. It is considered as a distribution-based defense which makes the packet length distribution of a certain website appear as coming from a different website distribution. It has an advantage over the pre-packet padding techniques, like Pad to MTU, in that it requires less overhead by appending less bytes depending on the distribution of the target webpage. Furthermore, as a distribution-based technique, DTS defense proves to be more effective than the traditional packet padding.

As an example, let's consider two webpages $S$ and $T$ where $S$ is the source and $T$ is the target. We derive two distributions $D_S$ and $D_T$ from their packet length histograms. From $D_T$ probabilities, we build the target Cumulative Distribution Function ($CDF_T$) to sample random variables (packet lengths) for each packet in $S$ by running a pseudorandom number generator to get a number between zero and one inclusive. So for every $P_i$ (packet of length $i$ in $S$) , we sample $P_j$ (a packet of length $j$ using $CDF_T$). If $j > i$, we pad $P_i$ to length $j$ and send it, otherwise, we send $P_i$ as is. We continue random sampling from $D_T$ until all $S$ packets have been consumed. The result is a new distribution $D_N$.

In addition, we continue sampling from $D_T$ until the $L1$ distance between the new distribution $D_N$ and the target distribution $D_T$ is less than a predefined threshold, which empirically was determined to be 0.3 [143].

- **Traffic Morphing (TM)**. TM [143] is similar to DTS but reduces the cost or overhead by using Convex Optimization methods. Wright et al. [143] introduced the cost function as the objective function that we like to minimize. The convex optimization parameters are probabilities in an $m \times m$ two dimensional array $A$ where $m$ is the MTU in TCP/IP transmission.

  Figure 2.2 depicts this process. Each column in $A$ is a Probability Mass Function ($PMF$) whose values sum up to 1. Similar to DTS, from each column's $PMF$, we generate the corresponding $CDF$. We do that in DTS, but we do it once for the target website distribution $D_T$.

  As an example, to morph the source website $S$ to the target website $T$, we need to learn the parameters in $A$ such that $T = AS$. For each packet of length $i$ in $S$, $P_i$, we go to the $i^{th}$ column in $A$ and run a pseudorandom number generator over its cumulative

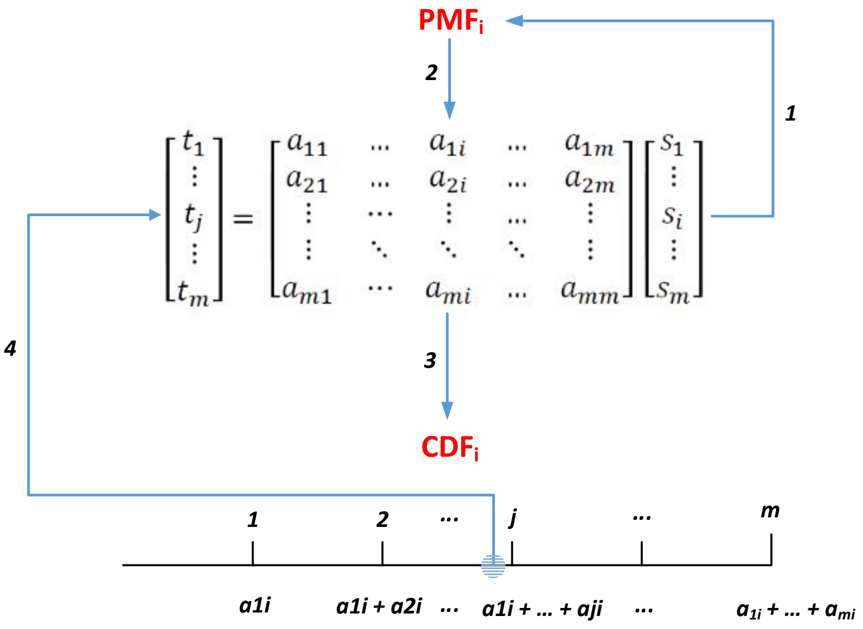


Figure 2.2: Traffic Morphing.

distribution function ($CDF_i$) to sample $P_j$ (a packet of length $j$ from $T$). If $j > i$, then we pad $P_i$ to length $j$, otherwise, we split packet $P_i$ and send. Wright et al. [143] introduced other constraints in the convex optimization method as $i < j$ so there is no need to split packets from the source website as this affects the quality of some applications like streaming data as in audio or video. The result is a new distribution $D_N$.

Similar to DTS, we continue sampling from $D_T$ until the L1 distance between $D_N$ and the target distribution $D_T$ is less than 0.3.

In our study, we evaluate our approaches by applying these padding techniques to packets while performing the closed-world settings in website fingerprinting.

In the case of open-world setting, Dyer et al. [57] introduced a defense mechanism, called Buffered Fixed Length Obfuscator (or *BuFLO*), that not only uses packet padding, but also modifies packet timing information by sending packets in fixed intervals. Cai et al. [34] improved BuFLO and introduced a lighter defense mechanism, called *Tamaraw*, which considers different time intervals for uplink and downlink packets in the open-world setting.

Wang et al. [140] proposed a one-to-one burst molding defense that fuses uni-bursts of source and target websites. Our introduced BiMorphing defense morphs bi-bursts using sampling and optimization techniques for a better bandwidth overhead and zero delay packet transmission.

### 2.1.2 App Fingerprinting

An increase in popularity of smartphone applications has attracted researchers to study the issues of user privacy and data security in apps developed by third-party developers [136]. In particular, many studies have proposed methods to perform traffic analysis while a user uses an app. Dai et al. [46] first proposed a method to identify an app by using the request-response mechanisms of API calls found in HTTP packets. They perform UI fuzzing on apps whose network packets are captured using an emulator. Similarly, [102] proposes a method to fingerprint apps using comprehensive traffic observations. These studies perform app identification (or fingerprinting) using only HTTP traffic. Such methods cannot be applied on HTTPS traffic since the packet content is encrypted and not readily available.

Studies on performing traffic analysis over HTTPS app network traffic explore varied applications including smartphone fingerprinting [128], user action identification [44, 43], user location tracking [24], and app identification [102]. They use packet features such as packet length, timing information, and other statistics to build classifiers for identification (or prediction). Note that this is similar to the Wfin setting mentioned in §2.1.1. Recently, a study [131] performed Afin using both HTTP and HTTPS data. They use features such as burst statistics and network flows. Here, a flow is a set of network packets belonging to the same TCP session. They train a random forest classifier (ensemble of weak learners) and a support vector machine (SVM) using features extracted from network traffic of about 110 apps from the Google play store. Evaluation of their method is similar to the closed-world setting of Wfin, where network traffic from apps considered for training and testing the model belong to a closed set, i.e., the user has access to only a finite known set of apps. The method resulted in an overall accuracy of 86.9% using random forest, and 42.4% using SVM. These results are based on a small dataset of apps which may have both HTTP and HTTPS traffic. Furthermore, they only show a closed-world setting. However, with a large number of apps present on various app stores, these results may not reflect a realistic scenario of the open-world setting in Afin.

Similar to that of Wfin, the open-world setting in Afin assumes that the man-in-the-middle monitors the use of a small set of apps called the *monitored set*. The goal is to determine whether a user is running an app that belongs to this set. In our evaluation, we use our proposed technique for traffic analysis on a larger dataset of apps that only use HTTPS for connecting to remote services. Contrary to Wfin where the network is anonymized, apps do not use an anonymity network. However, the effect of anonymization is similar to that of Wfin. In Wfin, anonymization results in removal of destination website identifiers (i.e., IP address). In Afin, apps connect to multiple remote hosts deriving remote services from them. However, multiple apps may connect to the same host. A mere list of hosts or IP addresses is not sufficient to deterministically identify an app. This property effectively anonymizes such apps with respect to the network. We therefore rely on traffic analysis to perform Afin. In this work, we show the applicability of both closed-world and open-world settings while utilizing the BIND feature extraction method.

## 2.2 Vector Space Models

In this section, we present relevant background regarding Vector Space Models (VSMs) that is used in many Natural Language Processing (NLP) and Machine Learning (ML) applications. Specifically, we explain the Global Vector for word representation model (GloVe) [112].

### 2.2.1 GloVe

Vector space models (VSMs) of language have proven to be very useful in many NLP and machine learning tasks. In VSMs, each word in a corpus is represented as a real-valued vector. These vectors can be used as features in many applications. Word to vector [98] and GloVe [112] are two of the most recent algorithms used for building word vectors. Mikolov et al. [98] presented a word to vector algorithm that uses neural networks to learn representations of words. Most recently, Pennington et al. [112] proposed Global Vectors for Word Representations (GloVe in short). GloVe was shown by authors to outperform many VSMs including the word to vector [98] method mentioned above. Hence, in this dissertation, we use GloVe as one of the newest methods, to model the website fingerprinting attack.

Given a text corpus as input, GloVe builds word vectors in an unsupervised learning manner. The basic idea is to use word statistics as the primary source of information by examining word co-occurrences in the corpus. In a high level overview, we can summarize the GloVe algorithm as follows. Before training the model, we first construct the word-word co-occurrences matrix. Then considering word pairs, GloVe finds a log-bilinear regression model that includes word vectors as parameters. Finally, using any gradient descent algorithm, the model parameters (word vectors) are computed. We now present the GloVe algorithm in more details.

- ***The Matrix***. GloVe starts off by running through the corpus once to build the global word-word co-occurrence matrix $X$. Based on a sliding context window, each entry

$X_{ij}$ tabulates the number of times word $i$ occurs in the context of word $j$. The result is a sparse matrix with a lot of zero entries. If the corpus is large, this counting step may be expensive. However, it is just a single pass that happens only once. The advantage about GloVe is that it trains the model on the non-zero entries of $X$ which makes the training iterations much faster. Now that $X$ is ready, we will use it in place of our corpus.

- ***The Model***. Generally speaking, given a sample data on two variables $x$ and $y$, the equation $y = \beta_1 x + \beta_0$ is considered one of the simplest linear models, where $\beta_1$ is the slope and $\beta_0$ represents the intercept with the $y$-axis. Learning the optimal parameters $\beta_0$ and $\beta_1$ gives the best line (predictor) that ties variables $x$ and $y$. One of the techniques used is to minimize a loss or objective function by using the gradient descent iterative algorithm.

  GloVe follows a similar approach. It constructs a model for the variable $X_{ij}$ in the co-occurrence matrix $X$. The model has parameters (word vectors) to be learned by minimizing an objective function using a gradient descent-like algorithm. GloVe's concept revolves around the notion that word vector spaces have substructure that should be considered when designing algorithms to build word vectors. Typically, for nearest neighbor tasks, the existing similarity metrics such as Euclidean distance (or Cosine similarity) produce a single scalar value that may not capture intricate relationships between words. The GloVe model suggests using the vector difference between the two word vectors as this captures more interesting and useful meanings. The word vector learning model has been built considering the ratios of co-occurrence probabilities between words which can be calculated directly from $X$. The result is the log-bilinear regression model in Eqn. 2.1 for each word pair of word $i$ and word $j$.

$$w_i^T w_j + b_i + b_j = \log X_{ij}, \tag{2.1}$$

where the $d$-dimensional word vectors $w_i, w_j \in \mathbb{R}^d$ and $b_i, b_j$ are scalar bias terms associated with words $i$ and $j$. The model in Eqn. 2.1 constructs word vectors that are guaranteed to retain useful information about co-occurrence of words $i$ and $j$.

- ***The Objective Function***. To learn the parameters $w_i$, $w_j$, $b_i$, and $b_j$, we need to minimize an objective or cost function that considers the model in Eqn. 2.1. However, the problem with this model is that it produces equal weights for all word-word co-occurrences in $X$. As some words may co-occur rarely, their co-occurrences are noisy and can be neglected. To eliminates the noise effect, the following weighted least squares model is introduced.

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})\ (w_i^T w_j + b_i + b_j - \log X_{ij})^2, \tag{2.2}$$

where $V$ is the vocabulary size and $f(X_{ij})$ is a weighting function to eliminate noise. In order for $f(X_{ij})$ to work as such, it should satisfy some properties. It should be non-decreasing to deal with rare co-occurrences. Also, for large values of $X_{ij}$, $f(X_{ij})$ should return 1 so frequent co-occurrences are not overweighted. The following equation shows how $f(X_{ij})$ satisfies the properties mentioned above.

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^{\alpha}, & \text{if } X_{ij} < x_{max} \\ 1, & \text{otherwise.} \end{cases} \tag{2.3}$$

The weighting function simply returns 1 if $X_{ij} \geq x_{max}$. For all other co-occurrences, a value between 0 and 1, controlled by $\alpha$, will be returned. The authors have found the model performs best with $x_{max} = 100$ and $\alpha = 3/4$.

Running gradient descent on Eqn. 2.2 learns the values of the word vectors which can be used as features in many NLP and machine learning tasks.

## 2.3 Intrusion Detection and Cyber Deception

In this section, we survey existing approaches in Intrusion Detection and Cyber Deception.

Machine learning-based intrusion detection systems [63, 103, 86, 91, 38, 133] detect patterns that deviates from expected system behavior. Typically, they can be categorized into host-based and network-based systems.

Detectors that are host-based find intrusions in the form of abnormal system call trace sequences where co-occurrence of calls is key to characterizing malicious behavior. For instance, malware behavior and privilege escalation often dominate specific system call patterns [38]. Seminal work in this area has analogized intrusion detection via statistical profiling of system events to the human immune system [61, 73]. A number of related approaches have followed this work where histograms constructing profiles of normal behaviour have been used [95]. Another prominent host-based method utilizes sliding window approaches to convert system call event sequences to values used as input to classical machine learning classification [141, 42]. More recently, long call sequences have been studied to detect attacks buried in long execution paths [120].

Network-based intrusion detection systems use network data to detect intrusions. Typically, they are deployed at the network level to detect abnormal behaviors as results of external threats such as unauthorized access [29]. The area of network intrusion detection has been intensively researched in the literature [29, 12]. Studied approaches can be categorized into classification approaches (e.g., SVM [60], Bayesian network [83]), information-theoretic approaches [88], and statistical techniques [84].

Network-based detectors has the capability to monitor many hosts in the organization with relatively low costs. However, they are vulnerable to inside attacks or malicious threats that uses encrypted traffic. Operating at the host level, host-based detectors, on the other hand, overcome encrypted traffic issues and obfuscation methodologies [81] as they have complete visibility of malicious system events. In this work, we introduce an approach that utilizes existing techniques and integrates host- and network-based detection mechanisms. We show how this approach can offer capabilities to the intrusion detection systems that can detect abnormal behavior and resist adversarial evasion techniques.

Another area of research is *web-based* malware detection that aims at detecting drive-by-download attacks using static analysis, dynamic analysis, and machine learning techniques [79, 36]. In addition, other studies focus on *flow-based* malware detection by extracting features from proxy-logs and use machine learning [27].

### 2.3.1 Cyber-Deception in Intrusion Detection

Honeypots are computer security resources designed to counteract attempted threats by attracting, detecting, and gathering malicious activities [127]. By design, any interaction with such resources is likely to be a malicious activity. Shadow honeypots [19] combine the honeypots design and anomaly detection concept by providing feedback to the anomaly prediction for a better classification.

### 2.3.2 Intrusion Detection Feature Generation

Feature generation and machine learning classification approaches have been extensively studied in the literature to perform host- and network-based intrusion detection [96]. One of the challenges that network-based detectors face is they are usually opaque to encrypted traffic. Extracting features from encrypted network packets has been researched in other domains such as *traffic fingerprinting*, where attackers attempt to reveal which destination websites are visited by users (victims) for different purposes. As a consequence, users typically utilize anonymous networks, such as The Onion Router (Tor) [137, 14], to protect their privacy. However, by training machine learning classifiers directly on features extracted from packet headers only (i.e., encrypted packets), attackers can still predict websites and hence threaten users' privacy. Examples of such features include packet size, time, and direction, which are augmented in various ways to construct histogram feature vectors [92, 107, 57, 137, 14]. Other approaches leverage natural language processing techniques via the use of vector space representations to convert encrypted packets to word vectors for improving the fingerprinting attack [13]. On the other hand, from unencrypted data, host-based detectors build features augmented from sequences of system calls [33, 95]. In this work, we introduce a novel classification ensemble technique that utilizes a host- and network-based hybrid model.

# CHAPTER 3

# FINGERPRINTING WITH BI-DIRECTIONAL DEPENDENCE[1]

## 3.1 Approach

In this chapter, we detail the BIND approach introduced in Chapter 1 and present the methodology to extract the BIND features, and detail the AdaBind approach [14].

### 3.1.1 Features

With encrypted payload of each packet in a trace, we extract features from packet headers only. The main idea is to extract features from consecutive bursts to capture any *dependencies* that may exist between them. As illustrated in Figure 3.1, we call the burst directed from a user/client (or app) to server (e.g., burst $a$), an uplink uni-burst (or *Up uni-burst*), and the burst directed from server to the user, a downlink uni-burst (or *Dn uni-burst*) (e.g., burst $b$). Similar to packets, a burst or uni-burst has features such as size (or length), time, and direction. Uni-burst size is the summation of lengths of all its packets. Packet time is the departure/arrival timestamp in the uplink/downlink direction, measured near the user-end of the network by a man-in-the-middle. Uni-burst time is the difference between the last packet's timestamp and the first packet's timestamp within a burst, i.e., the time taken to transmit all packets of a burst in a specific direction. Here, the term burst and uni-burst are equivalent. The name uni-burst emphasizes on the fact that features are extracted from a single burst, as opposed to *Bi-Burst* which is a tuple formed by a sequence of two adjacent uni-bursts in opposite direction (e.g., bursts $b$ and $c$ in Figure 3.1).

[1] This chapter contains material previously published as: K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K.W. Hamlen, and B. Thuraisingham. "Adaptive encrypted traffic fingerprinting with bi-directional dependence." In Proceedings of the 32nd Annual Computer Security Applications Conference, pp. 177-188. ACM, 2016. DOI: https://doi.org/10.1145/2991079.2991123. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

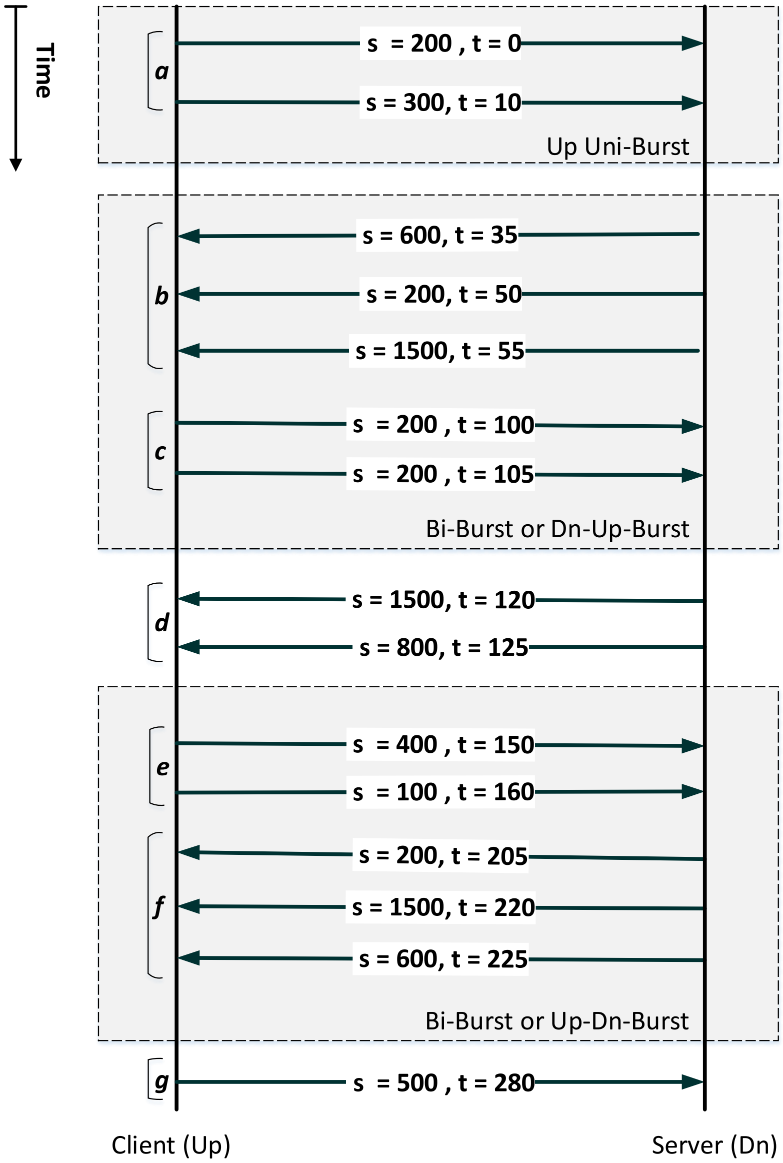


Figure 3.1: An example illustrating BIND Features.

**Bi-Burst features**.

Features extracted from Bi-Bursts are as follows.

1. **Dn-Up-Burst size**: Dn-Up-Burst is a set of tuples formed by downlink (Dn) - uplink (Up) consecutive bursts. Here, unique tuples are formed according to the corresponding uni-burst lengths where each tuple forms a new feature.

Table 3.1: Features from Packets, Uni-Bursts, and Bi-Bursts.

| **Category** | **Features** |
|---|---|
| Packet (Up/Dn) | Packet length |
| Uni-Burst (Up/Dn) | Uni-Burst size |
| | Uni-Burst time * |
| | Uni-Burst count |
| Bi-Burst (Up-Dn/Dn-Up) | Bi-Burst size * |
| | Bi-Burst time * |

*new features introduced in this work

2. **Dn-Up-Burst time**: This set of features considers unique consecutive uni-burst time tuples between adjacent Dn uni-burst and Up uni-burst sequences.

3. **Up-Dn-Burst size**: Similar to Dn-Up-Burst size features, these features consider burst length tuples of adjacent Up uni-burst and Dn uni-burst sequences.

4. **Up-Dn-Burst time**: Similar to Dn-Up-Burst time features, this set of features considers burst time tuples formed by adjacent Up uni-burst and Dn uni-burst sequences.

In each trace, we count such unique tuples to generate a set of features. To overcome dimensionality issues associated with burst sizes, *quantization* [53] is applied to group bursts into correlation sets (e.g., based on frequency of occurrence).

**Packet and Uni-Burst features**. In addition to the Bi-Bursts features, we also use burst size and burst time features. Previous studies [57] only consider total trace time as a feature, contrary to the burst time feature we use in this dissertation. Furthermore, we also consider the count of packets within a burst as a feature. In order to capture variations of the packet features, we use an array of unique packet lengths as well. The set of features, termed as BIND, are listed in Table 3.1. All these features are concatenated to form a large array of features (*histograms*) to be extracted from each trace. A set of multiple traces represented in this manner forms the training and testing set.

**Summary**. We first establish some notations. Let $s^{\uparrow}$ and $s^{\downarrow}$ be the uplink packet size and downlink packet size, respectively. Similarly, let $t^{\uparrow}$ and $t^{\downarrow}$ be the uplink packet time and downlink packet time, respectively. The uni- and bi-burst features are formulated in Equation 3.1.

$$\begin{aligned}
\mathit{Up\text{–}Uni\text{–}Busrt\ Size} &= \sum_{k=0}^{n^{\uparrow}} s_k^{\uparrow}, \\
\mathit{Dn\text{–}Uni\text{–}Busrt\ Size} &= \sum_{k=0}^{n^{\downarrow}} s_k^{\downarrow}, \\
\mathit{Up\text{–}Uni\text{–}Busrt\ Count} &= \sum_{k=0}^{n^{\uparrow}} 1, \\
\mathit{Dn\text{–}Uni\text{–}Busrt\ Count} &= \sum_{k=0}^{n^{\downarrow}} 1, \\
\mathit{Up\text{–}Uni\text{–}Busrt\ Time} &= t_j^{\uparrow} - t_i^{\uparrow}, \qquad (3.1) \\
\mathit{Dn\text{–}Uni\text{–}Busrt\ Time} &= t_j^{\downarrow} - t_i^{\downarrow}, \\
\mathit{Up\text{–}Dn\text{–}Bi\text{–}Busrt\ Size} &= \sum_{k=0}^{n^{\uparrow}} s_k^{\uparrow} + \sum_{k=0}^{n^{\downarrow}} s_k^{\downarrow}, \\
\mathit{Dn\text{–}Up\text{–}Bi\text{–}Busrt\ Size} &= \sum_{k=0}^{n^{\downarrow}} s_k^{\downarrow} + \sum_{k=0}^{n^{\uparrow}} s_k^{\uparrow}, \\
\mathit{Up\text{–}Dn\text{–}Bi\text{–}Busrt\ Time} &= (t_j^{\uparrow} - t_i^{\uparrow}) + (t_j^{\downarrow} - t_i^{\downarrow}), \\
\mathit{Dn\text{–}Up\text{–}Bi\text{–}Busrt\ Time} &= (t_j^{\downarrow} - t_i^{\downarrow}) + (t_j^{\uparrow} - t_i^{\uparrow}),
\end{aligned}$$

Here, $n$ is the number of packets in a burst, $i$ is the first packet in a burst, and $j$ is the last packet in a given burst. Notice that bi-burst features should be in order (i.e., uplink followed by downlink and vice versa).

**Example**. Figure 3.1 depicts a simple trace where packet sequences between uplink and downlink are shown. Each packet in the figure has size $s$ in bytes and time $t$ in milliseconds. We set time for the first packet in the trace to zero, as a reference. An example of a uni-burst is shown as burst $a$, whose size is 500, computed by adding packet sizes $s = 200$ and $s = 300$ that form the burst. Its time is computed as 10, which is the absolute time difference between the last packet ($t = 10$) and the first packet ($t = 0$) in the burst. Similarly, a Bi-Burst example is shown as well, formed with a combination of bursts $b$ and $c$. This is denoted as Dn-Up-Burst. In this case, the Bi-Burst tuple using the burst size (i.e., Dn-Up-Burst size) is represented as {DnUp_2300_400}, where 2300 is the burst size of $b$, and 400 is the burst size of $c$. We count the number of such unique tuples in the trace. In this case, the count for {DnUp_2300_400} is 1.

### 3.1.2 Learning

In the closed-world setting, we use the BIND features to train a support vector machine (SVM) [45] classifier. SVM applies convex optimization and maps non-linearly separated data to a higher dimensional linearly separated feature space. Whereas in the open-world setting, using the BIND features, we apply the weighted $k$-Nearest Neighbor ($k$-NN) approach proposed in [137]. Feature weights are computed using traces from the monitored set. During testing of traces with unknown class labels, these feature weights are applied. Majority class voting among $k$-Nearest Neighbors is performed to predict class label of a test trace. Additionally, we also use a Random Forest classifier in the open-world setting. Instead of performing feature weighing, which is computationally expensive, we use a set of weak learners to form an ensemble of decision trees (random forest).

**Static Learning**.

Typically, previous studies (mentioned in §2.1.1) have focused on performing fingerprinting by collecting traces for a short period of time. Classifiers are trained on traces collected within this time period, and used to predict class labels thereafter. We refer to this type of classifier training as static. On the contrary, Wfin and Afin can be viewed as a continuous process involving trace collection over a long period of time. Moreover, data collection is time consuming. Changes in data content transmitted between end-nodes affect patterns captured in the model. Using a static model to predict class labels of test traces in this situation drastically affects classification performance.

**Adaptive Learning**.

We now present the details of AdaBind. In this section, we show how we model encrypted data fingerprinting in an adaptive manner. As discussed in §3.1.2, over time, the data patterns of the current traces may be different from the patterns in previously seen training traces. This is known as *concept drift* [67, 68]. To address this challenge, the model has to be updated (re-trained) regularly. We study the effect of re-training as follows.

**Fixed update**. One simple approach is to apply fixed updates to re-train the model periodically. We refer to this approach as BindFup (BIND Fixed UPdate). BindFup updates the model periodically, regardless of any concept drift that may happen. The model will be re-trained regularly (e.g., at the end of every week) with freshly obtained training data. There are two possible scenarios, *early update* and *late update*. In early update, BindFup updates the model in a way that ensures no concept drift in data. Although this update is more accurate and stable, it may suffer from unnecessary re-training which will add significant overhead to the classification process. On the other hand, late update may miss possible concept drift in data over time which affects the overall performance of the model.

**Dynamic update**. In this approach, as depicted in Figure 3.2, we update the model whenever there is a drift between the current data and previously seen training data. $R$ is a training window that builds the model, while $S$ is a sliding window that probes this model for any possible concept drift (i.e., model needs update). Algorithm 1 describes this dynamic update mechanism. We refer to this algorithm as BindDup (BIND Dynamic UPdate). BindDup starts by considering a portion of data as a training window to initialize the AdaBind model (lines 2 and 3). Then, the subsequent instances are considered within a

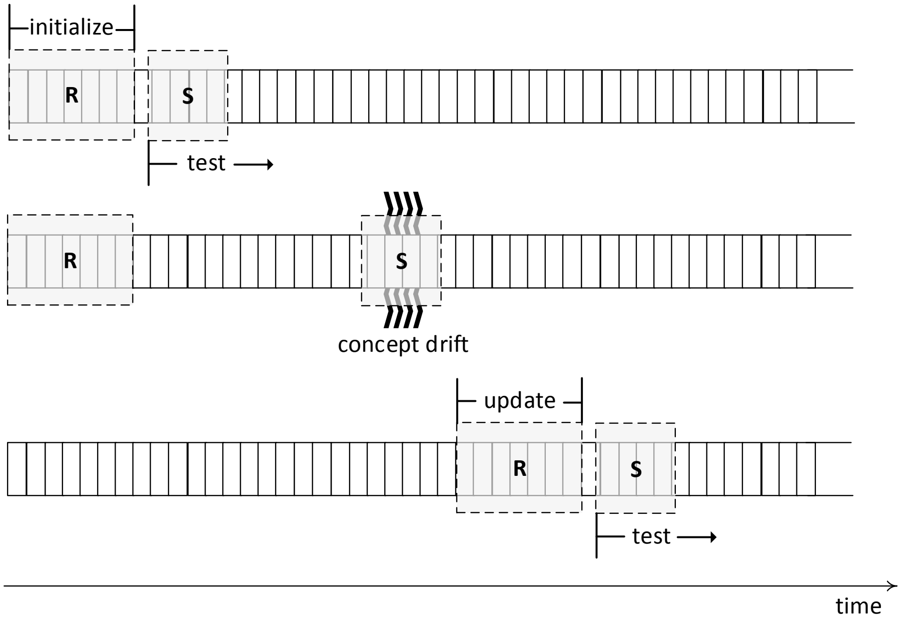


Figure 3.2: Illustration of AdaBind.

sliding window to validate the performance of this model over time (lines 5 and 6). If the accuracy drops below a predefined threshold (line 7), the initial AdaBind model becomes obsolete (i.e., concept drift) and the training window moves (line 8) to get new instances to re-train and update the model (lines 9 and 10). BindDup utilizes the AdaBind updated model to test incoming new data in a continuous fashion.

## 3.2 Evaluation

In this section, we present the empirical results of using BIND for Wfin and Afin, comparing it with other existing methods.

### 3.2.1 Datasets

We use two existing datasets for evaluating Wfin, one using HTTPS and the other using the Tor anonymity network, referred to as Https and Tor respectively. These datasets have been widely used in previous research on traffic fingerprinting. For Afin, we collect our own dataset from apps that use the HTTPS protocol.

```
Algorithm 1: BindDup
Data: Training Data: $TrainX$, Testing Data: $TestX$
Input: Training Window: $\mathcal{R}$, Sliding Window: $\mathcal{S}$, Threshold: $\mathcal{T}$
begin
    $\mathcal{F}^{train} \leftarrow extractFeatures(TrainX^{\mathcal{R}})$;
    $initializeModel(\text{AdaBind}, \mathcal{F}^{train})$;
    for $each\ \mathcal{S}$ do
        $\mathcal{F}^{test} \leftarrow extractFeatures(TestX^{\mathcal{S}})$;
        $accuracy \leftarrow validateModel(\text{AdaBind}, \mathcal{F}^{test})$;
        if $accuracy < \mathcal{T}$ then
            $move\ \mathcal{R}$;
            $\mathcal{F}^{train} \leftarrow extractFeatures(TrainX^{\mathcal{R}})$;
            $updateModel(\text{AdaBind}, \mathcal{F}^{train})$;
            $move\ \mathcal{S}$;
        end
    end
end
```

**Website Datasets**. The first dataset presented in [92], which we denote as Https, was collected while browsing websites using the HTTPS protocol along with a proxy server to imitate an anonymity network. The authors followed a ranking procedure to select the most accessed websites in their school department. The second dataset is described in [137]. This dataset is collected by capturing packets generated from a browser connected to the Tor anonymity network. We denote this dataset as Tor.

Https consists of 1000 websites with 200 traces each. For Wfin, we evaluate the closed-world setting by randomly picking a subset of these 1000 websites. For the open-world setting, we randomly select 30 websites as the monitored set, and the rest as the non-monitored one.

The other dataset (Tor) consists of two sets of traces. The first is a set of 100 websites that have 90 traces each. These websites were selected from a list of blocked websites by some countries. We use this for the closed-world experiments. The second set consists of 5000 websites that have one trace each. These websites were selected from Alexa's top websites [17]. In the open-world setting, we use the set of 100 websites as monitored, and

Table 3.2: Statistics for Website Fingerprinting datasets in the open-world setting.

| **Dataset** | **# of websites** | | **# of traces per website** |
|---|---|---|---|
| HTTPS [92] | Monitored | 30 | 70 |
| | Non-Monitored | 970 | 1 |
| TOR [137] | Monitored | 100 | 90 |
| | Non-Monitored | 5000 | 1 |

the set of 5000 websites as non-monitored. The summarized statistics of these datasets are provided in Table 3.2. These two datasets enable us to perform an unbiased comparison of BIND with other competing methods.

**App Dataset**. For AFIN, we evaluate BIND using a dataset that we collected by executing multiple Android apps on a Samsung Galaxy S device, running Android version 4.3.1. We randomly select about 30,000 apps from three different categories in *Google Play Store*. The categories include Finance, Communication, and Social. We refer to them as APP-FIN, APP-COMM, and APP-SOCIAL respectively. We then install and launch these apps on the phone which is connected to the Internet via a wireless router. Each trace per app is collected over a 30-sec period passively using a mirroring switch at the wireless router. Figure 3.3 illustrates this data collection setup. We filtered the captured traffic to contain packets from ports 80, 8080, and 443. We then identify apps that use only HTTPS data from the captured traces. These traces from such apps are then used to perform the closed-world and open-world AFIN. It is important to note that we uninstall each app as soon as we complete capturing a trace to avoid any background noise during further trace generation.

Similar to WFIN, multiple traces of apps are required to train a classifier in the closed-world and open-world settings. We use the APP-FIN dataset for performing the closed-world experiments as we capture multiple traces for each app. We only capture a single trace per app for APP-COMM and APP-SOCIAL to be used for the open-world experiments as the non-monitored set. The dataset statistics for the open-world setting are shown in Table 3.3. Note

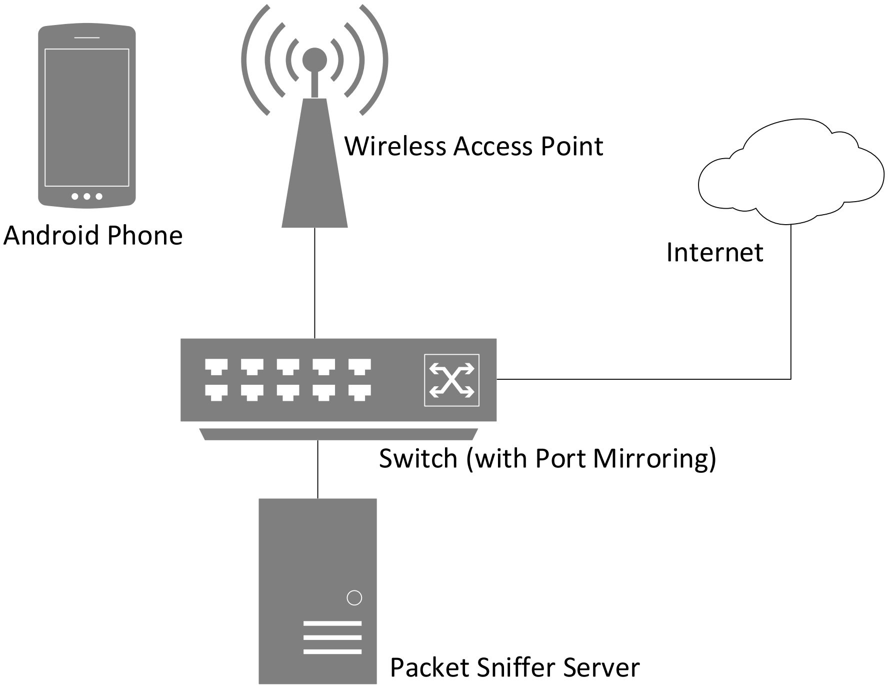


Figure 3.3: Illustration of the app trace data collection process

Table 3.3: Dataset statistics for App Fingerprinting in the open-world setting.

| **Category** | **# of apps** | | **# of traces per app** |
|---|---|---|---|
| App-Fin | Monitored | 30 | 20 |
| | Non-Monitored | 2238 | 1 |
| App-Comm | Non-Monitored | 1061 | 1 |
| App-Social | Non-Monitored | 1290 | 1 |

that in the closed-world setting, we only evaluate using apps from App-Fin. In the case of open-world, the monitored apps are considered only from App-Fin and the non-monitored apps are considered from all categories shown in Table 3.3.

