IDENTIFYING NETWORK TRAFFIC SIGNATURES FOR TEXTING VIA INSTANT MESSAGING APPLICATIONS: A MACHINE LEARNING APPROACH

by

Srivathsan Thirumurugan

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

at

Dalhousie University
Halifax, Nova Scotia
August 2024

Table of Contents

List of Tables

List of Figures

Abstract

As Instant Messaging Applications become more popular, they can be used for a variety of purposes, including sending files, making voice and video calls as well as texting in groups and private channels for both personal and professional purposes. I aim to explore network traffic signatures in instant messaging applications for network monitoring and analysis purposes. To achieve this I designed and developed a framework to automatically generate and capture traffic from the seven most used and popular instant messaging applications, namely Discord, Messenger, Signal, Skype, Teams, Telegram, and WhatsApp. I have analyzed and discovered patterns of texting via IMAs from the perspective of texting behaviour such as synchronous and asynchronous communications. Moreover, their traffic is generated and captured with different types of user communication based on the number of participants such as private texting with two users, group texting with three users, and group texting with four users. The resulting end-to-end encrypted traffic is analyzed using a machine-learning-based approach to traffic metadata without using deep packet inspection. Evaluations show that it is possible to identify between private, groups with different users for asynchronous and synchronous instant messaging applications. This in return could help better planning and management of the network operations for user quality of service.

List of Abbreviations

IMA	Instant Message Application
HTTP	HyperText Transfer Protocol
SSH	Secure Shell
VPN	Virtual private network
QR	Quick Response
HD	High definition
DEMA	Deactivated Except Messenger Account
PIN	Personal Identification Number
WPM	Words Per Minute
ETL	Event Trace Log
PCAP	Packet capture
PCAPNG	Packet capture Next Generation
PC	Personal Computer
GUI	Graphical User Interface
IP	Internet Protocol
SYN	Synchronize
SYN-ACK	Synchronize - Acknowledgement
ACK	Acknowledge
XGBoost	Extreme Gradient Boosting

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor and co-supervisor, Dr. Nur Zincir-Heywood, and Dr. Riyad Alshammari, for their motivation, guidance, support, and feedback throughout this research. Their support played a key role in bringing this work to completion.

I am also thankful to Dalhousie University for providing a learning space to meet and work with experts from various fields, and to everyone from the Network Information Management and Security (NIMS) research lab for accompanying me throughout this journey.

Finally, I would like to thank my parents and friends for their unwavering support and encouragement, which have been the driving force behind this research.

Chapter 1

Introduction

With the rise of a number of employers/employees offering/choosing to work from home (more than 50% when compared to during COVID lockdown) [1], the incorporation of instant messaging into the workplace is done as an additional means of communication. Most instant messaging applications (IMAs) have audio, and video call functionalities along with the ability to send text messages. Additionally, within messaging, there are private (one-to-one) and group (many-to-many) functions. Maina et al. [2] focused on the Instant messaging application for the workplace by discussing the advantages and disadvantages of the instant messaging functions. Maina et al. concluded that one of the main reasons why IMAs were so popular was the texting functionality of IMAs in real-time without face-to-face meetings. This is still considered to be one of the main reasons for using instant messaging in the workplace. This in return increases IMA-based traffic on an organization's network and necessitates further research to understand the different usage behaviours in such traffic for better network/service operations and management.

To this end, Ede et al. [3] proposed a semi-supervised machine learning-based approach for fingerprinting mobile applications based on their encrypted network traffic. Pektas et al. [4], focused more on the IMA traffic and used a machine learning-based approach to differentiate IMA traffic from non-IMA traffic. In a more recent work [5], Erdenebaatar et al. used a machine learning-based approach to identify IMA vs. Non-IMA traffic as well as identify different IMAs within the IMA traffic. In particular, they focused on the texting function of IMAs for only 2 users. While these works are important as the initial steps to better understand the different usage patterns of IMAs in network traffic, more research is still needed to identify network traffic signatures for different functionalities of IMAs.

In this thesis, my research objective is to explore whether it is possible to identify network traffic signatures specifically for different texting behaviours that can be found in IMAs traffic. To achieve this, I designed and developed a framework to automatically generate and capture traffic from the seven popular IMAs, namely Discord, Messenger, Signal, Skype, Teams, Telegram, and WhatsApp. My framework involves generating and capturing realistic data using texting via these IMAs from individual devices instead of virtual machines. For realistic texting behaviour, I have employed various techniques based on existing research. I have analyzed and discovered patterns of texting via IMAs from the perspective of different texting behaviours such as synchronous and asynchronous texting as well as types of user communication with different numbers of users, namely private texting with two users, group texting with three users, and group texting with four users.

The resulting end-to-end encrypted traffic is analyzed using a Machine Learning (ML) approach to traffic metadata without deep packet inspection. For this analysis, four ML models are used, namely Random Forest, Naive Bayes, Multi-Layer Perceptron, and Extreme Gradient Boosting. My results show that it is possible to identify network signatures for different texting behaviours by just using network traffic metadata without using deep packet inspection. To this end, drilling deeper into the best-performing Random Forest model enables the discovery of unique network signatures for different texting behaviours, namely synchronous, asynchronous, and different numbers of users.

Thus, the new research contributions of this thesis include:

1. Automating the process of generating and capturing encrypted texting traffic on individual devices for conditions including two, three, and four users for the aforementioned IMAs.

2. Automating the process of generating and capturing encrypted texting traffic for the above users and IMAs under synchronous and asynchronous communications.

3. Discovering the Network Signatures of different IMA traffic to identify different texting behaviours from a number of users to synchronous and asynchronous communications.

4. Designing, developing, and evaluating a comprehensive framework to deliver the above activities on the seven IMAs for traffic analysis purposes.

The rest of the thesis is organized as follows: Chapter 2 summarizes the existing literature in this field. Chapter 3 introduces the methodology describing the IMAs used and the framework built around which different texting behaviours are integrated, how they are captured, filtered, and exported into flows to extract metadata as well as the Machine Learning Models used. Chapter 4 presents the performance metrics, and the feature sets utilized as well as all the evaluations conducted and the network signatures obtained. Finally, conclusions are drawn and the future work is discussed in Chapter 5.

Chapter 2

Literature Review

The growth of mobile applications has led to an increase in encrypted network traffic, which has posed a challenge for application detection using network traffic. Traditional fingerprinting methods are inadequate without the knowledge of the applications due to the dynamic mobile environment where applications are usually installed, updated, or uninstalled. To overcome these limitations Ede et al. [3] proposed a ”FLOWPRINT” a semi-supervised model for fingerprinting mobile applications from encrypted network traffic. The ”FLOWPRINT” doesn’t rely on prior knowledge of the applications but rather identifies the pattern created by the network traffic. This was done by grouping encrypted TCP/UDP flows based on the app’s destinations and or any features that correlated to the destination features. Ede et al. has utilized various datasets from different sources where datasets are obtained under different circumstances, namely ReCon which consists of various Android applications from the Google Play Store including the same applications of different versions over eight years, Cross Platform which is not only has user-generated data from android but also from iOS devices, Andrubis which also only contains data from android applications but this also has dataset for the potentially harmful applications that were classified by the VirusTotal, as the final addition the author used the browser dataset by scraping Alexa websites on an android 6.0.1 device. Despite using datasets from varying working conditions, time, and environments such as Android and iOS, the author could detect both seen and previously unseen applications with an accuracy of 89.2% and detect applications within 300 seconds with the F1 score of 82.4%. Their approach showed the need for an increase in security and application management in mobile networks and proposed the need for VPNs as an additional user safeguard. The author did not focus on any specific collection of messaging applications or messaging application traffic with more than 2 users.

Pektaş et al. [4] have presented a classification method by utilizing the existing public NIMS1 [6] dataset which contains the network traffic that belongs to 20 application categories such as SFTP, Telnet, SCP, HTTP, SSH, etc. In addition to the existing dataset, new network traffic of the instant messaging application was also collected. The new data collection is done where a smartphone is connected to a hotspot hosted by a Windows laptop and with the use of the Wireshark sniffing tool all the traffic originating from the laptop is captured. Finally, the NetMate tool is used as a means of feature extraction to extract statistical features. As the dataset consisted of imbalanced datasets, the author utilized a 10-fold stratified K-Folds validator split for their binary classification cross-validation approach. The author achieved an average F1-Score of 99%.

Alan et al. [7] aimed to use the TCP/IP headers of the network traffic instead of the traditional Deep Packet Inspection or analysis utilizing the HTTP header, to do that the author collected data from 86,109 application launches total of 1,595 applications by running them repeatedly on 4 distinct devices. The author used 2 Android mobile and 2 Android tablets as the devices for collecting data. The author were able to create a model by training and testing using the first 64 packets of its packet size from the same device, which proved it achieved a classification result of 88%. However, this was dropped to 38% and 67% when trained with different OS and devices. Despite the author working across various devices and applications, none have specifically addressed messaging applications with varying numbers of users.

Erdenebaatar et al. [8] proposed a new framework for data collection, and analysis for 6 Instant Messaging Applications. The framework was built utilizing the Android studio virtual machine for emulating the text asynchronous and synchronous texting between the users. The message input and sent actions were coded using the ADB (*Android Debug Bridge*) functions. With the help of Netstat, they filtered the network traffic of a particular application and stored it as *PCAP* files [8]. The two-way traffic data collection, i.e. when text was sent and received by two users done for the 6 IMAs including WhatsApp, Team, Telegram, Discord, Messenger, and Signal [9]. They utilized synchronous and asynchronous text-based communications. They used 8 ML models and reported that the Random Forest model was the best-performing model in their evaluations. They were able to achieve more than 99% of the F1 score.

Although various IMAs were analyzed in their work, texting with more than 2 users and creating more realistic emulations as a means to mimic real-world behaviours were not the focus of their research [5].

Wang et al. [10] aimed to demonstrate that even under the presence of encrypted network traffic an attacker can determine the user's behaviour. To support their claim, the author utilized a series of mobile devices running either iOS or Android OS to collect data from wide categories of applications while performing various actions. The author has also included IMAs which he referred as 'Online Chatting' which included Facebook's messenger, Tencent QQ, and Snapchat while sending and receiving text and pictures. The author was able to achieve a best score of 87.23% when using all of the features and a worse case score of 68.59% using selected features. This work does include some messaging applications, but this does not include some of the most used IMAs and IMAs with 3 or more users.

Vu et al. [11] used a different method for their analysis of the encrypted traffic, where the author took the time series as a feature set for their train and test dataset. Along with their time series feature set the author has employed a deep learning-based technique using LSTM (Long Short-Term Memory) nodes. From the entire time series attributes, the author have extracted the attributes that are significant by analyzing the receiving packets. The author has utilized the VPN and Non-VPN dataset which was created by the Canadian Institute of Cybersecurity where the data was generated and collected under a network transmission between users 'Bob' and 'Alice'. For the evaluation of their deep learning, the author have utilized precision, recall, and F1 score as their choice of metric for the model which was trained with 80% of the dataset, and the rest is used as testing. For each of their epoch in the deep learning model, the author have used the 10% of the dataset as a validation dataset. From their findings, in their model, the return for accuracy per feature size starts to diminish after 55 feature size with a continuous increase in training and testing at and after 205 feature size. By analyzing the author was able to decrease the flow size from 20 to 2 and increase in accuracy by $\sim$2%. The author achieved an F1 score of 98% for all of their classes such as chat, email, VPN-chat, VPN-P2P, VPN-VoIP, etc. The inclusion of more data from IMA such as VoIP and P2P with VPN was present, but did work with different IMA and test their proposed model.

Liu et al. [12] used the Multi-attribute Markov Probability Fingerprints for classifying encrypted traffic. Utilizing critical features such as 'length block sequence' which effectively captures the time-series packets. 18 different application of various categories such as mailing, music streaming, browser, and messaging were used in the data collection from which flows were extracted combined to give a dataset of 950,000+ flows. As for the results, the author were able to achieve a True Positive Rate (TPR) of 96.4% and a False Positive Rate (FPR) of 0.2%. Despite using various types of applications it produced good results, the Tencent QQ was the only messaging application used, and judging by any lack of additional IMAs the Tencent QQ was just included for the sake of having a messaging application and was not attempted to explore.

Zou et al. [13] proposed a Convolutional Neural Network as a means for packet feature extraction, as for the time sequence feature extraction which is done on a flow level using a Long Short-Term Memory (LSTM). The author has utilized a public dataset published by the University of New Brunswick, where the author has relabeled the existing labels to emails, chat, streaming, file VoIP, and P2P for both VPN and non-VPN types. The raw data consists of 25GB of the network traffic mentioned in a pcap format. Three consecutive packets at a random location in a flow are made and split as 80% and 20% for training and testing respectively. The recurrent layer is introduced using an LSTM where all the states are initialized to be zero. The three 256-dimension packet feature vectors are sent to the LSTM cell, furthermore, the hidden unit in the LSTM is set to 256. In the aim of preventing the overfitting problem in LSTM, output probability is set to 0.8 and finally accompanied by softmax which is used as an activation function. As for the evaluation metrics, the author have used precision and recall for both their CNN and CNN-LSTM. The author were able to create a model that can automatically extract features from both the packet level and flow level. The author were able to achieve a score of over 95% in precision and recall as a best case for both the P2P and VPN VoIP for the CNN and CNN-LSTM and between 70% and 85% for both precision and recall as the worst case from chat. Similar to the previous literature the use of VPN and Non-VPN for the same type of VoIP and P2P is also present here along with other network traffic types, the use of different chats with more than two users or asynchronous and synchronous texting via IMAss is not present, and mentioned. The author also lacks variation in the number of IMAs used.

Casas et al. [14] proposed a WebScanner. They captured the network traffic of the Chrome web browser accessing the top most visited sites from Alexa's top-sites list for the desktop and Android smartphone platforms. Additionally, encrypted traffic from YouTube, Facebook, Amazon, and BBC News was captured from a web browser and also from their application under for their training and testing purposes. They were able to perform the classification of applications with an F1 score greater than 80%. Even though they worked on identifying encrypted network traffic in their research, the inclusion of any kind of IMA was not the focus.

Ren et al. [15] proposed a model that can classify encrypted application traffic with their TCP-stream layer attributes such as TCP stream length and SSL/TLS handshake message type. Ren et al. used data collection equipment consisting of a Wi-Fi Access Point and wireshark was used as a means to collect communication access points and the Internet. With the help of volunteers who would connect to the access point and collect the traffic generated by the smartphone applications. The author used applications such as Bilibili, QQ, WeChat, TikTok, and Weibo making a good collection of applications with applications capable of video transfer, browsing content, application behaviour, and chat functionality. Random forest was utilized as a classification model, where their proposed system was able to achieve 99.3% of True Positive Rate and 0.2% of False Positive Rate. There isn't a presence of a wide range of messaging applications even when focusing on only private chat.

Jay et al. [16], aims to explore the feasibility of detecting the malware without decrypting HTTPS traffic using machine learning. Jay et al. defined their feature model by analyzing the data logs generated. They categorized the feature sets into three namely, connection features, SSL features, and certificate features. The author has used publicly available datasets such as honeypots. Extreme Gradient Boosting is used with the dataset from the previously mentioned feature sets to classify the network traffic. Text messaging was not included nor discussed in their work.

Sun et al. [17] created automatically identifying applications utilizing mobile devices' encrypted traffic with the help of a custom system designed to collect, pre-process, and filter the traffic, and cluster the HTTP to extract the fingerprints. The application's traffic was captured under different user-interacting activities. For their use case, the author has chosen some of the popular social applications from China such as Weibo, weixin, and Zedge. To capture the network traffic the author has connected the smartphone of choice to a Linux computer where the Wireshark is utilized as a means to capture the data generated by the smartphone. For evaluating the fingerprinting mechanism that was developed, the authors conducted 4 different experiments, where for the first experiment, the generated fingerprints were used to match the corresponding applications in a dataset without background traffic and all application traffic. For the second experiment, the effectiveness of each app's fingerprint was tested separately along with data collected from each application specifically. For the third experiment, the fingerprints of one application were used to match the traffic data of the other applications, which will tell if the generated fingerprints work as expected against unintended applications. The fourth experiment is a repeat of the first step but along with background traffic to mimic real-world scenarios better. Messaging applications are included as their primary focus of testing, but the inclusion of traffic generated from more than 2 users is not investigated.

Cha et al. [18] explored the impact of encrypted traffic on network performance to improve IDS (Intrusion Detection System). The proposal is to use CART classification which is supported by the tests done based on the real-world dataset. The dataset used was captured from the Hacker 2013 honeypot competition. Based on comparing the 4 classification algorithms namely; Naive Bayes, Support Vector Machine, CART, and AdaBoost, it was found that CART has an accuracy of 99.9% and also performs 2.9 times more efficiently. The inclusion of Instant Messaging application traffic was absent.

Li et al. [19] proposed FusionTC as a means for encrypted traffic identification, instead of the existing single-modal feature pattern extraction approaches. FusionTC on the other hand follows a more multi-modal approach, by following a classification based on feature fusion of flow sequence. To achieve this, the author captured their dataset which is obtained from mobile phones by imitating the behavior of application users. 15 mainstream applications were installed and tested 25 times for about 10 minutes. Applications with audio and video functions were also included in their data collection. FusionTC is proposed as an enhancement specifically because it comprises of two-level subclassifiers for performing multi-modal feature fusion on a decision level by using an upgraded stacking method. From using FusionTC on an application built by the author, the results were shown to be 3.2% better than the state-of-the-art approaches that are currently employed. The author has utilized real devices to capture their data, but messaging application involving more than 2 users was not explored.

