

GENERATING AND ANALYZING ENCRYPTED TRAFFIC OF
INSTANT MESSAGING APPLICATIONS: A COMPREHENSIVE
FRAMEWORK

by

Zolboo Erdenebaatar

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2023

© Copyright by Zolboo Erdenebaatar, 2023

Dedicated to my family

Table of Contents

List of Tables	vi
List of Figures	viii
Abstract	x
Acknowledgements	xii
Chapter 1 Introduction	1
Chapter 2 Related Work	5
2.1 Analyzing Smartphone Traffic	5
2.2 Analyzing User Actions	7
2.3 Analyzing Instant Messaging Traffic	8
2.4 Summary	9
Chapter 3 Methodology	10
3.1 IMAs employed	10
3.1.1 Whatsapp	10
3.1.2 Messenger	12
3.1.3 Telegram	13
3.1.4 Teams	13
3.1.5 Discord	13
3.1.6 Signal	14
3.2 Describing IMA Flows	14
3.3 Realistic Data Generation	16
3.3.1 Equipment Setup	16
3.3.2 Emulating Inputs	17
3.3.3 Two-way traffic generation	18
3.3.4 Realistic Dialogue	18
3.3.5 Two Types of Text Conversations	20
3.3.6 Opening and Closing IMAs	22
3.3.7 Running the complete emulation	23
3.3.8 Collecting IMA traffic as PCAP files	23
3.3.9 Collecting non-IMA traffic as PCAP files	25

3.3.10	Isolating IMA Traffic	25
3.3.11	Limitations	27
3.4	Classification Tasks	28
3.5	Extracting Features	29
3.6	Preprocessing	30
3.7	Varying Feature Sets	31
3.8	Building Machine Learning Classifiers	32
3.8.1	Nearest Neighbors	33
3.8.2	Linear SVM	33
3.8.3	C4.5 Decision Tree	34
3.8.4	Random Forest	34
3.8.5	Neural Network	35
3.8.6	Naive Bayes	35
3.8.7	Logistic Regression	36
3.8.8	Gradient Boost	36
3.8.9	Training and Evaluating Models	37
3.9	Feature Selection Analysis	38
3.10	Summary	39
Chapter 4	Evaluations and Results	41
4.1	Dataset Generation	41
4.2	IMA Traffic Protocol	42
4.2.1	Ports and protocols used	44
4.2.2	Flows generated by the IMAs	46
4.3	Building Machine Learning Classifiers	46
4.3.1	Distinguishing IMA traffic from non-IMA traffic	47
4.3.2	Distinguishing between different IMAs	47
4.3.3	Identifying IMA Traffic From Ground Truth	48
4.4	Feature Selection	48
4.4.1	All Features	48
4.4.2	Statistical Features	49
4.4.3	Packet Size/Timing Features	49
4.5	Summary	49
Chapter 5	Conclusion And Future Work	58

Bibliography	61
Appendix A Appendix	65
A.1 Features extracted by <i>Tranalyzer2</i>	65
A.2 Improving my packet size/timing features classifier	69

List of Tables

Table 3.1	IMAs employed in this thesis.	10
Table 4.1	Summary of the traffic generated by each IMA application. . .	41
Table 4.2	Summary of the traffic generated by each non-IMA application (Web-browsing, e-mail, and video streaming).	42
Table 4.3	The ports and protocols essential for IMAs. \emptyset indicates that the IMA becomes non-functional after I block the specified port, while \checkmark indicates that the IMA remains functional.	43
Table 4.4	Distinguishing IMAs from non-IMAs using all features.	50
Table 4.5	Distinguishing between different IMAs using all features.	51
Table 4.6	Distinguishing IMAs from non-IMAs using strictly statistical features.	51
Table 4.7	Distinguishing between different IMAs using only statistical features.	51
Table 4.8	Distinguishing IMA vs Non-IMA using only packet size and timing features.	52
Table 4.9	Distinguishing between different IMAs using only packet size and timing features.	52
Table 4.10	Identifying IMA traffic using all features.	52
Table 4.11	Identifying IMA traffic using only statistical features.	53
Table 4.12	Identifying IMA traffic using only packet sizing and timing features.	53
Table 4.13	Statistical features selected by Mutual Information to build a classifier to distinguish IMA vs. Non-IMA, achieving the baseline performance as defined in Section 3.9.	55
Table 4.14	Statistical features selected by Mutual Information to build a classifier to distinguish between IMAs, achieving the baseline performance as defined in Section 3.9.	55
Table 4.15	Packet size/timing features selected by Mutual Information to build a classifier to distinguish IMA vs. Non-IMA, achieving the baseline performance as defined in Section 3.9.	56

Table 4.16	Packet size/timing features selected by Mutual Information to build a classifier to distinguish between IMAs, achieving the baseline performance as defined in Section 3.9.	57
A.1	The list of features extracted by Tranalyzer2	69
Table A.2	Classifying between IMAs using packet/timing features.	71
Table A.3	Classifying between Discord, Telegram and WhatsApp.	72

List of Figures

Figure 3.1	Overview of the proposed framework.	11
Figure 3.2	Visual indication - the IMA is non-functional.	15
Figure 3.3	IMA traffic generation - Synchronous and Asynchronous.	19
Figure 3.4	Distribution of delays sampled to emulate synchronous conversations.	21
Figure 3.5	Distribution of delays sampled to emulate asynchronous conversations.	21
Figure 3.6	Pipeline of emulating user interactions, including opening and closing IMAs.	24
Figure 3.7	Process of isolating traffic generated by a specific IMA.	26
Figure 3.8	Examples of the connections maintained by Android during the data generation process. The highlighted connections are maintained by the IMA.	27
Figure 3.9	Training two types of classifiers to differentiate and identify specific IMA traffic using data with ground truth (labels).	29
Figure 3.10	Feature sets employed in this research.	32
Figure 4.1	Signal only uses 2 elephant connections to manage all of the data transfers necessary for its functionalities.	43
Figure 4.2	Teams uses 2 persistent elephant connections to intermittent short periods of connections for its data transfer.	44
Figure 4.3	Compared to IMA traffic, web-browsing traffic makes a large number of parallel connections. Each horizontal line represents a unique flow.	45
Figure 4.4	F1 score of models that distinguish between flows from different IMAs, trained on varying sets of statistical features selected by different feature selection algorithms.	53
Figure 4.5	F1 score of models that distinguish IMA vs. Non-IMA traffic, trained on varying sets of statistical features selected by different feature selection algorithms.	54

Figure 4.6	F1 score of models that distinguish IMA vs. Non-IMA traffic, trained on varying sets of packet size/timing features selected by different feature selection algorithms.	54
Figure 4.7	F1 score of models that distinguish between flows from different IMAs, trained on varying sets of packet size/timing features selected by different feature selection algorithms.	56
Figure A.1	Classifying between IMAs using packet/timing features.	71
Figure A.2	Comparing the importance of features for my 6-class and 3-class models.	72
Figure A.3	Combining my 6-class model with my 3-class model.	73
Figure A.4	My combined model performs significantly better than the original model.	74

Abstract

Instant Messaging Applications (IMAs) are the primary communication tools for smartphone users. However, analyzing encrypted network traffic from IMAs poses challenges due to end-to-end encryption, user privacy, and dynamic port usage. Limited research exists on encrypted network traffic analysis of IMAs on mobile devices. This thesis proposes a comprehensive framework for generating and analyzing encrypted IMA traffic on mobile devices. The framework utilizes open-source tools to emulate user behavior and capture, filter and label resulting traffic on Android devices. It employs a data-driven approach using machine learning classification models to automatically extract features from network traffic and distinguish between different IMAs. Evaluation results show that it is possible to accurately identify different IMAs with high F1 scores. The thesis also evaluates the behavior of six popular IMAs and provides insights that could assist network operators and security experts to monitor and analyze network traffic effectively.

List of Abbreviations

DNS Domain Name System.

DoH DNS over HTTPS.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IM Instant Messaging.

IMA Instant Messaging Application.

JSON Javascript Object Notation.

ML Machine Learning.

non-IMA Any Application that is not used for Instant Messaging.

PC Personal Computer.

PCAP Packet Capture.

SSL Secure Socket Layer.

SVM Support Vector Machine.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UDP User Datagram Protocol.

UID Unique Identifier.

VPN Virtual Private Network.

XML Extensible Markup Language.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Dr. Nur Zincir-Heywood and Dr. Riyad Alshammari, for their invaluable guidance and support throughout the course of my research. Their expertise and dedication have been instrumental in shaping this work, and I am immensely grateful for their contributions. I would also like to thank Dr. Marwa Elsayed, and our collaborators at Solana Networks for their insights and support in this research.

I would also like to thank the faculty and staff of the Department of Computer Science at Dalhousie University, who have provided a supportive and stimulating academic environment that has allowed me to flourish as a researcher.

Finally, I would like to express my gratitude to my family and friends, who have provided unwavering support and encouragement throughout my research journey. Their love and support have been the foundation upon which I have built my academic success.

Chapter 1

Introduction

As the summer of 2022, mobile network traffic produced by smartphones constituted more than 50% of the entire Internet traffic [1]. In addition, Instant Messaging Applications (IMAs) have emerged as the dominant mode of communication on these devices, surpassing the number of text messages sent via cellular connections [1]. Instant Messaging (IM) is a service that supports text message transmission over the Internet. IMAs refer to any smartphone application that facilitates Instant Messaging, with WhatsApp being an example of a popular IMA [2]. Since the massive increase in the popularity of smartphone usage, Instant Messaging Applications (IMAs) have become widely used among smartphone owners.

There are various types of IMAs with various types of user bases. For example, WhatsApp and Messenger are for general users, each boasting more than 2 billion active monthly users who use the IMAs mainly for text messaging [1]. On the other hand, Discord is an IMA with a younger audience who use the IMA for extended voice chats, and Teams is an IMA with a professional audience [3].

With the widespread adoption of IMAs, a comprehensive understanding of their network communication patterns necessitates further investigation. Understanding and modeling IMA communication can yield numerous benefits, including but not limited to:

- Enabling network administrators to allocate network resources based on business requirements [4],
- Identifying IMA traffic from ground truth to investigate the services being on the network [5], and
- Analyzing threat models of attackers who violate user privacy by executing side-channel attacks [6]

Previous research in the field of network traffic analysis focus on various computer platforms [7]. However, there is a lack of understanding, modeling, and identifying IMAs on smartphones. This is shown to be challenging for several reasons. Firstly, the traditional method of port-based application/service identification does not work in this case, because IMAs often use HTTP/HTTPS to deliver their data [6]. In this thesis, I show this to be the case; as I will present in Section 4, all IMAs (except for WhatsApp) require ports 443 or 80 to be functional. Secondly, IMAs traffic is encrypted, and therefore opaque, which makes it challenging to employ traditional deep packet inspection approaches. Despite these challenges, there are some works on smartphone application traffic analysis in more general terms [6][8][9].

While the importance of the study of IMAs cannot be overstated, research in this area is currently limited. Notably, no comprehensive research has been conducted on how IMAs transmit and receive packets over the Internet. As such, this thesis aims to provide a comprehensive framework to study the ports and protocols necessary for IMAs as well as the connections they typically establish.

Furthermore, the limited amount of existing research on IMA traffic analysis is not reproducible, primarily because the existing research uses either network traffic data that cannot be made publicly available due to privacy reasons [1] and traffic datasets derived from real user traffic [6]. Thus, such datasets were not shared with the research community at large due to ethical concerns. To address this, I propose a framework that includes emulation and capturing of IMA traffic using empirical research findings from the literature. This then enables the datasets to be provided publicly for the reproducibility of the research to the benefit of the research community. Thus, this thesis presents a method for generating realistic IMA traffic that takes into account the various qualities that real IMA users exhibit, including message content, delivery times, typing speed, and patterns of opening/closing IMAs on a smartphone. Using these data, I perform flow-level encrypted traffic analysis via Machine Learning (ML) based approach employing six IMAs, namely WhatsApp, Messenger, Discord, Signal, Microsoft Teams, and Telegram [2][3][10]. For the purposes of my analysis, a flow is defined as a communication channel that represents aggregated packets with the same unique 5-tuple values, including the source and destination IP addresses, source and destination ports, and protocol in use [11]. Moreover, it is important to note that

this study is concerned specifically with encrypted traffic analysis. While there is a wide variety of user bases and use cases in the IMAs I study, there are similarities in the nature of their traffic. Specifically, all IMA traffic is encrypted although there are differences in the specific encryption scheme used by the IMAs. The descriptions of all the IMAs used are detailed in Section 3.1. It should also be noted here that this thesis does not contain any results from deep packet inspections or content analysis. Cryptography analysis of the schemes used for IMA traffic is beyond the scope of this research.

In the following, I present a multi-faceted approach to analyzing encrypted traffic from major IMAs. In particular, I aim to answer the following research questions:

- How do IMA flows behave, and are they distinguishable from other types of network traffic?
- Are there any differences between the network traffic generated by different IMAs?

To study the above research questions, I focus on the aforementioned IMAs on Android smartphones. To analyze and study IMA traffic characteristics on Android platforms, I first generate a dataset using Android emulators. To generate IMA traffic as close to real life as possible, I first use the latest versions of the IMAs that smartphone owners use. Secondly, I design my user emulation process using state-of-the-art empirical research on how smartphone users interact with IMAs [12][13]. Thirdly, I have made this dataset public for not only the benefit of the network research community but also for the reproducibility of my research ¹. Using the datasets generated, I perform a comprehensive analysis of the IMA network traffic behavior. To this end, I determine the network protocols and ports that IMAs typically use. Then, I train machine learning (ML) models (classifiers) to model the traffic behavior of the IMAs. As I analyze IMAs, I find that they produce different behaviors in terms of how they create and maintain flows.

The results of this research are valuable to both network operators and engineers. By reliably identifying IMA traffic, resources can be better allocated and efficiency increased. Further insights can also be gained into the nature of traffic characteristics

¹<https://iee-dataport.org/documents/encrypted-mobile-instant-messaging-traffic-dataset>

and behavior within their network for security, operations, and management purposes. Therefore, the main contributions of this research are as follows:

- Generating and capturing IMA traffic as close to real-life behavior as possible [14][15],
- Understanding the traffic characteristics of IMAs, including what ports and protocols they use to communicate with their respective servers [16],
- Distinguishing IMA encrypted traffic from non-IMA encrypted traffic using datasets generated by the most popular IMAs on the market,
- Distinguishing the aforementioned IMAs using their encrypted traffic characteristics as well as identifying the most significant attributes of different IMAs, and
- Designing, developing, and evaluating a comprehensive framework to deliver the above activities on IMAs for monitoring, operations, and management purposes.

The rest of the thesis is organized as follows. Chapter 2 summarizes related literature in this field. Chapter 3 introduces the proposed framework and the methodology followed. Chapter 4 details the evaluations, results, and comparisons to the related literature. Finally, conclusions and future work are discussed in Chapter 5.

Chapter 2

Related Work

This research builds on an existing literature in the field of encrypted traffic analysis and mobile application traffic analysis. I will present this array of existing work in three sections. Section 2.1 gives an overview of the state-of-the-art in the area of identifying smartphone applications. Section 2.2 summarizes the state-of-the-art in the area of identifying user actions on smartphone applications using their network traffic. Section 2.3 presents the existing works which have studied the IMAs in the literature. Finally, the last section summarizes the research gaps that are not addressed in the literature.

2.1 Analyzing Smartphone Traffic

Fingerprinting smartphone applications using their encrypted network traffic is a well-explored area of research. Researchers have fingerprinted the traffic of several Android and iOS applications using a variety of methods.

Taylor et al. analyze a set of 110 most commonly used Android applications [6]. Using the Random Forest classifier, the authors achieve a classification accuracy of over 90% in identifying these applications by analyzing their network traffic. The authors show that traffic patterns of these studied applications change over time with version updates, and highlight the need to continuously retrain their models to maintain reliability. While popular IMAs, namely Facebook Messenger and WhatsApp, are studied in this research, the authors do not focus on these IMAs and analyze them as a category.

Extending the work of Taylor et al., Cai et al. use Markov Chains and Graph Neural Networks to more accurately identify 29 widely used smartphone applications, not including any IMAs, using their network traffic [8]. The proposed method utilizes Markov chains to extract hidden topological information from the traffic flow. Based on this topological information, the authors construct a graph structure and

incorporate the sequence information of traffic into node features of the graph. The authors test the method on real-world datasets and achieve a classification accuracy of over 96%, a significant improvement over [6]

While Android is the main OS used for smartphones due to its open-source nature and ease of use, some researchers have also analyzed iOS traffic. Wang et al. detect mobile application traffic using only packet-level features [17]. The authors also perform feature selection to prune the most important features and achieve a classification accuracy of over 68.59% with selected features and over 87.23% with complete features. The authors find that Random Forest is the best classification algorithm for the task of classifying between 13 iOS applications. While these applications use Facebook Messenger and Snapchat, they do not study IMAs as a category.

Alan et al. examine the possibility of identifying Android applications using only TCP/IP header contents [18]. Their approach was based on packet sizes within the launch time traffic, which are expected to yield good feature sets for application identification. The experiments were carried out on 1,595 applications across four Android devices, none of which are IMAs, and achieved a classification accuracy of over 88%.

Jiang et al. focus on encrypted traffic used by remote desktop applications. Specifically, the authors study the information leakage of six remote desktop software programs on Windows 10 and 7 platforms: Anydesk, ConnectWise, MicroRDS, RealVNC, Teamviewer, and Zoho Assist. The authors use a variety of supervised machine learning algorithms built on statistical features of flow bursts to analyze the data. The results indicate that an adversary can accurately classify 5 types of daily activities, including editing documents, reading documents, surfing the web, watching videos, and installing software, with a true positive rate of over 95% and a false positive rate of less than 3%.