While performing app selection for creating our dataset, we observed a few interesting statistics that would further motivate the problem of Afin. Figure 3.4 shows the percentage of apps that use HTTP and HTTPS data at launch in our initial set of $30,000$ apps. Ob-

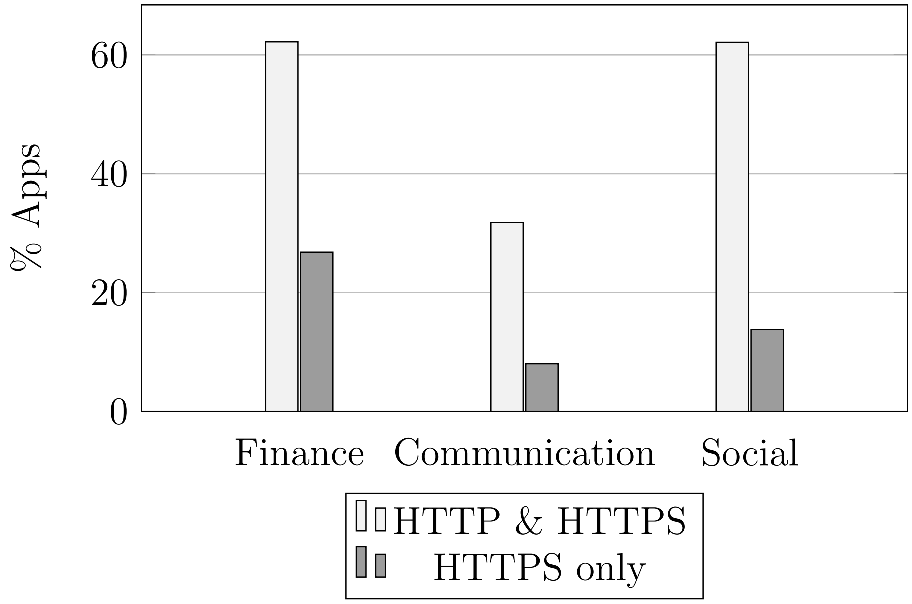


Figure 3.4: Empirical Statistics of Android Apps

serve that most apps use HTTP along with HTTPS while a sizable portion of apps use only HTTPS, for communication over the Internet. Furthermore, we obtained a list of IP addresses from HTTPS apps in each category We found a total of 1115 unique IP addresses for App-Fin, 820 for App-Comm, and 900 for App-Social. Additionally, each app connects to 3 different IP addresses on average over the whole dataset. This clearly indicates that the IP addresses found on HTTPS traffic overlap across apps, and do not provide sufficient information to identify the app generating a trace by itself.

### 3.2.2 Experimental Settings

Using these datasets, we perform our analysis on both closed-world and open-world settings. For a comparative evaluation, we consider existing traffic analysis techniques developed for Wfin. These techniques are listed in Table 3.4. The table details the features and classifiers used for our evaluation in both Closed-world (Closed) and Open-world (Open) settings. For brevity of representation, we term websites (in the case of Wfin) or apps (in the case of Afin) as *entities*.

Table 3.4: Traffic Analysis Techniques used for the evaluation

| Data Analysis Method | Setting Type | Features | Classifier |
|---|---|---|---|
| VNG++ [57] | Closed | Uni-Burst Size & Count<br>Total Trace Time<br>Uplink/Downlink Bytes | Naïve Bayes |
| P [107] | Closed | Uni-Burst Size & Count<br>Packet Size<br>Packet Ordering | SVM |
| OSAD [138] | Closed | Cell Traces | Optimized SVM |
| BindSvm * | Closed | BIND features:<br>Bi-Burst Size & Time<br>Uni-Burst Size, Time, & Count<br>Packet Size | SVM |
| Wknn [137] | Open | Same features as P | Weighted k-NN |
| BindWknn * | Open | BIND features:<br>Same features as BindSvm | Weighted k-NN |
| BindRF * | Open | BIND features:<br>Same features as BindSvm | Random Forest |

*new approaches introduced in this work

**Closed-world**. Using BIND features, we use a support vector machine classifier (SVM) in the closed-world setting. We refer to this approach as BindSvm as shown in Table 3.4. In our experiments, we use a publicly available library called LibSVM [39] with a Radial Basis Function (RBF) kernel having the parameters $Cost = 1.3 \times 10^5$ and $\gamma = 1.9 \times 10^{-6}$ (following recommendations in [107]). We consider varied subsets of entities to evaluate the feature set. Particularly, we use 16 randomly selected traces per entity (class) for training a classifier, and 4 randomly selected traces per entity for testing. For each experiment, we chose the number of selected (monitored) entities in $\{20, 40, 60, 80, 100\}$.

**Open-world**. For the open-world scenario, as discussed in §3.1.2, we use two classification methods with the BIND features. First, we use the weighted $k$-NN mechanism proposed in [137]. Specifically, we use $k = 1$ since it is shown to produce the best results on the Tor dataset in [137]. We denote this method as BindWknn as shown in Table 3.4. Furthermore, we also use the Random Forest classifier with BIND features, denoted as BindRF in Table 3.4. We use a set of 100 weak learners to form an ensemble of decision trees. We use the scikit-learn [110] implementation for our evaluation. The complete set of monitored and non-monitored traces mentioned in Tables 3.2 and 3.3 are considered for evaluation.

**Evaluation Measure**. The results of the closed-world evaluation are measured by computing the average accuracy of classifying the correct class for all test traces. We randomly select traces from the corresponding dataset and repeat each experiment 10 times with different entities and traces. Average accuracy is computed across these experiments. In the open-world evaluation, we measure the true positive rate (TPR) and false positve rate (FPR) of the binary classification. These are defined as follows: $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$. Here, $TP$ (True Positive) is the number of traces which are monitored, and predicted as monitored by the classifier. $FP$ (False Positive) is the number of traces which are non-monitored, but predicted as monitored. $TN$ (True Negative) is the number of traces which are non-monitored and predicted as non-monitored. $FN$ (False Negative) is the number of traces which are monitored, but predicted as non-monitored. We perform a 10-fold cross validation on each dataset, which gives randomized instance ordering.

In order to evaluate the performance of BIND against defenses discussed in §2.1.1, we consider one of the most sophisticated and complex defenses, Traffic Morphing (TM). Furthermore, to evaluate BIND against existing approaches, for the open-world setting on the Tor dataset, we apply the *Tamaraw* defense mechanism, designed specifically for Tor, as evaluations in [137, 34] show that this defense performs exceptionally well against Tor.

### 3.2.3 Experimental Results

We use the notations given in Table 3.2 and Table 3.3 to denote the Wfin and Afin datasets respectively.

**Traffic Analysis**.

We first perform Wfin and Afin experiments in the closed-world setting. Here, a set of randomly chosen entities are classified using competing methods. We vary the set size from 20 to 100. The results are presented in Table 3.5 using the Https and Tor datasets for Wfin, and the App-Fin dataset for Afin. In some cases, we can see BindSvm performs comparatively closer to or lower than the other competing methods, while outperforming them in other cases. For example, with 80 websites considered, the average accuracy of BindSvm (BIND using SVM) on the Https dataset is 88.4%. This is marginally greater than 88.3% obtained from the P method. Similarly in Afin, BIND resulted in an average accuracy of 87.8%, compared to a marginally better accuracy of 88% resulting from the P method. Moreover for the Tor dataset, it is not surprising that the OSAD method performs the best in all experimental settings since it uses a distance measure that is specifically applicable to Tor data. In the closed-world setting, most methods listed in Table 3.4 use features that overlap or hold similar information about the class label. Some features provide better characteristic information about the class than others. When selecting the websites at random during evaluation, each classification method outperforms the other in a few cases depending on the data selected for training and testing. Therefore, the average accuracy across these are marginally superior than others in most of the cases.

However, the greatest impact of using BIND features can be observed in the more realistic open-world setting. Table 3.6 presents the results of the open-world setting for all competing methods. Here, a high value of TPR and a low value of FPR are desired. As mentioned earlier in this section, we use two types of classifiers while using the BIND features, i.e., BindWknn and BindRF. In the case of Wfin, it is clear that the TPR for both BindWknn and BindRF is significantly better compared to that of Wknn. For instance, consider the result of the Tor dataset. The TPR obtained from BindWknn method is 90.4% and that obtained from BindRF is 99.8%, as compared to 89.6% of Wknn. The BindRF method

Table 3.5: Accuracy (in %) of the closed-world traffic analysis for website fingerprinting (HTTPS and Tor) and app fingerprinting (App-Finance) without defenses.

| Dataset | | Https | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # websites | 20 | 87.5 | 93.5 | **94.1** | 94.0 |
| | 40 | 83.8 | **91.4** | 89.0 | 91.3 |
| | 60 | 85.2 | **92.3** | 91.0 | 91.6 |
| | 80 | 81.6 | 88.3 | 87.7 | **88.4** |
| | 100 | 82.4 | **90.3** | 89.2 | 90.0 |

| Dataset | | Tor | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # websites | 20 | 78.0 | 85.3 | **90.0** | 86.5 |
| | 40 | 67.8 | 77.6 | **92.1** | 80.9 |
| | 60 | 63.7 | 77.0 | **86.7** | 79.5 |
| | 80 | 62.9 | 75.8 | **89.5** | 77.6 |
| | 100 | 56.9 | 71.4 | **85.7** | 73.9 |

| Dataset | | App-Fin | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # websites | 20 | 81.3 | 92.0 | 88.7 | **93.3** |
| | 40 | 73.6 | **88.3** | 85.1 | 87.3 |
| | 60 | 72.3 | 86.5 | 83.6 | **86.7** |
| | 80 | 72.8 | **88.0** | 79.6 | 87.8 |
| | 100 | 66.0 | 83.1 | 77.2 | **84.2** |

outperforms Wknn even though the Wknn method was specifically designed for high quality results on this dataset. In terms of FPR, BindWknn method performs better than Wknn.

A more significant result can be observed in the open-world setting of Afin. Both TPR and FPR are greatly improved with the BindWknn and BindRF methods on all app fingerprinting datasets, as indicated in Table 3.6. For example, the average TPR resulting from BindWknn method on the App-Fin dataset is 78%, compared to the average TPR

Table 3.6: TPR and FPR (in %) of open-world setting for website fingerprinting (HTTPS and Tor) and app fingerprinting (App-Finance, App-Communication and App-Social) without defenses.

| Dataset | Score | Method | | |
|---|---|---|---|---|
| | | Wknn | BindWknn | BindRF |
| HTTPS | TPR | 73.0 | 91.0 | **98.2** |
| | FPR | 29.0 | **16.0** | 18.3 |
| Tor | TPR | 89.6 | 90.4 | **99.8** |
| | FPR | 2.1 | **1.9** | 3.4 |
| App-Fin | TPR | 53.0 | 78.0 | **88.5** |
| | FPR | 10.0 | 7.0 | **1.9** |
| App-Comm | TPR | 64.0 | 82.0 | **93.1** |
| | FPR | 5.0 | 2.0 | **0.8** |
| App-Social | TPR | 61.0 | 75.0 | **92.1** |
| | FPR | 5.0 | 2.0 | **0.1** |

of 53% reported by the Wknn method. Similarly, the average FPR of 7% reported by the BindWknn method is better than the average FPR of 10% resulting from the Wknn method. This clearly demonstrates the effectiveness of using BIND features for traffic analysis in Afin as well.

Moreover, the average TPR and FPR are largely improved when using the BindRF method. It is important to note that while using monitored and non-monitored traces from different categories, i.e., in the case of the App-Comm and App-Social datasets, the average TPR and FPR are better when compared with the results from the App-Fin dataset where the monitored and non-monitored sets are from the same category. Especially, a low FPR of less than 1% is obtained on these datasets. This indicates that there exist differentiating characteristics between apps from different categories as expected.

The open-world setting is a binary classification problem. Features extracted and the classifier used for determining class boundary significantly impact the TPR and FPR results. In the case of Wknn, the monitored entities are made as close as possible via an iterative weighing mechanism. When using BIND features, we count unique bi-burst tuples. These

Table 3.7: Accuracy (in %) of closed-world website fingerprinting on HTTPS dataset with Traffic Morphing.

| Dataset | | Https | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # websites | 20 | 79.1 | 76.0 | 86.6 | **87.5** |
| | 40 | 74.4 | 73.6 | 79.1 | **82.6** |
| | 60 | 68.4 | 68.0 | 74.6 | **79.7** |
| | 80 | 61.2 | 65.1 | 69.8 | **75.2** |
| | 100 | 64.1 | 60.6 | 67.4 | **73.2** |

provide additional features to the existing feature set of uni-burst used in [137]. These features aid the weighing mechanism by bringing out more relevant dimensions, suppressing less relevant ones in BindWknn. Random forest uses decision trees that divide the feature space effectively using the information gain measure rather than the Euclidean distance measure used by the $k$-NN method. An ensemble of such classifiers typically reduces bias and variance during training, compared to a single classifier [32]. Consequently, this classifier, along with BIND features, shows superior performance in TPR results.

**Traffic Analysis with Defenses for Website Fingerprinting**.

We now consider the evaluation of BIND in an adversarial environment, specifically for Wfin, similar to relevant studies in this area. Here, we apply a defense mechanism to trace packets for with the aim of reducing effectiveness of a fingerprinting attack (classifier), and study the robustness of BIND when used by an attacker against such defenses.

With defense mechanisms such as Traffic Morphing (TM) used by defenders to thwart classifiers, the features extracted from the data play an important role while performing an adversarial attack. Table 3.7 shows the average accuracy obtained on the Https dataset when TM is applied on all websites in the closed-world setting. It is important to note that for every experiment, we apply TM by selecting a random target website. BindSvm performs with significant improvement in average accuracy on all experiment settings compared to

Table 3.8: Accuracy (in %) of closed-world website fingerprinting on Tor dataset with Traffic Morphing.

| Dataset | | Tor | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # websites | 20 | 77.8 | 81.3 | 68.5 | **82.3** |
| | 40 | 66.6 | 74.9 | 58.5 | **77.6** |
| | 60 | 61.0 | 70.3 | 51.2 | **72.3** |
| | 80 | 58.7 | 67.6 | 42.6 | **69.9** |
| | 100 | 65.8 | 65.8 | 39.3 | **68.7** |

Table 3.9: TPR and FPR (in %) in open-world setting for website fingerprinting on HTTPS dataset with Traffic Morphing, and Tor dataset with Tamaraw.

| Dataset | Score | Method | | |
|---|---|---|---|---|
| | | Wknn | BindWknn | BindRF |
| HTTPS | TPR | 74.0 | 82.0 | **98.5** |
| | FPR | 29.0 | **24.0** | 72.4 |
| Tor | TPR | 2.7 | 2.7 | **100.0** |
| | FPR | 0.0 | 0.0 | 0.0 |

other competing methods. For instance, BindSvm reports an average accuracy of 73.2% with 100 closed-world websites. This is better than the average accuracy of 67.4% reported by OSAD, which is the second highest accuracy in this setting.

Similarly, Table 3.8 shows the average accuracy obtained on the Tor dataset when TM is applied on all websites. From the table, we can observe that the BindSvm method outperforms other methods.

In the open-world setting, we apply TM on the Https dataset. The TPR and FPR results are shown in Table 3.9. The BindRF method reports an average TPR of 98.5%. However, it also reports an undesirable high FPR of 72.4%. This high FPR indicates that more false alarms are reported by this classifer. In contrast, the BindWknn method reports 82% average TPR, which is greater than 74% reported by the Wknn method. Moreover, it also reports the lowest average FPR of 24% on the dataset. This shows the effectiveness of this defense on Https dataset. It also indicates that BIND features aid the weighted $k$-Nearest Neighbors algorithm to classify more accurately than merely using Uni-Burst features.

Table 3.9 also shows the average TPR and FPR obtained on the Tor dataset when using competing methods while applying the Tamaraw defense mechanism. In the case of methods that use the weighted $k$-NN algorithm, i.e., Wknn and BindWknn, we obtain a low TPR of 2.7%. This result agrees with that reported by Wang et al. [137] who use the Wknn method on the same dataset. Yet rather remarkably, we obtain an average TPR of 100% and an average FPR of 0% from the BindRF method. This highly accurate classification is a result of a combination of BIND features and random forest classifiers, where features of monitored websites are morphed by Tamaraw. Moreover, the morphing scheme involves changing packet time and size values. In the BIND feature set, we consider quantized tuple counts as features (Bi-Burst), along with other Uni-Burst features. Changing the packet time information by a constant may not successfully destroy characteristic information in a trace. Furthermore, the tree structure of weak learners (decision trees) in the random forest classifier aids in a better classification as illustrated in Table 3.6. This combination provides a perfect classification of the morphed dataset in this case.

**Traffic Analysis with Defenses for App Fingerprinting**.

We evaluated our proposed data analysis technique in an adversarial environment for Wfin. A user may visit any website s/he desires using an anonymity network to protect against surveillance from external adversaries on the network. However, this case may not be directly applicable to Afin. An app is typically deployed on a well-recognized app store such as Google play. These apps typically may not provide users an ability to configure network traffic to use a user-desired anonymity network such as Tor. They use the default network configuration set on the host device. However, the goal of an adversary in Afin might be to identify vulnerable apps or malware installed on a device in order to perform attacks such as privilege escalation [47] targeted on the user. Therefore, we perform experiments on app traffic when defenses such as TM are applied to reduce chances of app identification.

We assume that defenses like packet padding could be applied to app traffic and evaluate the data analysis techniques when the padding technique of TM is used. Instead of morphing the packet distribution of a website with another one in the case of WFIN, packet distribution of an app is morphed to appear similar to another app. Table 3.10 shows the accuracy of this scenario in the closed-world setting on the APP-FIN dataset with the morphed traffic. Similar to the results in Table 3.7, the average accuracy reported by BINDSVM method is higher than other competing methods in most cases. Results of the open-world setting are given in Table 3.11. Clearly, BIND performs better than other competing methods. A low FPR with a high TPR are reported by the BINDRF method compared to WKNN. Another important observation is that the TPR resulting from the APP-FIN dataset is lower than other categories. This shows that intra-category differentiating characteristic features may be affected more than inter-category features while using morphing techniques. Overall, these results reinforce our hypothesis that BIND methods provide good characteristic properties from traces which can be used for better entity identification.

However, we realize that TPR is low when compared to that of the WFIN datasets in Table 3.9. The network signature of an app is different from that of a website. Apps use the Internet to connect to services and communicate minimal amount of data as necessary. In contrast, browsing a website could potentially generate a larger network trace since all the components of a website have to be downloaded to the browser. A smaller network footprint may affect the fingerprinting process.

**Execution Time**.

Figure 3.5 shows the execution time for experiments in Table 3.5 on the TOR dataset, where OSAD outperforms the other methods. The $x$-axis in the figure represents the number of websites, while the $y$-axis represents the execution time (in seconds) in logarithmic scale

Table 3.10: Accuracy (in %) of closed-world app fingerprinting while using Traffic Morphing.

| Dataset | | App-Fin | | | |
|---|---|---|---|---|---|
| Method | | VNG++ | P | OSAD | BindSvm |
| # Apps | 20 | 71.5 | 68.3 | **77.6** | 77.0 |
| | 40 | 58.3 | 59.1 | 61.0 | **67.0** |
| | 60 | 50.2 | 51.7 | 56.0 | **59.2** |
| | 80 | 44.6 | 44.8 | 49.3 | **53.8** |
| | 100 | 42.9 | 42.1 | 49.2 | **50.4** |

Table 3.11: TPR and FPR (in %) of open-world app fingerprinting while using Traffic Morphing.

| Dataset | Score | Method | | |
|---|---|---|---|---|
| | | Wknn | BindWknn | BindRF |
| App-Fin | TPR | 16.0 | **22.0** | 20.5 |
| | FPR | 14.0 | 13.0 | **5.1** |
| App-Comm | TPR | 41.0 | 46.0 | **66.8** |
| | FPR | 7.0 | 5.0 | **4.1** |
| App-Social | TPR | 67.0 | 68.0 | **68.6** |
| | FPR | 5.0 | 4.0 | **1.2** |

(base 10). The execution times of VNG++, P, and BindSvm classifiers are low compared to that of OSAD. For instance, with 60 websites, OSAD takes 2340 sec while VNG++, P, and BindSvm take 25, 31, and 39 sec, respectively. This shows how OSAD incurs extra overhead which may render it impractical in some scenarios. In the case of open-world setting, we observed that Wknn and BindWknn (> 30 mins) took significantly longer time than BindRF (< 60 secs), due to weight computations. Yet, BindRF outperformed BindWknn (or Wknn) in Table 3.6 and Table 3.11 on most cases.

**Base Detection Rate Analysis**.

In this section, for the open-world scenario, we study the effect of BIND in a more realistic scenario which considers the probability of a client visiting a website or using an

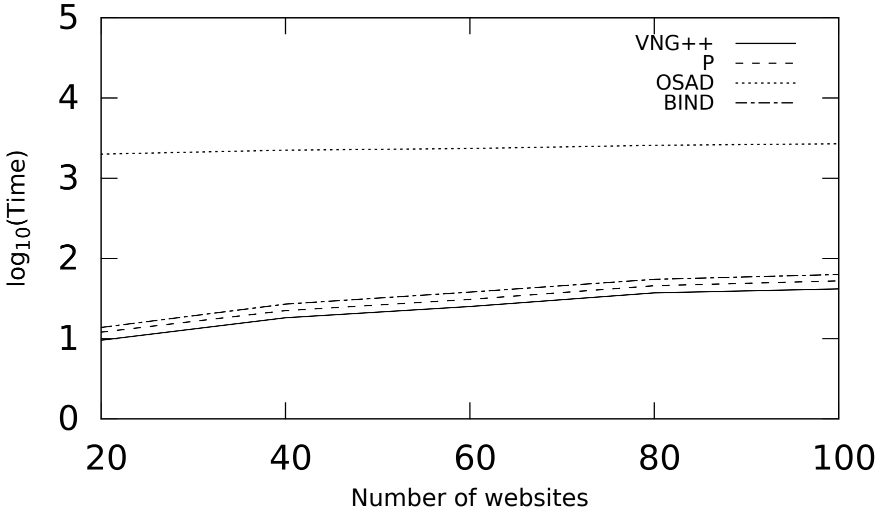


Figure 3.5: Running time (in seconds) for the experiments in Table 3.5, on TOR dataset. Note that time axis is in logarithmic scale to the base 10.

app in the monitored set, referred to as *prior* or *base rate*. This has been recently raised as a concern in the research community in Wfin [75].

The *base detection rate* (*BDR*) is the probability of a trace being actually monitored, given that the classifier predicted (detected) it as monitored. Using the Bayes Theorem, *BDR* is formulated as:

$$P(M|D) = \frac{P(M)\ P(D|M)}{P(M)\ P(D|M) + P(\neg M)\ P(D|\neg M)]}, \tag{3.2}$$

where $M$ and $D$ are random variables denoting the actual monitored and the detection as monitored by the classifier, respectively. We use TPR and FPR, from Table 3.6, as approximations of $P(D|M)$ and $P(D|\neg M)$, respectively.

Table 3.12 presents the *BDR* computed for the open-world classifiers. We assume $P(M)$ or *prior* is calculated as the size of the monitored set divided by the world size (the size of the monitored and non-monitored set), i.e., $P(M) = \frac{|monitored|}{|monitored|+|non-monitored|}$. The table shows the *BDR* for the different datasets.

Although BIND methods ourperform other methods, as the results in Table 3.12 indicate, the numbers expose a practical concern in fingerprinting research: despite having high

Table 3.12: Base detection rate percentages in the open-world setting.

| Method | Dataset | | | | |
|---|---|---|---|---|---|
| | HTTPS | TOR | APP-FIN | APP-COMM | APP-SOCIAL |
| WKNN | 7.4 | 46.4 | 6.7 | 27.14 | 22.5 |
| BINDWKNN | **15.3** | **49.2** | 13.1 | **54.4** | 47.1 |
| BINDRF | 14.6 | 37.4 | **38.7** | 25.3 | **68.6** |

accuracy values, typical fingerprinting detection methods are rendered ineffective when confronted with their staggeringly low base detection rates. This is in part due to their intrinsic inability to eliminate false positives in operational contexts.

However, we follow a similar approach to the results of a recent study [55] in Anomaly Detection to approximate the prior for the specific scenario of a *targeted user*. The study assumes a model with a determined attacker leveraging one or more exploits of known vulnerabilities to penetrate a typical organization's internal network, and approximates the *prior* of a directed attack to 6% (using threat statistics from 2011). Similarly, we model a targeted user where the *prior* increases given other estimates. For example, consider a government tracking a suspicious user (targeted) with a prior knowledge or estimate that increases the probability of such user visiting certain websites or using certain apps (monitored) or carrying out specific online activities (e.g. suspicious activities).

Figure 3.6 depicts this process using TPR and FPR obtained from Table 3.6 with the TOR dataset. In this figure, we show the effect of increasing the *prior*, starting from 2% which is the actual $P(M)$. Similarly, Figure 3.7 shows the effect of increasing this *prior* on the same dataset while applying the Tamaraw defense, using TPR and FPR from Table 3.9. The figures show how increasing the *prior* improves the *BDR* significantly. As our confidence about the *prior* raises, the corresponding *BDR* increases to practical values.

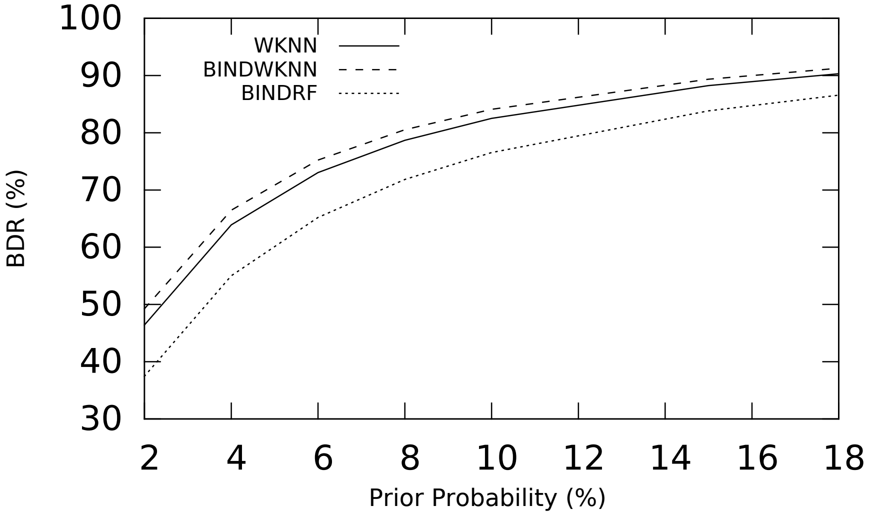


Figure 3.6: Increasing prior effect on BDR using the Tor dataset for open-world without defense.

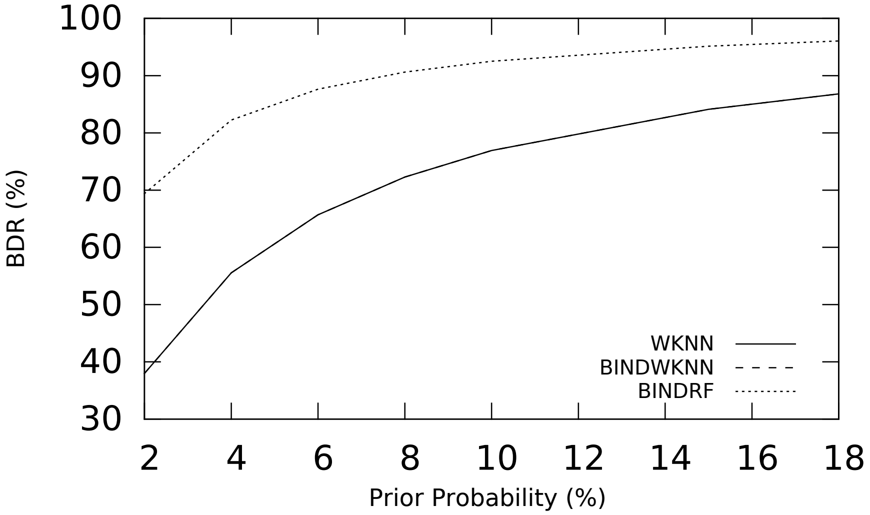


Figure 3.7: Increasing prior effect on BDR using the Tor dataset for open-world while applying the Tamaraw defense.

**Adaptive fingerprinting**.

We now present the experimental results of adaptive learning (AdaBind) discussed in §3.1.2. The experiment in Figure 3.8 shows the effect of concept drift on the model, and the BindDup dynamic update (re-training) process in Wfin. Here, the $x$-axis represents time (in days) and the $y$-axis represents accuracy (%). We consider 20 websites from the HTTPS dataset with a training window of 16 traces per website for training the ADABIND model ($R = 16$, starting at day 1 to day 16). Then, a sliding window of 4 traces (starting at day 17) per website is considered for validating this model by testing its accuracy.

It is important to note the training and testing data are collected at different times, under different experimental settings. As the 4-day validating window slides, if the accuracy drops below a certain threshold (85% in this experiment), the model becomes obsolete. So, we re-train the model at that point (i.e., at day 33, 94, 119, and 148 as shown in the figure). This dynamic re-training mechanism improves the accuracy, resulting in values above the assigned threshold. The average accuracy of this approach is 92.6%.

Figure 3.8 also shows how the accuracy drops to low values if no update is considered. In this experiment, we train the model once in the beginning and use the 4-day sliding window to validate test traces. The resulting average accuracy of this static learning method is 76%, which illustrates the need for re-training the model to adapt for possible data drifts over time.

In addition, Figure 3.8 shows the same experiment where we apply the BINDFUP fixed update approach by re-training the model every 24 days instead of the dynamic update in BINDDUP. We use the same 4-day validating window as before. The figure shows how the model becomes more accurate and stable. Yet, this results in an extra training overhead due to unnecessary updates. The average accuracy of this approach is 93.3%, which is marginally better from the average accuracy of BINDDUP (92.6%). The number of updates in this experiment for BINDFUP is 8, which is twice as many as the number of updates in the dynamic update approach (BINDDUP). As discussed earlier in §3.2.3, a classifier may have large execution time, resulting in significantly large re-training cost. This shows the trade-off between performance and cost of re-training the model.

To see the effect of the training window ($R$), Figure 3.9 shows the BINDDUP dynamic update experiments when varying the value of $R$ in the range $\{4, 8, 12, 16, 20\}$. If $R$ is small,

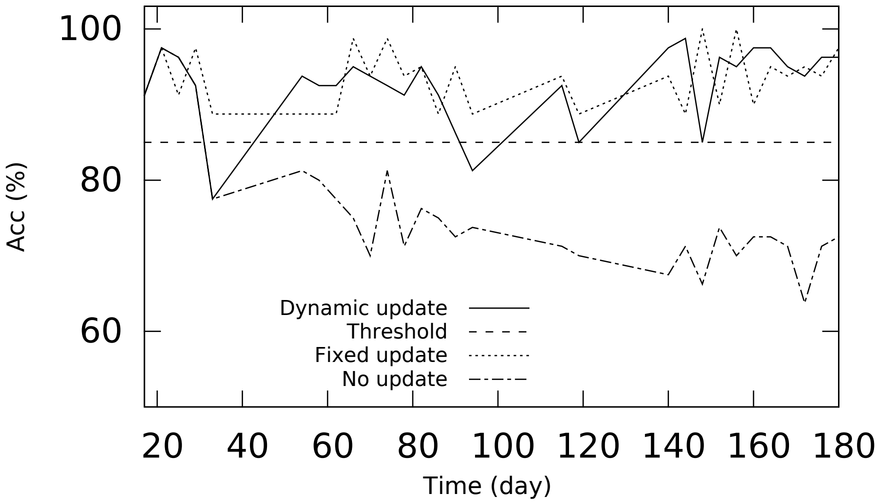


Figure 3.8: Adaptive Learning.

Table 3.13: Average accuracies and number of updates with different values of the training window (R)

| R | 4 | 8 | 12 | 16 | 20 |
|---|---|---|---|---|---|
| Average accuracy (%) | 86.6 | 89.3 | 89.9 | **92.6** | 91.7 |
| Number of updates | 10 | 7 | 5 | 4 | 2 |

the number of training instances may not be enough to build a good model, and may lead to frequent updates. On the other hand, choosing large values of $R$ incurs extra training overhead and may cause the model to miss some drifts in data. Table 3.13 shows the average accuracies and number of updates/re-trains for the experiments shown in Figure 3.9. When $R$ increases, the average accuracy improves to a certain level, and then goes down. We obtained the best results when $R = 16$ with a moderate number of updates (i.e., 4 re-trains).

For the previous experiments which used SVM, we observed similar conclusions for the other datasets. We did not include them because of space limitations. In general, the adaptive learning algorithm can be applied to any classification approach.

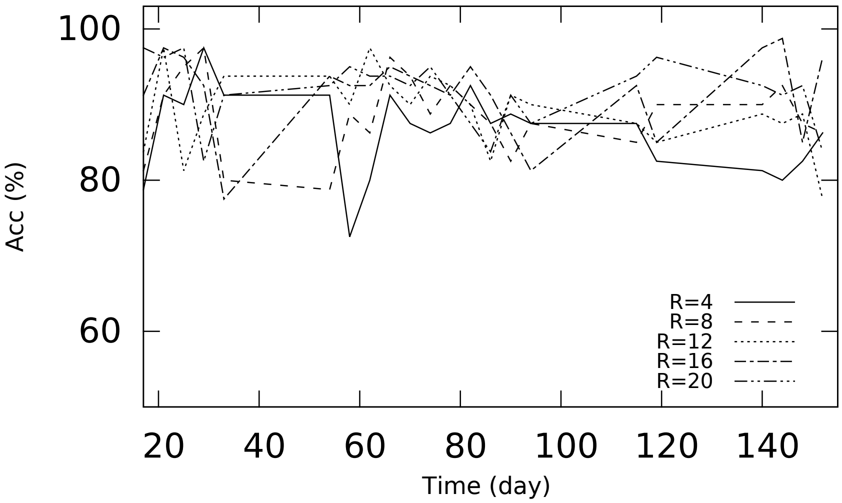


Figure 3.9: Dynamic update with different values of the training window (R)

### 3.3 Discussion

We introduced BIND, a new feature extraction and classification method for data analysis on encrypted network traffic with two case studies including Wfin and Afin. We discuss the challenges and limitations, resulting from the assumptions in our evaluation, as well as future work.

A study in Wfin [75] describes the effects of various assumptions on the evaluation results. Major assumptions include single-tabbed browsing or absence of other background noise, small time gap (or freshness) in data collection between training and test set, page load parsing, and replicability. Recent studies [139, 64] tried to address these issues by evaluating classifiers in conditions with relaxed assumptions. In particular, a long time gap (or staleness) in data collection between training and testing sets can have a significant impact on classifier accuracy. This limitation is true for the BIND approach as well since similar base features that are affected with time, i.e., packet statistics such as length, sequence, and timing are used. The challenge can be addressed by periodically training a new model with fresh training data as introduced in this dissertation using AdaBind which models fingerprinting in an adaptive manner.

# CHAPTER 4

# FINGERPRINTING USING VECTOR SPACE REPRESENTATIONS[1]

## 4.1 P2V Approach

In this chapter, we present the details of our Packet to Vector (P2V) approach introduced in Chapter 1 and explain how we utilize word vector representations to improve the website fingerprinting attack.

### 4.1.1 Concept

Previous studies [57, 71, 107, 137] on website fingerprinting used features such as time taken to load the webpage, packet size with direction of data transmission, packet order, and the length of combined sequential packets in the same direction, called burst (for instance, see Uplink burst in Figure 4.1). These features are extracted from a trace of network traffic belonging to a single website. New features are then created by bucketizing the transmission length and counting the frequencies within each bucket [107]. Therefore, each trace of a webpage would have a large number of features. If $m$ is the total number of features, then each trace can be seen as an $m$-dimensional vector. We will see how this dimensionality issue is to be solved by the P2V model where a low $d$-dimensional vector is produced and used for classification. A class label (i.e. webpage name) is assigned to this trace.

In this work, we take a new packet to vector (P2V) [13] approach to improve this attack. We show how we model website fingerprinting using word vector representations. More specifically, we use the GloVe model described in §2.2.1. The GloVe model uses the context-counting model which leverages statistical counts by training on elements in a word-word

[1] This chapter contains material previously published as: ©2015 IEEE. Reprinted, with permission, from K. Al-Naami, G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan and B. Thuraisingham. "P2V: Effective Website Fingerprinting Using Vector Space Representations." IEEE Symposium Series on Computational Intelligence, December 2015. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

co-occurrence matrix. Each element in this matrix tabulates the number of times word $i$ co-occurs in the context of word $j$. In the TCP protocol, each packet transmission affects the following packet transmission. TCP uses flow control mechanisms such as window size and scaling, ACK packets and other methods to ensure safe arrivals of packets in both directions. This means there is a dependence between consecutive packet flows in a TCP connection. We will shortly explain how we construct the corpus by going through packets in sequence in order to build a packet-packet co-occurrence matrix which guarantees to give us meaningful counts for each trace.