Wang et al. [20] has done a comparative study on 10 different algorithms that are typically used for malicious traffic detection. Three different datasets are used for this comparative study namely; IoT honeypot data of 2017-2018 comprising of data traffic of one and half years, a sample of Internet traffic from Japan to the USA collected over different periods, UNIBS dataset comprising 3 consecutive days collected in their university campus. Two experiments were done in total, each with a specific target ideology. The experiments focus on analyzing the statistical numerical feature sets and algorithms using a mixed dataset. It was found that RF, XGBoost, C4.5, AdaBoost, CART, and KNN performed specifically well, respectively in the mentioned order. Encrypted traffic classification was the focus, but Instant Messaging Applications were not included in their classifications.

Table 2.1: Literature Overview

Reference	Type of App	IMAs Used	Machine Learning Models	Platform	Dataset	Private and Group	Simulation	Emulation	Real vs Virtual devices
Ede et al. [3]	*	*	1	Android and iOS	ReCon, Cross Platform, Andrubis	*	*	*	*
Erdenebaatar et al. [5]	IMA, Non-IMA	Discord, Messenger, Signal, Teams, Telegram, and WhatsApp	8	Android	Generated	-	-	✓	V
Pektaş et al. [4]	IMA, Non-IMA	Skype, Telegram Viber, wechat and WhatsApp	3	Android	NIMS dataset [6]	-	-	-	V
Vu et al. [11]	Chat, Non-IMA	*	1	*	UNSW-NB15, NSL-KDD	-	-	-	V
Zou et al. [13]	Chat, Non-IMA	*	1	*	ISCX	-	-	-	V
Alan et al. [7]	*	*	1	Android	Generated	-	✓	-	R
Wang et al. [10]	IMA, Non-IMA	Messenger, Tencent QQ and Snapchat	1	Android and iOS	Generated	-	-	✓	R
Liu et al. [12]	IMA, Non-IMA	Weibo, Tencent QQ	1	*	Generated	-	-	-	R
Ren et al. [15]	IMA, Non-IMA	WeChat and Weibo	1	Mobile device	Generated	-	✓	-	R
Casas et al. [14]	Non-IMA	-	1	Desktop and Smartphone	Generated	-	✓	-	R
Sun et al. [17]	IMA, Non-IMA	Weixin and Weibo	2	Android	Generated	-	✓	-	R
Li et al. [19]	*	*	2	Android	Generated	-	✓	-	R
Cha et al. [18]	*	*	4	*	*	-	-	-	V
Wang et al. [20]	*	*	10	*	Hacker 2017 to 2018 honeypot competition	*	*	*	*
Shah et al. [16]	*	*	1	*	Traffic from Honeypots	-	-	-	V
My Proposal	IMA	Discord, Messenger, Signal, Skype, Teams, Telegram, and WhatsApp	4	Windows PCs	Generated	✓	-	✓	R

Legend:

- * (not mentioned)
- - (not used)
- R (Real Devices)
- V (Virtual Devices)

Table 2.1 gives an overview of the literature summarized in this Chapter in terms of their datasets, approaches, platforms, and IMA included. This enables me to put my thesis research into the context of the literature.

2.1 Summary

Various types of analysis and research are done for instant messaging applications, from working with the existing publicly available dataset to creating a new framework and capturing network traffic via emulation. But work on instant messaging applications without using virtual machines is very limited, and to add to that there isn't any existing framework that was built to mimic real-world texting behaviours such as Synchronous and Asynchronous texting between 2 or more users. To bridge the gap a new framework has been made to shed light on Instant Messaging Application characteristics that best represent real-world communication of the various instant messaging applications based on different numbers of users and their behaviours.

Chapter 3

Methodology

This chapter introduces the methodology followed and the framework proposed in this research to analyze the private and group messaging traffic of Instant Messaging Applications (IMAs). First I will go over in detail of the different IMAs used for this research. Followed by the two types of texting behaviours used, and techniques employed in making a realistic conversation. The framework created is explained along with how they use private and group texting using different PCs. Finally, I have also explained how the generated data is captured and flows are extracted.

3.1 Instant Messaging Applications Used

Seven Instant Messaging Applications are used in this research. All of them are account-oriented rather than platform-oriented. In other words, for users to communicate with each other using these IMAs, they need to have user accounts for each IMA they would like to use. However, each user can have a different operating system for the particular IMA they choose. In this thesis, various source and login methods were utilized to set up the IMAs. These are discussed in more detail in the following.

3.1.1 Messenger

Owned by Facebook and created to work alongside a Facebook account. Among the 7 IMAs mentioned in this section, Messenger is one of the 2 which offers various means of logging into an account. User can log in either using their existing Facebook account or the email or phone number of the account created based on. Among all the 6 other IMAs, to use Messenger the user needs to have a Facebook account which can be created based on the methods mentioned above. Messenger is the only IMA where social media is required to access the application. But Facebook also gives users the option to deactivate their Facebook account while retaining the Messenger account created from the account, facebook refers to this function as DEMA which is Deactivated Except Messenger Account.

Installation: Messenger can be installed from the Microsoft Store or by downloading the executable file from the Messenger site, for this thesis the messenger was downloaded from the official site [21].

Feature and functionality: Though Messenger and Facebook are linked in a way that users are required to create a Facebook account to access Messenger, functionality such as story, and notifications are independent. Unlike some of the other IMAs mentioned here, Facebook doesn't allow users to add another account; in comparison, its mobile counterpart does.

3.1.2 Telegram

Telegram being one of the biggest competitors for other IMAs, has been steadily increasing in terms of users and downloads going from 35 million users in 2014 to 500 million users in the recent 2021 [22]. Telegram is one of the two IMAs that has a paid subscription version as "Telegram Premium" with additional functionality and features.

Installation: To get the application up and running, the user can either download it from the telegram site [23] or also from the Windows store or both which gives the user the ability to use the application installed to have different accounts. The process to log into Telegram is where the user can either use a QR code which appears on the application, or the user number, the only difference is that when using the user number option the code is sent from a Telegram chat rather than getting it via the SMS as default. Unfortunately, the requirement for the user to log into an application is either having access to the carrier number used in creating the account or the existing account logged in a separate device. Telegram can also be logged into its web version from any of the Windows browsers.

Feature and functionality: Most if not all the functionality and settings such as chat, privacy, and security for a Telegram account can be accessed and changed on the Windows application which will also be reflected in its mobile app. On top of that, the Telegram application also allows the user to have more than one account on a single application. It also provides features such as a chat folder where users can put private or group chat under a custom folder such as family, friend, etc. Telegram also has features such as bots that can interact with users in a group chat.

3.1.3 Discord

Discord gives the user the ability to log in either by email or phone number of their account, or the user can also utilize the QR code and log in from any mobile device where the desired account is logged in. This has been around since 2015 designed for gamers which was seen in their functionality such as providing channels and servers in addition to the private or group chat [24]. Discord was able to successfully capitalize on the gaming community and younger age group users [25].

Installation: Similar to messenger and telegram the discord application can be installed from the Windows store or executable file from the discord site. For this thesis discord was downloaded from its official site [26] and installed via executable file. The user can log into their Discord with either login credentials or a QR scan, discord also gives users the option to create a new account.

Feature and functionality: Among the 7 IMAs discord is the only one other than the Telegram application that provides its version of the Discord shop which can give the user the option to customize their account. Along with Telegram, Discord also has a paid 2-tier Nitro version where users can subscribe to get more benefits such as more upload speed, HD video streaming, etc. In terms of features and settings available between its mobile variant and PCs, all the accounts are accessible for users from both platforms with the addition of more settings that are specific to the Windows Platform such as open on start-up, minimize to the tray, etc,

3.1.4 Signal

The Signal application provides the strongest security features and encryptions when compared to other IMAs [27]. There are more settings and features for users to increase security which can be accessed from mobile devices. Signal also has only local backup in comparison to other IMAs.

Installation: For users to have a Signal application on the Windows system, the only source for them is to get from the developer's site [28], which they have access to the executable file and install those files onto the Windows system. There is only one way to use Signal,i.e., log in via QR scan. The user is required to log into their existing account from their mobile by scanning the QR code appearing on the screen. Signal requires the user to have an account created using a carrier number to log into the application. Unlike some of the other IMAs, Signal doesn't have an option for a web version and will require them to install the Signal application.

Feature and functionality: Signal doesn't have a feature for a user's last seen or online status but has an option for typing status which is off by default, these features may be considered basic by many since they are available in most of the IMAs (also present in all the 6 other IMAs mentioned here) but were purposely omitted to preserve the privacy of the user.

3.1.5 Skype

Skype is the oldest messaging application of the 7 IMAs here, and it does show in the interface which hasn't gone through any major changes in recent years. In terms of functionality and features, it is not too far behind the other considering the age of the application and getting only minor updates. The big update that Skype received was the access to copilot natively, which should be the impact of being under the umbrella of Microsoft. Due to Skype being available for a long time, it has been used for face-to-face virtual interviews minimizing the gap between geographical restrictions [29].

Installation: Skype is installed alongside Microsoft Office but can also be downloaded from the Microsoft Store [30] which was what followed here. Users can log in with an account created with either a mail ID or mobile number. For users who don't have an account, they can create one from the Skype application itself.

Feature and functionality: Skype has all the basic features, but in comparison to other IMAs does lack some new features such as automatic replies, broadcast, etc, The skype PC application provides the user with all the necessary options to change or reset their accounts, this cannot be said to some of the other IMAs mentioned here.

3.1.6 Teams

Similar to Skype, Teams was created with a focus on workplace use. From [31], the majority of the age group in which the user falls is 35 and above. This is understandable as this was made in focus with work and comparatively the exact opposite of Discord.

Installation: Teams can be installed alongside Microsoft Office or can be downloaded from Microsoft Store [32]. Like some of the other IMAs used, user can either log into their existing account or be given the option to create a new account.

Feature and functionality: Teams among the 7 IMAs, were from the ground up made for professional use, from [33] this can be seen from the spike in the usage of teams during the Covid and hasn't declined much after that. Unlike the other IMAs, teams provide many functions that can help in workflow space such as setting up meetings and tracking meetings from calendars, notes, access to user's Onedrive plugin applications, and more. In terms of options and settings available, it can be changed or accessed from all the platforms. The more I explore the more it's evident in teams being built with the primary use in the workplace.

3.1.7 WhatsApp

WhatsApp has been one of the most popular IMA in terms of the number of active users, and the number of messages that are being sent daily. Since 2013, it has steadily continued to increase the number of active and shows no signs of slowing down [34]. In 2018, WhatsApp had a new addition known application as the WhatsApp business app, this was streamlined with features that would attract a business owner to incorporate WhatsApp into their workflow by providing automatic greeting replies, away messages, connecting Instagram accounts, and more. For this research purpose in hopes of being consistent between all the IMAs, the consumer version of WhatsApp will be analyzed.

Installation: The application was downloaded directly via the built-in Microsoft store that is present in the latest Windows operating system [35]. To set up and start using WhatsApp on a PC users should have an existing account or have a new account created using their carrier number. Users can utilize two methods for login, One is where they can scan a QR code appearing on the PC's application from their mobile device in which the required existing account is logged in, and the second method is where users can enter their carrier number which was used in creating the WhatsApp account and request for a unique code to be generated by the WhatsApp application which can be entered from the mobile device either navigating through their "Linked Devices" in the application or from the notification of the code. On top of having an application installed, the user can also utilize the browser to access WhatsApp web which can have a different account than the application installed in Windows following the same instructions to log in as the app.

Feature and functionality: There are some major functionality differences between the Windows application and the mobile application. Privacy settings are non-existent from Windows and any changes on needs can be only done on the mobile device. In addition to that, not all the chats from users are available, some of the very old chats will require the users to refer to their mobile devices.

3.2 Types of Text Communications

In this thesis, I aim to analyze IMA traffic under asynchronous and synchronous communications for texting, specifically private, groups with 3 and 4 users. In this context, the synchronous and asynchronous concepts represent different ways in which real-world texting takes place. Here I will explore the similar and different factors for my research purposes.

Synchronous Communication: Synchronous texting happens when all the users who engage in a message are texting each other in real-time. The Working and characteristics of the synchronous communication are as follows:

1. Initiating texting: User 1 will start texting when all the other users are live in the communication.

2. Continuation: The other Users will continue the communication by replying to the message sent by User 1, and this exchange of messages will continue, imitating real-time communication.

3. Data collection Duration: The duration is set based on the number of flows extracted. The aim is to have at least 5000 flows from all the 7 IMAs from

private, groups with 3 and 4 users. The duration for the private communication is greater than or equal to the duration set for the group communications. This is because in private communication the traffic is captured from two Users in comparison to group communication where the capture is obtained from 3 and 4 Users for groups of 3 and 4 users, respectively.

Table 3.1: Data collection duration for synchronous

IMA	**Private (Hours)**	**Group of 3 (Hours)**	**Group of 4 (Hours)**
Skype	4	2	2
Discord	4	3	2
Messenger	2	2	1
Teams	3	2	1
WhatsApp	5	3	2
Teams	5	4	3
Signal	5	3	2

Table 3.5 can be referred to as the number of flows extracted under a synchronous text conversation from the data collected for the duration shown in Table 3.1. More about synchronous communication is to be explained in the next Section 3.3 with the help of Figure 3.2 along with delays used to means to mimic real-world conversation.

Asynchronous Communication: Asynchronous communication is where the users join the texting at a different time than the user who initiated the texting. This is done by using different types of delays such as '*reply delay*'. The Working and characteristics of the asynchronous communication are as follows:

1. Initiating texting: Similar to the synchronous, User 1 will initiate the conversation by sending the first text message while other users are not live in the communication.

2. Continuation: Unlike the synchronous setup, User 2 (in case of private) or User 3 or User 4 (in case of the group) will join the communication after a randomly specified delay referred to as 'reply delay'. The 'reply delay', as mentioned in section 3.2 is a predefined variable that was critical for emulating scenarios where users do not respond immediately. This delay helped in understanding

how the network traffic will behave and change its pattern due to the delayed messages.

3. Data collection Duration: Unlike synchronous texting, in asynchronous texting via IMAs, I have used a constant 6-hour period consisting of varying numbers of users. This is done because asynchronous texting uses a reply delay of an average of 20 minutes in addition to the read and type delays, to incorporate more communication in the data collection, the emulation is performed during 6 hours. Additionally, I didn't need to use varying durations because in asynchronous texting the number of flows generated was generally high. This might be because it needs to make the necessary connections. More on this along with the delays employed to mimic the time taken to reply are explained in section 3.3 along with Figure 3.1.

3.3 Generating Realistic Texting Behaviours

The SAMsum corpus [36] is a product of collaborative research aimed at developing a model capable of summarizing conversations. These conversations range from informal to formal, incorporating slang, emoticons, and typos to reflect realistic text interactions. The corpus includes dialogues and corresponding summaries generated by models for those dialogues. This data is particularly valuable for my research purposes, as it encompasses conversations with varying numbers of participants.

Isolating text conversations with 2, 3, and 4 users allows me to utilize these dialogues for different types of user communication, such as private texting with 2 users and group texting with 3 or 4 users. Emoticons are then removed from all communications for implementing delays based on the number of words present in the text. The delays introduced for this purpose are determined by the number of words in each text message sent and received between the users. To emulate the time taken by a user to type and read a text message, I have introduced delays such as "*type delay*" referring to the time taken by the user to type a message, and "*read delay*" referring to the time taken by a user to read the received message. The SAMsum corpus is then employed to create "*type*" and "*read*" delays for messages sent using synchronous and asynchronous texting behaviours between users.

For calculating these delays, I used Baker et al. [37] work which revolved around analyzing the computer typing style and speed. Where Baker et al. were able to determine the average typing speed on a computer was 50.4 WPM, this is used as a means to incorporate the typing duration users make when sending a text message. As for the read delay, work done by Mpofuet et al. [38] on determining reading speed on different platforms, where they concluded that the average reading speed using computers is 106 WPM. This is used to mimic the duration taken by the users to read an opened message.

Since this research focused on textual messages, both synchronous and asynchronous texting via IMAs are studied. In synchronous texting, all users of a given communication continuously send text messages for the desired duration. In asynchronous texting, one of the users, i.e. user 1, initiates the texting by sending the first text message, and the remaining users join after a randomly generated "*reply delay*". In addition to that, after a series of messages from all users, the application is closed and reopened, to emulate the asynchronous texting behaviours. The reply delay is based on a range where the average is 1200 seconds (20 minutes) and the standard deviation is 600 seconds (10 minutes) [39].

