

**NETWORK TRAFFIC ANALYSIS USING  
APPROXIMATE HASH MATCHING**

BY

**ABDULLAH MOHAMMED ABDU QASEM**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In

**SECURITY AND INFORMATION ASSURANCE**

MAY 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS  
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

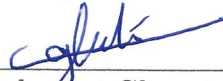
This thesis, written by **ABDULLAH MOHAMMED QASEM** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SECURITY AND INFORMATION ASSURANCE** .

Thesis Committee

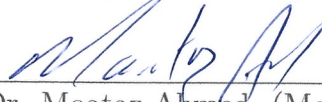


Dr. Sami M. Zhioua (Adviser)

Dr. (Co-adviser)

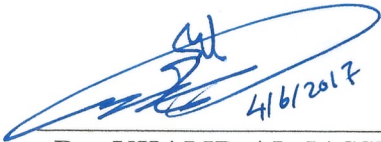


Dr. Lahouari Ghouti (Member)



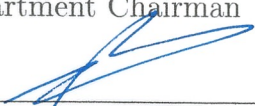
Dr. Moataz Ahmed (Member)

Dr. (Member)



4/6/2017

Dr. KHALID AL-JASSER  
Department Chairman



Dr. Salam A. Zummo  
Dean of Graduate Studies

6/6/17  
Date



## *Dedication*

I dedicate this work to my father, mother, sisters and brothers.

My father, making you proud was my biggest motivation, I would not do it without your encouragement and support. I did it for you.

My mother, thank you for your constant prayers and support, your certainly made it easier for me. I was able to do it because of you.

My sincere brothers and sisters, I would never forget the constant faith you all have in me to finish this journey; it certainly made it easier for me to reach my destination.

# ACKNOWLEDGMENTS

I would like to offer my deepest gratitude to my supervisor, Dr. Sami Zhioua for his unstinted guidance and support. Without his passion, patience and direction, this work would never have been possible.

I am also indebted to the committee members Dr. Lahouari Ghouti and Dr. Moataz Ahmed for their valuable suggestions and time.

My special thanks go to KFUPM Community for granting me this opportunity to be a student at KFUPM, which has been one of the best education experiences I have ever been through. I am also grateful to all my dear professors, teaching staff, and classmates for their help.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT</b>	<b>iv</b>
<b>LIST OF TABLES</b>	<b>viii</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>ABSTRACT (ENGLISH)</b>	<b>xi</b>
<b>ABSTRACT (ARABIC)</b>	<b>xiii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Website fingerprinting . . . . .	2
1.2 Network traffic classification . . . . .	3
1.3 Malware traffic detection . . . . .	3
1.4 Traffic analysis historical phases . . . . .	6
1.4.1 Port-based Technique . . . . .	6
1.4.2 Payload-based Techniques . . . . .	7
1.4.3 Pattern-based Techniques . . . . .	7
1.5 Problem statement . . . . .	8
1.6 Thesis contribution . . . . .	9
1.7 Thesis organization . . . . .	11

<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>12</b>
2.1 Network traffic classification . . . . .	12
2.2 Malware traffic detection . . . . .	16
2.3 Website fingerprinting . . . . .	22
2.4 Similarity hashing . . . . .	25
2.5 Gap analysis . . . . .	28
<b>CHAPTER 3 RESEARCH QUESTIONS</b>	<b>30</b>
<b>CHAPTER 4 APPROXIMATE HASHING: THE TRAFFIC ANALYSIS VERSION</b>	<b>33</b>
4.1 Approach motivation . . . . .	33
4.2 Hash-based signature generation . . . . .	37
4.2.1 Feature entropy vs feature hash . . . . .	38
4.2.2 Feature selection criteria . . . . .	40
4.2.3 Building fingerprint representations . . . . .	43
4.2.4 Comparing similarity digests . . . . .	45
4.2.5 Similarity hashing time complexity . . . . .	48
4.2.6 Parameters values . . . . .	50
<b>CHAPTER 5 WEBSITE FINGERPRINTING OVER VPN</b>	<b>51</b>
5.1 Selecting parameter values . . . . .	52
5.1.1 Cross-validation results . . . . .	53
5.2 Effect of the number of training samples . . . . .	56
5.3 Identifying a website within a sequence of web activity . . . . .	57
5.4 Effect of the size of the test data . . . . .	60
<b>CHAPTER 6 WEBSITE FINGERPRINTING OVER TOR AND ENCRYPTED WIRELESS</b>	<b>63</b>
6.1 Website fingerprinting on Tor traffic . . . . .	63
6.1.1 One-to-one cross-validation results . . . . .	64

6.1.2	Identifying a website within a sequence of web activity through Tor . . . . .	65
6.1.3	Effect of website's instances number in both accuracy and time calculation . . . . .	67
6.2	Website fingerprinting over encrypted wireless connection . . . . .	68
6.2.1	Data collection . . . . .	69
6.2.2	One-to-one cross-validation results . . . . .	69
6.2.3	Identifying a website within a sequence of web activity encrypted wireless connection . . . . .	70
6.3	Observations . . . . .	72
<b>CHAPTER 7 NETWORK MALWARE DETECTION</b>		<b>74</b>
7.1	Data collection . . . . .	76
7.2	Clustering procedure . . . . .	78
7.3	Chosen parameters . . . . .	82
7.4	Evaluation . . . . .	84
7.4.1	Compactness . . . . .	84
7.4.2	Conciseness . . . . .	86
7.4.3	Cluster centroids against single large dataset . . . . .	87
<b>CHAPTER 8 CONCLUSION</b>		<b>90</b>
8.1	Summary of contribution . . . . .	90
8.2	Threat to validity . . . . .	92
8.3	Future work . . . . .	93
<b>REFERENCES</b>		<b>95</b>
<b>VITAE</b>		<b>109</b>

# LIST OF TABLES

1.1	Traffic Classification . . . . .	8
2.1	Traffic Classification . . . . .	16
2.2	Malware Traffic Analysis . . . . .	19
2.3	Website fingerprinting Classifiers . . . . .	24
6.1	Recall (TPR) for our AHBM, Cai et al., and Tao et al. approaches on three datasets. . . . .	65
6.2	Recall (TPR) for our AHBM, Cai et al., and Tao et al. approaches on three datasets. . . . .	68



# LIST OF FIGURES

4.1	Data Object and Data Feature. . . . .	36
4.2	Probability density function of normalized Hash values for VPN web browsing dataset, Cai’s tor dataset and malware dataset. . .	41
4.3	Computing the popularity score ( $S_{pop}$ ) on a data sample of size 28 packets and using a sliding window of size 9. . . . .	42
4.4	Process of inserting a feature into a Bloom filter . . . . .	46
5.1	Accuracy in terms of feature size and window size. . . . .	53
5.2	Precision, Recall, and F1 measure values for cross-validation on VPN dataset. . . . .	55
5.3	10-fold cross-validation using different number of training samples. . . . .	56
5.4	The similarity score computation between the website’s SD and the unknown traffic’s SD. . . . .	59
5.5	Distribution of the similarity scores for each website (1) when the website is part of the testing data and (2) when the website is removed from the testing data. . . . .	60
5.6	Precision, recall, and F1 measures for a modified 10-fold cross-validation (All test samples are consolidated in a single sequence. . . . .	61
5.7	Precision, recall, and F1 measures for different test data sizes. . . . .	62
6.1	Distribution of the similarity scores for each website when the website is part of the testing data (green) and when the website is removed from the testing data (red) for (a) our dataset , (b) Cai et al. dataset, and (c) Tao et al. dataset. . . . .	66

6.2	Precision, Recall, and F1 measure values for cross-validation on wireless dataset. . . . .	70
6.3	Distribution of the similarity scores for each website (1) when the website is part of the testing data and (2) when the website is removed from the testing data. . . . .	71
6.4	Precision, recall, and F1 measures for a modified 10-fold cross-validation (All test samples are consolidated in a single sequence. . . . .	72
7.1	Accuracy in terms of feature size and window size. . . . .	82
7.2	Clusters Compactness. . . . .	85
7.3	Precision, Recall, and F1 measure values for cross-validation on Malware dataset. . . . .	86
7.4	Centroids Signature VS three single big test datasets. . . . .	88

# THESIS ABSTRACT

**NAME:**

**TITLE OF STUDY:**

**MAJOR FIELD:      DATE OF DEGREE:**

June 7, 2017

Communication networks, even small ones, generate a huge amount of network traffic every day. Furthermore, networking traffic can be captured to be either analyzed at run-time or stored to be later inspected. Captured network traffic packets include a lot of sensitive data that can be extracted. Traffic analysis is the process of extracting useful/sensitive information from observed network traffic. Typical use cases include Internet traffic classification, malware detection, and website fingerprinting attacks. Given the large size of network traffic data, the most important feature of efficient traffic analysis techniques is scalability. The Internet is expanded every minute.

Machine learning techniques have shown promising results in traffic analysis attacks, in particular, website fingerprinting. However, to be scalable, such

techniques need parallel computation. Furthermore, network traffic classification using machine learning performs poorly when the number of classes incorporated in the training increases. In addition, it needs sufficient training samples in order to produce good accuracy results. Consequently, high accuracy traffic analysis techniques use heavy machine learning algorithms (e.g. SVM) making them not applicable in large scale and real-time scenarios.

Recently, efficient tools in data fingerprinting have been developed to help digital forensics investigators to identify artifacts within hard disk images with Terabytes of data. Such techniques turned out very efficient and providing high accuracy with a small false positive rate.

Inspired by digital forensics techniques, we propose a new network traffic analysis approach based on similarity digest. The approach features several advantages compared to existing techniques, namely, fast signature generation, a compact signature representation using Bloom filters, efficient similarity detection between packet traces of arbitrary sizes, etc. Experimental results show very promising results for VPN and malware traffic, encrypted wireless traffic, but low results for Tor traffic.

## ملخص الرسالة

الاسم الكامل: عبد الله محمد عبده قاسم

عنوان الرسالة: تحليل بيانات شبكة الحاسب باستخدام التشابه الجزئي.

التخصص: أمن معلومات

تاريخ الدرجة العلمية: مايو- ٢٠١٧

حجم البيانات التي تنتجها شبكات الحاسب بما فيها الصغيرة منها في اليوم الواحد كبير جدا. هذه البيانات من اليسر جدا اعتراضها وتخزينها سواء من أجل معالجتها وتحليلها بشكل آني أو في وقت لاحق. هذه البيانات التي تنتجها الشبكات تحتوي على العديد من البيانات الحساسة والتي من الممكن استغلالها. تحليل بيانات شبكات الحاسب هو أحد المجالات التي تهتم بإدارة ومعالجة وتحليل هذه البيانات.

تحليل بيانات شبكة الحاسب له عدة تطبيقات منها هجوم بصمة المواقع الالكترونية، والتي من خلاله يهدف المخترق لتحديد هوية الموقع الذي يزوره المستخدم الضحية، تحسس وجود البرمجيات الخبيثة على شبكة الحاسب وغيرها من التطبيقات.

أحد التحديات الرئيسية التي تواجه الحلول المقدمة في مجال تحليل البيانات هو قابلية التوسع، خصوصا أن شبكة الأنترنت تتوسع يوميا بشكل كبير جدا.

خوارزميات الذكاء الاصطناعي أسهمت بشكل كبير في مجال تحليل بيانات شبكة الحاسب، لكن هذه الحلول لكي تصبح قابلة للتوسع تحتاج الى أجهزة حاسب ذات سرعات عالية جدا لتشغيلها. بالإضافة الى انها تحتاج الى عدد كبير من النماذج لتستطيع من خلالها التعلم من أجل الحصول حل شبه مثالي.

قابلية التوسع مشكلة رئيسية في مجالات عدة منها مجال التحقيق الجنائي الالكتروني. حيث أن المحققون الجنائيون يواجهون مشكلة فحص كم كبير من البيانات المخزنة في أقراص صلبة ذات سعة تخزينية كبيرة في زمن قياسي.

لحل تلك المشكلة لجأ الباحثون الى تطبيق نظرية بصمة البيانات. نتيجة لهذا التوجه تم تصميم العديد من الأدوات. تلك الأدوات أثبتت التجارب أنها كفؤه. حيث أنها تقوم بإنشاء بصمة للبيانات ومعالجتها ومقارنتها بشكل سريع. لحل مشكلة قابلية التوسع في مجال تحليل بيانات شبكة الحاسب قمت بتقديم حل يتبنى نظرية بصمة البيانات للاستفادة من خاصية التوسع التي تمتلكها. لتقييم جودة الحل المقترح قمت بمحاكاة هجوم بصمة المواقع الالكترونية على الشبكات التي تستخدم طبقة واحدة من التشفير مثل الشبكات الافتراضية الخاصة والشبكة الداخلية اللاسلكية أو الشبكات التي تستخدم عدة طبقات من التشفير كتلك المستخدمة في متصفح تور الشهير. الحل المقدم أثبت كفاءة عالية في بيئة الشبكات التي تستخدم طبقة تشفير واحدة، بينما أظهر كفاءة متواضعة على الشبكات التي تستخدم عدة طبقات من التشفير.

إضافة الى ذلك، الحل المقترح أثبت كفاءة عالية في عملية تقسيم البرمجيات الخبيثة الى مجموعات بناء على بياناتها المتناقلة في الشبكة. حيث تم خلق بصمة الكترونية لكل مجموعة بحيث يسهل عملية تحسس وجودها على شبكة الحاسب.

## CHAPTER 1

# INTRODUCTION

Network traffic analysis is the process of capturing and examining network communications in order to retrieve useful/sensitive information from the observed traffic. It has been used since the early days of digital networking. Network traffic contains a tremendous amount of information that can be extracted with small effort compared to the other methods that can be used to extract the same information. Using network traffic techniques requires only a sniffer positioned at a central point to capture either every packet passes through the central point or only once that matches specific filters. In contrary to other methods that require an agent to be installed on different places that have to be compatible with the targeted environment, that the installed agent monitors. Based on the captured traffic, Network traffic analysis has been utilized for network management, quality-of-service, law enforcement, and for security and privacy purposes. There are at least three major use cases for traffic analysis: attacks on privacy(website fingerprinting), network traffic classification, and malware traffic analysis.

## 1.1 Website fingerprinting

Traffic analysis is commonly used to attack privacy of users. Indeed, by carefully analyzing users traffic, one can reveal sensitive information about the communication (e.g. session tokens, web browsing activity, etc.). Attacks on privacy can be carried out by any entity having access to the traffic (local eavesdropper, gateway, ISP, censorship entity, etc.). When carried out legally (government), such attack is called Lawful Intercept (LI). One notable traffic analysis attack targeting user privacy is website fingerprinting. The aim of the attack is to reveal the identity of the website visited by the targeted user [1]. The attacker puts his effort trying to figure out what type of content a given user is surfing over the internet when the user is protecting himself with Privacy enhancing technologies such as single-hop solutions (e.g. Stunnel, OpenSSH, CiscoVPN, OpenVPN, etc.) and mutli-hop (e.g. Tor and JAP). Privacy enhancing technologies basically establish an encrypted tunnel/s. Therefore, all the network traffic are totally obfuscated. This scenario can be vulnerable to local traffic analysis attacks. the attacker simply should have access to a copy of the user's network traffic either by tapping the local link medium used by the victim or have access to the local gateway. Then, the attacker can compare the sniffed filtered traffic to his previously labeled collected database of network traffic, that are captured and filtered in a simulated environment. In this scenario, the attacker does not try to decrypt the encrypted traffic. Otherwise, he utilizes the statistical flaw features of the network headers' traces such as size, inter-arrival-time etc to make his prediction. Evaluation of the



Privacy enhancing technologies in [2] shows that Privacy enhancing technologies such as single-hup solutions are weak whereas multi-hup solutions such as Tor and JAP are resilient to website fingerprint attack. However, recent contribution in [3] has achieved greater than 90 % detection rate; but to run their attack demo, they used high performance parallel computing.

## 1.2 Network traffic classification

Another area of network traffic analysis in the literature focuses on traffic classification<sup>1</sup> [4] which aims at associating traffic flows with the applications that generated them (e.g. web browsing, VoIP, P2P, etc.). Recognizing accurately which type of traffic is flowing through a network is very valuable for quality of service (QoS) purposes. It allows network service providers to optimize the download/upload speed depending on the type of the traffic. Other benefits of traffic classification include a better understanding of the network utilization, dealing with congestion problems, traffic accountability, etc.

## 1.3 Malware traffic detection

The third very important area of network traffic analysis is detecting the existence of the malicious applications via network traffic. In particular, traffic analysis is commonly used to detect malware and botnet infections [5]. Relying on network traffic analysis to detect malware activity is very attractive since it allows to cover

---

<sup>1</sup>Also known as traffic identification

a large number of hosts without requiring any of these hosts to install any software.

Malware becomes very sophisticated. By using obfuscation techniques in addition to borrowing rootkit services, malware can easily bypass the anti-malware. Fortunately, even if the malware is able to hide its activities within the system; it won't be able to hide their existence within the network. Malware that does not communicate with their source; their damage is maintainable. Organizations really terrified from malware that can successfully steal classified documents and send them to their source.

During the war between malware authors and malware analysts, several methodologies/techniques have been developed to detect the existence of the malware within the network traffic. Almost all methods used to classify application's network traffic have been used to identify the existence of malware within the network traffic. Therefore, malware traffic detection suffers from the same limitations faced by the network classification methods. Sophisticated malware, to bypass port-based filtering, starts using dynamically allocated ports or hide themselves via using popular ports such as HTTP port to bypass firewalls. Furthermore, they use a secure protocol to decrypt their packet traffic. In addition, between now and then, a new malware appears. Consequently, network traffic classifier will not be able to recognize them. Another method used to detect malware traffic is anomaly based detection technique that tries to learn traffic behavior in order to differentiate between normal and abnormal network traffic. However, IDS suffers from high false positive rate.