### 4.1.2 PORDs

The GloVe model takes a text corpus as input and produces a vector for each word in the vocabulary. In website fingerprinting, as depicted in Figure 4.1, all we have is just trace packets collected while downloading websites. We need a mechanism to translate these packets into text tokens. We call these tokens *PORDs* (for Packet wORDs). PORDs are extracted from the sequences of packets. Unlike previous studies on website fingerprinting which ignored the ACK packets as they were considered noise, our P2V model does use the ACK packets to generate PORDs. In the evaluation section, we show how the ACK packets enrich the vocabulary and produce better results. For ease of analysis, we organize generating PORDs into the following categories.

**Packet Length PORDs**.

For each packet, we take the packet length in bytes and construct the packet PORDs. We consider both uplink and downlink directions. An uplink (or *Tx*) packet PORD with a length $l$ is different than a downlink (or *Rx*) packet PORD with the same length $l$.

**Uni-Burst Size PORDs**.

Burst (or Uni-Burst) consists of consecutive packets in the same direction. As illustrated in Figure 4.1, we call the burst going from a user to a server an uplink or *Tx burst* and the burst coming from a server to a user a downlink or *Rx burst*. Burst size is the summation of all of its packet sizes. We take each uni-burst size as a PORD. We also consider the direction in this category as well. We bucketize as this gives us best results as to be shown in the evaluation.

**Uni-Burst Time PORDs**.

Packet time is the departure/arrival timestamp in uplink/downlink direction. This is measured at the client side by the eavesdropper (attacker). Burst time is the difference between the last packet and first packet times (the time it takes the burst packets to get transmitted/received by the client in any direction). A PORD is constructed here from each Tx/Rx uni-burst time.

**Uni-Burst Count PORDs**.

Uni-Burst count is the total number of packets contributing in the burst. We take each Tx/Rx uni-burst count as a PORD.

**Bi-Burst Size PORDs**.

Bi-Burst is the sequence of two adjacent bursts. As shown in Figure 4.1, Rx-Tx-Burst is the combination of downlink and uplink consecutive bursts. We take the two sizes of each of the two bursts as a new PORD. Direction is considered here as well. Tx-Rx-Burst size is different than Rx-Tx-Burst size. We use bucketizing as above.

**Bi-Burst Time PORDs**.

This set of PORDs is similar to the Bi-Burst size PORDs approach described above but we take the two time differences in each of the adjacent bursts.

### 4.1.3 POCUMENTs

As described in §2.2.1, the primary source of information is the word co-occurrences matrix. Running through words (or PORDs) in the documents, we build statistical counts of

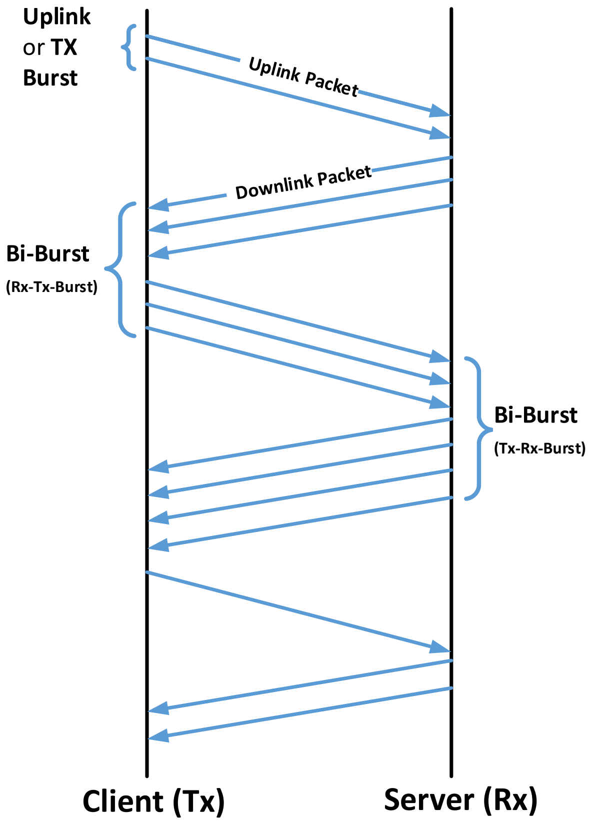


Figure 4.1: Sequence Diagram between Client (Tx) and Server (Rx). Packet, uni-burst and bi-burst transmissions between two ends are illustrated.

the number of times any two words co-occur together in a context window. In §4.1.2, we discussed how to generate PORDs. In this section, we show how to organize these PORDs in POCUMENTs (short for Packet dOCUMENTs). The juxtaposition of the PORDs in our POCUMENTs is important as GloVe captures useful statistics specified by that. Each trace (webpage load) is considered as a POCUMENT.

For a single pass in each trace used to train the model, we run through packets in order of departure/arrival from/to client side to construct the POCUMENT. For the purpose of illustration, in Figure 4.1, we run through packets from top to bottom and buffer all PORDs we will use. We insert the PORDs in each POCUMENT in an order described as follows (notice that this is the best order after trying multiple combinations).

- We first insert all Packet Length PORDs.
- Second, we consider all Uni-Burst Size PORDs.
- Third, Uni-Burst Time PORDs are inserted.
- Then, we put all Uni-Burst Count PORDs.
- Next, we take all Bi-Burst Size PORDs.
- Finally, Bi-Burst Time PORDs are considered.

We insert the above PORDs in the order of their appearances in the trace.

### 4.1.4 Example

We give an example to clarify our approach. Figure 4.2 depicts a website trace where packet sequences between Tx and Rx are shown. Each packet in the figure has size $s$ in bytes and time $t$. Time $t$ is the number of seconds since Epoch (1 January, 1970). Times shown in this example are for illustration purposes only. We set the time for the first packet in the trace to zero to have it as a reference. Figure 4.2 shows a uni-burst example of size 500 (200 plus 300) and time difference of 10 (10 minus 0). Figure 4.2 illustrates some PORDs constructed by the P2V model.

### 4.1.5 Classification

The input to GloVe is the PORPUS (for Packet cORPUS) which is a collection of POCUMENTS used for training the GloVe model. GloVe produces word (PORD) real-valued vectors. Now we can use these PORD vectors as features in website fingerprinting classification. For each trace (from training set or testing set), we construct a *Trace Vector* by averaging all PORD vectors in that trace. It is worth mentioning that the testing set traces

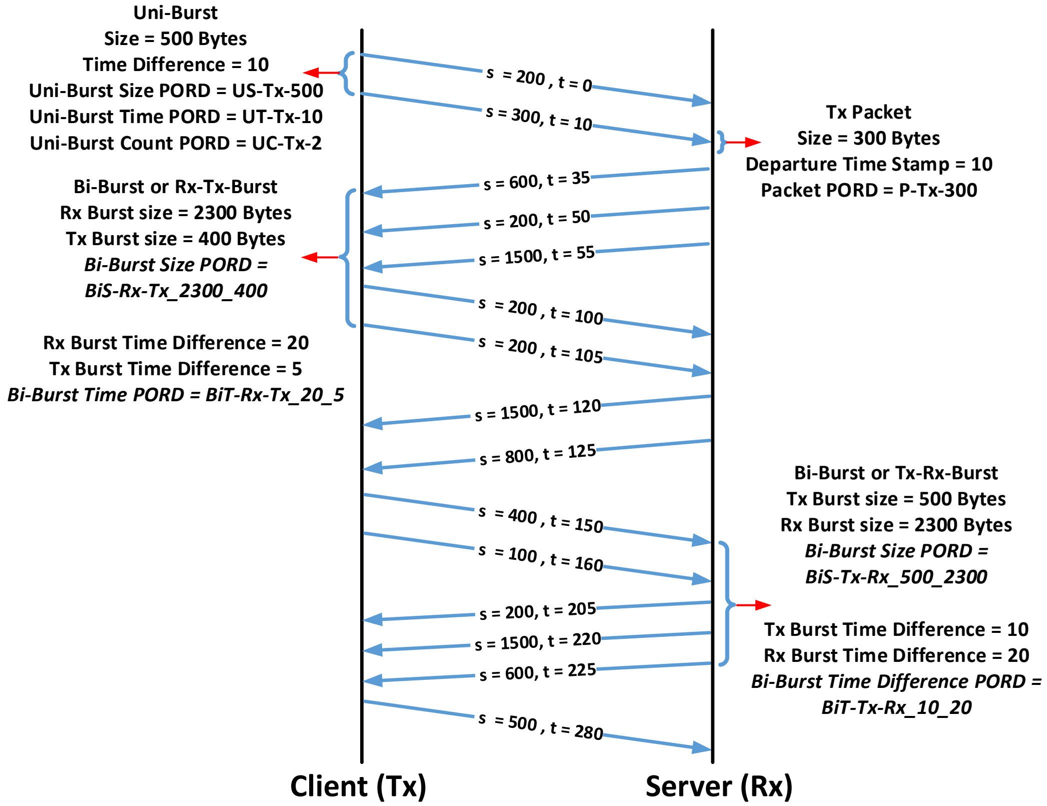


Figure 4.2: Example of how P2V generates PORDs (Packet Words) from a trace.

are not used to train the GloVe model to avoid overfitting. Notice that the trace vector is a fixed-length ($d$-dimensional) vector as every PORD is originally a $d$-dimensional vector. These trace vectors are used for the regular machine learning classification task where we classify using the naïve-bayes (NB) algorithm. In our evaluation, we show how the trace vectors improve the website fingerprinting attack.

## 4.2 Evaluation

### 4.2.1 Dataset

We use a dataset collected by Liberatore and Levine [92]. We call it the *HTTPS* dataset. The traces were collected while browsing websites using HTTPS protocol. As this dataset has been widely used in previous studies, this enables us to compare the results of our approach with other techniques. The traces have been collected for over two months using 2000 websites.

### 4.2.2 Experimental Results

We now present the results of our experiments. For each experiment, we varied the number of selected websites between 20, 40, 60, 80, and 100. To train GloVe, we use 64 randomly selected traces from each website to build the model. We use the P2V approach described in §4.1 to produce PORD vectors. The GloVe word vector size we use is 300. We experiment with a context window of 8. For the gradient descent algorithm, GloVe trains the model using AdaGrad [54]. We use an initial learning rate of 0.05.

For the machine learning classification task, we use 16 randomly selected traces per website (class) for training the classifier, and 4 randomly selected traces per class for testing. We generate trace vectors as described in §4.1.5. To avoid overfitting, none of the testing set traces is used to build the GloVe model. Each experiment has been run ten times with the websites randomly selected from the pool of websites in each run. Then the average accuracy has been obtained.

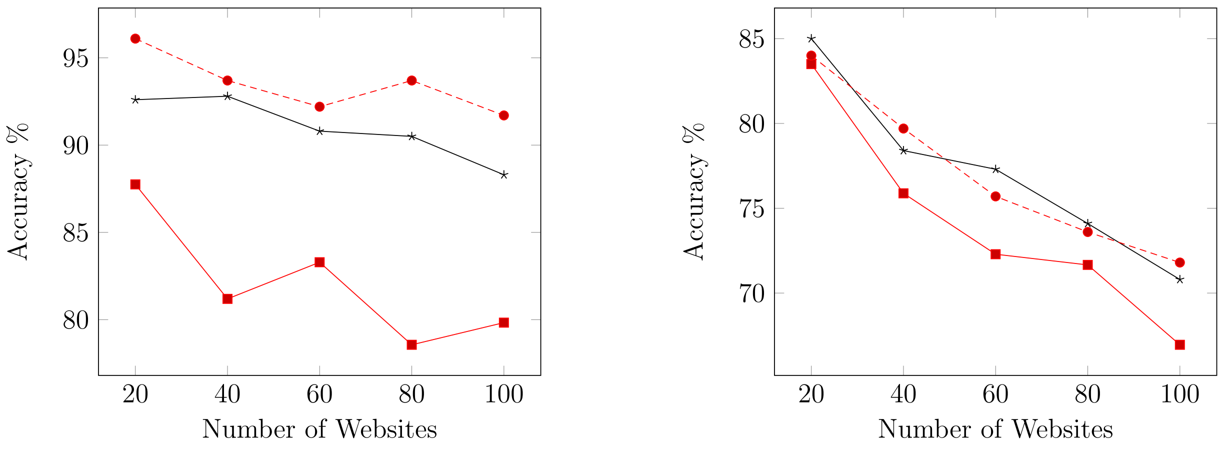
In order to evaluate the performance of our approach, we considered three of the most effective defense mechanisms as well as no applied defense. These defenses are (1) Pad To MTU. (2) Direct Target Sampling. (3) Traffic Morphing.

We compare our results with state-of-the-art website fingerprinting classifiers. These classifiers are VNG++ [57] and Panchenko [107]. There are other classifiers in the website fingerprinting literature such as LL [92], OSAD [138] and others. However, recent studies concluded that VNG++ and Panchenko are two of the most accurate classifiers. As indicated in [57], VNG++ performs better than LL. Also, a recent study [101] shows that Panchenko outperforms the OSAD classifier. VNG++ classifier uses total website upload time, uplink and downlink bandwidth, and uni-bursts as features. It applies the naïve-bayes (NB) classifier to get the prediction. Panchenko classifier uses a large collection of features like packet order, HTML markers, uni-bursts, and others. Panchenko utilizes the support vector machine (SVM) classifier. Our P2V approach produces fixed-length trace vectors that are used in classification. We apply our approach against the two classification methods (VNG++ and Panchenko) using naïve-bayes. We use the *Weka* [66] implementation of these classifiers.

We now present the evaluation by running the experiments against the HTTPS dataset. Figure 4.3a shows the average accuracy when evaluating HTTPS with no defense considered. The X-axis represents the number of websites where we evaluate the experiments against 20, 40, 60, 80, and 100 websites. The Y-axis represents the ten-experiment average accuracy for each classifier including our P2V one. For example, with 20 websites, the accuracies for VNG++, Panchenko, and P2V are 87.75, 92.6, and 96.1 respectively. Our approach proves to perform well even with large number of classes where it achieves accuracies above 90 %. VNG++ does not perform well even when there is no defense applied.

On the other hand, applying defense techniques to the transmitted packets changes the characteristics of the website traffic distribution. This makes classification harder and more sophisticated methods should be used to extract the right patterns.

This concept is clearly illustrated in Figure 4.3b. We can see the overall accuracies drop for all classifiers, including P2V, in this figure as compared to Figure 4.3a where there is no defense applied. Pad to MTU defense pads each packet to the maximum size (MTU) which is 1500 bytes. This defense is less used in practice as it incurs more overhead. When every packet is padded to 1500 bytes, the vocabulary for the P2V model is not rich and hence the statistics will not build an accurate model. More training data and possibly a more sophisticated model may be required to handle this padding case.



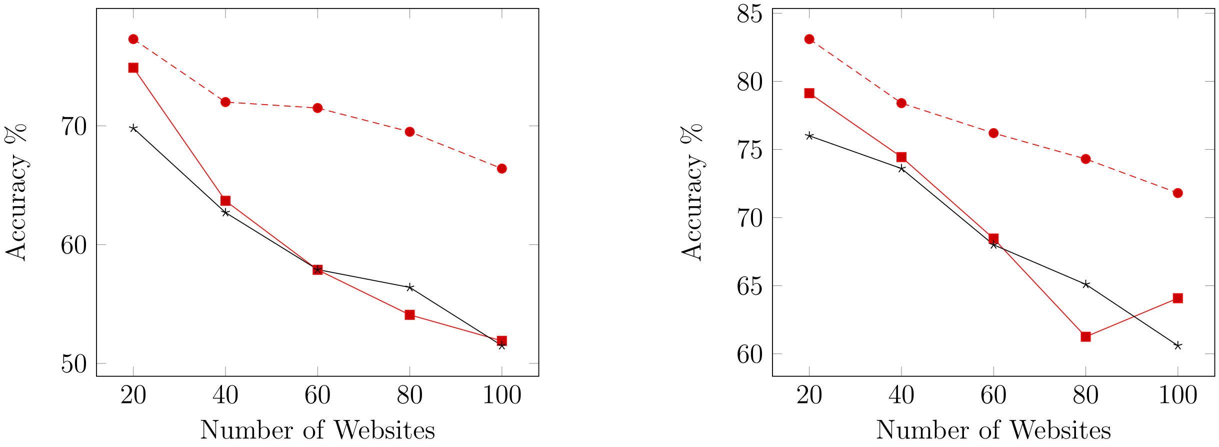
(a) No Defense

(b) Pad To MTU

Figure 4.3: No Defense and Pad To MTU - HTTPS data: —■— VNG++,—⋆— Panchenko, - -●- - P2V.

To show that our approach resists advanced distribution-based defenses, we run the experiments with Direct Target Sampling and Traffic Morphing. Figures 4.4a and 4.4b show the HTTPS dataset results when considering these distribution-based defenses. The figures show the superior performance of the P2V model over the other methods. In DTS, for example, when running the experiments with 60 randomly selected websites, the accuracy for P2V is 71.5% while for VNG++ and Panchenko, the accuracies are 57.88% and 57.9% respectively.

As discussed throughout the dissertation, the fact that packet flows in the TCP protocol affect subsequent packet flows helps construct meaningful statistics in the co-occurrence matrix which is the primary source of information to the GloVe model. P2V produces trace vectors that capture the characteristics of the website even though the defender tries to disguise the network packets actual distribution. We notice that Panchenko classifier preforms worse than previous experiments in Figures 4.3a and 4.3b. This may be due to the fact that some features used in Panchenko classifier such as HTML Markers do not capture the actual characteristics of morphed packets. When comparing DTS and TM in Figures 4.4a



(a) Direct Target Sampling

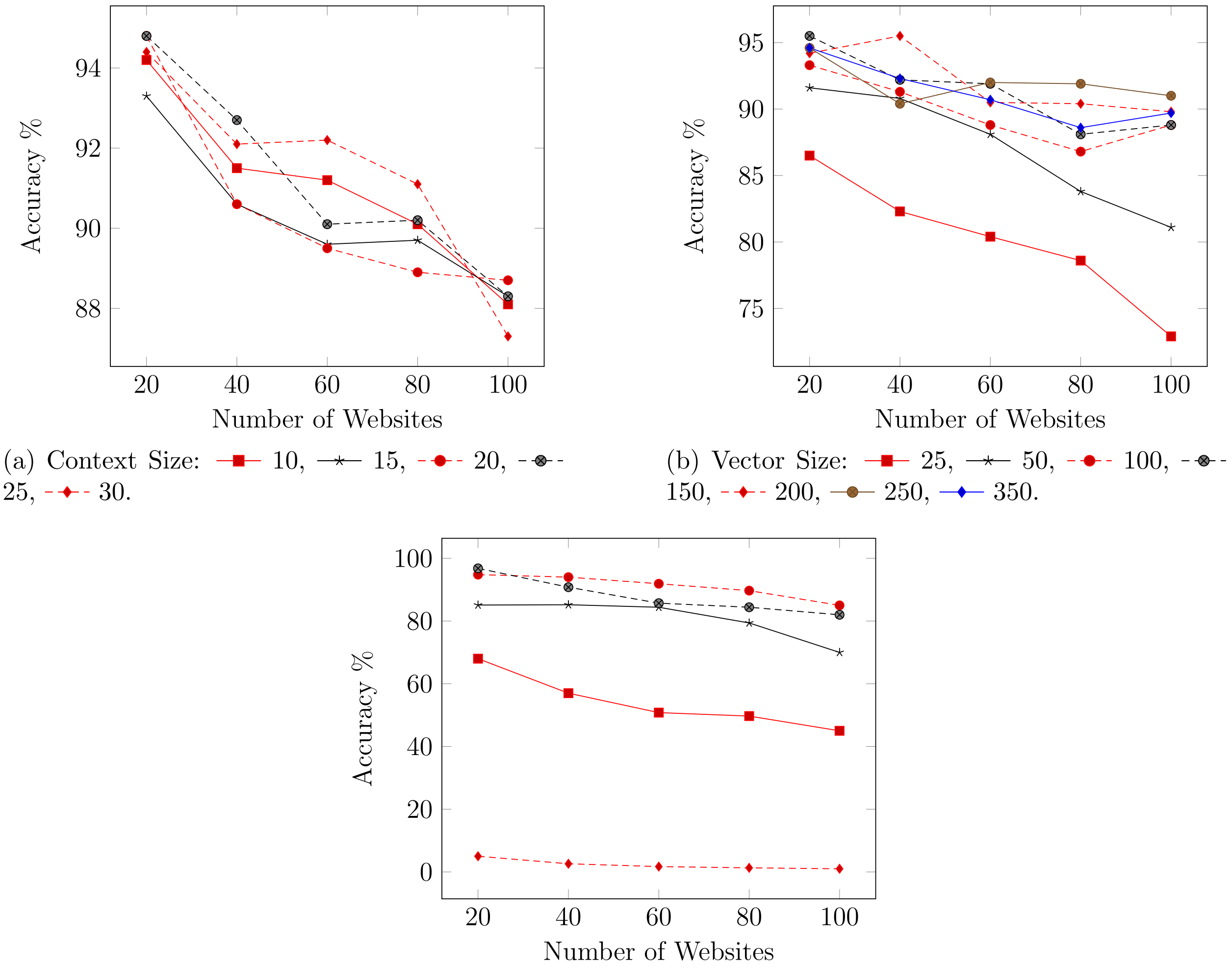
(b) Traffic Morphing

Figure 4.4: Direct Target Sampling and Traffic Morphing - HTTPS data: —■— VNG++, —★— Panchenko, - -●- - P2V.

and 4.4b, we can see that DTS defense is better than TM as it fools the attacker's classifiers and causes the accuracy to be less. There is a trade-off though. DTS incurs more overhead as it generates more bytes to be padded. In contrast, TM uses convex optimization to lower this cost.

### 4.2.3 Model Analysis

Figure 4.5 shows the effect of varying context size, vector length, and initial learning rate. We run these experiments when no defense is applied. Varying the context size does not improve the results significantly as compared to the other two parameters, vector length and initial learning rate. Small vector lengths result in low-dimensional trace vectors that do not help the machine learning classifier much. On the contrary, large initial learning rates give extremely bad results. This is because the gradient descent algorithm does not learn the word vectors well which will make the P2V model produce bad trace vectors.



(a) Context Size: ■ 10, ★ 15, ● 20, ⊗ 25, ♦ 30.

(b) Vector Size: ■ 25, ★ 50, ● 100, ⊗ 150, ♦ 200, ● 250, ♦ 350.

(c) Initial Learning Rate: ■ 0.001, ★ 0.01, ● 0.05, ⊗ 0.1, ♦ 0.5.

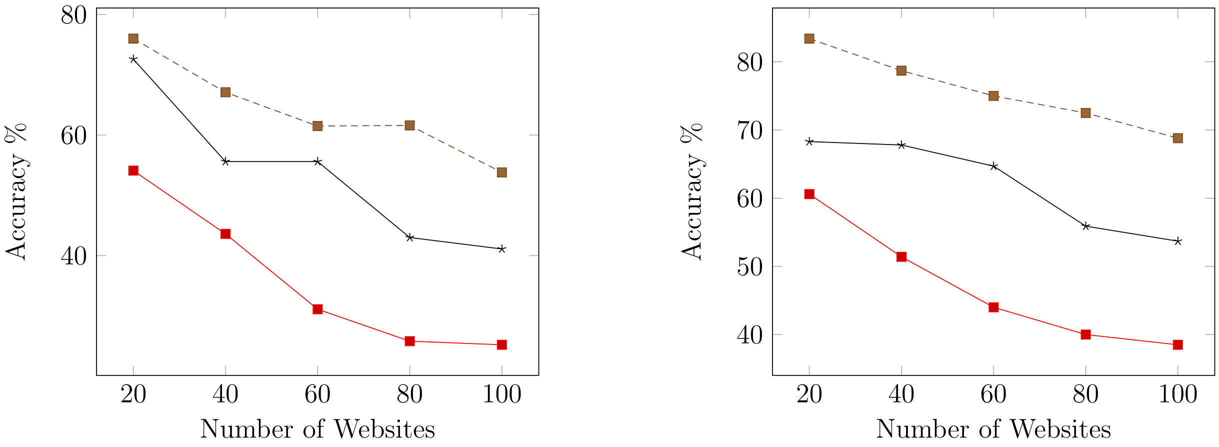
Figure 4.5: P2V Model Analysis - HTTPS data - No defense.

## 4.3 Discussion

Website fingerprinting is the ability for the attackers to identify the websites accessed by users. Attackers may be tyrannical governments who try to suppress freedom. However, attackers may be organizations or even governments who try to track malicious activities. Defenders such as Tor anonymity network apply countermeasures (or defenses) to hinder the attackers ability to threaten web navigation privacy.

Previous studies on website fingerprinting used features such as packet size with direction of data transmission, the length of combined sequential packets in the same direction, called burst, and time taken to load the webpage. These features are extracted from a trace of network traffic belonging to a single website. Moreover, previous studies assumed independence between features extracted. In this work, we model the website fingerprinting attack using GloVe (a word vector representation model). Communication between ends is stacked over the TCP protocol. TCP uses flow control mechanism to ensure certain understanding between client and server. We view this understanding as a dialogue between two ends. In TCP, each packet flow affects the subsequent packet flow. This means there is a dependence between consecutive packet flows. We use this fact to build a packet-packet co-occurrence matrix which is the main source of information used by the GloVe model.

Figure 4.6 shows the effect of fewer vocabulary (PORDs) with DTS and TM defenses. (1) uses Packet Length, Uni-Burst Size, and Uni-Burst Time PORDs only. (2) uses Packet Length, Uni-Burst Size, Uni-Burst Time, Uni-Burst Count, and Bi-Burst Size PORDs. It is clear from the results how enriching the vocabulary improves the P2V classifier. (3) uses the same PORDs as in (2) but with the ACK packets included. We can see how the ACK packets improve the P2V model. These packets can be viewed as filter or stop words in NLP where they are considered crucial for some tasks like Author Attribution as they retain signatures of author style [23]. Furthermore, GloVe uses the context-counting approach where the stop words are considered with a weighting function that limits the effect of frequent co-occurrences. This is the first time study which considers the ACK packets in the website fingerprinting attack design.



(a) DTS: —■— 1, —*— 2, - -■- - 3.

(b) TM: —■— 1, —*— 2, - -■- - 3.

Figure 4.6: DTS and TM when increasing the vocabulary and considering the ACK packets - HTTPS data.

# CHAPTER 5

# BI-DIRECTIONAL BURSTING DISTORTION DEFENSE[1]

## 5.1 Approach Overview

As discussed in Chapter 3, the BIND model is shown to be a powerful website fingerprinting attack even with the presence of defenses that try to disguise packet sequences and make a source website distribution look like it is coming from a different target website distribution. In this section, we introduce a new approach, called BiMorphing, as a novel defense against website fingerprinting attacks. We show how BiMorphing can defend BIND and outperform the state-of-the-art methods.

## 5.2 Methodology

In order to defeat traffic fingerprinting attacks, it is not adequate to morph the packet sequences by just using size padding techniques or even more sophisticated time delay methods. As mentioned in Chapter 3, the bursting nature of website traffic makes it easy to classify a website even when such defenses are applied. In addition, website fingerprinting attacks that leverage bi-directional bursting characteristics have been shown to be effective website fingerprinting attacks even with the presence of defenses that try to disguise packet sequences and make a source website distribution look like it is coming from a different target website distribution.

In this section, we introduce a new approach, called BiMorphing, as a novel defense against website fingerprinting attacks. The proposed defense morphs the bi-bursting patterns (uplink to downlink or downlink to uplink) and makes sure there is no time delay to the

[1] The work presented in this chapter was performed in collaboration with L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Lead author Al-Naami conducted the majority of the research, including most of the design, the full implementation, and the full evaluation.

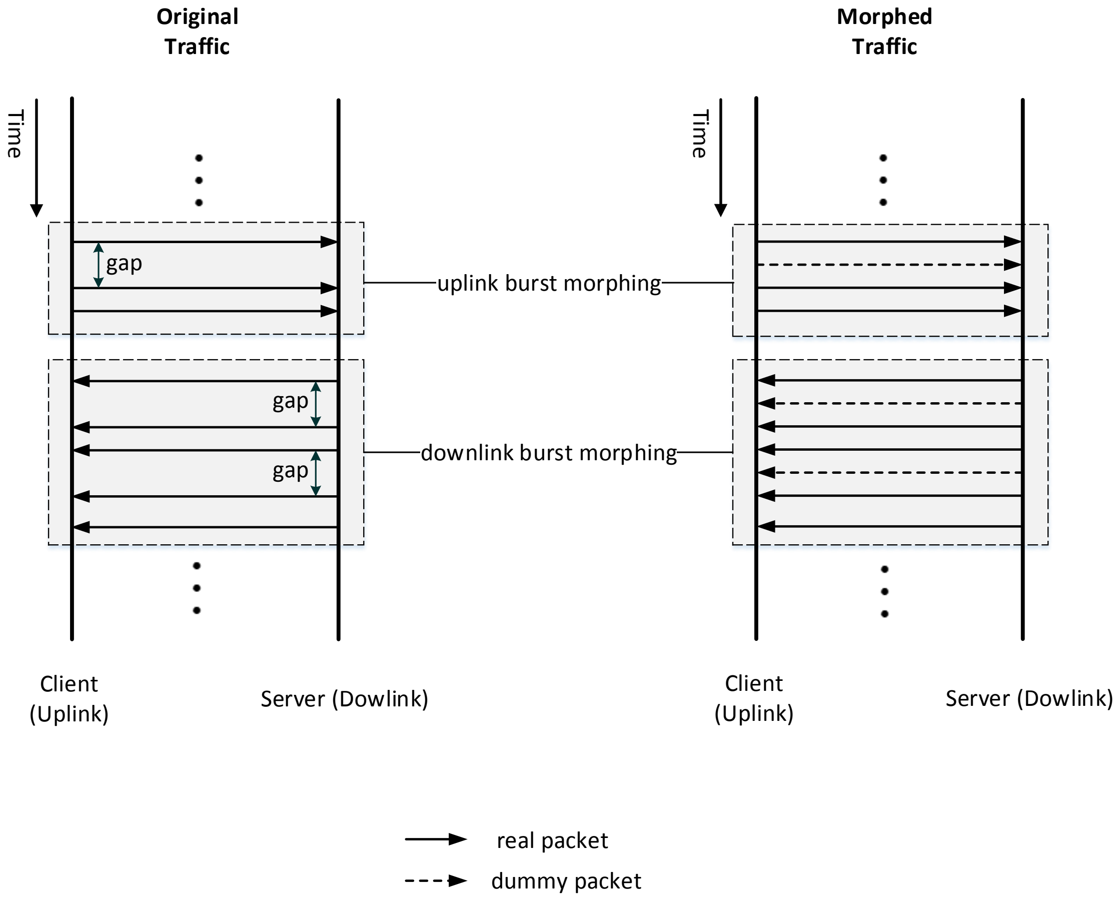


Figure 5.1: BiMorphing Example

actual packets exchanged between client and server. Figure 5.1 presents an example of morphing two bursts (uplink and downlink). For the uplink burst, BiMorphing samples and injects a dummy packet in the gap between the first and second real packets without any delay (i.e., the first two packets in the uplink burst get transmitted on time). Similarly, for the downlink burst, the approach samples and sends two dummy packets in gaps of real packets.

As attackers exploit the bi-bursting size and time nature of encrypted packet sequences to extract useful features, to counteract such attacks, we implement a defense mechanism

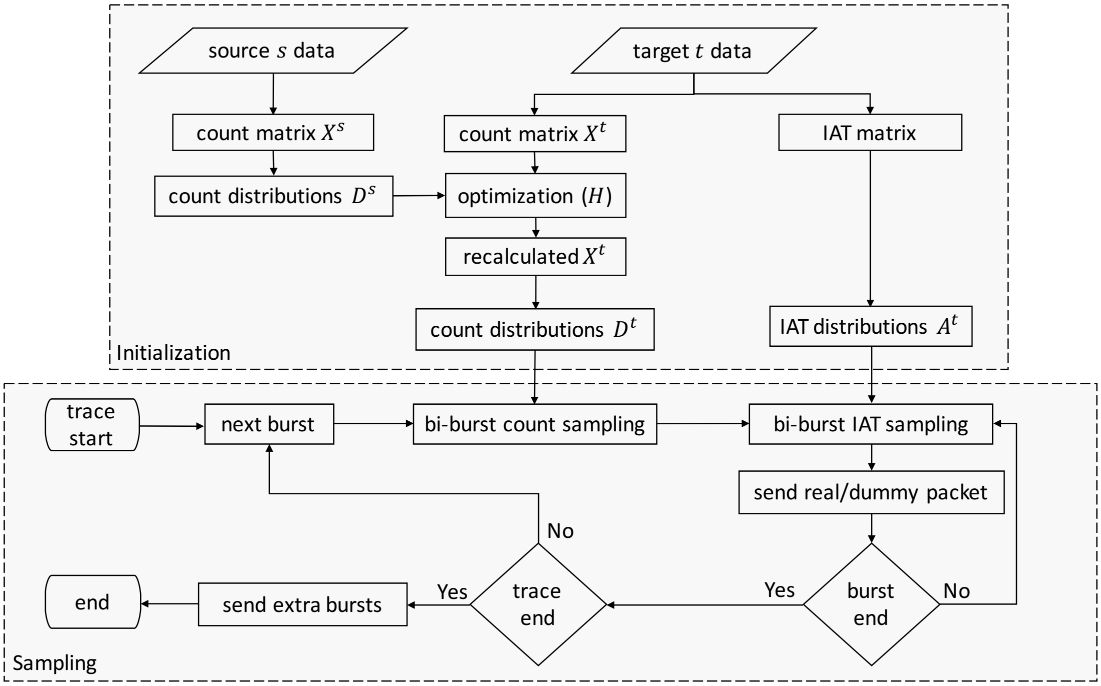


Figure 5.2: BiMorphing Architecture

that hides these characteristics by applying bi-burst sampling techniques to a source website and make it appear as coming from a target website.

BiMorphing's architecture, depicted in Figure 5.2, embodies this approach through the use of optimization and double sampling techniques. The architecture shows the two phases of BiMorphing. The *initialization* phase (top half) is responsible for building distributions that will be used in the double *sampling* phase (bottom half). The architecture will be explained in detail in the following sections.

BiMorphing consists of three main components, bi-bursting count sampling, an optimization technique to lower the padding overhead, and bi-bursting inter-arrival time (IAT) sampling. We now explain the three components in detail.

### 5.2.1 Bi-bursting count sampling

As discussed earlier in this section, an effective defense should change the bi-bursting nature of a website as bi-directional dependence between consecutive bursts reveal characteristics about traffic. Toward this end, the first component of our BiMorphing defense morphs bursts taking into consideration the dependence nature between uplink-downlink and downlink-uplink bursts. BiMorphing is a distribution-based defense with the objective of morphing bi-burst patterns such that these bi-bursts appear to come from a pre-determined target distribution.

**Count Distribution Matrices**. First we define some notations that we use in our figures (such as Figure 5.2) and throughout the chapter. Let $s$ and $t$ be the source and target websites, respectively. Let $X^t = [x_1, x_2, ..., x_n] \in \mathbb{N}^{m \times n}$ be the uplink-downlink (up-dn) or downlink-uplink (dn-up) bi-burst co-occurrence matrix built from the target website, where $x_i = [x_{1i}, x_{2i}, ..., x_{mi}]^T$ is a column vector and each entry $x_{ji}$ tabulates the number of times a burst of count $i$ (i.e., the number of packets) in a specific direction is followed by a burst of count $j$ in the opposite direction. Similarly, $X^s$ is the bi-burst co-occurrence matrix built from the source website. In this work, every individual packet is padded to the maximum transmission unit (MTU).