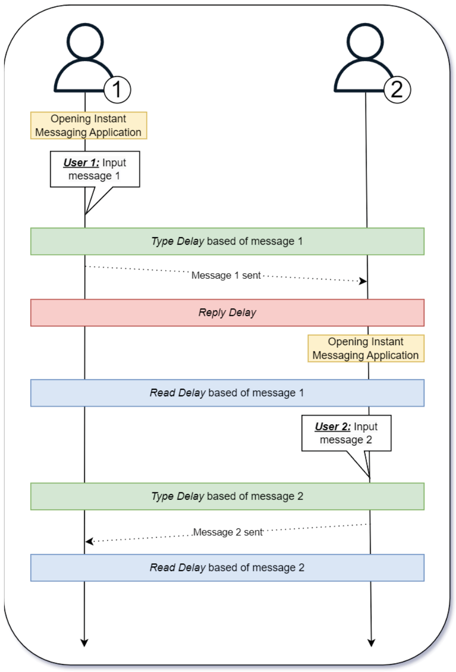
1
2
Opening Instant Messaging Application
User 1: Input message 1
Type Delay based of message 1
Message 1 sent
Reply Delay
Opening Instant Messaging Application
Read Delay based of message 1
User 2: Input message 2
Type Delay based of message 2
Message 2 sent
Read Delay based of message 2

Figure 3.1: Asynchronous texting emulation

Figure 3.1 illustrates the asynchronous texting between 2 users. In this case, User 1 will open an IMA type the desired text, and wait for the "*type delay*" duration to be completed. Only after that will the message be sent. Then, User 2 will wait for a set duration including first the "*Read delay*" and then the "*reply delay*". Once the reply message is typed by User 2, the next message is then sent after "*reply, read, and type delays*"

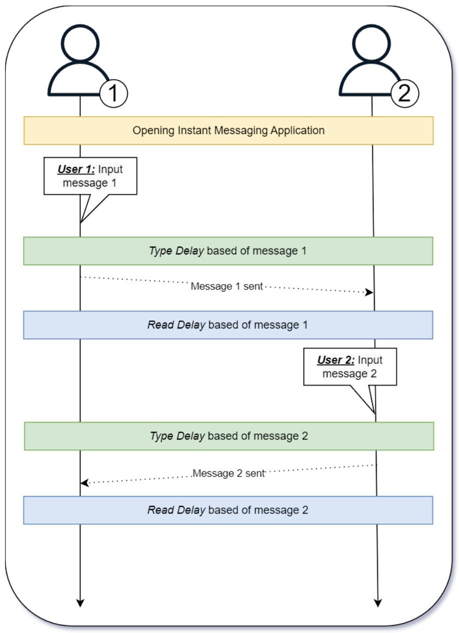
1
2
Opening Instant Messaging Application
User 1: Input message 1
Type Delay based of message 1
Message 1 sent
Read Delay based of message 1
User 2: Input message 2
Type Delay based of message 2
Message 2 sent
Read Delay based of message 2

Figure 3.2: Synchronous conversation emulation

Figure 3.2 illustrates the synchronous conversation. Unlike the asynchronous one, all the users participating in the synchronous conversation will join and end at the same time eliminating the need for '*reply delay*" and only using the "*type delay*" and "*read delay*".

3.4 Capturing the Traffic

The data collection for the IMAs was conducted using four Windows 11 PCs. These devices were designated as follows: two for private chat, three for group chats with three users, and all four for group chats with four users. The data collection encompassed both synchronous and asynchronous communication in IMAs.

”*Pktmon*” or Packet Monitor, available natively from Windows 10 (build 19041) [40], is used for packet capturing, event tracing, packet drop detection, packet filtering, and counting. One of the benefits of Pktmon is its integration within Windows, making it a built-in tool accessible through various commands, allowing users to utilize many of its functions. By default, the traffic capture is saved in ETL files and includes a command to convert ETL files into PCAP [41] (more precisely, *PCAPNG*) format. Table 3.2 and Table 3.3 show the number of packets captured under different texting conditions discussed in the previous Section 3.2 and Section 3.3.

Table 3.2: Packet Count for Asynchronous

IMA	**Private**	**Group of 3**	**Group of 4**
Skype	1028246	1542910	1900652
Discord	1002436	1253352	1721634
Messenger	1304538	1894746	2632752
Teams	1099418	1541740	3424762
WhatsApp	1866830	2591169	3410086
Telegram	2144322	3130956	3935270
Signal	2033120	2766604	3318780

Table 3.3: Packet Count for Synchronous

IMA	**Private**	**Group of 3**	**Group of 4**
Skype	803094	674352	835646
Discord	518546	625622	639950
Messenger	1370378	993868	586480
Teams	738508	740216	410484
WhatsApp	509262	615916	721540
Telegram	480832	2019776	1167214
Signal	653580	737162	1039966

The captured traffic was saved with a specific naming format to prevent overwriting, starting with ”PC” followed by the IMA, continuing with synchronous or asynchronous, and followed by private, group_3, or group_4 chat, representing private communication, the group with 3 users communication and group with 4 users communication respectively and then finally timestamp. Table 3.4 refers to the naming convention along with an example using Signal as an Instant Messaging Application with synchronous texting from private chat:

Table 3.4: Naming format for the traffic captured

Format	**PC_X_{ima}_{Texting Via IMAs}_{chat type}_{timestamp}.etl** X: 1,2,3,4 Texting Via IMAs: syn/asyn communication: private/group_3/group_4 Timestamp: year_month_date_hh_mm_ss
Example	PC_1_signal_syn_private_0000_00_00_00_00_00.etl

3.5 Framework for Texting Traffic Generation

As shown in Figure 3.3, four devices were used in the framework for data collection namely PC 1, PC 2, PC 3, and PC 4. The network connection was established for all four devices via a hotspot from the main device. The data collection is based on 2 users (one per device) for private communications, and 3 and 4 users for group communications (again one user per device).

All the devices mentioned in the framework for data collection were debloated i.e., software that is not relevant to our use case was removed. On top of that when data collection for a specific Instant messaging application is undergoing the rest of the IMAs are not installed in the PCs. This is to avoid any interference between IMAs during data collection. Data collected are then moved to a controlled environment where the flow extraction, filtering background, and classification are performed.

3.6 Emulation: Automation of User Behaviours

Python's library "pyautogui" was used to automate the process of generating texting of a user opening the application, selecting texting, typing a message, and sending it. The library provides functions such as moving pointers, sending left and right-click commands, input text, hotkeys, etc. A combination of all the functions mentioned

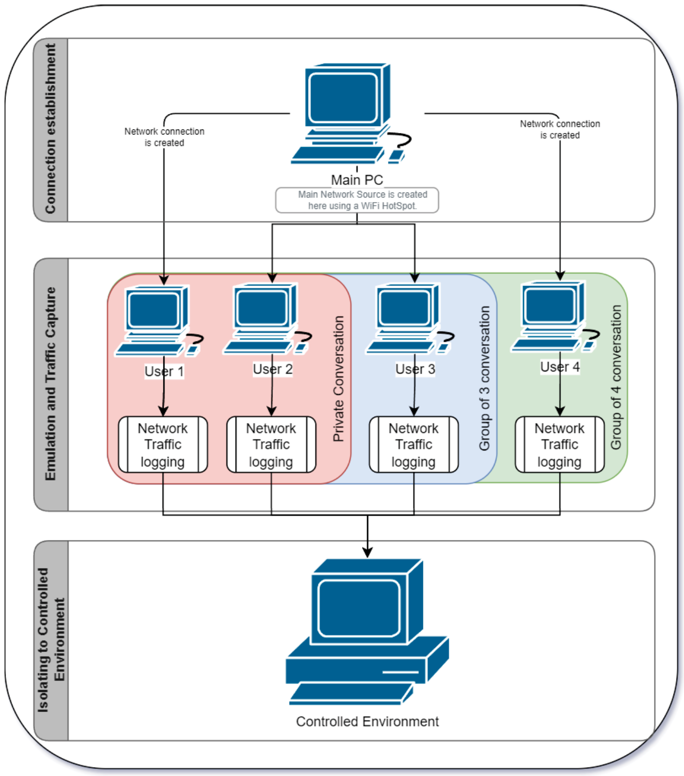

Figure 3.3: Overview of the Framework

above was used. The integration of ‘pyautogui‘ into the framework ensured that the actions performed were consistent and replicable across multiple test runs. The typical workflow involved:

1. Initializing PyAutoGUI: Setting up the library and calibrating initial positions and parameters. The automation process began with emulating the user opening the chat application. Using the ‘pyautogui‘ library, the script could locate the x and y coordinates of the screen and use it to interact with the application on the device or taskbar and initiate and open the application (Instant Messaging Application in this case).

2. Opening the Application: I have pinned the instant messaging application on the taskbar, with the help of hotkey events I can start the the instant messaging application. Closing the application is custom-made as some instant messaging applications in some cases could require to be closed through the taskbar in addition to closing the application’s window.

3. Selecting the Texing Window: Once the application was open, the script navigated to the texting window. This involved using a series of mouse movements and clicks to ensure that the correct chat interface was selected. Functions such as ‘pyautogui.moveTo(x, y, duration)‘ and ‘pyautogui.click()‘ were employed to perform these actions seamlessly. Navigate through the application’s interface to the correct texting window.

4. Typing and Sending Messages: Emulating the sending of messages with the use of ’pyautogui.typewrite(text)’ is the process of typing and sending messages with the type and read delays discussed in section 3.3, both in private and group texting.

3.7 Flow Extractor: Tranalyzer2

Flow extractors are used to extract statistical features over aggregated packets (based on 5-tuples, namely source/destination IP addresses, source/destination Port numbers, and the Protocol) using the captured network traffic (Pcapng). This approach eliminates the need to perform a deep packet inspection of the captured traffic. *Tranalyzer2* is a popular lightweight flow extractor (used by [5]) that can extract flows even from large sets of captured data. *Tranalyzer2* is an open-source system built upon libpcap library and implemented in C. These characteristics of *Tranalyzer2* enables it to handle large volumes of network traffic, therefore making it suitable for flow generation, network monitoring, and traffic analysis.

By default, the *Tranalyzer2* produces 107 features, this can be increased to generate more statistical features by utilizing additional plugins depending on the goals of the research conducted.

In my case, I have enabled the variance feature flag known as ’BS_VAR’ under the plugin ’*basicStats*’ [42] which gives additional features such as ’varPktSize’ and ’varIAT’ [43] in addition to the base 107 features using the Algorithm 1. The final flow extracted will have 109 features which now includes two new additional features. Tables 3.5 and 3.6 show the number of flows extracted from the *PCAPNG* traffic capture files of the respective IMAs for synchronous and asynchronous texting. As

Algorithm 1 Setup and Configuration

```
1: procedure Setup and Configuration
2:     python3 -m pip install multipledispatch
3:     sudo apt-get install python3-pip
4:     python3 -m pip install pandas pdoc3
5:     t2py
6:     T2Utils.list_config('basicStats')
7:     T2Utils.get_config('basicStats', 'BS_AGGR_CNT')
8:     T2Utils.set_config('basicStats', 'BS_VAR', 1)
9:     T2Utils.build('basicStats')
10: end procedure
```

discussed in section 3.2, the aim is to have at least 5000 flows from each IMA (following the previous work [5]) for the machine learning model to have enough data for training and testing purposes without overfitting.

Table 3.5: Number of Flows - Synchronous communications

IMA	Private	Group of 3	Group of 4
Skype	5633	5446	7005
Discord	5672	7037	6407
Messenger	5004	8089	5565
Teams	5695	7256	5034
WhatsApp	5318	6876	7168
Telegram	5267	7624	7371
Signal	6121	6139	5460

Table 3.6: Number of Flows - Asynchronous communications

IMA	Private	Group of 3	Group of 4
Skype	12418	18815	17779
Discord	11004	16187	18893
Messenger	14950	22561	27521
Teams	11330	16433	45955
WhatsApp	36125	46652	60821
Telegram	36784	55246	56985
Signal	37396	39435	44204

3.8 Filtering the background traffic

To the best of my knowledge, there is not one way to capture the network traffic of an instant messaging application in Windows. In some instances, applications such as Teams and Messenger have more than 32 ports (the maximum number of filters that can be added) active during their usage. Thus, in this research, I captured the entire network traffic first. And then, I filtered out any traffic other than the Instant Messaging Application that is being studied.

Algorithm 2 Filtering Non-IMA Traffic

```
1: procedure Filtering Non-IMA Traffic(pcapng)
2:     for each packet in pcapng do
3:         Check for the $ACK\_packets$ and store its port Y
4:         Check for the $SYN\_packets$ with same port Y
5:         if packet contains 'ACK' and doesn't have 'SYN' for the same port then
6:             Mark the IP X as Non-IMA Traffic
7:         end if
8:     end for
9:     for flow in $flows$ do
10:        if destination IP belongs to series Non-IMA Traffic IP X then
11:            Drop the flow entry
12:        end if
13:    end for
14: end procedure
```

To this end, to determine the non-IMA traffic, a Python script is developed, Algorithm 2, to identify the 'ACK' packet present in the *PCAPNG* captured (traffic). The script searches for a 'SYN' packet for the ports of IMAs. This is used to determine whether a connection was initialized before the data collection was started. If so, then it is categorized as non-IMA traffic. The IP address of all non-IMA traffic is then converted into the network traffic class and all flows from the same class are then dropped.

The flows extracted from an IMA's traffic are then filtered, Tables 3.7 and 3.8 show the number of flows available for the synchronous and asynchronous texting

Table 3.7: Number of Flows after filtering - Synchronous Communications

IMA	**Private**	**Group of 3**	**Group of 4**
Skype	4896	4667	6020
Discord	5329	6664	5879
Messenger	4860	7829	5413
Teams	5159	6557	4535
WhatsApp	4907	6421	6579
Telegram	4934	7077	6807
Signal	5680	5676	4976

Table 3.8: Number of Flows after filtering - Asynchronous Communications

IMA	**Private**	**Group of 3**	**Group of 4**
Skype	11230	16951	15555
Discord	10597	15500	18122
Messenger	14375	21795	26516
Teams	10051	14645	43807
WhatsApp	35629	45820	59746
Telegram	36234	54405	55860
Signal	36826	38529	43145

traffic, respectively (after filtering out the flows which started before the data capturing process was started).

3.9 Machine Learning Models Employed

This section presents the machine learning models used for classification. These include Random Forest, Extreme Gradient Boosting, Multi-Layer Perceptron, and Naive Bayes classifiers along with their advantages and disadvantages.

3.9.1 Naive Bayes

The Naive Bayes algorithm is based on Bayes's Theorem and works on the principle that features independence hence the name ”Naive” [44]. This model works by assuming that every feature of a dataset is conditionally independent of each other, this classification algorithm helps in making a simple and effective model.

For approximating the target function X $\rightarrow$ Y or P(X—Y), T is a Boolean values random variable and X is a vector containing n Boolean attributes. For a given input

Naive Bayes works as follows:

1. Calculate the prior probabilities for each class based on the training data.
2. Computing likelihood for each feature for a given class by estimating the probability distribution for each feature per class.
3. Calculate the posterior probability for each class of a given input using Bayes theorem.
4. Assign the class with the highest posterior probability to the input example.

This is a probabilistic classifier meaning the model prediction is made based on the probability calculated for an instance of the data for a given feature value, assuming that the feature contributing to predictions has no relation with any other features.

3.9.2 Multi-Layer Perceptron

A Multi-Layer Perceptron is an artificial neural network [44]. It consists of an input and an output layer with one or more hidden layers between them. The activation function acts as a weighted sum of inputs which introduces nonlinearity to the network to learn complex patterns in the data. When data is passed through the input layer during the feedforward, each neuron calculates its weighted sum of the input based on the activation function employed and moves it to the next layer. The weights are adjusted by comparing the network's output layer value and actual value. During the backpropagation, weights are adjusted based on the gradient loss function used [44]. For real input values, Multi-Layer Perceptron works as follows:

1. The preprocessed data is first fed to the input layer.
2. The hidden layer calculates the output from each neuron via weights and activation functions which are then fed into the next layer.
3. The error is calculated by the difference between the actual values and predicted output.

4. Apply error propagation and gradient descent for optimization weights to minimize the error.

5. The steps are repeated for a given epoch until error reduction.

3.9.3 Extreme Gradient Boosting

Extreme Gradient Boosting which is also known as XGBoost is an ensemble boosting technique that combines multiple decision trees to create a robust final model [44]. It builds a series of decision trees, where every new tree will try to correct the errors made by the previous tree. For the final model to make an accurate prediction on the training dataset, a gradient descent algorithm is used to determine an optimal weight in each tree. XGBoost works as follows:

1. As an initial approximation with the target values an initial prediction is made.

2. Error from the initial prediction is then calculated by the difference between the actual and predicted values.

3. A new tree which is a weak learner is trained to reduce the error made by the initial model.

4. Update the model's predictions by adding the output from the newly constructed tree to the existing predictions.

5. The prediction of all the trees combined to make a prediction.

3.9.4 Random Forest

The Random Forest algorithm is also an ensemble learning technique [44]. The bagging method is improved upon by the Random Forest strategy, which combines both bagging and feature randomness. Random Forest uses many Decision trees as basic classifiers which are completely independent of each other and take into account all potential feature splits, providing an essential comparison [44]. The likelihood of overfitting can be reduced, and produce more precise predictions by taking into account all possible sources of variation in the data. Feature bagging adds a degree of randomness to the dataset while decreasing the correlation between decision trees. Random Forest works as follows:

1. Input the sample preprocessed data.
2. For the given data sample, several random subsets of features is chosen as training.
3. The error is calculated by the difference between the actual values and predicted output.
4. Testing is done using the trained model containing a collection of decision trees.
5. Final prediction is made by a majority vote.

3.10 Summary

In this Chapter, I presented the different IMAs used along with all of their application-oriented features. Also further described the framework and how the IMA was set for working with different texting via IMAs such as asynchronous and synchronous from the perspective of user communication from different numbers of users such as 2 users for private, a group of 3 users, and a group of 4 users. I also discussed the various delays incorporated which were created and used as a way to generate more realistic user data based on the time taken by users to read, write, and reply to a message sent/received. Detailed explanation was given on the two different texting behaviours, namely asynchronous and synchronous, along with what makes them different. I also went over the naming convention used for the traffic capturing method and described the automation of realistic text behaviours for traffic generation and capturing of *PCAPNG* files. Additionally, I explained how I created the Python scripts, and how they were utilized with the help of available libraries. These were then used to emulate the text behaviours for opening, closing, and sending messages from IMAs used.