Today, with the increased use of automated obfuscation tools, that are used to generate multiple variants of malware, the number of the generated malware is increased exponentially. Obfuscation tools purpose is to change syntax of malware via inserting nulls, packing, code encryption etc. Consequently, resulted malware will have different internal structure but has similar semantic behavior. Attackers resort to these measure to avoid signature-based anti-malware as well as static malware analysis. Indeed anti-malware fail to maintain an up-to-date database of malware signatures. as a result, malware scanners suffers from high false positive rate[6]. Furthermore, Moser et al. [7] had designed an obfuscation problem, that is proved to be NP-hard to be detected by classical anti-malware signature as well as by static analysis tools. This escalation from the malware authors triggers researchers to develop behavioral countermeasure since it becomes impractical to generate a signature for each malware. Behavioral malware clustering has two approaches: monitor malware while being executed via observing CPU/memory usage and system call, registry operations, setting a registry key and resource allocation etc. to either create a signature from or to learn a model[8, 9, 10]. The second approach monitor malware networking behavior such as protocol used, packets' features or flaw features etc. Behavioral approach utilized the idea that variant obfuscated malware shares the same semantic behavior. Basically, behavioral system will cluster malware that shares similar activities. Then, it decides common features for each group to be used as their group signature . Consequently, behavioral defense system would be able to catch almost all the malware

variants. According to [11, 12] network behavioral system more attractive than system level competitive one. First of all, it is easy to implement and deploy. Second, it does not require to use virtualized environment as well as costly dynamic analysis.

## 1.4 Traffic analysis historical phases

Traffic analysis techniques went through three major historical phases.

### 1.4.1 Port-based Technique

The first phase consists of using TCP/UDP port numbers to recognize identify the type of the traffic or to define signatures. This simple approach suffers from at least three limitations:

- Many applications increasingly avoid using non-registered IANA ports (unpredictable ports)
- To avoid tagging, many applications use port numbers assigned to other applications.
- With IPv4 address exhaustion, an increasing number of web servers are offering various services through port address translation (PAT) sharing the same public IP address but on different ports.

With these limitations, port-based traffic analysis is still very commonly used due to its simplicity and efficiency.

## 1.4.2 Payload-based Techniques

To address port-based traffic analysis limitations, payload-based techniques emerged<sup>2</sup>.

As its name indicates, it classifies the traffic by inspecting packets payloads looking for application-specific data. In turn, payload-based techniques suffer from the following limitations:

- Inspecting third-party packet payloads involves a significant privacy legal issues. Typically, privacy laws prevents access to third-party packet payloads.
- Applications are increasingly using new and customized protocol and header formats while packet inspection assumes the knowledge of payload syntax.
- Inspecting all packets payloads is computationally expensive.
- Applications are increasingly encrypting/obfuscating their traffic.

## 1.4.3 Pattern-based Techniques

The latest phase of traffic analysis evolution is pattern-based techniques which do not require deep payload inspection [13]. Instead, these techniques rely on statistical patterns in externally observable attributes of the traffic such as size, direction, time delay, etc. Pattern-based techniques overcome all shortcomings of payload-based techniques but are slightly less accurate. A key advantage is that pattern-based techniques can still be usable if the traffic is encrypted/obfuscated.

---

<sup>2</sup>Also known as Deep Packet Inspection (DPI).

Table 1.1: Traffic Classification

Paper	Classifier	Features	Application and Accuracy
Este et al. [14](MTCLASS)	SVM	Sizes of the first 4 packets of each flow	Focused on scalability
Finamore et al. [15, 16](Tstat and KISS)	Euclidean distance and SVM	First n bytes of each of the first C UDP packets of each flow	True positive between 84% and 99%
Donato et al. [17](TIE)	Naive Bayes, Majority Voting, Priority Based, etc.	Per-session features: number of packets, total bytes in each direction, IPT, etc.	Between 50% and 99%

## 1.5 Problem statement

A large body of works in the literature focused on applying machine learning techniques to network traffic in order to extract patterns from the traffic traces. These patterns can then be used to recognize similar traces. On the downside, existing pattern-based techniques suffer from the following limitations:

- They are good at checking if a sequence of packets matches a signature but they are not suitable for telling if a large traffic trace contains the trace of a specific item (e.g. website or malware).
- Being based on machine learning algorithms, they are not efficient enough for real-time checking.

For the first limitation, consider the following common scenario. An entity through which a huge amount of traffic is flowing (e.g. ISP) is trying to pinpoint a specific artifact (e.g. website, malware, etc.) in the traffic and in real-time. Existing pattern-based traffic analysis techniques are not suitable for this task because they work by matching each part of the traffic with each signature. The

signature matching is based on comparing "apples to apples". That is, extracting each time a limited sequence of packets and matching them with the signature. For example, if a signature is generated from  $n$  packets, at each step, only around  $n$  packets should be extracted from the traffic and matched with the signature. In presence of a large number of artifacts (A bank of malware, a bank of websites, etc.) with their signatures, keeping up with the real-time intake becomes quickly infeasible.

For the second limitation, machine learning techniques, in particular SVM based classifiers which produced the best results in the literature [18, 3], are relatively slow. For example, Tao and Wang [3] resorted to using SHARCNET, a Canadian academic consortium that offers high performance parallel computing. Additionally, as has been noted previously [1], if the samples used for signature generation are not fresh, the traffic analysis accuracy will decrease significantly. Hence, to keep high accuracy levels, signatures have to be updated frequently. This requires efficient signature generation in particular when the number of classes is large (Websites, Malware, etc.).

## 1.6 Thesis contribution

This work can be seen as another contribution to pattern-based traffic analysis but that addresses the above limitations. Unlike the majority of existing works in the literature which apply machine learning techniques, we apply techniques from digital forensics. The intuition behind our approach is that some algorithms from

digital forensics try to solve problems very similar to traffic analysis problems. A common scenario in data forensics is to check if a large search space (e.g. hard disk) contains traces of source data (e.g. a picture file). This scenario has natural resemblance with the problem of checking if a large amount of network traffic contains traces of a specific artifact (e.g. website, malware etc).

Inspired by digital forensics technique proposed by [19, 20], We propose a new network traffic analysis approach based on similarity digest (AHBM). The approach features several advantages compared to existing techniques in network traffic analysis, namely, fast signature generation, compact signature representation using Bloom filters, efficient similarity detection between objects of arbitrary sizes, etc. To achieve my goal, We have customized approximate hash based matching (AHBM), a recent and more flexible approach for similarity identification which can accommodate various modifications (insertion, deletion, reordering, etc.) in the source data. Furthermore, AHBM tries to identify statistically-improbable chunks of bytes and use them to compute the similarity between two sources of data. Unlike simple hash based techniques, which support yes/no results, AHBM allows requests to be answered approximately, that is, with a value between 0 and 100.

With customized similarity hashing (AHBM), We have addresses the above two limitations of pattern-based approaches. First, signature generation is much simpler, and consequently much faster, with AHBM, Second, hash based techniques are efficient in presence of a large search space (typical scenario in digital



forensics). The downside of using AHBM, however, is a slight decrease in the traffic analysis accuracy. We show that by carefully tuning the parameters of the AHBM algorithm, we could significantly improve the traffic analysis accuracy. In the empirical analysis, We use four datasets representing the three major use cases of traffic analysis, namely, Web browsing through VPN dataset, Web browsing through Tor dataset, Web browsing through secure wireless network(WiFi), and Malware traffic dataset. Experimental results show very promising results for VPN, WiFi, and malware traffic, but low results for Tor traffic.

## **1.7 Thesis organization**

The remaining of the document is organized as follows: Chapter 2, we go through the literature review. In Chapter 3, we demonstrate how the similarity hashing could be tailored in order to be used in network traffic analysis. In Chapter 4, we evaluate our proposed approach via emulating website fingerprinting over secure VPN. In addition, we show and evaluate the scalability of our proposed approach. In Chapter 5, we further evaluate our approach over Tor network as well as secure wireless network(WiFi) using website fingerprinting attack. In Chapter six, we show how our proposed approach can be utilized to build behavioral malware traffic detection system. In Chapter 7, we summarize our thesis work as well as threat to avidity and future work.

## CHAPTER 2

# LITERATURE REVIEW

### 2.1 Network traffic classification

The first major use case of network traffic analysis is to classify Internet traffic. Being able to identify the type of traffic flowing through the network allows service providers to prioritize, protect, or prevent certain types of traffic. Dainotti et al. [4] provide a state of the art of traffic classification and point to key issues and future directions in the field. In particular, they mention the difficulty to collect large and reliable data for traffic classification benchmarking. The obstacles to collect and share such data sets are mainly the privacy and sensitivity of the traces. Without such sharable benchmarks it is difficult to validate and compare the efficiency of proposed algorithms.

The simplest and historically the most commonly used technique for traffic classification is port-based [21]. Classification is based on the source and/or destination port numbers. This technique assumes that the applications are using

well-known TCP and UDP ports consistently [22]. As mentioned in the previous section, this approach has several limitations [23], in particular, some applications do not have registered port numbers (e.g. P2P services), some applications are using other applications port numbers, and some web service providers are using port address translation (PAT) to share the same public IP address. Port-based approach is by far the fastest traffic classification technique and it is still commonly used in practice (when accuracy is not important).

High accuracy traffic classification can be reached with payload-based techniques (DPI) which consist in inspecting packet payloads in order to identify the type of the traffic. Several open source (L7-Filter [24], nDPI [25], ntopng [26], libprotoident [27]) and commercial (CloudShield [28], NetFlow [29], NBAR [30]) payload-based traffic analysis tools are available in the market. Each one of these tools come with a database of signatures that capture the pattern of each type of traffic. Traffic classification is achieved by matching the payload of each observed packet with the database signatures. Karagiannis et al. [21] were the first to generate packet payload signatures (P2P applications). However, the signature generation is based on protocol reverse engineering and is mainly manual. Most of subsequent contributions focused on automating the signature generation process [31, 32, 33, 34, 35]. As mentioned previously, payload-based techniques suffer from four major drawbacks, namely, efficiency (inspecting all packets payloads is computationally expensive), privacy (access to third party packet payloads is illegal), new protocols syntax (new protocols with different syntax are quickly emerg-

ing), and encryption (the approach is ineffective with encryption/obfuscation).

Pattern-based traffic analysis techniques overlook port numbers as well as payload content and focus on statistical and externally observable properties of the traffic that can be extracted by examining only TCP/IP headers. Since Frank [36] showed how machine learning can be used in intrusion detection, machine learning became a major approach for pattern-based traffic analysis. Nguyen et al.[13] surveyed the state-of-the-art approaches proposed in network traffic classification using machine learning techniques. It mentioned the applications of machine learning classification in Quality of service, network management, law enforcement and security. Furthermore, it outlines the evaluation metrics used to evaluate proposed network classification models. what is needed to build network classification module using machine learning? What features have been used so far in state-of the-art of network traffic classification? The surveyed papers are categorized according to the machine learning strategy used either supervised or unsupervised as well as how they contribute to these field. It also addresses the challenge of building a real-time classifier model. Furthermore, it mentioned the area of integrating the supervised and unsupervised learning techniques in order to automate the network classification process. The features used in the surveyed works are packet length statistics, byte counts, connection duration, Fourier transform of packet, inter-arrival time, size of TCP/IP control fields, total number of pushed packets, total number of ACK packets, average inter packet gap etc. Zhang et al.[37] proposed a non-parametric approach to network traffic

classification. They incorporated the flow correlation in to supervised learning classification. BoF- bag of flows - is used to model the flow correlation. They conducted the correlation analysis on three-tuple heuristic dst-ip; dst-port; Protocol. They evaluated their work in two datasets. They show that supervised classification incorporated with flow correlation outperform NN, SVM and Neural nets classifiers. They further show that classification with correlation can perform constantly even when trained with small training data samples. Zhang et al.[38] proposed a new scheme to tackle zero-day application classification. The new scheme basically utilizes both supervised classification and clustering techniques. Therefore, it has the capability to discriminate among pre-known applications and zero-day applications. Their proposed scheme consists of three modules: unknown discovery utilizing clustering, and BoF based classification using flow correlations and system update. Furthermore, they propose a new method that intelligently optimizes the proposed scheme parameters. They test their proposed scheme against six different datasets; each dataset has been collected at different time, different location using different network media types. Finally, their evaluation showed that their proposed scheme outperforms the four state-of-the-art methods: one-class SVM, semi-supervised clustering, correlation-based classification, and random forest. Currently, machine learning based classifier implementations include Tstat [15], TIE [17], MTClass [39], and CoMo [14] (Table 2.1).

Table 2.1: Traffic Classification

Paper	Classifier	Features	Application and Accuracy
Este et al. [14](MTCLASS)	SVM	Sizes of the first 4 packets of each flow	Focused on scalability
Finamore et al. [15, 16](Tstat and KISS)	Euclidian distance and SVM	First n bytes of each of the first C UDP packets of each flow	True positive between 84% and 99%
Donato et al. [17](TIE)	Naive Bayes, Majority Voting, Priority Based, etc.	Per-session features: number of packets, total bytes in each direction, IPT, etc.	Between 50% and 99%

## 2.2 Malware traffic detection

Pattern-based techniques have been extensively used for network malware detection [5]. The work of Livadas et al. [40] is a typical example of pattern-based malware detection where models are learned from a testbed environment with virtual machines. The classification is based on relatively simple set of features, namely, duration of the flow, average bytes per packet per flow, average bits per second per flow, variance of packet inter-arrival time for flow, etc.

Boukhtout et al.[41] propose malware traffic detection using machine learning. They have employ five different classification techniques -J48, Boosted J48, Naive Bayesian (NB), Boosted NB, and SVM. They use IP level features proposed by [42, 43], that have been proposed to classify VoIP traffic over an encrypted channel. The five created classifier has been trained and tested using two datasets. The first dataset consists of malware traffic, that has been collected in a controlled environment using GFI sandbox. For one year they have executed 1.5 million different malware. They end up with 100000 pcap files labeled with the name of malware's hash. the second dataset used to represent benign traffic; they basically

use the benign part of DARPA dataset. Their evaluation showed that J48 and Boosted J48 achieved the superior malware detection rate approached 99% with less than 1 % false positive. However, when they tried with known malware samples (Zeus, etc.) with a smaller dataset, they obtained lower rates.

Nelms et al.[44] propose a new approach named ExecScent to detect botnets that use http protocol to communicate their C&C. They state that their solution learns from the known C&C botnet communication to identify the existence of new http C&C botnet as long as the newly discovered botnet has shared communication behavior of the previously known botnet. The approach basically starts by learning itself from previously known botnet samples. It automatically generates a signature template for known malware sample via locating the initial communication of the botnet sample. Therefore, the generated template can be employed at the edge of the network to detect exact or similar botnets. They have evaluated their proposed solution on three large networks. Consequently, they were able to detect many new botnets in addition to discovering hundreds of newly infected machine. Their result outperforms up to date commercial C&C domain blacklist. However, the approach is quite classic because it relies on the content of the packets.

BotMiner [45] is a general framework able to detect botnets regardless of their type, their structure, or their communication protocol. BotMiner's main objective is to detect a group of infected machines with a botnet within the target network. It utilizes the fact that bots usually behave in a similar manner due to the deterministic

nature of the programs behavior. It functions as the following: it clusters similar connections together at one side, and similar malicious activities on the other side, then it performs cross cluster correlation trying to find hosts that share the two clusters. Such finding is an indication of an existence of members of a botnet.

BotSnifer[46] targets centralized botnet structures that use either IRC or HTTP protocols. BotSnifer is anomaly network detection system employ correlation and similarity and statistics algorithms to detect activities and responses patterns issued by the infected bots, taking the advantage of the preprogrammed nature of the software installed on the infected machines.

The more recent BotFinder system [47] used five different flow-based features, namely, average time interval between flows, average duration of flows, average number of bytes in each direction (source and destination), and fourier transformation over flow start times. However, unlike other malware detection systems, BotFinder uses a computed score to find matchings in test traffic.

Rosow and Dietrich [48], through ProVex system, deal with encrypted malware communications differently: packets are decrypted using brute-force. Then, botnet signatures are expressed in terms of probabilities to have specific bytes at specific positions inside the packets (e.g. byte C4 occurs at position 47 with probability 0.8). The approach has two drawbacks: it works only for botnets using simple encryption (e.g. XOR, etc.) and it requires plain text (not encrypted) training samples to learn the botnet signatures. To overcome this latter drawback, Gu et al. [49, 45, 46] intersect two types of data inputs: network traffic



Table 2.2: Malware Traffic Analysis

Paper	Classifier	Features	Application and Accuracy
Perdisci et al.[50]	3-steps clustering using distance measure between http requests	http requests	between 20% and 85%
Livadas et al. [40]	Naive Bayes	TCP Stream properties (total bytes, total packets, ports, etc.)	False Negative Rate (FNR) 8%
Tegeler et al. [47](BotFinder)	Clustering using CLUES [51]	characteristics of network trace (sequence of tcp streams between two IPs): time interval between streams, duration, total bytes in each direction	Up to 90%
Roscow and Dietrich [48](ProVeX)	Probabilistic signature matching	Specific encrypted bytes	between 80% and 100%
Gu et al. [45](BotMiner)	2-steps clustering using X-mean [52]	TCP/UDP flow characteristics (number of packets, average number of bytes per packet, average number of bytes per second)	between 75% and 100%

and host activity. The two data inputs are intersected to compute a score of the likelihood for a host to be infected. The similarity between flows is computed using several traffic features: number of packets per flow (ppf), average number of bytes per packet (bpp), average number of bytes per second (bps), etc. The main problem of this approach is that the clustering is too coarse-grained: the traffic volume is huge and the obtained flows are too long even with extensive filtering and white-listing. In addition, the approach depends on having several hosts infected in the same network so that similarities can be detected. Notable traffic analysis techniques for malware detection are summarized in Table 2.2.

Another direction of malware traffic analysis is interesting in clustering malware sharing similar behavior into groups in order to generate a general signature

for these group of malware. Researchers put their intentions on these area when they clearly found that classical signature based anti-malware detection as well as static analysis are not enough in capturing polymorphic and obfuscated malware. Moser et al. [7] had design a 3SAT obfuscation problem to demonstrate the difficulty of the obfuscation problem in the area of static analysis and signature based anti-malware. Konrad et al.[9] use machine learning to train a model over the behavioral features extracted from reports generated by sandbox environment. Then, based on the weights generated by the learned model, they demonstrate the most important features used the learned model. The main issue with their contribution is with their reliance on inaccurate anti-malware labels.

Network-based approaches for malware detection have the advantage of covering a large number of hosts without requiring these hosts to install any software. This makes deployment easier and incurs no performance penalty for end users. one prominent work in this area presented by Perdisci et al.[50]. They propose and implement a behavioral clustering system to group unlabeled similar that uses HTTP protocol as a way of communication and similar networking activities. Their proposed system looks into a pool of HTTP malware's traffic samples and reveal similarities among malware samples. Then it prepare extracted features in a suitable format accepted by Token-Subsequences algorithm[53] to generate signature in snort format. To achieve their goal, they employ multi-clustering refinement. At the beginning, they utilize statistical properties of HTTP features such HTTP requests' numbers, GET request Numbers, Post request numbers etc.