Similarly, Shi et al. propose a method that is able to accurately identify smartphone applications within a limited number of transmission packets or bytes [9]. The proposed method, called SpBiSeq (Simple Bi-Directional Sequence), generates highly robust early-stage classification fingerprints for encrypted mobile traffic with better classification performance. Furthermore, SpBiSeq has low time and space complexity, making it well-suited to high-speed network environments. The authors use 60

popular Android applications which they do not specify, and achieve over 86% accuracy. While the works in this section fingerprint network traffic from a wide variety of smartphone applications, none of them specifically explore the nature of IMA traffic. In contrast, this research examines IMA traffic exclusively and as a category.

2.2 Analyzing User Actions

Extending smartphone application analysis, some researchers aimed to analyze specific user actions on mobile applications. In this section, I detail such work since most of them intersect with the aim of this research.

Conti et al. fingerprint a set of user actions for traffic resulting from smartphone applications [19]. The authors use categorical information (such as IP addresses, domain filtering, and port numbers) as well as statistical features to build a variety of ML models. The authors focus on three widely used mobile applications: Facebook, Gmail, and Twitter. They obtain accuracy and precision of over 95%.

Fiscione et al. use side-channel analysis to identify specific user actions on WhatsApp using traffic analysis [20]. These actions include starting/ending a call, and joining/leaving a group chat since these actions produce consistent network traffic packets. The authors achieve a classification accuracy of over 90% when fingerprinting most of these actions.

In a multidimensional study in the traffic analysis of Instagram, Wu et al. demonstrate that it is possible to fingerprint the network traffic pattern that is generated by Instagram IMA when a user opens and uses direct messaging [21]. However, beyond Instant Messaging, the authors fingerprint a variety of behaviors one can exhibit when using Instagram, including sharing pictures, opening settings, etc. Specifically, the authors suggest an approach to divide the distribution ranges of these stable features based on the principle of maximum entropy and map the feature space into the support vector machine (SVM) vector space. On a dataset generated by real users, the authors report a classification accuracy of over 99%.

Liu et al. have developed an analyzer capable of classifying encrypted mobile traffic into application usage activity [22]. To achieve this, the authors select discriminative features from traffic packet sequences, using similarity measurements. Specifically, the authors group time windows generated from the same service usage

activity using recursive KMeans clustering (rCKC). The authors then use the feature vectors of cluster centers as input to a Random Forest classifier to identify corresponding service usages. For experiments, the authors analyze WeChat, WhatsApp, and Facebook applications. The authors report a classification accuracy of over 86% for all of the applications.

Jiang et al. focus on encrypted traffic used by remote desktop applications. Specifically, the authors study the information leakage of six remote desktop software programs on Windows 10 and 7 platforms: Anydesk, ConnectWise, MicroRDS, RealVNC, Teamviewer, and Zoho Assist. The authors use a variety of supervised machine learning algorithms built on statistical features of flow bursts to analyze the data. The results indicate that an adversary can accurately classify 5 types of daily activities, including editing documents, reading documents, surfing the web, watching videos, and installing software, with a true positive rate of over 95% and a false positive rate of less than 3%. While these works fingerprint actions from IMA applications, they do not provide descriptive analysis, such as ports/protocols used and flows generated.

2.3 Analyzing Instant Messaging Traffic

This section lists previous works that are most similar to this research. Specifically, I describe all of the works that study IMA traffic specifically for network analysis and potential security vulnerabilities.

Coull et al. report on one of the first studies to characterize Instant Messaging traffic [23]. The authors study iMessage, Telegram and Viber IMAs and found that there are significant differences in packet sizes for traffic generated by different user actions. The user actions include typing a text, sending a text and reading a text. The authors then developed a classifier to distinguish between user actions [23], with a classification accuracy of over 96% for all actions. While this work acts as the first study that profiles IMA traffic specifically, it only measures the differences in packet size distribution between these IMAs. Since they do not further profile IMA traffic, I extend this work to provide a more comprehensive analysis on this topic.

Sina et al. show that IP addresses that a flow uses are accurate indicators to classify IMAs [1]. Then, using those IP addresses, they profile traffic from four popular

IMAs- Facebook Messenger, Google Hangouts, Snapchat, and WeChat- on a campus network over several days where the profile includes traffic volume, TCP connection characteristics, and throughput [1]. The authors report that it is possible to identify IMAs with over 99% accuracy using only IP addresses. However, similar to [23], this work is limited in its scope of IMA traffic analysis. Specifically, this research only analyzes the IP address of the IMA traffic and the volume of traffic that IMA produce. Extending this type of analysis, we also study the statistical profile of IMA traffic.

Bahramali et al. show that traffic analysis can be used by researchers to reveal memberships in group chats hosted on Telegram [24]. To accomplish this, the authors first join a group chat, $g1$, as an active rogue user. Then, they collect IMA traffic at the target location and determine if the target is subscribed to $g1$ using timing-based traffic analysis [24]. The authors achieve a classification accuracy of over 98% when identifying group membership. While this work analysis IMA traffic and show that it is possible to carry a specific type of attack, it does not report any further analysis of IMA traffic in general. Furthermore, this work only focuses on one IMA- Telegram.

2.4 Summary

Existing literature either focuses on one IMA to show the minimum viability of IMA traffic identification, or hunt for specific vulnerabilities in the encrypted traffic generated by an IMA. Therefore, as discussed above, there has been a lack of work that qualitatively analyzes IMA traffic in general.

In this research, I use a wide variety of IMAs (differing in their demographics and functionalities) to cover a variety of IMA traffic (Section 3.1) [2][3]. To the best of my knowledge, this research is the first to describe IMA traffic as a category, aiming to distinguish IMA traffic from other types of encrypted network traffic as well as aiming to distinguish one type of IMA traffic from another type of IMA (Section 3.2 and Section 3.8). Furthermore, this thesis is the first to generate encrypted IMA network traffic incorporating realistic user behaviors and releasing it to the research community at large (Section 3.3).

Chapter 3

Methodology

In this chapter, the proposed framework is detailed and the methods used to analyze IMA traffic are discussed. In particular, section 3.1 details the set of Instant Messaging Applications used for this research. Section 3.2 details my approach to determine the traffic characteristics that IMAs utilize. Section 3.3 details how the IMAs traffic datasets are generated, captured, and analyzed. Figure 3.1 illustrates the overview of the proposed framework.

3.1 IMAs employed

In this research, I study six of the most widely-used IMAs on the market with diverse user bases (Table 3.1). As IMAs vary in terms of user demographics and position in the marketplace, I picked applications that most comprehensively represent a cross-section of IMA users. I review each of the six IMAs below.

3.1.1 Whatsapp

WhatsApp is a cross-platform instant messaging application that is owned by Facebook. The application uses end-to-end encryption to protect user conversations and data, ensuring that only the sender and recipient can access the messages. WhatsApp

	WhatsApp	Messenger	Telegram	Teams	Discord	Signal
Monthly Users	~2 billion	~1.3 billion	~700 million	~270 million	~150 million	~40 million
Owned By	Meta	Meta	Telegram Messenger Inc.	Microsoft	Discord Inc.	Open Whispers Systems
Extra Functionalities	Calls, Files, Private Group Chats, Handling Payments	Calls, Files, Private Group Chats	Calls, Files, Public and Private Group chats	Calls, Files, Private Group Chats	Calls, Files, Customizable Public and Private Group Chats	Calls, Files, Private Group Chats
Requirement for Usage	Valid phone number	An email address	Valid phone number	An email address	An email address	Valid phone number
Primary User Base	General (used heavily in Asia and Europe)	General (used heavily in Asia and North America)	General (used heavily in Asia)	Catered towards Professional settings.	Catered towards young audience	Catered towards privacy-focused audience

Table 3.1: IMAs employed in this thesis.

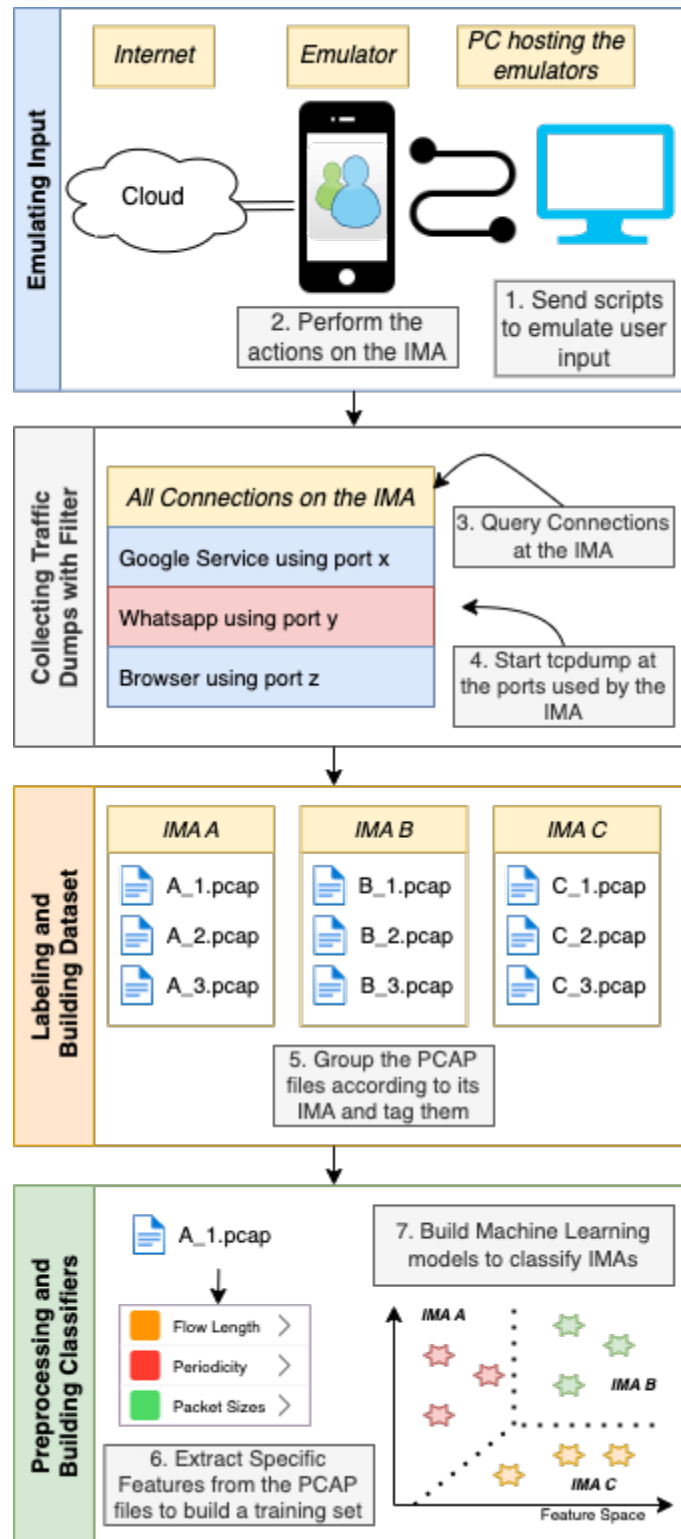


Figure 3.1: Overview of the proposed framework.

provides end-to-end encryption, implemented using the Curve25519 cryptographic algorithm [25]. Interestingly, this end-to-end encryption functionality is implemented by Open Whispers Systems which owns Signal. As of June 2022, WhatsApp has over 2 billion monthly active users, with a majority of its users residing in developing countries. The application is popular among individuals, families, and small businesses for its ease of use, free messaging and voice/video calls, and ability to send multimedia content.

WhatsApp was founded in 2009 by two former Yahoo employees, Jan Koum and Brian Acton [25]. It quickly gained popularity for its simple user interface and the ability to send messages, images, and videos without incurring additional charges beyond the user's internet data plan. In 2014, Facebook acquired WhatsApp for \$19 billion. In January 2021, WhatsApp updated its privacy policy, causing some backlash from users concerned about how their data would be used [25]. The policy would have allowed Facebook, which owns WhatsApp, to access more user data, such as phone numbers and transaction data. After the backlash, WhatsApp delayed the implementation of the policy to May 2021 and clarified that the changes would not affect the privacy of users' conversations.

3.1.2 Messenger

Messenger is a messaging application that is also owned by Facebook. It has similar features to WhatsApp, including end-to-end encryption for private conversations. As of June 2022, Messenger has over 1.3 billion monthly active users. The application is also used by businesses for customer support and marketing purposes [26]. While it is mainly for individual consumers, businesses can also use Messenger to deploy scripts that send automated replies to business inquiries [26].

Messenger was first introduced as a standalone application in 2011 and was integrated into the Facebook platform in 2014. Messenger's end-to-end encryption is also based on the Signal Protocol, ensuring that user data is private and secure. Messenger has also been criticized for its handling of user data, particularly in relation to the Cambridge Analytica scandal in 2018 [27].

3.1.3 Telegram

Telegram is a cloud-based messaging application that emphasizes user privacy and security. Telegram also offers a Secret Chat feature, which uses a self-destruct timer and does not allow forwarding or screenshotting of messages. As of June 2022, Telegram has over 700 million monthly active users.

Telegram is popular among users who prioritize privacy and customization, as it allows for the creation of custom themes, channels, and bots. As it promises secure and private communication channels (including public forums), it has been recently relied on in a number of protest movements [24]. Therefore, Telegram has faced criticism for its potential to be used by extremist groups, with some countries, such as Russia and Iran, banning the application [24].

3.1.4 Teams

Microsoft Teams is a collaboration platform that allows users to chat, meet, call, and collaborate on projects. Teams was launched in 2017 as a part of Microsoft's Office 365 suite of productivity tools. The platform has a wide range of features, including the ability to host virtual meetings, collaborate on files, and integrate with other Microsoft applications. It uses Microsoft's Secure Real-time Transport Protocol (SRTP) for audio and video calls, which provides encryption for media transmission. Teams also supports end-to-end encryption for chat conversations, which is currently in preview mode. As of June 2022, Teams has over 270 million monthly active users. Teams is popular among businesses and organizations for its collaboration features, including file sharing, co-authoring, and project management. Microsoft also announced new features, such as the ability to host interactive webinars and integration with its Viva platform for employee engagement.

3.1.5 Discord

Discord is a messaging and voice chat application that is popular among gamers and online communities. Specifically, Discord is marketed as a place for users to gather in a community-oriented manner using customizable and public servers. Due to this nature, Discord mainly appeals to a younger audience. Discord was launched

in 2015 and has gained popularity among the gaming community for its voice chat capabilities, ease of use, and customizable features.

Because Discord does not offer end-to-end encryption for private conversations, messages are technically accessible to Discord’s servers and could potentially be accessed by third parties. As of June 2022, Discord has over 150 million monthly active users. Discord is popular among gamers, creatives, and online communities for its ease of use, voice chat capabilities, and customizable features.

3.1.6 Signal

Signal was launched in 2014 by the non-profit organization Signal Foundation. The application has gained a reputation for its strong focus on user privacy and security, with features such as end-to-end encryption, self-destructing messages, and the ability to hide the sender’s metadata. Signal’s encryption method, the Signal Protocol, is open-source and has been independently audited for its security [10]. Signal also does not collect user data, ensuring that conversations remain private. It uses the Signal Protocol for end-to-end encryption, ensuring that user conversations and data are private and secure. Signal also uses the Double Ratchet Algorithm for message key generation, which provides additional security for user data [10].

As of June 2022, Signal has over 40 million monthly active users. Signal is popular among privacy-conscious individuals and activists for its secure messaging and voice call capabilities. In May 2021, Signal announced a new feature called “Signal Payments,” allowing users in the UK to send and receive cryptocurrency through the application. The feature is currently only available for UK users, but Signal plans to expand to other regions in the future [10].

3.2 Describing IMA Flows

This research aims to analyze the ports and flows utilized by different IMA applications.

Determining Ports Used. To determine the ports and protocols required for an IMA, I blocked certain widely-used TCP and/or UDP ports, and determine if the IMA remains functional (Figure 4.3). I consider an IMA functional if they are able

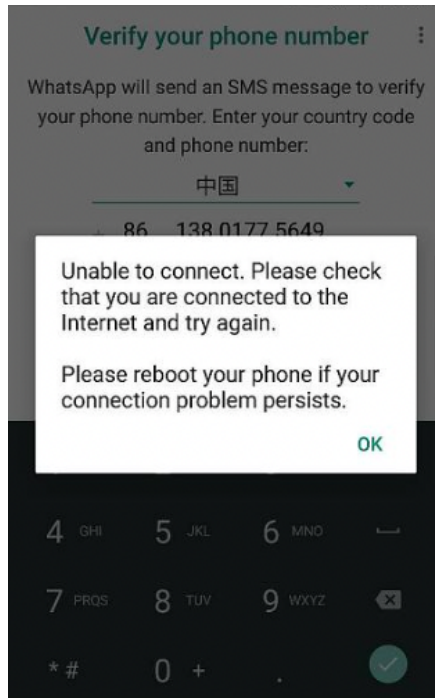


Figure 3.2: Visual indication - the IMA is non-functional.

to send messages and the messages are received on the other end. In particular, for an IMA to remain functional, I verify the following:

1. The sender sends a message without receiving an error,
2. The receiver of the message receives the message, and
3. The sender receives a receipt notifying that the message was delivered.

For example, Figure 3.6 shows the visual cues that IMAs typically display when they are operating normally. If any of these conditions are not met, I consider the IMA to be non-functional. For example, Figure 3.2 represents the error message that IMAs typically display when they are non-functional, such as when they cannot connect to the network. For each scenario that I studied, I verify the above conditions manually during testing. Section 4.2 presents the results from this analysis.