I also explained the *Tranalyzer2* flow extractor utilized in converting the packet captures, *PCAPNG* files, to flows for extracting statistical features to represent the IMA traffic. This section also included how to enable all the necessary plugins which result in obtaining additional features. Following, I presented the algorithm used in the process of filtering out all the traffic established before the data collection was started as a means to focus on the Instant Messaging Applications. Traffic before and after filtering the background traffic was also discussed in the same section. Moreover, all the Machine Learning Models used were discussed in this Chapter.

Chapter 4

Evaluations and Results

In this chapter, I present the training and testing results obtained based on the captured traffic as described in Chapter 3. The following presents the evaluation metrics used, the features employed, and the results of the classification process. In addition to the results, the feature importance is analyzed for the best-performing machine learning model, namely the Random Forest classifier, to gain insight and uncover any patterns created by the IMAs.

4.1 Evaluation Metrics

As introduced in the previous chapter, various machine learning models are employed, and utilized under several different scenarios or texting via IMAs including (i) Private Synchronous Texting, (ii) Private Asynchronous Texting, (iii) Group (3 or 4 users) Synchronous Texting, and (iv) Group (3 or 4 users) Asynchronous Texting. Since there may be overlays between models in terms of which scenario they would perform better, I evaluated those machine learning models using the following performance metrics. For all performance metrics used in this research, Higher Values Indicate Better Performance, namely Precision, Recall, and F1-Score. Furthermore, TP, TN, FP, and FN refer to True Positive, True Negative, False Positive, and False Negative ratios.

Precision: Precision also refered to as positive predictive value is the fraction of relevant instances (TP) among all retrieved instances [45].

$$\frac{TP}{TP + FP} \tag{4.1}$$

Recall: Recall (sometimes called sensitivity) is the fraction of relevant instances that were retrieved [45].

$$\frac{TP}{TP + FN} \tag{4.2}$$

F1-score: F1-score is the harmonic mean of the precision and recall [45].

$$\frac{2 * (Precision * Recall)}{Precision + Recall} \tag{4.3}$$

Accuracy: It is the ratio of the correctly predicted values by a total number of predictions. It can be calculated by [45]:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{4.4}$$

4.2 Classification Process

4.2.1 Classification Dataset

Table 4.1: Sample for filtered Skype Private Asynchronous vs other

IMA	**Private**		**Group_3**		**Group_4**	
	asyn	**syn**	**asyn**	**syn**	**asyn**	**syn**
Skype	11230	4893	16951	4667	15555	6020
	Class 1	Class 0 (Random sample 11230)				

Table 4.1, shows a sample of how a dataset is built from the collected flows. Binary classification is performed in this case between the Skype private asynchronous class 1 (in-class) which is the class that is needed to classify and the rest is regarded as class 0 (out-class). In the above example, the goal is to classify private asynchronous from the data point of asynchronous and synchronous groups with 3 and 4 users along with private synchronous. Class 0 is always of the same size as class 1, this is made to prevent an imbalanced training dataset and have structure between the other combinations of in and out classes of Skype such as *Skype Private Synchronous vs other*, *Skype Group of 3 users Synchronous vs other*, *Skype Group of 3 users Asynchronous vs other*, *Skype Group of 4 users Synchronous vs other*, and *Skype Group of 4 users Asynchronous vs other*. This is repeated for all the 7 IMAs used in the data collection.

4.2.2 Feature Set

The 109 default features output by *Tranalyzer2* discussed in section 3.7 include a combination of time and size-related packet features, statistical features, and also what I call the 'biased' features. Biased features include IP addresses, port numbers as well as country codes, etc. These were all excluded to minimize the bias created by these in terms of their correlation to the ground truth (label) of the data. I considered these as biased features because I only have four devices on my framework. This then creates a high correlation between different texting behaviours and devices used in the framework.

With the means to create a Machine learning model that could generalize well for the given dataset the destination and source of IP and port numbers are removed from the dataset. '*timefirst*' and '*timelast*' are also considered biased for the nature of the classification used here. This could easily influence the classification since the flow extracted will be from the connection created by the message sent. Therefore, the '*timefirst*' and '*timelast*' are also regarded as biased features and are removed.

4.3 Results and Discussions

Here the aim is to determine a reliable and consistent model explained in previous sections that can distinguish the different texting such as asynchronous and synchronous from different numbers of users for the 7 IMAs used. All the results for the group with 3 users were from four machine learning models mentioned in Section 3.9 with the same training and testing datasets.

4.3.1 Group of 3

Out of the four models' results, Naive Bayes is among the poorer-performing models here. From Table 4.2, we can observe that even the best-performing IMA for the asynchronous vs. other is Skype when compared with the other models here it is easily outshined in most cases. Hence for asynchronous vs other groups with 3 users, Naive Bayes isn't a viable route. Shifting the focus to the Multi-Layer Perceptron from Table 4.3, had a noticeable increment in performance in IMAs Discord, Teams, Telegram, and Messenger when compared with the Naive Bayes. However, the results

Table 4.2: Naive Bayes *Test Results*: **Group of 3**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.577	0.494	0.481	0.524	0.531	0.505	0.500
F1	0.565	0.349	0.335	0.436	0.436	0.353	0.350
Recall	0.575	0.501	0.505	0.523	0.529	0.505	0.500
Precision	0.583	0.514	0.675	0.560	0.587	0.588	0.498
Synchronous vs other							
Accuracy	0.555	0.503	0.487	0.589	0.529	0.469	0.494
F1	0.555	0.363	0.332	0.534	0.396	0.320	0.334
Recall	0.556	0.508	0.501	0.578	0.513	0.498	0.500
Precision	0.556	0.581	0.551	0.637	0.572	0.335	0.514

Table 4.3: Multi-Layer Perceptron *Test Results*: **Group of 3**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.554	0.495	0.625	0.514	0.465	0.531	0.516
F1	0.515	0.463	0.592	0.378	0.401	0.508	0.471
Recall	0.551	0.491	0.640	0.512	0.466	0.531	0.516
Precision	0.574	0.489	0.725	0.592	0.440	0.539	0.524
Synchronous vs other							
Accuracy	0.580	0.629	0.604	0.597	0.730	0.793	0.712
F1	0.580	0.605	0.597	0.528	0.724	0.791	0.712
Recall	0.581	0.632	0.601	0.586	0.726	0.800	0.713
Precision	0.582	0.678	0.606	0.685	0.742	0.812	0.713

are not close to the other two models.

Moving on to the next model Extreme Gradient Boosting, the results from Table 4.4 show us that there is a significant increase in performance throughout the IMAs for asynchronous vs other when compared to the other two models explained prior. WhatsApp and Telegram are among the poorest results from the Extreme Gradient Boosting Model despite still performing about 50% better than the Naive Bayes and Multi-Layer Perceptron. Finally, the Random Forest results from Table 4.5 bring the best performance among the four different models used here for asynchronous vs other. It is either on pair or performance better than the extreme Gradient Boosting models even with the WhatsApp and Telegram IMAs.

Table 4.4: Extreme Gradient Boosting *Test Results*: **Group of 3**

IMAs / Metric	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.758	0.676	0.760	0.617	0.615	0.610	0.670
F1	0.757	0.670	0.758	0.599	0.589	0.604	0.665
Recall	0.758	0.678	0.766	0.616	0.614	0.610	0.670
Precision	0.760	0.694	0.779	0.641	0.651	0.618	0.679
Synchronous vs other							
Accuracy	0.825	0.824	0.870	0.910	0.913	0.878	0.877
F1	0.825	0.823	0.870	0.910	0.913	0.878	0.877
Recall	0.826	0.824	0.871	0.911	0.913	0.881	0.876
Precision	0.826	0.828	0.871	0.910	0.912	0.881	0.880

Table 4.5: Random Forest *Test Results*: **Group of 3**

IMAs / Metric	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.764	0.690	0.765	0.618	0.620	0.626	0.687
F1	0.764	0.688	0.765	0.607	0.604	0.619	0.686
Recall	0.764	0.691	0.768	0.618	0.619	0.626	0.687
Precision	0.766	0.695	0.770	0.632	0.642	0.635	0.689
Synchronous vs other							
Accuracy	0.819	0.813	0.869	0.908	0.901	0.874	0.876
F1	0.819	0.813	0.869	0.908	0.901	0.874	0.876
Recall	0.820	0.814	0.869	0.909	0.901	0.876	0.876
Precision	0.820	0.815	0.869	0.909	0.901	0.875	0.879

For the synchronous vs. other, as observed from Table 4.2 and 4.3 representing the results from the Naive Bayes and Multi-Layer Perceptron respectively, except WhatsApp IMA which performed similarly in both the Multi-Layer Perceptron, was consistently better than the Naive Bayes by around 50% in some cases. Moreover, the performance from Telegram, Skype, and Messenger IMAs is significantly higher than its asynchronous vs other results from the same IMAs.

Extreme Gradient Boosting and Random Forest results from Table 4.4 and 4.5 respectively have performed better than Naive Bayes and Multi-Layer Perceptron. Results from Extreme Gradient Boosting were increased for WhatsApp and Telegram IMAs which produced the least performance in asynchronous vs other. Overall in comparison with their own asynchronous vs other results, both models here produced better results of more than 80% throughout the 7 IMAs making them pair with each other.

In summary of the results discussed, from the 4 models used Random forest and extreme gradient boosting had performed better. From the asynchronous vs other and synchronous vs other, Random forest performed better in some IMAs whereas Extreme gradient boosting performed better in other IMAs. However extreme gradient boosting performed below 60% in Telegram and Signal IMAs of asynchronous vs other, therefore making Random Forest the more reliable and consistent of the two here.

4.3.2 Group of 4

Table 4.6: Naive Bayes *Test Results*: **Group of 4**

IMAs / Metric	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.484	0.504	0.541	0.507	0.506	0.503	0.494
F1	0.333	0.360	0.432	0.363	0.368	0.362	0.368
Recall	0.500	0.504	0.531	0.507	0.505	0.505	0.502
Precision	0.504	0.538	0.611	0.570	0.535	0.552	0.514
Synchronous vs other							
Accuracy	0.521	0.535	0.485	0.631	0.649	0.555	0.540
F1	0.389	0.393	0.332	0.589	0.614	0.407	0.433
Recall	0.503	0.508	0.500	0.621	0.642	0.525	0.535
Precision	0.515	0.544	0.493	0.693	0.712	0.695	0.634

For the group of 4 users results of the Naive Bayes can be referred from Table 4.6, which shows that Naive Bayes is among the poorer side performance for classifying the asynchronous vs other for the IMAs used. Although the feature-independent nature of the model uses an efficient algorithm, it provides an easier understanding and is inexpensive and easy to implement when compared to black-box models such as the multi-layer perceptron used here. But this doesn't help much in favor of this model as expecting to have an independent feature has caused it to make a model with poor generalization being easily influenced by irrelevant features. Shifting to the multi-layer perceptron model performance as shown in Table 4.7, although this

Table 4.7: Multi-Layer Perceptron *Test Results*: **Group of 4**

IMAs / Metric	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.581	0.497	0.561	0.506	0.538	0.502	0.594
F1	0.557	0.488	0.477	0.412	0.512	0.464	0.594
Recall	0.589	0.496	0.571	0.506	0.538	0.503	0.595
Precision	0.618	0.496	0.718	0.516	0.548	0.504	0.595
Synchronous vs other							
Accuracy	0.538	0.628	0.546	0.794	0.645	0.719	0.625
F1	0.532	0.600	0.534	0.793	0.606	0.718	0.621
Recall	0.543	0.646	0.542	0.799	0.637	0.718	0.627
Precision	0.547	0.723	0.546	0.814	0.714	0.718	0.634

Table 4.8: Extreme Gradient Boosting *Test Results*: **Group of 4**

IMAs / Metric	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.854	0.681	0.811	0.637	0.624	0.637	0.715
F1	0.854	0.678	0.810	0.634	0.619	0.633	0.713
Recall	0.856	0.681	0.813	0.637	0.625	0.636	0.716
Precision	0.860	0.688	0.821	0.642	0.633	0.642	0.722
Synchronous vs other							
Accuracy	0.804	0.823	0.901	0.917	0.925	0.875	0.899
F1	0.804	0.823	0.901	0.917	0.925	0.875	0.899
Recall	0.806	0.826	0.901	0.918	0.925	0.877	0.899
Precision	0.806	0.825	0.901	0.917	0.925	0.875	0.901

is not producing results as poor as the Naive Bayes it's still not close to the other two models.

In comparison, Random Forest and Extreme Gradient Boosting produce better results as seen in Table 4.9 and Table 4.8 respectively as they are much more noticeable for the IMA Messenger, Skype and Teams of asynchronous vs other results. The classification of the group of 4 users is much better than the group of 3 users from the same Models results. And, the performance improvement from Extreme Gradient Boosting is now more significant than between the same models from a group of 3.

Continuing with the results, it is observed that in synchronous vs other for Naive

Table 4.9: Random Forest *Test Results*: **Group of 4**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.865	0.691	0.819	0.642	0.629	0.643	0.730
F1	0.865	0.691	0.819	0.640	0.625	0.641	0.730
Recall	0.866	0.691	0.819	0.642	0.629	0.643	0.731
Precision	0.867	0.693	0.819	0.646	0.635	0.647	0.733
Synchronous vs other							
Accuracy	0.814	0.828	0.908	0.917	0.916	0.869	0.897
F1	0.814	0.828	0.908	0.917	0.916	0.869	0.897
Recall	0.815	0.830	0.908	0.918	0.916	0.870	0.896
Precision	0.814	0.829	0.908	0.917	0.916	0.869	0.899

Bayes from Table 4.6 there is a noticeable increase from its asynchronous vs other results in the case of IMAs such as WhatsApp, Telegram, Signal, and Messenger. Shifting to Multi-Layer Perceptron, which improves from the results of the Naive Bayes across the board for all the IMAs, where even in the best case is more than 50% increment for IMA such as Signal.

Despite the improvement Multi-Layer Perceptron from the Naive Bayes, it still isn't within the range of the best-performing models here such as Extreme Gradient Boosting and Random Forest. The results for Extreme Gradient Boosting and Random Forest from Table 4.8 and 4.9 respectively show that both the models were able to perform better for the classification of the Synchronous vs other except Skype when compared with the asynchronous vs other. Similar to the group of 3, Random forest was able to slightly outperform the extreme gradient boosting in most of the IMAs results.

In summary, random forest and extreme gradient boosting provide comparable results of the 4 Machine Learning models used in this section. Between the two models, apart from the results of IMAs Telegram, Signal, and Messenger from synchronous vs other, Random Forests was able to perform slightly better making it a more consistent of the two models. From the Naive Bayes poor performance, we can conclude that the dataset isn't easily linearly separable. One of the main reasons for Random Forest's better performance is due to its ability to handle non-linear relationships of the dataset well. In addition to that random forest being an ensemble of decision trees is less prone to overfitting as the sizable trees used are the average of the uncorrelated trees which in turn helps in reducing the overall prediction error and variance.

4.3.3 Private

Table 4.10: Naive Bayes *Test Results*: **Private**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.512	0.495	0.484	0.530	0.521	0.527	0.512
F1	0.445	0.347	0.331	0.435	0.420	0.439	0.383
Recall	0.511	0.499	0.502	0.531	0.523	0.527	0.504
Precision	0.521	0.490	0.682	0.594	0.580	0.573	0.521
Synchronous vs other							
Accuracy	0.504	0.494	0.514	0.535	0.590	0.471	0.498
F1	0.339	0.365	0.344	0.402	0.536	0.342	0.337
Recall	0.502	0.502	0.500	0.521	0.579	0.495	0.499
Precision	0.668	0.512	0.529	0.627	0.636	0.461	0.449

Table 4.11: Multi-Layer Perceptron *Test Results*: **Private**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.541	0.587	0.553	0.510	0.545	0.572	0.523
F1	0.507	0.584	0.553	0.363	0.507	0.567	0.507
Recall	0.540	0.586	0.553	0.510	0.544	0.572	0.520
Precision	0.555	0.589	0.553	0.628	0.563	0.576	0.522
Synchronous vs other							
Accuracy	0.570	0.562	0.458	0.793	0.613	0.726	0.555
F1	0.570	0.561	0.453	0.792	0.564	0.725	0.510
Recall	0.570	0.562	0.461	0.792	0.625	0.724	0.555
Precision	0.570	0.562	0.459	0.794	0.749	0.726	0.587

The pattern for Naive Bayes seen from the results for 4.3.1 and 4.3.2 continues to be present in Private also. The results are still worse among the four models used here which can be observed from Table 4.10 for asynchronous vs other. Continuing, the Multi-Layer Perceptron does show an increment in the results from Naive Bayes in all the IMAs apart from WhatsApp asynchronous vs other as seen in Table 4.11.