Then fine-grained clustering used over structural resemblance among consecutive HTTP request (Get,Post,Header etc). After that, the centroid of the resulted clusters are represented as a set of the network signatures. Each summarizing its related HTTP traffic generated by the malware in the cluster. The main issue with Perdisci contribution is that, it only work over HTTP traffic while malware now a day uses diverse communication protocols. Furthermore, it will be come useless when HTTP traffic goes over an encrypted tunnels. To remedy the first limitation, zubair et al. [54] propose a tools named FIRMA which able to extract malwrar traffic signature from a pool of malware traffic samples that use diverse communication protocols such as HTTP, IRC, SMTP, TCP,UDP). Firstly, it cluster unlabelled malware that share similar behavior into families. Then it generate a general signature for each family. Since it is hard to create a general network traffic signature for all malware. Doing so will end up with high false positive rate. Finally, their generated signature are printed in two well known IDS format-Snort and Suricata to be used in real time system. Even though, they show that their proposed tool are 4.5 more faster than the implemented approach proposed by Perdisci[50]. Then perdisci etl al [55] improve their previous work [50] to make their clustering system scalable, they have replaced the hierarchical clustering approach, that they have used previously, with popular incremental clustering algorithm (BIRCH ) which is more efficient with big dataset. This new implementation has reduced the processing time from several hours into few minutes. Even though proposed network behavioral clustering system show very

high accuracy in grouping unlabeled families into groups, they still useless when network payloads are encrypted because both proposed approaches relies on features extracted from used protocol structure. therefore, they suffer from the same limitation faced by payload based detection mentioned earlier.

## 2.3 Website fingerprinting

Users of anonymity systems (e.g. Tor [56]) are increasingly the target of traffic analysis attacks which threatens their privacy. In particular, website fingerprinting is a variant of pattern-based traffic analysis aiming at revealing the identities of websites accessed by users.

Herrmann et al.[2]applied the test data mining techniques to fingerprint the web sites. This work target the following privacy enhancing technologies: single-hup such as Stunnel, OpenSSH, CiscoVPN, OpenVPN and mutli-hup such as Tor and JAP. They have used the normalized distribution of the IP packet size and direction as features. The have employs the multinomial naive-bayes classifier on the extracted features. The dataset used for training and testing has been generated by the authors in a controlled environment. This contribution shows a remarkable achievement on a single-hup. However, it shows very poor results on multi-hup techniques.

Panchenko et al.[57] employ the support vector machine to show a possible websites fingerprint attack in both Tor and JAP anonymity solutions. They play with the following features: volume, time and direction of the IP packet of each

sample. They start by test each feature individually. Then the combine each features altogether. Consequently, they have achieved 50 % and 80% in Tor and JAP respectively. After that, they launched open world experiment; they have achieved 73% true positive and 0.05 false positive in Tor.

Cai et al. [18] proposes a new attack towards website fingerprints. The idea of the attack basically works as the following: firstly, network traffic traces are transformed into strings. Then, Damerau-Levenshtein distance is used to compare between the generated strings. This attack has been evaluated against, SSH tunnel, SSH integrated with HTTPPOS, SSH with Sample-based morphing, Tor and Tor with randomized pipelining. Furthermore, they compared their methods against previously published papers in this field. Their attack method scores 80% success rate of 100 web pages dataset. In addition, they have extended their proposed web pages classifier to create a website classifier using HMM.

Wang and Goldberg [3] proposed a new website fingerprint attack. They employ SVM with their new proposed distance metrics. They derived their distance metrics from the way how web pages are loaded. The novelty of this work is that; they work with Tor cell traces instead of IP packet traces. Each tor cell consists of 512 bytes. Furthermore, the delete every SENDME cell traces that used by the Tor to instruct command. To evaluate their attack, they conducted two experiments: closed-world scenario and open-world scenario. They proposed a new procedure to create dataset to evaluate website fingerprint against Tor. In the closed-world experiment, they used the 100 Alexa websites URLs each with

Table 2.3: Website fingerprinting Classifiers

<b>Paper</b>	<b>Classifier</b>	<b>Features</b>	<b>Application and Accuracy</b>
Liberatore and Levine [58]	Naive Bayes	Packet lengths	90% on https traffic
Herrmann al. [2]	Multinomial Naive Bayes	Packet lengths	94% on SSH traffic, 3% on Tor traffic
Panchenko al. [57]	Adhoc SVM	Packet lengths, order, total bytes	54% on Tor traffic
Cai et al. [18]	SVM with Damerau-Levenstein edit distance	Packet lengths, order, direction	80% on Tor traffic
Wang and Goldberg [3]	SVM with a fast variant of Levenstein distance	Tor Cells	91% on Tor traffic

40 instances. Then they apply 10-cross fold validation. As a result, they achieve 91% precessions. In the open-world scenario, they use four web sites each with 40 instances. Then, the testing phase consists of trying to identify visits to those websites in the middle of 860 other website visits chosen from the top 1000 Alexa websites. The precision was also around 90%. Training the SVM is very processing intensive, therefore the authors used a high-performance parallel computing by the Canadian academic consortium. The authors proposed a fast version of their classifier which runs much quickly than the regular one but with slightly less precision.

The most recent contributions focused on Tor anonymity protocol and showed very promising precision results are summarized in (Table 2.3).

## 2.4 Similarity hashing

In digital forensics, most of the tools used to determine similarity of binary data use hashing. Vassil Roussev [59] went through the history of data fingerprint and its application so far and predict for promising future of similarity hashing uses in digital forensic. In 1981, Michael Rabin[60] has utilized the random polynomial to design a real-time string matching algorithm to detect unauthorized changes to files. Later, his work has been extended to improve pattern matching [61]. These research got interest after that. Consequently, several applications showed up such as *sif* tool for Unix to measure similarities among text files [62], copy-detection scheme [63], and detect similarities among Web pages [64].

One major area of our interest that utilize data fingerprinting is Payload attribution systems (PASs), which are a major area of network forensics. PAS basically take the packets' traces, take their digest hashes, store those hashes, then it provides a look up interface used to inquiry the existence of a specific byte sequence. In practice, normally Bloom Filter used as a data structure for PDAs implementation. It has been Implemented in different flavor: hierarchical Bloom filters (HBFs)[65], rolling Bloom filter (RBF)[66] and spam filtering [67].

Another pioneered work of fingerprint related to our work named Autograph [67]. Autograph is a spam filter tool. It is the first work that employs data fingerprinting approaches. It utilizes the Rabin scheme [60]. First, it separates suspicious TCP flows from innocuous ones. Then, It breaks the input network traces into blocks. Then, it computes the most likely statistical ones, and finally

hashing them using Rabin fingerprint.

A first implemented similarity hashing called fuzzy hashing or similarity hashing [68]. It utilizes the idea behind the spam filter work Autograph. Fuzzy hashing simply works as the following: First, it divided provided stream data in equal blocks, after that it create a hashed for each block-basically 6-digest. Finally, it combines all the blocks hashes as a one-line string, which is base64 encoded. To see whether two objects have similarities or not, it compares the string hashes of the two objects using edit distance measure. Finally, it produces result as a score between 0 and 100. This approach is not suitable to estimate similarity between traffic traces because a simple bit insertion or deletion anywhere in the data turn all block hashes different.

After that, Roussev propose a new similarity hashing data fingerprint tool named *sdhash* [19, 20]. It also called approximate hash based matching (AHBM). It works by selecting features that are statically impossible to happen by accident. Selecting the statistically improbable features consists in picking the features with the lowest precedence values. The precedence score is proportional to how common the corresponding entropy value is found in the data. That is, if a feature has a very common normalized entropy ( $H_{norm}$ ) value, it gets a high value; Whereas features with low  $H_{norm}$  values will get small  $S_{prec}$  values. finally, selected features are represented by consecutive Bloom filters. Computing the similarity between two data objects consists in comparing their Bloom filters. Hence, the core of the similarity computation process is the comparison of two Bloom filters which is



based on a modified version of Hamming distance. Hamming distance measures the minimum number of substitutions required to change one object into the other distance. A detailed explanation of this approach as well as how We have customized its heart of internal implementation to be suitable in network traffic analysis field is elaborated in chapter three.

*ssdeep* and *sdfhash* have common objective but they use different methodologies. They Intend to check the similarity between two objects using data fingerprinting approach. Roussev [69] had conducted experiments to test the capability of both *ssdeep* and *sdfhash*-our customized used tool(AHBM). The experiments showed that *sdfhash* is capable of identifying the existence of an embed object on the target object, regardless of the size of target object that contains an embedded object. On contrary with *ssdeep*; it is incapable to identify the existence of an embed object on the target object as long as the size of the embedded object is less than the one-third of the size of targeted object. For example, *ssdeep* is not capable of identifying the existence embedded jpg image which it average size 143 KB within a targeted document (doc/pdf/ppt/xls) that their size is in between 516-1,982 KB.

In a more recent work, Jang et al. [10] used feature hashing to identify similarities between malware binaries. Each malware binary is split into 16 bytes blocks (n-grams). These n-grams are hashed and for each malware a bitvector is generated to indicate if a feature is present or not (0 or 1). Unlike Bloom filters which use several hash functions, feature hashing uses only one hash function. Finally,

Jaccard similarity computed on bitvectors is used for malware clustering. The implemented system, BitShred, has similar accuracy to existing malware clustering systems, but is much more faster.

## 2.5 Gap analysis

Based on the literature, we have presented earlier, we cover these gaps:

1. No previous work has tailored similarity hashing approach into pattern-based network traffic analysis.

We tailored the similarity hashing approach to work on the most stable network packets' features- packet length, direction, and order- to train our final network traffic analysis models.

2. No previous work has used similarity hashing to evaluate privacy enhancing technologies.

To evaluate privacy enhancing technologies, we have emulated a well-known privacy attack named website fingerprinting over Secured VPN, secured wireless network(WiFi) and Tor browser.

3. No previous pattern-based network traffic analysis proposed solution suitable for telling if a large traffic trace contains a trace of a specific item (e.g. website or malware)
4. No previous work has employed similarity hashing approach to cluster malware into groups based on malware's network traffic behavior.

we have utilized similarity hashing comparisons equations as a metric distance to cluster malware.

## CHAPTER 3

# RESEARCH QUESTIONS

Considering the gaps discussed earlier, in this section 2.5, we have formulated the following research question:

- *RQ1* : Can similarity Hashing be used in network traffic analysis?
  - *RQ1.1* :How similarity hashing approach could be tailored in order to be used in network traffic analysis fields?
- *RQ2* : Can Similarity hashing solution -network traffic version- are efficient against privacy enhancing technologies?
  - *RQ2.1* :How similarity hashing solution -network traffic version- models are going to be trained and tested?
  - *RQ2.2*: Does the similarity hashing solution- network traffic version- offer scalability?
- *RQ3* : Can similarity hashing solution emulate finding a needle in a haystack scenario -network traffic version?

- *RQ4* : How similarity hashing solution- network traffic version- can be utilized in malware traffic detection?

Considering *RQ1*, similarity hashing has been previously employed in network traffic analysis. However, it was incorporated as payload-based inspection solution while our proposed solution is considered under the umbrella of pattern-based techniques. As our work, in Autograph [67], signatures are hashes of traffic data blocks. Another similarity is that they carried out an experimental analysis of the different parameters of the system (size of block, percentage of covered flows, etc.) to optimize their values. However, Autograph deviates from our approach in four main aspects. First, the signatures are formed by simply concatenating selected block hashes, while we use the more efficient bloom filters representation. Second, the most frequent blocks are selected as features, while in our case we select the least frequent blocks as features. Third, it is applicable on plain-text data (not encrypted), in particular HTTP traffic, while our technique is applicable for plain-text as well as encrypted traffic. Fourth, the system is tailored to only worm detection<sup>1</sup> while ours is usable for any traffic analysis task, in particular, malware detection, traffic classification, and website fingerprinting.

Considering *RQ2*, according to our knowledge no previous work has used similarity hashing over privacy enhancing technologies.

Considering *RQ3*, a large body of works in the literature focused on applying machine learning techniques to network traffic in order to extract patterns from the traffic traces. The existing pattern-based techniques suffer from that, they are

---

<sup>1</sup>Autograph does not work for typical malware.

not suitable for telling if a large traffic trace contains the trace of a specific item (e.g. website or malware).

Considering *RQ4*, similarity hashing has been previously used to fingerprint and cluster malware. Similarity hashing was employed over malware's executable binary files while our approach utilizes only the malware exchanged packets features to cluster malware.

In the following chapters, we address the answers of previous gap's questions.

## CHAPTER 4

# APPROXIMATE HASHING: THE TRAFFIC ANALYSIS VERSION

### 4.1 Approach motivation

Most of the existing pattern-based traffic analysis techniques learn a model/signature for each class (website, malware, etc.) using a set of labelled traffic samples for each class. The signature is then used to identify occurrences of that class in unknown traffic. The signature is expressed in terms of data features such as the packet lengths, packets direction, total number of packets in each direction, etc. For example, in presence of  $n$  features, using SVM consists in considering each labelled sample as a point in the  $n$ -dimensional space and trying to find separating hyperplanes that maximize the gaps between classes. Hence, models tend to be

complex and require non-trivial computations.

Finding similarities between data objects<sup>1</sup> is a typical problem in digital forensics. A typical scenario consists in looking for a similarity between a reference data object (image or office document) and a target data object under investigation (live memory dump, captured traffic dump, etc.). Compared to the previously mentioned traffic analysis techniques, the ones used in digital forensics are simpler but more scalable. It is important to note that digital forensics techniques are based on string comparison where each data object is considered as a string of bytes. Hence measuring similarity is based on string matching.

Hashing is very common forensics tool for string matching. A typical use case is when checking the integrity of a whole target (file, sequence of packets, etc.) by comparing before-and-after states of the file. Simple hashing is considered a good compression mechanism since it allows to generate a unique fixed-size signature for data objects of any size. However, if the goal is to discover occurrences of only parts of the reference data object in the target data object, simple hashes are not suitable. Instead, a common method is to increase the hashes' granularity. That is, splitting the reference and target data objects into smaller blocks, hash each one of them, and keep a list of hashes for both data objects. Finding matches between hashes indicate the same block is occurring as-is in the reference and target data objects.

Simple hashing, regardless of its granularity, has two limitations. First, it

---

<sup>1</sup>In this discussion, we refer to the first and second data object as *reference and target* respectively.



allows to pinpoint occurrences of exact copies. Any alteration (even 1 bit) yields a hash mismatch. This problem is also known as hash fragility. Second, hashed blocks should be aligned the same way in the reference and target data objects. Any displacement in block boundaries yields a hash mismatch.

To address the hash fragility problem, data fingerprinting [60] is typically used to find similar objects instead of exact object copies. The idea is to select a set of representative features for each object, then the similarity is computed in terms of the level of correlation between the features. Data fingerprinting is known to be resilient to small alterations.

To address the block alignment problem, a technique called Winkowing [70] is used. Winkowing works as follows. Assuming the block size is  $k$  and starting from the beginning of the data object, for each sequence of  $k$  consecutive characters, compute the hash and store it in an array. Hence, the first entry of the array is the hash of the characters from 1 to  $k$ , while the second entry is the hash of the characters from 2 to  $k + 1$ , and so on. Then, using a sliding window select a hash for each window position.

Using data fingerprinting and Winkowing raises three important questions:

1. how features are selected from a data object?
2. how features are stored efficiently?
3. how the correlation/similarity between two sets of features is measured?

Before discussing these questions and how we address them in the context of traffic analysis, we need to define the data object and data feature terms. A data

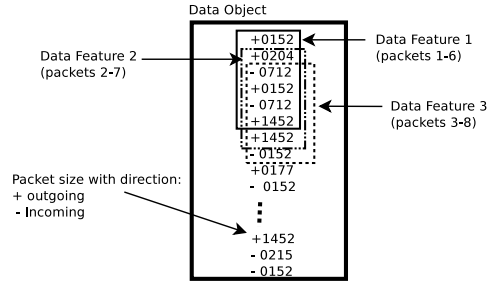


Figure 4.1: Data Object and Data Feature.

object is a sequence of packets corresponding to a specific network activity (e.g. website visit, malware communication session, video communication session, etc.). For each sequence, only the following information is kept: packet lengths, packet order, and packet directions. This "meta-data" information is available even if the traffic is encrypted. A data feature is a sub-sequence of packets in a data object. Figure 4.1 shows a snippet of a data object with the first three data features.

The proposed traffic analysis technique is based on data fingerprinting and windowing and is inspired by Roussev's approach of data fingerprinting with similarity digests [19, 20]. Using this approach, the short answers to the above questions are as follows. Given a data object (e.g. samples of a website visit), the technique aims at selecting features (Question 1) that are least likely to occur in other data objects (e.g. samples of a different website visits) by chance. Selected features are then hashed and stored using Bloom filters (Question 2) which allow significantly compact representation and fast membership queries. Hence, each data object will be represented by a Bloom filter. Measuring the correlation/similarity of two data objects (Question 3) consists in comparing their Bloom filter representations. Full details of the approach is provided in the next sections.

The proposed traffic analysis approach has several attractive advantages:

- Generating the signature of a class of traffic (website, malware, etc.) is very efficient.
- A signature can be generated using a few number of samples without significantly decreasing the detection accuracy (As an extreme case, a signature can be generated based on a single sample).
- Signature representation is very compact since it is using Bloom filters.
- Similarity can be computed efficiently between data objects of arbitrary sizes.
- It removes the need to split the traffic in order to match a given signature (Splitting problem [71]).

In the light of these advantages, the proposed approach can be applied at a new level of scalability. A typical use case can be an attacker through which a huge amount of traffic is flowing (e.g. ISP) and who is trying to pinpoint a specific type of session (e.g website, malware, etc.) in the traffic and in real-time. In addition, updating the bank of signatures can be done quickly and without requiring a large number of samples.