Determining Flows Generated. I test flows generated by each of the IMAs during a one-hour period of constant use, studying also their associated lifetime. I isolate the network traffic for each IMA using the method described in Section 3.3.10.

Using the flow characteristics exhibited by IMAs, this research can shed light into its network behavior patterns. This study is interested in the following:

1. How many flows each IMA generate within the period,
2. How long these flows are kept alive, and
3. How much data does the IMA send through each of the flows.

3.3 Realistic Data Generation

As one of the main contributions in this thesis, I describe my method of generating realistic encrypted traffic from IMAs, utilizing previous research on user behavior on IMAs. In this subsection, I detail the procedures of emulating general user inputs, collecting traffic resulting from emulation, and the qualities of various user behavior I considered to make my traffic set realistic. Figure 3.3 outlines the four main qualities that this research incorporated into the data generation process. They consist of two-way conversational traffic, realistic dialogue contents, different methods of responses, and realistic interactions with the IMA applications.

3.3.1 Equipment Setup

All IMAs are installed on an emulated instance of Google Pixel 4a running AndroidOS 13. The emulator tool that this study uses is provided by the Android Software Development Kit. I deploy twelve instances of the emulator, two for each IMA. This research uses two emulators for every IMA which I run both at the same time to emulate two-way conversations. These instances are installed on a powerful modern computer (Processor - Core i9, 8-core, 3.9 GHz CPU; 32 GB RAM) running Windows 11. I install no more than one IMA for each emulator instance. I install the latest the IMA as an *apk* file and push it into the emulated device using the Android Debug Bridge (*ADB*)¹. To intercept the network traffic, I install *tcpdump* on all emulators and capture encrypted traffic at the source.

The setup that I used posed a number of challenges. Interestingly, I found that IMAs generally required at least 2 GB of emulated host random access memory.

¹<https://developer.android.com/studio/command-line/adb>

When using less than 2 GB of memory, the application would often throttle or quit at random times. Therefore, all of my IMAs have at least 2 GB of emulated memory. Then, I found that my machine was not enough to run all 12 emulators at the same time. Therefore, as I collected my data, I performed the emulation in succession, running no more than two emulators at a time.

3.3.2 Emulating Inputs

I use *ADB* commands to send messages at specified intervals. Through *ADB*, I can send touch inputs to emulate interactions involving taps and swipes performed by actual users. Then, by using scripts, I can send a specific sequence of inputs to mimic messaging and interact with each of the IMAs. For example, to mimic a user sending a message on Signal, I build a script to:

1. Open the Signal IMA
2. Open a chat messenger for a dummy user that I created for experimental purposes
3. Type a message, such as "Hello", in the chat box
4. Click send to send the message
5. Close the Signal IMA

Then, I translate the task detailed above into the following sequence of *ADB* commands:

1. *adb shell input tap 350 500*
2. *adb shell input tap 500 200*
3. *adb shell input tap 400 1400*
4. *adb shell input text Hello*
5. *adb shell input tap 800 1400*

To automate the traffic collection process, I run these commands using *python* scripts. Since different IMA have different UI designs, I build specific scripts to handle user actions for each IMA. However, as I perform the collection, the scripts synchronize so that the IMAs open, send a message, and close at the same time.

3.3.3 Two-way traffic generation

Initially, I had used generated traffic where messages only flow in one direction. However, real conversations for an IMA comprise outgoing messages and incoming messages. Towards realistic traffic generation, I use two emulators for each IMA that represent two parties in a conversation. For each conversation that I emulate, I consider one of the emulators as speaker *A* and the other speaker *B*. The conversations that I use (as detailed in Section 3.3.4) assume there are two speakers for a conversation, *A* and *B*. Then, I can map these roles to my emulators in a one-to-one manner. Therefore, the encrypted traffic that I generate transmits both incoming and outgoing messages as well as resulting push notifications.

3.3.4 Realistic Dialogue

I use the SAMSum dataset as a repository of dialogues to replay [28]. This dialogue dataset consists of natural conversations, written by linguists who are proficient in English. The conversations exhibit a diverse range of styles and registers, including informal, semi-formal, and formal language, as well as slang phrases, emoticons, and typos. The authors collaborated with professional linguists for this task. The linguists were instructed to create conversations that were similar to those they engage in on a daily basis, with topics that reflected the proportion of real-life IMA conversations. These topics include casual conversation, gossip, meeting arrangements, political discussions, and university assignments. The dataset does not contain any sensitive information or fragments from other corpora. Each dialogue was created by a single person and subsequently annotated by language experts with summaries that are short, extract important information, include the names of the interlocutors and are written in the third person. Each dialogue contains only one reference summary.

The dataset contains a list of 14732 realistic and complete textual dialogues which all contain two speakers and realistic messaging patterns. Since researchers have not

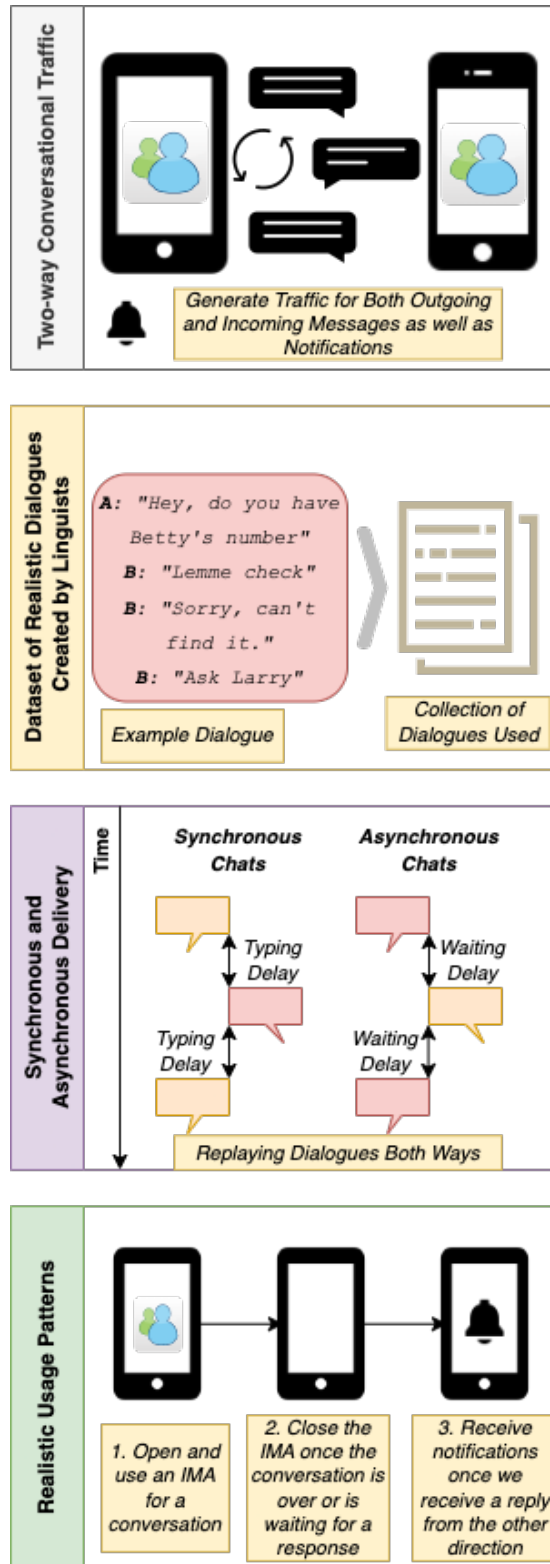


Figure 3.3: IMA traffic generation - Synchronous and Asynchronous.

released real-world conversational dialogues due to privacy concerns, I was forced to resort to an artificially-created dataset of dialogues. To the best of my knowledge, SAMSum is the only corpus of chat dialogues that I could incorporate for my research. To incorporate this dataset into my dataset generation process, I replay the provided dialogues where each of the two emulators for each IMA assumes a role of a separate speaker (Figure 3.1).

3.3.5 Two Types of Text Conversations

I found that IMA users engage in two different types of messaging conversations: synchronous and asynchronous [29]. Synchronous messaging involves two or more parties actively engaging in a text conversation in real-time (similar to a phone call) while asynchronous messaging concerns conversations where participants are not actively responding to each other in real time (similar to an email thread). Therefore, asynchronous messaging is characterized by delayed response times between messages. Depending on the type of delivery, I implement different types of delays to emulate real users more closely.

When emulating synchronous messaging, I emulate the delay that it takes for a user to type and send a message. Isaacs et al. [12] shows that the median response time in synchronous text chats is 18.7 seconds. For this conclusion, the authors recorded and analyzed a dataset consisting of 61,833 messages, comprising 3,096 conversations between 138 pairs of users. The chats were recorded at a workplace as the participants discussed mainly work-related topics. While this dataset contains work-specific and non-casual conversations, I concluded that it would still be an accurate representation of the time it takes for a user to prepare a response to a message, type it and send it. Using their findings, I build a probability distribution model to sample delays when emulating realistic synchronous messaging (Figure 3.4).

Similarly, when emulating asynchronous messaging, I emulate the delay that it typically takes for an IMA user to respond. Pielot et al [13] show that the average response time in asynchronous text chats is 369.0 seconds. In an effort to study the attentiveness of smartphone users when interacting with instant messaging applications, the authors develop a logging application and installed it on the phones of 24 Android phone users. For two weeks, the authors recorded both contextual data and

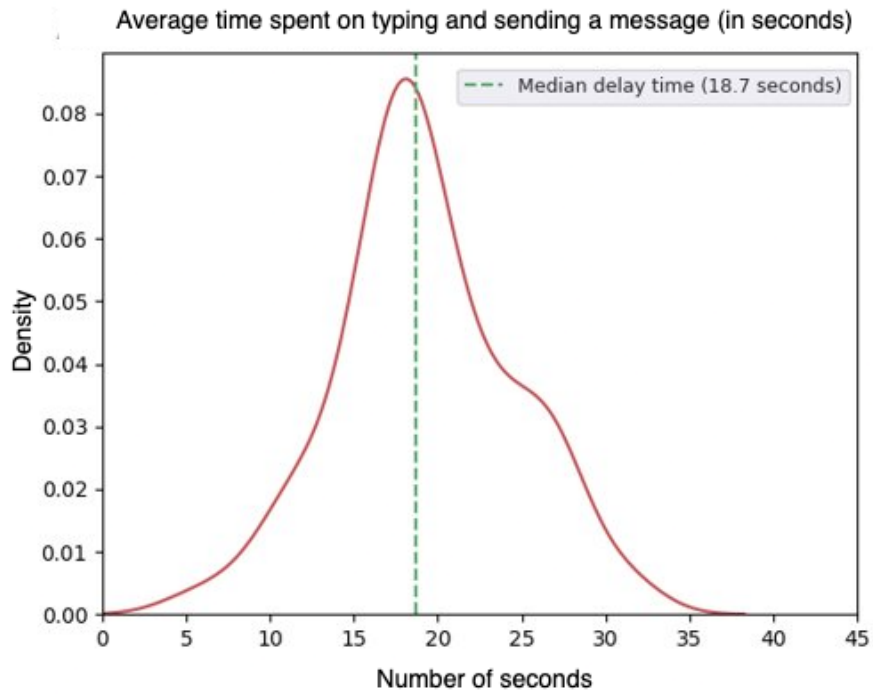


Figure 3.4: Distribution of delays sampled to emulate synchronous conversations.

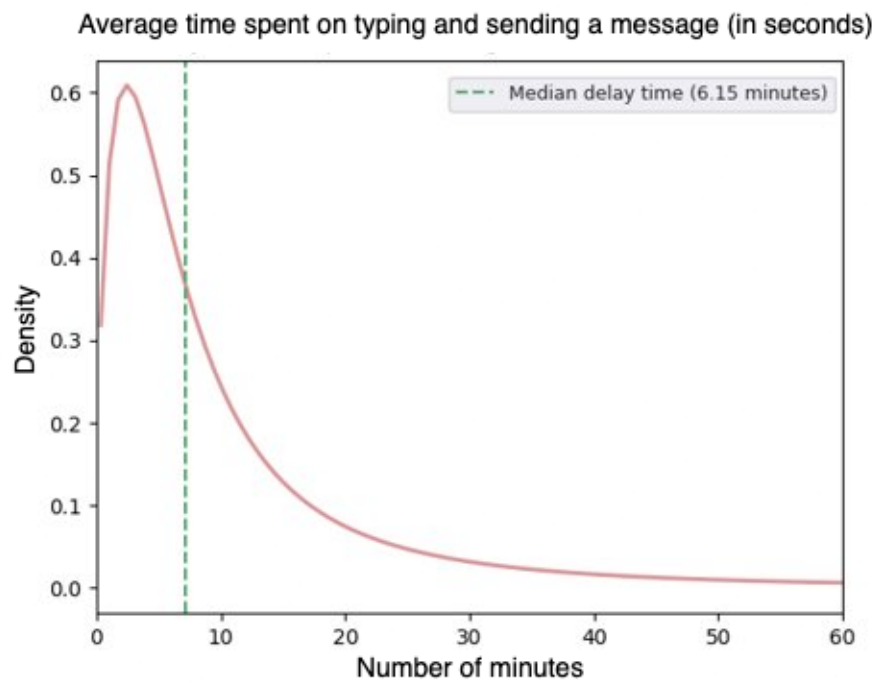


Figure 3.5: Distribution of delays sampled to emulate asynchronous conversations.

attentiveness information. While there was an array of surveys that may possibly indicate waiting delays for asynchronous messaging, I resorted to this study as my reference to building my asynchronous messaging model. This is because the authors note that there is a mismatch between users' answers to surveys and their actual attentiveness and response times as factually logged on their smart devices. Using their findings, I build a probability distribution model to sample delays when emulating realistic asynchronous messaging (Figure 3.5).

3.3.6 Opening and Closing IMAs

I implement a methodical approach to open and close the IMAs during the traffic generation process. As presented in Section 4.2, all IMAs close any network connection when the user exits the respective Android application and opens new connections as soon as a user opens an IMA. Since this demarcates the beginning and the end of a network flow, I implement careful rules to closely replicate user behavior. Let A and B be the two parties in my emulated conversation where A starts the conversation. Then, my process of emulating user behavior is as follows:

- The IMA applications are closed during the times when A and B are not actively engaged in the conversation i.e. during the times of "texting delay" when emulating asynchronous conversations.
- If the chat is asynchronous, both A and B close the IMA immediately after sending a message or a reply. If the chat is synchronous, the IMA is kept open and alive.
- To start a dialogue, A always opens the IMA and sends the first message.
- Before B sends the first reply, B always waits for the first message from A to arrive as a push notification.
- Once all the messages in a dialogue are sent, both A and B close the IMA. Then, it waits for one minute before starting the next dialogue.

Therefore, my user emulation comprehensively includes dialogues with realistic

content, realistic types of delivery and response, and realistic IMA interactions. Figure 3.6 shows an example of the emulation process, including opening and closing the IMAs.

3.3.7 Running the complete emulation

Incorporating all of the qualities of a real conversation as discussed above, including delay times and overall user interactions with an IMA, I build a comprehensive pipeline to emulate user IMA conversations. For my final dataset, I emulate synchronous conversations for 48 hours total for each IMA and asynchronous conversations for 24 hours total for each IMA. The characteristics of the resulting dataset is detailed in Section 4.

It is important to note my dataset is unbalanced in that the significant majority of the flows it contains are from synchronous conversations. While I spend a considerable amount of time collecting traffic from asynchronous conversations, it does not produce as many flows due to its highly time-consuming nature. In fact, due to the distribution models I built, some delays are as long as 6 hours. Since I close the IMA on my emulators during waiting delay periods, there are long periods during my asynchronous conversation data collection process where I do not collect any flows at all.

3.3.8 Collecting IMA traffic as PCAP files

In this research, *tcpdump* is used on the Android emulators to capture resulting packets from my emulation as *.pcap* files. This tool, *tcpdump*, is a network packet capture tool that allows users to intercept and analyze network traffic in real time. It operates by capturing packets that are transmitted over a network interface and displaying their contents in a human-readable format (*.pcap* files in my case). For my benefit, *tcpdump* is powerful and highly configurable. I set the "vantage point" of my traffic capture at the device as there are no other devices that are involved in this data collection process. The steps I took to process these *.pcap* files and turn them into a dataset is described in Section 3.5.

However, collecting all packets sent during my data collection periods does not give us an accurate dataset to model traffic from specific IMAs, even when there were

no other active applications running besides IMAs. Specifically, when I collected all traffic from the emulated device during the traffic generation process, the resulting *pcap* included non-IMA traffic including Android's own service traffic and traffic from other background applications. Therefore, I first need to filter the packets to clean the PCAP files such that they contain traffic from only my IMAs of interest. The process I used to filter for traffic from specific IMAs is described in Section 3.3.10.

3.3.9 Collecting non-IMA traffic as PCAP files

In order to build a model, in addition to collecting traffic from IMA applications, I also require traffic from non-IMA applications - mobile applications that are not used for Instant Messaging. Specifically, I collect traffic from three commonly used services on smartphones: Web-browsing, Video streaming, and Sending emails. For each of these services, I use the mobile applications Chrome, YouTube, and Gmail, respectively. To collect traffic from these non-IMA applications, I use similar scripts to those detailed in 3.3.2. Specifically, for each of the scripts I issue the following inputs to the emulators:

1. Open a non-IMA application.
2. Issue touch events to the emulator at random points on the screen every five seconds.
3. Repeat the previous steps for 1 minute.
4. Close the non-IMA application.