Table 4.12: Extreme Gradient Boosting *Test Results*: **Private**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.745	0.696	0.757	0.697	0.682	0.709	0.697
F1	0.744	0.696	0.756	0.689	0.672	0.703	0.695
Recall	0.745	0.697	0.760	0.697	0.683	0.709	0.698
Precision	0.746	0.698	0.767	0.720	0.711	0.727	0.705
Synchronous vs other							
Accuracy	0.819	0.810	0.865	0.929	0.918	0.881	0.852
F1	0.818	0.810	0.865	0.929	0.918	0.881	0.852
Recall	0.819	0.811	0.866	0.929	0.918	0.883	0.852
Precision	0.822	0.812	0.867	0.929	0.917	0.883	0.855

Table 4.13: Random Forest *Test Results*: **private**

Metric \ IMAs	Skype	Discord	Teams	WhatsApp	Telegram	Signal	Messenger
Asynchronous vs other							
Accuracy	0.746	0.699	0.752	0.689	0.676	0.705	0.702
F1	0.746	0.699	0.752	0.682	0.666	0.699	0.701
Recall	0.746	0.700	0.754	0.689	0.677	0.705	0.703
Precision	0.747	0.701	0.755	0.710	0.702	0.722	0.705
Synchronous vs other							
Accuracy	0.811	0.797	0.862	0.929	0.896	0.868	0.853
F1	0.811	0.797	0.862	0.929	0.896	0.868	0.853
Recall	0.811	0.797	0.863	0.930	0.896	0.869	0.853
Precision	0.812	0.798	0.863	0.929	0.896	0.868	0.855

Moving over to Extreme Gradient Boosting Model results as shown in Table 4.12 it is observed that the extreme gradient boosting performs significantly better than the previously mentioned models. When compared with the Random Forest results from Table 4.13, extreme gradient boosting still maintains performance on par with random forest for the asynchronous vs other results.

Jumping to the synchronous results starting with Naive Bayes from Table 4.10, the models were able to perform better in the Telegram when compared to the results from asynchronous vs other for the same IMA. The rest of the results for the synchronous vs other are either similar to that of the results from asynchronous vs other or slightly worse. It is observed there to be an increase in performance in the model Multi-Layer Perceptron synchronous vs other results from Table 4.11 where results from WhatsApp and Telegram had significant improvement in comparison to its asynchronous vs other results.

Similar to the previous sections 4.3.1 and 4.3.2, Extreme Gradient Boosting and Random Forest outperformed the other two models used in the synchronous vs other and this is evident from their results observed in Table 4.12 and 4.13. Surprisingly, the extreme gradient boosting performed noticeably better for Telegram and Singal IMAs in comparison to the results from Random Forest of the same IMAs.

Out of the 4 models discussed here, Random Forest and Extreme Gradient boosting are the best-performing models by a huge margin. Among the two models, either they are within the marginal difference in performance or extreme gradient boosting performed better. Therefore among the two models extreme gradient boosting performed better for private chat in both asynchronous and synchronous.

4.4 Identification of Network Siganures

My results show that the Random Forest model consistently performs better in the group of 3 users and group of 4 users among the 4 machine learning models used in this thesis. The best and worst results are 92.9% and 60.7% from Private synchronous vs other, and Group of 3 users asynchronous vs other in WhatsApp, respectively. I have used the inbuilt 'features importance' of the Random Forest classifier to compare the asynchronous vs other, and synchronous vs other models. The feature importance from Random Forest is determined based on the Gini importance and Mean Decrease in Impurity (MDI) by measuring the model's performance when certain variables are excluded. To this end, the top 10 common and uncommon features are extracted from the asynchronous vs other models, and synchronous vs other models for each IMA, and compared to identify the network signatures.

Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 show the top 10 features and their importance given the Random Forest model used for the evaluations conducted. Tables 4.13, 4.5 and 4.9 show the performances of the models. Among the top 10 important features the common ones between asynchronous vs other, and synchronous vs other communications are shown in the bar chart, and the uncommon features (if any), are shown in the Venn diagram.

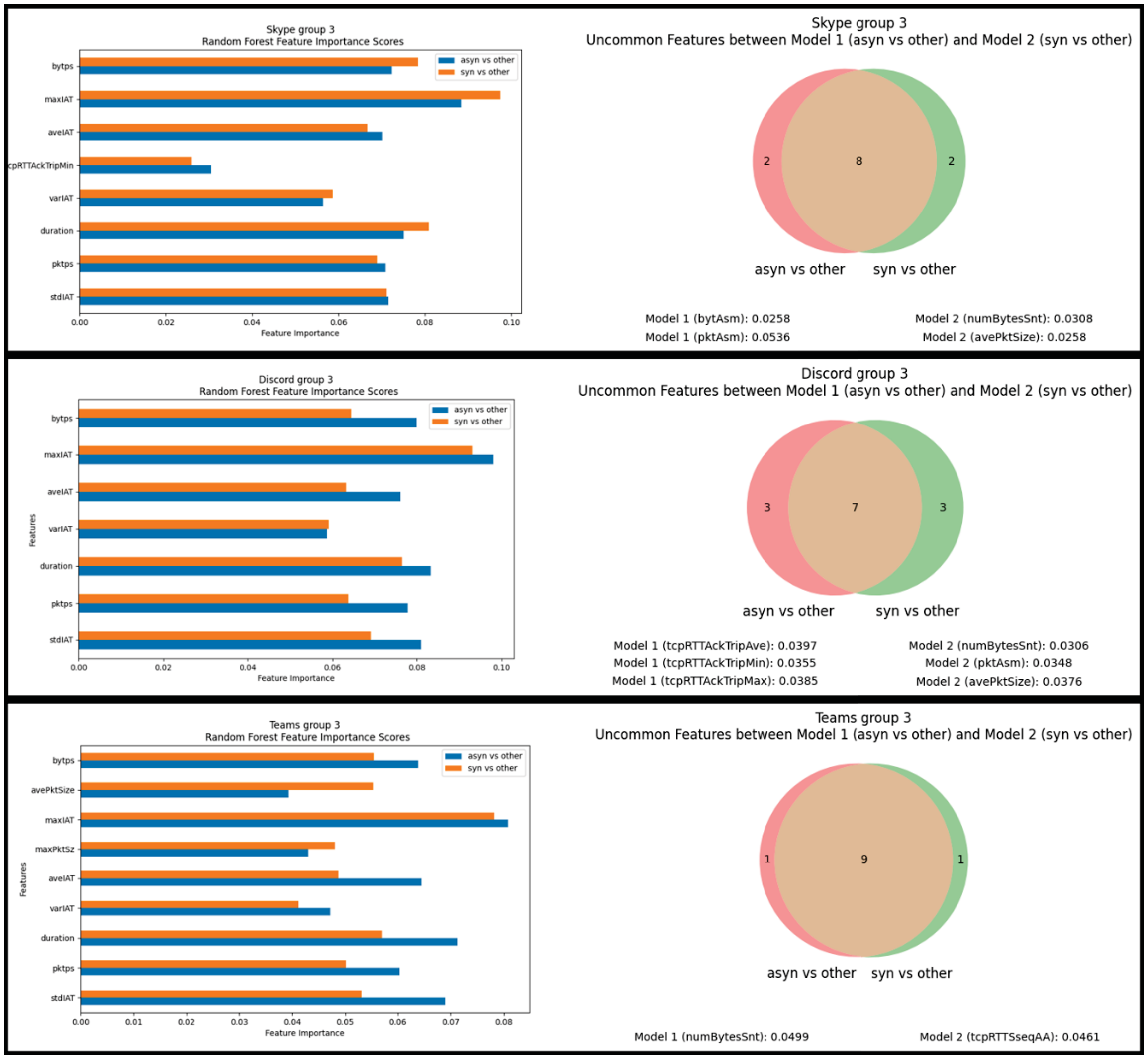

Figure 4.1: Comparing Random forest group of 3 user feature importance for Skype, discord, and Teams

Looking into the Group of 3 results between the asynchronous vs other, and synchronous vs other in Figure 4.1 and Figure 4.2, the very first observation is that among the 7 IMAs, there are unique sets of top 10 important features. What is noticed is that among the uncommon features, the synchronous vs other models consistently utilize more Packet size-related features than the asynchronous vs other models. This can be seen in the Venn diagrams for the IMAs. Additionally, what was observed from the common features is that asynchronous vs other models utilizes and gives more preference to the Time-based features such as Inter-Arrival Time (IAT). This is evident from the feature importance figures shown. Moreover, asynchronous vs other models also utilize features that are responsible for establishing a connection such as Round Trip Time (RTT).

Moving on to the features important for a Group of 4 Users in Figure 4.3 and Figure 4.4, it is observed that the feature importance of the model for the group of 4 users also shows similar patterns found in the group of 3 feature importance. Again, the synchronous vs other model utilizes packet size-related features, and the asynchronous vs other model utilizes the connection establishment features such as RTT as seen in the Venn diagram. The bar chart using the common features shows that IAT is again utilized by the asynchronous vs other models at a higher importance than the synchronous vs other models.

Similar to the network signatures identified for the group of 3 and group 4 users results above, the private asynchronous vs other model and synchronous vs other model's features importances are shown in Figure 4.5 and Figure 4.6. These figures demonstrate that the signatures identified are consistent across the board. In all the cases, synchronous vs other texting behaviours can be identified using the packet size-related features, and asynchronous vs other texting behaviours can be identified using time-based features such as IAT and RTT.

My results are also supported by the work of Bahramali et al., [46]. They also worked on IMAs and utilized delays (similar to what I use) which they termed as '*Inter-Message Delay*' for their experiments. They stated that '*extremely close messages create a combined traffic burst in the encrypted IM traffic*'. In my evaluations, I recognize this as well. While the asynchronous vs other model utilizes time-related features, the synchronous vs other model utilizes packet size-related features.

Figure 4.7 presents the total number of ports opened in the captured encrypted traffic concerning different IMAs. It is observed that each IMA has a different number of connections established. In both synchronous and asynchronous texting via IMAs, the number of ports opened are as follows (from the highest to the lowest): Messenger, Signal, Teams, Discord, Skype, WhatsApp, and Telegram. Moreover, though the number of ports opened is different between asynchronous and synchronous, the hierarchy remains the same. This indicates that in asynchronous texting, the ports opened are always higher than the number of ports opened in synchronous texting. This could contribute to the signatures identified, given the feature importance of the asynchronous vs other models based on different IMAs.

4.5 Summary

In this chapter, I have discussed the evaluation metrics used and how they are calculated to obtain the results. I also went over how the binary classification dataset was prepared for the classification of different IMAs and respective texting types such as Signal private asynchronous vs other. As the next step, I focused on 4 machine learning models used, which are Naive Bayes, Multi-Layer Perceptron, Random Forest, and Extreme Gradient Boosting. I went over the results individually for a group of 3 users, a group of 4 users, and 2 users, private, for asynchronous vs other, and synchronous vs other communications. Lastly, based on the results discussed, I showed that the Random Forest model performed better than the other 3 models, in Section 4.3. I presented the results of my analysis using the model's built-in feature importance. This was used to understand the patterns present. From Section 4.4 it is evident that there is a difference in the characteristic of how encrypted traffic is generated between the different texting via IMAs such as asynchronous vs other that utilize time and connection-based features, and synchronous which prioritizes packet size based features more. These network signatures were discovered based on the Random Forest training model results for each of the 7 IMAs.

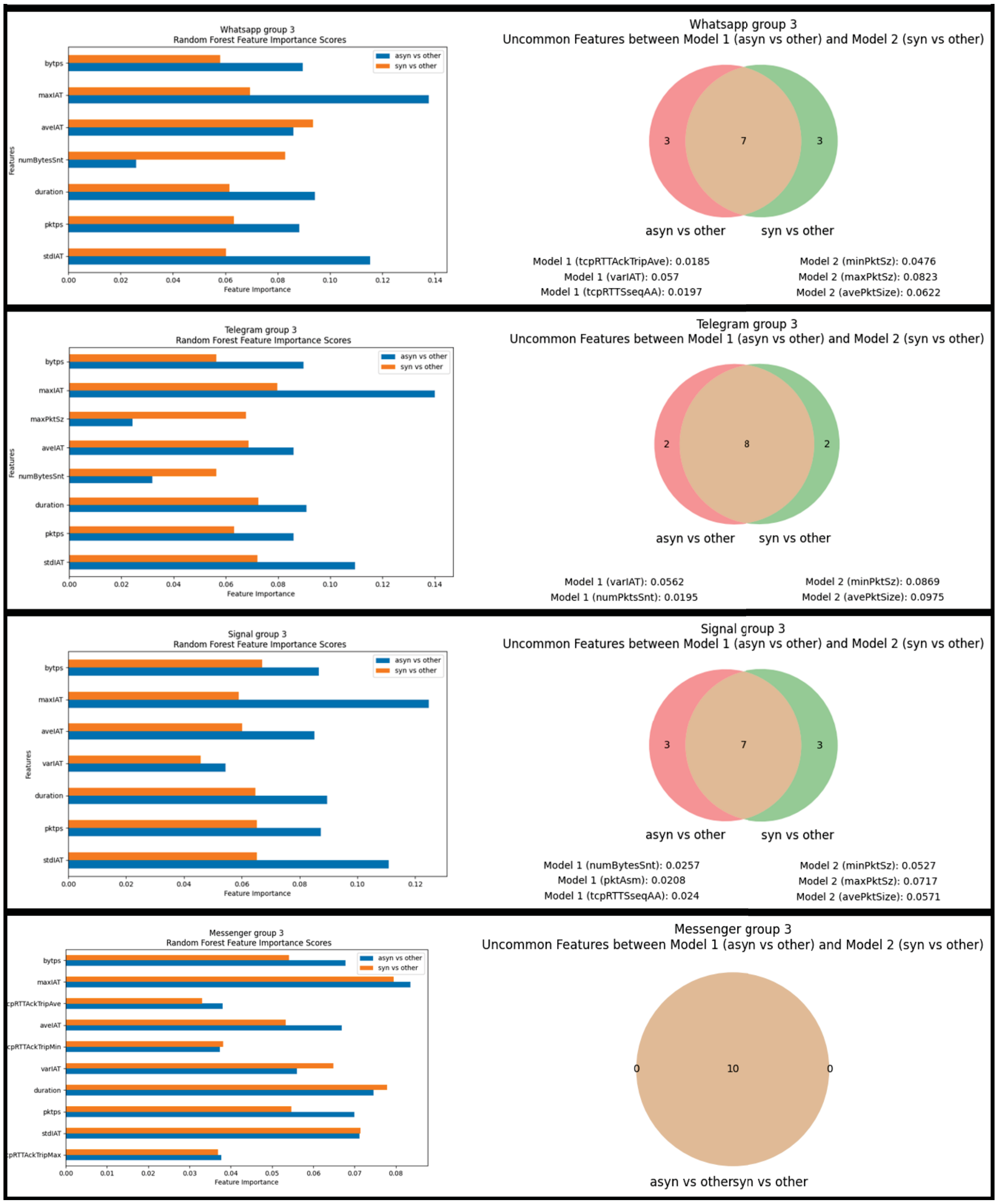

Figure 4.2: Comparing Random forest group of 3 user feature importance for WhatsApp, telegram, signal, and Messenger

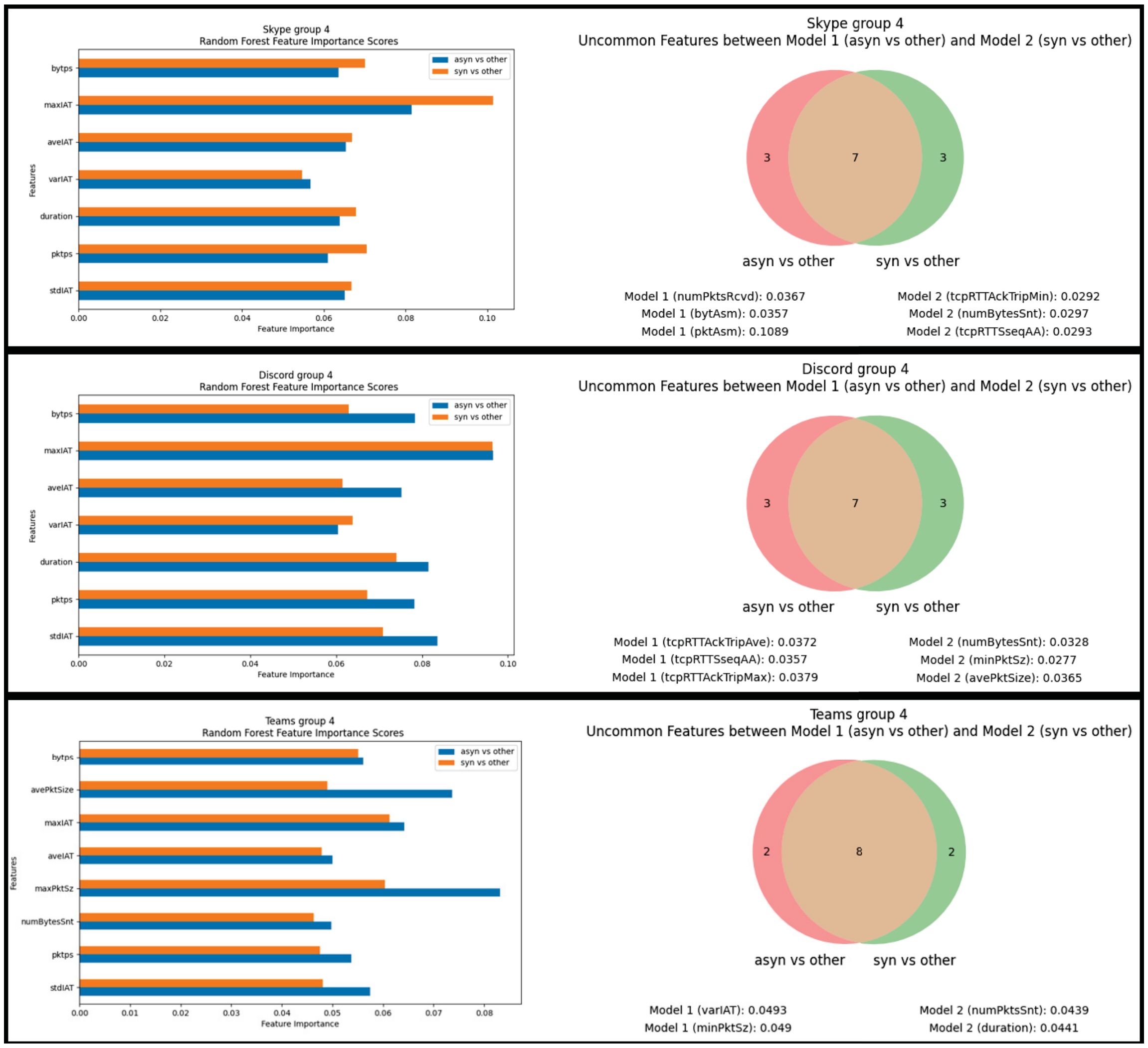

Figure 4.3: Comparing Random forest group of 4 user feature importance for Skype, discord, and Teams