## 4.2 Hash-based signature generation

Our approach's signature generation process is based on data fingerprinting [60]. Given a set of samples of a specific network communication (website visit, malware

communication, etc.), data fingerprinting consists in selecting a set of features that uniquely identify that specific data network communication. For example, if the samples correspond to facebook website visits, selected features should have a high probability to be part of facebook visit samples, but a low probability to be part of other traffic samples. The criteria we use for selection and that satisfies these requirements is the following: choosing uncommon (statistically improbable) features that happen to be part of the available samples. In order to characterize common and uncommon features, two approaches are possible: using entropy measure or a hash function.

#### 4.2.1 Feature entropy vs feature hash

In order to characterize statistically improbable features, Roussev [19, 20] used entropy values. Entropy measures the amount of information contained in the data. Intuitively, it measures the level of uncertainty in the data. For example, a feature composed of repeated packet lengths in the same direction results in a relatively small entropy value.

Let  $\{X_1, X_2, \dots, X_B\}$  denotes the sequence of packets of a feature  $F$ . The entropy of the feature  $F$  is defined as follows:

$$E(F) = - \sum_i P(X_i) \log_2 P(X_i) \tag{4.1}$$

where  $P(X_i)$  is the empirical probability of  $X_i$  in the string of packets.  $P(X_i)$  is proportional to the number of times it is repeated in the feature. For example,

consider the following small feature  $F_1$ :  $\{1392, -56, 56, 204, -56, 204, -115, 204\}$ .

The alphabet of the feature is composed of 5 different packet items 1392, -56, 56, 204, and -115. The empirical probability for every symbol is:  $P(1392) = 0.125$ ,  $P(-56) = 0.25$ ,  $P(56) = 0.125$ ,  $P(204) = 0.375$ ,  $P(-115) = 0.125$ . Using Equation (4.1) results in the value  $H(F_1) = 2.155$ . The maximum entropy value is obtained when all bytes of the feature are different, that is,  $\log_2 B$ .

To simplify processing and in order to make the entropy value independent from the size of the feature ( $B$ ), a normalized entropy value is used:

$$E_{norm}(F) = \lfloor 1000 \times H(F) / \log_2 B \rfloor \quad (4.2)$$

The normalized entropy leads a value between 0 and 1000 regardless of the length  $B$  of the feature.

On the positive side, entropy values are easy to compute and can be normalized to values between 0 and 1000. On the negative side, entropy measure does not take into consideration the order of the packets. That is, two features with the same set of packets but in different order produce the same entropy value. For example,  $F_2$ :  $\{56, 1392, 204, -56, 204, -56, -115, 204\}$  produces the same entropy value as  $F_1$  ( $E(F_1) = E(F_2) = 2.155$ ) since they share the same set of packets, but in different order.

Since in our traffic analysis scenario, the order of packets is important, we use instead a hash function to characterize features. For every feature, a SHA-1 digest is computed. In order to map the 160 bits SHA-1 digest to a value between 0 and

1000, we consider only the 10 rightmost bits.

$$H_{norm}(F) = 2^{first10bits(SHA-1(F))} \quad (4.3)$$

Both entropy and hash functions lead to mapping conflicts (several different features mapping to the same value). Entropy measures are slightly faster to compute, but with hashing, packet order is preserved.

### 4.2.2 Feature selection criteria

A data object of length  $L$  packets contains  $L - B + 1$  features. Data fingerprinting approach consists in selecting only a subset of these features to uniquely identify the data object. A good feature candidate for selection should satisfy two criteria:

- criteria 1: it should be statistically improbable
- criteria 2: it should be part of most of (or all) data samples

For criteria 1, the statistical probability of a feature depends on the likelihood of its normalized hash value. To this end, we estimate the probability distribution of all possible normalized hash values  $[0, 1000]$ . For example, Figure 4.2 shows the empirical probability density function for website browsing through VPN dataset and Tor dataset(Cai's dataset) in addition to malware dataset. The dataset is generated by automatically fetching the Alexa's top 100 websites through VPN 40 times. For each feature in the data object, we compute its normalized SHA-1 digest and use the value to identify its statistical probability from the distribution.

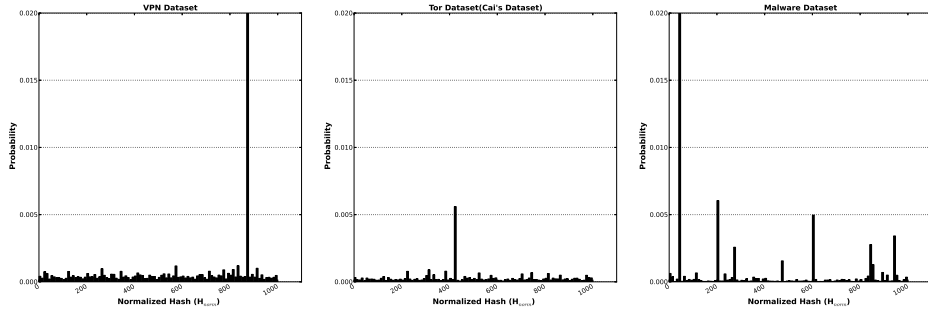


Figure 4.2: Probability density function of normalized Hash values for VPN web browsing dataset, Cai’s tor dataset and malware dataset.

This is implemented by a rank table of size 1000 mapping normalized hash values to a precedence score ( $S_{prec}$ ) which is a value between 0 and 1000 proportional to the probability value. It is important to note that a different rank table is generated for each type of traffic. For the sake of our experimental analysis, 3 rank tables have been generated from 3 different datasets:

- Web browsing through VPN dataset
- Malware traffic dataset
- Web browsing through Tor dataset

This technique of selecting statistically improbable feature using empirical probability density functions is discussed in details by Roussev [20].

For every feature, the precedence score indicates how common the corresponding normalized hash value is found in the empirical data. That is, if a feature has a very common  $H_{norm}$  value, it gets a high  $S_{prec}$  value. Whereas features with low  $H_{norm}$  values will get low  $S_{prec}$  values. Selecting the statistically-improbable

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
$H_{norm}$	460	557	580	510	575	805	575	515	575	566	568	566	515	566	568	562	460	566	575	575	
<b>Step 1</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 2</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 3</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 4</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 5</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 6</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 7</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 8</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0
<b>Step 9</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	4	0	0	0	0	0	0	0	0	0	1	0	0	0
<b>Step 10</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	4	0	0	0	0	0	0	0	0	0	2	0	0	0
<b>Step 11</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	4	0	0	0	0	0	0	0	0	0	3	0	0	0
<b>Step 12</b>	$S_{prec}$	520	610	600	592	604	612	604	596	604	614	628	614	596	614	628	638	520	614	604	604
	$S_{pop}$	1	0	0	3	0	0	4	0	0	0	0	0	0	0	0	0	4	0	0	0

Figure 4.3: Computing the popularity score ( $S_{pop}$ ) on a data sample of size 28 packets and using a sliding window of size 9.

features consists in picking the features with the lowest  $S_{prec}$  values. A straightforward approach consists in ordering all the data features across the data object in ascending order of their  $S_{prec}$  values and choosing the top ones. The problem is that all selected features will originate from the same region/cluster of the data object. Winnowing [70] technique uses a sliding window to pick features with local minima. This guarantees that features are selected from various locations in the data object.

Let  $W$  be the window size. While the window is sliding across the data, a feature with the lowest  $S_{prec}$  is marked<sup>2</sup> at every step. If the same feature is marked a number of times ( $1 \leq k \leq W$ ), it can be considered as a feature with a local  $S_{prec}$  minimum and consequently selected as statistically improbable. This is achieved by maintaining a popularity counter score  $S_{pop}$  which keeps track of how many times each feature has been marked due to a minimum  $S_{prec}$ .

The actual feature selection is based on setting a threshold value  $t$  ( $1 \leq t \leq W$ )

<sup>2</sup>If two features happen to have the same  $S_{prec}$  value in the current window, only the leftmost one is marked.



such that any feature with a popularity score  $S_{pop} \geq t$  is selected as statistically-improbable. In our case, we use a threshold value  $t = 2$ .

Figure 4.3 shows how  $S_{pop}$  values are updated while a sliding window of size 9 is moving through a data input sample. The upper row shows the  $H_{norm}$  values. After the sliding window reaches the end of the data object, the final  $S_{pop}$  values (step 12) are used to select the statistically-improbable features. For instance, if the threshold  $t = 2$ , three features will be selected (positions 4, 8 and 17) whereas for  $t = 4$ , only two features will be selected (positions 8 and 17).

Once a feature is considered statistically improbable (passed the winnowing process), it passes through the second filter (criteria 2) which checks if the feature occurs in the other samples of the same class. Keeping only the features that occur in all other samples is too restrictive and result in a small number of features. On the other hand, allowing all features that occur in at least half of the samples is too permissive and result in a large number of features. Empirical analysis showed that the best threshold value is to keep only features that appear in at least 75% of the other samples.

### 4.2.3 Building fingerprint representations

Having selected a set of features to represent a data object, the next step is to build a fingerprint representation out of the features. A simple approach consists in computing the digest (e.g. MD5) of each feature and concatenating the digests to obtain a fingerprint representation of the data input. This approach is inefficient

in two aspects. It is space inefficient and it handles membership queries (checking if an element is part of the set) inefficiently. Note that this simple approach does not incur any false positive rate for membership queries. A more efficient approach for set representation is using Bloom filters [72]. The approach trades space and membership queries efficiency for a small false positive rate in membership queries.

A Bloom filter is an array of bits of a fixed size  $m$  initially set to 0. Given a set of elements  $\{s_1, s_2, \dots, s_n\}$ , inserting an element  $s_i$  in a bloom filter consists in computing hash values of  $s_i$  according to  $k$  different hash functions  $h_1, h_2, \dots, h_k$ . Each hash function maps the universe of possible element values to an index in the bloom filter (i.e. in the range  $1 \dots m$ )<sup>3</sup>. For each of those hash values, the Bloom filter entry is set to 1. Checking the membership of an element  $s_j$  goes through the same process, that is, computing the  $k$  indices using the  $k$  hash values. If all the bits on the  $k$  indices are equal to 1, the membership query returns true. Otherwise, it returns false. As mentioned above, Bloom filter approach for set representation trades efficiency for a small false positive rate in membership queries. Indeed, in membership queries, if at least one entry of the computed  $k$  indices is set to 0, we are sure that the element is not in the set. However, if all the  $k$  entries are set to 1, we are not 100% sure that the element is in the set. The  $k$  entries might be set by chance. The good news is that, given the number of elements already inserted in the Bloom filter, the false positive rate can be easily estimated [72].

Features are inserted in Bloom filters as follows (Figure 4.4). The sequence

---

<sup>3</sup>The hash functions are independent and maps the input to the range  $1 \dots m$  uniformly.

of packets (packet lengths and direction) is hashed using SHA-1 algorithm which generates 160 bits. The hash is then split into five 32 bits sub-hashes. Each sub-hash is considered as different hash value. Hence,  $k$  value in this case is 5. The 11 least significant bits of each 32 bits sub-hash is used as index in the Bloom filter array of bits. The features are inserted in 256-byte Bloom filters ( $m = 256 \times 8 = 2048 = 2^{11}$ ). To keep the false positive rate negligible, the capacity of a single Bloom filter is fixed to a default value of 128 features. Hence, the maximum number of marked bits in the Bloom filter is  $128 \times k = 128 \times 5 = 640$  bits. When the capacity is reached, a new Bloom filter is created, and so on, until all the features are represented.

The process of inserting features in Bloom filters is illustrated in Figure 4.4.

The fingerprint representation of a data object is the concatenation of all Bloom filters preceded by their total number. The obtained representation is called similarity digest (SD):

$$SD(dob) = s|bf^1|bf^2|bf^3|\dots|bf^s \quad (4.4)$$

where  $s$  is the number of Bloom filters and  $bf^i$  is the  $i^{th}$  Bloom filter of data object  $dob$ .

#### 4.2.4 Comparing similarity digests

Computing the similarity between two data objects consists in comparing their Bloom filters. Hence, the core of the similarity computation process is the com-

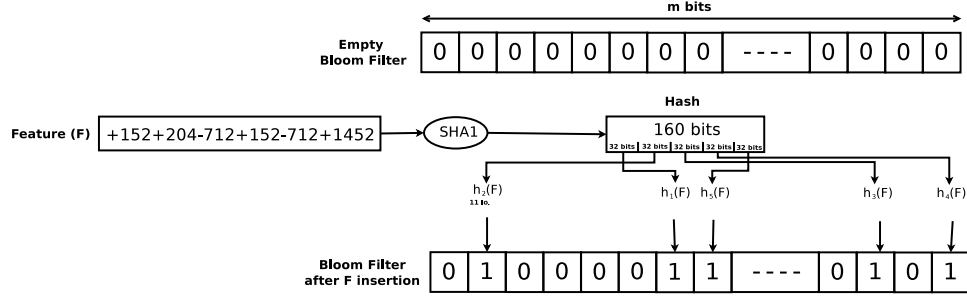


Figure 4.4: Process of inserting a feature into a Bloom filter

parison of two Bloom filters which is based on a modified version of Hamming distance. Hamming distance measures the minimum number of substitutions required to change one object into the other.

Let  $bf_1$  and  $bf_2$  be two Bloom filters and consider the following variables:

- $n_1$  and  $n_2$ : the number of features in  $bf_1$  and  $bf_2$  respectively.
- $e_1$  and  $e_2$ : the number of bits set to one in  $bf_1$  and  $bf_2$  respectively.
- $e_{12} = bf_1 \cap bf_2$ : the number of bits set to one in both  $bf_1$  and  $bf_2$ .

The modified Hamming distance (MHD) between two Bloom filters is defined as follows:

$$MHD(bf_1, bf_2) = \begin{cases} -1, & \text{if } n_1 \leq N_{min} \\ 0, & \text{if } e_{12} \leq t_c \\ 100 \frac{e_{12} - t_c}{E_{max} - t_c}, & \text{otherwise} \end{cases} \quad (4.5)$$

where,

- $N_{min}$  is the minimum number of features in a Bloom filter that is needed to compute a meaningful  $MHD$  value.

- $t_c$  is a threshold below which any bit matching is assumed to be due to chance:

$$t_c = \alpha(E_{max} - E_{min}) + E_{min} \quad (4.6)$$

- $E_{max}$  is the maximum number of matching bits due to chance:

$$E_{max} = \min(n_1, n_2) \quad (4.7)$$

- $E_{min}$  is the minimum number of matching bits due to chance:

$$E_{min} = m(1 - p^{ke_1} - p^{ke_2} + p^{k(e_1+e_2)}) \quad (4.8)$$

- $p$  is the probability that a specific bit is still 0 after the insertion of all features.
- $\alpha$  is a calibrating parameter that is set experimentally.

We now have all the ingredients to compute the similarity score (SC) between two data objects. Let  $dob_1$  and  $dob_2$  be two data objects. Let  $bf_1^1, bf_2^1, bf_3^1, \dots, bf_s^1$  be the set of Bloom filters from  $dob_1$  and  $bf_1^2, bf_2^2, bf_3^2, \dots, bf_t^2$  be the set of Bloom filters from  $dob_2$ . Assuming that  $s \leq t$ , the similarity score (SC) between the two data objects is defined as follows:

$$SC(dob_1, dob_2) = \frac{1}{s} \sum_{i=1}^s \max_{j \in [1..t]} MHD(bf_i^1, bf_j^2) \quad (4.9)$$

The similarity score as defined in Equation (4.9) is typically used to find a similarity between a small data object (e.g. file) and a much larger data object (e.g. hard drive), hence,  $s \ll t$ , that is, the number of Bloom filters is much larger in  $dob_2$  than in  $dob_1$ . The similarity score computation consists in comparing each Bloom filter of  $dob_1$  with every Bloom filter in  $dob_2$  and keep the highest value. The kept highest values are then averaged to produce the similarity score. Because the score is the average of the  $s$  maximum distances between the bloom filters of  $dob_1$  and all the bloom filters of  $dob_2$ , a large size of  $dob_2$  (i.e.  $t$ ) does not dilute the similarity score.

Bloom filters are just arrays of bits. Two data objects having similar data regions will have similar Bloom filters around those regions. By comparing each Bloom filter of the first data object with all Bloom filters of the second and taking the maximum value, the similarity score will reflect the presence of such overlapping.

#### 4.2.5 Similarity hashing time complexity

Building and comparing final signatures based on the operations over bloom filters.

Initialization operation (*Initialize*) simply assign 0 to each bloom filter bytes. Therefore, if the bloom filter size is  $m$  then the time complexity of the bloom filter's initialization operation is  $O(m)$ . However, in our implementation, bloom filter size is 256 bytes; consequently, initialization operation can be done in a constant time.

Bloom filter insertion Operation (*Insert*) insert selected operation into bloom filter. In theory, inserting an element into a bloom filter requires  $k$ -independent hash functions applied over the selected features. Therefore the time complexity for this operation would be  $O(K)$ . However, in our implementation, only one hash function is applied (SHA1) but the resulted hash digest is split into five parts each 32-bits. Each part is treated as if it is a separated hash function. Consequently, the time complexity of our implementation could be achieved in a constant time  $O(1)$ .

Bloom filters Comparisons operation (*Compare*) is the heart of the similarity hashing approach. To do so, both bloom filter should be of the same size as well as use the same hash function. Bloom filters Comparisons operation basically compares the two bloom filters via applying the AND bit-wise operation which has a linear time complexity  $\Theta(m)$ , where  $m$  is the size of each bloom filter.

In our implementation, each signature consists of a variable number of bloom filters. It utilizes the Multi-resolution similarity hashing proposed in [73] which allow Data objects of variant sizes to be compared against each other. Having two similarity digest signatures each consists of  $n$  bloom filters while every bloom filter's size is  $m$ , Asymptotically, the time complexity would be  $O(n^2m)$  since every bloom filter of the first signature has to be compared against all the bloom filters of the second data object bloom filters. where  $n$  represent the number of bloom filters in each data object and  $m$  is the size of each bloom filter.

According to the author of Multi-resolution similarity hashing [73] going through

1024 bloom filter comparisons- 256KB AND bit-wise operation- is a trivial task for the modern machine. However, 256MB bloom filters comparisons would not be a trivial task. Furthermore, 1TB bloom filters comparisons would be unfeasible. Unfortunately, manipulating similarity digest approach parameters can alleviate the amount of calculation as pinpointed in the next section.