As depicted in Figure 5.2, from $X^s$ and $X^t$, BiMorphing starts by building matrices of probability distributions $D^s$ and $D^t$ over bi-directional bursting counts from $s$ and $t$, respectively. $D^{\uparrow\downarrow}$ is the uplink-downlink distribution matrix while $D^{\downarrow\uparrow}$ is the downlink-uplink distribution matrix. For instance, as depicted in Figure 5.3, $D^{\uparrow\downarrow t} = [d_1^{\uparrow\downarrow t}, d_2^{\uparrow\downarrow t}, ..., d_n^{\uparrow\downarrow t}]$ is an $m \times n$ matrix that denotes the target uplink-donwlink distribution where $n$ is the number of all possible uplink burst packet counts and $m$ is the number of all possible downlink burst packet counts. The column vector $d_i^{\uparrow\downarrow t} = [d_{1i}^{\uparrow\downarrow t}, d_{2i}^{\uparrow\downarrow t}, ..., d_{mi}^{\uparrow\downarrow t}]^T$ represents the probability mass function (pmf) of the uplink burst count $i$ with all possible downlink burst packet counts (i.e., 1 to $m$). We build similar distribution matrices for the opposite direction of the target

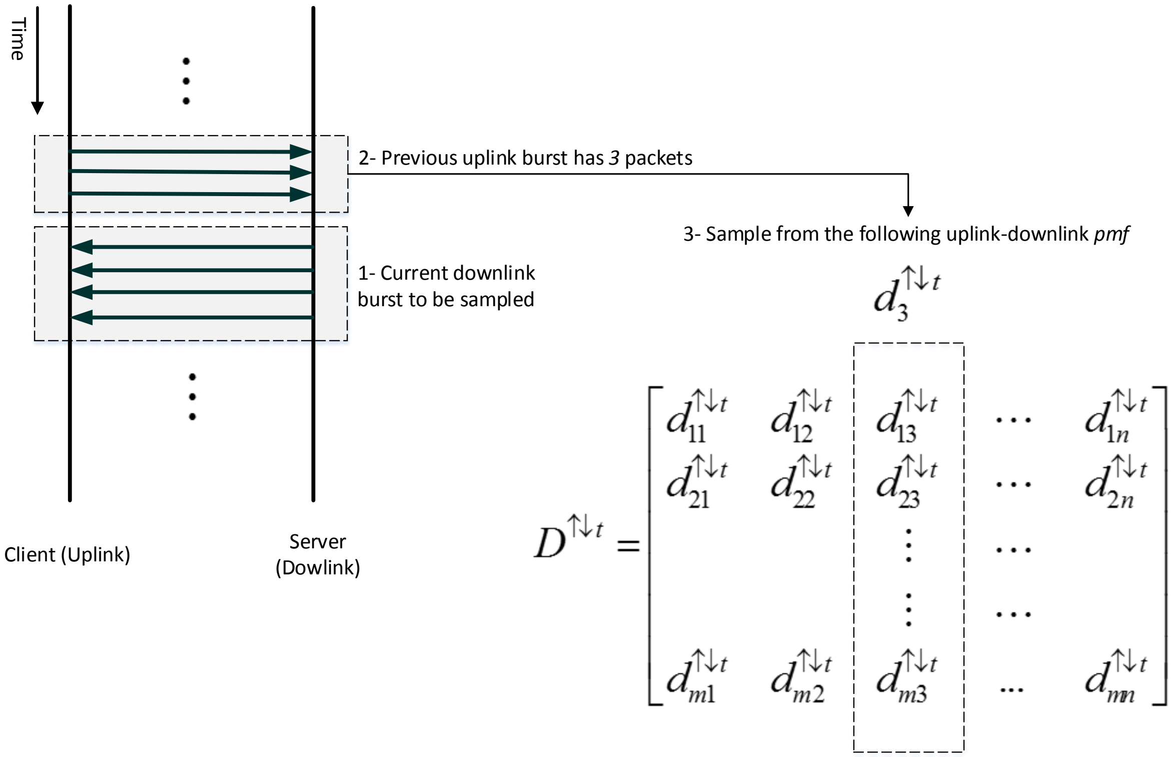

Time
2- Previous uplink burst has 3 packets
3- Sample from the following uplink-downlink pmf
1- Current downlink burst to be sampled
$d_3^{\uparrow\downarrow t}$
$D^{\uparrow\downarrow t} = \begin{bmatrix} d_{11}^{\uparrow\downarrow t} & d_{12}^{\uparrow\downarrow t} & d_{13}^{\uparrow\downarrow t} & \cdots & d_{1n}^{\uparrow\downarrow t} \\ d_{21}^{\uparrow\downarrow t} & d_{22}^{\uparrow\downarrow t} & d_{23}^{\uparrow\downarrow t} & \cdots & d_{2n}^{\uparrow\downarrow t} \\ & & \vdots & \cdots & \\ & & \vdots & \cdots & \\ d_{m1}^{\uparrow\downarrow t} & d_{m2}^{\uparrow\downarrow t} & d_{m3}^{\uparrow\downarrow t} & \cdots & d_{mn}^{\uparrow\downarrow t} \end{bmatrix}$
Client (Uplink)
Server
(Dowlink)

Figure 5.3: Bi-Burst Count Sampling

website (i.e., $D^{\downarrow\uparrow t}$) as well as for the source website (i.e., $D^{\uparrow\downarrow s}$ and $D^{\downarrow\uparrow s}$). The distributions are shown in Figure 5.2. Notice that, we don't show the arrows in Figure 5.2 for simplicity but for each case, we generate distributions for both directions (uplink to downlink and downlink to uplilnk).

**Bi-Burst Count Sampling**. In BiMorphing, we start by sending the first burst from the source website $s$ as is. Then, for each burst of count $i$ from $s$, we sample a burst of count $j$ from the $t$'s distribution matrix $D^t$ depending on the previous burst. The sampling process is illustrated in Figure 5.3. As an example, let $b_i^s$ be the current source downlink burst with count $i$. As this is a downlink burst, we sample based on the previous burst direction (i.e., uplink) and count (i.e., we sample from the column vector $d_k^{\uparrow\downarrow t}$ assuming the previous uplink burst has $k$ packets). Form this pmf, we build its corresponding Cumulative Distribution Function (CDF) and uniformally sample a burst. Let $b_j^t$ be the sampled burst with count $j$. If $j > i$, we add $(j - i)$ fake packets to the original burst from $s$ and send. Otherwise, we send the original burst and continue sampling until all source bursts are consumed. We interleave these fake packets with the original real packets from $s$ using an algorithm that ensures zero delay for the original real packets as will be explained shortly. Finally, if the total number of bursts in target is larger than the total number of bursts in source, we add the extra target bursts to the source. This ensures that small website patterns are not revealed to the attacker.

### 5.2.2 Learning Optimal Target Co-occurrence Distribution

The bi-bursting sampling proposed above may introduce a sampling bias in the target distribution. This bias comes from the fact that most of the bi-burst packet counts are small. Hence, this leads to a sampling bias towards these small bursts which may result in a misrepresentation of the target in the new generated distribution. In addition, adding fake packets during sampling may incur a high overhead to the bandwidth. Toward dealing with these two challenges (sampling bias and bandwidth overhead), we propose a balancing solution through the use of mathematical optimization as depicted in Figure 5.2. BiMorphing introduces two objective functions, one for the uplink-downlink distributions ($H_{\uparrow\downarrow}$) and the other one for the downlink-uplink distributions ($H_{\downarrow\uparrow}$). Equation 5.1 shows the objective function minimizing $H_{\uparrow\downarrow}$.

$$\min_{W \in \mathbb{R}^{m \times n}} H_{\uparrow\downarrow} = \sum_{i=1}^{n} \sum_{j=1}^{m} \; p_{ij} \; f(x_{ij}) \; [w_{ij} \; (|b_j^t| - |b_i^s|)]^2, \tag{5.1}$$

Here, $n$ and $m$ are the number of all possible uplink burst counts and all possible downlink burst counts, respectively. $p_{ij}$ is the probability from the pmf of the source website while $x_{ij}$

is the number of times an uplink burst of count $i$ is followed by a downlink burst of count $j$ in the target co-occurrence matrix $X^t$. Equation 5.2 explains $f(x_{ij})$ which is the same weighting function introduced in [112] with the same model parameters (i.e., $x_{max} = 100$ and $\alpha = 3/4$). $f(x_{ij})$ is a weighting function designed to eliminate noise between co-occurrences of consecutive words (bi-bursts in our case). It deals with rare co-occurrences as well as frequent co-occurrences of bi-bursts.

$$f(x_{ij}) = \begin{cases} (\frac{x_{ij}}{x_{max}})^{\alpha}, & \text{if } x_{ij} < x_{max} \\ 1, & \text{otherwise.} \end{cases} \tag{5.2}$$

The weights $w's$ are the parameters to learn. The overhead to be minimized is $(|b^t_j| - |b^s_i|)$ which denotes burst count difference between target and source. After learning the optimal $w's$, we recalculate $X^t$ using the Hadamard entrywise matrix product $X^t = X^t \circ W$ where $x_{ij} = x_{ij}\ w_{ij}$.

The partial derivative of Equation 5.1 with respect to each weight $w_{ij}$ is as follows.

$$\begin{aligned} \frac{\partial H_{\uparrow\downarrow}}{\partial w_{ij}} &= p_{ij}\ f(x_{ij})\ 2\ [w_{ij}\ (|b^t_j| - |b^s_i|)]\ \frac{\partial[w_{ij}\ (|b^t_j| - |b^s_i|)]}{\partial w_{ij}} \\ \frac{\partial H_{\uparrow\downarrow}}{\partial w_{ij}} &= p_{ij}\ f(x_{ij})\ 2\ [w_{ij}\ (|b^t_j| - |b^s_i|)]\ (|b^t_j| - |b^s_i|) \\ \frac{\partial H_{\uparrow\downarrow}}{\partial w_{ij}} &= 2\ p_{ij}\ f(x_{ij})\ (|b^t_j| - |b^s_i|)^2\ w_{ij} \end{aligned} \tag{5.3}$$

Accordingly, each iteration in gradient descent modifies each parameter $w_{ij}$ as follows.

$$w_{ij} = w_{ij} - \gamma\ .\ \frac{\partial H_{\uparrow\downarrow}}{\partial w_{ij}}, \tag{5.4}$$

where $\gamma$ is the step size. Equation 5.5 shows the downlink-uplink objective function minimizing $H_{\downarrow\uparrow}$ which is similar to the one in Equation 5.1 with flipping the directions of uplink and downlink and observing the downlink-uplink distribution values. Similarly, the partial derivative of $H_{\downarrow\uparrow}$ with respect to $w_{ij}$ is similar to Equation 5.3 but with the values coming form the downlink-uplink distributions.

$$\min_{W \in \mathbb{R}^{m \times n}} H_{\downarrow\uparrow} = \sum_{i=1}^{n} \sum_{j=1}^{m} \; p_{ij} \; f(x_{ij}) \; [w_{ij} \; (|b_j^t| - |b_i^s|)]^2 \tag{5.5}$$

This optimization technique ensures that co-occurring bi-bursts are not weighed equally (i.e., frequent co-occurrences are not overweighed and noisy rare co-occurrences do not carry more than deserving weights). It also minimizes the overhead of sampling from the target distribution which is crucial for any efficient defense mechanism.

### 5.2.3 Bi-burst Inter-arrival Time (IAT) Sampling

Although the above sampling methodology achieves the purpose of bi-burst morphing, a main drawback is that fake packets incur a time delay overhead as they are sent with original real packets. This leads to a delay to the actual traffic exchanged between client and server. To tackle this issue, we introduce a zero delay algorithm that is a modified and simplified version of the Adaptive Padding algorithm introduced in [119, 76]. The algorithm sends fake packets in gaps of real packets without delaying the actual traffic. Our approach combines bi-burst count sampling and bi-burst time sampling together which not only hides trace size characteristics but also disguises timing leaks that may be used by attackers to accurately fingerprint websites.

**IAT Distribution Matrices**. The departure/arrival (uplink/downlink) time difference between observations of two consecutive packets is the inter-arrival time (IAT). We first start by building the IAT distributions from the target website $t$. In a similar fashion to the bi-burst count distributions, the approach builds two inter-arrival time (IAT) distributions from bi-bursts, one for uplink-downlink ($A^{\uparrow\downarrow t}$) and the other for downlink-uplink ($A^{\uparrow\downarrow t}$). For the uplink-downlink case, $A^{\uparrow\downarrow t} = [a_1^{\uparrow\downarrow t}, a_2^{\uparrow\downarrow t}, ..., a_n^{\uparrow\downarrow t}] \in \mathbb{R}^{m \times n}$ denotes the target uplink-donwlink IAT distributions where $n$ is the number of all possible uplink burst packet counts and $m$ is the number of all possible downlink inter-arrival times. The column vector $a_i^{\uparrow\downarrow t} = [a_{1i}^{\uparrow\downarrow t}, a_{2i}^{\uparrow\downarrow t}, ..., a_{mi}^{\uparrow\downarrow t}]^T$ represents the probability mass function of the uplink burst count $i$ with all possible *next-burst* downlink inter-arrival times (i.e., 1 to $m$). As before, we build a similar matrix of the opposite direction for the target website $t$ (i.e., $A^{\downarrow\uparrow t}$). These matrices are shown in Figure 5.2.

**Bi-Burst IAT Sampling**. Bi-burst IAT sampling runs simultaneously with bi-burst count sampling introduced above (double sampling) to ensure sending fake packets in gaps between real packets without delaying the actual traffic. The process is shown in Figure 5.2. Whenever a real packet is ready to be sent, and depending on the previous burst direction and count, BiMorphing samples an inter-arrival time from the corresponding distribution. For example, if the source current burst is a downlink burst $b_i^s$, we sample based on the previous burst direction which is uplink (i.e., we sample an inter-arrival time from the column vector $a_k^{\uparrow\downarrow t}$ assuming the previous burst has a count of $k$ packets). Similarly, if the current burst is uplink, we sample from the previous downlink burst's pmf, i.e., $a_k^{\downarrow\uparrow t}$.

### 5.2.4 Zero Delay Packet Interleaving

As mentioned earlier, the BiMorphing defense runs bi-burst count sampling and bi-burst IAT sampling concurrently. The algorithm is depicted in Figure 5.4 using a finite state machine. Let's assume bi-burst count sampling gives us a pool of $f$ fake packets to interleave with real burst packets (*sample* $f$ from $D^{\uparrow\downarrow t}$, as coming from a downlink current burst, $b^{\downarrow s}$). Whenever a real packet is ready to be sent, BiMorphing sends it without delay ($send(p)$), samples a new inter-arrival time, and starts a timer $r$ (*sample* $r$ from $A^{\uparrow\downarrow t}$). If $r$ expires before another real packet comes, then BiMorphing sends a fake (dummy) packet ($send(d)$) from the pool $f$ and starts over by resampling another inter-arrival time. If a real packet arrives before $r$ expires, we send the real packet (without sending any fake packets) and resample an inter-arrival time.

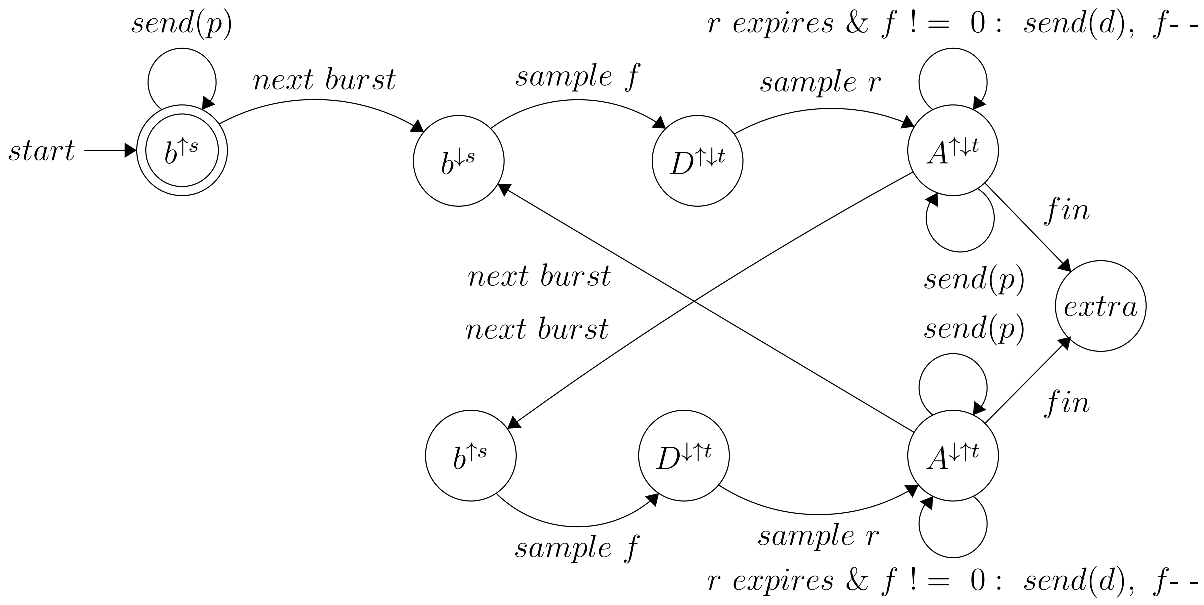

send(p)
next burst
sample f
sample r
r expires & f != 0 : send(d), f- -
start
$b^{\uparrow s}$
$b^{\downarrow s}$
$D^{\uparrow\downarrow t}$
$A^{\uparrow\downarrow t}$
fin
next burst
next burst
send(p)
send(p)
extra
fin
$b^{\uparrow s}$
$D^{\downarrow\uparrow t}$
$A^{\downarrow\uparrow t}$
sample f
sample r
r expires & f != 0 : send(d), f- -

Figure 5.4: Finite state machine to illustrate the BiMorphing algorithm. *send*($p$) denotes sending a real packet instantly. *send*($d$) denotes sending a dummy packet. $f$ is the bi-burst count sampling pool. $r$ is the countdown timer after sampling the bi-burst IAT. $fin$ refers to end of trace. *extra* denotes sending extra bursts from target if any.

The process continues until all current burst (uplink or downlink) real packets have been sent. If the pool $f$ is not exhausted yet at the end of the current burst, we continue sending these residuals using the IAT sampling process until receiving a packet from the other party (*next burst*). We continue a similar process with the next burst. At the end of trace ($fin$), we send extra *tail* bursts from target if the total number of bursts of target is greater than the total number of bursts in source (*extra*).

## 5.3 Evaluation

In this section, we demonstrate the effectiveness of the proposed traffic fingerprinting defense. We evaluate BiMorphing against a Tor dataset (denoted as Tor) using the methodology described in §5.2. We examine the closed-world and open-world scenarios when no defense is applied and when there is a defense mechanism.

Table 5.1: The Tor dataset

| Dataset | # of websites | | # of traces per website | Closed-world | Open-world |
|---|---|---|---|---|---|
| Tor [137] | Monitored | 100 | 90 | ✓ | ✓ |
| | Non-Monitored | 5000 | 1 | × | ✓ |

### 5.3.1 Dataset and Experimental Setup

The Tor dataset we use to validate our approach with was collected by capturing encrypted packets generated from a browser connected to the Tor anonymity network. The dataset is described in detail in [137]. As described in Table 5.1, the dataset consists of two groups of collections. The first one is a group of 100 websites with 90 traces (page loads) each. These websites were collected from a list of blocked websites by three censoring countries. We use these 100 websites for the closed-world experiments. The second collection consists of 5000 websites where each website has one trace. These websites were selected from the Amazon Alexa's top websites [17]. In the open-world setting, we consider the first group of 100 websites as the monitored set and the second group of 5000 website and the non-monitored set.

**Closed-world**. We use the BIND attack explained in Chapter 3 to evaluate our defense. BIND uses a support vector machine classifier (SVM). In our experiments, we use a publicly available library called LibSVM [39] with a Radial Basis Function (RBF) kernel having the parameters $Cost = 1.3 \times 10^5$ and $\gamma = 1.9 \times 10^{-6}$ [107]. We consider the first collection of websites for evaluating BiMorphing, i.e., the 100 blocked websites with 90 traces each. We perform a 10-fold cross validation for training and testing the classifier. The results of the closed-world evaluation are measured by computing the average accuracy of classifying the correct class for all test traces.

**Open-world**. We use the monitored set (100 websites with 90 traces each) and the non-monitored set (5000 websites with one trace each) for the open-world scenario. We use the

SVM classifier with the same parameters used for the closed-world case. We apply a 10-fold cross validation as well. Furthermore, as the open-world scenario is a binary classification problem (monitored or non-monitored), we measure the true positive rate (TPR) and false positve rate (FPR). These are defined as follows: $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$. Here, $TP$ (True Positive) is the number of traces which are monitored, and predicted as monitored by the classifier. $FP$ (False Positive) is the number of traces which are non-monitored, but predicted as monitored. $TN$ (True Negative) is the number of traces which are non-monitored and predicted as non-monitored. $FN$ (False Negative) is the number of traces which are monitored, but predicted as non-monitored.

**Optimization**. For learning the optimal target bi-burst co-occurrence weights explained in §5.2.2, we use the gradient descent algorithm. The number of iterations we use is 100 with the step size $\gamma = 0.001$. We initialize the values of each parameter $w_{ij}$ to one. As mentioned in §5.2.2, the optimal learned weights are then used to recalculate the distributions of the target website to correct any sampling bias to frequent bi-burst counts and ensure minimum bi-burst sampling overhead.

**Comparison**. In order to evaluate the performance of BiMorphing, we consider running it against the BIND attack and show how it decreases the accuracy. We also compare the BiMorphing defense against the most recent state-of-the-art defense (BurstMolding) introduced in [140]. BurstMolding morphs individual bursts of a source website to look like the target website bursts. Unlike our approach, BurstMolding is a one-to-one burst molding defense that merges uni-bursts of source and target websites by taking the maximum burst count of each source burst and its correspondent target burst, in order. Unfortunately, BurstMolding does not implement any approach to ensure zero delay of traffic transmission. Our defense BiMorphing not only modifies individual bursts, but also considers the dependency between bi-bursts and uses optimized sampling techniques with zero delay traffic transmission.

Table 5.2: BIND attack accuracy (%) in the closed-world setting against normal and morphed Tor data

| BIND Attack | |
|---|---|
| Method | Accuracy (%) |
| No Defense | 80.04 |
| BurstMolding Defense | 27.74 |
| BiMorphing Defense | **15.57** |

### 5.3.2 Results

Using the Tor dataset, we evaluate the BiMorphing approach in the closed-world and open-world settings. We show the results when no morphing is applied (normal traffic) and compare them to the morphed data (when packets are morphed).

**BiMorphing in Closed-world**. We use the first collection of the Tor dataset for the closed-world experiments with 9000 instances (100 websites with 90 traces each). We get the average accuracy over a 10-fold cross validation using the SVM classifier for this multi-class problem. Table 5.2 presents the results. The table shows the results for the original and defended (morphed) data. As shown in the table, the accuracy of the normal data is 80.04% classifying the 100 websites. When defenses are applied to traffic, the accuracy drops to 27.74% and 15.57% for BurstMolding and BiMorphing, respectively. This shows the effectiveness of the proposed BiMorphing defense which considers a zero delay optimized bi-burst sampling technique. Not only does BiMorphing disguise the bi-directional bursting patterns via the bi-burst count sampling, but it also protects against the inter-packet arrival time leak through the IAT sampling technique.

**BiMorphing in Open-world**. For evaluating BiMorphing in the open-world scenario, we use the whole Tor dataset. The monitored set consists of the 9000 instances of the 100 blocked websites in the first collection while the non-monitored set consists of the second collection websites (i.e., 5000 websites with one instance each). The classification becomes a binary classification problem with each monitored website as a positive point and each

Table 5.3: BIND attack accuracy (%) in the open-world setting against normal and morphed Tor data

| BIND Attack | | | | |
|---|---|---|---|---|
| Method | TPR (%) | FPR (%) | #TP | #FP |
| No Defense | 99.80 | 3.40 | 8982 | 170 |
| BurstMolding Defense | 92.72 | 17.86 | 8345 | 893 |
| BiMorphing Defense | **88.33** | **29.26** | **7950** | **1463** |

non-monitored website as a negative point. Similar to the closed-world setting, we use a 10-fold cross validation with the difference that we measure the true positive rate (TPR) and the false positive rate (FPR). The results are illustrated in Table 5.3. The table shows the results when no defense is considered as well as when applying the defenses techniques. An effective defense must decrease the classifier TPR while increasing its FPR value. The TPR value dropped from 99.80% (no defense) to competitive values of 92.72% and 88.33% for the BurstMolding and BiMorphing defenses, respectively. In addition, we see that FPR of each defense increases significantly when applying the defenses with the highest value achieved by BiMorphing. Along with the TPR and FPR ratios, the table also shows the number of true and false positive instances classified by each approach.

**Defense Overhead**. Surely, morphing burst sequences by adding packets comes with an inevitable bandwidth overhead. An effective defense algorithm must minimize this overhead while achieving the desired goal of hiding the characteristics of the destination website. BiMorphing uses an optimization technique to get the bandwidth overhead to its lowest. On the other hand, if not dealt with properly by the algorithm, morphing can come with a possible time delay to the actual traffic. In reality, unlike bandwidth overhead, any delay overhead becomes a concern in low-latency networks like Tor. Most of the existing traffic fingerprinting defenses are imperfect when dealing with delay overhead. As discussed in §5.2.3, BiMorphing introduces a zero delay algorithm that sends the extra sampled packets in gaps of real packets in a way that ensures real packets arrive on time.

Table 5.4: Bandwidth and delay overhead with the BIND attack against defenses

| BIND Attack | | |
|---|---|---|
| Defense | BW Overhead (%) | Delay Overhead |
| BurstMolding | 86.90 | Yes |
| BiMorphing | **56.40** | **No** |

In this section, we show the bandwidth and delay overhead. We see in Table 5.4 that BiMorphing achieves a lower bandwidth (BW) overhead than the other competing algorithm (BurstMolding). Figure 5.5 presents the trade-off between the BiMorphing defense effectiveness and bandwidth overhead. For the delay overhead, BiMorphing scores a zero delay overhead to the actual traffic whereas BurstMolding can not avoid it. The overhead measure shown in this section does not consider the extra burst traffic sent after the real traffic gets transmitted. This is because when the last packet gets exchanged, control messages between client and server flag end of real data. The following data is full dummy and need not be considered in the measurements.

**Comparison to other approaches**. In our experiments, we evaluated BiMorphing against latest website fingerprinting studies (attack and defense). Specifically, we chose the BIND attack as it leverages bi-directional bursting and our defense utilizes the same concept as well (i.e., bi-directional sampling). However, we observed similar behavior of the proposed approach when running it against other attacks and defenses. For instance, we applied the 10-fold cross validation approach on the 100 monitored websites using CUMUL [106] and P-SVM [107] attacks. When using CUMUL, BiMorphing decreased the accuracy from 59.04% (no defense) to 3.05%. For the P-SVM attack, the accuracy dropped from 79.72% in the case of no defense to 11.79% when applying BiMorphing. We also compared our defense to Traffic Morphing (TM) as it is an optimized sampling defense. Running the BIND attack against TM defense resulted in an accuracy of 68.7% whereas BiMorphing took the accuracy down to 15.57%.

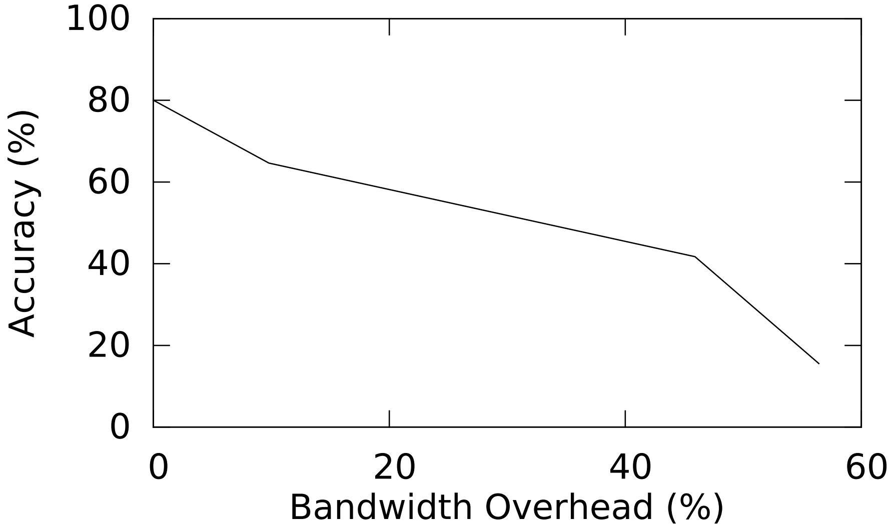


Figure 5.5: Accuracy and bandwidth overhead

**Pool of target websites**. BiMorphing deforms the bursting nature of a source website by making its distribution resemble a predetermined target distribution (i.e., one target website). In this experiment, we morph the source website to resemble a pool of target websites. We do that by increasing the number of target websites and derive the distributions and run the optimization explained in §5.1 against the combined co-occurrence matrices. The results are presented in Figure 5.6. Apparently, increasing the number of target websites results in affecting the defense negatively (i.e., attack accuracy gets higher). For instance, having a pool of two target websites results in an accuracy of 39.01% while a ten-target-website pool increases the accuracy to 44.97%.

### 5.4 Discussion

**Methodology**. In this work, we proposed BiMorphing, a new defense to thwart the traffic fingerprinting passive attack. One of the challenges that any defense mechanism faces is the design of an effective defense that prevents attackers from extracting knowledge from encrypted traffic taking into account minimizing the bandwidth and time overhead.

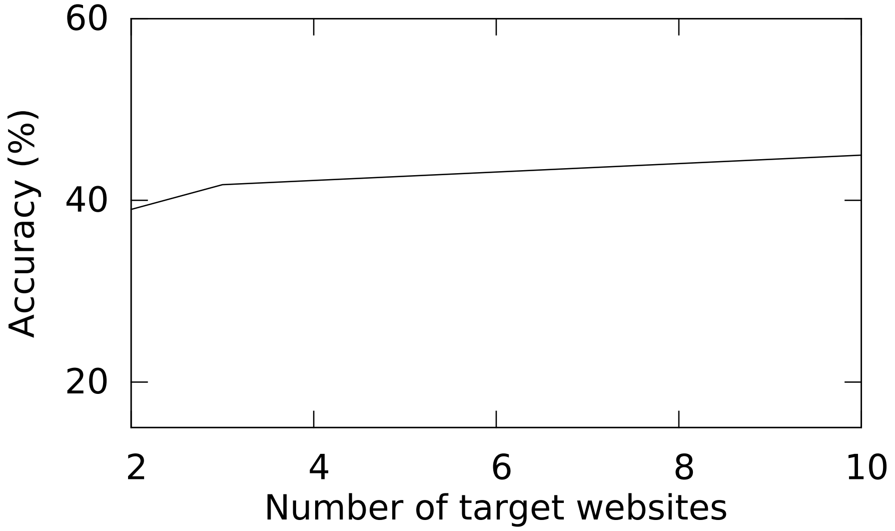


Figure 5.6: Increasing the number of target websites effect

BiMorphing introduces bi-directional dependence size and time sampling with optimization that ensures the lowest bandwidth overhead possible. The defense achieves a zero delay packet transmission as it sends the extra dummy packets in gaps of real packets that get to be sent without any delay.

**Target Distributions**. In order for the algorithm to achieve its best, and as the approach leverages sampling from target distributions, the choice of target should be made carefully. On the one hand, the bi-burst co-occurrence distributions may become sparse if the target does not have large sequences. This definitely affects the overall performance of the algorithm. On the other hand, if one chooses a target that has very large sequences, the approach may result in a higher-than-desired bandwidth overhead. Thus, there is a trade-off between the two cases.

# CHAPTER 6

# CYBER-DECEPTIVE FINGERPRINTING[1]

## 6.1 Approach Overview

This chapter presents the cyber-deceptive intrusion detection system (IDSes) introduced in Chapter 1. We first outline practical limitations of traditional machine learning techniques for intrusion detection, motivating our research. We then overview our approach DeepDig for automatic attack labeling and feature extraction via honey-patching.

### 6.1.1 Intrusion Detection Obstacles

Machine learning-based intrusion detection systems discover deviations from expected patterns, with the basic assumption that malicious activities exhibit properties that are abnormal relative to legitimate usage of a system [50]. Typically, these systems utilize machine learning algorithms (e.g., information theory [89], neural networks [147], clustering [118], or genetic algorithms [121]) to train a model of normal activity and to discover non-conforming patterns, such as in network packets, system calls, and application logs. This is called anomaly-based intrusion detection.

In spite of the rising use of machine learning in intrusion detection systems, its success in real environments has been hindered by specific challenges that arise in the cyber security domain. Indeed, machine learning algorithms perform better at discovering similarities than at identifying previously unseen instances (i.e., outliers). As benign, non-malicious data is usually more plentiful than realistic, current attack data, many approaches are trained almost solely from the former, necessitating an almost perfect model of normality for any efficient classification [124].

[1] The work presented in this chapter was performed in collaboration with F. Araujo, A. Gbadebo, A. Mustafa, L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Al-Naami led the machine learning half of the research, including feature generation, classification design, implementation, and experiments.

Another challenge is *Feature generation* [30] which is commonly difficult in intrusion detection context as security-related features are often not known by defenders in advance. Feature generation or extraction is the process of deriving informative, non-redundant statistical information from raw data to improve the accuracy and performance of the detection model. Selecting proper features to discover possible threats (e.g., features that generate the most distinguishing intrusion patterns) often creates a bottleneck in designing effective classifiers, since it requires extensive empirical evaluation. Particularly, identifying attack traces among collected traces for constructing realistic, unbiased training sets is challenging. Current approaches usually require human expert interventions to do manual analysis [38, 29]. This severely decreases model evolution and update capabilities to enhance the model with new threats and to cope with attacker evasion techniques.

Analysis of encrypted data (encrypted packets) poses a third challenge. Generally, encryption is utilized to prevent unauthorized access to sensitive data transmitted through network links or stored in file systems. However, since existing network-based detectors typically discard encrypted traffic, their efficacy is greatly reduced by the widespread use of encryption technologies [63]. Unfortunately, adversaries benefit from encrypting their malicious payloads, making it harder for standard classification strategies to distinguish attacks from normal activity.

Another obstacle is generating high false alarms (false positive rates) which is another challenge to the machine learning-based detectors [108]. Raising too many alarms causes intrusion detection systems to become inefficient in most cases, as real attacks are often lost among the many false alarms. Typically, effective intrusion detection systems necessitates very low false alarm rates [25]. It is therefore critical that new strategies for intrusion detection meet the requirement of reducing the rate of false alarms.

In this work, we address all of the aforementioned challenges through the exploration and development of a novel and accurate intrusion detection system (called DeepDig).

DeepDig incorporates information from different layers of the software stack by extending machine learning-based intrusion detection with the capability to effectively and accurately detect malicious threats bound to the application layer. The new system automatically and continuously extracts security-relevant features for attack detection, affording detection approaches a lightweight and inexpensive tool.

### 6.1.2 Deception-Enhanced Threat Data Digging

To overcome the limitations of existing intrusion detection systems, we introduce DeepDig as a new methodology to enhance machine learning-based intrusion detection. DeepDig utilizes threat data sourced from honey-patched applications (discussed below). Figure 6.1 shows the approach overview. Unlike traditional anomaly-based detection approaches, DeepDig incrementally builds a model of *benign* and *malicious* data based on audit streams and attack traces collected from honey-patched web servers. This augments the detection classifiers with security-related feature generation abilities not attainable by typical network intrusion detectors.

These capabilities are transparently integrated into the framework, requiring no additional developer efforts (apart from routine patching) to convert the target application into a potent feature extractor for intrusion detection. Since traces extracted from decoys are always contexts of *true* malicious activity, this results in an effortless labeling of the data and supports the generation of higher-accuracy detection models.

Honey-patches add a layer of deception to confound exploits of known (patchable) vulnerabilities. Previously unknown (i.e., zero-day) exploits can also be mitigated through IDS cooperation with the honey-patches. For example, a honey-patch that collects identifying information about a particular adversary seeking to exploit a known vulnerability can convey that collected information to train a classifier, which can then potentially identify the same adversary seeking to exploit a previously unknown vulnerability. This enables training intrusion detection models that *capture features of the attack payload*, and not just features of the

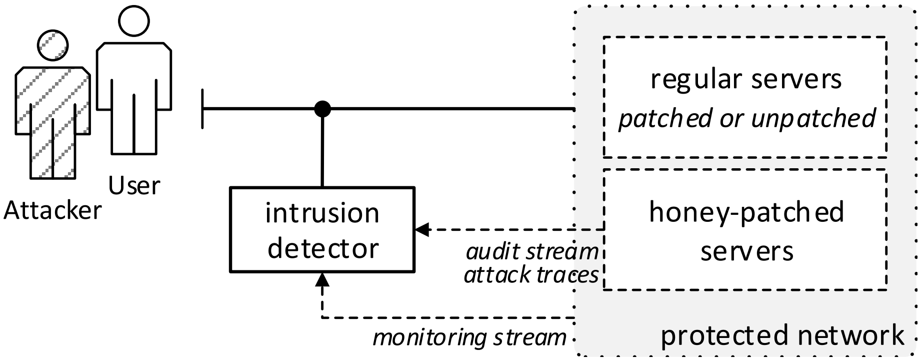


Figure 6.1: DeepDig approach overview

actual exploitation of the vulnerability, thus more closely approximating the true invariant of an attack.

To facilitate such learning, our approach classifies *sessions* as malicious, not merely the individual packets, commands, or bytes within sessions that comprise each attack. For example, observing a two-phase attack consisting of (1) exploitation of a honey-patched vulnerability, followed by (2) injection of previously unseen shellcode might train DeepDig to recognize the shellcode. Subsequent attacks that exploit an unpatched zero-day to inject the same (or similar) shellcode can then be recognized by DeepDig even if the zero-day exploit is not immediately recognized as malicious. Conventional, non-deceptive patches often miss such learning opportunities by terminating the initial attack at the point of exploit, before the shellcode can be observed.