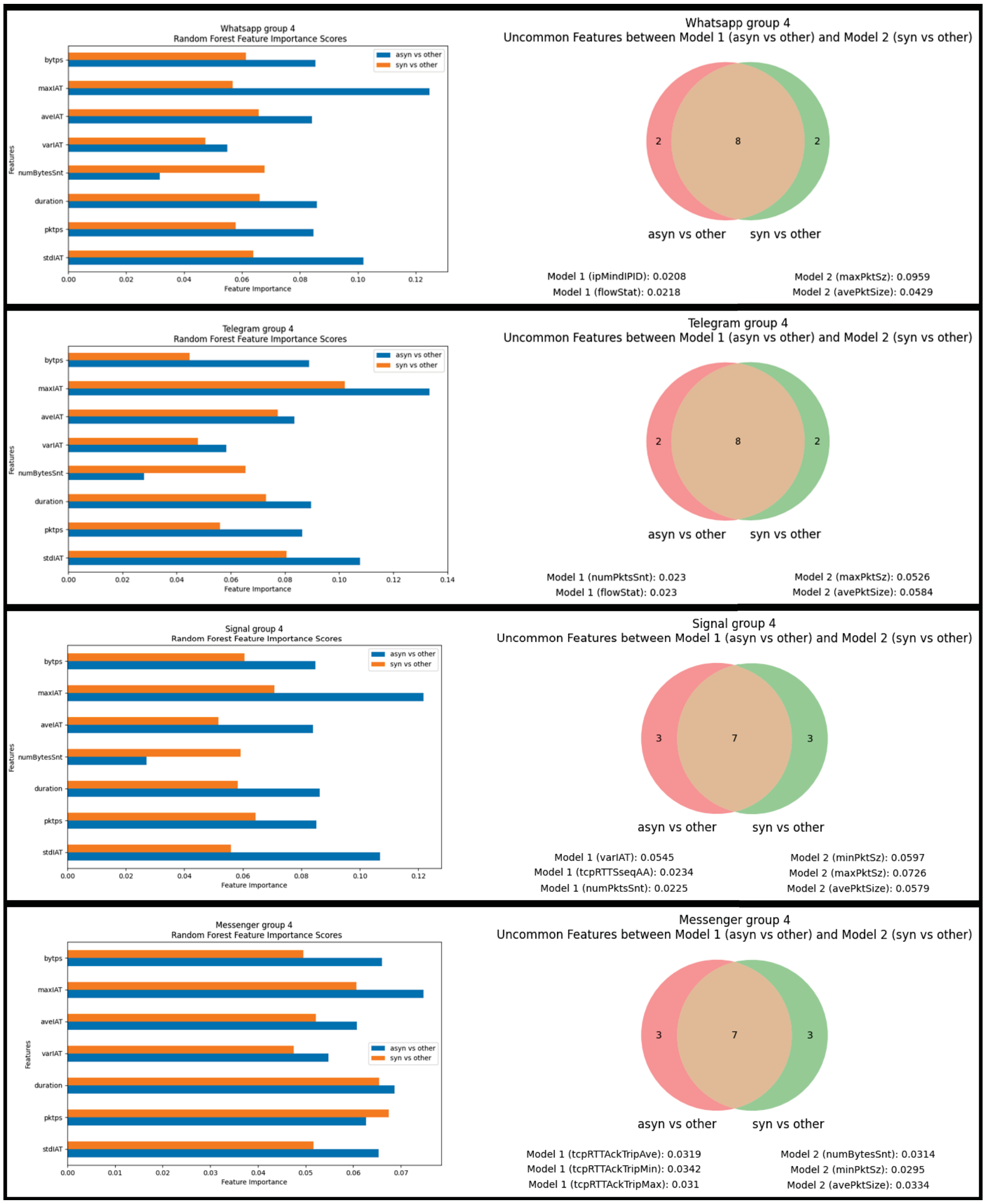

Figure 4.4: Comparing Random forest group of 4 user feature importance for WhatsApp, telegram, signal, and Messenger

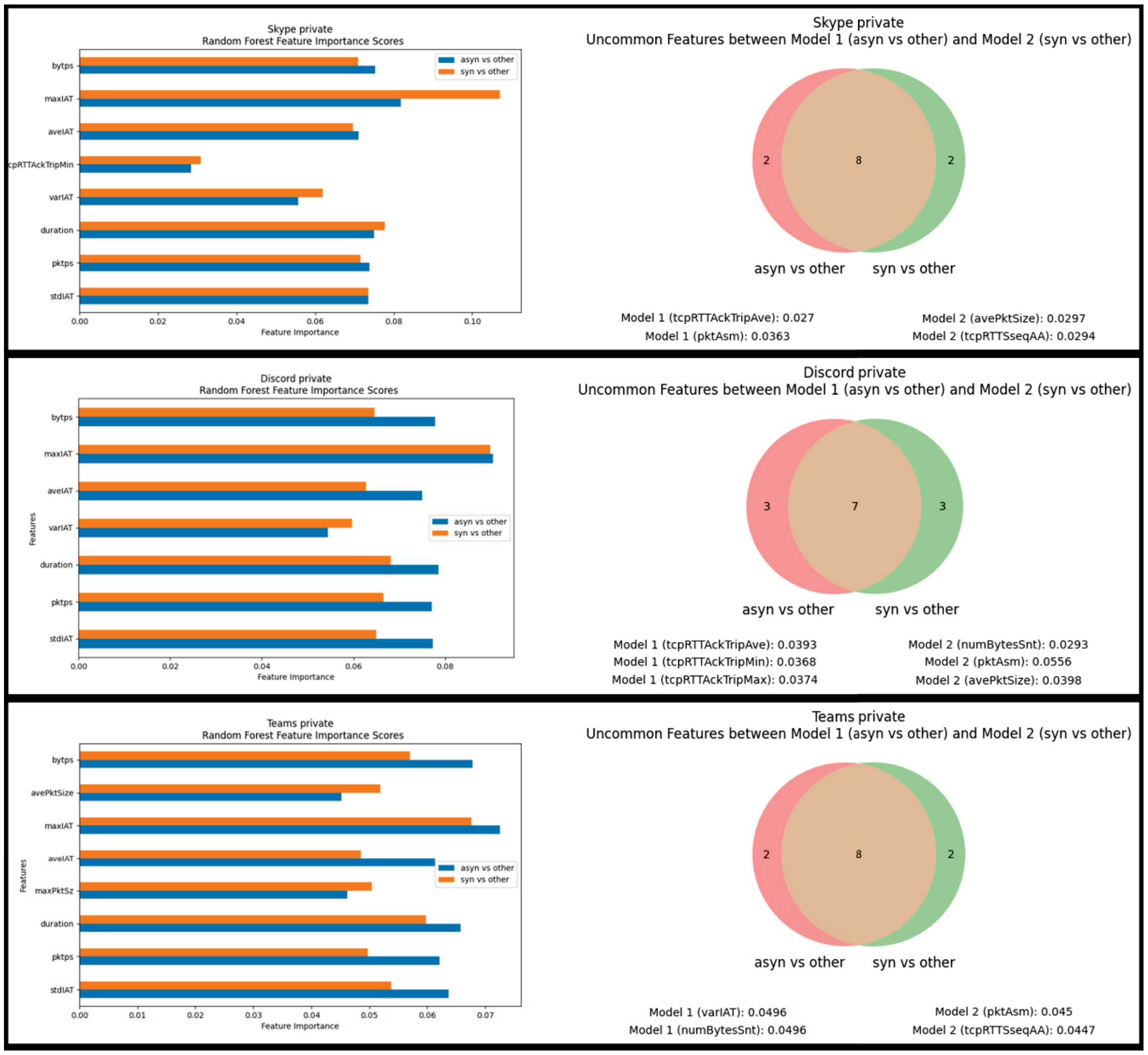

Figure 4.5: Comparing Random forest private user feature importance for Skype, discord, and Teams

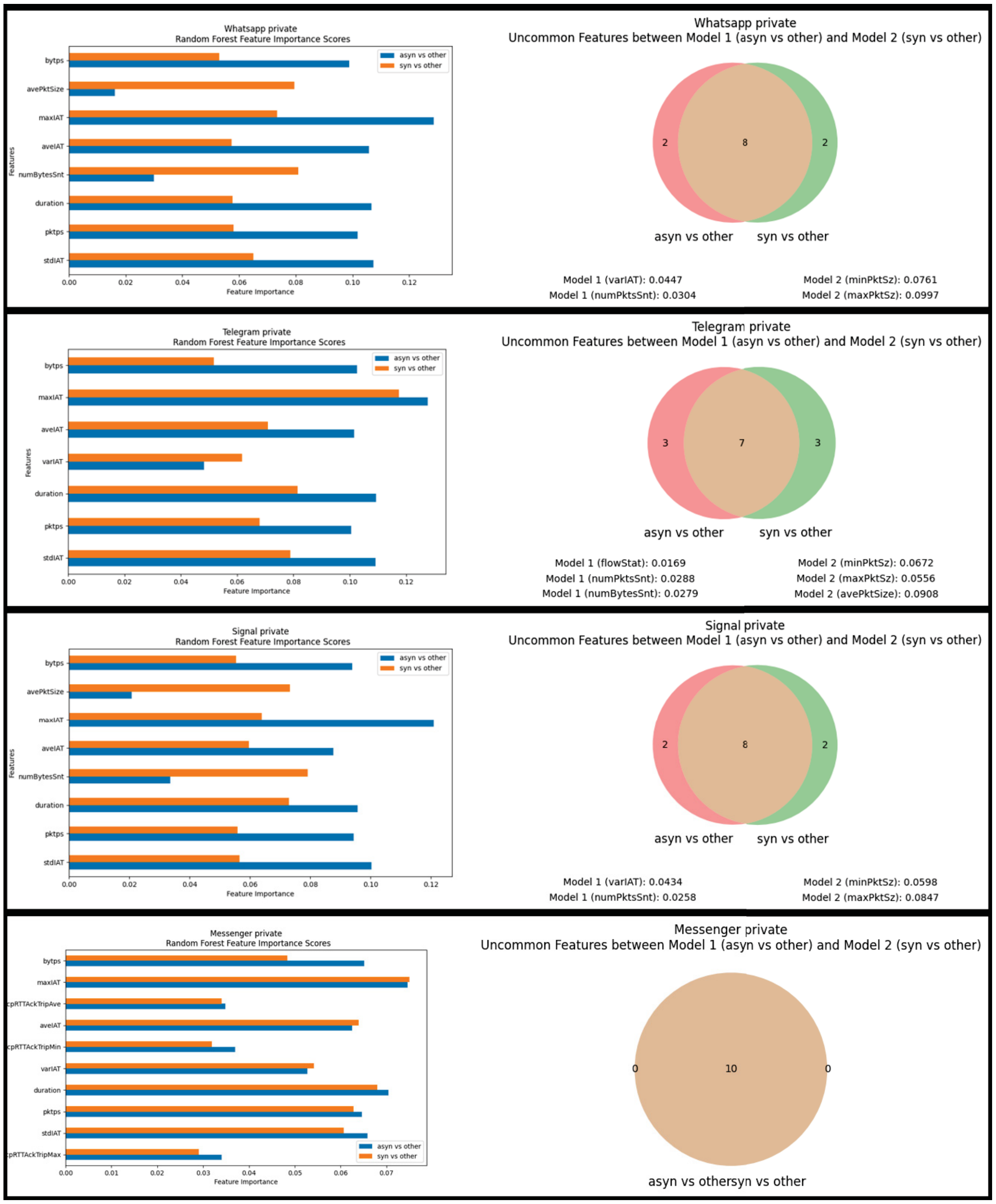

Figure 4.6: Comparing Random forest private user feature importance for WhatsApp, telegram, signal, and Messenger

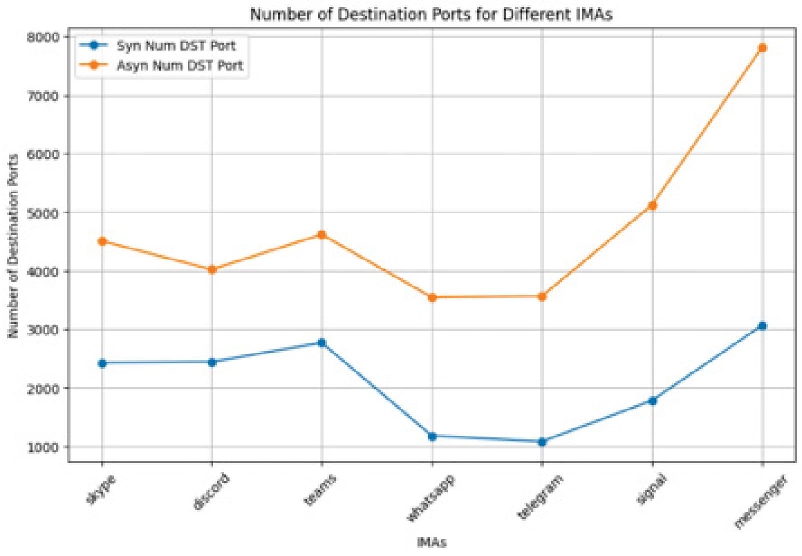

Figure 4.7: Number of Ports opened by each IMA

Chapter 5

Conclusion and Future Work

As shown in the Messaging application Market Data [47], there has been a steady growth in the number of users joining the instant messaging application platforms. Considering that instant messaging applications are also being used in the workplace, it is crucial to understand the network signatures of different types of user communication of such applications.

In this thesis, I have proposed, designed, and implemented a framework using individual devices to explore network traffic signatures using Instant Messaging Applications for texting. To achieve this I automated the process of emulating sending and receiving text messages as part of a communication between 2 users, i.e. private texting, a group of 3 users, and a group of 4 users, i.e. group texting. Messages were sent as synchronous and asynchronous texting behaviours with delays generated for '*read*' [38], '*type*' [37], and '*reply*' [38] activities based on real-world behaviours published in the literature. Synchronous and asynchronous texting is made to use the delay in such a way that it aims to mimic the delay seen in real-world texting. The data generation and collection were conducted based on these behaviours for different types of user communication using seven popular IMAs, namely Skype, Discord, Teams, Telegram, WhatsApp, Signal, and Messenger.

To ensure the ground truth-based labeling of the traffic is automated, IMA traffic was filtered from the entire collected traffic. Thus, a new filtering method was proposed. The proposed filtration method checks that for every 'ACK' present whether the connection initiation packet of the same port is also present or not. If the connection initiation packet from the same port is not present then the connection is labeled as the background connection and the IPs of the background connection are stored. All the IP classes that belong to the stored background connection are then removed from the dataset before splitting the data for training and testing for the machine learning models.

The flow extractor *Tranalyzer2* is then used on the collected data to export the flows for Machine Learning analysis. Then, Naive Bayes, Multi-Layer Perceptron, Random Forest, and Extreme Gradient Boosting machine learning algorithms are used. Furthermore, a feature importance analysis is utilized based on the best-performing classifier, namely the Random Forest trained model, in identifying and understanding the patterns present in different texting behaviours. This is done by studying the features that are common and uncommon between the trained models of the different texting behaviours for the types of user communication. The result of this analysis enabled me to identify the network traffic signatures for texting via instant messaging applications.

This thesis research used the ML classifiers by their default parameters. Thus, future work will explore generalization as well as hyperparameter tuning of ML classifiers for the better optimization and performance of all the classifiers used in this thesis. Utilizing binary classification, unique signatures are discovered for asynchronous and synchronous communications for each IMA under private and group texting in this thesis. Future work will explore all the IMAs collectively by performing a Multi-class classification approach. This might enable further signatures for understanding different communication behaviours. Furthermore, IMAs provide various functions in addition to sending private and group messages. Most IMAs support VoIP and video calls. Thus, future research will also study, and analyze their strengths and weaknesses. Also, publicly available data is scarce and future research might help to close the gap. Additionally, IMAs have started to include money transactions as a premium feature either via purchase or subscription. Moreover, some IMAs have introduced an electronic funds transfer which gives the user an option to send money to another user. As more functionalities are introduced, especially currency transactions into an IMA, more research is needed to understand their behaviours and identify their signatures for providing a better-operated and managed networks and services.

Bibliography

[1] Intuition. *Remote Working Statistics You Need to Know in 2024*. https://www.intuition.com/remote-working-statistics-you-need-to-know-in-2024/. Accessed: December 2023. Dec. 2023.

[2] Tirus Muya Maina. *Instant messaging an effective way of communication in workplace*. 2013. arXiv: 1310.8489 [cs.CY]. URL: https://arxiv.org/abs/1310.8489.

[3] Thijs van Ede et al. "FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic". In: *Network and Distributed System Security Symposium (NDSS)* 27 (). DOI: 10.14722/ndss.2020.24412. URL: https://par.nsf.gov/biblio/10192513.