#### 4.2.6 Parameters values

The AHBM approach to traffic analysis depends on a set of parameters. The first parameter is the feature size  $B$  which represents the number of packets to include in a feature. Choosing a small feature size increases the granularity of the data object representation while losing specificity. The number of selected features tends to be larger. Choosing a large feature size increases the specificity of the selected features while reducing their numbers.

The window size specifies the number of features to consider at each step before marking the feature with the lowest  $S_{prec}$  score. A large window size means that a feature will be marked among a large number of considered features. This yields to fewer features marked with higher popularity scores. A small window size yields to more features marked but with lower popularity scores.

A small threshold  $t$  value results in a large number of selected features which leads to a bulky representation of input data but reduces the false positives rate. On the other hand, a large threshold value results in a small number of selected features which improves compression but may lead to a higher false positives rate.



## CHAPTER 5

# WEBSITE FINGERPRINTING OVER VPN

Typically, Virtual Private Networks (VPNs) are used to extend a private network across public networks [74]. Packets are communicated through an encrypted tunnel (SSL, IPsec, SSH, etc.). Since an observer of a VPN traffic can only see encrypted packets to and from the VPN server, VPN is also used to hide user activity and to bypass simple proxy filtering and censorship. Therefore VPN is a typical target of website fingerprinting attacks [2, 75].

The aim of this first experiment is to evaluate the accuracy of the proposed AHBM approach on VPN traffic. The used dataset is generated by visiting the Alexa's top 100 websites through VPN. The VPN service is provided by an OpenVPN server on Amazon Web Services (AWS). Each website is fetched 40 times using Firefox 48.0.1 running on Ubuntu 16.04. A python script is used to automate the website visits. At each visit, the script clears the browser cache, launches

a tcpdump process for packet capturing, sets a timer and then fetches the URL. Each visit is given 20 seconds beyond which the connection is closed and the captured packets are stored in a pcap file. Websites are fetched in a round-robin fashion (A first sample of each website is collected, then a second sample, etc.). The data has been collected during December 2016.

## 5.1 Selecting parameter values

As mentioned above, the proposed AHBM approach depends on a set of crucial parameters, in particular, the feature size (how many packets are considered at a time) and the window size (how many features are considered in the winnowing based selection). In order to empirically choose the optimal values for those parameters, we used a modified version of 10-fold cross-validation. At each fold, 36 samples (out of 40) of each website are used to learn the model (selecting the features). The remaining 4 samples of each website constitute the testing data. However, unlike typical cross-validation, for each website we compute the AHBM similarity score between the website model (similarity digest) and all the test data (all 4 samples of all websites as a single sequence). Then, for every website, we remove the corresponding 4 samples from the test data and we recalculate the similarity. This yields, for each website and for each fold, 2 similarity scores: one score when the website samples are in the test data (typically, a high similarity score) and one score when the website samples are not part of the test data (typically, a low similarity score). Good parameter values (feature size and window

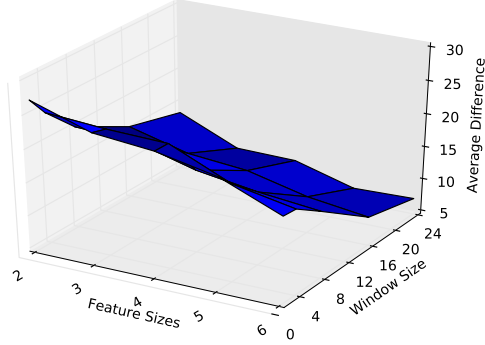


Figure 5.1: Accuracy in terms of feature size and window size.

size) should maximize the difference between the two similarity scores. Figure 5.1 shows the average difference between the two similarity scores, for various feature size and window size values.

The combination that yields the best results is a feature size of 2 and a window size of 4. Although a feature size of 2 packets is relatively small, but this corroborates previous results [76, 77] where the optimal traffic analysis results were obtained by considering consecutive pairs of packet size values.

### 5.1.1 Cross-validation results

Most of existing website fingerprinting techniques are evaluated through typical cross-validation. Cross-validation is a statistical method for accuracy estimation. It works by dividing data into two segments: one used to train the model (or models) and the other used to validate the model. In typical cross-validation, the training and validation sets must cross-over in successive rounds such that

each data point has a chance of being validated against. The basic form is  $k$ -fold cross-validation where the data is first partitioned into  $k$  equally sized segments or folds. Subsequently,  $k$  iterations (or folds) of training and validation are performed using a different segment each time. Our experiment consists in applying a 10-fold cross-validation on the collected data. That is, in each fold 36 samples are used for selecting the features and the remaining 4 samples are used for testing. This process is repeated 10 times (folds) choosing in each fold 4 different samples for testing.

The measures we used for the precision are: *precision*, *recall* and *F-measure*. In our context, *precision* measures the fraction of website samples identified correctly. *Precision* is computed as follows:

$$precision = \frac{tp}{tp + fp} \quad (5.1)$$

where  $tp$  and  $fp$  represent respectively the number of true positives and the number of false positives. *Recall*, on the other hand, measures the fraction of the total set of correct websites in the traffic that are identified correctly by system, that is,

$$recall = \frac{tp}{tp + fn} \quad (5.2)$$

where  $fn$  represents the number of false negatives.

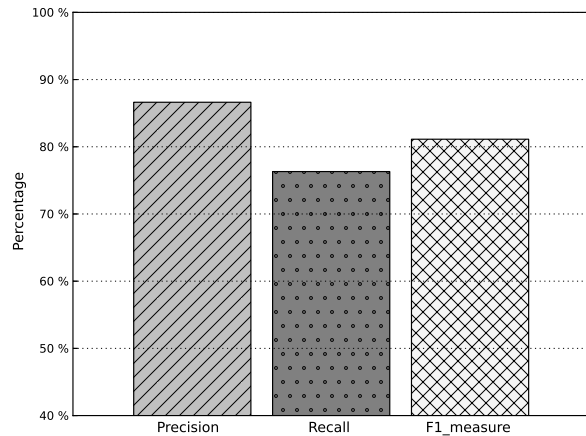


Figure 5.2: Precision, Recall, and F1 measure values for cross-validation on VPN dataset.

Finally,  $F1$  combines both *precision* and *recall* as follows:

$$F1 = 2 \times \frac{\textit{precision} \times \textit{recall}}{\textit{precision} + \textit{recall}} \quad (5.3)$$

The 1 in  $F1$  comes from the fact that *recall* and *precision* are evenly weighted. There is a more general form of the  $F$ -measure where *precision* and *recall* have different weights.

Figure 5.2 shows the results of cross-validation on the VPN dataset which are 87% for the precision, 76% for the recall, and 81% for the F1 measure. Previous work on VPN traffic reported slightly better results: Herrman et al. [2] reported a 94% accuracy while Fegghi et al. [75] reported between 90 and 95% accuracy. The proposed approach is, however, significantly more scalable than previous work.

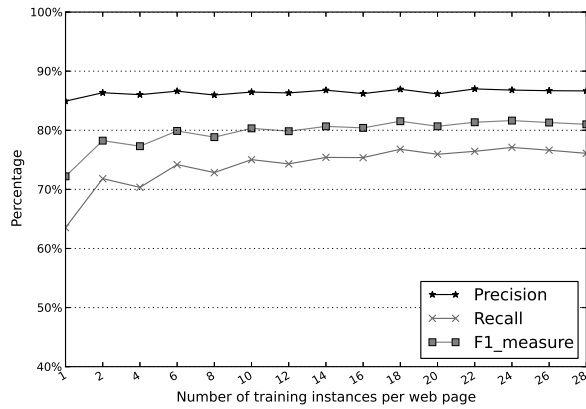


Figure 5.3: 10-fold cross-validation using different number of training samples.

## 5.2 Effect of the number of training samples

Figure 5.3 shows how the different accuracy measures (precision, recall, and F1) behave as the number of samples used to generate the similarity digest changes. The experiment consists in repeating 10-fold cross-validation starting from only one training samples up to 28 training samples. As expected, more training samples produces better accuracy measures. Interestingly, the accuracy stays acceptable (85% precision with 63% recall) even when only one sample is used to generate the similarity digest. In that extreme case, it is important to mention that all features considered statistically improbable are selected and added to the bloom filters because the second criteria (Section 4.2.2) can't be used with only one sample.

## 5.3 Identifying a website within a sequence of web activity

In all existing website fingerprinting techniques [18, 3, 57, 2] learning a website "fingerprint" requires to visit the website several times and to record the set of packet sequences (samples) resulting from those visits. The set of samples have typically comparable sizes (number of packets). This allows to extract features such as the number of incoming packets, the number of outgoing packets, the number of received bytes, etc. which allow to build a model/signature for that website. Consequently, for a website model to be efficient in identifying website occurrence in unknown traffic, it should be matched, each time, with an extracted sequence of packets having a size similar to the samples used in the training phase. Captured traffic, however, comes in the form of a long sequence of packets corresponding to all sort of user activity without clear separation. This issue was not considered in most of previous work on website fingerprinting because k-fold cross-validation is typically used for the experimental evaluation. k-fold cross-validation calculates the accuracy in terms of already "split" samples.

To identify a website occurrence within a long sequence of captured packets, there are currently two main approaches: splitting the traffic and using a sliding window. Tao and Goldberg [71] discussed the splitting problem and proposed two ways of implementing it: time-based splitting and classification-based splitting. Time-based splitting is efficient provided that the time-gap between two website visits is large. If the time-gap is small, classification-based splitting is used which

has an additional processing overhead. Either ways, the website fingerprinting attack needs to go through an initial splitting pre-processing step.

Feghhi and Leith [75] adopt a sliding window approach with a 10 packets moving step. At each step, a sequence of  $n$  packets is extracted from the traffic and matched with the website model where  $n$  is the size of samples used in the training phase. Even with a moving step of 10 packets, this approach is clearly not scalable for realistic scenarios.

The proposed AHBM approach is particularly efficient to identify website occurrences in a network traffic capture of arbitrary size. This is achieved by comparing bloom filters composing the similarity digest of the website with bloom filters corresponding to different regions of the network traffic capture. By taking the average of the maximum distances between each bloom filter of the website and all bloom filters of the unknown traffic, the final similarity score is not diluted by the size of the unknown traffic capture. Figure 5.4 shows the similarity score computation process in terms of bloom filters.

We carried out a modified version k-fold cross-validation where all samples of the test data are consolidated to form a single long sequence of packets. For each website, the similarity score with test data is computed two times: one time when the website packets are part of the test data and one time when the website packets are excluded from the test data. For each case and for each website, Figure 5.5 shows the range of all values observed in the 10-folds (dashed lines) and where 75% of the values are concentrated (closed boxes). The following interesting facts



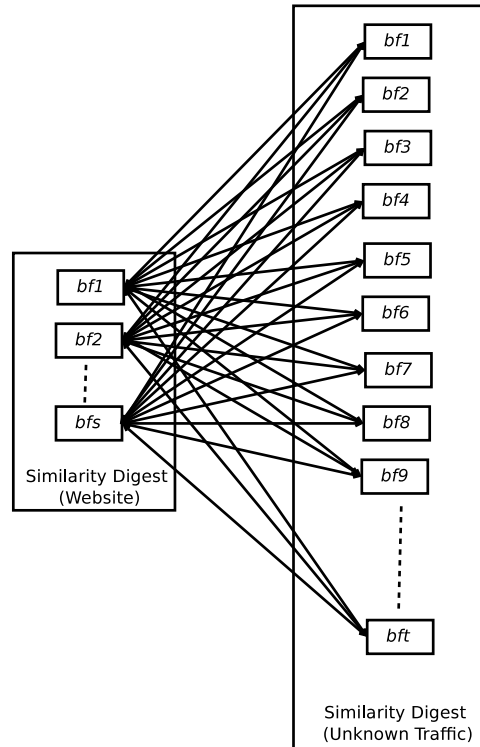


Figure 5.4: The similarity score computation between the website’s SD and the unknown traffic’s SD.

can be observed from the figure:

1. The average similarity scores, when the website is part of the test data and when it is excluded, are clearly far apart with the former larger than the latter for all 100 websites.
2. The standard deviation of the similarity scores when the website is not part of the test data is interestingly small. This justifies the selection of the threshold  $t_{sc}$  on the basis of this case (the similarity score with arbitrary traffic capture not including the website).
3. The standard deviation of the similarity scores when the website is part of the test data is relatively large. One explanation is that from one fold to

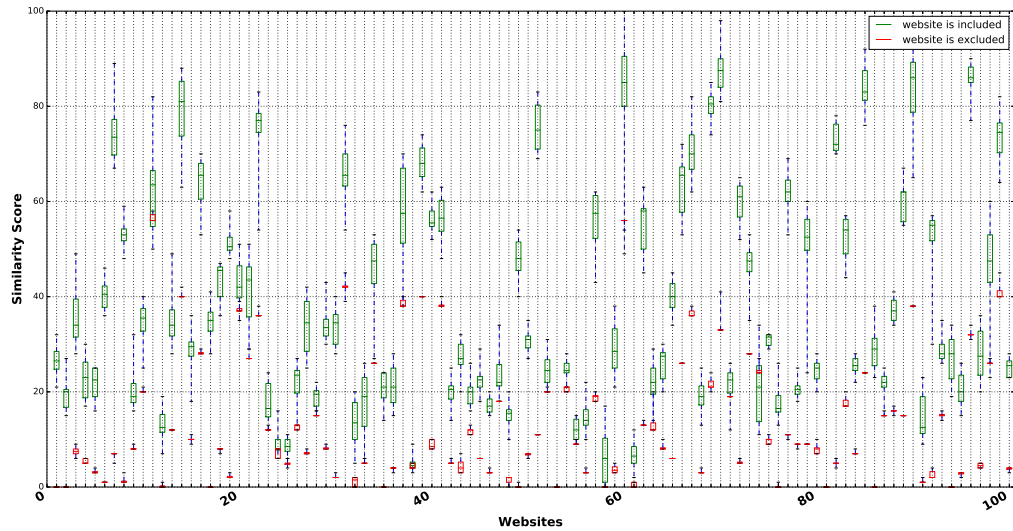


Figure 5.5: Distribution of the similarity scores for each website (1) when the website is part of the testing data and (2) when the website is removed from the testing data.

another, not all distinguishing features are selected for the website model.

Recall that features are selected provided they are part of at least 75% of the training samples.

Figure 5.6 shows the precision, recall and F1 measures for the same experiment.

Compared to Figure 5.2, values are 10% lower but are still acceptable.

## 5.4 Effect of the size of the test data

In the modified k-fold cross-validation experiment of the previous section, all samples of the test data are consolidated into a single long sequence of packets.

The website similarity digest is then matched with that long sequence. Having 100

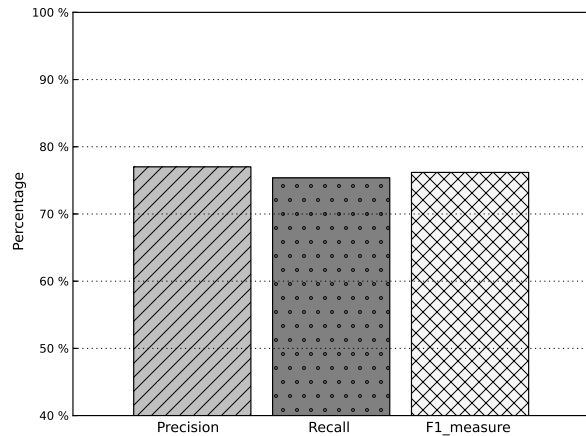


Figure 5.6: Precision, recall, and F1 measures for a modified 10-fold cross-validation (All test samples are consolidated in a single sequence).

websites, the size of the test data in each fold is approximately 1GB. To see the effect of the size of the test data on the accuracy measures, we repeated the k-fold cross-validation (Figure 5.6) while increasing the size of the test data. To this end, we collected additional web browsing data by visiting the next top 900 websites according to alexa (top 100 - 1000 websites) using the same setup (OpenVPN on AWS Cloud Service). The test data is then augmented incrementally using chunks of 200MB from the additional collected data. Figure ?? shows the accuracy measures as the test data goes from 1GB to 3GB. As the test data increases, the measure values do not change significantly. The recall (TPR) keeps the same value ( $\tilde{77}\%$ ). The precision, however, decreases slowly. The explanation is that as the test data increases, the website presence in the large test data is missed slightly more often.

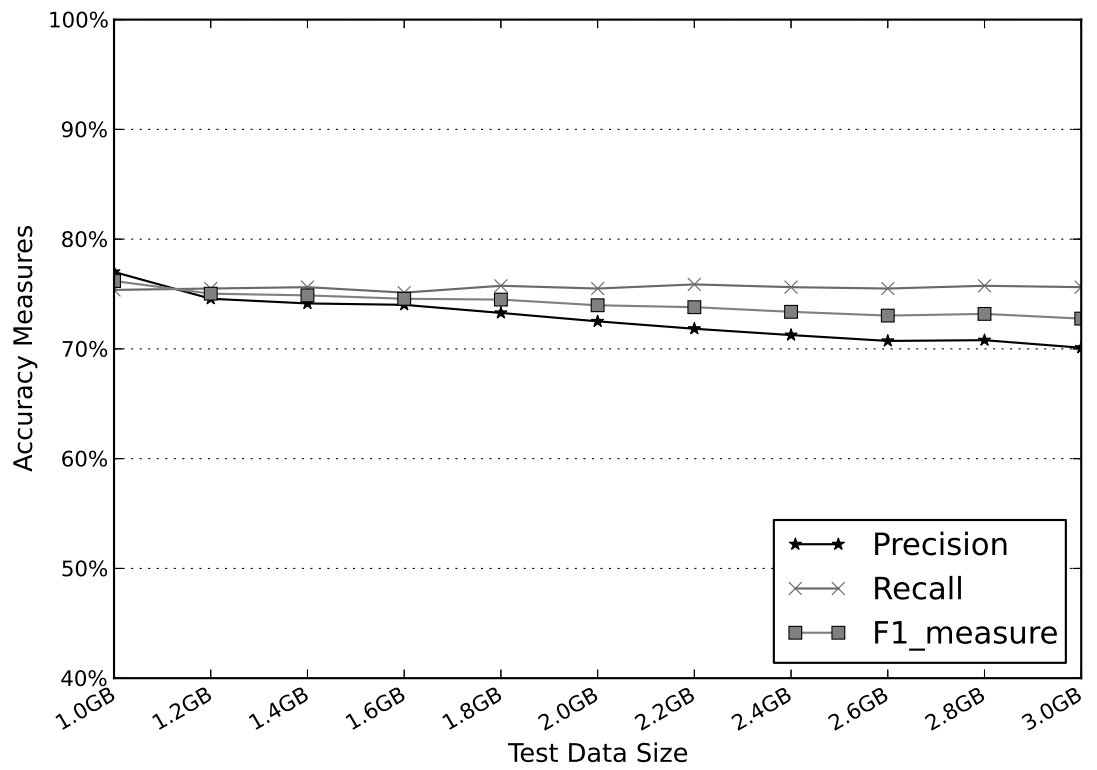


Figure 5.7: Precision, recall, and F1 measures for different test data sizes.