I also collect and include, in my model, all the background flows that are generated by the emulators that are not related to any of the IMAs employed in this research.

3.3.10 Isolating IMA Traffic

As outlined in Section 3.3.8, the *.pcap* files I obtain from *tcpdump* regularly contain traffic from unrelated connections, and I must filter traffic to isolate network traffic that only corresponds to my IMAs. Specifically, such traffic filtering allows the labels for the ground truth dataset to be accurate. Since there is no straightforward method

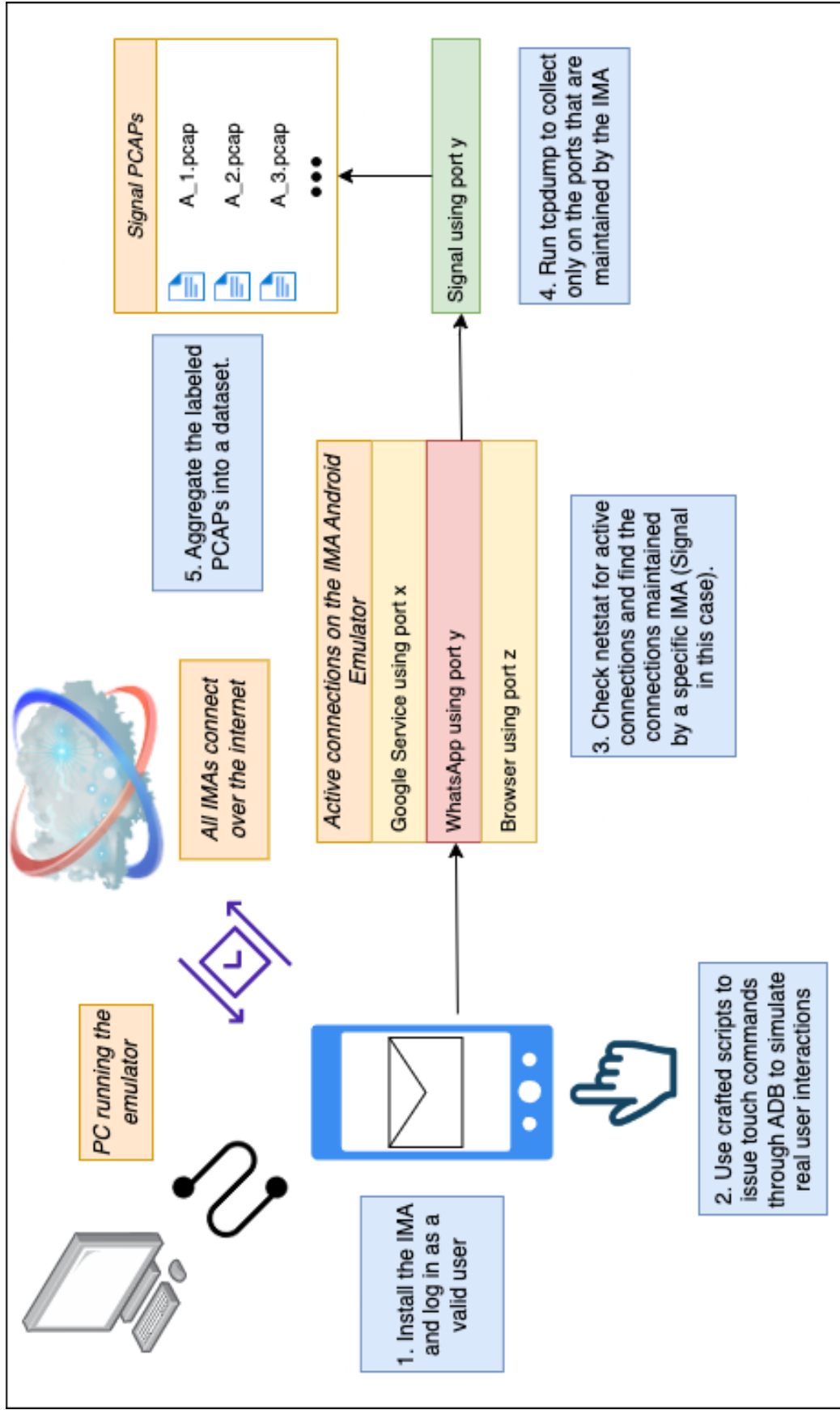
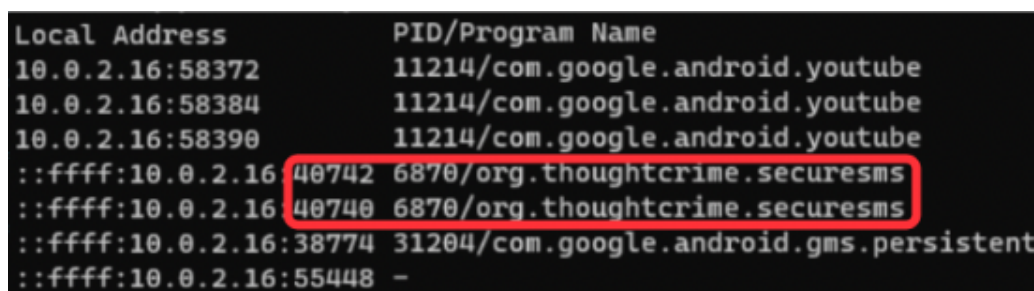


Figure 3.7: Process of isolating traffic generated by a specific IMA.

or an available tool to isolate traffic from specific applications, I use *netstat* to create my method and accomplish the task. I describe this method in this section.

All apps on Android operate as distinct users with their own UID. Furthermore, Linux's *netstat*² tool tracks and reports all active connections that an Android device has established. The *netstat* tool also reports the UIDs of the Linux users that use each connection as well as the associated port numbers.

Then, to collect the resulting traffic from a specific application, I use *netstat* to track connections that are used by my IMAs. Specifically, as collect traffic, I develop a script to issue *netstat* commands to obtain the list of ports (UDP and TCP) maintained by my IMA of interest (Figure 3.1). Then, I have a timeline of when all the ports, P , used by my IMA are opened and closed. Using this, I filter my captured PCAP files such that I isolate the packets that are sent through the ports, P , during the times that they were active.



```

Local Address          PID/Program Name
10.0.2.16:58372       11214/com.google.android.youtube
10.0.2.16:58384       11214/com.google.android.youtube
10.0.2.16:58390       11214/com.google.android.youtube
::ffff:10.0.2.16:40742 6870/org.thoughtcrime.securesms
::ffff:10.0.2.16:40740 6870/org.thoughtcrime.securesms
::ffff:10.0.2.16:38774 31204/com.google.android.gms.persistent
::ffff:10.0.2.16:55448 -

```

Figure 3.8: Examples of the connections maintained by Android during the data generation process. The highlighted connections are maintained by the IMA.

3.3.11 Limitations

There are a few limitations in our data generation process which could be improved by further research. To this end, adding new IMAs to the dataset requires a custom script that fits the particular application design and layout. For each IMA, this process needs to be repeated. Furthermore, a non-practitioner may not be skilled enough to perform the traffic isolation as described in Section 3.3.10. Additionally, while I collect flows from both synchronous and asynchronous conversations, the resulting dataset is heavily unbalanced in favor of traffic from synchronous conversations. The reason is

²<https://linux.die.net/man/8/netstat>

that emulating synchronous conversations consume significantly larger amount of time since the waiting delays can get hours long. Lastly, the strategy that we employed for non-IMA traffic generation could be improved by incorporating further user behavior patterns. While this may affect the classification performance, this dataset can still highlight the significant differences between IMA and non-IMA traffic.

3.4 Classification Tasks

My goal with this research is to model IMA traffic. Specifically, I aimed to model IMA traffic in two stages. Firstly, I aimed to model all IMA traffic as a homogeneous group of mobile application traffic. This means that I aimed to explore whether IMA traffic, as a class, is distinguishable from non-IMA traffic. This would allow network operators to identify IMA traffic as a category and identify IMA traffic from ground truth traffic that can be captured realistically on the internet. Secondly, I aimed to explore whether there are any differences between different IMAs. Finally, I combine these types of classifiers to determine if it is possible to identify specific IMAs from ground truth traffic.

Therefore, using the traffic set I generated (Figure 4.1 and 4.2), I perform two types of classification analysis to distinguish:

1. IMA flows from non-IMA flows,
2. Specific IMA flows from other IMA flows, and
3. Specific IMA flows from all flows (IMA and non-IMA).

Since these are two different classification problems, I use two different types of labeling when generating the dataset used for training machine learning (ML) models.

IMA vs non-IMA classification. IMA vs non-IMA classification. For this type of classification model, I label the flows as either "IMA" or "non-IMA", accordingly. Then, the model for this classification only has two possible classes. Section 4.3 reports the result of this type of classification model.

Between-IMAs Classification. For this type of classification model, I label the flows with the name of the IMA. As a result, the model for this classification has six possible classes. Section 4.3.2 reports the result of this type of classification.

Combining Classifiers. I combine the previous two types of classifiers to build a 2-stage pipeline to identify traffic from each IMA from ground truth (Figure 3.9). Therefore, I label the flows with all seven possible different classes- six IMAs and non-IMA. Section 4.3.3 reports the result of this approach.

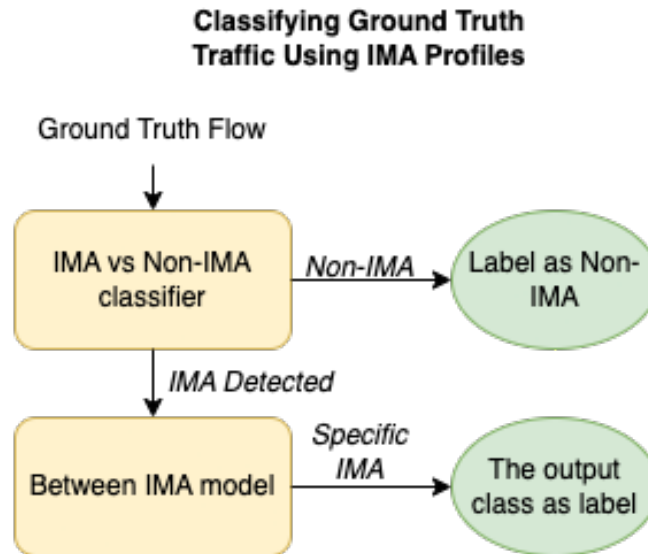


Figure 3.9: Training two types of classifiers to differentiate and identify specific IMA traffic using data with ground truth (labels).

3.5 Extracting Features

There are a number of tools researchers typically use to extract features from *.pcap* files and convert them into a dataset. The tools that researchers for this purpose are called "flow exporters". The popular options for flow exporters include Maji, Yaf, Softflowd, Netmate and Tranalyzer2 [30]. Haddadi et al. compared several flow exporter tools and found that Tranalyzer demonstrated the best performance [30]. Therefore, I use the newest version of Tranalyzer, *Tranalyzer2*, as my tool to export flows from the *.pcap* files that I collected.

Tranalyzer2 is a software tool used for generating network traffic flows and analyzing packet dumps, even those of large sizes [31]. The software comprises a core and plugins that users can selectively activate. By aggregating packets into flows, the software allows for better network operation analysis. *Tranalyzer2* supports packet mode, similar to *tshark*, but it also links each packet with its flow using a unique

numerical ID. The software can capture packets, allocate packets to flows, handle flow timeouts, invoke plug-in functions, and generate flow/packet-based output formats. Unlike other flow exporters, *Tranalyzer2* also provides information about flow direction, labeling A and B flows (client to server and server to the client respectively). Researchers often use *Tranalyzer2* to preprocess network traffic before training ML classifiers for malicious traffic detection.

In total, *Tranalyzer2* extracts 109 features from each flow, ranging from size statistics to protocol header statistics [31]. After I collect network traffic with labels indicating each IMA, I extract flows and the standard 109 features from them using *Tranalyzer2* [31]. Table A.1 lists all of the features that are extracted. I use these statistical data points as the features for my ML classifier model. Before I use these features as data points for my study, I first preprocess them to remove features that are either irrelevant or introduce heavy bias.

3.6 Preprocessing

I use every flow generated by my IMAs as data points for my analysis. Before I train my classifiers using the dataset, I preprocess my data to ensure the integrity of my research.

As explained in Section 3.3.1, I could not generate data simultaneously for all of my IMAs. Therefore, the flows that I collected for my IMAs have differing timestamps. Since these timestamps differ in amounts of up to a week, it introduces bias to my classification tasks without representing any insight into my analysis. Therefore, I first eliminate timestamp features since the time of my data collection is unrelated to the unique characteristics of my IMA traffic. These features include *timeFirst* and *timeLast* (Figure 3.10).

Furthermore, there are various features that are not exported by *Tranalyzer2* due to the hardware and the environment that I used for data collection. Specifically, my flow exporter outputs empty *ethType* and *ethVlanID* fields. Therefore, I eliminate these features from my dataset.

After preprocessing, I am left with a set of 105 total features that I use for my study.

3.7 Varying Feature Sets

I explore my dataset by building ML classifiers using three different sets of features (Figure 3.10). By using different feature sets, I sought to explore the following questions:

1. Can I profile IMA traffic when it is not obfuscated?
2. Are there any statistical markers that distinguish IMA traffic?
3. Can I profile IMA traffic when it is obfuscated by VPN?

Many smartphone applications use cloud Content Delivery Networks (CDN) to provide services [6]. This means that IP addresses are poor indicators when analyzing mobile application network traffic. However, my results, as well as previous research [1], show that all major IMA services have their own dedicated IP address and domain name spaces. Therefore, I can accurately identify IMA traffic using only their IP address.

However, to understand IMA encrypted traffic comprehensively, I aim to analyze whether there are any statistically significant markers of IMA traffic. Therefore, I separate my features obtained in Section 3.5 into two main types: descriptive features and statistical features. Descriptive features provide categorical information about the flows. Examples of descriptive features include IP addresses, protocols used, and header flags. On the other hand, statistical features are obtained by deriving numerical analysis on the aggregate flow. Examples of statistical features include the number of TCP retransmissions and flow duration. Lastly, I separate the statistical features that relate strictly to packet sizes and arrival times which I refer to as "packet size and timing features".

Furthermore, network traffic can be obfuscated in various ways. The most common way to obfuscate traffic is through a Virtual Private Network (VPN). Jayatilleke et al. conducts a survey into VPN usage and finds that 85% of the respondents have used VPN services on their smartphones [32]. All traffic, encrypted or not, that are routed through VPN is encrypted by the VPN client; therefore, any header content from the original packets is completely obfuscated. The only metadata or side-channel information that I can use to profile VPN-routed traffic are packet sizes

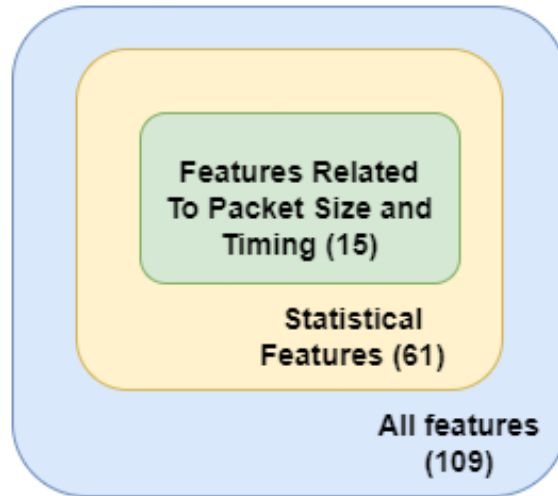


Figure 3.10: Feature sets employed in this research.

and interarrival times. Therefore, I experiment using a feature set that only contains packet size and timing features.

Therefore, I group the flow-level features, generated by *Tranalyzer2*, into three types and build distinct classifiers for each of these groups. These are: All features, Statistical features, and Packet Size and Timing features (Figure 3.10). Moreover, to eliminate any biases based on source and destination IP addresses and port numbers, I exclude these features while training the ML models employed in this thesis.

All features: By using all of the features, I aim to discover if I can reliably identify and profile IMA traffic under normal conditions.

Statistical features: By using this set of features, I aim to discover the statistical differences between the flows generated by my IMAs.

Packet size and timing features: In this approach, I only use features derived from packet sizes and interarrival times. This means that I do not use any features that are derived from any of the packet headers or protocol usage. I include this approach to discover the viability of IMA traffic analysis in the presence of VPN, which hides the original per-flow packet headers.

3.8 Building Machine Learning Classifiers

In this thesis, I trained and tested a series of ML models for each of the feature sets using eight distinct supervised learning algorithms: Nearest Neighbors, Linear SVM,

Decision Tree, Random Forest, Neural Network, Naive Bayes, Logistic Regression, and Gradient Boost. In doing so, my goal is to compare the performance of all these models and identify the most appropriate one for the research tasks in this thesis. In the following, an overview of each ML algorithm is presented. More detailed information on each algorithm could be found in [33].