[4] Abdurrahman Pektaş. "Proposal of Machine Learning Approach for Identification of Instant Messaging Applications in Raw Network Traffic". In: *International Journal of Intelligent Systems and Applications* 6 (June 2018). DOI: 10.18201/ijisae.2018642060.

[5] Zolboo Erdenebaatar et al. "Depicting Instant Messaging Encrypted Traffic Characteristics through an Empirical Study". In: *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*. 2023, pp. 1–10. DOI: 10.1109/ICCCN58024.2023.10230093.

[6] Network Information Management and Security Group (NIMS). *NIMS Dataset*. https://projects.cs.dal.ca/projectx/Download.html. 2017.

[7] 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. Darmstadt, Germany: Association for Computing Machinery, 2016, pp. 61–66. ISBN: 9781450342704. DOI: 10.1145/2939918.2939929. URL: https://doi.org/10.1145/2939918.2939929.

[8] Zolboo Erdenebaatar et al. “Instant Messaging Application Encrypted Traffic Generation System”. In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 2023, pp. 1–3. DOI: 10.1109/NOMS56928.2023.10154297.

[9] Zolboo Erdenebaatar et al. “Analyzing Traffic Characteristics of Instant Messaging Applications on Android Smartphones”. In: *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*. 2023, pp. 1–5. DOI: 10.1109/NOMS56928.2023.10154386.

[10] Qinglong Wang et al. “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)*. 2015, pp. 433–441. DOI: 10.1109/CNS.2015.7346855.

[11] Ly Vu et al. “Time Series Analysis for Encrypted Traffic Classification: A Deep Learning Approach”. In: *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*. 2018, pp. 121–126. DOI: 10.1109/ISCIT.2018.8587975.

[12] Chang Liu et al. “MaMPF: Encrypted Traffic Classification Based on Multi-Attribute Markov Probability Fingerprints”. In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, pp. 1–10. DOI: 10.1109/IWQoS.2018.8624124.

[13] Zhuang Zou et al. “Encrypted Traffic Classification with a Convolutional Long Short-Term Memory Neural Network”. In: *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2018, pp. 329–334. DOI: 10.1109/HPCC/SmartCity/DSS.2018.00074.

[14] Pedro Casas et al. “Fingerprinting Web Pages and Smartphone Apps from Encrypted Network Traffic with WebScanner”. In: *2022 IEEE 11th International Conference on Cloud Networking (CloudNet)*. 2022, pp. 1–9. DOI: 10.1109/CloudNet55617.2022.9978877.

[15] Qiuning Ren, Chao Yang, and Jianfeng Ma. "App identification based on encrypted multi-smartphone sources traffic fingerprints". In: *Computer Networks* 201 (2021), p. 108590. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108590. URL: https://www.sciencedirect.com/science/article/pii/S138912862100493X.

[16] Jay Shah. "Detection of malicious encrypted web traffic using machine learning". In: (2018).

[17] Jianhua Sun et al. "Automatically identifying apps in mobile traffic". In: *Concurrency and Computation: Practice and Experience* 28.14 (2016), pp. 3927–3941. DOI: https://doi.org/10.1002/cpe.3703. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3703. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3703.

[18] Seunghun Cha and Hyoungshick Kim. "Detecting Encrypted Traffic: A Machine Learning Approach". In: Mar. 2017, pp. 54–65. ISBN: 978-3-319-56548-4. DOI: 10.1007/978-3-319-56549-1_5.

[19] Shengbao Li et al. "FusionTC: Encrypted App Traffic Classification Using Decision-Level Multimodal Fusion Learning of Flow Sequence". In: *Wireless Communications and Mobile Computing* 2023.1 (2023), p. 9118153. DOI: https://doi.org/10.1155/2023/9118153. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/9118153.

[20] Zihao Wang, Kar Wai Fok, and Vrizlynn L.L. Thing. "Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study". In: *Computers amp; Security* 113 (Feb. 2022), p. 102542. ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102542. URL: http://dx.doi.org/10.1016/j.cose.2021.102542.

[21] Meta Platforms, Inc. *Messenger for Desktop Download.* Accessed: December 2023. Dec. 2023. URL: https://www.messenger.com/desktop.

[22] Laiby Thomas and Subramanya Bhat. "A Comprehensive Overview of Telegram Services - A Case Study". In: *International Journal of Case Studies in Business, IT, and Education (IJCSBE)* 6.1 (2022), pp. 288–301. DOI: https://doi.org/10.5281/zenodo.6513296.

[23] Telegram Messenger LLP. *Telegram Desktop Download*. Accessed: December 2023. Dec. 2023. URL: https://desktop.telegram.org/.

[24] Jack Vance. *STS Research Paper*. https://libraetd.lib.virginia.edu/downloads/9593tv939?filename=Vance_Jack_STS_Research_Paper.pdf. Accessed: December 2023. Dec. 2023.

[25] TechReport. *Discord Statistics*. https://techreport.com/statistics/software-web/discord-statistics/. Accessed: December 2023. Dec. 2023.

[26] Discord Inc. *Discord Download*. Accessed: December 2023. Dec. 2023. URL: https://discord.com/download.

[27] Corina-Elena Bogos, Razvan Mocanu, and Emil Simion. "A security analysis comparison between Signal, WhatsApp and Telegram". In: (Jan. 2023).

[28] Signal Foundation. *Signal for Windows Download*. Accessed: December 2023. Dec. 2023. URL: https://signal.org/download/windows/.

[29] Rebecca Mirick and Stephanie Wladkowski. "Skype in Qualitative Interviews: Participant and Researcher Perspectives". In: *Qualitative Report* 24 (Dec. 2019), pp. 3061–3072. DOI: 10.46743/2160-3715/2019.3632.

[30] Microsoft. *Another Microsoft App*. https://apps.microsoft.com/detail/9wzdncrfj364?hl=en-us&gl=US. Accessed: December 2023. Dec. 2023.

[31] Business of Apps. *Microsoft Teams Statistics*. https://www.businessofapps.com/data/microsoft-teams-statistics/. Accessed: December 2023. Dec. 2023.

[32] Microsoft Corporation. *Microsoft Teams Download for Desktop*. Accessed: December 2023. Dec. 2023. URL: https://www.microsoft.com/en-ca/microsoft-teams/download-app#download-for-desktop.

[33] Focus on Business. *Microsoft Teams Grew Over 90% in 2020 Due to Pandemic: 145M Daily Active Users in 2021*. https://focusonbusiness.eu/en/news/microsoft-teams-grew-over-90-in-2020-due-to-pandemic-145m-daily-active-users-in-2021/4223. Accessed: December 2023. Dec. 2023.

[34] Rasayel. *WhatsApp User Statistics*. https://learn.rasayel.io/en/blog/whatsapp-user-statistics/. Accessed: December 2023. Dec. 2023.

[35] Microsoft. *Microsoft App*. https://apps.microsoft.com/detail/9nksqgp7f2nh?hl=en-US&gl=US. Accessed: December 2023. Dec. 2023.

[36] Bogdan Gliwa et al. "SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization". In: *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Ed. by Lu Wang et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 70–79. DOI: 10.18653/v1/D19-5409. URL: https://aclanthology.org/D19-5409.

[37] Nancy A. Baker and Mark S. Redfern. "The Association between Computer Typing Style and Typing Speeds". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 51.15 (2007), pp. 869–873. DOI: 10.1177/154193120705101501.

[38] Bongeka Mpofu. "University Students Use of Computers and Mobile Devices for Learning and their Reading Speed on Different Platforms". In: *Universal Journal of Educational Research* 4 (Apr. 2016), pp. 926–932. DOI: 10.13189/ujer.2016.040430.

[39] Avi Rosenfeld et al. "WhatsApp usage patterns and prediction of demographic characteristics without access to message content". In: *Demographic Research* 39 (Sept. 2018), pp. 647–670. DOI: 10.4054/DemRes.2018.39.22.

[40] Microsoft. *Pktmon - Packet Monitor*. https://learn.microsoft.com/en-us/windows-server/networking/technologies/pktmon/pktmon. Accessed: December 2023. Dec. 2023.

[41] Microsoft. *Pktmon ETL to PCAP File Conversion*. https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/pktmon-etl2pcap?source=recommendations. Accessed: December 2023. Dec. 2023.

[42] Tranalyzer. *T2Py Tutorial*. https://tranalyzer.com/tutorial/t2py. Accessed: May 2023. May 2023.

[43] Tranalyzer. *Tranalyzer Documentation*. https://tranalyzer.com/download/doc/documentation.pdf. Accessed: May 2023. May 2023.

[44] Ethem Alpaydin. *Introduction to Machine Learning*. Accessed: December 2023. MIT Press, 2020. ISBN: 9780262043793. URL: https://mitpress.mit.edu/9780262043793/introduction-to-machine-learning/.

[45] Towards AI. *Confusion Matrix*. 2023. URL: https://pub.towardsai.net/confusion-matrix-179b9c758b55.

[46] Alireza Bahramali et al. "Practical Traffic Analysis Attacks on Secure Messaging Applications". In: Jan. 2020. DOI: 10.14722/ndss.2020.24347.

[47] Business of Apps. *Messaging App Market Data (2023)*. Accessed: December 2023. Dec. 2023. URL: https://www.businessofapps.com/data/messaging-app-market/.

Appendix A

Feasture Description

A.1 Flow extracted from *Tranalyzer*

Col No.	Name	Description
1	dir	Flow direction
2	flowInd	Flow index
3	flowStat	Flow status and warnings
4	timeFirst	Date time of first packet
5	timeLast	Date time of last packet
6	duration	Flow duration
7	numHdrDesc	Number of different headers descriptions
8	numHdrs	Number of headers (depth) in hdrDesc
9	hdrDesc	Headers description
10	srcMac	Mac source
11	dstMac	Mac destination
12	ethType	Ethernet type
13	ethVlanID	VLAN IDs
14	srcIP	Source IP address
15	srcIPCC	Source IP country
16	srcIPOrg	Source IP organisation
17	srcPort	Source port
18	dstIP	Destination IP address
19	dstIPCC	Destination IP country
20	dstIPOrg	Destination IP organisation

Col No.	Name	Description
21	dstPort	Destination port
22	l4Proto	Layer 4 protocol
23	macStat	macRecorder status
24	macPairs	Number of distinct source/destination MAC addresses pairs
25	srcMac_dstMac_numP	Source/destination MAC address, number of packets of MAC address combination
26	srcMacLbl_dstMacLbl	Source/destination MAC label
27	dstPortClassN	Port-based classification of the destination port number
28	dstPortClass	Port-based classification of the destination port name
29	numPktsSnt	Number of transmitted packets
30	numPktsRcvd	Number of received packets
31	numBytesSnt	Number of transmitted bytes
32	numBytesRcvd	Number of received bytes
33	minPktSz	Minimum layer 3 packet size
34	maxPktSz	Maximum layer 3 packet size
35	avePktSize	Average layer 3 packet size
36	varPktSize	Variance layer 3 packet size
37	stdPktSize	Standard deviation layer 3 packet size
38	minIAT	Minimum inter-arrival time
39	maxIAT	Maximum inter-arrival time
40	aveIAT	Average inter-arrival time
41	varIAT	Variance inter-arrival time

Col No.	**Name**	**Description**
42	stdIAT	Standard deviation inter-arrival time
43	pktps	Sent packets per second
44	bytps	Sent bytes per second
45	pktAsm	Packet stream asymmetry
46	bytAsm	Byte stream asymmetry
47	tcpFStat	tcpFlags status
48	ipMindIPID	IP minimum delta IP ID
49	ipMaxdIPID	IP maximum delta IP ID
50	ipMinTTL	IP minimum Time to Live
51	ipMaxTTL	IP maximum Time to Live
52	ipTTLChg	IP Time to Live change count
53	ipToS	IP Type of Service hex
54	ipFlags	IP aggregated flags
55	ipOptCnt	IP options count
56	ipOptCpCl_Num	IP aggregated options, copy-class and number
57	ip6OptCntHH_D	IPv6 Hop-by-Hop destination option counts
58	ip6OptHH_D	IPv6 aggregated Hop-by-Hop destination options
59	tcpISeqN	TCP initial sequence number
60	tcpPSeqCnt	TCP packet seq count
61	tcpSeqSntBytes	TCP sent seq diff bytes
62	tcpSeqFaultCnt	TCP sequence number fault count
63	tcpPAckCnt	TCP packet ACK count
64	tcpFlwLssAckRcvdBytes	TCP flawless ACK received bytes

Col No.	Name	Description
65	tcpAckFaultCnt	TCP ACK number fault count
66	tcpBFlgtMx	TCP Bytes in Flight MAX
67	tcpInitWinSz	TCP initial effective window size
68	tcpAveWinSz	TCP average effective window size
69	tcpMinWinSz	TCP minimum effective window size
70	tcpMaxWinSz	TCP maximum effective window size
71	tcpWinSzDwnCnt	TCP effective window size change down count
72	tcpWinSzUpCnt	TCP effective window size change up count
73	tcpWinSzChgDirCnt	TCP effective window size direction change count
74	tcpWinSzThRt	TCP packet count ratio below window size WINMIN threshold
75	tcpFlags	TCP aggregated protocol flags (FINACK, SYNACK, RSTACK, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN)
76	tcpAnomaly	TCP aggregated header anomaly flags
77	tcpOptPktCnt	TCP options packet count
78	tcpOptCnt	TCP options count
79	tcpOptions	TCP aggregated options
80	tcpMSS	TCP maximum segment size
81	tcpWS	TCP window scale

Col No.	Name	Description
82	tcpMPTBF	TCP MPTCP type bitfield
83	tcpMPF	TCP MPTCP flags
84	tcpMPAID	TCP MPTCP address ID
85	tcpMPDSSF	TCP MPTCP DSS flags
86	tcpTmS	TCP time stamp
87	tcpTmER	TCP time echo reply
88	tcpEcI	TCP estimated counter increment
89	tcpUtm	TCP estimated up time
90	tcpBtm	TCP estimated boot time
91	tcpSSASAATrip	TCP trip time (A: SYN, SYN-ACK, B: SYN-ACK, ACK)
92	tcpRTTAckTripMin	TCP ACK trip min
93	tcpRTTAckTripMax	TCP ACK trip max
94	tcpRTTAckTripAve	TCP ACK trip average
95	tcpRTTAckTripJitAve	TCP ACK trip jitter average
96	tcpRTTSseqAA	TCP round trip time (A: SYN, SYN-ACK, ACK, B: ACK-ACK)
97	tcpRTTAckJitAve	TCP ACK round trip average jitter
98	tcpStatesAFlags	TCP state machine anomalies
99	icmpStat	ICMP Status
100	icmpTCcnt	ICMP type code count
101	icmpBFTypH_TypL_Code	ICMP Aggregated type H (>128), L (<32) & code bit field
102	icmpTmGtw	ICMP time/gateway
103	icmpEchoSuccRatio	ICMP Echo reply/request success ratio

Col No.	**Name**	**Description**
104	icmpPFindex	ICMP parent flowIndex
105	connSip	Number of unique source IPs
106	connDip	Number of unique destination IPs
107	connSipDip	Number of connections between source and destination IP
108	connSipDprt	Number of connections between source IP and destination port
109	connF	The f number: connSipDprt / connSip [EXPERIMENTAL]

Appendix B

Extra Results

B.1 10-Fold Cross Validation: *Training Dataset*

Table B.1: Random Forest group 3 asyn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.775	0.761-0.794	0.684	0.672-0.692	0.770	0.756-0.781
F1 Score	0.788	0.770-0.807	0.705	0.693-0.720	0.781	0.769-0.790
Recall	0.823	0.800-0.850	0.749	0.726-0.768	0.831	0.820-0.847
Precision	0.754	0.739-0.770	0.664	0.652-0.681	0.737	0.721-0.755

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.621	0.607-0.630	0.623	0.619-0.630	0.630	0.620-0.638	0.685	0.677-0.694
F1 Score	0.678	0.671-0.687	0.687	0.682-0.694	0.671	0.665-0.681	0.701	0.687-0.717
Recall	0.793	0.772-0.808	0.820	0.809-0.830	0.753	0.730-0.771	0.738	0.712-0.758
Precision	0.592	0.578-0.604	0.590	0.582-0.598	0.606	0.594-0.612	0.667	0.653-0.681

Table B.2: Random Forest group 3 syn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.823	0.809-0.838	0.823	0.804-0.838	0.877	0.858-0.891
F1 Score	0.823	0.800-0.844	0.831	0.809-0.843	0.878	0.851-0.889
Recall	0.848	0.823-0.881	0.858	0.845-0.866	0.899	0.880-0.912
Precision	0.795	0.765-0.825	0.804	0.774-0.834	0.859	0.824-0.884

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.906	0.891-0.916	0.903	0.890-0.919	0.879	0.861-0.895	0.879	0.866-0.887
F1 Score	0.907	0.894-0.915	0.901	0.885-0.919	0.879	0.860-0.902	0.884	0.874-0.892
Recall	0.925	0.912-0.932	0.915	0.902-0.941	0.905	0.875-0.923	0.922	0.898-0.934
Precision	0.889	0.863-0.909	0.891	0.863-0.913	0.852	0.820-0.881	0.850	0.842-0.854

Table B.3: Random Forest group 4 asyn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.865	0.861-0.871	0.699	0.682-0.707	0.824	0.818-0.832
F1 Score	0.868	0.861-0.873	0.714	0.702-0.723	0.825	0.816-0.831
Recall	0.895	0.885-0.907	0.744	0.735-0.762	0.806	0.795-0.814
Precision	0.842	0.831-0.862	0.682	0.671-0.696	0.847	0.843-0.850