## CHAPTER 6

# WEBSITE FINGERPRINTING OVER TOR AND ENCRYPTED WIRELESS

### 6.1 Website fingerprinting on Tor traffic

Tor traffic is a common target for website fingerprinting attacks as Tor is an overlay network that aims to provide privacy and anonymity to its users. Previous work reported high accuracy website fingerprinting attacks on Tor [18, 3, 78, 79]. However, several concerns have been raised regarding the practicality of these attacks in realistic scenarios [80, 1, 71].

We used three datasets to assess the accuracy of the proposed AHBM approach on Tor traffic:

- Cai dataset [18]: 100 websites, 40 samples each.

- Tao dataset [3]: 100 websites, 40 samples each.
- Our dataset: 100 websites, 40 samples each.

Our dataset have been collected by visiting the top 100 Alexa websites, 40 times each, using Tor Browser. The websites are fetched in round-robin fashion and is automated using the Tor-selenium plugin. Every visit lasts for a maximum of 30 seconds and there is a 5 seconds time gap between each successive visits. Packets are stored in pcap files which are then parsed to keep only the direction, order, and size of packets (Figure 4.1). The data is collected during January 2017.

### 6.1.1 One-to-one cross-validation results

Table 6.1 shows the result of 10-fold cross-validation on the three datasets. The values are clearly lower than previous work (Cai et al. [18] and Tao and Goldberg [3]). As in the case of VPN dataset, for our AHBM approach, the best results were obtained with a feature size of 2 and window size of 4. The only exception is Cai et al. dataset where the best results were obtained with a feature size of 6 and a window size of 4. The explanation is that in Cai et al. dataset, packet sizes are rounded to multiple of 600 and consequently the number of different packet sizes becomes very limited (600, 1200, 1800, etc.) which requires a longer feature size to allow "distinguishability".

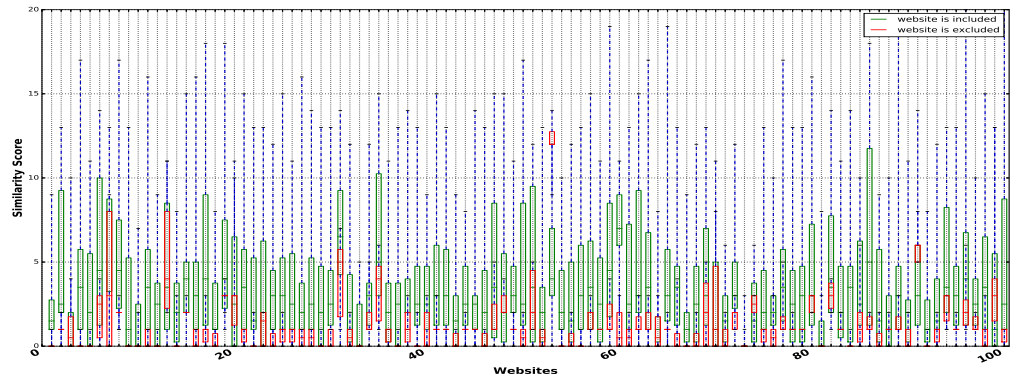
	Recall (TPR) (AHBM)	Recall (TPR) (Cai et al.)	Recall (TPR) (Tao et al.)
Our dataset	58%	–	–
Cai dataset (rounded)	64%	86%	91%
Tao dataset	53%	88%	99%

Table 6.1: Recall (TPR) for our AHBM, Cai et al., and Tao et al. approaches on three datasets.

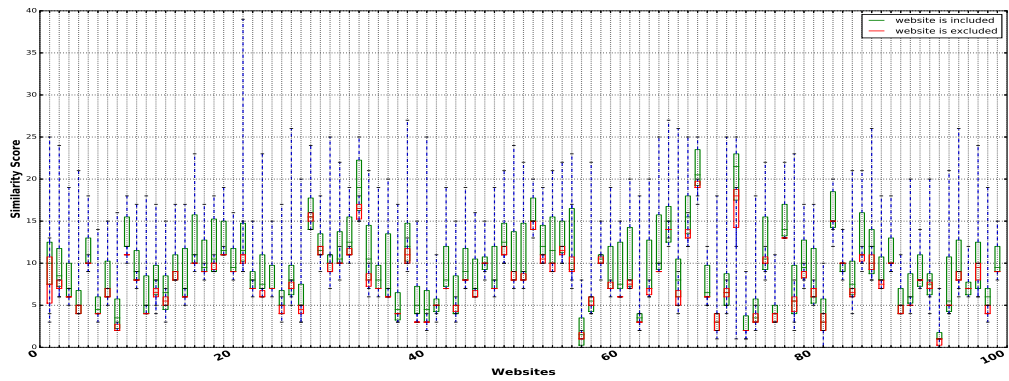
### 6.1.2 Identifying a website within a sequence of web activity through Tor

In order to assess the efficiency of the proposed AHBM approach to identify a website visit within a long sequence of network traffic packets (without splitting and using a sliding window), we used one-to-all cross-validation on the three datasets. For each website, Figure 6.1 shows the range of all values observed in the 10-folds (dashed lines) and where 75% of the values are concentrated (closed boxes).

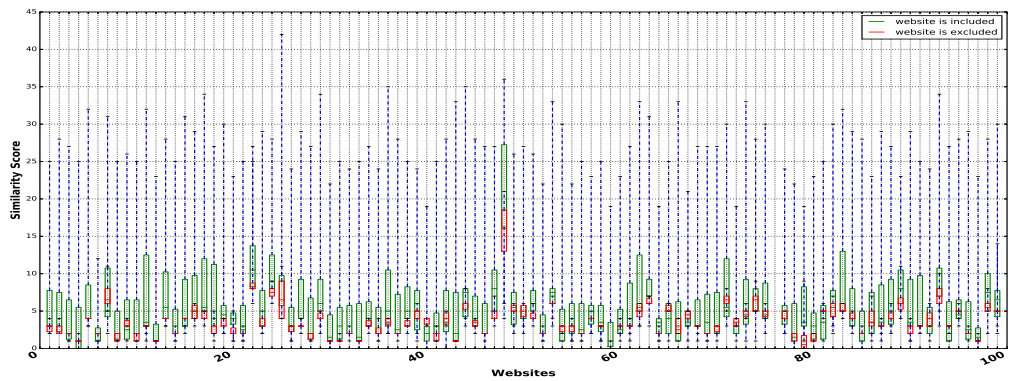
For the majority of websites, the similarity score between the website similarity digest and the consolidated test data is smaller when the website is not part of the test data. However, there is a clear overlap between the two cases (website is included and website is excluded). This explains the low recall (TPR) values for the same 3 datasets: 44% on our data, 41% on Cai data, and 31% on Tao data.



(a) Our dataset



(b) Cai et al. dataset



(c) Tao et al. dataset

Figure 6.1: Distribution of the similarity scores for each website when the website is part of the testing data (green) and when the website is removed from the testing data (red) for (a) our dataset , (b) Cai et al. dataset, and (c) Tao et al. dataset.



### 6.1.3 Effect of website's instances number in both accuracy and time calculation

To test the scalability of the state of the art contribution into website fingerprinting field on tor, We have run each implemented work with their companion dataset. every time Weconfigure each implementation with different training instances number. starting with 4 instances for each website then , 8, 12,...,40. Accuracy for each run as well as the the time in hours each implementation needed to train its final model/s. The values are summarized in table 6.2. Table shows clearly that Toe and Cai work has the superiority in this field respectively. However, to reach their high accuracy tens of hours are need to train their classifier. Our implemented approach are very quick and scalable, but it has a decent accuracy against Tor. The good news it that our approach has a very promising in other fields on network analysis while these Superior contribution at tor are not practical at the other fields due to their intensive calculation. Furthermore, a pre-processing operation has to be achieved before their classifier get trained such as packets' length rounding as with Cai et al.[18] or working with cells instead of working with packets or splitting process as with Toe et al. [3] work.

		Cai et al.		Toe et al.		Our Approach		
W.No	Inst.No	Acc	Time	Acc	Time	Acc	Time1	Time2
100	4	0	0.17	0	1.16	9	0.045	1.51
100	8	0	0.72	0	4.61	23	0.051	1.54
100	12	79.7	1.62	85.1	10.42	32	0.048	1.56
100	16	81.7	2.82	85.3	17.70	38	0.047	1.58
100	20	83.5	4.31	86.45	27.68	41	0.050	1.56
100	24	84.3	6.12	86.5	39.67	42	0.049	1.51
100	28	85.6	8.18	87.55	54.35	45	0.050	1.58
100	32	87	10.78	88.00	71.41	46	0.049	1.54
100	36	87.7	13.61	88.83	89.06	49	0.048	1.53
100	40	88.1	16.93	–	–	50	0.049	1.50

Table 6.2: Recall (TPR) for our AHBM, Cai et al., and Tao et al. approaches on three datasets.

## 6.2 Website fingerprinting over encrypted wireless connection

The aim of this experiment is to evaluate the accuracy of the proposed AHBM approach on Encrypted Wireless network traffics. Our evaluation of encrypted wireless connection based on side-channel information leaks where the attacker has no access to the targeted encrypted WLAN. The attacker’s goal is to infer user’s internet browsing activities. According to the literature, a similar attack could be achieved on cellular networks with relative ease [81].

To achieve such attack successfully, an attacker should have a foreknowledge about the targeted device MAC address. If not, MAC address still can be obtained via utilizing the organizationally unique identifier (OUI) lists to identify the targeted user device MAC address. Moreover, the targeted device MAC address can be also obtained through utilizing wireless Localization techniques [82].

### 6.2.1 Data collection

Dataset has been collected in an automated controlled environments that consist of three devices: Victim PC with a wireless adapter, Wireless Access Point and attacker PC powered by Kali Linux and a wireless adapter in monitor mode. Victim PC with wireless adapter equipped with a python script that utilizes selenium library to control the chrome Browser. It visited 30 websites of Alexa's top websites each 40 times in a round-robin fashion (the first sample of each website is collected, then a second sample, etc.). During website fetching process from Victim PC over the encrypted wireless connection, attacker PC monitors all the wireless traffic between victim PC and the Access Point over their dedicated channel. To ensure that our labeling process is accurate for classification process, victim PC and attacker PC were synchronized over socket communication. *airodump-ng* tool is used for capturing process while *Wireshark* was used for the filtering process. During both processes, filtering was achieved over MAC address. The data has been collected during April 2017.

### 6.2.2 One-to-one cross-validation results

As in the case of VPN dataset, we have evaluated our approach against the wireless dataset. Only features that are shared among the 75% of the training instances will be incorporated in creating the final model for each website. The good result is obtained with a feature size of 2 and window size of 2 instead of 4 as with VPN experiment because it gets better accuracy. With Windows size of 4 recall,

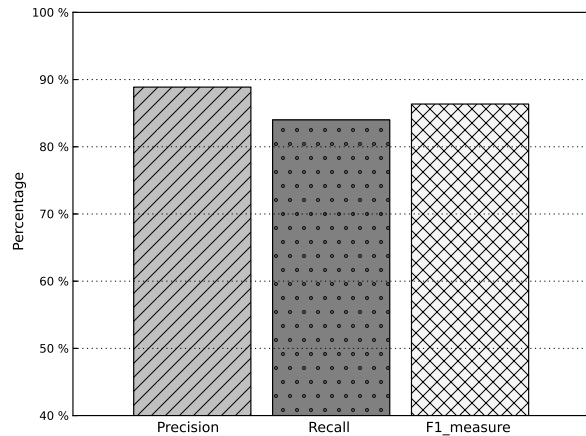


Figure 6.2: Precision, Recall, and F1 measure values for cross-validation on wireless dataset.

precision and F-measure were 0.75, 0.71 and 0.73 respectively while with window size 2 recall, precision and F-measure were 0.84, 0.89 and 0.86 as shown in figure 6.2. It is logical that window size 2 produce better results than window size 4. As mentioned in chapter 3 that smaller window size resulted with more number of features are incorporated in generating the final bloom filters. Consequently, false positive will be decreased while true positive are increased. The price paid is slightly increases in feature processing. However, the increase in processing could be negligible. The change will in the following, instead of chosen the least likely features among 4 features; it will be selected between two features.

### 6.2.3 Identifying a website within a sequence of web activity encrypted wireless connection

Again, in order to assess the efficiency of the proposed AHBM approach to identify a website visit within a long sequence of network traffic packets (without split-

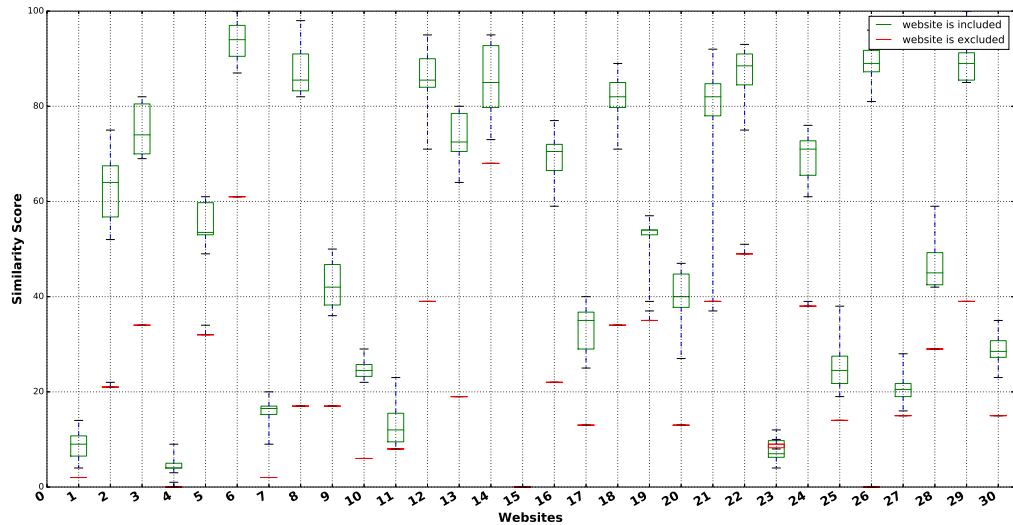


Figure 6.3: Distribution of the similarity scores for each website (1) when the website is part of the testing data and (2) when the website is removed from the testing data.

ting and using a sliding window), we used one-to-all cross-validation on wireless dataset. For each website, Figure 6.1 shows the range of all values observed in the 10-folds (dashed lines) and where 75% of the values are concentrated (closed boxes).

For the majority of websites, the similarity score between the website similarity digest and the consolidated test data is smaller when the website is not part of the test data. However, there is a clear separation between the two cases (website is included and website is excluded). This explains the high values resulted from the one-to-one cross validation demonstrated earlier.

Figure 6.4 shows the precision, recall and F1 measures for the same experiment. Compared to Figure 6.2, values are slightly lower but are still acceptable.

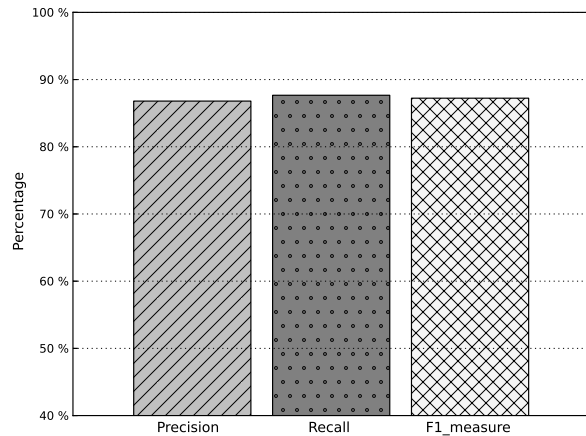


Figure 6.4: Precision, recall, and F1 measures for a modified 10-fold cross-validation (All test samples are consolidated in a single sequence).

### 6.3 Observations

Based on results acquired from website fingerprinting over encrypted wireless connection, configured with WAP2, and secure VPN client software installed on user computer device, We can see that there are similarities between VPN and wireless experiments. This similar behavior resulted from the fact that both secure VPN and wireless access point configured with encrypted protocol do nothing except that they encapsulate the exchanged data in a secure tunnel even though the both techniques are functions at different network layers. While website fingerprinting over tor browser has shown different and poor accuracy result with our approach is because of the following: firstly, tor browser is utilizing the tor network which uses anonymity protocol-Onion routing- in addition to applying three layer of encryption. Furthermore, tor browser has built in randomization technique that its purpose is to exchanged data in a randomized manner in order to hide browsing fingerprinting activities. Moreover, tor browser exchange data using cell unit.

Each unit is of size 512 bytes. Even though if the exchanged data is less than 512; the exchanged data will be padded with random data. Cells and padding are intentionally incorporated in order to wipe out fingerprinting resulted from the generated packets size. Therefore, the only things remaining for the attacker to infer the signature from is the order of the consecutive packets.

## CHAPTER 7

# NETWORK MALWARE DETECTION

Detecting malware activity through traffic analysis is attractive since it allows to cover a large number of hosts without requiring any of these hosts to install any software. Deep packet inspection (DPI) is the major approach for network malware detection which consists in checking the packet payloads in search of specific bytes. As mentioned above, DPI assumes that malware communicates through plain-text protocols (e.g. http, IRC, etc.). As malware are increasingly using encrypted protocols as well as secure tunneling, recent network malware detection approaches resorted to pattern-based traffic analysis [40, 47, 83]. Furthermore, incorporating obfuscation techniques as well as rootkit services borrowed from other malware make malware detection by anti-malware very challenging problem. According to the literature, classical anti-malware have shown a failure to keep an up-to-date database of malware signatures to defend against rapid mal-



ware generations. This escalation from the malware authors triggers researchers to develop behavioral countermeasure since it becomes impractical to generate a signature for each malware.