3.8.1 Nearest Neighbors

Nearest Neighbors is a simple yet powerful classification algorithm that works by finding the closest training examples in the feature space to a new input example and classifying the new example based on the majority class of its nearest neighbors [33]. It can be used for both regression and classification problems. Mathematically, the k-Nearest Neighbors algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the k-Nearest Neighbors algorithm works as follows:

1. Calculate the distance between x and each training example x_i
2. Find the k training examples with the smallest distance to x (i.e., the nearest neighbors)
3. Assign the class label of the majority of the k nearest neighbors to x

3.8.2 Linear SVM

Support Vector Machines (SVMs) are a popular machine learning algorithm used for both classification and regression problems [33]. SVMs work by finding the best hyperplane that separates the two classes in the feature space. Linear SVM is a variant of SVMs that uses a linear kernel to compute the hyperplane. Mathematically, the Linear SVM algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Linear SVM algorithm works as follows:

1. Find the hyperplane that separates the two classes in the feature space
2. Assign the class label to the new example based on which side of the hyperplane it falls

3.8.3 C4.5 Decision Tree

A decision tree is a simple yet powerful algorithm that works by recursively partitioning the feature space into smaller and smaller subsets until each subset contains only examples from a single class [33]. Each partition is made based on the value of a feature, and the partitions are chosen to maximize the information gained at each step. Mathematically, the Decision Tree algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Decision Tree algorithm works as follows:

1. Choose the feature that maximizes the information gained at the current node
2. Partition the data based on the value of the chosen feature
3. Repeat the process recursively for each partition until all subsets contain examples from a single class.

3.8.4 Random Forest

Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting [33]. The algorithm works by randomly selecting a subset of features and a subset of training examples for each tree and then averaging their predictions. Mathematically, the Random Forest algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Random Forest algorithm works as follows:

1. Choose a random subset of features
2. Choose a random subset of training examples
3. Build a decision tree using the selected features and examples
4. Repeat the process to build multiple trees
5. Average the predictions of all trees to get the final prediction for the new example

3.8.5 Neural Network

A neural network is a machine learning algorithm inspired by the structure and function of the human brain [33]. It consists of multiple layers of interconnected nodes (neurons) that learn to represent the input data in a hierarchical manner. The algorithm works by feeding the input data through the network and adjusting the weights between the neurons to minimize the error between the predicted output and the actual output. Mathematically, the Neural Network algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Neural Network algorithm works as follows:

1. Feed the input example through the network
2. Compute the output of each neuron in each layer using the weights between the neurons
3. Compute the error between the predicted output and the actual output
4. Backpropagate the error through the network and adjust the weights using gradient descent to minimize the error
5. Repeat the process for multiple epochs until the error is minimized

3.8.6 Naive Bayes

Naive Bayes is a simple probabilistic algorithm that makes predictions based on the probability of each feature belonging to each class [33]. It assumes that the features are conditionally independent given the class, which simplifies the computation of the probabilities. Mathematically, the Naive Bayes algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Naive Bayes algorithm works as follows:

1. Compute the prior probability of each class
2. Compute the conditional probability of each feature given each class

3. Multiply the probabilities of each feature given the input to get the probability of the input belonging to each class
4. Assign the class label with the highest probability to the input

3.8.7 Logistic Regression

Logistic Regression is a classification algorithm that models the probability of the positive class using a logistic function [33]. It works by finding the weights that maximize the likelihood of the training examples, and then using the logistic function to compute the probability of the positive class for the input. Mathematically, the Logistic Regression algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Logistic Regression algorithm works as follows:

1. Find the weights that maximize the likelihood of the training examples: $\max_w \sum_{i=1}^n \log P(y_i | x_i, w)$
2. Compute the probability of the positive class for x : $P(y = 1 | x, w) = \frac{1}{1 + \exp(-w^T x)}$

3.8.8 Gradient Boost

Finally, Gradient Boost is an ensemble learning algorithm that combines multiple weak learners to form a strong learner [34]. It works by sequentially adding new learners to the ensemble, where each new learner is trained to correct the errors of the previous learners. The algorithm uses a loss function to measure the errors of the ensemble and uses gradient descent to minimize the loss function. Mathematically, the Gradient Boost algorithm can be represented as the following:

Given a training set of examples $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ and a new input example x , the Gradient Boost algorithm works as follows:

1. Initialize the ensemble with a constant value: $F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$
2. For $m = 1$ to M :
 - (a) Compute the negative gradient of the loss function with respect to the current ensemble: $r_{im} = - \left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} \right]_{i=1}^n$

(b) Train a new learner to predict the negative gradient:

$$h_m(x) = \operatorname{argmin}_h \sum_i 1^n L(r_{im}, h(x_i))$$

(c) Update the ensemble by adding the new learner: $F_m(x) = F_{m-1}(x) + \eta h_m(x)$

(d) Return the final ensemble: $F_M(x)$

where L is a loss function, η is the learning rate, and h_m is the m -th learner in the ensemble.

3.8.9 Training and Evaluating Models

To train and evaluate the ML models, I utilized the *Scikit-learn*³ tool, which contains all the necessary packages to build models with the algorithms I employed. By default, I used the settings provided by *Scikit-learn* for all the ML models generated, to ensure consistency and facilitate benchmarking.

To evaluate the performance of each algorithm on each classification task, I implemented 10-fold Monte Carlo cross-validation. Specifically, my dataset is unbalanced with Teams producing more flows than the rest of the IMAs combined. The reason for the difference in flow volumes between IMAs is unclear since the content of all communication is encrypted. To combat the unbalanced dataset, I use the same number, x , of training data points from each IMA. I split my dataset into training and testing using a 60-40 split. To ensure that I have a sufficient amount of training data from every class, I pick x to be 60% of total data points for WhatsApp (the class with the lowest number of data points). Therefore, for each fold, I randomly select 3981 data points from each IMA to be my training set. Then, I use the rest for testing.

For each fold, I measured the Accuracy, Precision, Recall, and F1 score. Accuracy is defined as $\frac{TP+TN}{TP+TN+FP+FN}$ across all classes. I calculate the Precision and Recall using the formulas $\frac{TP}{TP+FP}$, and $\frac{TP}{TP+FN}$ respectively. Finally, I define my F1 score as $\frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$. In multi-class cases, I calculated these scores using the unweighted mean of all classes.

³<https://scikit-learn.org/>

3.9 Feature Selection Analysis

For each algorithm used on each feature set, I present a range of scores resulting from the 10-fold Monte-Carlo cross-validation. Specifically, I report the lowest and the highest scores observed among all 10 folds.

The feature sets that I use have a relatively high number of features which increases computational complexity and decrease interpretability. However, network monitoring is typically performed on high-performance networking equipment [35]. In such environments, the number of features could increase resource consumption and computational speed exponentially [35]. Therefore, I perform feature selection analysis on the ML models to potentially decrease computational overhead. Specifically, for each task and feature set described previously, I aimed to discover the least number of features necessary to achieve comparable performance and identify those features. Therefore, the resulting feature set can be used without any performance costs.

To achieve this, I rank my features according to three different techniques commonly used in feature selection: Information Gain, Mutual Information and Chi Squared Test. Information Gain is a measure of the amount of information that a feature provides about the class variable. It is used to determine which features are the most informative and relevant for the classification problem [36]. Features with high Information Gain are considered more important for the model. Mutual Information is another measure of the dependence between two variables, which in the context of feature selection, is used to evaluate the relationship between the features and the target variable [37]. It calculates the amount of information that a feature provides about the target variable, taking into account the correlation between the two variables. Chi-Squared Test is a statistical measure that evaluates the independence between two variables [38]. In feature selection, Chi-Squared test is used to determine the most significant features by measuring the difference between the observed distribution and the expected distribution of a feature.

Our goal is to find a minimal set of features I need to achieve the baseline performance- the performance that I get from using all the features. Therefore, using each of my feature selection algorithms, I first rank them according to the output. Then, I iteratively add the features one by one according to the rankings. Specifically,

I perform the following procedure for each feature selection algorithm, a_x :

1. Start with the most important feature, f_1 , as ranked by a_x , and add it to my current set of features, $f_c = \{f_1\}$.
2. Using f_c and the method outline in Section 3.8, train a classifier.
3. Measure the F1 score of the model trained using f_c .
4. If the F1 score is lower than the baseline score, I add the next feature in the rank to f_c , and go back to step 2.
5. If the F1 score is the same as the baseline score, I consider f_c to be the minimal feature set required for comparable performance.

3.10 Summary

In this chapter, I detail my proposed framework from generating a realistic dataset, to understanding the network characteristics of IMA traffic to feature selection and analysis via ML classifiers. Using the proposed framework, my first priority in this research was to generate a set of realistic encrypted IMA traffic flows. To achieve this, I use Android equipment to emulate two-way conversational user interactions using behavior patterns based on the results provided in the state-of-the-art literature. Such behavior patterns include a delay between messages, typing speed, and opening/closing IMAs under real-world conditions. Then, I detail my methods of analyzing the descriptive qualities of IMA traffic. Specifically, I report on the identified ports and the protocols each IMA uses for functionality. Furthermore, I analyze the quality of the ports that are typically created and maintained by the IMAs employed in this thesis. This is followed by how I transform the captured traffic into datasets using the flow exporter *Tranalyzer2*. I first describe the characteristics of the flow exporter and the features I extracted using this tool. Then, I present how I preprocess the datasets such that the biases are minimized before ML training starts. Moreover, I describe the classification tasks I aim to accomplish, the classifiers used for these purposes, and their justification. Finally, I describe how I selected the minimum

necessary feature set using various feature selection algorithms to decrease computational complexity. The evaluations and results based on the proposed framework are discussed in Chapter 4.

Chapter 4

Evaluations and Results

In this chapter, I present the evaluations and the results of my analysis on the encrypted traffic generated by the IMAs and non-IMAs employed in this research. First, Section 4.1 describes the datasets that are captured, including the total size, number of flows, and number of packets. Then, Section 4.2 describes the qualitative characteristics of IMA traffic, including the ports and protocols IMAs use, and the characteristics of the flows they create. Then, 4.3 details the result of the ML models that are built to identify text messaging-based IMA traffic as well as specific IMAs used in this research. Lastly, Section 4.4 details the result of the feature engineering analysis and the insights gained.

4.1 Dataset Generation

Using the methods described in Section 3.3, I collect data over 72 hours for each IMA. To build the datasets, I emulate synchronous messaging for 48 hours and asynchronous messaging for 24 hours. Each of the IMAs generates at least 5000 flows over the same period. Table 4.1 details the characteristics of the resulting IMA traffic. Table 4.2 details the characteristics of the traffic that is produced using non-IMA mobile

	WhatsApp	Messenger	Telegram	Teams	Discord	Signal
# of Flows	6636	8904	12740	40562	8996	10804
Protocol	TCP	TCP	TCP	TCP	TCP	TCP
Total Size	17 MB	85 MB	154 MB	526 MB	99 MB	195 MB
# of Packets	19164	207885	553887	941198	265517	142854
# of DNS pkts	DoH	DoH	DoH	2118	1498	DoH
Encryption	TLSv1.2	TLSv1.2	SSL	TLSv1.2	TLSv1.2	TLSv1.2

Table 4.1: Summary of the traffic generated by each IMA application.

	Web-browsing	Streaming Videos	Sending Emails	Background
# of Flows	9040	14420	1120	9188
Main Protocol	TCP and QUIC	UDP and QUIC	TCP, UDP and QUIC	TCP, UDP and QUIC
Size	358 MB	953 MB	14 MB	127 MB
# of Packets	463361	925152	39166	294652
# of DNS pkts	DoH	DoH	DoH	n/a
Encryption	TLSv1.3 /QUIC	QUIC	TLSv1.3 /QUIC	TLSv1.3 /QUIC

Table 4.2: Summary of the traffic generated by each non-IMA application (Web-browsing, e-mail, and video streaming).

applications.

We can see that there is a large diversity in the quantity of the flows and the size of the traffic that IMAs produce. Teams produce the most traffic with over 40562 flows total, while WhatsApp produces the least number of flows at 6636. To combat this significant imbalance in my dataset and validate my results comprehensively, I use Monte-Carlo cross validation method as described in Section 3.8. The traffic set mainly contains traffic that uses TCP as its main protocol. While Discord and Teams occasionally use UDP, I found that the overwhelming majority of the traffic was TCP.

On the other hand, I see that non-IMA traffic has more variability in the protocols they use. Since all of the non-IMA applications I used to produce non-IMA traffic were Google products (Chrome, YouTube, and Gmail apps), I see that all of them use the QUIC protocol.

In total, I have generated 88642 flows (with a total size of 1076 MB) from my six IMAs and 33762 flows (with a total size of 1452 MB) from my four non-IM applications.

4.2 IMA Traffic Protocol

Using the methods described in Section 3.2, I perform a qualitative analysis of the IMA traffic. First, I identify the necessary ports and protocols used by the IMAs (Table 4.3). Then, I present the nature of the traffic flows generated by the IMAs.

IMA	Signal	Discord	Telegram	Teams	WhatsApp	Messenger
Block TCP	∅	∅	∅	∅	∅	∅
Block HTTP	∅	∅	∅	∅	✓	∅
Block UDP	✓	∅	✓	∅	✓	✓
Block DNS	✓	∅	✓	∅	✓	✓
Block UDP except DNS	✓	✓	✓	✓	✓	✓

Table 4.3: The ports and protocols essential for IMAs. ∅ indicates that the IMA becomes non-functional after I block the specified port, while ✓ indicates that the IMA remains functional.

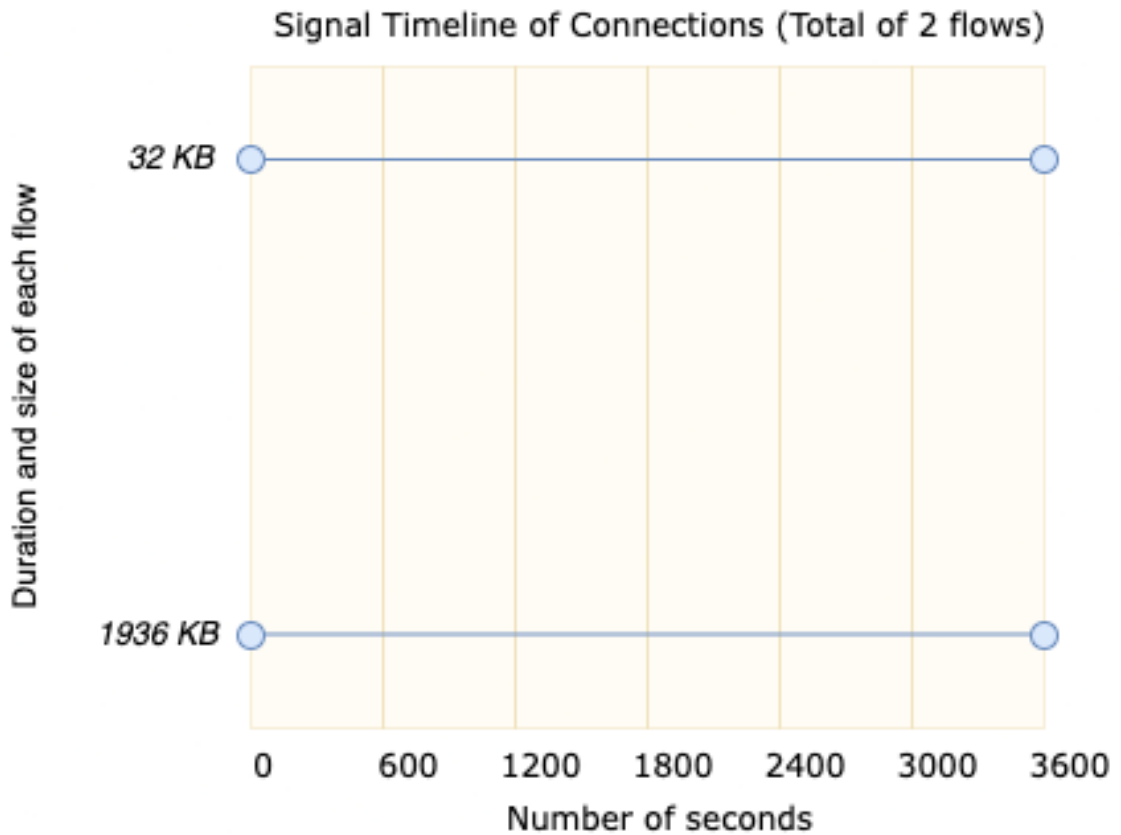


Figure 4.1: Signal only uses 2 elephant connections to manage all of the data transfers necessary for its functionalities.

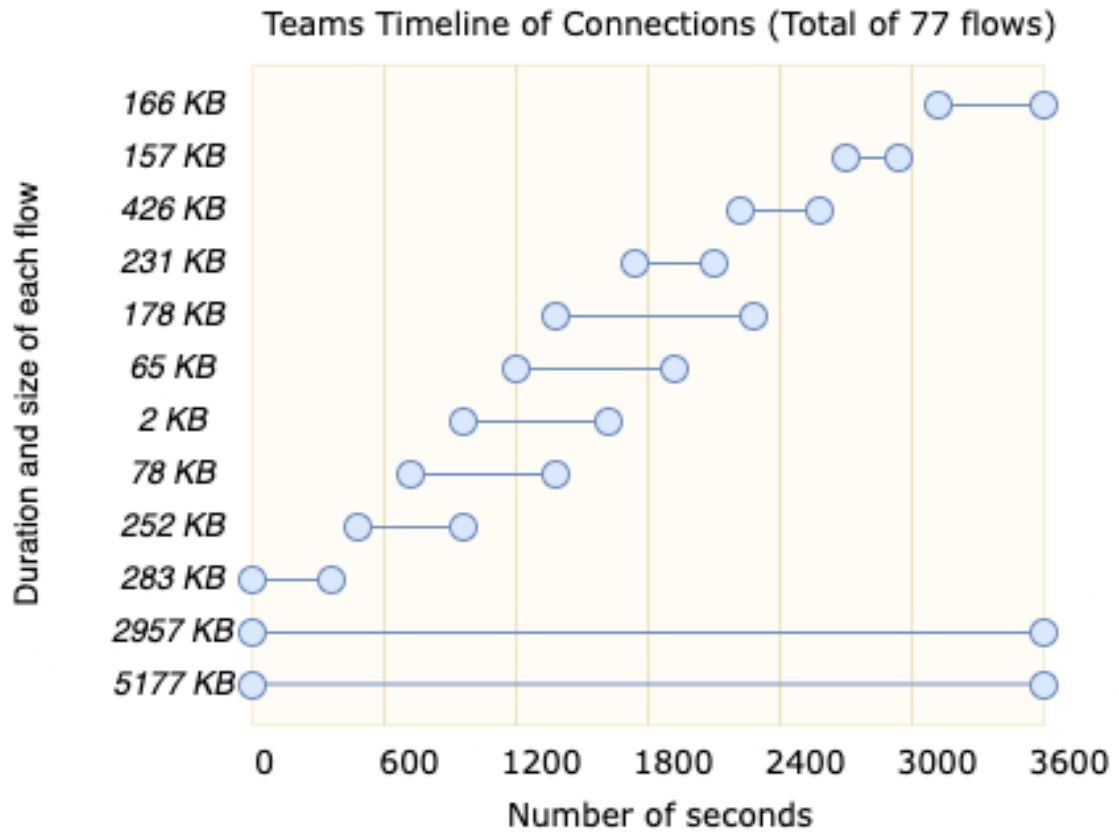


Figure 4.2: Teams uses 2 persistent elephant connections to intermittent short periods of connections for its data transfer.