Our central insight is that software security patches can be repurposed in an IDS setting as automated, application-level feature extractors. The maintenance of these extractors is crowd-sourced: the collective expertise of the entire software development community creates new feature extractors for free as it develops software security patches. Honey-patching transduces that collective expertise into a highly accurate, rapidly co-evolving feature extraction module for an IDS. The extractor can effortlessly detect previously unseen payloads that exploit known vulnerabilities at the application layer, which can be prohibitively difficult to detect by a strictly network-level IDS.

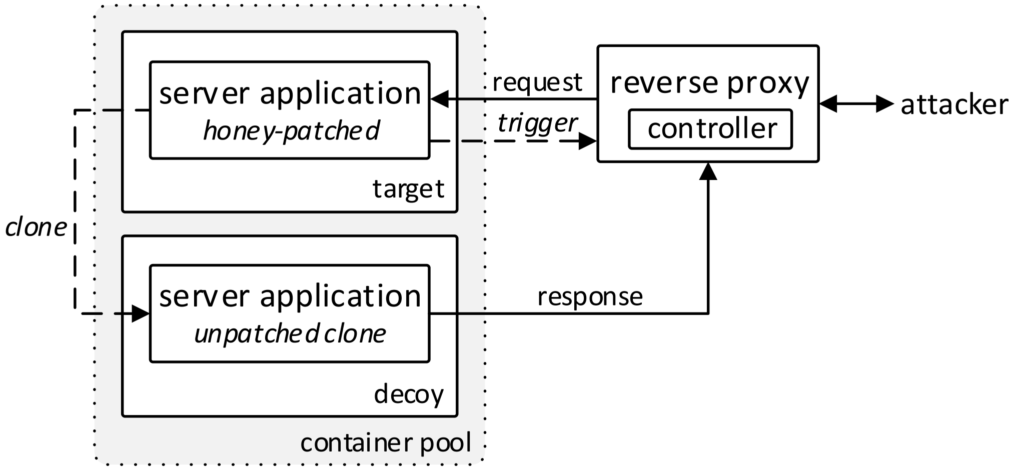


Figure 6.2: Overview of honey-patching

By living inside web servers that offer legitimate services, our deception-enhanced IDS can target attackers who use one payload for reconnaissance but reserve another for their final attacks. The facility of honey-patches to deceive such attackers into divulging the latter is useful for training the IDS to identify the final attack payload, which can reveal attacker strategies and goals not discernible from the reconnaissance payload alone. The defender's ability to thwart these and future attacks therefore derives from a synergy between the application-level feature extractor and the network-level intrusion detector to derive a more complete model of attacker behavior.

**Honey-patching** [21], depicted in Figure 6.2, adds deceptiveness to software security patches. In response to malicious inputs, honey-patched applications clone the attacker session onto a confined, ephemeral, decoy environment, which behaves henceforth as an unpatched, vulnerable version of the software. The decoy is a vulnerable replica of the victim process, optionally laced with disinformation [20]. This effectively augments the live server with an embedded honeypot that waylays, monitors, and disinforms criminals. Deceptive honey-patching capabilities thereby constitute an advanced, active defense technique that can impede, confound, and misdirect adversaries, significantly raising attacker uncertainty.

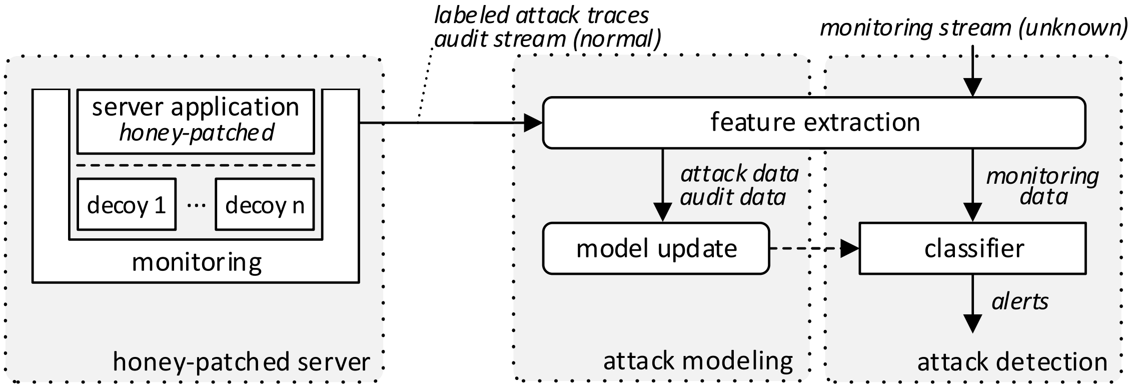


Figure 6.3: DeepDig system architecture overview

## 6.2 Architecture

DeepDig's architecture, depicted in Figure 6.3, embodies this approach by leveraging application-level threat data gathered from attacker sessions misdirected to decoys. Within this framework, developers use honey-patches to misdirect attackers to decoys that automatically collect and label monitored attack data. The intrusion detector consists of an *attack modeling* component that incrementally updates the anomaly model data generated by honey-patched servers, and an *attack detection* component that uses this model to flag anomalous activities in the monitored perimeter.

### 6.2.1 Monitoring & Threat Data Collection

The decoys into which attacker sessions are forked are managed as a pool of continuously monitored Linux containers. Each container follows the life cycle depicted in Figure 6.4. Upon attack detection, the honey-patching mechanism *acquires* the first available container from the pool. The acquired container holds an attacker session until (1) the session is deliberately closed by the attacker, (2) the connection's *keep-alive* timeout expires, (3) the ephemeral container crashes, or (4) a session timeout is reached. The last two conditions are common outcomes of successful exploits. In any of these cases, the container is released back to the pool and undergoes a recycling process before becoming available again.

After decoy *release*, the *container monitoring component* extracts the session trace (delimited by the acquire and release timestamps), labels it, and stores the trace outside the decoy for subsequent feature extraction. Decoys only host attack sessions, so precisely collecting and labeling their traces (at both the network and OS level) is effortless.

DeepDig distinguishes between three separate input data streams: (1) the *audit stream*, collected at the target honey-patched server, (2) *attack traces*, collected at decoys, and (3) the *monitoring stream*, the actual test stream collected from regular servers. Each of these streams contains network packets and operating system events captured at each server environment. To minimize performance impact, we used two powerful and highly efficient software monitors: *sysdig* (to track system calls and modifications made to the file system), and *tcpdump* (to monitor ingress and egress of network packets). Specifically, monitored data is stored outside the decoy environments to avoid possible tampering with the collected data.

### 6.2.2 Attack Modeling & Detection

Using the continuous audit stream and incoming attack traces as labeled input data, DeepDig incrementally builds a machine learning model that captures legitimate and malicious

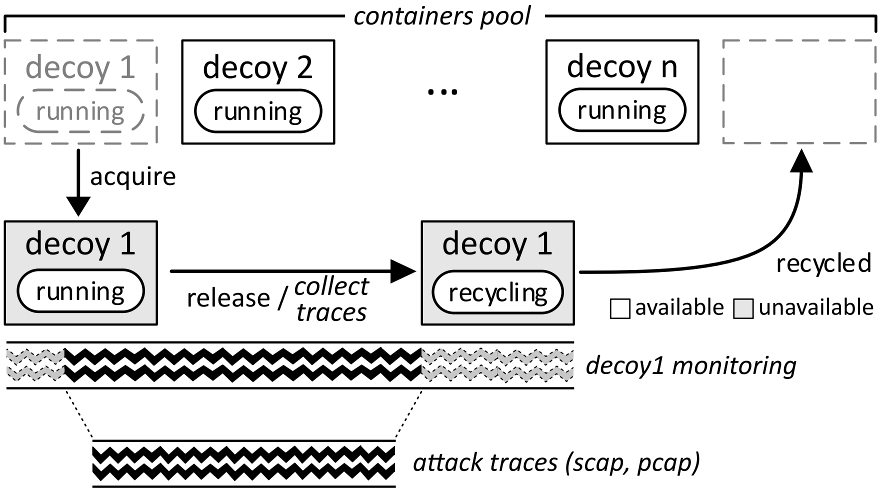


Figure 6.4: Decoy lifecycle and attack traces collection

behavior. The raw training set (composed of both audit stream and attack traces) is piped into a feature extraction component that selects relevant, non-redundant features (see §6.3) and outputs feature vectors—*audit data* and *attack data*—that are grouped and queued for subsequent model update. Since the initial data streams are labeled and have been preprocessed, feature extraction becomes very efficient and can be performed automatically. This process repeats periodically according to an administrator-specified policy. Finally, the *attack detection* module uses the most recently constructed attack model to detect malicious activity in the run-time *monitoring data*.

## 6.3 Attack Detection

To assess our framework's ability to enhance intrusion detection data streams, we have designed and implemented two feature set models: (1) *Bi-Di* detects anomalies in security-relevant network streams, and (2) *N-Gram* finds anomalies in system call traces. The model *Bi-Di* is similar to BIND (discussed in Chapter 3) with minor modifications and the terms will be used interchangeably.

### 6.3.1 Network Packet Analysis

Bi-Di (Bi-Directional) is a packet-level network behavior analysis approach that extracts features from sequences of packets and *bursts*—consecutive packets oriented to the same direction (*viz.*, uplinks from client to server, or downlinks from server to client). It uses distributions from individual burst sequences (*uni-bursts*) and sequences of two adjacent bursts (*bi-bursts*). To be robust against encrypted payloads, we limit feature extraction to packet headers.

Network packets flow between client ($Tx$) and server ($Rx$). Bi-Di constructs histograms using features extracted from packet lengths and directions. To overcome dimensionality issues associated with burst sizes, *bucketization* is applied to group bursts into correlation

Table 6.1: Packet, uni-burst, and bi-burst features

| **Category** | **Features** |
|---|---|
| Packet (Tx/Rx) | Packet length |
| Uni-Burst (Tx/Rx) | Uni-Burst size<br>Uni-Burst time<br>Uni-Burst count |
| Bi-Burst (Tx-Rx/Rx-Tx) | Bi-Burst size<br>Bi-Burst time |

sets (e.g., based on frequency of occurrence). Table 6.1 summarizes the features used in our approach. It highlights new features proposed for uni- and bi-bursts as well as features proposed in prior works [13, 57, 107, 137].

**Uni-burst features** include burst *size*, *time*, and *count*—i.e., the sum of the sizes of all packets in the burst, the amount of time for the entire burst to be transmitted, and the number of packets it contains, respectively. Taking direction into consideration, one histogram for each is generated.

**Bi-burst features** include time and size attributes of *Tx-Rx-bursts* and *Rx-Tx-bursts*. Each is comprised of a consecutive pair of downlink and uplink bursts. The size and time of each are the sum of the sizes of the constituent bursts, and the sum of the times of the constituent bursts, respectively.

Bi-bursts capture dependencies between consecutive packet flows in a TCP connection. Based on connection characteristics, such as network congestion, the TCP protocol applies flow control mechanisms (e.g., window size and scaling, acknowledgement, sequence numbers) to ensure a level of consistency between Tx and Rx. This influences the size and time of transmitted packets in each direction. Each packet flow (uplink and downlink) thereby affects the next flow or burst until communicating parties finalize the connection.

### 6.3.2 System Call Analysis

The monitored data also includes system streams comprising a collection of OS events, where each event contains multiple fields including event type (e.g., *open*, *read*, *select*), process name, and direction. Our prototype implementation was developed for Linux x86_64 systems, which exhibit about 314 distinct possible system call events. DeepDig builds histograms from these system calls using N-Gram—a system-level approach that extracts features from contiguous sequences of system calls.

There are four feature types: *Uni-events* are system calls, and can be classified as enter or exit events. *Bi-events* are sequences of two consecutive events, where system calls in each bi-event constitute features. Similarly, *tri-* and *quad-events* are sequences of three and four consecutive events (respectively).

Bi-Di and N-Gram differ in feature granularity; the former uses coarser-grained bursting while the latter uses only individual system call co-occurrences.

### 6.3.3 Classification

Bi-Di and N-Gram both use SVM for classification. Using a convex optimization approach and mapping non-linearly separated data to a higher dimensional linearly separated feature space, SVM separates positive (attack) and negative (benign) training instances by a hyperplane with the maximum gap possible. Prediction labels are assigned based on which side of the hyperplane each monitoring/testing instance belongs.

**Ens-SVM**. Bi-Di and N-Gram can be combined to obtain a better predictive model. A naïve approach concatenates features extracted by Bi-Di and N-Gram into a single feature vector and uses it as input to the classification algorithm. However, this approach has the drawback of introducing normalization issues. Alternatively, *ensemble methods* combine multiple classifiers to obtain a better classification outcome via majority voting techniques.

```
Algorithm 2: Ens-SVM
   Data: training data: $TrainX$, testing data: $TestX$
   Result: a predicted label $\mathcal{L}_{\mathcal{I}}$ for each testing instance $\mathcal{I}$
 1 begin
 2     $\mathbb{B} \leftarrow updateModel(\text{Bi-Di}, TrainX)$;
 3     $\mathbb{N} \leftarrow updateModel(\text{N-Gram}, TrainX)$;
 4     for each $\mathcal{I} \in TestX$ do
 5         $\mathcal{L}_{\mathbb{B}} \leftarrow label(\mathbb{B}, \mathcal{I})$;
 6         $\mathcal{L}_{\mathbb{N}} \leftarrow label(\mathbb{N}, \mathcal{I})$;
 7         if $\mathcal{L}_{\mathbb{B}} == \mathcal{L}_{\mathbb{N}}$ then
 8             $\mathcal{L}_{\mathcal{I}} \leftarrow \mathcal{L}_{\mathbb{B}}$;
 9         else
10             $\mathcal{L}_{\mathcal{I}} \leftarrow label\left(\arg\max_{c \in \{\mathbb{B}, \mathbb{N}\}} confidence(c, \mathcal{I}),\ \mathcal{I}\right)$;
11         end
12     end
13 end
```

For our purposes, we use an ensemble, *Ens-SVM*, which classifies new input data by weighing the classification outcomes of Bi-Di and N-Gram based on their individual accuracy indexes.

Algorithm 2 describes the voting approach for Ens-SVM. For each instance in the monitoring stream, if both Bi-Di and N-Gram agree on the predictive label (line 7), Ens-SVM takes the common classification as output (line 8). Otherwise, if the classifiers disagree, Ens-SVM takes the prediction with the highest SVM confidence (line 10). Confidence is rated using Platt scaling [113], which uses the following sigmoid-like function to compute the classification confidence:

$$P(y = 1|x) = \frac{1}{1 + \exp\left(Af(x) + B\right)} \tag{6.1}$$

where $y$ is the label, $x$ is the testing vector, $f(x)$ is the SVM output, and $A$ and $B$ are scalar parameters learned using Maximum Likelihood Estimation (MLE). This yields a probability measure of how much a classifier is confident about assigning a label to a testing point.

## 6.4 Implementation

We developed an implementation of DeepDig for 64-bit Linux (kernel 3.19). It consists of two main components: the monitoring controller and the attack detection component. The monitoring controller provides the server monitoring and attack trace extraction capabilities from decoys. It consists of about 150 lines of JavaScript code, and leverages *tcpdump*, *editcap*, and *sysdig* for network and system call tracing and preprocessing. The attack detection component is implemented as two Python modules: The feature extraction module, comprising about 1200 lines of code [111] for data preprocessing and feature generation; and the classifier component, comprising 230 lines of code that references the *Weka* [66] wrapper for LIBSVM [39]. The source-code modifications required to honey-patch vulnerabilities in Apache HTTP, Bash, PHP, and OpenSSL consist of a mere 35 lines of C code added or changed in the original server code, showing that the required deceptive capabilities can be added to production-level web services with very little effort.

## 6.5 Evaluation

This section demonstrates the practical advantages and feasibility of the deception-enhanced intrusion detection capabilities of DeepDig. First, we present our approach for generating realistic web traffic to emulate normal and malicious user behavior, which we harness to automatically generate training and test datasets for our experiments. Then, we discuss our experimental setup and investigate the effects of different attack classes and varying numbers of attack instances on the predictive power and accuracy of the intrusion detection. Finally, we assess the performance impact of the deception monitoring scheme that captures network packets and system events.

All experiments were performed on a 16-core host with 24 GB RAM running 64-bit Ubuntu 14.04 (Trusty Tahr). Regular and honey-patched servers were deployed as LXC containers [93] running atop the host using the official LXC Ubuntu template.

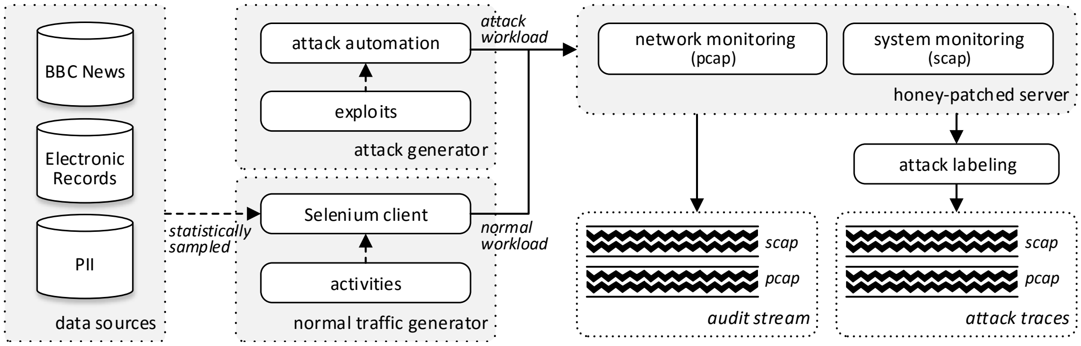


Figure 6.5: Web traffic generation and testing harness

### 6.5.1 Web Traffic Generation

To evaluate our approach, we built a web traffic generator and testing harness. Figure 6.5 shows an overview of our traffic generation framework, inspired by prior work [31]. It streams realistic *encrypted* legitimate and malicious workloads onto a honey-patched web server, resulting in labeled audit streams and attack traces (collected at decoys) for training set generation.

**Legitimate data generation**. Normal traffic is created by automating complex user actions on a typical web application, leveraging *Selenium* to automate user interaction with a web browser (e.g., clicking buttons, filling out forms, navigating a web page). We generated web traffic for 12 different user activities (each repeated 200 times), including web page browsing, e-commerce website navigation, blog posting, and interacting with a social media web application. The setup included a CGI web application and a PHP-based Wordpress application hosted on a monitored Apache web server. To enrich the set of user activities, the Wordpress application was extended with *Buddypress* and *Woocommerce* plugins for social media and e-commerce web activities, respectively.

To create realistic interactions with the web applications, our framework feeds from online data sources, such as the BBC text corpus, online text generators for personally identifiable information (e.g., usernames, passwords), and product names to populate web forms. To ensure diversity, we statistically sampled the data sources to obtain user input values and dynamically generated web content. For example, blog title and body is statistically sampled from the BBC text corpus, while product names are picked from the product names data source.

**Attack data generation**. As shown in Table 6.2, attack traffic is generated based on real vulnerabilities. For this evaluation, we selected 16 exploits for eight well-advertised, high-severity vulnerabilities. These include CVE-2014-0160 (Heartbleed), CVE-2014-6271 (Shellshock), CVE-2012-1823 (improper handling of query strings by PHP in CGI mode), CVE-2011-3368 (improper URL validation), CVE-2014-0224 (Change Cipher specification attack), CVE2010-0740 (Malformed TLS record), CVE-2010-1452 (Apache mod_cache vulnerabilty), and CVE-2016-7054 (Buffer overflow in openssl with support for ChaCha20-Poly1305 cipher suite). In addition, nine attack variants exploiting CVE-2014-6271 (Shellshock) were created to carry out different malicious activities (i.e., different attack payloads), such as leaking password files and invoking bash shells on the remote web server. These vulnerabilities are important as attack vectors because they range from sensitive data exfiltration to complete control and remote code execution. To emulate realistic attack traffic, we interleaved attacks and normal traffic following the strategy of Wind Tunnel [31].

**Dataset**. The traffic generator was deployed on a separate host to avoid interference with the test bed server. To account for operational and environmental differences, our framework simulated different workload profiles (according to time of day), against various target configurations (including different background processes and server workloads), and network settings, such as TCP congestion controls. In total, we generated 12 GB of (uncompressed) network packets and system events over a period of three weeks. After feature extraction, the training data comprised 1200 normal instances and 1600 attack instances. Monitoring or testing data consisted of 2800 normal and attack instances gathered at unpatched web servers, where the distribution of normal and attack instances varies per experiment.

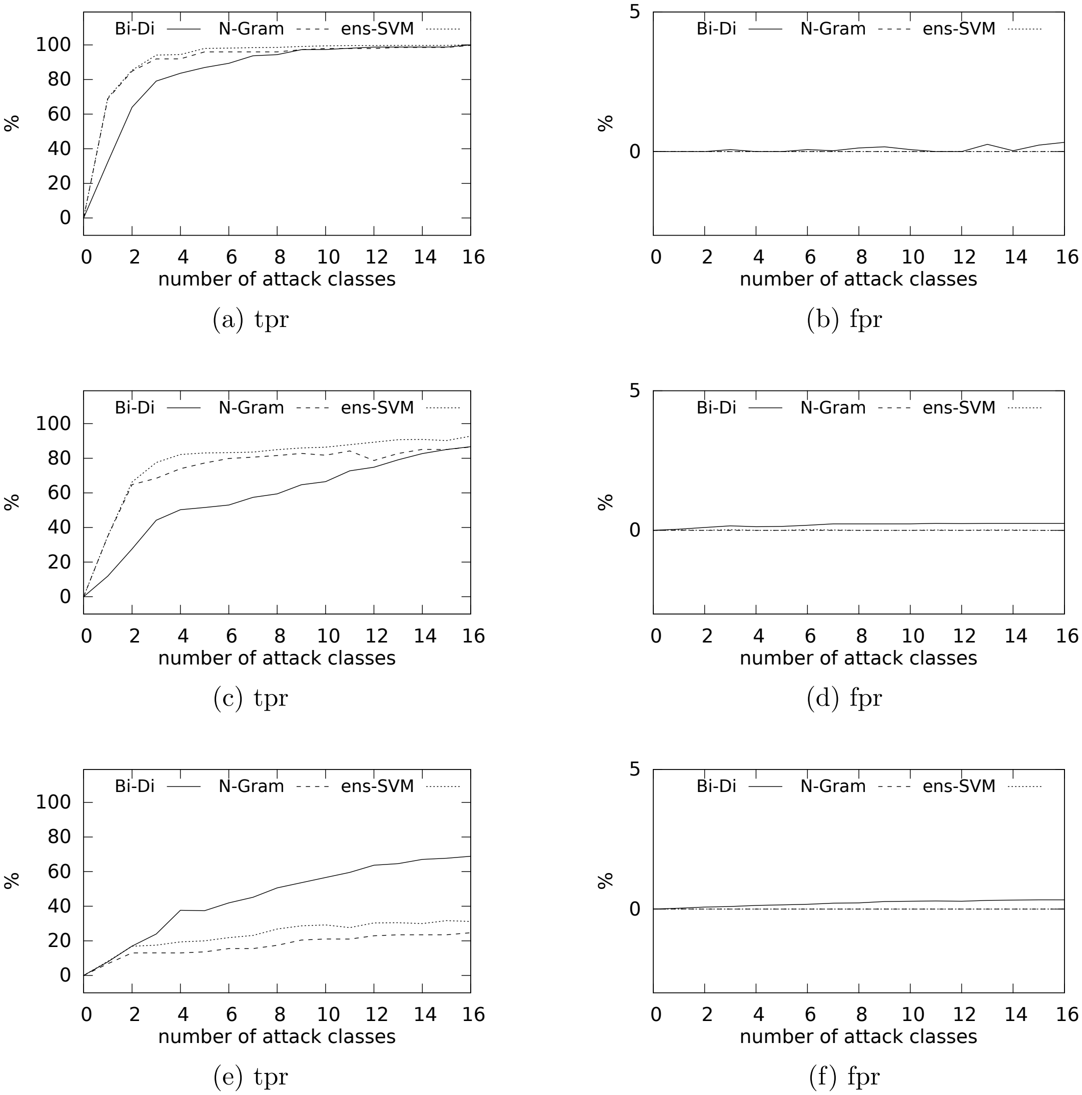
Table 6.2: Summary of attack workload

| # | **Attack Type** | **Description** | **Software** |
|---|---|---|---|
| 1 | CVE-2014-0160 | Information leak | Openssl |
| 2 | CVE-2012-1823 | System remote hijack | PHP |
| 3 | CVE-2011-3368 | Port scanning | Apache |
| 4–10 | CVE-2014-6271 | System hijack (7 variants) | Bash |
| 11 | CVE-2014-6271 | Remote Password file read | Bash |
| 12 | CVE-2014-6271 | Remote root directory read | Bash |
| 13 | CVE-2014-0224 | Session hijack and information leak | Openssl |
| 14 | CVE-2010-0740 | DoS via NULL pointer dereference | Openssl |
| 15 | CVE-2010-1452 | DoS via request that lacks a path | Apache |
| 16 | CVE-2016-7054 | DoS via heap buffer overflow | Openssl |

### 6.5.2 Experimental Results

Using this dataset, we trained the classifiers presented in §6.3 and assessed their individual performance against test streams containing both normal and attack workloads. In the experiments, we measured the true positive rate (*tpr*) where true positive represents the number of actual attack instances that are classified as attacks, false positive rate (*fpr*) where false positive represents the number of actual benign instances classified as attacks, accuracy (*acc*), and $F_2$ score of the classifier, where the $F_2$ score is interpreted as the weighted average of the precision and recall, reaching its best value at 1 and worst at 0. An RBF kernel with $Cost = 1.3 \times 10^5$ and $\gamma = 1.9 \times 10^{-6}$ was used for SVM [107].

**Detection accuracy**. To evaluate the accuracy of intrusion detection, we verified each classifier after incrementally training it with increasing numbers of attack classes. Each class consists of 100 distinct variants of a single exploit, as described in §6.5.1, and an $n$-class model is one trained with up to $n$ attack classes. For example, a 3-class model is trained with 300 instances from 3 different attack classes. In each run, the classifier is trained with 1200 normal instances and $100 * n$ attack instances where $n \in [1, 16]$ attack classes. In addition, in each run, we execute ten experiments where the attacks are shuffled in a cross-validation-like fashion and the average is reported. This ensures training is not biased toward any specific attacks.



(a) tpr

(b) fpr

(c) tpr

(d) fpr

(e) tpr

(f) fpr

Figure 6.6: DeepDig classification accuracies for 0–16 attack classes for (a)–(b) training and testing on decoy data, (c)–(d) training on decoy data and testing on unpatched server data, and (e)–(f) training on regular-patched server data and testing on unpatched server data.

*Testing on decoy data.* The first experiment measures the accuracy of each classifier against a test set composed of 1200 normal instances and 1600 uniformly distributed attack instances gathered at decoys. Figure 6.6(a)–(b) presents the results, which serve as a preliminary (sanity) check that the classifiers can accurately detect attack instances resembling the ones comprised in their initial training set. The attack instances used to train the classifiers were sourced from decoys.

*Testing on unpatched server data.* The second experiment also measures each classifier's accuracy, but this time the test set was derived from monitoring streams collected at regular, *unpatched* servers, and having a uniform distribution of attacks. Figure 6.6(c)–(d) shows the results, which indicate that the detection models of each classifier generalize beyond data collected in decoys. This is critical because it demonstrates the classifier's ability to detect previously unseen attack variants. DeepDig thus enables administrators to add an additional level of protection to their entire network, including hosts that cannot be promptly patched, via the adoption of a honey-patching methodology. The attack instances used to train the classifiers were sourced from decoys.

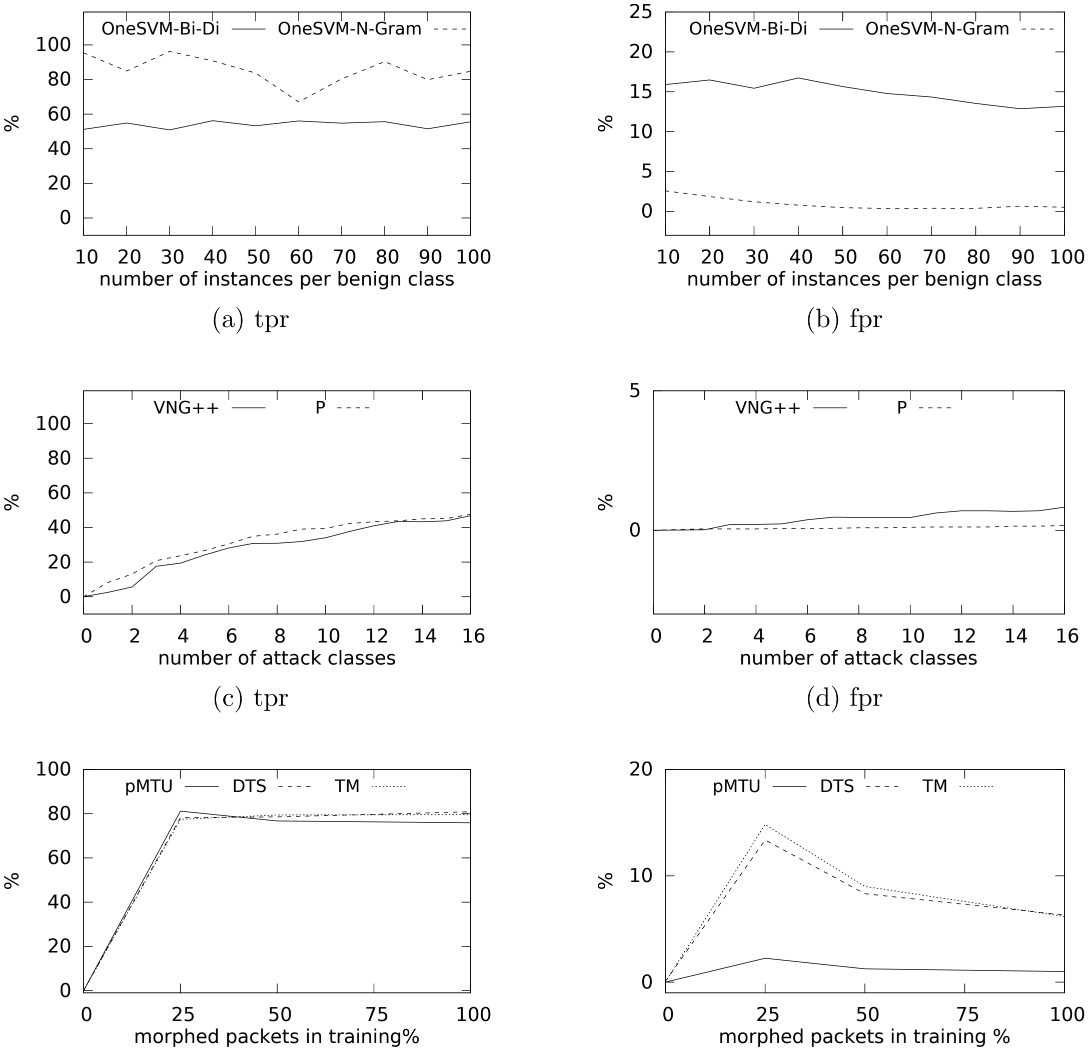
The results also show that as the number of training attack classes increases—which are proportional to the number of vulnerabilities honey-patched—a steep improvement in the true positive rate of both classifiers is observed, reaching an average *tpr* of above 92% for the compounded Ens-SVM, while average false positive rate in all experiments remained below 0.01%. This demonstrates the positive impact of the *feature-enhancing* capabilities of deceptive application-level attack responses like honey-patching.

*Training without deceptive-enhanced data.* To compare DeepDig against analogous, standard IDSes that do not employ deception, we trained each classifier on data collected from non-deceptive, regular-patched servers, and tested them on the unpatched server data, using the same set of attacks. Figure 6.6(e)–(f) shows the results, which outline the inherent challenges of traditional intrusion detection models on obfuscated, unlabeled attack traces. Unlike honey-patches, which capture and label traces containing patterns of successful attacks, conventional security patches yield traces of failed attack attempts, making them unfit to reveal patterns of attacks against unpatched systems.

These results highlight a key advantage of our approach: it enables timely evolution of intrusion detection models via run-time labeling of attack data streams. They also demonstrate that our framework is impervious to network encryption and obfuscation techniques. Although widespread, standard middleboxes and IDS products that rely on TLS/HTTPS interception techniques for retaining visibility of network traffic pose a real threat to organizations by introducing severe vulnerabilities and reducing connection security [48, 56]. By leveraging application-level traps as automated feature extractors for intrusion detection, DeepDig overcomes many of the practical challenges associated with the analysis of encrypted traffic.

**Baseline evaluation**. This experiment compares the accuracy of our detection approach to the accuracy of an unsupervised outlier detection strategy, which is commonly employed in typical intrusion detection scenarios [38], where labeling attack data is not feasible or prohibitively expensive. For this purpose, we implemented two *One-class SVM* classifiers, *OneSVM-Bi-Di* with a polynomial kernel and $\nu = 0.1$ and *OneSVM-N-Gram* with a linear kernel and $\nu = 0.001$, using Bi-Di and N-Gram models for feature extraction, respectively. We fine tuned the One-class SVM parameters and performed a systematic grid search for the kernel and $\nu$ to get the best results.

One-class SVM uses an unsupervised approach, where the classifier trains on one class and predicts whether a test instance belongs to that class, thereby detecting *outliers*—test instances outside the class. To perform this experiment, we incrementally trained each classifier with an increasing number of *normal* instances, and tested the classifiers after each iteration against the same unpatched server test set used in the previous experiments. The results presented in Fig. 6.7(a)–(b) highlight critical limitations of conventional outlier intrusion detection systems: reduced predictive power, lower tolerance to noise in the training set, and higher false positive rates.



(a) tpr

(b) fpr

(c) tpr

(d) fpr

(e) tpr

(f) fpr

Figure 6.7: Baseline evaluations: (a)–(b) OneSVM-Bi-Di and OneSVM-N-Gram, and (c)–(d) VNG++ and Panchenko (cf. Fig. 6.6(c)–(d)). Resistance to attack evasion: (e)–(f) DeepDig accuracy when training the classifier with increasingly large proportions (0–100%) of morphed packets.

In contrast, our supervised approach overcomes such disadvantages by automatically streaming onto the classifiers labeled security-relevant features, without any human intervention. This is possible because honey-patches identify security-relevant events at the point where such events are created, and not as a separate, *post-mortem* manual analysis of traces.

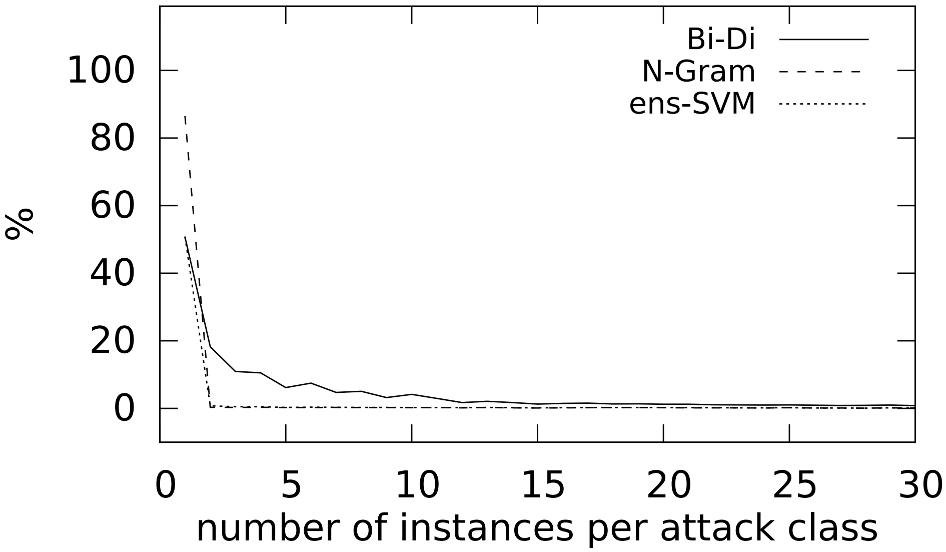


Figure 6.8: False positive rates for various training set sizes

**Comparison to previous approaches**. This experiment extends our baseline assessment to previous supervised approaches on encrypted network traffic. Towards this end, we adapted two state-of-the-art supervised approaches [57, 107], which are widely used in the literature on encrypted traffic analysis [82]. Figure 6.7(c)–(d) shows the results for VNG++ [57] and Panchenko (P) [107], which underscore perennial challenges encountered in this domain: reduced detection accuracy and high incidence of false alarms.

**False alarms**. To evaluate the fpr-reducing effects of DeepDig, we trained each classifier with data sets containing 1–30 normal/attack instances per class in 30 incremental training iterations. We tested each classifier after every iteration step and plotted the results in Figure 6.8. Observe that with just a few attack instances (≈ 5 per attack class), the false positive rates dropped to close to zero percent, demonstrating DeepDig's continuous feeding back of attack samples into classifiers greatly reduces false alarms.