Malware network signatures are defined in terms of features such as number of packets, number of bytes, average bytes per packet, etc. which requires, again, to split the traffic into packet sequences corresponding to potential malware sessions.

This section details the empirical analysis of applying AHBM approach on a malware dataset. We will demonstrate how our similarity hashing approach can be incorporated in building behavioral network-based clustering system. To reach our goal, we have utilized our customized approximate hash based matching as a similarity metric. Consequently, our proposed system clusters unlabeled malware into groups based on the similarity of the groups' members. The goal of clustering malware is not to discriminate between malware families. However, it is for creating behavioral model for each group. Therefore, learned models should have the ability to identify their representative similarities and variants.

We have two scenarios to achieve this. The first option could be achieved via, starting by assuming each malware sample acts as a cluster. Then, a hierarchical clustering algorithm is applied. The second option, which we have applied is through applying K-mean clustering algorithm. One of the main issue with K-mean algorithm is deciding the number of the clusters that the algorithm should start with. To solve this issue, we resort to using AVCLASS tool [84] that utilize virus total to label binary malware samples. even though, according to

the literature, anti-malware labeling is not accurate since there is no labeling standard that majority of anti-malware agree on. Even though our used tool for labeling tries to use the majority name resulted from virus-total. However, our clustering uses this labels just to decide the number of initial clustering as an input to k-mean clustering algorithm. One more benefit of using major anti-malware labeling before indulging in clustering procedure is that, we can see during the clustering operation how the family members grouped together. For example, we have seen the two biggest families- trymedia and installcore respectively- spilted into several clusters. On the other hand, two different malware families-imaili and zenzue- are grouped together into one cluster with all their instances. Each group is represented by a centroid. The Centroid is a malware sample instance that has the highest similarity connection relationship among all the other members of its related group.

## 7.1 Data collection

The list of malware binaries used in the data collection is retrieved from virusshare<sup>1</sup> malware repository. The list consists of 16000 malware binaries posted on the repository on September 2016. To execute the binaries and capture their traffic, we used a sandbox-based isolation program, namely, sandboxie<sup>2</sup> running on a Windows XP SP3 32-bits virtual machine. As an initial filtering step, We wrote a script to automatically check the PE header of the binaries and filter out all

---

<sup>1</sup>[www.virusshare.com](http://www.virusshare.com)

<sup>2</sup>[www.sandboxie.com](http://www.sandboxie.com)

non-windows, non-32 bit, and corrupted header binaries. Then, each valid binary is executed once through the sandbox while capturing its network communication. Each execution lasts 2 minutes. As expected, a significant portion of valid malware did not yield network activity. The list of possible reasons include:

- malware using anti-VM
- malware using anti-sandbox
- malware using anti-debugging
- Command and Control (C2) server is down.

Malware that exchanged less than 50 packets is filtered out. Among the 16000 initial malware binary retrieved from virusshare, only 1050 passed the two filtering steps. Among this 1050 binary, we chose the first 1000 for the data collection. Malware data collection consists of executing the filtered 1000 binaries, 10 times each, in a round-robin fashion through the sandbox on the same Windows XP SP3 virtual machine. Each execution lasts 2 minutes. From the 1000 malware, we kept only the malware that yields 10 valid samples. That is, if a malware has at least one sample (out of 10) with less than 50 packets, it is dropped. After this last filtering, we ended up with 587 malware, with 10 valid samples each.

Malware execution has been done automatically via utilizing the sandboxie command line for either running the malicious program as well as halting it. Each malware allowed two minutes to be executed. During malware execution, tshark-Wireshark command line- used to sniff the network generated by the executed

malware and filter out all unrelated network traffic such as DNS, NBNS. Furthermore, background services are implicitly stopped, that automatically initiate internet connection between now such as windows update, google service update, multimedia sharing, Firefox browser update and error reporting, are stopped. Moreover, before running the data collection process, we have made our virtualized environment powered on for about one hour while making sure nothing is running on except Wireshark sniffer on it to inspect if there were some unobserved program or service that still use the Internet connection without our knowledge. After dataset collection process finished, all malware samples that generate less than 50 packets has been deleted for each related instance. Consequently, only 587 samples remained in our dataset. Then, we use AVCLASS tool [84] that utilize virus total to label binary malware samples according to their family name. Malware that can not be recognized by the labeling tool unrecognized label is assigned to them. At the end, we got 27 families. The biggest family consist of 92 samples whereas the small family has two.

## 7.2 Clustering procedure

Applying cross-validation on the malware dataset the same way as VPN and Tor datasets produced very low accuracy results. The main problem is that considering each binary as a separate class is not appropriate for malware traffic analysis because several binaries correspond to the same malware due to obfuscation<sup>3</sup>.

---

<sup>3</sup>Malware obfuscation consists of a set of technique to change the shape of binaries in order to bypass antivirus detection. This includes encryption, packing, polymorphism, metamorphism,

To overcome this issue, malware binaries are clustered into families sharing the same features. Malware clustering is a commonly used technique to deal with the redundancy of malware binaries [50, 10].

Typical malware clustering approaches rely either on malware executable bytes [85, 86] malware system call traces [87, 88], or non-encrypted network activity [50, 10, 8]. The malware clustering we propose in this section relies only on packets size, order, and direction of the malware network traffic. This makes the clustering applicable for any malware having network activity.

Our data set consist of 587 samples. Each sample has 10 instances. To test our proposed approach on behavior malware clustering based on network traffic, we have applied five fold cross-validation. In each fold step, we take eight instances from each malware sample in our dataset to participate in the clustering process. The remaining two instances from each malware sample are kept to be used lately as a testing dataset against the generated final signature of each final cluster.

Clustering Procedure goes through the following steps: firstly, fold one training data are grouped based on labels using AVCLASS tool, which assign a label for each malware via utilizing virus-total reports as mentioned earlier. However, this labeling is not mandatory as We will explain later. Then, a distance is calculated between each pair in every group using our customized approximate hashing (AHBM). Consequently, a malware instance that has the highest similarity connections relationship among the group members and satisfies the similarity threshold will be nominated as the centroid of that group. If it happens that

---

etc.

more than three instances within a group have not matched with threshold similarity(out layers); They will be moved to their new cluster and the same centroid calculation will be applied to them in a recursive way. After this step, each cluster will have its own centroid. Then, all centroid will be compared against each other. If two centroids happen to mach the similarity threshold; the cluster with small members will be merged with with largest cluster. The remaining clusters that their centroid have no similarity, each instance of that cluster will be compared with every other clusters' centroids. If it happens that a higher similarity value has been achieved with other centroids other than its current one, then this instance will move to that centroid cluster. After every instance has got its chance to see the best cluster to stay in, centroid calculation algorithm will calculate again to fit the new movement and merging process. Centroid calculation, merging and movement process will be repeated several time until no need for further merging or movement steps any more. Then, all centroids similarity hashes digest using AHBM will be selected as the final behavior model for each group. To see the effect of the selected centroid models, each centroid will be tested against its related test data samples and tested also against two normal data sets that have been used as a benchmark for the intrusion detection systems [89]. After that fold two, fold three. Fold four and fold five go through the same steps that fold one went through but with their training and test data. The whole clustering procedure can summarize in the following pseudo code.

---

**Algorithm 1** Malware clustering using AHBM similarity score

---

**INPUT:**  $\mathcal{S} = \{\text{the set of all malware traffic samples } mts\}$

**INPUT:**  $th_{clst}$ : a similarity score threshold

**OUTPUT:**  $\mathcal{C}$ : a set of malware clusters

Divide  $\mathcal{S}$  randomly into malware clusters  $\mathcal{C}$  (e.g. clustering  $\mathcal{S}$  based on virus-total family names)

**repeat**

  //centroid Calculation

**for** each cluster  $C_k$  in  $\mathcal{C}$  **do**

*centroid\_calculation*( $C_k$ )

**end for**

  //Merging

$\mathcal{C}_{temp} = \mathcal{C}$

**for** each pair of clusters  $C_i$  and  $C_j$  in  $\mathcal{C}_{temp}$  **do**

**if**  $SC(ctroid_i, ctroid_j) \geq th_{clst}$  **then**

      Merge  $C_i$  and  $C_j$  into a new cluster

      Remove  $C_i$  and  $C_j$  from  $\mathcal{C}_{temp}$

**end if**

**end for**

  //Migrating outliers

**for** each cluster  $C_k$  still in  $\mathcal{C}_{temp}$  **do**

**for** each instance  $mts$  in  $C_k$  **do**

      Find the closest cluster to  $mts$

$closestC \leftarrow \arg \max_{i \dots |C|} SC(mts, ctroid_i)$

      Migrate  $mts$  to  $closestC$

**end for**

**end for**

**until** no more splitting and no more merging

**function** *centroid\_calculation*( $C_k$ )

  //Select a centroid ( $ctroid_k$ ) for  $C_k$

$ctroid_k \leftarrow \arg \max_{i=1 \dots |C_k|} \sum_{j=1 \dots |C_k|} SC(mts_i^k, mts_j^k)$

  //Look for outliers in cluster  $C_k$

$outliers_K \leftarrow \{mts_i^k | SC(mts_i^k, ctroid_k) < th_{clst}\}$

  //Create a new cluster and split recursively

**if**  $|outliers_k| > 3$  **then**

    Create a new cluster  $C_n$  and add it to  $\mathcal{C}$

*Split\_recursively*( $C_n$ )

**end if**

**end function**

---

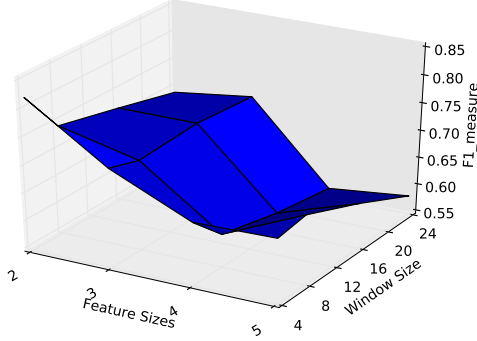


Figure 7.1: Accuracy in terms of feature size and window size.

### 7.3 Chosen parameters

several parameters participated in our proposed system. First of all anti-malware labeling, as We have mentioned earlier that it is not mandatory to resort to it because we can start with arbitrary as if we start to used hierarchical clustering algorithm. We argue with this because of the following: first of all the purpose of the proposed system is to create a behavioral network clustering models for unlabeled malware group that has similar network behavior to be used as their signature. Secondly, merging and movement operation have the capability to end up with almost similar group resulted from k-mean. Furthermore, most of the labeling approaches are taken from either strings within binary code, system behavior; its rarely a name has been generated network behaviour. Last but not the least, several malware today are composed of more than one malware type or different malware types are using the same network payloads delivery mechanisms or two malware variant use different network payload delivery; that is why we



notice that two or more different families have similar network behavior. What is matter practitioner's more is the false positive. They suffer from false positive especially with using intrusion detection systems.

Score similarity threshold is the second parameter participated in malware clustering procedure. Score similarity threshold means the minimum similarity score needed so that we can tell that the two compared instances are similar. In our evaluation we have used similarity score ( $t=20$ ) based on experimental experiments. We have tested it with three different values  $t=[15,20,30]$ . Larger the similarity score threshold value used will result in a large number of generated clusters. Consequently, more centroids will be generated with high False negative. This typically happens with large malware families that include a big number of instances. Consequently, it will be hard to represent all the malware with one or two centroids. For example, in our experiment, installCore and trymedia families, which are the largest malware families in our dataset, have been represented with around six clusters each.

As with VPN experiment mentioned in chapter 3, feature size (how many packets are considered at a time) and the window size (how many features are considered in the winnowing based selection). In order to empirically choose the optimal values for those parameters, Clustering Procedure has been applied several times using a different combination for both features size and widow size. Good parameter values (feature size and window size) should maximize the F1-measure calculated for each five-fold cross validation clustering experiment. Figure 7.1

shows F1-measure resulted from the different combinations of feature size and window size. Clearly, it can be seen that the best result acquired resulted from using feature size equals to 2 and window size equals to 4. This combination is similar to the best values chosen at VPN experiment demonstrated at chapter 3.

## 7.4 Evaluation

As long as the purpose of our contribution is to test the applicability of our customized approximate hashing as a similarity metric to cluster the unlabeled malware into groups. So that we can generate a behavioral signature for each group as it has been demonstrated clearly in the previous section. Therefore, we will evaluate our clustering procedure using two measures: Conciseness and Compactness. The purpose of using these two accuracy measures is to test how good our proposed similarity metric at clustering malware into groups as well as how are members of each generated cluster are close to each other respectively.

### 7.4.1 Compactness

Compactness Basically measures the relationship connectivity among resulted clusters' members. A good generated cluster its members are very close to each other. In our context, a good cluster exists when all cluster's members have high similarity among each other. To check similarity connectivity among cluster's members in our case we applied the following formula: the number of relations exists that satisfy the assigned similarity score thresholded over the maximum num-

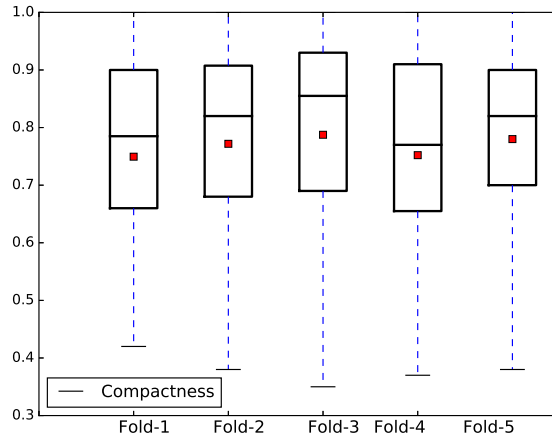


Figure 7.2: Clusters Compactness.

ber of the all possible relations supposed to be existed. Where the maximum number of the all possible relations calculated via following formula:

$$compactness = A / (n * (n - 1) / 2) \quad (7.1)$$

where n is the number of instances in cluster, A represents the number of relations exists that satisfy the assigned similarity score threshed.

To evaluate how good our generated clusters using our proposed approach, compactness has been calculated for each final resulted cluster using the previous formula. Figure 7.2 shows the compactness values calculated for every cluster that have been generated during the five cross-fold validation with confidence interval equals 0.99. From the figure, we see that almost all the clusters members have high similarity value matching the similarity score threshold assigned at the start of clustering procedure ( $t=20$ ). Furthermore, from quartile figure, we can see that for each cluster around half its members have full similarity relationship

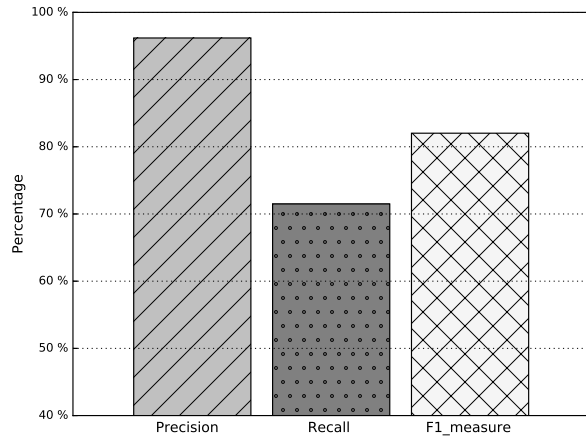


Figure 7.3: Precision, Recall, and F1 measure values for cross- validation on Malware dataset.

connection with all their siblings.

## 7.4.2 Conciseness

Conciseness demonstrates the capability of our proposed similarity metric as a function to cluster malware into groups that share similar network behavior. Furthermore, it will test how good our final generated centroids' approximate hash bloom filters are a good representative to there related group members. To fulfill this objective, we will test our final generated centroid's approximate hash bloom filters resulted from each fold mentioned in the cluster procedure mentioned earlier against their related test dataset. For each fold, we calculate the Recall, Precision and F1-measure. Through these values, we can see how good our clustering process as depicted in figure 7.3.

Figure 7.3 shows the results of malware behavioral clustering procedure which are 96% for the precision, 71% for the recall, and 82% for the F1-measure. As we

can see that Precision is very high compared to the Recall which means that, whenever we faced a similar malware test instance point, we have a high confidence that our centroids approximate hash based filters will be able to catch it. However, regarding the recall's low degrade compare to the precision is because of that, a test malware instance might be get caught by another centroid's signature other than its related one or might be tested against a different centroids other than its related centroid. This situations might happen when some malware instances spread into different clusters. Therefore, its related test instances will be tested against two different centroids because our implementation allows each centroids to have a list of all its member's test data instances to be tested later against them. Another reason behind is that, a malware instances spread into different groups during clustering procedure is that, as we mentioned in the dataset collection earlier that, each malware samples has been executed ten times at different time intervals. Therefore, a malware might decide to change his networking behaviour at different time it has been executed in order to avoid classical anti-malware or firewall rules.

### **7.4.3 Cluster centroids against single large dataset**

In these evaluation, we check the capability of our approach to check the existence of a needle in a haystack instead of comparing an apple with an apple as it usually happened with other proposed solutions in this field. We accomplish these via checking the existence of a cluster's centroid or any malware instance belonging

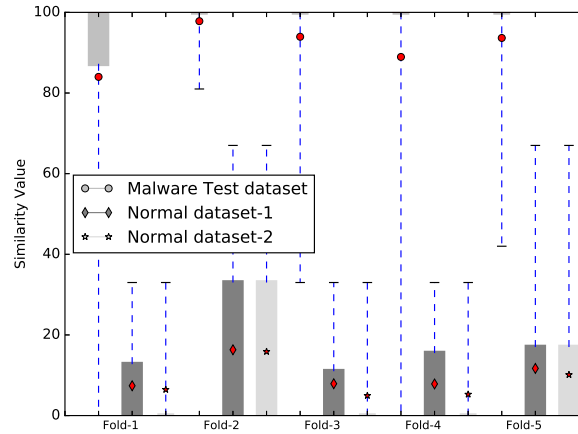


Figure 7.4: Centroids Signature VS three single big test datasets.

to its group within one thousand of malware traffic gathered into one single test file. Basically, the test dataset used has been taken from cross fold test data used during the clustering process. The difference between this experiment and the ordinary cross-fold validation experiment performed earlier is that, during cross-fold validation each cluster's centroid checked against each test malware instance in one-to-one comparison fashion while this experiment is similar to finding a needle in a haystack as mentioned earlier. Every cluster's centroid is tested against the all its fold malware instances that are combined into a single file. Therefore, we do not know where the centroids test data instances exists. It may be at the beginning, at the middle or scattered through the combined single test data file. During the same time, each cluster's centroid is also tested against two normal traffic datasets borrowed from the intrusion detection system test data benchmark[89]. The first dataset size is 16GB while the second dataset size is 4GB. The result of this experiment is demonstrated in fig 7.4. We can see from

the figure that similarity values of malware centroids against the malicious test data vary between 85 and 100 similarity score. Furthermore, as the quartiles show that, more than half of the similarity values hit 100 similarity with confidence interval equals 0.99. While centroids similarity values against the two normal dataset vary between 0 and 33 similarity score. In the same way, half of similarity scores between the malware clusters' centroids and the two normal traffic dataset is zero with confidence interval equals 0.99. These values give us a high confidence that our proposed approximate hashing used as a similarity metric as well as the way to represent the final clusters' centroid by hash bloomed filter is an effective method for clustering malware in groups in order to create a behavioral network signature for each group.