4.2.1 Ports and protocols used

I observed that TCP is used by all IMAs and that most communication is encrypted over TLS. Furthermore, all IMAs use port 443 on the server side except for WhatsApp which uses port 5222. Therefore, when I block port 443 on my emulators, all IMAs, except WhatsApp, are rendered non-functional. When I block port 5222 on WhatsApp, it starts using port 443 instead and when both ports 443 and 5222 are blocked, WhatsApp is also rendered non-functional. This indicates that all IMAs rely on TCP connections over port 443 to transfer the required data. Furthermore, I find that all IMAs use TLS (mostly TLS version 1.2) as the default encryption protocol

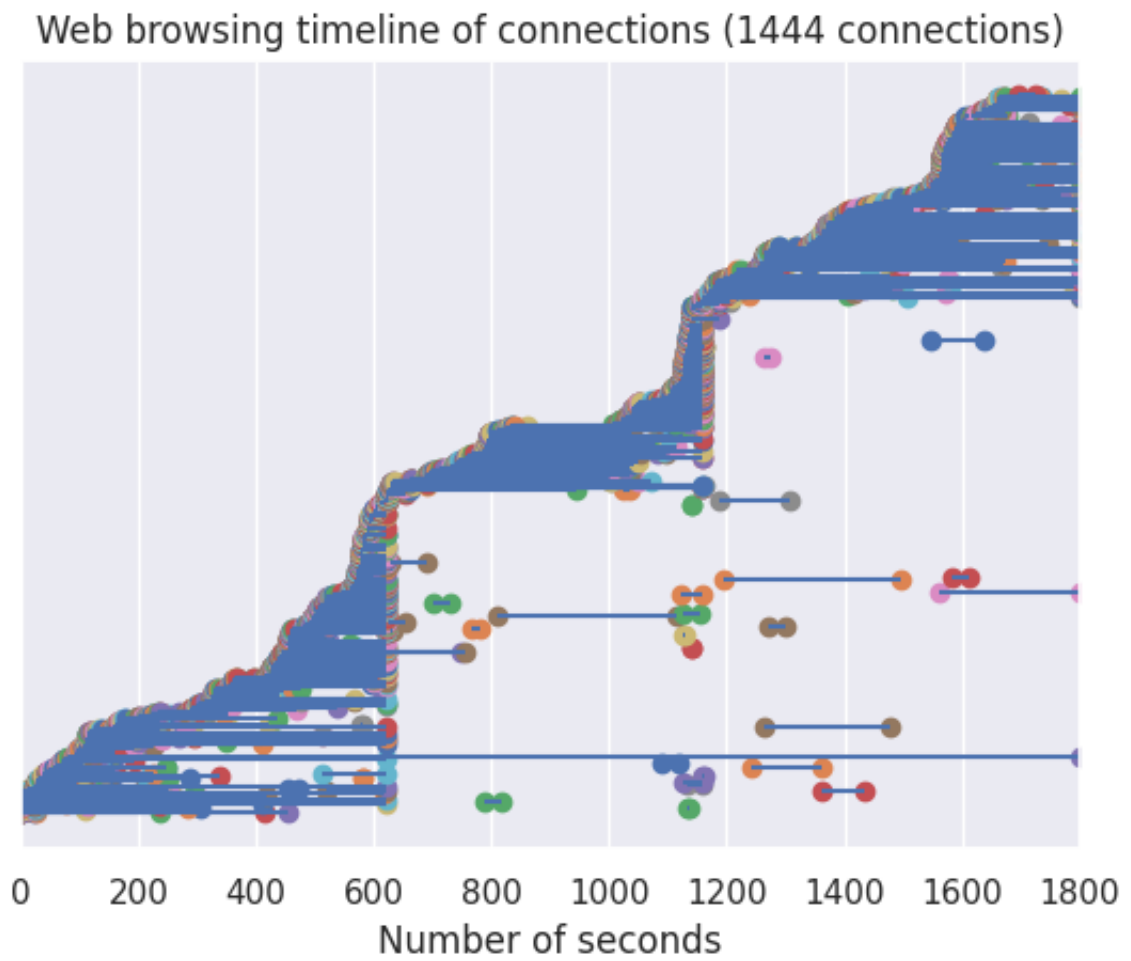


Figure 4.3: Compared to IMA traffic, web-browsing traffic makes a large number of parallel connections. Each horizontal line represents a unique flow.

except Telegram which uses a proprietary encryption protocol known as MTProto.

Also, I observed that most IMAs do not utilize UDP. In fact, when I block all UDP ports, only Discord and Teams are affected. I also found that these two IMAs use UDP only for DNS inquiries, as presented in Figure 4. I presume that the rest of the IMAs use DNS over HTTPS (DoH) since they remain functional when UDP traffic is blocked as presented in Table 4.3.

4.2.2 Flows generated by the IMAs

My results show that IMAs vary in the number of unique flows they create. However, I found that all IMAs utilize two persistent connections. My observations show that all IMAs consistently maintain two persistent flows through which the majority of the data is transferred. These two flows are never interrupted as long as the IMA is open. Furthermore, I consider these connections as elephant flows since they handle a majority of the necessary data transfer for their respective IMA (Figure 4.1 and Figure 4.2). This behavior is in stark contrast with flows that other types of network services typically use. For example, Figure 4.3 shows that web browsers use a large number of short bursts of flows (10 minutes in duration) where each flow transfers a small amount of data.

Signal and Telegram utilize only the two elephant flows, regardless of user input and data received by these IMAs (Figure 4.1). In contrast, the other IMAs constantly create new parallel connections that are short in duration and low in bandwidth. I present Teams as an example in Figure 4.2. I see that Teams utilize a high number of flows- a total of 77 flows under the same conditions and timeframe (Figure 4.2). However, I can see that all of these flows except for the two elephant flows are short and transfer low amounts of data.

4.3 Building Machine Learning Classifiers

I build three types of classifiers to analyze IMA traffic and test them on the feature groupings I defined in Section 3.5. For each of these feature groupings, I build a model for two classification tasks: classifying IMA vs non-IMA, and classifying between IMAs. I found that the Random Forest (RF) is the best-performing classifier for this task, achieving the highest F1 score under all scenarios.

4.3.1 Distinguishing IMA traffic from non-IMA traffic

I show that the objective of distinguishing IMA traffic from non-IMA traffic is well achieved.

Using all features. When I use all flow features, I see that I can obtain an excellent F1 score (Figure 4.4), with Random Forest performing the best. Table 4.13 shows the features with the highest importance for this model (defined using the method specified in Section 3.8).

Using statistical features. After eliminating descriptive features (refer to Section 3.9), I see that my model performance for Random Forest holds strong. This shows that IMAs generate flows with statistically different characteristics compared to non-IMA applications.

Using packet size and timing features. When I use strictly packet size and timing features, I still obtain high F1 scores over 92% when using Random Forest (Figure 4.8). This means that even if the original packet headers are missing, for example when the traffic is routed through a VPN, IMA traffic could still be identifiable.

4.3.2 Distinguishing between different IMAs

I obtain comparable results in these classification tasks to those in Section 4.3.

Using all features. Similar to the results in Section 4.3, I see that I can obtain strong results when using all of the flow features (Figure 4.5), with Random Forest boasting the highest performance.

Using statistical features. my model remains highly accurate when using only statistical features (Figure 4.7). This shows different IMAs produce network traffic with different statistical characteristics under the same conditions.

Using packet size and timing features. When I use strictly packet size and timing features, I obtain high F1 scores over 81% using Random Forest or Gradient Boost classifiers (Figure 4.9). This is promising for future research into identifying the usage of common IMAs even when their traffic is routed and obfuscated through VPNs.

4.3.3 Identifying IMA Traffic From Ground Truth

Through sections 4.3.2 and 4.3, I have established that I can both distinguish IMA from non-IMA and distinguish between IMAs. Then, I combine these two types of classifiers to show that I can identify and classify IMA traffic from non-IMA as shown in Section 3.4. I use strictly Random Forest for this section since it proved to be the most effective model for modeling IMA traffic.

Using all features, I can build a resilient model with F1 scores over 99% for all classes as shown in Figure 4.10. Furthermore, my model holds strong when I only use statistical features, boasting F1 scores over 96% for all classes (Figure 4.11). This shows that I can identify IMA traffic from ground truth extremely reliably.

Lastly, Figure 4.12 shows the performance of my model that only uses packet sizing and timing features. Overall, I achieve a median F1 score of 80.5% across the 10 folds of my cross-validation. This shows that I can identify IMA traffic from ground truth even when it is obfuscated by VPN highly reliably. Furthermore, Figure 4.12 shows that Teams, Signal, and Telegram all boast F1 scores of over 90% which indicates that those IMAs have highly specific traffic characteristics with unique packet sizes and interarrival times.

This shows that I can build statistically accurate models for all of the feature sets I defined according to my research questions in Section 3.9.

4.4 Feature Selection

I perform feature selection analysis as described in Section 3.7 on all three of my feature sets. I aim to find the minimal feature set needed to achieve performance comparable to the baseline performance. I define baseline performance as the performance that I achieve using all features in each feature set. For example, I achieve an overall F1 score of 99.2% when classifying between IMAs using statistical features. Therefore, I consider the baseline performance for this scenario to be 99.2%.

4.4.1 All Features

All three of my algorithms rank IP addresses as the top two most important features. Furthermore, the pair of IP addresses is a unique identifier when identifying IMA

traffic. In fact, I can achieve an overall F1 score of 99.9% when only the following features are considered: source and destination IP addresses. Therefore, the minimal set of features (both statistical and categorical) that is needed for comparable performance consist of the IP addresses.

4.4.2 Statistical Features

Using the methods presented in Section 3.9, I discover the least number of statistical features required for each classification task. My results show that I can achieve baseline performance using only 20 features (out of 61 total statistical features) when classifying IMA vs Non-IMA (Figure 4.4). This minimal set of features used to achieve the baseline performance is listed in Table 4.13. When classifying between different IMAs, I achieve similar results using only eight features (Figure 4.14). Similarly, the features required for this task are listed in Table 4.14. For both of the classification tasks, features selected using Mutual Information achieve the highest F1 scores.

4.4.3 Packet Size/Timing Features

I discover the minimal set of packet size/timing features required for each classification task. My results show that I can achieve baseline performance using only 10 packet size/timing features (out of 13 total packet size/timing features) when classifying IMA vs Non-IMA (Figure 4.7). This minimal set of features used to achieve the baseline performance is listed in Table 4.15. When classifying between different IMAs, I achieve similar results using only 10 packet size/timing features (Figure 4.6). Similarly, the features required for this task are listed in Table 4.16. For both of the classification tasks, features selected using Mutual Information achieve the highest F1 scores.

4.5 Summary

In this section, I describe the results of my IMA encrypted traffic analysis. Firstly, this chapter reports on the characteristics of the dataset that I produced using the approach described in Section 3.1. Specifically, I detail the number of flows produced from each IMA as well as non-IMA traffic, the total size of all the flows generated, as

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.863-0.871	0.701-0.797	0.995-0.999	0.998-0.999
Recall	0.820-0.837	0.167-0.167	0.996-0.999	0.998-0.999
F1	0.840-0.852	0.720-0.755	0.996-0.999	0.997-0.999
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regress	Gradient Boost
Precision	0.867-0.910	0.405-0.715	0.405-0.931	0.997-0.999
Recall	0.784-0.933	0.500-0.824	0.5-0.739	0.992-0.997
F1	0.811-0.841	0.448-0.730	0.448-0.792	0.994-0.998

Table 4.4: Distinguishing IMAs from non-IMAs using all features.

well as the main protocols that the IMAs used.

Then, I include a report on the ports and the protocols that the IMAs require for functionality. The report follows the methods that is described in Section 3.2. The study shows that all IMAs use HTTP/HTTPS protocols and ports 80/443 to transmit, except for WhatsApp. Furthermore, it is found that all IMAs use TCP as blocking TCP traffic rendering all IMAs non-functional. This chapter also describes the number of flows that IMAs create and compare them to traffic patterns produced by more traditional services.

Next, I detail the results of a thorough evaluation of the ML models that were built. I report the performance achieved using various supervised learning algorithms for the two classification tasks studied and the feature sets used. Random Forest shows the best performance for all tasks and scenarios studied in this research. The results show that it is possible to achieve near-perfect scores when using all features and only statistical features, and significantly high results when using only packet size/timing features.

Lastly, I report the findings of my feature engineering analysis. Specifically, three different feature selection algorithms are tested to find the minimal set of features required for baseline performance for both classification tasks and all feature sets. It is possible to achieve baseline performance when using 10-20 statistical features and 10 packet size/timing features. This shows that by using these minimal feature sets, it is possible to significantly reduce the necessary feature set size to decrease computational complexity.

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.863-0.871	0.701-0.797	0.995-0.999	0.998-0.999
Recall	0.820-0.837	0.729-0.772	0.996-0.999	0.998-0.999
F1	0.840-0.852	0.720-0.755	0.996-0.999	0.997-0.999
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regression	Gradient Boost
Precision	0.867-0.910	0.405-0.715	0.405-0.931	0.997-0.999
Recall	0.784-0.933	0.500-0.824	0.5-0.739	0.992-0.997
F1	0.811-0.841	0.448-0.730	0.448-0.792	0.994-0.998

Table 4.5: Distinguishing between different IMAs using all features.

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.318-0.374	0.073-0.073	0.992-0.997	0.997-0.999
Recall	0.316-0.348	0.167-0.167	0.993-0.996	0.992-0.998
F1	0.315-0.353	0.101-0.101	0.993-0.996	0.995-0.999
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regression	Gradient Boost
Precision	0.229-0.470	0.405-0.684	0.405-0.934	0.994-0.996
Recall	0.171-0.208	0.500-0.792	0.500-0.741	0.982-0.988
F1	0.041-0.172	0.449-0.680	0.449-0.795	0.988-0.992

Table 4.6: Distinguishing IMAs from non-IMAs using strictly statistical features.

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.189-0.201	0.072-0.073	0.937-0.954	0.977-0.984
Recall	0.241-0.257	0.167-0.167	0.969-0.980	0.991-0.994
F1	0.127-0.133	0.101-0.101	0.953-0.966	0.984-0.989
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regression	Gradient Boost
Precision	0.580-0.706	0.025-0.192	0.016-0.016	0.974-0.983
Recall	0.512-0.672	0.167-0.167	0.167-0.167	0.990-0.994
F1	0.525-0.667	0.044-0.045	0.029-0.029	0.982-0.989

Table 4.7: Distinguishing between different IMAs using only statistical features.

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.873-0.898	0.072-0.073	0.918-0.925	0.939-0.946
Recall	0.829-0.847	0.167-0.167	0.905-0.917	0.914-0.924
F1	0.852-0.870	0.101-0.101	0.915-0.921	0.927-0.934
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regression	Gradient Boost
Precision	0.732-0.808	0.405-0.582	0.405-0.643	0.942-0.948
Recall	0.601-0.786	0.500-0.527	0.500-0.526	0.814-0.832
F1	0.621-0.784	0.448-0.518	0.448-0.508	0.860-0.875

Table 4.8: Distinguishing IMA vs Non-IMA using only packet size and timing features.

Model	Nearest Neighbor	Linear SVM	Decision Tree	Random Forest
Precision	0.484-0.496	0.01-0.188	0.670-0.699	0.754-0.774
Recall	0.830-0.842	0.167-0.172	0.885-0.902	0.909-0.919
F1	0.572-0.588	0.008-0.023	0.742-0.769	0.812-0.819
Model	Multi-Layer Perceptron	Naive Bayes	Logistic Regression	Gradient Boost
Precision	0.366-0.519	0.269-0.313	0.174-0.198	0.698-0.730
Recall	0.627-0.760	0.305-0.331	0.269-0.343	0.899-0.912
F1	0.368-0.517	0.073-0.131	0.085-0.143	0.752-0.775

Table 4.9: Distinguishing between different IMAs using only packet size and timing features.

Class	Discord	Messenger	Telegram	Teams
Precision	0.993-0.999	0.999-0.999	0.992-0.998	0.998-0.999
Recall	0.996-0.999	0.999-0.999	0.996-0.999	0.998-0.999
F1	0.995-0.998	0.999-0.999	0.996-0.999	0.997-0.999
Class	WhatsApp	Signal	Non-IMA	
Precision	0.995-0.999	0.999-0.999	0.996-0.999	
Recall	0.996-0.999	0.999-0.999	0.995-0.999	
F1	0.995-0.998	0.999-0.999	0.996-0.998	

Table 4.10: Identifying IMA traffic using all features.