### 6.5.3 Base Detection Analysis

In this section we measure the success of DeepDig in detecting intrusions in the realistic scenario where attacks are a small fraction of the interactions. Although risk-level attribution for cyber attacks is difficult to quantify in general, we use the results of a recent study [55] to approximate the probability of attack occurrence for the specific scenario of *targeted attacks against business and commercial organizations.* The study's model assumes a determined attacker leveraging one or more exploits of known vulnerabilities to penetrate a typical organization's internal network, and approximates the *prior* of a directed attack to $P_A = 1\%$ (using threat statistics from 2011).

In a similar way that we used in §3.2.3, to estimate the success of intrusion detection, we use a *base detection rate* (*bdr*) [75], expressed using the Bayes theorem as:

$$P(A|D) = \frac{P(A)\ P(D|A)}{P(A)\ P(D|A) + P(\neg A)\ P(D|\neg A)]}, \tag{6.2}$$

where $A$ and $D$ are random variables denoting the occurrence of a targeted attack and the detection of an attack by the classifier, respectively. We use *tpr* and *fpr*, from Figure 6.6(c)–(d) and Figure 6.7(a)–(b) and (c)–(d), as approximations of $P(D|A)$ and $P(D|\neg A)$, respectively.

Table 6.3 presents the accuracy values and *bdr* for each classifier, assuming $P(A) = P_A$. The numbers expose a practical problem in intrusion detection research: Despite having high accuracy values, typical intrusion detection systems are rendered ineffective when confronted with their staggering low base detection rates. This is in part due to their intrinsic inability to eliminate false positives in operational contexts. In contrast, the *fpr*-reducing properties of our framework—i.e., the ability to suppress false alarms through automatic labeling of network- and system-level attack features—affords the construction of detection systems that can detect intrusions much more effectively in realistic settings.

Table 6.3: Base detection rate percentages for an approximate targeted attack scenario ($P_A \approx 1\%$) [55]

| **Classifier** | ***tpr*** | ***fpr*** | ***acc*** | $\boldsymbol{F_2}$ | ***bdr*** |
|---|---|---|---|---|---|
| OneSVM-Bi-Di | 55.56 | 13.17 | 68.96 | 59.69 | 4.09 |
| OneSVM-N-Gram | 84.77 | 0.52 | 91.07 | 87.09 | 62.22 |
| VNG++ | 46.81 | 0.83 | 69.25 | 52.31 | 36.29 |
| Panchenko | 47.69 | 0.17 | 70.04 | 53.24 | 73.92 |
| Bi-Di | 86.69 | 0.25 | 92.29 | 89.02 | 77.79 |
| N-Gram | 86.52 | 0.01 | 92.30 | 88.89 | 98.98 |
| Ens-SVM | **92.76** | **0.01** | **95.86** | **94.12** | **99.05** |

### 6.5.4 Resistance to Attack Evasion

In this section, to further test the robustness of our approach against an adversarial setting, where attackers deliberately try to confuse the classifier (e.g., by performing benign activities in the decoy environment after sending their attacks), we conduct experiments where the attacks are morphed to resemble benign traffic. In our analysis, we considered three encrypted traffic evasion techniques: *Pad-to-MTU*, *Direct Target Sampling*, and *Traffic Morphing*. Pad-to-MTU (pMTU) [57] adds extra bytes to each packet length until it reaches the Maximum Transmission Unit (1500 bytes in the TCP protocol). Direct Target Sampling (DTS) [143] is a distribution-based technique that uses statistical random sampling from benign traffic followed by attack packet length padding. Traffic Morphing (TM) [143] is similar to DTS but it uses a convex optimization methodology to minimize the overhead of padding.

Table 6.4 shows the results of the Ens-SVM classifier against these evasion techniques. The table also shows the results when no evasion is considered. In each experiment, the classifier is trained with 1200 normal instances and 1600 *morphed* attack instances. Similarly, the test set consists of 1200 normal instances and 1600 *morphed* attack instances. These results show that DeepDig is able to resist adversarial scenarios with considerable accuracy.

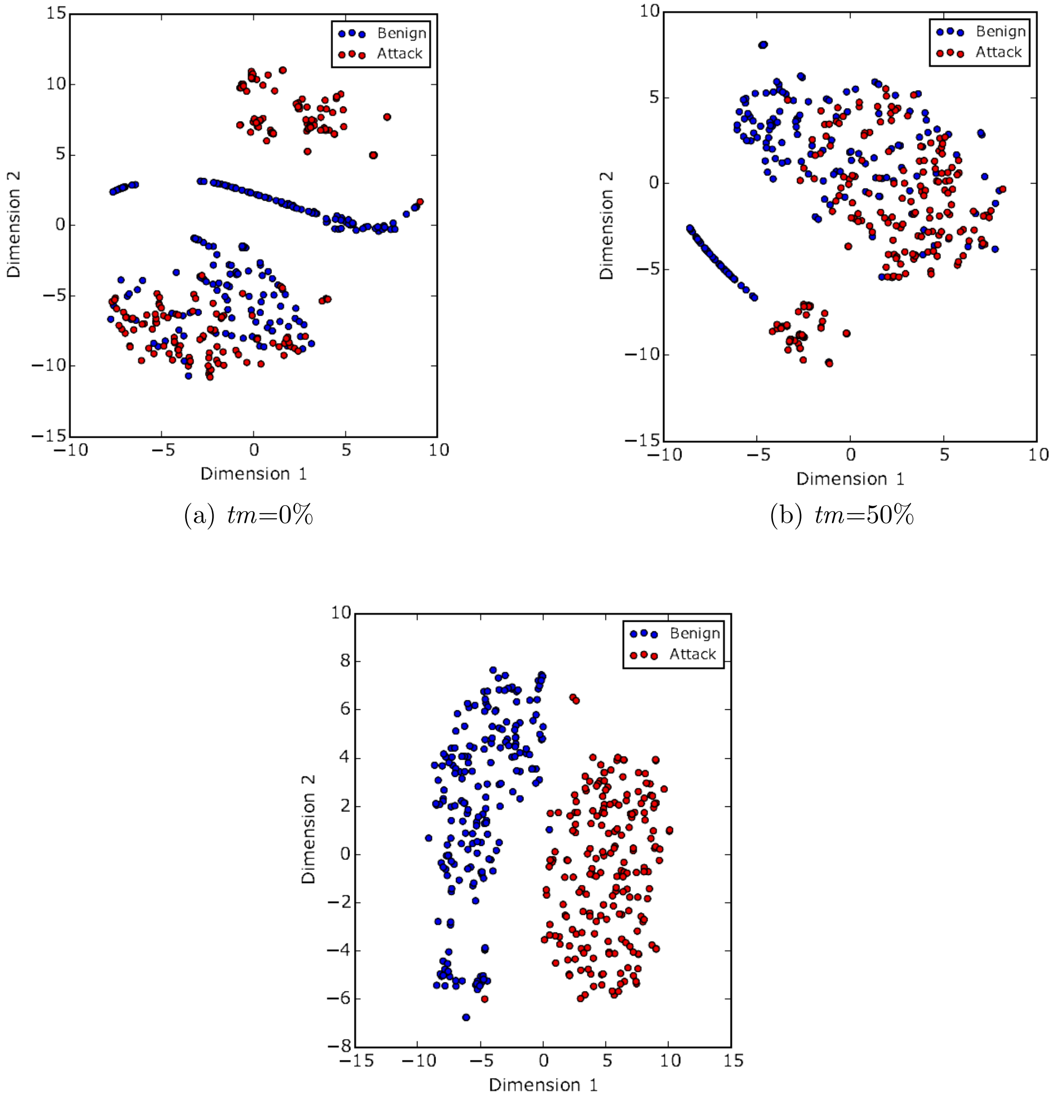
Table 6.4: Detection performance in adversarial settings

| **Evasion technique** | ***tpr*** | ***fpr*** | ***acc*** | $\boldsymbol{F_2}$ |
|---|---|---|---|---|
| No evasion | **86.69** | **0.25** | **92.29** | **89.02** |
| pMTU | 75.84 | 0.96 | 85.78 | 79.57 |
| DTS | 82.78 | 6.02 | 87.58 | 84.91 |
| TM | 79.29 | 6.17 | 85.52 | 81.91 |

*Analysis.* Since our goal is not to classify individual packets as attacks, but mine attack patterns from entire data streams, mixing benign with malicious activities in the decoy environment does not impair DeepDig's ability to learn attacker patterns, even in the presence of evasive behavior. In the above experiments, we trained the classifier in the presence of attacker evasion. This is practical and reveals that DeepDig captures the entirety of the attacker's activity, feeding it back to the classifier.

Figure 6.7(e)–(f) shows the measured *tpr* and *fpr* when gradually training the classifier with increasingly large proportions of morphed packets in the training set. The horizontal axis represents the percentage of the morphed packets in the training phase. For instance, 25% signifies that the classifier was trained with 1/4 of morphed packets and 3/4 of non-morphed packets.

Although *tpr* remains stable after 25% of morphed traffic, the results highlight an improvement for the false positive rate. This underscores the positive impact of honey-patching on overcoming adversarial behavior: The more morphed, evading samples attackers feed DeepDig, the better the IDS becomes in classifying future attack patterns. Thus, adversarial attempts to obfuscate attacks against honey-patched vulnerabilities are actually a gift to the defender, since they only help train the classifier to learn the attacker's obfuscation strategies. Figure 6.9 illustrates this by showing the convergence of the classifier's decision boundary as it observes morphed samples.



(a) *tm*=0%

(b) *tm*=50%

(c) *tm*=100%

Figure 6.9: High-dimensional visualization of decision boundary convergence in the presence of evasion, showing traffic morphing (*tm*) at 0%, 50%, and 100%. t-SNE transformation [134] was used to reduce dimensionality to two dimensions for this visualization.

### 6.5.5 Monitoring Performance

To assess the performance overhead of DeepDig's monitoring capabilities, we used *ab* (Apache HTTP server benchmarking tool) to create a massive user workload (more than

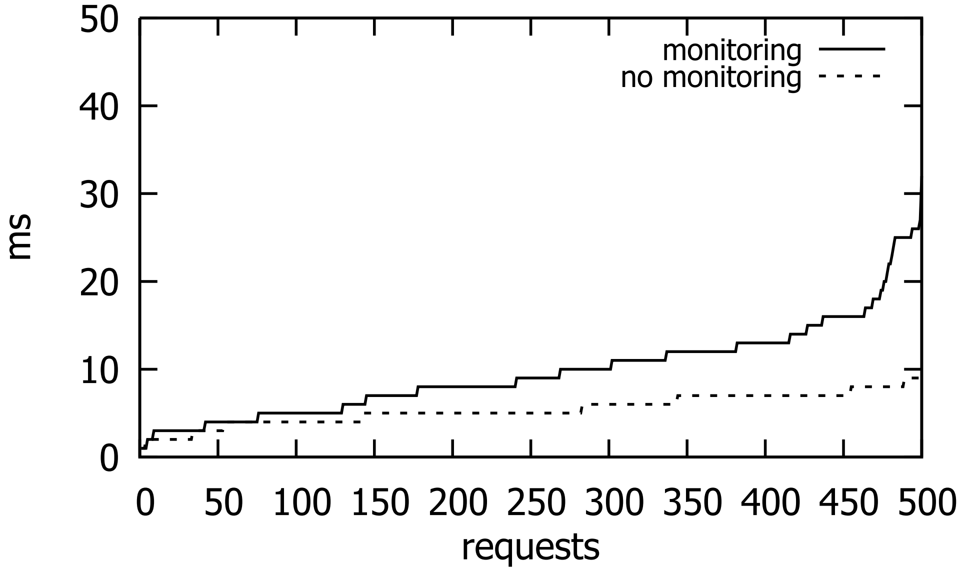


Figure 6.10: DeepDig performance overhead measured in average round-trip times (workload ≈ 500 req/s)

5,000 requests in 10 threads) against two web server containers, one deployed with network and system call monitoring and another unmonitored.

Figure 6.10 shows the results, where web server response times are ordered ascendingly. Our measurements show average overheads of 0.2×, 0.4×, and 0.7× for the first 100, 250, and 500 requests, respectively, which is expected given the heavy workload profile imposed on the server. Since server computation accounts for only about 10% of overall web site response delay in practice [125], this corresponds to observable overheads of about 2%, 4%, and 7% (respectively).

While such overhead characterizes feasibility, it is irrelevant to deception because unpatched, patched, and honey-patched servers are all slowed equally by the monitoring activity. The overhead therefore does not reveal which apparent vulnerabilities in a given server instance are genuine patching lapses and which are deceptions, and it does not distinguish honey-patched servers from servers that are slowed by any number of other factors (e.g., fewer computational resources).

## 6.6 Discussion

**Methodology**. Our experiments show that just a few strategically chosen honey-patched vulnerabilities accompanied by an equally small number of honey-patched applications provide a machine learning-based IDS sufficient data to perform substantially more accurate intrusion detection, thereby enhancing the security of the entire network. Thus, we arrive at one of the first demonstrable measures of value for deception in the context of cyber security: its utility for enhancing IDS data streams.

**Supervised learning**. Our approach facilitates supervised learning, whose widespread use in the domain of intrusion detection has been impeded by many challenges involving the manual labeling of attacks and the extraction of security-relevant features [38]. Our results demonstrate that the language-based, active response capabilities provided via application-level honey-patches significantly ameliorates both of these challenges. The facility of deception for improving other machine learning-based security systems should therefore be investigated.

**Generalization**. The results presented in §6.5 show that our approach substantially improves the accuracy of intrusion detection, reducing false alarms to much more practical levels. Although we used many variations of well-known attacks and showed how DeepDig generalizes when increasing the pool of attacks, future work should explore larger numbers of attack classes to simulate threats to high-profile targets. Due to the high-dimentional nature of the collected data, we chose SVM in Bi-Di and N-Gram. Linearly separating such data is complicated by various feature interactions, such as network burst sequences and system IO events. SVM is suitable for this task as it maps non-linear data points to another linearly separable feature space using the *kernel trick*.

**Class imbalance**. Standard concept-learning IDSes are frequently challenged with imbalanced datasets [69]. Such class imbalance problem arises when benign and attack classes are not equally represented in the training data, since machine learning algorithms tend to misclassify minority classes. To mitigate the effects of class imbalance, sampling techniques have been proposed [41], but they often discard useful data (in the case of under-sampling), or lead to poor generalizations (in the case of oversampling). Thus, this scarcity of realistic, balanced datasets has heretofore greatly contributed to hinder the applicability of machine learning approaches for web intrusion detection. By feeding back labeled attack traces into the classifier, DeepDig alleviates this data drought and enables the generation of adequate, balanced datasets for classification-based intrusion detection.

**Intrusion detection datasets**. One of the major challenges in evaluating intrusion detection systems is the dearth of publicly available datasets, which is often aggravated by privacy and intellectual property considerations. To mitigate this problem, security researchers often resort to synthetic dataset generation, which affords the opportunity to design test sets that validate a wide range of requirements. Nonetheless, a well-recognized challenge in custom dataset generation is how to capture the multitude of variations and features manifested in real-world scenarios [29]. Our evaluation approach builds on recent breakthroughs in dataset generation for IDS evaluation [31] to create statistically representative workloads that resemble realistic web traffic, thereby affording the ability to perform a meaningful evaluation of IDS frameworks.

**Evaluation**. Establishing a straight comparison of our results to prior work can be very challenging. The majority of machine learning-based intrusion detection techniques are still tested on extremely old datasets [12, 124], and approaches that account for encrypted traffic are scarce [82]. For instance, recently-proposed SVM-based approaches for network intrusion detection have reported true positive rates in the order of 92% for the DARPA/KDD datasets, with false positive rates averaging 8.2% [94, 146]. Using the model discussed in §6.5.3, this corresponds to an approximate base detection rate of only 11%, in contrast to 99.05% estimated for our approach. However, such comparison can lead to erroneous conclusions, as the assumptions made by DARPA/KDD do not reflect the contemporary attack protocols and recent vulnerabilities targeted by our model.

# CHAPTER 7

# CONCLUSION [1] [2] [3] [4]

## 7.1 Dissertation Summary

This dissertation advanced and enhanced cyber security using encrypted traffic fingerprinting techniques. We introduced, implemented, and evaluated BIND, a new data analysis method on encrypted network traffic for end-node identification. The method leverages dependence in packet sequences to extract characteristic features suitable for classification. In particular, we study two cases where our method is applicable: website fingerprinting and app fingerprinting. We empirically evaluate both these cases in the closed-world and open-world settings on various real-world datasets over HTTPS and Tor. Empirical results indicate the effectiveness of BIND in various scenarios including the realistic open-world setting. Our evaluations also include cases where defense mechanisms are applied on website and app fingerprinting. We showed how the proposed approach achieves a higher performance com-

[1] This chapter contains material previously published as: K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K.W. Hamlen, and B. Thuraisingham. "Adaptive encrypted traffic fingerprinting with bi-directional dependence." In Proceedings of the 32nd Annual Computer Security Applications Conference, pp. 177-188. ACM, 2016. DOI: https://doi.org/10.1145/2991079.2991123. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[2] This chapter contains material previously published as: ©2015 IEEE. Portions Reprinted, with permission, from K. Al-Naami, G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan and B. Thuraisingham. "P2V: Effective Website Fingerprinting Using Vector Space Representations." IEEE Symposium Series on Computational Intelligence, December 2015. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[3] Some of the work presented in this chapter was performed in collaboration with L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Lead author Al-Naami conducted the majority of the research, including most of the design, the full implementation, and the full evaluation.

[4] Some of the work presented in this chapter was performed in collaboration with F. Araujo, A. Gbadebo, A. Mustafa, L. Khan, and K.W. Hamlen at the University of Texas at Dallas. This work is currently submitted for publication. Al-Naami led the machine learning half of the research, including feature generation, classification design, implementation, and experiments.

pared to other existing techniques. In addition, we introduced the AdaBind approach that addresses temporal changes in data patterns over time while performing traffic fingerprinting.

To advance the defense in website fingerprinting, we proposed the P2V model which views network packets as words. We showed how the proposed model can be used to improve the website fingerprinting accuracy and defeat the existing defenses. Our experimental results showed that our proposed attack can achieve higher accuracy in identifying a website from a given trace, than claimed in previous studies. The experimental results showed also our new approach is more resilient to website fingerprinting defenses than previous works.

To defeat encrypted traffic fingerprinting attacks, we proposed the BiMorphing defense which combines size and time bi-directional dependence sampling, ensures low bandwidth overhead through the use of mathematical optimization, and incurs zero delay for real packets exchanged between client and server. We proved the effectiveness of the proposed approach empirically by examining the defense against passive attacks and comparing it with state-of-the-art methods. The promising results, low bandwidth overhead, and real packets zero latency give a new perspective for a more practical fingerprinting defense.

This work introduced, implemented, and evaluated a new approach for enhancing web intrusion detection systems with threat data sourced from deceptive, application-layer, software traps. Unlike conventional machine learning-based detection approaches, DeepDig incrementally builds models of legitimate and malicious behavior based on audit streams and traces collected from these traps. This augments the IDS with inexpensive and automatic security-relevant feature extraction capabilities. These capabilities require no additional developer effort apart from routine patching activities. This results in an effortless labeling of the data and supports a new generation of higher-accuracy detection models.

## 7.2 Future Work

In this section, we discuss possible avenues of future directions to advance each of our proposed works.

### 7.2.1 Traffic Fingerprinting

The introduced AdaBind method updates the model with new training batches which requires a significant number of training instances. Furthermore, the re-training process assumes the availability of testing instance labels which may not be valid in certain cases. To address these challenges, in future we would like to identify the right point in the incoming stream from where we need to re-train the model incrementally (i.e., keeping old useful data) in an unsupervised manner (i.e., without labels). Hence, one of the future directions of BIND is to apply the concept of *Change Point Detection* (CPD) [67, 68] to decide when to update the models in an unsupervised fashion and re-train incrementally.

The proposed methods in our dissertation assume sequential user access to end-nodes and ignore background noise, as mentioned in §2.1.1 regarding Wfin [75]. Nevertheless, these methods can be augmented with techniques relaxing such assumptions. We also note that such assumptions are applicable to Afin as well. In a smartphone, multiple apps may run background services, such as auto-sync, within the device that access the Internet periodically. Moreover, services offered by an app can change over time with newer versions released by developers periodically. Each updated version of an app may have dissimilar network signature or fingerprint, which could affect classifier performance as well. Furthermore, exploring different activities of an app would generate different network signatures compared to a signature obtained by merely launching it. One could use dynamic analysis techniques [126, 28] to explore an app automatically for a better understanding of network behaviors. We leave these for future work.

### 7.2.2 Cyber-Deceptive Intrusion Detection

**Streaming Data**. As data is continuously hitting IDSes in streaming manner, it is necessary to have online and fast classification. Major challenges are the notions of *concept-drift* and *delayed labeling* [67]. Concept-drift occurs as data characteristics change over time which requires updating the classification model periodically. Updating the model requires data points that are readily labeled (as benign or attack) in order to feed them back for the learning process to discard obsolete models. Indeed, DeepDig is suitable for this task as it labels data in a near zero buffering manner without the need for human intervention. This is a substantial advantage that means updating classification models now can occur instantly to detect attacker's new behavior that have not been learned previously. One of the future directions of DeepDig is to apply the concept of *Change Point Detection* (CPD) [67, 68] to decide when to update the models. CPD ensures that instead of taking fixed chunk sizes to update the model which is an expensive process, a sliding window technique is used to track any significant change to data. Not only this ensures updating the model in a precise manner (i.e., if only a concept-drift happened), but also it does not miss changes that may occur in small chunk sizes.

**Feature Enhancement**. In addition to the encrypted packet- and system-level features we presented in this work, one direction of future work is to utilize system call arguments as well. These arguments can be fed into certain techniques such as implementing a $k$-Nearest Neighbor ($k$-NN) approach which employs longest common subsequence (LCS) as its distance measure. In general, the diversity of packet- and system-level information can be explored as they contain many more discriminating features.

# APPENDIX

# MAPREDUCE GUIDED SPATIAL QUERY PROCESSING AND ANALYTICS SYSTEM[1][2]

This appendix presents some of my prior research works in "geo-spatial query system using big data" that had been conducted and published during the course of the Doctoral study.

## A.1 Summary

The Global Database of Event, Language, and Tone (*GDELT*) is the only global political georeferenced event dataset with more than 250 million observations covering all countries in the world since January 1, 1979. TABARI and CAMEO are the tools which are used to collect and code events from all international news coverage. To query such big geospatial data, traditional RDBMS can no longer be used and the need for parallel distributed solutions has become a necessity. MapReduce paradigm has proved to be a scalable platform to process and analyze Big Data in the cloud. Hadoop, as an implementation of MapReduce, is an open source application that has been widely used and accepted in academia and industry. However, when dealing with Spatial Data, Hadoop is not equipped well and does not perform efficiently. SpatialHadoop is an extension of Hadoop with the support of spatial data. In this work, we present Geographic Information System Query and Analytics Framework (*GISQAF*) [15, 16] which has been built on top of SpatialHadoop. GISQAF focuses on two

[1] This appendix contains material previously published as: ©2014 IEEE. Reprinted, with permission, from K. Al-Naami, S. E. Seker and L. Khan. "GISQF: An Efficient Spatial Query Processing System." 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, 2014, pp. 681-688. doi: 10.1109/CLOUD.2014.96. Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

[2] This appendix contains material previously published as: ©2016 by John Wiley & Sons, Inc., or related companies. Reprinted, with permission, from K. Al-Naami, S. E. Seker, and L. Khan. "GISQAF: MapReduce Guided Spatial Query Processing and Analytics System." Software Practice and Experience, 46: 13291349. doi: 10.1002/spe.2383 (2016). Lead author Al-Naami conducted the majority of the research, including most of the design, implementation, and evaluation.

parts: Query Processing (*QP*) and Data Analytics (*DA*). For the Query Processing part, we show how this solution outperforms Hadoop query processing by orders of magnitude when applying queries on GDELT dataset with a size of 60 GB. We show the results for various types of queries. For the Data Analytics part, we present an approach for finding Spatial co-occurring events. We show how GISQAF is suitable and efficient to handle Data Analytics techniques.

## A.2 Introduction

Living in the Big Data era, one can see the enormous growth of data that has reached an explosion. The advances in technology have resulted in ease of collecting data from numerous resources. Location-aware and position-based devices generate huge amounts of geospatial datasets which have led to the need to extract, process, and query such information in a timely and efficient manner. GDELT, Global Data of Events, Language and Tone, is a publicly available dataset with more than 250 million observations (called events) with global coverage since 1979 [4]. This dataset contains records and observations from all international news coverage. TABARI [78] and CAMEO [116] are the systems used to collect and prepare the GDELT dataset. The purpose of this widely used dataset is to help understand and uncover trends and behaviors of the social and international system [4].

The challenge is how to query such Big Data with a satisfactory performance. Traditional RDBMS can no longer be used [62, 37] because of scalability and query time issues. Furthermore, as storing the GDELT data follows the "write once read many" concept, there is no need to worry about updates. Thus, parallel distributed solutions based on "Not only SQL" (NoSQL) have emerged [74]. Examples of these solutions include Hadoop [2], BigTable [40] Cassandra [85], and others.As a matter of fact, this is why many giant corporations have replaced traditional RDBMS with parallel distributed solutions. In their work [132], prior to 2008, Facebook query processing was built on commercial RDBMS which became not only expensive but also inefficient to handle the growth of the daily data generated by users and thus they have switched to Hadoop [2].

Google [49] has invented a MapReduce programming model as a parallel data processing paradigm which has proved to be a scalable and reliable platform. Hadoop [2] is a popular open source implementation of MapReduce that has been used for years in academia and industry [90]. On the other hand, Hadoop has some limitations. Previous attempts have been made to tune Hadoop. Tan et al. [130] presented a solution to fine-tune Hadoop to process input data efficiently. Liao et al. [90] proposed an event trigger mechanism to refine the Hadoop system. As studied in [9, 26, 87], one of the MapReduce limitations is that it is schema-free and index-free as it requires to evaluate each record when consuming input, causing performance degradation. Files are distributed into blocks which are stored in multiple nodes. When querying these files, MapReduce jobs are distributed across nodes in parallel. However, within each node, Map jobs process records in a sequential manner which takes long processing times. As a result, Hadoop has not been suited to process Spatial Data that obviously require spatial indexing techniques such as grid index [105], or R-tree [65]. As an alternative, SpatialHadoop [59] is an open-source Hadoop-extended framework for processing spatial datasets efficiently. SpatialHadoop has been developed on top of Hadoop so for regular MapReduce jobs, it runs exactly as Hadoop but it has the awareness of Spatial Data when it encounters spatial constructs and operations. Unfortunately, although SpatialHadoop is well-suited for Spatial Data, it has not been tuned to handle schema-like spatial datasets with such spatial archives. An example is the GDELT dataset which has events represented by a schema.

In this work, we introduce GISQAF (Geographic Information System Query and Analytics Framework) that extends SpatialHadoop [15]. GISQAF achieves two objectives: Query Processing (QP) and Data Analytics (DA). The contributions of this work are as follows.

1. This work extends SpatialHadoop framework so it can handle heterogeneous datasets. As a matter of fact, trying to integrate GDELT dataset into SpatialHadoop by itself will not work as SpatialHadoop does not support schema-like datasets with multiple attributes and longitude/latitude events. In this work, we convert these events into geometry EventPoint shapes and show how SpatialHadoop has been customized and extended to index, decode, and query the GDELT georeferenced dataset and similar datasets.

2. GISQAF adds new queries to the SpatialHadoop framework. For the Query Processing (QP) part and based on the GDELT field parameters passed along with the query type, two types of evaluations will be shown in this study: one for the Apache Hadoop system and one for the modified SpatialHadoop system. Three types of queries have been implemented. The first is the spatial selection query. The second query is a circle-area query which gives all events in a specific region. Aggregation query is the third type of query we show in the results.

3. We present an approach to find co-occurring events from massive spatial datasets. We show how our framework is well-suited to process such tasks. For the Data Analytics (DA) part, we focus on generating spatial co-occurring events using GISQAF. Finding co-occurring events from big Spatial datasets is challenging but useful for Association Rule Mining [10, 11] and Spatio-Temporal Rule Mining [97] to extract patterns and rules. This will help in understanding and uncovering spatial and perceptual trends and behaviors of the social and international system. We show how we generate two-co-occurring events and use the pruning technique to generate $3, 4, ..., c$ co-occurring events where $c$ is the number of frequent itemsets required.

To the best of our knowledge, this is the first attempt to address scalability for big and complex geospatial datasets that are processed and indexed geospatially over a MapReduce framework with high performance and efficient query response time. We use the GDELT dataset as a case study. Running experiments on a multi-node cluster shows that GISQAF with SpatialHadoop achieves a much better performance than Hadoop query processing. The rest of the work is structured as follows. Section A.3 presents a background about Hadoop and SpatialHadoop. Section A.4 gives an overview of the Framework. Section A.5 presents Experimental Results. Section A.6 highlights Related Work. Finally, Section A.7 discusses the Conclusion.

## A.3 Background

Before we start explaining the GISQAF framework, we briefly introduce the first two layers this framework has been built on top of. This section introduces Hadoop [2] and Spatial-Hadoop [59] applications. Then we describe GDELT data points.

### A.3.1 Hadoop and SpatialHadoop

**Hadoop** [2] is an open-source MapReduce implementation for processing big volumes of data in a distributed manner on large clusters. Inspired by Lisp and proposed by Google [49], MapReduce is a programming model that follows a certain functional style to process large datasets. This functional style hides the details of data distribution, task scheduling, failure handling, and communication from user. Basically, the user writes map and reduce functions to achieve a particular task. Depending on the number of jobs associated with the program, jobs are submitted for execution to the master node. Assuming data has been already distributed into chunks between slave nodes using Google File System (GFS) or Hadoop Distributed File System (HDFS) in case of Hadoop, the master node divides the job into tasks, each of which is associated with particular chunks or blocks, and -by following pull scheduling strategy- assigns each task to a slave node. Slave nodes run map and reduce functions and report to the master node when done. Each map function expects key-value pairs as input and produces intermediate key-value pairs as well. The intermediate key-value pairs are partitioned and each unique key is sent to a single reducer along with its list of values emitted by multiple mappers. Reduce function, usually iterates over a list of values for a specific key, processes the computation assigned to it, and emits the final output as a key-value pair which is written back to the distributed file system.

**SpatialHadoop** [59] is an open-source Hadoop-extended geospatial framework for processing massive spatial datasets efficiently. SpatialHadoop is an extension to Hadoop and the aim of this project is to process large spatial datasets on Hadoop with the native support for spatial data. It is available at [7] as an open-source so contributors can extend its functionality and modify it to process different spatial queries and operations. Programs in this application run the same way as in Hadoop, implementing map and reduce functions with the awareness of spatial operations. The basic idea of SpatialHadoop is to index georeferenced datasets using Grid file [105], R-Tree [65], or R+-Tree [117] indexing techniques. The spatial indexing is done as a MapReduce job and stored in the HDFS. Two-level indexing is used, global and local.

In general, the global index holds information about each cell or MBR (Minimum Bounding Rectangle) that point to an individual file which contains all spatial objects (Point, Rectangle, or Polygon) in that cell. Following the trend of Hadoop, the local index is responsible for distributing files in each slave node.

SpatialHadoop constructs the index in three phases, partitioning, local indexing, and global indexing. In the partitioning phase, the input file is divided into spatial partitions of 64MB sizes such that each partition is contained in a rectangle. This is based on the type of indexing used (Grid index, R-Tree, or R+-Tree). In the local indexing phase, an in-memory local index is constructed for each partition and sent to HDFS as a 64MB block. The global indexing phase concatenates all local indexes and builds one big global index file which holds the MBRs and partition names. While the local indexes reside in the slave nodes, the global index resides in the master node.

When submitting spatial queries such as range queries, SpatialHadoop applies a pruning step before starting MapReduce jobs. The pruning or filtering step loads only partitions that intersect with the query MBR and prunes the ones that do not. This ensures that only shapes contained in the MBR are processed. This leads to a significant performance improvement as compared to Hadoop which loads all entries from all partitions. A shape might fall in more than one partition. To avoid reporting the shape twice to the query result, SpatialHadoop uses Duplicate Avoidance Technique [52]. In a recent study [58], different computational geometry problems have been implemented successfully using SpatialHadoop.

### A.3.2 GDELT Data Points

Handled by TABARI system, each event (date point) in GDELT is georeferenced by latitude and longitude locations. Events happen between two actors or parties which usually represent two countries. Figure A.1 and Table A.1 show an example of an event that took place on Sept. 02, 2013. GDELT is concerned about how to translate this textual context to a georeferenced CAMEO-coded observation. This is done through crawling international news sources and implementing spatial archiving techniques to get the 58 coded fields for each event using TABARI system and other software. In CAMEO geocoding system, each event in the the GDELT dataset has two Actors. These are the two main parties of each political observation. Each event is supposed to happen between Actor1 and Actor2. In some cases, Actor1 is the sole party of the event. As Figure A.1 and Actor Attributes Fields in Table A.1 show, Actor1 in this event is Russia and Actor2 is Syria. The news talks about a summit in Russia that discussed the conflict in Syria in which chemical weapons have been used against civilians and the possible UN international response to that. Actor1Geo_Lat, Actor1Geo_Long, Actor2Geo_Lat, and Actor2Geo_Long are the location fields that hold latitude and longitude for Russia and Syria respectively. As shown in Figure A.1 and Event

Table A.1: An Example of a GDELT Dataset Event

| FIELDS | VALUES |
|---|---|
| EVENTID AND DATE ATTRIBUTES | 266765660, 20130902, 201309, 2013, 2013.6630 |
| ACTOR ATTRIBUTES | RUS ST PETERSBURG RUS, SYR SYRIA SYR |
| EVENT ACTION ATTRIBUTES | 1, 036, 036, 03, 1, 4.0, 6, 2, 6, 0.816326530612245 |
| EVENT GEOGRAPHY | 4, Petersburg, Sankt-Peterburg, Russia, RS, RS66, 59.8944, 30.2642, -2996338, 1, Syria, SY, SY, 35, 38, SY, 1, Syria, SY, SY, 35, 38, S |
| DATA MANAGEMENT FIELDS | 20130909, http://www.newkerala.com/news/story/65045/world-should-respond-if-syria-used-chemical-weapons-ban.html |

Geography Fields in Table A.1, the latitude/longitude values of Russia and Syria are of 59.8944/30.2642 and 35.0000/38.0000 respectively.

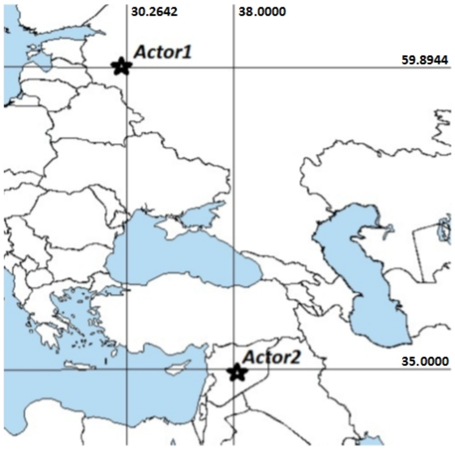


Figure A.1: A GDELT event between two actors

From the record example in Table A.1, data fields or attributes used in GDELT events represent the following:

1. EVENTID AND DATE ATTRIBUTES. The first few fields represent a unique identifier and information about the event date (day, month, and year).

2. ACTOR ATTRIBUTES. Next fields describe the two actors. An example of these attributes is Actor1Code which is represented by three alphabetic CAMEO-coded letters to describe the first party of the event. Other fields include Actor1 name, ethnic code, religion code, and others. The same fields are repeated for Actor2 as well.

3. EVENT ACTION ATTRIBUTES. These fields offer the importance or impact of the event. An example is IsRootEvent which helps to track the stream or chain of events. Another example is EventCode which describes the action that Actor1 performed against Actor2. QuadClass integer parameter specifies the primary classification of the event type as follows: 1 means Verbal Cooperation, 2 means Material Cooperation, 3 means Verbal Conflict, and 4 indicates Material Conflict. GoldsteinScale numeric coded field describes the theoretical effect of this event on the stability of a country.

4. EVENT GEOGRAPHY. These are the attributes that make GDELT georeferenced. Each event contains a longitude/latitude point. In these fields, Actor1 and Actor2 locations are represented. As mentioned earlier, some of the fields include Actor1Geo_Lat, Actor1Geo_Long, Actor2Geo_Lat, and Actor2Geo_Long which represent Actor1 latitude, Actor1 longitude, Actor2 latitude, and Actor2 longitude respectively.

5. DATA MANAGEMENT FIELDS. The last set of fields provide data management information for the event. This is useful for database management purposes. DATEADDED is an integer field that tells when the event was added to the database. Another field is the SOURCEURL field which adds the URL of the news source. If multiple news sources are used, only one is recorded. This field was added to the GDELT collection on April 1, 2013. It was not present prior to that date.