## CHAPTER 8

# CONCLUSION

### 8.1 Summary of contribution

Networking traffic can be easily captured. Captured Traffic can be either analyzed at run-time or stored for later inspection. Privacy enhancing technologies, that applies extensive encryption, aim at protecting clients from malicious software, that has been implemented intentionally to intrude internet users' privacy. Several works, that utilize different machine learning algorithms, have shown that VPN product is vulnerable to website fingerprint attack. Highest contribution achieved 94%. However, their proposed solution are calculation intensive. Tens of hours are needed to train a classifier on only 100 websites. Furthermore, currently available contributions can not handle needle in a haystack scenario without extensive pre-processing. Therefore, scalability is a major issue for current works.

The main Contribution of this thesis is to tackle the scalability faced earlier proposed solutions. It is inspired by similarity digest hashing(ABHM), that it has



been incorporated recently in digital forensics. We have customized the core of the similarity digest approach to be tailored with network traffic analysis. My proposed approach has shown superiority over the current website fingerprint attacks. It generates the final models in minutes instead of hours with scalability merit as well. It achieved 87% precision rate over VPN and 0.89 over an encrypted wireless connection. This accuracy have been achieved while taking a fixed threshold values. Selected thresholds are learned for each website while testing each website's generated model against a large test data when that website is not included within that test data. Consequently, our final models consist of both sdhash bloom filters, that generated by our customized tool, and the learned threshold.

In addition, we have also shown that my proposed approach has a very promising application in designing behavioral malware traffic detection system. Customized similarity digest has been tailored to be used as a distance metric. My implemented approach was able to cluster 587 malware into (17-19) clusters- resulted from five different runs. At the end, the centroids generated for each group are selected as a final malware signatures. Last but not the least, we have also shown that our generated malware clusters' centroid signatures were able to differentiate between normal and malicious traffic with a very small false positive rate. We can control false positive until approaches zero via adjusting selected thresholds, that are companion to our generated models as we have demonstrated in the previous chapter. Empirical experiments showed that there are a clear sep-

aration when clusters' centroids signature are tested against both normal traffic and malware traffic testing data.

To summarize, experimental results show a very promising results for VPN, malware traffic, encrypted wireless traffic, but low results for Tor traffic.

## 8.2 Threat to validity

Regarding incorporating the tailored similarity hashing approach into network traffic analysis, if a new formula used for calculating the similarity score instead of the one mentioned in section 4.2.4; a major difference in reported results may appear especially for website fingerprinting over tor browser with which we have got a poor result compared to the other experiments.

On the other hand, to detect malware's network traffic, our proposed solution has weakness which commonly shared with dynamic analysis solutions. First of all, our collected dataset is collected from the execution of malware under sandbox environment. Therefore malware that employs anti-sand-boxing techniques will evade disclosing their networking behavior. Consequently, they will not contribute in clustering procedure. To remedy this limitation, instances that did not generate network sample should be given another chance to be executed in real physical environment. The second limitation is due to the time interval allowed for each malware to be executed. In our experiment, each malware instance allowed only two minutes. Therefore, some malware might not expose a distinctive behavior during this time period limit waiting for a certain activity or related issues as

mentioned in [90]. In the same way, similar limitation resulted from malware samples that relies on user input to perform their actions as demonstrated in [91]. Another limitation is related to the environments that we used for executing malware samples, which is over windows XP SP3. According to Moser et al. [92] running malware in multiple environment in parallel with other computer programs such as anti-malware or with different network infrastructure might yield different behavior.

Last but not the least, any traffic manipulation such as Morphing-Packet size, HTTPoS, Traffic padding etc. might reduce the accuracy our approach. For example, Tor browser applies packets padding. As a result, we have got a poor result compared to the other experiments.

### **8.3 Future work**

A possible area of work is to keep optimizing the parameters of the proposed approach to gain more accuracy while keeping the same scalability levels.

Another application of our approach could be classifying different internet application(p2p, video streaming, etc.) that exchange packet over the internet network.

One more potential application of our approach is about secured wireless network. We intend to further classify smart phones application (Facebook, twitter, email-client, ..etc.) either over Android and iPhone. According the literature, classifying Facebook application is easier than fingerprinting its related website

visit through internet browsers.

# REFERENCES

- [1] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, “A critical evaluation of website fingerprinting attacks,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 263–274.
- [2] D. Herrmann, R. Wendolsky, and H. Federrath, “Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier,” in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 31–42.
- [3] T. Wang and I. Goldberg, “Improved website fingerprinting on tor,” in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 2013, pp. 201–212.
- [4] A. Dainotti, A. Pescapé, and K. C. Claffy, “Issues and future directions in traffic classification,” *Network, IEEE*, vol. 26, no. 1, pp. 35–40, 2012.

- [5] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, “Prudent practices for designing malware experiments: Status quo and outlook,” in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 65–79.
- [6] J. Oberheide, E. Cooke, and F. Jahanian, “Clouday: N-version antivirus in the network cloud.” in *USENIX Security Symposium*, 2008, pp. 91–106.
- [7] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007, pp. 421–430.
- [8] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, “Automated classification and analysis of internet malware,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 178–197.
- [9] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, “Learning and classification of malware behavior,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 108–125.
- [10] J. Jang, D. Brumley, and S. Venkataraman, “Bitshred: feature hashing malware for scalable triage and semantic analysis,” in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 309–320.

- [11] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, “Panorama: capturing system-wide information flow for malware detection and analysis,” in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 116–127.
- [12] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer, “Behavior-based spyware detection.” in *Usenix Security*, vol. 6, 2006.
- [13] T. T. Nguyen and G. Armitage, “A survey of techniques for internet traffic classification using machine learning,” *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.
- [14] A. Este, F. Gringoli, and L. Salgarelli, “On-line svm traffic classification,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE, 2011, pp. 1778–1783.
- [15] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, and D. Rossi, “Experiences of internet traffic monitoring with tstat,” *Network, IEEE*, vol. 25, no. 3, pp. 8–14, 2011.
- [16] A. Finamore, M. Mellia, M. Meo, and D. Rossi, “Kiss: Stochastic packet inspection classifier for udp traffic,” *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1505–1515, 2010.
- [17] W. de Donato, A. Pescapé, and A. Dainotti, “Traffic identification engine: an open platform for traffic classification,” *Network, IEEE*, vol. 28, no. 2, pp. 56–64, 2014.

- [18] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, “Touching from a distance: Website fingerprinting attacks and defenses,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2012, pp. 605–616.
- [19] V. Roussev, “Data fingerprinting with similarity digests,” in *Advances in digital forensics vi*. Springer, 2010, pp. 207–226.
- [20] —, “Building a better similarity trap with statistically improbable features,” in *System Sciences, 2009. HICSS’09. 42nd Hawaii International Conference on*. IEEE, 2009, pp. 1–10.
- [21] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, “Is p2p dying or just hiding?[p2p traffic measurement],” in *Global Telecommunications Conference, 2004. GLOBECOM’04. IEEE*, vol. 3. IEEE, 2004, pp. 1532–1538.
- [22] A. M. M. K. K. O. M. S. L. E. A. M. W. E. Joe Touch, Eliot Lear and A. Zimmermann, “Service name and transport protocol port number registry,” IANA, 2016.
- [23] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, “Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification,” in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM, 2004, pp. 135–148.
- [24] “L7 filter,” ”<http://l7-filter.sourceforge.net>”, Clear Foundation, 2009.



- [25] “Ndpi,” ”<http://www.ntop.org/products/deep-packet-inspection/ndpi>”, NTOP, 2016.
- [26] “ntopng,” ”<http://www.ntop.org/products/traffic-analysis/ntop>”, NTOP, 2016.
- [27] S. Alcock and R. Nelson, “Libprotoident: traffic classification using lightweight packet inspection,” *WAND Network Research Group, Tech. Rep*, 2012.
- [28] “Cloudshield (lookingglass,” ”<https://www.lookingglasscyber.com>”, LookingGlass, 2016.
- [29] “Netflow traffic analyzer,” ”<http://www.solarwinds.com/netflow-traffic-analyzer>”, Solarwinds, 2016.
- [30] “Network based application recognition,” ”<http://www.cisco.com>”, CISCO, 2016.
- [31] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, “Acas: automated construction of application signatures,” in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. ACM, 2005, pp. 197–202.
- [32] B.-C. Park, Y. J. Won, M.-S. Kim, and J. W. Hong, “Towards automated application signature generation for traffic identification,” in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*. IEEE, 2008, pp. 160–167.

- [33] Y. Wang, X. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo, “A semantics aware approach to automated reverse engineering unknown protocols,” in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 2012, pp. 1–10.
- [34] Y. Wang, Y. Xiang, W. Zhou, and S. Yu, “Generating regular expression signatures for network traffic classification in trusted network management,” *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 992–1000, 2012.
- [35] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, “Towards self adaptive network traffic classification,” *Computer Communications*, vol. 56, pp. 35–46, 2015.
- [36] F. J., “Machine learning and intrusion detection: current and future directions,” in *In Proceedings of the 17th National Computer Security Conference*, October 1994.
- [37] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, “Network traffic classification using correlation information,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 104–117, 2013.
- [38] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, “Robust network traffic classification,” *IEEE/ACM transactions on networking*, vol. 23, no. 4, pp. 1257–1270, 2015.

- [39] F. Gringoli, L. Nava, A. Este, and L. Salgarelli, “Mtclass: enabling statistical traffic classification of multi-gigabit aggregates on inexpensive hardware,” in *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*. IEEE, 2012, pp. 450–455.
- [40] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer, “Using machine learning techniques to identify botnet traffic,” in *In 2nd IEEE LCN Workshop on Network Security*, 2006, pp. 967–974.
- [41] A. Boukhtouta, N.-E. Lakhdari, S. A. Mokhov, and M. Debbabi, “Towards fingerprinting malicious traffic,” *Procedia Computer Science*, vol. 19, pp. 548–555, 2013.
- [42] R. Alshammari and A. N. Zincir-Heywood, “Investigating two different approaches for encrypted traffic classification,” in *Privacy, Security and Trust, 2008. PST’08. Sixth Annual Conference on*. IEEE, 2008, pp. 156–166.
- [43] —, “Machine learning based encrypted traffic classification: Identifying ssh and skype.” *CISDA*, vol. 9, pp. 289–296, 2009.
- [44] T. Nelms, R. Perdisci, and M. Ahamad, “Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates,” in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 589–604.
- [45] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in

- Proceedings of the 17th conference on Security symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154.
- [46] G. Gu, J. Zhang, and W. Lee, “Botsniffer: Detecting botnet command and control channels in network traffic.” in *NDSS*. The Internet Society, 2008.
- [47] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, “Botfinder: Finding bots in network traffic without deep packet inspection,” in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 349–360.
- [48] C. Rossow and C. J. Dietrich, “Provex: Detecting botnets with encrypted command and control channels,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2013, pp. 21–40.
- [49] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “Bothunter: detecting malware infection through ids-driven dialog correlation,” in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16.
- [50] R. Perdisci, W. Lee, and N. Feamster, “Behavioral clustering of http-based malware and signature generation using malicious network traces.” in *NSDI*, 2010, pp. 391–404.
- [51] X. Wang, W. Qiu, and R. H. Zamar, “Clues: A non-parametric clustering method based on local shrinking,” *Computational Statistics & Data Analysis*, vol. 52, no. 1, pp. 286–298, 2007.

- [52] D. Pelleg, A. W. Moore *et al.*, “X-means: Extending k-means with efficient estimation of the number of clusters.” in *ICML*, vol. 1, 2000.
- [53] J. Newsome, B. Karp, and D. Song, “Polygraph: Automatically generating signatures for polymorphic worms,” in *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005, pp. 226–241.
- [54] M. Z. Rafique and J. Caballero, “Firma: Malware clustering and network signature generation with mixed network behaviors,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013, pp. 144–163.
- [55] R. Perdisci, D. Ariu, and G. Giacinto, “Scalable fine-grained behavioral clustering of http-based malware,” *Computer Networks*, vol. 57, no. 2, pp. 487–500, 2013.
- [56] R. Dingledine, N. Mathewson, and P. Syverson, “Tor : the second-generation onion router,” in *Proceedings of the 13th Usenix Security Symposium*, August 2004.
- [57] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, “Website fingerprinting in onion routing based anonymization networks,” in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, ser. WPES ’11. New York, NY, USA: ACM, 2011, pp. 103–114.
- [58] M. Liberatore and B. N. Levine, “Inferring the source of encrypted http connections,” in *Proceedings of the 13th ACM conference on Computer and*

- communications security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 255–263.
- [59] V. Roussev, “Hashing and data fingerprinting in digital forensics,” *Computing in Science and Engineering*, vol. 7, no. 2, pp. 49–55, 2009.
- [60] M. O. Rabin *et al.*, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.
- [61] R. M. Karp and M. O. Rabin, “Efficient randomized pattern-matching algorithms,” *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.
- [62] U. Manber *et al.*, “Finding similar files in a large file system.” in *Usenix Winter*, vol. 94, 1994, pp. 1–10.
- [63] S. Brin, J. Davis, and H. Garcia-Molina, “Copy detection mechanisms for digital documents,” in *ACM SIGMOD Record*, vol. 24, no. 2. ACM, 1995, pp. 398–409.
- [64] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, “Syntactic clustering of the web,” *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1157–1166, 1997.
- [65] K. Shanmugasundaram, H. Brönnimann, and N. Memon, “Payload attribution via hierarchical bloom filters,” in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 31–41.

- [66] C. Y. Cho, S. Y. Lee, C. P. Tan, and Y. T. Tan, "Network forensics on packet fingerprints," in *IFIP International Information Security Conference*. Springer, 2006, pp. 401–412.
- [67] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection." in *USENIX security symposium*, vol. 286. San Diego, CA, 2004.
- [68] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.
- [69] V. Roussev, "An evaluation of forensic similarity hashes," *digital investigation*, vol. 8, pp. S34–S41, 2011.
- [70] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 76–85.
- [71] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 21–36, 2016.
- [72] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
- [73] V. Roussev, G. G. Richard, and L. Marziale, "Multi-resolution similarity hashing," *digital investigation*, vol. 4, pp. 105–113, 2007.

- [74] R. Stanton, “Securing vpns: Comparing ssl and ipsec,” *Computer Fraud & Security*, vol. 2005, no. 9, pp. 17–19, 2005.
- [75] S. Feghhi and D. J. Leith, “A web traffic analysis attack using only timing information,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1747–1759, Aug 2016.
- [76] Y. Shi and S. Biswas, “Detecting tunneled video streams using traffic analysis,” in *Communication Systems and Networks (COMSNETS), 2015 7th International Conference on*. IEEE, 2015, pp. 1–8.
- [77] C. V. Wright, L. Ballard, F. Monroe, and G. M. Masson, “Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob?” in *USENIX Security*, vol. 3, no. 3.6, 2007, p. 3.
- [78] J. Hayes and G. Danezis, “Website fingerprinting at scale,” *University College of London (UCL), number: Technical report*, 2015.
- [79] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, “Website fingerprinting at internet scale,” in *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2016.
- [80] M. Perry, “A critique of website traffic fingerprinting attacks,” [”https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks”](https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks), The Tor Blog, 2013.
- [81] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, “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. ACM, 2013, pp. 7–12.

- [82] R. W. Ouyang, A. K.-S. Wong, and C.-T. Lea, “Received signal strength-based wireless localization via semidefinite programming: Noncooperative and cooperative schemes,” *IEEE Transactions on Vehicular Technology*, vol. 59, no. 3, pp. 1307–1318, 2010.
- [83] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, “Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection.” in *USENIX Security Symposium*, vol. 5, no. 2, 2008, pp. 139–154.
- [84] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, “Avclass: A tool for massive malware labeling,” in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.
- [85] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, “Mutantx-s: Scalable malware clustering based on static features.” in *USENIX Annual Technical Conference*, 2013, pp. 187–198.
- [86] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, “Scalable, behavior-based malware clustering.” in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.
- [87] M. Z. Rafique and J. Caballero, “Firma: Malware clustering and network signature generation with mixed network behaviors,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2013, pp. 144–163.

- [88] I. Gurrutxaga, O. Arbelaitz, J. M. Perez, J. Muguerza, J. I. Martin, and I. Perona, “Evaluation of malware clustering based on its dynamic behaviour,” in *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*. Australian Computer Society, Inc., 2008, pp. 163–170.
- [89] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.
- [90] J. R. Crandall, G. Wassermann, D. A. de Oliveira, Z. Su, S. F. Wu, and F. T. Chong, “Temporal search: Detecting hidden malware timebombs with virtual machines,” in *ACM Sigplan Notices*, vol. 41, no. 11. ACM, 2006, pp. 25–36.
- [91] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, “A crawler-based study of spyware in the web.” in *NDSS*, vol. 1, 2006, p. 2.
- [92] A. Moser, C. Kruegel, and E. Kirda, “Exploring multiple execution paths for malware analysis,” in *Security and Privacy, 2007. SP’07. IEEE Symposium on*. IEEE, 2007, pp. 231–245.

# Vitae

- Name: Abdullah Mohammed Qasem
- Nationality: Yemeni
- Date of Birth: 1/1/1987
- Email: *abdullahqasem87@gmail.com*
- Permenant Address: Yemen-Taiz
- Education: King Fahd University of Petroleum and Minerals, Saudi Arabia  
(2014-2017).  
M.Sc. in Information and Computer Science.  
Major: Security and Information Assurance.