Class	Discord	Messenger	Telegram	Teams
Precision	0.944-0.961	0.985-0.990	0.988-0.993	0.999-0.999
Recall	0.995-0.999	0.995-0.999	0.994-0.998	0.997-0.997
F1	0.969-0.979	0.990-0.994	0.992-0.995	0.998-0.998
Class	WhatsApp	Signal	Non-IMA	
Precision	0.915-0.957	0.998-0.999	0.991-0.995	
Recall	0.986-0.999	0.992-0.996	0.991-0.999	
F1	0.953-0.973	0.996-0.997	0.992-0.996	

Table 4.11: Identifying IMA traffic using only statistical features.

Class	Discord	Messenger	Telegram	Teams
Precision	0.641-0.672	0.720-0.741	0.898-0.902	0.995-0.995
Recall	0.885-0.920	0.956-0.974	0.918-0.925	0.917-0.918
F1	0.744-0.772	0.822-0.840	0.909-0.913	0.954-0.955
Class	WhatsApp	Signal	Non-IMA	
Precision	0.143-0.167	0.977-0.985	0.854-0.869	
Recall	0.860-0.916	0.987-0.993	0.902-0.917	
F1	0.247-0.282	0.983-0.989	0.877-0.891	

Table 4.12: Identifying IMA traffic using only packet sizing and timing features.

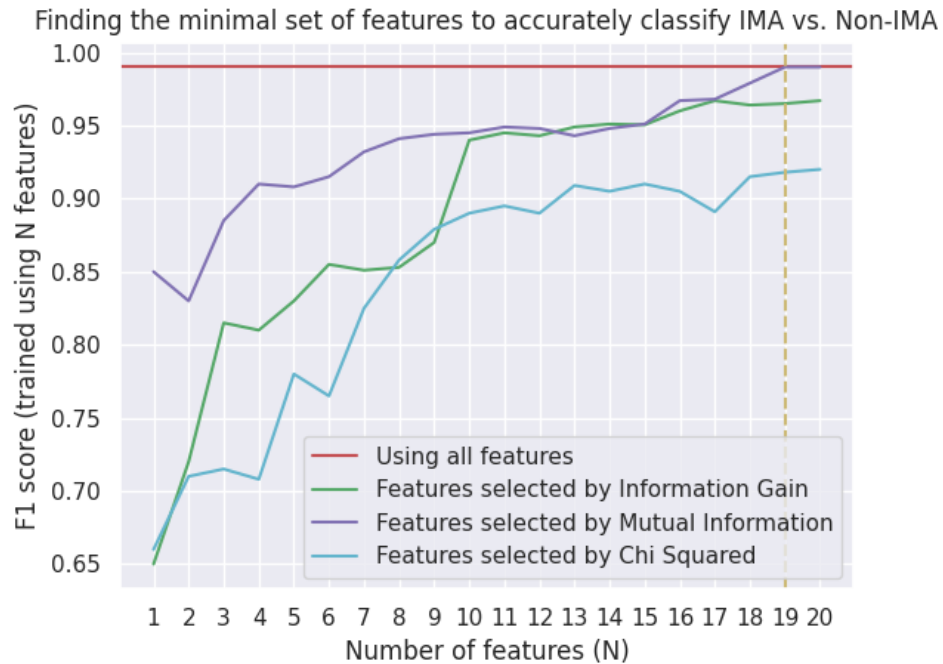


Figure 4.4: F1 score of models that distinguish between flows from different IMAs, trained on varying sets of statistical features selected by different feature selection algorithms.

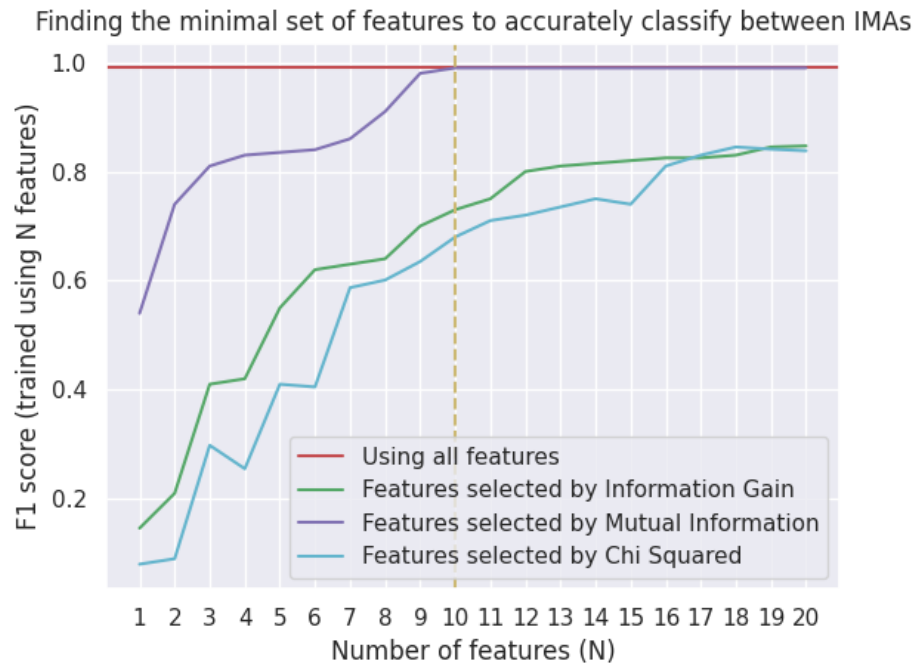


Figure 4.5: F1 score of models that distinguish IMA vs. Non-IMA traffic, trained on varying sets of statistical features selected by different feature selection algorithms.

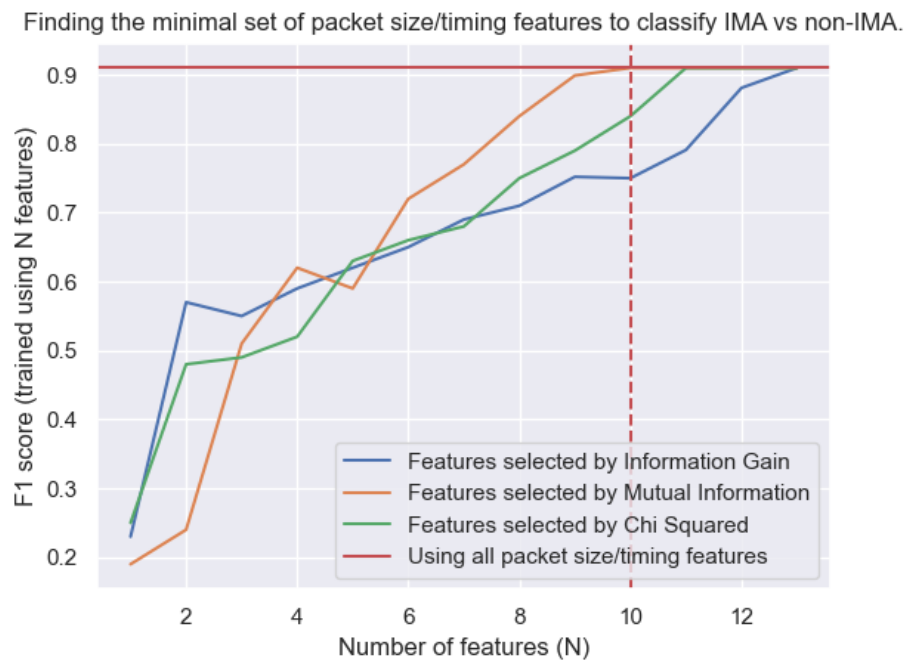


Figure 4.6: F1 score of models that distinguish IMA vs. Non-IMA traffic, trained on varying sets of packet size/timing features selected by different feature selection algorithms.

Feature
Number of unique destination IPs
TCP initial sequence number
TCP trip time for SYN, SYN-ACK
TCP maximum window size
IP maximum delta IP ID
TCP average window size
TCP packet ACK count
TCP packet sequence count
TCP max bytes in flight
TCP initial window size
Minimum packet size
Maximum interarrival time
IP maximum TTL
Packet stream asymmetry
Standard deviation in interarrival time
Number of packets received
Number of connections between source and destination IPs
Number of unique source IPs
Standard deviation in packet size
Byte stream asymmetry

Table 4.13: Statistical features selected by Mutual Information to build a classifier to distinguish IMA vs. Non-IMA, achieving the baseline performance as defined in Section 3.9.

Feature
TCP max bytes in flight
Number of unique destination IPs
Maximum packet size
Average packet size
Standard deviation in packet size
Number of connections between source and destination IPs
Number of unique source IPs
Maximum interarrival time

Table 4.14: Statistical features selected by Mutual Information to build a classifier to distinguish between IMAs, achieving the baseline performance as defined in Section 3.9.

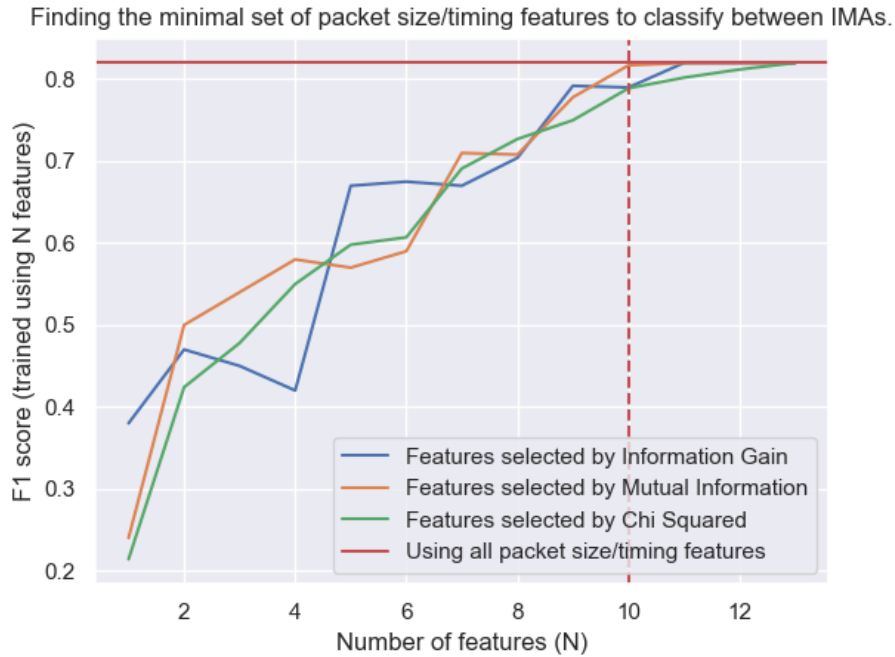


Figure 4.7: F1 score of models that distinguish between flows from different IMAs, trained on varying sets of packet size/timing features selected by different feature selection algorithms.

Feature
Maximum packet size
Minimum packet size
Maximum interarrival time
Standard deviation in interarrival time
Number of packets received
Bytes per second
Minimum interarrival time
Number of bytes received
Number of bytes send
Average packet size

Table 4.15: Packet size/timing features selected by Mutual Information to build a classifier to distinguish IMA vs. Non-IMA, achieving the baseline performance as defined in Section 3.9.

Feature
Maximum packet size
Average packet size
Standard deviation in packet size
Maximum interarrival time
Number of bytes received
Number of bytes sent
Average interarrival time
Standard deviation in interarrival time
Bytes per second
Number of packets received

Table 4.16: Packet size/timing features selected by Mutual Information to build a classifier to distinguish between IMAs, achieving the baseline performance as defined in Section 3.9.

Chapter 5

Conclusion And Future Work

Due to its wide use and impact, I study Instant Messaging Applications (IMAs) and their traffic characteristics. As discussed in previous chapters IMAs have been more and more widely used over the last five years. As such more functionalities are added to IMAs from text messaging to voice over IP to audio/video to file exchanges and more. These could make the IMAs also very attractive for malicious intents. On one hand, attackers could also use IMAs for their information exchange but they could also use IMAs for data theft. Thus understanding and analyzing IMA encrypted traffic requires further research. To this end, I first study how IMAs generate and maintain flows, and provide descriptive analysis. In particular, I build ML classifiers to differentiate IMA traffic from non-IMA traffic as well as to distinguish between different IMAs. Additionally, I report my findings on my feature engineering analysis.

I carefully select six most widely-used IMAs for my study: Discord, WhatsApp, Messenger, Teams, Signal, and Telegram. While they are all widely used and recognized, these IMAs have a diverse set of user bases, use cases, and functionalities.

As a major contribution to this research, I design and develop a methodology to generate labeled IMA traffic traces using Android emulators with Linux networking tools to help alleviate the severe lack of public datasets of IMA traffic. To make my traffic set realistic, I implement a number of user behavior patterns, backed by research, and emulate two-way conversations using Android emulators and custom scripts. I design a method of emulating synchronous conversations using typing delays. Then, I also implement a method of emulating asynchronous conversations using waiting delays. Finally, I use existing research to incorporate realistic patterns of opening/closing IMAs.

Using this method of data generation, I build a dataset of over 80 thousand flows generated by IMAs. I publicly release this dataset of over 1GB total size to the benefit of the wider network research community. This is the first publicly available dataset

of encrypted IMA network traffic.

To comprehensively profile IMA traffic, I analyze the ports and the protocols that IMAs use and require. Similar to most general smartphone applications, IMAs use HTTPS protocols over port 443. In fact, all IMAs, except WhatsApp, render non-functional when port 443 is blocked. While TCP is required for all IMAs, UDP is only required for Discord and Teams. Unlike traditional web services, such as browsers, IMAs do not use a large number of flows to send information. In fact, all IMAs use two main elephant flows through which they transfer the majority of the packets. While Signal and Telegram only use two flows, others use extra short bursts of flows to send information in parallel.

Next, using the dataset generated, I build a comprehensive set of ML models to profile the traffic from all of my IMAs. First, I extract 109 features from my traffic set using *Tranalyzer2* and I remove any biased features. I test and evaluate my models using three different feature sets- all features, statistical features, and packet size/timing features. Further, I perform two general classification tasks- IMA vs non-IMA classification, and between IMAs. Then, I combine these two types of classifiers to build a 7-class model that can identify IMA traffic from the ground truth. I achieve a 99.9% F1 score when using all features which means that I can identify specific IMAs using their traffic highly accurately. Then, I achieve a 99.9% F1 score when using only statistical features which indicates that IMA traffic has a highly unique statistical signature. Lastly, I achieve an 80% F1 score when only using packet size/timing features which indicates that IMA traffic can be identified accurately even when the traffic is completely obfuscated by VPN.

My last contribution to this research is my feature engineering analysis. Specifically, I utilize three different feature selection algorithms to evaluate the importance of features in my feature sets and find the minimal set of features needed without degradation in performance. I can achieve baseline performance when only the pair of IP addresses features. However, a set of 10-20 statistical features is required to identify IMAs without a drop in performance, and 10 packet size/timing features for the same results. Using a smaller feature set, it is shown possible significantly reduce my computational complexity and training time.

Future work would address the limitations described in Section 3.3.11. Specifically,

an easier to use approach could be developed to make the dataset scalable and the data generation process more accessible for non-practitioners. Furthermore, the traffic collection for asynchronous traffic could be extended over longer periods of time such that enough flows are generated to make the dataset balanced in terms of conversation types. To this end, more realistic user behavior patterns could be implemented for non-IMA traffic generation as well.

It is worth noting that the scope of this research includes analysis of traffic resulting from only the core functionality of IMAs sending text messages. To better understand IMA network traffic behavior, it would be beneficial to extend my analysis to include other functionalities. As discussed earlier, most IMAs are capable of a wide variety of functionalities, including sending files, voice calls, and even monetary transactions. Thus, future research will include collecting traffic from such interactions involving these extended IMA functionalities, and then perform further analysis to more comprehensively characterize IMA traffic.

While I selected a comprehensive set of IMAs for my study, it is not exhaustive. Ideally, the network research community could benefit from analyzing and studying traffic from every Instant Messaging mobile application. Therefore, future work could explore IMAs beyond the ones I studied in this research. Specifically, I limited my set to IMAs that are widely used in North America. However, there are specific IMAs that are specific to a geo-location, such as WeChat and KakaoTalk. To extend IMA traffic analysis to a global level, future work could explore IMAs used in specific regions.

Bibliography

- [1] Sina Keshvadi, Mehdi Karamollahi, and Carey Williamson. Traffic characterization of instant messaging apps: A campus-level view. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 225–232, 2020.
- [2] Jitendra Soni. Whatsapp pay: How to setup, send and receive money. *techradar*, Nov 2021.
- [3] Susan D’Agostino. How do college students use discord?, Apr 2023.
- [4] Peng Yang, Ning Zhang, Shan Zhang, Li Yu, Junshan Zhang, and Xuemin Shen. Content popularity prediction towards location-aware mobile edge caching. *IEEE Transactions on Multimedia*, 21(4):915–929, 2019.
- [5] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2019.
- [6] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2018.
- [7] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.
- [8] Wei Cai, Gaopeng Gou, Minghao Jiang, Chang Liu, Gang Xiong, and Zhen Li. Memg: Mobile encrypted traffic classification with markov chains and graph neural network. In *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 478–486, 2021.
- [9] Junzheng Shi, Chen Yang, Qingya Yang, Jialin Zhang, Mingxin Cui, and Gang Xiong. Spbiseq: An early-stage fingerprint generation method with high robustness for encrypted mobile application traffic. In *2021 IEEE 4th International Conference on Computer and Communication Engineering Technology (CCET)*, pages 185–193, 2021.
- [10] Andy Greenberg. Signal’s cryptocurrency feature has gone worldwide, Jan 2022.