## A.4 Framework

### A.4.1 Architecture

GISQAF is an extension to SpatialHadoop. Figure A.2 shows how it has been built on top of SpatialHadoop. Layer 1 (bottom layer) of this design is the HDFS storage layer. Layer 2 (middle) consists of the Extended SpatialHadoop and Hadoop. The terms SpatialHadoop and Extended SpatialHadoop will be used interchangeably through the rest of the work. Layer 3 is responsible for the following:

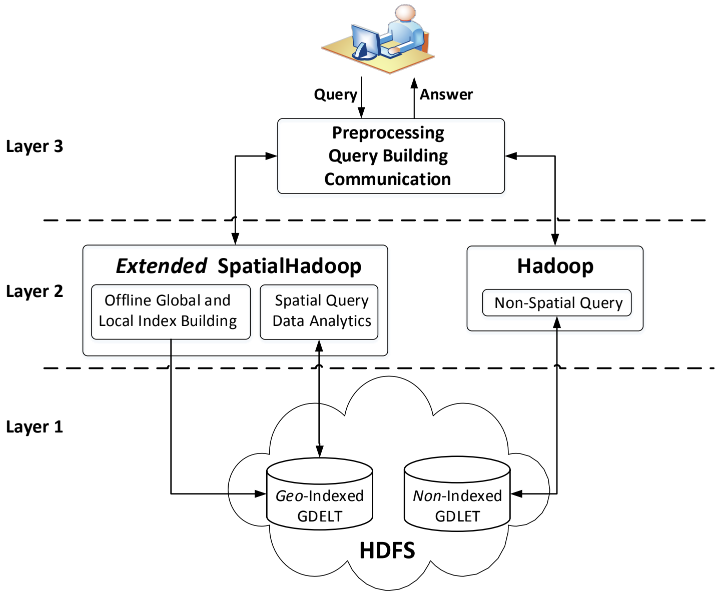


Figure A.2: GISQAF Architecture

1. **Preprocessing the datasets.** Since SpatialHadoop expects shape files as an input, each data point in the required-to-be-indexed dataset has to be converted to a form of a shape object depending on the data point geometry type (i.e., Point, Rectangle, or Polygon). In GDELT, a data point represents a longitude/latitude location so each event is represented as an EventPoint which is an extension of the Point shape.

2. **Spatial Indexing.** Layer 3 communicates with the Extended SpatialHadoop in layer 2 to index the dataset. The Extended SpatialHadoop communicates with the storage layer (layer 1) to construct the index and save it in the HDFS.

3. **Query Processing.** Here Layer 3 interacts with the client to process the queries. Depending on the query type, the Extended SpatialHadoop is called to deal with spatial queries while Hadoop is called to handle non-spatial queries and regular MapReduce jobs.

   More details are presented in the following sections.

### A.4.2 GDELT Preprocessing and Spatial Indexing

The GDELT spatial dataset has to be preprocessed and transformed into a format that SpatialHadoop can deal with. SpatialHadoop is able to index shapes (Points, Rectangles, and Polygons) and moves them to the HDFS storage layer. Following a bottom-up fashion, local indexes are constructed first followed by the global index. An event in GDELT dataset has 58 fields including the location. Figure A.3 depicts this process as follows:

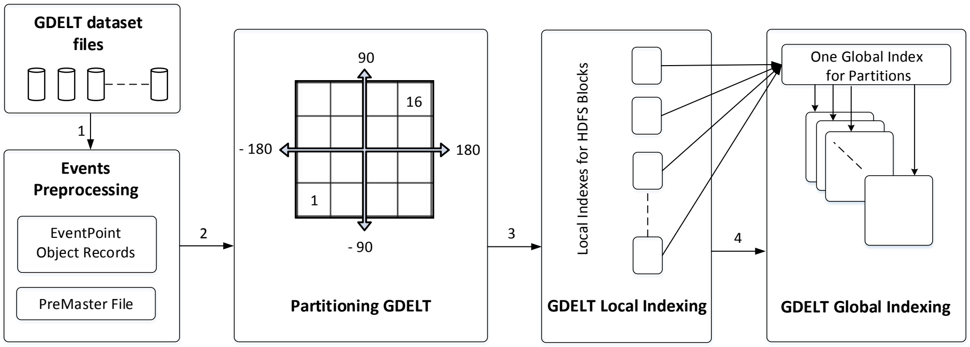


Figure A.3: GDELT Preprocessing and Spatial Indexing

1. We extract the longitude/latitude in a way that converts each event to an *EventPoint* Object that contains x and y geometry points. In the mean time, this object has to keep all the CAMEO-coded fields of the event in order to be used in the spatial queries. A PreMaster File is prepared to be used by the Extended SpatialHadoop for the partitioning phase. Each dataset in the GDELT collection is represented by a record (one line) in the PreMaster File. This record has two components: the MBR of all the data points in the file and the file name. A rectangle is represented by two points, (x1, y1) as the left lower corner and (x2, y2) as the dimensions. Thus, in the PreMaster file, the MBR's two points are (-180, -90) and (180, 90) which considers the longitude and latitude of the whole geographic map area. This step is shown at lines 2 to 9 in the pseudo code in Algorithm 3. As mentioned in Section A.3.2, in some cases, Actor1 may be the sole party of the event. This is why we extract the point of Actor1 in Algorithm 3, line 5. As shown in line 11 in Algorithm 3, the preprocessed GDELT datasets along with the PreMaster File are saved to the HDFS. This is the non-indexed GDELT shown in Figure A.2.

2. This step partitions the GDELT by dividing it into spatial partitions. This is based on the type of indexing used (Grid index, R-Tree, or R+-Tree). For clarity, and as an example, Figure A.3 shows 16 uniform partitions. Partition 1 or rectangle 1 is represented by the left lower corner (-180, -90) and dimensions (-90, -45). Partition 2 is represented by (-90, -90) and (0, -45) and partition 16 is represented by (90, 45) and (180, 90) and so on. This is represented in line 13 in Algorithm 3.

3. Local indexing is constructed for each partition, prepared in step 2, and sent to HDFS as a 64 MB block. Each local index file has meta data for some blocks and their MBRs. See line 14 in Algorithm 3.

**Algorithm 3:** Preprocessing and Spatial Indexing

```
1  begin
2      preparePreMasterFile();
3      for each GDELT dataset file do
4          for each event do
5              EventPoint ← extractActor1LongitudeLatitude(event);
6              EventPointRecord ← convertToCartesianCoordinates(EventPoint);
7              updateEvent(EventPointRecord);
8          end
9      end
10     /* End Processing all GDELT dataset files */
11     storeNonIndexGDELT();
12     /* Start Spatial Indexing */
13     partitionGDELT();
14     buildLocalIndex();
15     buildGlobalIndex();
16     storeIndexedGDELT();
17 end
```

4. All local indexes are concatenated into one big global index stored in main memory of the master node. Basically, each line in the global index file has the partition name along with the MBR or cell that contains the relative EventPoints shapes. Line 15 in Algorithm 3 depicts this step.

   Building spatial index is a bottom-up process as local indexes are built first followed by the global index. On the other hand, querying is a top-down process as global index is called first followed by the local index. We will see that in the following section.

   Finally, line 16 in Algorithm 3 shows how the indexed GDELT is stored in the HDFS.

### A.4.3 GDELT Querying

As the system now is aware of spatial data, when submitting the spatial queries to GISQAF, EventPoints partitions that do not contribute to the final answer will not be examined. As mentioned earlier, this is done in a top-down fashion by calling the global index first then the local indexes. Figure A.4 illustrates the query execution in GISQAF as follows:

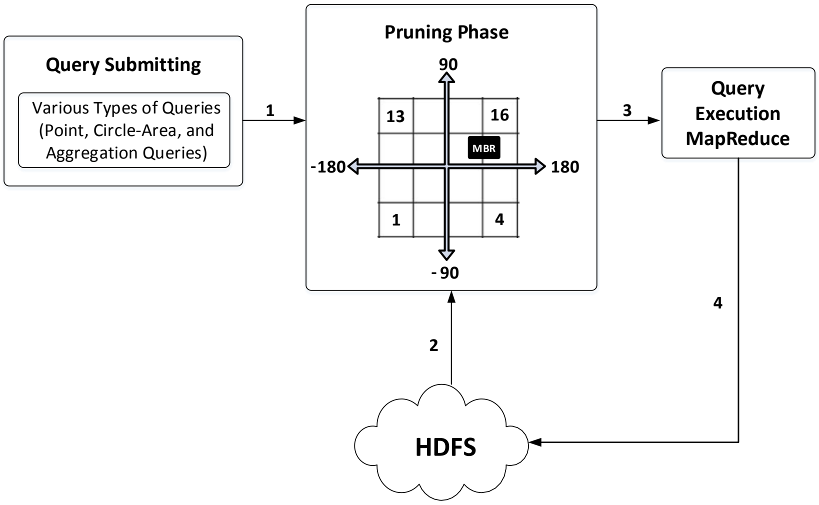

Query Submitting
Various Types of Queries (Point, Circle-Area, and Aggregation Queries)
1
Pruning Phase
90
13
16
MBR
-180
180
1
4
- 90
3
Query Execution MapReduce
4
2
HDFS

Figure A.4: Spatial-Indexed GDELT Query Execution

1. Framework passes the query along with the MBR filter. The MBR filter might be of any type of geometry shape (i.e., Point, Rectangle, Circle, or Polygon). In this step also, the framework distinguishes between spatial and non-spatial queries. In Figure A.4, we consider the case of spatial query. This step is shown in lines 2 through 6 in the pseudo code of Algorithm 4.

2. According to the MBR of the filter shape passed, step 2 gets only partitions that contribute to the query result. Global index, which is kept in the master node is used to determine the partitions that overlap with the MBR filter. This is done using SpatialFileSplitter in SpatialHadoop [59]. For clarity, we assume there are 16 partitions as Figure A.4 shows. Partitions number 11 and 12 overlap with the query MBR filter (shown as a black box). Accordingly, only these two partitions are to be picked to be processed. Algorithm 4, line 7 shows this global index pruning step.

**Algorithm 4:** Query Processing Pseudo Code

```
1  begin
2      passQueryAndMBR();
3      if spatialQuery then
4          /* Extended SpatialHadoop Data Handling */
5          getIndexedGDELT();
6          getQueryMBR();
7          BP ← findAppropriatePartitionsUsingGlobalIndex(); /* Big Partitions */
8          SB ← findSmallerBlocksUsingLocalIndex(BP); /* Smaller Blocks */
9          runMapReduceExtendedSpatialHadoop(SB); /* Algorithm 5 */
10     end
11     else
12         /* Hadoop Query Processing */
13         getNonIndexedGDELT();
14         runMapReduceHadoop();
15     end
16 end
```

3. Before executing the map function, the local index is utilized to load records using SpatialRecordReader [59]. Map function processes these EventPoints according to the query requirements. Local index use and MapReduce call are shown in lines 8 and 9 of the pseudo code of Algorithm 4. GDELT coded fields awareness has been injected to the map and reduce functions to choose the particular fields passed in the query line to get the final results.

4. The results are written back to the HDFS storage layer for client to view them.

### A.4.4 Query Types

From GDELT, extracting and analyzing events that happen in specific regions on the globe are highly demanded. This helps decision makers implement policies and uncover patterns of social evolution. For example, selection queries of protest events in specific locations can be mapped to understand global protest trends [4]. In this study, three types of queries have been implemented as follows. The first is the spatial selection point query. This query simply selects observations and events that are associated with any latitude/longitude location. The latitude/longitude location is transformed to be dealt with as an x-y geometry point. Every GDELT event has particular fields for latitude/longitude locations that show where the event has taken place. For instance, we might be interested to select all events happening at the Capital of Iraq, Baghdad. Query latitude/longitude parameters would be 33.3335/44.3521. As this is a selection query, map tasks simply emit the selected events and no reducer is needed.

The second query is a circle-area query which extracts all events in a specific region surrounding a point of interest. In this query, geometry circle class has been implemented in similar notion to SpatialHadoop range query. However, dataset schema has been considered to parse events in GDELT. Besides, the query region is passed to the framework as x, y, r where x and y are the coordinates of the center of attention and r is the radius of the coverage. For instance, the center of attention may be the Capital of South Sudan, Juba with the latitude/longitude value of 4.8455/31.5859 and all events around this point with any particular radius. Similar to query 1, no reducer is needed in this query as well.

Aggregation query is the third type of query we show in the results. In this query, we show a query of interest when dealing with GDELT data points. We explain this query with an example. The aggregate function of interest here is to count the number of events between any two actors such as China and Taiwan with the parameters, Actor1Code = "CHN" and Actor2Code = "TWN" and apply "group by" clause for the year, month, and QuadClass of the event. This type of query can be used to explore evolving situations between two parties [4]. CHN and TWN will be translated into georeferences to get the MBR (Minimum Bounding Rectangle) in order for the search to be more efficient using spatial filtering and pruning. Algorithm 5 shows the MapReduce pseudo code for the third query. The map function key is the MBR (Rectangle shape) of China and Taiwan calculated from the geographic world map. The map function value is the EventPoint shape which

**Algorithm 5:** Aggregation Count Query Psuedo Code

```
1  begin
2      Function MAP (k, v)
3          /* Input to MAP is SB, Smaller Blocks Records */
4          /* k is the queryMBR Rectangle shape, v is the EventPoint shape */
5          groupByClause ← YearMonthQuadClass; /* From EventPoint */
6          Actor1Code ← "CHN"; /* Job Configuration */
7          Actor2Code ← "TWN"; /* Job Configuration */
8          if v.isIntersected(k) then
9              if EventActor1.equals(Actor1Code) & EventActor2.equals(Actor2Code) then
10                 emit(groupByClause, 1);
11             end
12         end
13     Function REDUCE (k, v)
14         /* k is the group by clause, v is the list of counts */
15         for each value in v do
16             count += value;
17         end
18         emit(k, count);
19 end
```

is basically a GDELT data point that holds the x-y location and all the CAMEO-coded fields. See lines 3 through 4 in Algorithm 5. Notice that there is a pruning step before this MapReduce job as explained earlier in lines 7 and 8 of the pseudo code in Algorithm 4 which shows Smaller Blocks (SB) as the blocks that the map function will consume. As represented in lines 8 and 9 of Algorithm 5, each EventPoint shape from the SBs is examined to check if intersected with the MBR shape (query filter). If intersected, then actor1 and actor2 are checked to make sure they match China and Taiwan passed by query condition. If matched, then the EventPoint record count of one is emitted by the map function with the group by clause as the intermediate key and one as the intermediate value. See line 10 in Algorithm 5. In this query, the combination of Year, Month, and QuadClass forms the intermediate key. This ensures that the reducer gets a unique key with a list of ones as the value. After summing up these ones, reduce emits the group by clause as the final key and the total count as the final value. Reduce function is shown in lines 13 through 18 in Algorithm 5. Other types of queries are possible but we focused on these three types in this study as they can be extended to include other query requirements. For instance, queries can include time as well.

Comparing SpatialHadoop with Hadoop, as we will show in the experiments section, Hadoop will not work efficiently with such queries as no indexing is being followed.

### A.4.5 Finding Co-occurring Events Approach

**Problem Statement**.

To develop marketing strategies, discovering association rules between purchased items from a large collection of transactions, called *market-basket data*, was introduced in [10, 11]. As an example, Table A.2 shows a set of transactions. Each transaction has a set of purchased items. The idea is to understand customer buying habits by finding association rules that will predict occurrences of purchased items. Each rule has two parts, an *if* antecedent and a *then* consequent. For instance, from these transactions, the following rules may be generated.

$$\{BREAD, MILK\} \Rightarrow \{EGGS\}$$
$$\{PANCAKE\} \Rightarrow \{SYRUP\}$$
$$\{CEREAL\} \Rightarrow \{MILK\}$$

The first rule implicates that customers who buy bread and milk tend to buy eggs as well. Notice that the implication in the previous three rules means co-occurrance not causality.

To achieve this, *Association Rule Mining* algorithms find frequent itemsets and extract rules from these itemsets. Frequent itemsets are the k-itemsets that satisfy a predefined minimum support. From the previous table, if we take the minimum support as 40%, then {CEREAL, MILK} is one example of a 2-itemset as it has the support 3/7 which is greater than 40%. Similarly, we can find all the $< 1, 2, 3, ... >$-itemsets. From these itemsets, rules

| TID | Items |
|---|---|
| 100 | MILK, BREAD, EGGS |
| 200 | BREAD, PANCAKE, SUGAR, SYRUP |
| 300 | BREAD, CEREAL, MILK |
| 400 | MILK, BREAD, SUGAR, EGGS |
| 500 | SYRUP, MILK, CEREAL, PANCAKE |
| 600 | PANCAKE, SYRUP |
| 700 | CEREAL, MILK |

Table A.2: Market-Basket transactions

that satisfy a certain minimum confidence can be generated. The confidence of a rule $X \Rightarrow Y$ is defined as the ratio of the number of times items in Y appear in transactions that contain X to the number of times items in X occur.

For instance, from the frequent 2-itemset, {CEREAL, MILK}, the number of times cereal and milk happen together is 3 and the number of times milk happens is 5. So for the rule {CEREAL} ⇒ {MILK} to be considered, the confidence threshold should be at most 60% or 3/5.

Finding frequent itemsets from a large database is expensive. If we have 100 purchased items, the number of candidate itemsets is $2^{100}$ which is very large. Frequent itemsets generation requires efficient algorithms like Apriori [11] which prunes all parent itemsets that have infrequent children itemsets. For instance, {BREAD, MILK, EGGS} is to be evaluated only if each of the subsets (parent nodes) has a support that is greater than or equal the minimum support. If {MILK, EGGS} has a support less than the minimum support, all children nodes of {MILK, EGGS} including {BREAD, MILK, EGGS} will be pruned. This ensures to reduce the running time complexity tremendously.

In our case, instead of having purchased items, we have geo-spatial political events in the GDELT dataset. We are interested in generating rules such as {PROTEST} ⇒ {VIOLENCE AGAINST CIVILIANS}, {VIOLENCE AGAINST CIVILIANS} ⇒ {MIGRATION}, and

{TERRORIST ATTACKS} $\Rightarrow$ {MIGRATION} and see how they are associated over time around the globe. For example, the migration of Syrian refugees, due to the civil war, through the European countries Greece, Macedonia, Serbia, Hungary, Austria, and Germany can be analyzed to generate such associated rules. We can pick the month of August 2015 and the MBR of Eastern Europe to extract meaningful migration patterns. We show how GISQAF is very efficient to handle such tasks. Similar to the general rule mining technique described above, finding frequent itemsets (eventsets) is important to extract meaningful rules. We define frequent events as co-occurring events (i.e., events located next to each other that happen at a particular time window). Given a large collection of events like the GDELT dataset, the problem this work addresses is as follows. From spatial database tuples (or events) collection, find all $2, 3, 4, ..., c$ conflicting co-occurring events (frequent itemsets or eventsets) that happen in a particular geographical region (denoted by MBR, Minimum Bounding Rectangle) considering a specific time window and a particular geographical radius.

To generate co-occurring events from big datasets like GDELT, we need to use an efficient spatial-aware framework. We show how GISQAF is an efficient geospatial framework using this scenario as a case study. After finding frequent itemsets, Spatio-Temporal Rule Mining [97] can be used to generate rules to enable policy makers to develop strategies. The terms co-occurring events and frequent itemsets/eventsets will be used interchangeably in this work. The problem can be formulated as follows.

Given:

$R(t_1, t_2, t_3, ..., t_n)$;

$S(t_1, t_2, t_3, ..., t_m)$;

$\chi = \{< t_i, t_j >, < t_i, t_j, t_k >, ..., < t_i, t_j, t_k, ..., t_m >\}$;

$\delta$;

$M$;

$r$;

$c$.

Find all co-occurring tuples:

$\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, ..., < t_i, t_j, t_k, ..., t_c >\}.$

Such that:

$S \subseteq R,\ and\ hence\ m \leq n;$

$\zeta \subseteq \chi,\ and\ hence\ c \leq m;$

$\forall t,\ t \in S:\ t.LatLongPoint \in\ M\ and\ t.time \in \delta;$

$\forall < t_a, ..., t_b > \ \in\ < t_i, t_j, t_k, ..., t_c >:\ \ < t_a, ..., t_b > .radius \leq r.$

Where:

$R$: Collection of spatial database events in a time window $\delta$ ;

$S$: Subset of $R$;

$\chi$: All possible candidate itemsets in $S$;

$\delta$: Time window;

$M$: Minimum Bounding Rectangle (MBR);

$r$: Radius;

$c$: Number of frequent itemsets required;

$\zeta$: All $2, 3, 4, ..., c$-itemsets (co-occurring events);

$< t_i, t_j >$: All two co-occurring events;

$< t_i, t_j, t_k >$: All three co-occurring events;

$< t_i, t_j, t_k, ..., t_c >$: All $c$ co-occurring events;

$LatLongPoint$: Latitude and Longitude of an event.

**Example 1**.

To clarify, we present an example using the GDELT dataset. Given the year 2012 and month of December as the time window $\delta$ and the MBR $M = (36.6723, 36.5979, 38.1665, 37.1078)$ as longitude1, latitude1, longitude2, latitude2 which is located at the borders of Syria and Turkey, we are interested in finding all 2, 3, 4 co-occurring events within a 5 mile radius so $c = 4$ and $r = 0.09°$ which is equivalent to 5 miles using the Haversine Formula [104] to convert from miles to longitude/latitude distance. Applying range query produces $m = 242$ events. All candidate co-occurring events in $\chi$ are shown in Figure A.5.

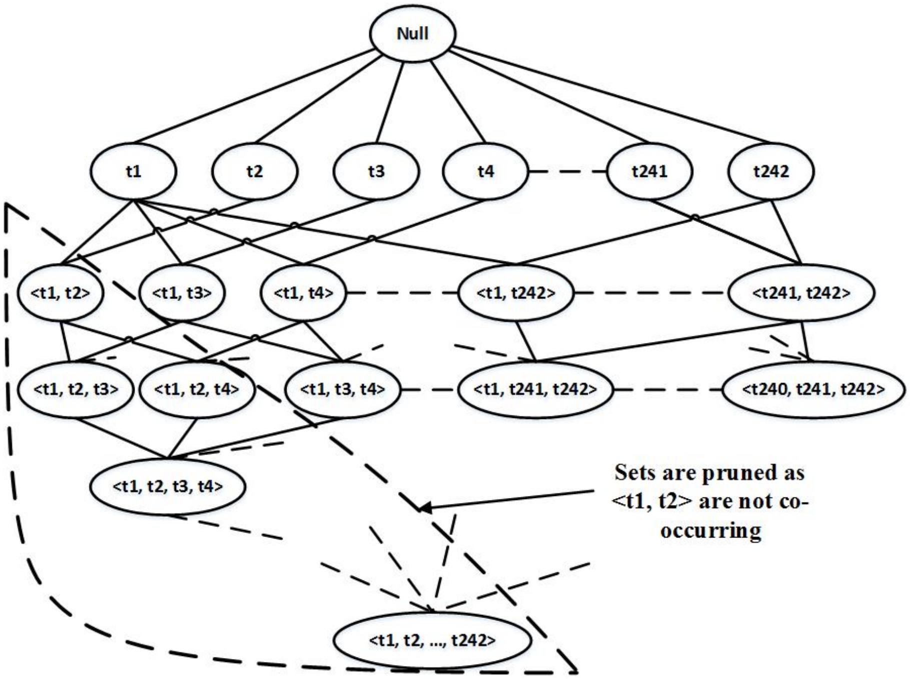


Figure A.5: All candidate co-occurring tuples (events) in $\chi$

The number of all candidate itemsets is $2^m = 2^{242}$ which gives a very large number. Figure A.5 shows how pruning excludes all super sets of $< t_1, t_2 >$ as the two events do not co-occur (not within 5 miles radius of each other). In this example we set $c = 4$ to get the subset $\zeta$. We will show how the algorithm works in the coming sections. After applying the algorithm to the GDELT dataset and getting the 242 events, we get 21796 two co-occurring, 356461 three co-occurring, and 9347136 four co-occurring events. As an example, we take one of the 21796 two co-occurring events $< t_i, t_j >=< t_{28}, t_{109} >$. The latitude/longitude values of $t_{28}$ *and* $t_{109}$ are 36.6439/37.0875 and 36.7161/37.115 respectively.

**Algorithm 6:** Pseudocode to find Spatial Co-occurring Events

```
1  Input:
2        $R$: Collection of spatial database events in a time window $\delta$ ;
3        $M$: Minimum Bounding Rectangle (MBR);
4        $r$: Radius;
5        $c$: Number of frequent itemsets required;
6  Output: All 2, 3, ..., $c$ co-occurring events.
7  begin
8      $Indexed\_R \leftarrow$ Geo-Index($R$)
9      $S \leftarrow$ Extended-Range-Query($Indexed\_R, M$)
10     for each tuple $t_i$ in $S$ do
11         $CQ = CQ +$ Circle-Query($t_i, r$)
12     end
13     $B \leftarrow$ Build-Matrix($CQ$)
14     for $d = 3 \rightarrow c$ do
15         $\zeta \leftarrow$ Find-$d$-Co-Occurring-Events($B$)
16     end
17 end
```

**An Approach to Find Spatial Co-occurring Events**.

In this section we show how we find the spatial co-occurring tuples or events. We use GISQAF to accomplish this task. Algorithm 6 and Figure A.6 explain this process as follows.

1. As Spatial Data requires spatial indexing techniques such as grid index, or R-tree, and for fast query response (Range Query and Circle Query used in the following steps), using GISQAF, we first index GDELT events that happened in the time window $\delta$ -denoted earlier as $R(t_1, t_2, t_3, ..., t_n)$- to build the geo-spatial indexes as shown in line 8 of Algorithm 6. Besides, as the problem specifies a Minimum Bounding Rectangle (MBR) region $M$, geo-indexing fastens query response by fetching HDFS blocks that contribute to the query result.

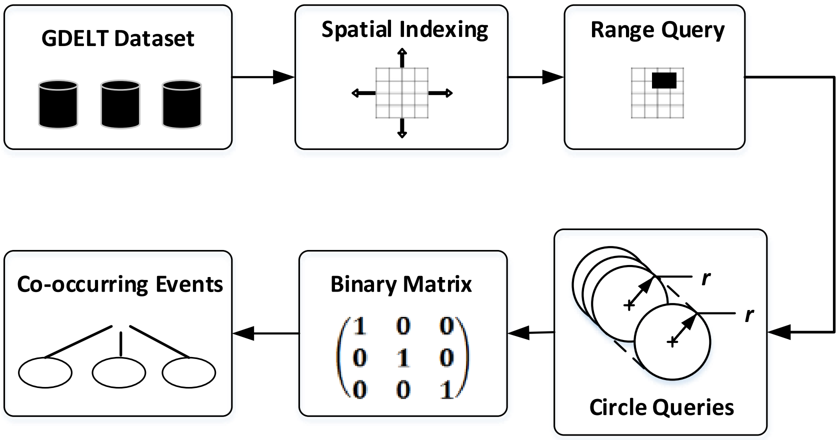


Figure A.6: Co-occurring Events Block Diagram

2. As the system is now aware of spatial data, when submitting the spatial queries to GISQAF, partitions that do not contribute to the final answer will not be examined. This step, Algorithm 6, line 9, applies Range Query (Extended SpatialHadoop Range Query to incorporate GDELT events) where framework passes the query along with the MBR $M$ filter. The result is the Database $S(t_1, t_2, t_3, ..., t_m)$ where $S \subseteq R$. Each tuple $t_i$ in $S$ contains the GDELT event ID and the event details. Notice that up to this point and as the example in Figure A.5 shows, all possible co-occurring events (candidate itemsets) $\chi = \{< t_i, t_j >, < t_i, t_j, t_k >, ..., < t_i, t_j, t_k, t_m >\}$ , can be built from $S$.

3. For each tuple $t_i \in S$, we execute a Circle Query (Extended SpatialHadoop MapReduce Query) to get all the 2 co-occurring events. The query in Algorithm 6, line 11, extracts all events in a specific region surrounding a point of interest. The query region is passed to the framework as $(x,\ y,\ r)$ where $x$ and $y$ are the coordinates of the center of attention which is $t_i$ and $r$ is the radius of the coverage. Then the Circle Query finds all two co-occurring events, $< t_i, t_j >$ where $t_j$ represents any tuple that is within the Radius $r$ from $t_i$. As the number of Circle Queries is equal to $|S|$, Extended

SpatialHadoop performs much faster than Hadoop as will be explained later in the Experiments Section.

4. After finding all two co-occurring events, line 13 of Algorithm 6 builds a Binary Matrix $B$ that contains only zeros and ones. The form of the Binary Matrix $B$ is represented as follows:

$$b_{ij} = \begin{cases} 1 & \text{if } < t_i, t_j > \quad co - occur \\ 0 & \text{otherwise} \end{cases}$$

   Notice that all values in the diagonal of $B$ are ones as each event co-occurs with itself by default (i.e., $< t_i, t_i >$ co-occur intuitively). Besides, $B$ is a symmetric matrix around the diagonal as $b_{ij} = b_{ji}$.

5. From the binary matrix $B$, to find $\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, ..., < t_i, t_j, t_k, ..., t_c >\}$ (lines 14 - 16, Algorithm 6), we define $< t_i, t_j, t_k, t_l, ..., t_d >$ co-occurring events where $3 \leq d \leq c$. For each $d$, to decide whether $< t_i, t_j, t_k, t_l, ..., t_d >$ co-occur or not, we check if each two events co-occur by examining the binary matrix $B$. For each $< t_a, t_b, ..., t_z >$ in $< t_i, t_j, t_k, t_l, ..., t_d >$, if $b_{ab} = 1, ..., b_{az} = 1, ..., b_{bz} = 1, ...,$ then all events in $< t_i, t_j, t_k, t_l, ..., t_d >$ co-occur. If any subset of $< t_i, t_j, t_k, t_l, ..., t_d >$ does not co-occur, then there is no need to continue as definitely there is no co-occurrence for the tuples in $< t_i, t_j, t_k, t_l, ..., t_d >$. Thus, as discussed earlier in Figure A.5, if tuples in a parent node do not co-occur, then this node and all descendent nodes will be pruned.

The pruning step in our algorithm is different then the one in the Apriori algorithm [11]. In Apriori, pruning of a node happens if any of its parents has a support less than a predefined threshold. Here, the node pruning takes place if events do not co-occur in any of its parents.

Lines 8 through 12 in Algorithm 6 present our framework solution to find co-occurring events. The rest of Algorithm 6 is simply implemented in a single machine as this does not require a distributed system.

**Example 2**.

In this section, we give an example for finding Spatial Co-occurring Events. Although this is not the case in practice, for the sake of clarity, we assume a small number of tuples which is $m = 6$ as the result of the Range Query with $M$ as our MBR. So we have $S(t_1, t_2, t_3, t_4, t_5, t_6)$. All events in $S$ happened in Dec 2012 which is our Time Window $\delta$. All possible co-occurring events in $S$ are 2, 3, 4, 5, 6 co-occurring events which gives us $\chi = \{< t_i, t_j >, < t_i, t_j, t_k >, < t_i, t_j, t_k, t_l >, < t_i, t_j, t_k, t_l, t_o >, < t_i, t_j, t_k, t_l, t_o, t_m >\}$. For each tuple $t_i \in S$, we execute the Circle Query to get all 2 co-occurring events and build our Binary Matrix $B$ as follows.

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Let us assume that $c = 4$ as we are interested in finding all 2, 3, 4 co-occurring events so $\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, < t_i, t_j, t_k, t_c >\}$. All 2 co-occurring events $< t_i, t_j >$ have been already obtained in the matrix $B$. To find all 3 co-occurring events $< t_i, t_j, t_k >$ where $1 \leq i, j, k \leq 6$, from $B$, we check if $< t_i, t_j >$, $< t_i, t_k >$, and $< t_j, t_k >$ co-occur by examining if $b_{ij} = b_{ik} = b_{jk} = 1$. If yes, then $t_i, t_j$, and $t_k$ co-occur and we add $< t_i, t_j, t_k >$ to the frequent eventsets.

Figure A.7 shows three co-occurring tuples. Notice that $(d_{i,j} \leq r)$ is the distance between the two geospatial events $t_i$ and $t_j$. Similarly, we find all 4-co-occurring events.

After running the algorithm for this example, the 3 co-occurring events are $< t_1, t_2, t_3 >$, $< t_1, t_2, t_5 >$, $< t_1, t_3, t_5 >$, $< t_2, t_3, t_5 >$, *and* $< t_3, t_4, t_5 >$. There is only one 4

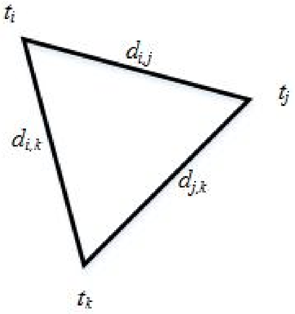


Figure A.7: Three co-occurring events

co-occurring event which is $< t_1, t_2, t_3, t_5 >$. Notice that event $t_6$ is an outlier as is does not co-occur with any other event.

The summary of this example is as follows. The number of events in the MBR $M$ is 6, the number of two co-occurring events is 21, the number of triple co-occurring events is 5, and the number of 4 co-occurring events is 1.

## A.5 Experiments

In this section we study the performance of GISQAF. We execute queries in two platforms, one using GISQAF which uses SpatialHadoop, and the other using Hadoop. We show the results when using a single-node cluster and a multi-node cluster. Both Hadoop and SpatialHadoop clusters run Apache Hadoop 1.2.1 and Java 1.6. All experiments have been implemented using internal university machines. The dataset used is the GDELT dataset explained earlier.

### A.5.1 Experimental Setup

**Single-node cluster** This single machine has a 24GB RAM, HDD of 2.2TB, and a 16 core CPU of 2.40GHz with Ubuntu 12.04 operating system.

**Multi-node cluster** The number of nodes in the cluster is nine. All machines are ho-

Table A.3: Query Types

| Query Type | # runs | Query Sample<br>$(Using \quad Extended \quad SpatialHadoop \quad Jar \quad file)$ |
|---|---|---|
| Point Query | 5 | Select events for a given Point<br>$pointquery \quad input \quad output \quad [long, lat]$ |
| Circle-Area Query | 5 | Select all events surrounding a Point with a given radius<br>$circlequery \quad input \quad output \quad [long, lat, radius]$ |
| Aggregation Query | 5 | Select events aggregate count between two Actors given a specific region<br>$countquery \quad input \quad output \quad actor1 \quad actor2 \quad [long1, lat1, long2, lat2]$ |

mogeneous. Every machine has 16GB of RAM, 256GB HDD, and a 4 core CPU of 2.2GHz. The operating system is CentOS 6.4.

### A.5.2 General Results

Table A.3 summarizes the query types, the number of runs for each query in the experiments, and the query samples. Average running time in seconds will be shown in the following tables and figures. Hadoop and the Extended SpatialHadoop emit the same query results and number of records. The terms SpatialHadoop and Extended SpatialHadoop will be used interchangeably in this section. This section shows the performance, in terms of time, of both the Extended SpatialHadoop and Hadoop using single-node and multi-node clusters. Table A.4 gives the average running time in seconds when processing the GDELT with a fixed 60GB size. In both SpatialHadoop and Hadoop, the multi-node cluster achieves much better performance than the single-node cluster in all queries. This comes as a result of the MapReduce distributed query processing. As shown in Table A.4, SpatialHadoop is much faster than Hadoop in all queries whether on a single-node machine or a multi-node cluster.

While SpatialHadoop uses spatial indexing to process only partitions overlapped with query MBR filter, Hadoop processes every record in all partitions sequentially which leads to performance degradation. The pruning function is the extra step that SpatialHadoop processes and the rest of the map and reduce functions are very similar. Indeed, this is

Table A.4: Experiments on the GDELT dataset

| | SpatialHadoop (sec) | | Hadoop (sec) | |
|---|---|---|---|---|
| | Single-node | Multi-node | Single-node | Multi-node |
| Point Query | 20.938 | **11.744** | 2459.310 | 280.226 |
| Circle-Area Query | 28.954 | **12.747** | 2471.257 | 288.716 |
| Aggregation Query | 344.660 | **84.327** | 2118.506 | 391.511 |

illustrated in more details in Table A.5 as the number of MapReduce tasks is shown for the Multi-node cluster experiments. While Hadoop has a fixed large number of map tasks as it iterates over all records in the GDELT dataset, SpatialHadoop has fewer number of map tasks as a result of pruning step. Point and circle-Area queries are faster in SpatialHadoop than aggregation query as the latter query involves implementing group by functionality to get the final counts which requires a reducer.