- [11] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, page 245–256, New York, NY, USA, 2004. Association for Computing Machinery.
- [12] Ellen Isaacs, Candace Kamm, Diane J. Schiano, Alan Walendowski, and Steve Whittaker. Characterizing instant messaging from recorded logs. In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, page 720–721, New York, NY, USA, 2002. Association for Computing Machinery.
- [13] Martin Pielot, Rodrigo de Oliveira, Haewoon Kwak, and Nuria Oliver. Didn't you see my message? predicting attentiveness to mobile instant messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 3319–3328, New York, NY, USA, 2014. Association for Computing Machinery.
- [14] Zolboo Erdenebaatar, Riyad Alshammari, Nur Zincir-Heywood, Marwa Elsayed, Bis Nandy, and Nabil Seddigh. Instant messaging application encrypted traffic generation system. In *2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)*, 2023.
- [15] Zolboo Erdenebaatar, Riyad Alshammari, Nur Zincir-Heywood, Marwa Elsayed, Bis Nandy, and Nabil Seddigh. Depicting instant messaging encrypted traffic characteristics through an empirical study. In *The 32nd International Conference on Computer Communications and Networks (ICCCN 2023)*, 2023.
- [16] Zolboo Erdenebaatar, Riyad Alshammari, Nur Zincir-Heywood, Marwa Elsayed, Bis Nandy, and Nabil Seddigh. Analyzing traffic characteristics of instant messaging applications on android smartphones. In *2023 IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)*, 2023.
- [17] Qinglong Wang, Amir Yahyavi, Bettina Kemme, and Wenbo He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 433–441, 2015.
- [18] Hasan Faik Alan and Jasleen Kaur. Can android applications be identified using only tcp/ip headers of their launch time traffic? In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, page 61–66, New York, NY, USA, 2016. Association for Computing Machinery.
- [19] Mauro Conti, Luigi V. Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, page 297–304, New York, NY, USA, 2015. Association for Computing Machinery.

- [20] Gianluca De Luca Fiscone, Raffaele Pizzolante, Arcangelo Castiglione, and Francesco Palmieri. Network forensics of whatsapp: A practical approach based on side-channel analysis. In Leonard Barolli, Flora Amato, Francesco Moscato, Tomoya Enokido, and Makoto Takizawa, editors, *Advanced Information Networking and Applications*, pages 780–791, Cham, 2020. Springer International Publishing.
- [21] Hua Wu, Qiuyan Wu, Guang Cheng, and Shuyi Guo. Instagram user behavior identification based on multidimensional features. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1111–1116, 2020.
- [22] Junming Liu, Yanjie Fu, Jingci Ming, Yong Ren, Leilei Sun, and Hui Xiong. Effective and real-time in-app activity analysis in encrypted internet traffic streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17*, page 335–344, New York, NY, USA, 2017. Association for Computing Machinery.
- [23] Scott E. Coull and Kevin P. Dyer. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *SIGCOMM Comput. Commun. Rev.*, 44(5):5–11, oct 2014.
- [24] Alireza Bahramali, Amir Houmansadr, Ramin Soltani, Dennis Goeckel, and Don Towsley. Practical traffic analysis attacks on secure messaging applications. In *Proceedings 2020 Network and Distributed System Security Symposium*. Internet Society, 2020.
- [25] Moxie Marlinspike. Whatsapp’s signal protocol integration is now complete. *Signal*, Apr 2016.
- [26] Arash Asli. How facebook messenger bots are revolutionizing business. *Forbes*, Jun 2018.
- [27] Frank M. Shipman and Catherine C. Marshall. Ownership, privacy, and control in the wake of cambridge analytica: The relationship between attitudes and awareness. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20*, page 1–12, New York, NY, USA, 2020. Association for Computing Machinery.
- [28] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China, November 2019. Association for Computational Linguistics.

- [29] Afarin Pirzadeh. Emotional communication in instant messaging. In *Proceedings of the 2014 ACM International Conference on Supporting Group Work*, GROUP '14, page 272–274, New York, NY, USA, 2014. Association for Computing Machinery.
- [30] Fariba Haddadi and A. Nur Zincir-Heywood. Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. *IEEE Systems Journal*, 10(4):1390–1401, 2016.
- [31] Stefan Burschka and Benoît Dupasquier. Tranalyzer: Versatile high performance network traffic analyser. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2016.
- [32] Asela Jayatilleke and Parakum Pathirana. Smartphone vpn app usage and user awareness among facebook users. In *2018 National Information Technology Conference (NITC)*, pages 1–6, 2018.
- [33] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.
- [34] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.
- [35] Mahdi Dashtbozorgi and Mohammad Abdollahi Azgomi. A scalable multi-core aware software architecture for high-performance network monitoring. In *Proceedings of the 2nd International Conference on Security of Information and Networks*, SIN '09, page 117–122, New York, NY, USA, 2009. Association for Computing Machinery.
- [36] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986.
- [37] Guoping Zeng. A unified definition of mutual information with applications in machine learning. *Mathematical Problems in Engineering*, 2015:201874, Mar 2015.
- [38] Mary L McHugh. The chi-square test of independence. *Biochem Med (Zagreb)*, 23(2):143–149, 2013.

Appendix A

Appendix

A.1 Features extracted by *Tranalyzer2*

	Feature ID	Description
1	%dir	Flow direction
2	flowInd	Flow index
3	flowStat	Flow status and warnings
4	timeFirst	Date time of the first packet
5	timeLast	Date time of the last packet
6	duration	Flow duration
7	numHdrDesc	Number of different headers descriptions
8	numHdrs	Number of headers (depth) in header description
9	hdrDesc	Headers description
10	srcMac	source MAC address
11	dstMac	destination MAC address
12	ethType	Ethernet type
13	ethVlanID	VLAN IDs
14	srcIP	source IP
15	srcIPCC	source IP country
16	srcIPOrg	source IP organization
17	srcPort	source port
18	dstIP	destination IP
19	dstIPCC	destination IP country
20	dstIPOrg	destination IP organization
21	dstPort	destination port
22	l4Proto	Layer 4 protocol

	Feature ID	Description
23	macStat	MAC statistics
24	macPairs	MAC pairs
25	srcMac	dstMac numP source/destination MAC addresses, number of packets
26	srcManuf	dstManuf source/destination MAC manufacturers
27	dstPortClassN	port-based classification of the destination port number
28	dstPortClass	classification of the destination port
29	numPktsSnt	number of packets sent
30	numPktsRcvd	number of packets received
31	numBytesSnt	number of bytes sent
32	numBytesRcvd	number of bytes received
33	minPktSz	minimum packet size
34	maxPktSz	maximum packet size
35	avePktSize	average packet size
36	stdPktSize	standard packet size
37	minIAT	minimum inter-arrival time
38	maxIAT	maximum inter-arrival time
39	aveIAT	average inter-arrival time
40	stdIAT	standard inter-arrival time
41	pktps	sent packets per second
42	bytps	sent bytes per second
43	pktAsm	packet stream asymmetry
44	bytAsm	byte stream asymmetry
45	tcpFStat	multiple values possible for TCP flag stat
46	ipMindIPID	IP Minimum delta IP Identification
47	ipMaxdIPID	IP Maximum delta IP Identification
48	ipMinTTL	IP Minimum Time to Live (TTL)

	Feature ID	Description
49	ipMaxTTL	IP Maximum Time to Live (TTL)
50	ipTTLChg	IP TTL Change Count
51	ipTOS	IP Type of Service
52	ipFlags	IP flags
53	ipOptCnt	IP options count
54	ipOptCpCl	Num the aggregated IP options are coded as a bit field in hexadecimal notation where the bit position denotes the IP options
55	ip6OptCntHH	IPv6 aggregated hop by hop dest option counts
56	ip6OptHH	IPv6 hop by hop destination options
57	tcpISeqN	TCP initial sequence number
58	tcpPSeqCnt	TCP packet sequence count
59	tcpSeqSntBytes	TCP sent sequence diff bytes
60	tcpSeqFaultCnt	TCP sequence number fault count
61	tcpPAckCnt	TCP packet ACK count
62	tcpFlwLssAckRcvdBytes	TCP flawless ACK received bytes
63	tcpAckFaultCnt	TCP ACK number fault count
64	tcpInitWinSz	TCP initial effective window size
65	tcpAveWinSz	TCP average effective window size
66	tcpMinWinSz	TCP minimum effective window size
67	tcpMaxWinSz	TCP maximum effective window size
68	tcpWinSzDwnCnt	TCP effective window size change down count
69	tcpWinSzUpCnt	TCP effective window size change up count
70	tcpWinSzChgDirCnt	TCP effective window size direction change count
71	tcpWinSzThRt	TCP packet count ratio below window size WINMIN

	Feature ID	Description
72	tcpFlags	TCP aggregated protocol flags (FIN, SYN, RST, & PSH, & ACK, URG, ECE, CWR)
73	tcpAnomaly	TCP aggregated header anomaly flags
74	tcpOptPktCnt	TCP options packet count
75	tcpOptCnt	TCP options count
76	tcpOptions	TCP aggregated options
77	tcpMSS	TCP maximum segment size
78	tcpWS	TCP window scale factor
79	tcpMPTBF	MPTCP type bitfield
80	tcpMPF	MPTCP flags
81	tcpMPAID	MPTCP address ID
82	tcpMPdssF	MPTCP DSS flags
83	tcpTmS	TCP time stamp
84	tcpTmER	TCP time echo reply
85	tcpEcI	TCP estimated counter increment
86	tcpUtm	TCP estimated up time
87	tcpBtm	TCP estimated boot time
88	tcpSSASAATrip time	(A) TCP trip time SYN, SYN-ACK,(B) TCP SYN-ACK, ACK
89	tcpRTTAckTripMin	TCP ACK trip minimum
90	tcpRTTAckTripMax	TCP ACK trip maximum
91	tcpRTTAckTripAve	TCP ACK trip average
92	tcpRTTAckTripJitAve	TCP ACK trip jitter average
93	tcpRTTSseqAA	(A) TCP round trip time SYN, SYN-ACK, ACK (B) TCP round trip time ACK-ACK
94	tcpRTTAckJitAve	TCP ACK round trip average jitter
95	tcpStates	TCP states
96	icmpStat	status
97	icmpTCcnt	type code count

	Feature ID	Description
98	icmpBFTypH	TypL Code aggregated type H(i 128),L(j 32) and code bitfield
99	icmpTmGtw	time/gateway
100	icmpEchoSuccRatio	echo reply/request success ratio
101	icmpPFindex	parent flow index
102	connSip	number of unique source IPs
103	connDip	number of unique destination IPs
104	connSipDip	number of connections between source and destination IPs
105	connSipDprt	number of connections between source and destination port
106	connF	the f number, experimental: connSipDprt/connSip

Table A.1: The list of features extracted by Tranalyzer2

A.2 Improving my packet size/timing features classifier

In Section 4.3.2, I presented the results of my Random Forest model that classifies between IMAs using only packet size/timing features. Table A.2 show the scores of this classifier for each class and I see that I get an overall average F1 score of over 81%.

However, I see that the classes Discord, Telegram, and WhatsApp have low precision and the confusion matrix for this classifier (Figure A.1) shows that these three classes are often mistaken for one another. Furthermore, the other three classes- Messenger, Teams, and Signal- exhibit unique characteristics as indicated by F1 scores that are higher than 0.91%. Therefore, I concluded that my original 6-class Random Forest model fails to capture and model the differences between Discord, Telegram, and WhatsApp.

Then, I investigated whether I could improve my results for this classification task. To determine if I can even accurately distinguish between Discord, Telegram, and WhatsApp, I built another Random Forest model for the three "culprit" IMAs. I will refer to this model as my "3-class model". Table A.3 shows the result of this 3-class and I can see that I achieve high F1 scores (average F1 score of 90.1%). Therefore, I concluded that my original 6-class model simply fail to capture the statistical differences between these three classes.

Furthermore, I rank the features by their importance for my original 6-class model and for my 3-class model. Figure A.2 shows that my 6-class model generally relies on traffic volume features (such as the number of bytes received and packet sizes) more than timing features (such as standard interarrival time), while the opposite is true for my 3-class model.

To improve the overall performance for this classification task, I combine my original 6-class model with my 3-class model. Figure A.3 describes how I combined these two classifiers. For each input, I first use my 6-class classifier. Then, if the output of this first classifier is Telegram, WhatsApp, or Discord, I reclassify the input using my 3-class model.

Figure A.4 presents the result of this combined model and compares it with my original 6-class model. I can see that the combined model shows clear improvements as evidenced by an increase in F1 scores for Discord, Telegram, and WhatsApp.

	IMA Class	Precision	Recall	F1
0	Discord	0.668-0.713	0.853-0.883	0.754-0.788
1	Messenger	0.833-0.886	0.956-0.968	0.893-0.923
2	Telegram	0.675-0.808	0.895-0.943	0.787-0.857
3	Teams	0.993-0.995	0.907-0.913	0.949-0.952
4	Whatsapp	0.276-0.364	0.82-0.877	0.417-0.504
5	Signal	0.96-0.986	0.973-0.987	0.967-0.983

Table A.2: Classifying between IMAs using packet/timing features.



Figure A.1: Classifying between IMAs using packet/timing features.

IMA Class	Precision	Recall	F1
Discord	0.93-0.947	0.867-0.885	0.903-0.914
Telegram	0.91-0.972	0.913-0.972	0.933-0.952
Whatsapp	0.762-0.824	0.839-0.914	0.82-0.844

Table A.3: Classifying between Discord, Telegram and WhatsApp.

6-class model		This model	
Feature	Importance	Feature	Importance
numBytesRcvd	0.148000	aveIAT	0.189000
stdPktSize	0.134000	stdIAT	0.174000
duration	0.103000	maxIAT	0.110000
maxIAT	0.091000	avePktSize	0.097000
numPktsSnt	0.082000	stdPktSize	0.073000
aveIAT	0.082000	numPktsRcvd	0.065000
avePktSize	0.078000	numBytesRcvd	0.062000
maxPktSz	0.066000	maxPktSz	0.060000
numPktsRcvd	0.062000	numPktsSnt	0.059000
bytps	0.053000	bytps	0.059000
numBytesSnt	0.048000	numBytesSnt	0.051000
stdIAT	0.044000		

Figure A.2: Comparing the importance of features for my 6-class and 3-class models.

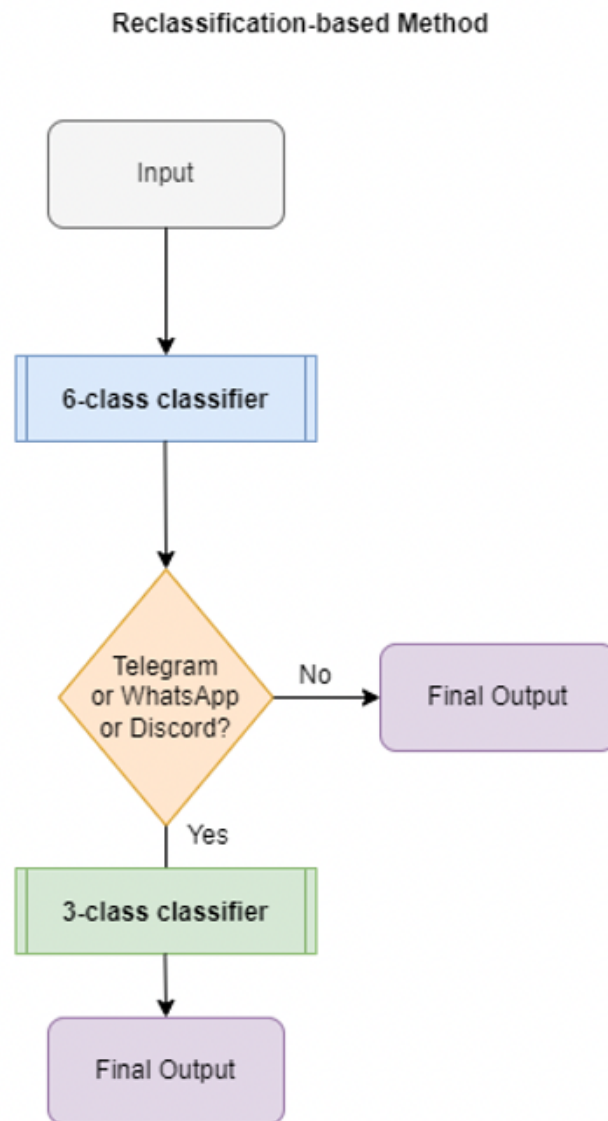


Figure A.3: Combining my 6-class model with my 3-class model.

Original 6-class model

IMA Class	Precision	Recall	F1
0 Discord	0.697051	0.873460	0.775348
1 Messenger	0.853828	0.953368	0.900857
2 Telegram	0.787852	0.910478	0.844738
3 Teams	0.993737	0.910920	0.950528
4 Whatsapp	0.313529	0.876000	0.461782
5 Signal	0.979127	0.973585	0.976348

Combined model

IMA Class	Precision	Recall	F1
0 Discord	0.745387	0.904815	0.817400
1 Messenger	0.853828	0.953368	0.900857
2 Telegram	0.794533	0.916582	0.851205
3 Teams	0.993737	0.910920	0.950528
4 Whatsapp	0.319386	0.916000	0.473630
5 Signal	0.979127	0.973585	0.976348

Figure A.4: My combined model performs significantly better than the original model.