In Point and Circle-Area queries, few partitions (rectangles) are chosen as the query MBR is small (a point and a small radius). This results in a small number of Map tasks. This is why, as shown in Table A.4, the reduction is just 2x when increasing machines by 9x (Single-node and Multi-node). The number of Map tasks are 3 and 12, as shown in Table A.5, for Point and Circle-Area queries respectively. This is why there are 199 Map tasks in Aggregation query as the query MBR covers both China and Taiwan. The number of Map tasks is proportional to the size of the query MBR. This is illustrated in Figure A.8 where Circle-Area queries have been executed considering different radii for both SpatialHadoop single-node and multi-node clusters. Here we can see clearly how the multi-node cluster outperforms the single-node cluster when increasing the radius (i.e., Query MBR). In Point and circle-area queries, there is no reducer required as the mappers emit the selected events as the final output. This means there is no reshuffling phase in these two queries. However, the aggregation query requires a reducer which involves an expensive reshuffling phase where data will be shipped over the network from mappers to the reducer. One reducer is sufficient

Table A.5: Number of tasks for multi-node cluster experiments

| | SpatialHadoop | | Hadoop | |
|---|---|---|---|---|
| | Map | Reduce | Map | Reduce |
| Point Query | 3 | 0 | 980 | 0 |
| Circle-Area Query | 12 | 0 | 980 | 0 |
| Aggregation Query | 199 | 1 | 980 | 1 |

as the number of distinct group by clauses which are the keys to the reducer is just a few hundred. Besides, as a result of passing China and Taiwan in the aggregation query parameters, the MBR area for this query is much bigger than the areas of the other queries. We discuss scalability in the coming subsections.

### A.5.3 Point Query

In this part, we run the point query considering different sizes of the GDELT dataset to show scalability. In Figures A.9, A.10, and A.11, the x-axis is the file size in GB and the y-axis is the running time in seconds. Again, as SpatialHadoop is aware of spatial data and uses spatial indexing to prune the data, it achieves a much better performance than Hadoop as shown in Figure A.9. The results show that SpatialHadoop achieves up to 9x, 16x, 20x and 30x better performance than Hadoop using 15GB, 30GB, 45GB, and 60GB respectively. While Hadoop values increase linearly as data grow, SpatialHadoop values do not. Actually they increase but very slightly. The reason is that when pruning data, the number of selected partitions stays the same even if the size of the data grows. This lends itself to the fact that same number of cells is used when indexing. The only thing that grows is the number of records in each of the selected partitions. Hence, as data size grows, the time increases unnoticeably (i.e., fraction of seconds). This lets us conclude that SpatialHadoop scales up very well with data growth.

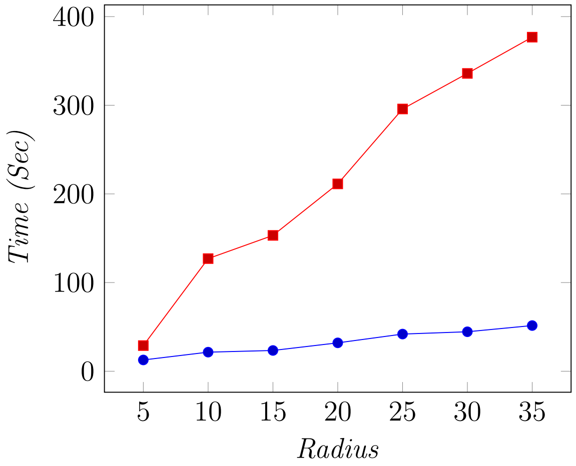


Figure A.8: Extended SpatialHadoop Circle-Area Query: ■ *Single-Node*; ● *Multi-Node*.

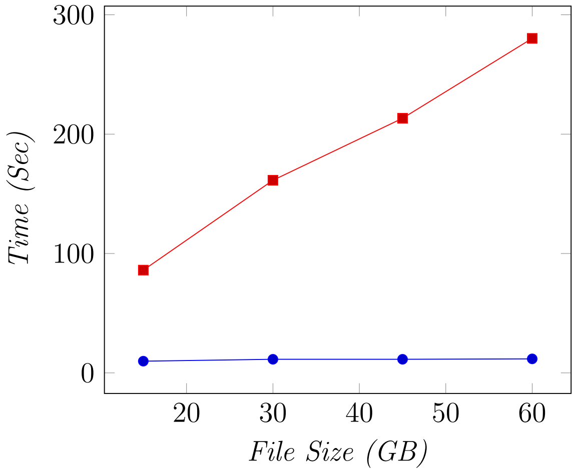


Figure A.9: Point Query: ● *Extended SpatialHadoop*; ■ *Hadoop*.

### A.5.4 Circle-Area Query

Circle-Area query is very similar to point query when it comes to discussing the results. As Figure A.10 shows, the running time is very similar to that in Figure A.9 with almost the same performance. SpatialHadoop achieves a better performance than Hadoop as well.

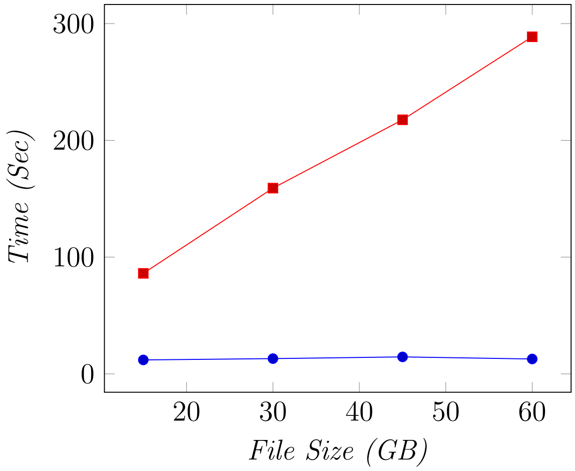


Figure A.10: Circle-Area Query: —●— *Extended SpatialHadoop*; —■— *Hadoop*.

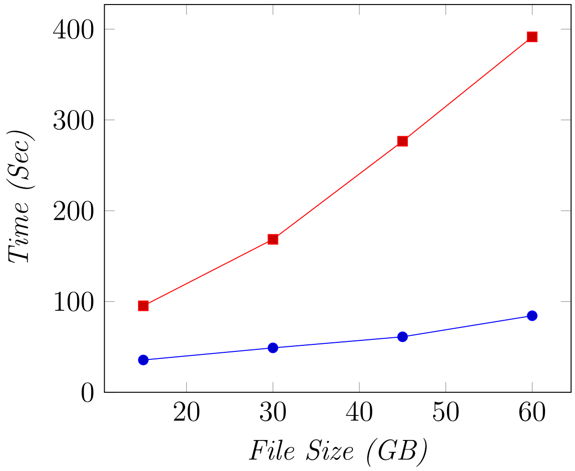


Figure A.11: Aggregation Count Query: —●— *Extended SpatialHadoop*; —■— *Hadoop*.

### A.5.5 Aggregation Query

Figure A.11 demonstrates aggregation query processing differences between SpatialHadoop and Hadoop. As mentioned earlier, SpatialHadoop implements data pruning to select related partitions based on the global and local indexes built beforehand and thus this leads to a better performance than processing all records in all partitions. Different than point and circle-area queries that are illustrated in Figures A.9 and A.10, here SpatialHadoop shows a noticeable increase in the running time. Although the number of partitions selected by

Table A.6: Results of finding $2, 3, 4$ co-occurring events

| | |
|---|---|
| Number of events in MBR $M$ | 242 |
| Number of $< t_i, t_j >$ | 21796 |
| Number of $< t_i, t_j, t_k >$ | 356461 |
| Number of $< t_i, t_j, t_k, t_l >$ | 9347136 |
| 2 co-occurring events execution time (242 Circle Queries) | (GISQAF: 48 min)<br>(Hadoop: 20 hours) |
| 3 and 4 co-occurring events time (using $B$) | 126 sec |

SpatialHadoop stays the same even when data grows, the running time increases and that can be seen clearly. The reason is that aggregation query needs more time for implementing the count and grouping results by the required fields.

### A.5.6 Co-occurring Events

In this section, we discuss the results of finding $2, 3, 4$ ($c = 4$) co-occurring events as an implementation to the example discussed in Section A.4.5. As shown in Table A.6, the number of geospatial events that are located in the MBR $M = (36.6723, 36.5979, 38.1665, 37.1078)$ is $m = 242$. The table shows the number of two co-occurring events $< t_i, t_j >$, three co-occurring events $< t_i, t_j, t_k >$, and four co-occurring events $< t_i, t_j, t_k, t_l >$. Each subset co-occurs in a radius $r = 0.09°$.

As mentioned in Section A.5.4, to find all two co-occurring events, we execute the Circle Query against each of the $m = 242$ events. Each Circle Query takes 12 seconds to execute on average as shown in Table A.4 using SpatialHadoop on the multi-node cluster. Table A.6 shows the time needed to execute 242 Circle Queries which is less than an hour. Notice that Hadoop will take around 20 hours to execute these Circle Queries. The table shows also the running time for finding 3 and 4 co-occurring events from the Binary Matrix $B$.

## A.6 Related Work

Existing non-distributed traditional RDBMS applications for querying GDELT have been used for a while [4, 78]. In [109], a geospatial DBMS approach has been used but lacks the ability for efficient spatial partitioning and boundary check. Besides, those applications experience scalability and performance issues.

A spatio-temporal indexing structure built on top of a column-family distributed solution has been suggested in [62] by Fox et al. They present a novel spatio-temporal geohashing index structure running on Apache Accumulo [1] which is a distributed key-value store based on Google's BigTable design. Apache Accumulo is built on top of Apache Hadoop, Zookeeper, and Thrift. However, the issue with this solution is that it does not support some types of queries like aggregation queries.

Some other scalable NoSQL based solutions such as GeoCouch [5] and neo4j-spatial [6] have implemented Spatial operations but they only consider limited queries with no support to data analytics.

Aji et al. presented Hadoop-GIS [8], a MapReduce spatial data warehousing system. Hadoop-GIS uses a two-level index structure, local and global. It supports various types of queries including aggregation queries. However, Hadoop-GIS focuses mainly on pathology analytical imaging medical data.

SpatialHadoop is a MapReduce framework built on top of Hadoop for processing spatial datasets efficiently [59]. Similar to Hadoop-GIS, SpatialHadoop exploits a two-level index structure, local and global. Programs in SpatialHadoop run as in Hadoop MapReduce with the awareness of spatial operations. It is open source and available for researchers to use and enhance. However, SpatialHadoop considers simple datasets and does not deal with data analytics and some complex queries like Aggregation Queries.

This work presents GISQAF (Geographic Information System Query and Analytics Framework). The framework customizes and extends SpatialHadoop to index, decode, and query the GDELT georeferenced dataset and similar datasets. GISQAF extends the work in [15] which does not consider any approach to implement spatial Data Analytics. Therefore, the work not only considers the Query Processing (QP), but it also considers the Data Analytics (DA). It addresses extracting geospatial co-occurring tuples (political spatial events) that happen in a particular geographical region (Minimum Bounding Rectangle) by considering a specific time window and a particular geographical radius.

### A.7 Conclusion

Processing massive spatial data like the Global Data of Events, Language, and Tone (GDELT) dataset becomes an issue as traditional RDBMS can no longer be used. MapReduce paradigm introduces alternatives to deal with big data. Unless MapReduce framework is well-equipped to process spatial data, it will not achieve better performances.

In this work, Geographic Information System Query and Analytics Framework (GISQAF) has been presented to process queries and implement data analytics for the GDELT dataset. The framework has been built on top of SpatialHadoop which is a Hadoop-extended framework to deal with spatial data. Experimental results using a single-node cluster and a multi-node nine-machine cluster show that GDELT query processing with GISQAF achieves up to 30x better performance than the traditional Hadoop query system.

Incorporating more query types into GISQAF is an avenue of future work. Another avenue is to make the preprocessing and spatial indexing phase of the queried dataset require less programming intervention. This will facilitate the client interaction with the framework.

In the Data Analytics part, classification and clustering can be explored as well. In addition, more optimization techniques may be examined. Some infrastructure to perform such optimizations has been already implemented in GISQAF.

As the GISQAF framework runs in an offline batch processing fasion, it does not consider online streaming preprocessing and spatial indexing. Incorporating GISQAF into a big data streaming tool is a promising future work. This will help in online scenarios as the GDELT dataset is generated in daily basis. This will require the use of incremental spatial indexing to accommodate the new generated data without the need to redo the indexing from scratch.

Apache Spark [3] is an open-source streaming distributed framework that utilizes Hadoop for its distributed file system. Unlike Hadoop, Spark utilizes in-memory cluster computing [145] which makes it up to 100 times faster than Hadoop MapReduce. Not only does implementing GISQAF over Spark help in online spatial indexing and query processing, but also with the data analytics part as Spark incorporates data mining algorithms as well. Spark has been shown to be very efficient for real-time data analytics [122, 123] where the authors use statistical and clustering techniques for online anomaly detection. Furthermore, Spark SQL has been implemented in [22] for easier querying. Incorporating GISQAF into Spark will increase the efficiency of the framework tremendously.

# REFERENCES

[1] Apache Accumulo. `https://accumulo.apache.org/`. Accessed Oct 10, 2017.

[2] Apache Hadoop. `http://hadoop.apache.org/`. Accessed Oct 10, 2017.

[3] Apache Spark. `http://spark.apache.org/`. Accessed Oct 10, 2017.

[4] GDELT Event Database. `http://gdeltproject.org/`. Accessed Oct 10, 2017.

[5] Geocouch. `https://github.com/couchbase/geocouch/`. Accessed Oct 10, 2017.

[6] Neo4j Spatial. `https://github.com/neo4j-contrib/spatial`. Accessed Oct 10, 2017.

[7] SpatialHadoop. `http://spatialhadoop.cs.umn.edu/`. Accessed Oct 10, 2017.

[8] A. Aji, F. Wang, H. V. R. L. Q. L.-X. Z. and J. Saltz (2013). Hadoop-gis: A high performance spatial data warehousing system over mapreduce. In *VLDB*.

[9] Abouzeid, A., K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin (2009, August). Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow. 2*(1), 922–933.

[10] Agrawal, R., T. Imielinski, and A. Swami (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on management of data, Washington DC, USA*, pp. 207–216.

[11] Agrawal, R. and R. Srikant (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, San Francisco, CA, USA, pp. 487–499. Morgan Kaufmann Publishers Inc.

[12] Ahmed, M., A. N. Mahmood, and J. Hu (2016). A survey of network anomaly detection techniques. *J. Network and Computer Applications 60*, 19–31.

[13] Al-Naami, K., G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan, and B. Thuraisingham (2015). P2V: Effective website fingerprinting using vector space representations. In *Proc. IEEE Sym. Computational Intelligence*, pp. 59–66.

[14] Al-Naami, K., S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham (2016). Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC '16, pp. 177–188. ACM.

[15] Al-Naami, K., S. Seker, and L. Khan (2014). Gisqf: An efficient spatial query processing system. In *2014 IEEE 7th International Conference on Cloud Computing*, Alaska, USA, pp. 681–688.

[16] Al-Naami, K. M., S. E. Seker, and L. Khan (2016). Gisqf: Mapreduce guided spatial query processing and analytics system. *Software: Practice and Experience 46*(10), 1329–1349. spe.2383.

[17] Alexa. The top visited sites on the web. `http://www.alexa.com/`. Accessed Oct 10, 2017.

[18] AlSabah, M., K. Bauer, and I. Goldberg (2012). Enhancing tor's performance using real-time traffic classification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 73–84. ACM.

[19] Anagnostakis, K. G., S. Sidiroglou, P. Akritidis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos (2010). Shadow honeypots. *Int. J. Computer and Network Security (IJCNS) 2*(9), 1–15.

[20] Araujo, F. and K. W. Hamlen (2015). Compiler-instrumented, dynamic secret-redaction of legacy processes for attacker deception. In *Proc. 24th USENIX Security Sym.*

[21] Araujo, F., K. W. Hamlen, S. Biedermann, and S. Katzenbeisser (2014). From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *Proc. 21st ACM Conf. Computer and Communications Security (CCS)*, pp. 942–953.

[22] Armbrust, M., R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia (2015). Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, New York, NY, USA, pp. 1383–1394. ACM.

[23] Arun, Saradha, V. Suresh, Murty, and C. E. Veni Madhavan (2009). Stopwords and Stylometry : A Latent Dirichlet Allocation Approach. In *NIPS Workshop on Applications for Topic Models: Text and Beyond*, Whistler, Canada.

[24] Ateniese, G., B. Hitaj, L. V. Mancini, N. V. Verde, and A. Villani (2015). No place to hide that bytes won't reveal: Sniffing location-based encrypted traffic to track a user's position. In *Network and System Security*, pp. 46–59. Springer.

[25] Axelsson, S. (1999). The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proc. 6th ACM Conf. Computer and Communications Security (CCS)*, pp. 1–7.

[26] Babu, S. (2010). Towards automatic optimization of mapreduce programs. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, New York, NY, USA, pp. 137–142. ACM.

[27] Bartos, K., M. Sofka, and V. Franc (2016). Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proc. 25th USENIX Security Sym.*, Austin, TX, pp. 807–822.

[28] Bhoraskar, R., S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall (2014). Brahmastra: Driving apps to test the security of third-party components. In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 1021–1036.

[29] Bhuyan, M. H., D. K. Bhattacharyya, and J. K. Kalita (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials 16*(1), 303–336.

[30] Blum, A. L. and P. Langley (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*(1), 245–271.

[31] Boggs, N., H. Zhao, S. Du, and S. J. Stolfo (2014). Synthetic data generation and defense in depth measurement of web applications. In *Proc. 17th Int. Sym. Recent Advances in Intrusion Detection (RAID)*, pp. 234–254.

[32] Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

[33] Cabrera, J. B., L. Lewis, and R. K. Mehra (2001). Detection and classification of intrusions and faults using sequences of system calls. *ACM SIGMOD Record 30*(4), 25–34.

[34] Cai, X., R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg (2014). A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 227–238. ACM.

[35] Cai, X., X. C. Zhang, B. Joshi, and R. Johnson (2012). Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 605–616. ACM.

[36] Canali, D., M. Cova, G. Vigna, and C. Kruegel (2011). Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proc. 20th Int. Conf. World Wide Web*, pp. 197–206.

[37] Castiglione, A., G. M. I. M. and F. Palmieri (2014, May). Modeling performances of concurrent big data applications. *Softw: Pract. Exper.. doi: 10.1002/spe.2269*.

[38] Chandola, V., A. Banerjee, and V. Kumar (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR) 41*(3), 15.

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

[40] Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber (2006). Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Berkeley, CA, USA, pp. 15–15. USENIX Association.

[41] Cieslak, D. A., N. V. Chawla, and A. Striegel (2006, May). Combating imbalance in network intrusion datasets. In *2006 IEEE International Conference on Granular Computing*, pp. 732–737.

[42] Cohen, W. W. (1995). Fast effective rule induction. In *Proc. 12th Int. Conf. Machine Learning*, pp. 115–123.

[43] Conti, M., L. V. Mancini, R. Spolaor, and N. V. Verde (2015). Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 297–304. ACM.

[44] Conti, M., L. V. Mancini, R. Spolaor, and N. V. Verde (2016). Analyzing android encrypted network traffic to identify user actions. *Information Forensics and Security, IEEE Transactions on 11*(1), 114–125.

[45] Cortes, C. and V. Vapnik (1995). Support-vector networks. *Machine Learning 20*(3), 273–297.

[46] Dai, S., A. Tongaonkar, X. Wang, A. Nucci, and D. Song (2013). Networkprofiler: Towards automatic fingerprinting of android apps. In *INFOCOM, 2013 Proceedings IEEE*, pp. 809–817. IEEE.

[47] Davi, L., A. Dmitrienko, A.-R. Sadeghi, and M. Winandy (2010). Privilege escalation attacks on android. In *Information Security*, pp. 346–360. Springer.

[48] de Carnavalet, X. d. C. and M. Mannan (2016). Killed by proxy: Analyzing client-end TLS interception software. In *Proc. Network & Distributed System Security Sym. (NDSS).*

[49] Dean, J. and S. Ghemawat (2008, January). Mapreduce: Simplified data processing on large clusters. *Commun. ACM 51*(1), 107–113.

[50] Denning, D. E. (1987). An intrusion-detection model. *IEEE Trans. Software Engineering (TSE) 13*(2), 222–232.

[51] Dingledine, R., N. Mathewson, and P. Syverson (2004). Tor: The second-generation onion router. Technical report, DTIC Document.

[52] Dittrich, J.-P. and B. Seeger (2000). Data redundancy and duplicate detection in spatial join processing. In *Data Engineering, 2000. Proceedings. 16th International Conference on*, pp. 535–546.

[53] Dougherty, J., R. Kohavi, M. Sahami, et al. (1995). Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, Volume 12, pp. 194–202.

[54] Duchi, J., E. Hazan, and Y. Singer (2011, July). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res. 12*, 2121–2159.

[55] Dudorov, D., D. Stupples, and M. Newby (2013). Probability analysis of cyber attack paths against business and commercial enterprise systems. In *Proc. IEEE European Intelligence and Security Informatics Conf. (EISIC)*, pp. 38–44.

[56] Durumeric, Z., Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson (2017). The security impact of HTTPS interception. In *Proc. Network & Distributed System Security Sym. (NDSS).*

[57] Dyer, K. P., S. E. Coull, T. Ristenpart, and T. Shrimpton (2012). Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 332–346. IEEE.

[58] Eldawy, A., Y. Li, M. F. Mokbel, and R. Janardan (2013). Cg-hadoop: Computational geometry in mapreduce. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, New York, NY, USA, pp. 294–303. ACM.

[59] Eldawy, A. and M. F. Mokbel (2013, August). A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *Proc. VLDB Endow. 6*(12), 1230–1233.

[60] Eskin, E., A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo (2002). A geometric framework for unsupervised anomaly detection. In *Applications Data Mining in Computer Security*, pp. 77–101. Springer.

[61] Forrest, S., S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff (1996). A sense of self for Unix processes. In *Proc. 17th IEEE Sym. Security & Privacy (S&P)*, pp. 120–128.

[62] Fox, A., C. Eichelberger, J. Hughes, and S. Lyon (2013). Spatio-temporal indexing in non-relational distributed databases. In *BigData Conference*, pp. 291–299. IEEE.

[63] Garcia-Teodoro, P., J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security 28*(1), 18–28.

[64] Gu, X., M. Yang, and J. Luo (2015). A novel website fingerprinting attack against multi-tab browsing behavior. In *Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on*, pp. 234–239. IEEE.

[65] Guttman, A. (1984, June). R-trees: A dynamic index structure for spatial searching. *SIGMOD Rec. 14*(2), 47–57.

[66] Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*(1), 10–18.

[67] Haque, A., L. Khan, and M. Baron (2016). SAND: Semi-supervised adaptive novel class detection and classification over data stream. In *Proc. 30th Conf. Artificial Intelligence (AAAI)*, pp. 1652–1658.

[68] Haque, A., L. Khan, M. Baron, B. Thuraisingham, and C. Aggarwal (2016, May). Efficient handling of concept drift and concept evolution over stream data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 481–492.

[69] He, H. and E. A. Garcia (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering 21*(9), 1263–1284.

[70] Herrmann, D., R. Wendolsky, and H. Federrath (2009). Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 31–42. ACM.

[71] Hintz, A. (2003). Fingerprinting websites using traffic analysis. In *Privacy Enhancing Technologies*, pp. 171–178. Springer.

[72] Hite, K. C., W. S. Ciciora, T. Alison, R. G. Beauregard, et al. (1998, June 30). System and method for delivering targeted advertisements to consumers. US Patent 5,774,170.

[73] Hofmeyr, S. A., S. Forrest, and A. Somayaji (1998). Intrusion detection using sequences of system calls. *J. Computer Security 6*(3), 151–180.

[74] Ismail, L. and L. Khan (2014, March). Implementation and performance evaluation of a scheduling algorithm for divisible load parallel applications in a cloud computing environment. *Softw: Pract. Exper.. doi: 10.1002/spe.2258*.

[75] Juarez, M., S. Afroz, G. Acar, C. Diaz, and R. Greenstadt (2014). A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 263–274. ACM.

[76] Juarez, M., M. Imani, M. Perry, C. Diaz, and M. Wright (2016). *Toward an Efficient Website Fingerprinting Defense*, pp. 27–46. Cham: Springer International Publishing.

[77] Juniper Research (2015). The future of cybercrime and security: Financial and corporate threats and mitigation.

[78] Kalev Leetaru, P. a. S. (2013). Gdelt: Global data on events, location and tone. In *Proceedings of the International Studies Association Annual Conference*, San Diego, CA.

[79] Kapravelos, A., Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna (2013). Revolver: An automated approach to the detection of evasive web-based malware. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, Washington, D.C., pp. 637–652. USENIX.

[80] Kihl, M., P. Ödling, C. Lagerstedt, and A. Aurelius (2010). Traffic analysis and characterization of internet user behavior. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, pp. 224–231. IEEE.

[81] Kim, J., P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross (2007). Immune system approaches to intrusion detection—a review. *Natural Computing 6*(4), 413–466.

[82] Kovanen, T., G. David, and T. Hämäläinen (2016). *Survey: Intrusion Detection Systems in Encrypted Traffic*, pp. 281–293. Springer.

[83] Kruegel, C., D. Mutz, W. Robertson, and F. Valeur (2003). Bayesian event classification for intrusion detection. In *Proc. 19th Annual Computer Security Applications Conf. (ACSAC)*, pp. 14–23.

[84] Krügel, C., T. Toth, and E. Kirda (2002). Service specific anomaly detection for network intrusion detection. In *Proc. 17th ACM Sym. Applied Computing (SAC)*, pp. 201–208.

[85] Lakshman, A. and P. Malik (2010, April). Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev. 44*(2), 35–40.

[86] Lazarevic, A., V. Kumar, and J. Srivastava (2005). Intrusion detection: A survey. In *Managing Cyber Threats*, pp. 19–78. Springer.

[87] Lee, K.-H., Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon (2012, January). Parallel data processing with mapreduce: A survey. *SIGMOD Rec. 40*(4), 11–20.

[88] Lee, W. and D. Xiang (2001a). Information-theoretic measures for anomaly detection. In *Proc. 22nd IEEE Sym. Security & Privacy (S&P)*, pp. 130–143.

[89] Lee, W. and D. Xiang (2001b). Information-theoretic measures for anomaly detection. In *Proc. 22nd IEEE Sym. Security & Privacy (S&P)*, pp. 130–143.

[90] Liao, C.-S., C. C.-P. and R.-S. Chang (2014, July). A novel monitoring mechanism by event trigger for hadoop system performance analysis. *Softw: Pract. Exper. doi: 10.1002/spe.2230 44*(7), 823–834.

[91] Liao, H.-J., C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung (2013). Intrusion detection system: A comprehensive review. *J. Network and Computer Applications 36*(1), 16–24.

[92] Liberatore, M. and B. N. Levine (2006). Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 255–263. ACM.

[93] LXC (2017). Linux containers. `http://linuxcontainers.org`.

[94] Manandhar, P. and Z. Aung (2014). Chapter Towards Practical Anomaly-Based Intrusion Detection by Outlier Mining on TCP Packets, pp. 164–173. Springer.

[95] Marceau, C. (2001). Characterizing the behavior of a program using multiple-length n-grams. In *Proc. New Security Paradigms Work. (NSPW)*, pp. 101–110.

[96] Masud, M., L. Khan, and B. Thuraisingham (2011). *Data Mining Tools for Malware Detection*. CRC Press.

[97] Mennis, J. and J. W. Liu (2005, January). Mining association rules in spatio-temporal data: An analysis of urban socioeconomic and land cover change. *Transactions in GIS, 9: 517. doi: 10.1111/j.1467-9671.2005.00202.x*.

[98] Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space. *ICLR Workshop*.

[99] Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed representations of words and phrases and their compositionality. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc.

[100] Mikolov, T., W.-t. Yih, and G. Zweig (2013, June). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North*

*American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Atlanta, Georgia, pp. 746–751. Association for Computational Linguistics.

[101] Miller, B., L. Huang, A. D. Joseph, and J. D. Tygar (2014). I know why you went to the clinic: Risks and realization of https traffic analysis. In *Privacy Enhancing Technologies*, pp. 143–163. Springer.

[102] Miskovic, S., G. M. Lee, Y. Liao, and M. Baldi (2015). Appprint: automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement*, pp. 57–69. Springer.

[103] Modi, C., D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan (2013). A survey of intrusion detection techniques in cloud. *J. Network and Computer Applications 36*(1), 42–57.

[104] Montavont, J. and T. Noel (2006). Ieee 802.11 handovers assisted by gps information. In *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2006. (WiMob'2006)*, Montreal, Que, pp. 166–172. IEEE.

[105] Nievergelt, J., H. Hinterberger, and K. C. Sevcik (1984, March). The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst. 9*(1), 38–71.

[106] Panchenko, A., F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel (2016). Website fingerprinting at internet scale. In *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)*. To appear.

[107] Panchenko, A., L. Niessen, A. Zinnen, and T. Engel (2011). Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pp. 103–114. ACM.

[108] Patcha, A. and J.-M. Park (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks 51*(12), 3448–3470.

[109] Patel, J., J. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, N. Hall, K. Ramasamy, R. Lueder, C. Ellmann, J. Kupsch, S. Guo, J. Larson, D. De Witt, and J. Naughton (1997, June). Building a scaleable geo-spatial dbms: Technology, implementation, and evaluation. *SIGMOD Rec. 26*(2), 336–347.

[110] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research 12*, 2825–2830.

[111] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay (2011). Scikit-learn: Machine learning in Python. *J. Machine Learning Research 12*, 2825–2830.

[112] Pennington, J., R. Socher, and C. Manning (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 1532–1543. Association for Computational Linguistics.

[113] Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pp. 61–74. MIT Press.

[114] Plonka, D. (2000). Flowscan: A network traffic flow reporting and visualization tool. In *LISA*, pp. 305–317.

[115] Raymond, J.-F. (2001). Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies*, pp. 10–29. Springer.

[116] Schrodt, P. A., mr Yilmaz, D. J. Gerner, D. Hermrick, A. Bron, A. Gregory, A. Ingram, M. Jekic, L. Mcmullen, L. Prather, and T. Price (2008). Coding sub-state actors using the cameo (conflict and mediation event observations) actor coding framework. In *in Annual Meeting of the International Studies Association.*

[117] Sellis, T., N. Roussopoulos, and C. Faloutsos (1987). The r+-tree: A dynamic index for multi-dimensional objects. pp. 507–518.

[118] Sequeira, K. and M. Zaki (2002). ADMIT: Anomaly-based data mining for intrusions. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 386–395.

[119] Shmatikov, V. and M.-H. Wang (2006). *Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses*, pp. 18–33. Berlin, Heidelberg: Springer Berlin Heidelberg.

[120] Shu, X., D. Yao, and N. Ramakrishnan (2015). Unearthing stealthy program attacks buried in extremely long execution paths. In *Proc. 22nd ACM Conf. Computer and Communications Security (CCS)*, pp. 401–413.

[121] Sinclair, C., L. Pierce, and S. Matzner (1999). An application of machine learning to network intrusion detection. In *Proc. 15th Annual Computer Security Applications Conf. (ACSAC)*, pp. 371–377.

[122] Solaimani, M., M. Iftekhar, L. Khan, and B. M. Thuraisingham (2014). Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data. In *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pp. 1086–1094.

[123] Solaimani, M., M. Iftekhar, L. Khan, B. M. Thuraisingham, and J. B. Ingram (2014). Spark-based anomaly detection over multi-source vmware performance data in real-time. In *2014 IEEE Symposium on Computational Intelligence in Cyber Security, CICS 2014, Orlando, FL, USA, December 9-12, 2014*, pp. 66–73.

[124] Sommer, R. and V. Paxson (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Proc. 31st IEEE Sym. Security & Privacy (S&P)*, pp. 305–316.

[125] Souders, S. (2007). *High Performance Web Sites: Essential Knowledge for Front-End Engineers.* O'Reilly.

[126] Sounthiraraj, D., J. Sahs, G. Greenwood, Z. Lin, and L. Khan (2014). Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In *Proceedings of the 19th Network and Distributed System Security Symposium.*

[127] Spitzner, L. (2002). *Honeypots: Tracking Hackers.* Addison-Wesley.

[128] Stöber, T., M. Frank, J. Schmitt, and I. Martinovic (2013). Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pp. 7–12. ACM.

[129] Symantec (2016). Internet security threat report, vol. 21.

[130] Tan, Y. S., T. J.-C. E. S. L. B.-S. L. J. D. S. C. H. P. X. X. and A. Narishige (2013, November). Hadoop framework: impact of data organization on performance. *Softw: Pract. Exper., 43: 12411260. doi: 10.1002/spe.1082*.

[131] Taylor, V., R. Spolaor, M. Conti, and I. Martinovic (2016, Mar). Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *1st IEEE European Symposium on Security and Privacy (Euro S&P 2016).* To appear.

[132] Thusoo, A., J. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy (2010, March). Hive - a petabyte scale data warehouse using hadoop. In *IEEE 26th International Conference on Data Engineering (ICDE)*, pp. 996–1005.

[133] Tsai, C.-F., Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications 36*(10), 11994–12000.

[134] van der Maaten, L. and G. E. Hinton (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research 9*, 2579–2605.

[135] Vasilomanolakis, E., S. Karuppayah, M. Mühlhäuser, and M. Fischer (2015). Taxonomy and survey of collaborative intrusion detection. *ACM Computing Surveys 47*(4).

[136] Vidas, T., D. Votipka, and N. Christin (2011). All your droid are belong to us: A survey of current android attacks. In *WOOT*, pp. 81–90.

[137] Wang, T., X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg (2014). Effective attacks and provable defenses for website fingerprinting. In *Proc. 23th USENIX Security Symposium (USENIX)*.

[138] Wang, T. and I. Goldberg (2013). Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pp. 201–212. ACM.

[139] Wang, T. and I. Goldberg (2015). On realistically attacking tor with website fingerprinting. Technical report, Technical Report 2015-08, CACR.

[140] Wang, T. and I. Goldberg (2017). Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, pp. 1375–1390. USENIX Association.

[141] Warrender, C., S. Forrest, and B. Pearlmutter (1999). Detecting intrusions using system calls: Alternative data models. In *Proc. 20th IEEE Sym. Security & Privacy (S&P)*, pp. 133–145.

[142] Wei, T., Y. Zhang, H. Xue, M. Zheng, C. Ren, and D. Song (2014). Sidewinder targeted attack against android in the golden age of ad libraries. *Black Hat USA 2014*.

[143] Wright, C. V., S. E. Coull, and F. Monrose (2009). Traffic morphing: An efficient defense against statistical traffic analysis. In *In Proceedings of the 16th Network and Distributed Security Symposium*, pp. 237–250. IEEE.

[144] Yuill, J., D. Denning, and F. Feer (2006). Using deception to hide things from hackers: Processes, principles, and techniques. *J. Information Warfare 5*(3), 26–40.

[145] Zaharia, M., M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, pp. 15–28. USENIX.

[146] Zhang, M., B. Xu, and D. Wang (2015). An anomaly detection model for network intrusions using one-class svm and scaling strategy. In *Int. Conf. Collaborative Computing: Networking, Applications, and Worksharing*, pp. 267–278. Springer.

[147] Zhang, Z., J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles (2001). HIDE: A hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proc. IEEE Work. Information Assurance and Security*, pp. 85–90.

# BIOGRAPHICAL SKETCH

In 2005, to quench his thirst for more knowledge, Khaled Al-Naami made a major step in his life and decided to pursue higher studies. He was lucky enough to get a Fulbright Scholarship to do his M.S. in Computer Science. In 2007, he graduated from The University of Texas at Dallas with an M.S. in Computer Science and joined the industry. With the urge to continue his education, in 2011 Khaled applied to the Ph.D. program in Computer Science at The University of Texas at Dallas and received a Teaching Assistant position. In 2012, he was nominated to get the Best Teaching Assistant Award from the Computer Science Department. Meanwhile, Khaled was part of the Big Data Analytics and Management Lab as a Research Assistant under the supervision of Professor Latifur Khan and co-supervision of Professor Kevin W. Hamlen. Not only does Khaled have research interests that span multiple areas such as Cybersecurity and Machine Learning, Author Attribution in Stream Mining, and applying Distributed Systems to improve massive datasets spatial queries, but also he enjoys working in Software Development and Applied Machine Learning areas.

**CURRICULUM VITAE**

# Khaled M. Al-Naami

## Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080, U.S.A.

Email: `khaled.al-naami@utdallas.edu`

## Educational History:

B.S., Telecommunications and Electronics Engineering, Sana'a University, 2000
M.S., Computer Science, The University of Texas at Dallas, 2007
Ph.D., Computer Science, The University of Texas at Dallas, Dec. 2017

*Enhancing Cybersecurity with Encrypted Traffic Fingerprinting*
Ph.D. Dissertation
Computer Science Department, The University of Texas at Dallas
Advisors: Prof. Latifur Khan and Prof. Kevin W. Hamlen

## Employment History:

Teaching & Research Assistant, The University of Texas at Dallas, Sept. 2011 – Dec. 2017
Software Developer, Alcatel-Lucent, Internships: May – August of 2012, 2013, and 2014
Telecom. Specialist & Programmer, Mobile Telecom. Network, October 2008 – July 2011
Software Developer, NAS, FTE: Aug. 2007 – Sep. 2008, PT: Oct. 2008 – Feb. 2010

## Professional Recognitions and Honors:

Ericsson Computer Science Fellowship, Computer Science, UTD, 2015 & 2016
Best Teaching Assistant Award, Computer Science, UTD, 2012
Fulbright Scholarship to pursue M.S. in Computer Science, 2005 – 2007
Graduated *1st in class*, Sana'a University, 2000

## Professional Memberships:

Institute of Electrical and Electronics Engineers (IEEE)