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.645	0.639-0.652	0.630	0.620-0.648	0.647	0.635-0.654	0.734	0.727-0.740
F1 Score	0.610	0.605-0.622	0.592	0.576-0.609	0.618	0.609-0.625	0.744	0.737-0.748
Recall	0.552	0.544-0.564	0.534	0.518-0.551	0.571	0.565-0.576	0.776	0.763-0.786
Precision	0.681	0.671-0.700	0.663	0.647-0.682	0.675	0.665-0.689	0.715	0.702-0.723

Table B.4: Random Forest group 4 syn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.825	0.810-0.848	0.833	0.814-0.852	0.911	0.904-0.919
F1 Score	0.826	0.810-0.846	0.835	0.822-0.856	0.910	0.900-0.921
Recall	0.848	0.828-0.876	0.873	0.855-0.897	0.933	0.920-0.948
Precision	0.809	0.782-0.829	0.802	0.763-0.825	0.891	0.859-0.911

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.918	0.906-0.928	0.923	0.911-0.934	0.883	0.872-0.896	0.897	0.885-0.918
F1 Score	0.917	0.903-0.931	0.921	0.906-0.931	0.882	0.862-0.897	0.901	0.889-0.922
Recall	0.935	0.924-0.950	0.934	0.920-0.947	0.897	0.874-0.921	0.937	0.918-0.960
Precision	0.898	0.884-0.909	0.908	0.886-0.927	0.865	0.836-0.883	0.866	0.837-0.898

Table B.5: Random Forest private asyn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.745	0.731-0.761	0.703	0.686-0.719	0.748	0.74-0.761
F1 Score	0.758	0.748-0.769	0.718	0.699-0.732	0.757	0.744-0.784
Recall	0.787	0.766-0.799	0.756	0.738-0.778	0.798	0.778-0.812
Precision	0.731	0.721-0.749	0.685	0.666-0.716	0.717	0.704-0.759

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.699	0.683-0.708	0.684	0.676-0.693	0.708	0.698-0.715	0.704	0.689-0.722
F1 Score	0.741	0.729-0.751	0.731	0.72-0.739	0.745	0.736-0.758	0.720	0.704-0.74
Recall	0.856	0.845-0.866	0.854	0.848-0.862	0.845	0.835-0.854	0.763	0.744-0.789
Precision	0.653	0.64-0.668	0.639	0.623-0.649	0.667	0.658-0.678	0.680	0.663-0.698

Table B.6: Random Forest private syn vs other on `Training Dataset`

	Skype		Discord		Teams	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.815	0.798-0.846	0.811	0.789-0.832	0.872	0.857-0.888
F1 Score	0.815	0.786-0.847	0.816	0.804-0.836	0.873	0.858-0.882
Recall	0.828	0.798-0.855	0.847	0.822-0.870	0.890	0.863-0.906
Precision	0.799	0.765-0.830	0.791	0.764-0.816	0.861	0.845-0.889

	WhatsApp		Telegram		Signal		Messenger	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.926	0.919-0.942	0.897	0.885-0.906	0.873	0.851-0.886	0.853	0.834-0.874
F1 Score	0.925	0.909-0.941	0.896	0.874-0.911	0.875	0.859-0.891	0.857	0.845-0.876
Recall	0.950	0.929-0.963	0.910	0.897-0.921	0.894	0.877-0.912	0.893	0.858-0.911
Precision	0.903	0.887-0.921	0.882	0.870-0.900	0.854	0.834-0.869	0.829	0.803-0.878

Table B.7: Extreme Gradient Boosting group 3 asyn vs other on `Training Dataset`

	Skype		Discord		Teams	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.763	0.75-0.78	0.669	0.661-0.685	0.771	0.767-0.777
F1 Score	0.778	0.765-0.794	0.711	0.702-0.728	0.794	0.788-0.803
Recall	0.820	0.804-0.844	0.811	0.796-0.826	0.901	0.887-0.913
Precision	0.740	0.73-0.754	0.633	0.618-0.654	0.710	0.695-0.724

	WhatsApp		Telegram		Signal		Messenger	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.615	0.601-0.625	0.619	0.613-0.625	0.614	0.605-0.620	0.670	0.663-0.676
F1 Score	0.686	0.672-0.694	0.694	0.687-0.702	0.657	0.642-0.667	0.703	0.692-0.714
Recall	0.837	0.819-0.859	0.861	0.838-0.880	0.737	0.707-0.759	0.781	0.770-0.796
Precision	0.581	0.569-0.591	0.582	0.571-0.589	0.593	0.577-0.601	0.639	0.623-0.652

Table B.8: Extreme Gradient Boosting group 3 syn vs other on `Training Dataset`

	Skype		Discord		Teams	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.811	0.787-0.839	0.828	0.814-0.840	0.879	0.864-0.898
F1 Score	0.818	0.794-0.849	0.835	0.817-0.853	0.881	0.863-0.903
Recall	0.860	0.830-0.887	0.874	0.862-0.895	0.908	0.898-0.923
Precision	0.780	0.753-0.826	0.800	0.772-0.815	0.855	0.829-0.885

	WhatsApp		Telegram		Signal		Messenger	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.908	0.900-0.915	0.910	0.900-0.924	0.879	0.862-0.891	0.877	0.863-0.891
F1 Score	0.907	0.897-0.918	0.908	0.897-0.922	0.881	0.865-0.894	0.883	0.872-0.892
Recall	0.919	0.909-0.934	0.918	0.902-0.939	0.921	0.900-0.942	0.924	0.903-0.936
Precision	0.895	0.871-0.904	0.899	0.881-0.913	0.846	0.817-0.877	0.845	0.830-0.858

Table B.9: Extreme Gradient Boosting group 4 asyn vs other on `Training Dataset`

	Skype		Discord		Teams	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.858	0.851-0.866	0.688	0.678-0.696	0.811	0.803-0.818
F1 Score	0.863	0.856-0.873	0.716	0.703-0.727	0.801	0.791-0.808
Recall	0.907	0.889-0.928	0.784	0.768-0.800	0.733	0.723-0.741
Precision	0.824	0.813-0.851	0.660	0.644-0.676	0.882	0.873-0.892

	WhatsApp		Telegram		Signal		Messenger	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.637	0.628-0.644	0.626	0.616-0.639	0.637	0.631-0.645	0.720	0.716-0.732
F1 Score	0.601	0.591-0.614	0.580	0.564-0.594	0.595	0.590-0.601	0.740	0.734-0.748
Recall	0.545	0.532-0.563	0.515	0.497-0.534	0.531	0.523-0.541	0.799	0.781-0.811
Precision	0.671	0.662-0.691	0.664	0.652-0.676	0.676	0.665-0.682	0.689	0.680-0.699

Table B.10: Extreme Gradient Boosting group 4 syn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.821	0.805-0.841	0.829	0.819-0.843	0.906	0.895-0.916
F1 Score	0.824	0.807-0.848	0.835	0.821-0.849	0.907	0.897-0.915
Recall	0.855	0.832-0.889	0.881	0.855-0.904	0.929	0.910-0.964
Precision	0.796	0.770-0.813	0.793	0.765-0.815	0.886	0.856-0.896

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.919	0.905-0.929	0.922	0.909-0.937	0.878	0.867-0.885	0.897	0.873-0.919
F1 Score	0.918	0.905-0.929	0.921	0.903-0.936	0.878	0.862-0.888	0.902	0.877-0.924
Recall	0.937	0.922-0.954	0.934	0.920-0.945	0.906	0.891-0.932	0.944	0.918-0.956
Precision	0.900	0.878-0.916	0.908	0.887-0.927	0.851	0.825-0.870	0.863	0.834-0.896

Table B.11: Extreme Gradient Boosting private asyn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.750	0.739-0.766	0.701	0.685-0.721	0.756	0.739-0.769
F1 Score	0.764	0.750-0.780	0.718	0.706-0.738	0.775	0.758-0.794
Recall	0.796	0.783-0.816	0.755	0.731-0.786	0.863	0.846-0.881
Precision	0.734	0.717-0.756	0.684	0.663-0.712	0.704	0.683-0.740

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.700	0.686-0.712	0.688	0.680-0.697	0.711	0.698-0.724	0.703	0.683-0.719
F1 Score	0.742	0.727-0.752	0.737	0.725-0.746	0.746	0.735-0.754	0.726	0.708-0.744
Recall	0.856	0.844-0.860	0.868	0.857-0.880	0.841	0.824-0.855	0.792	0.746-0.818
Precision	0.655	0.639-0.670	0.640	0.625-0.652	0.670	0.661-0.681	0.671	0.642-0.693

Table B.12: Extreme Gradient Boosting private syn vs other on `Training Dataset`

	Skype		**Discord**		**Teams**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.821	0.801-0.853	0.818	0.798-0.840	0.883	0.870-0.897
F1 Score	0.827	0.806-0.861	0.826	0.814-0.848	0.886	0.872-0.896
Recall	0.867	0.834-0.901	0.872	0.852-0.891	0.915	0.902-0.932
Precision	0.791	0.750-0.824	0.785	0.762-0.810	0.859	0.844-0.877

	WhatsApp		**Telegram**		**Signal**		**Messenger**	
	Avg	**Range**	**Avg**	**Range**	**Avg**	**Range**	**Avg**	**Range**
Accuracy	0.933	0.923-0.946	0.908	0.898-0.924	0.881	0.863-0.898	0.858	0.831-0.874
F1 Score	0.932	0.917-0.945	0.907	0.896-0.923	0.883	0.869-0.899	0.865	0.843-0.884
Recall	0.955	0.948-0.963	0.919	0.901-0.936	0.921	0.903-0.941	0.906	0.885-0.933
Precision	0.910	0.886-0.933	0.895	0.879-0.915	0.848	0.826-0.861	0.828	0.797-0.869

Appendix C

Feature Importance

C.1 Extreme Gradient Boosting

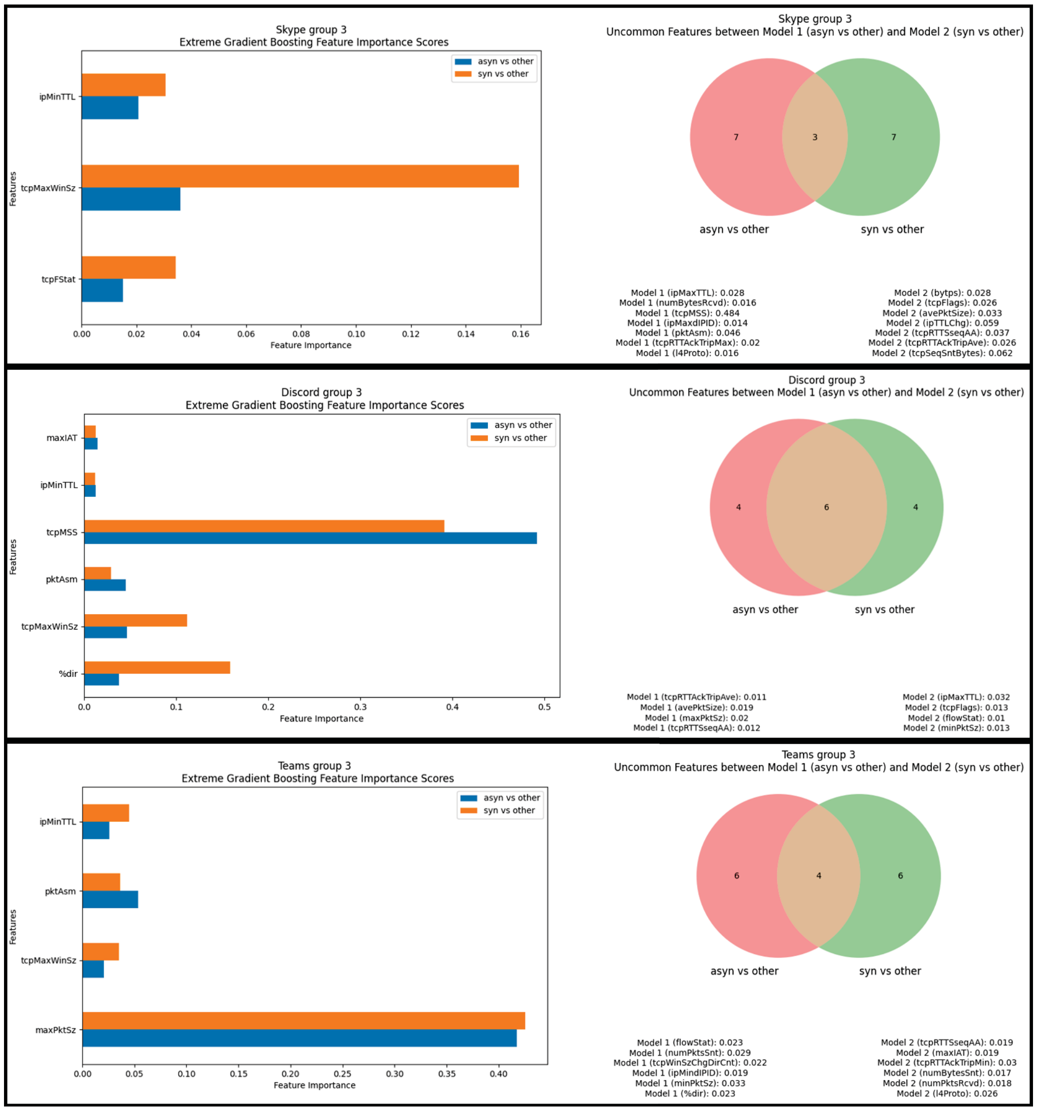

Figure C.1: Comparing Extreme Gradient Boosting Group 3 user feature importance for Skype, discord, and Teams

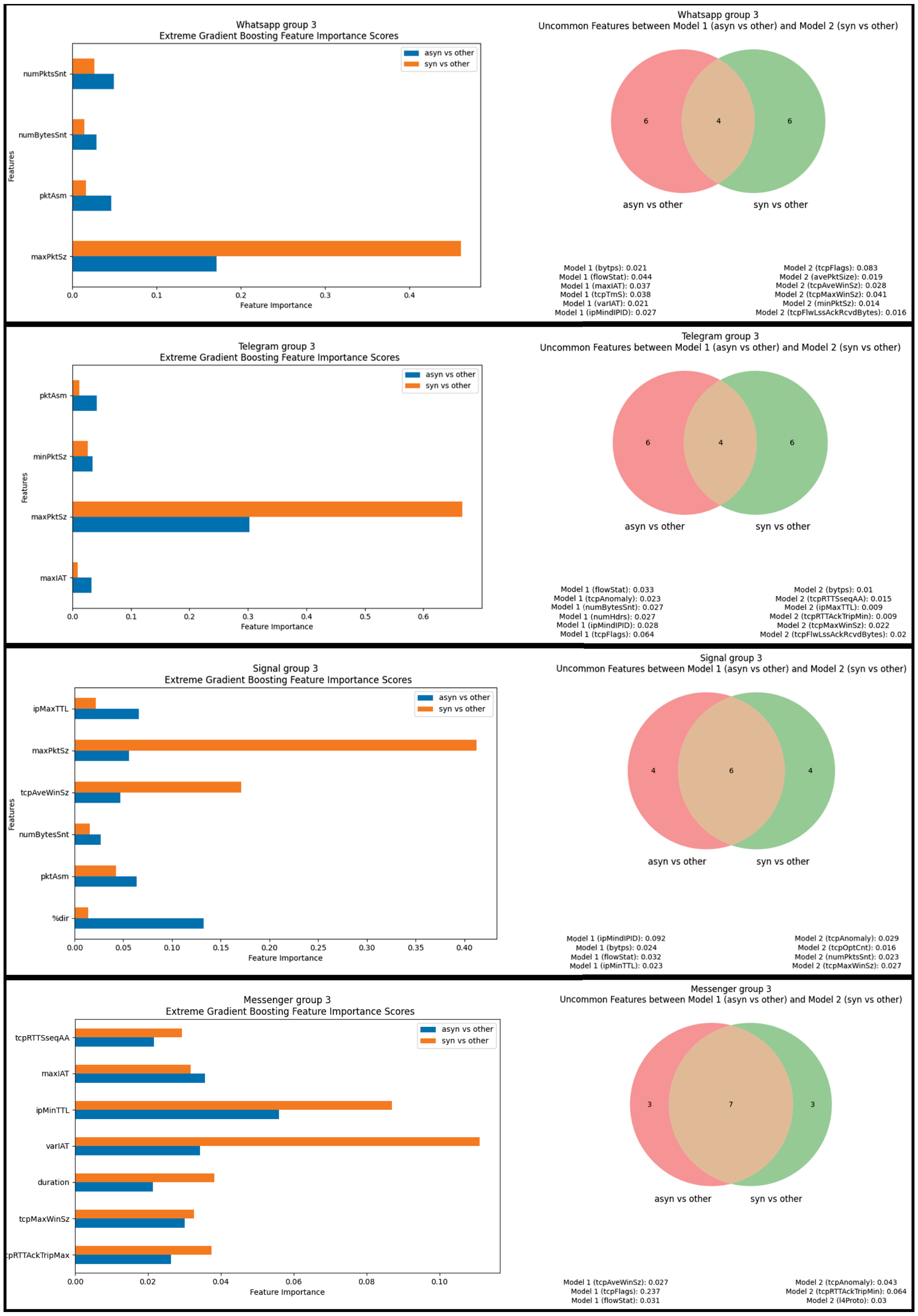

Figure C.2: Comparing Extreme Gradient Boosting Group 3 user feature importance for WhatsApp, telegram, signal, and Messenger

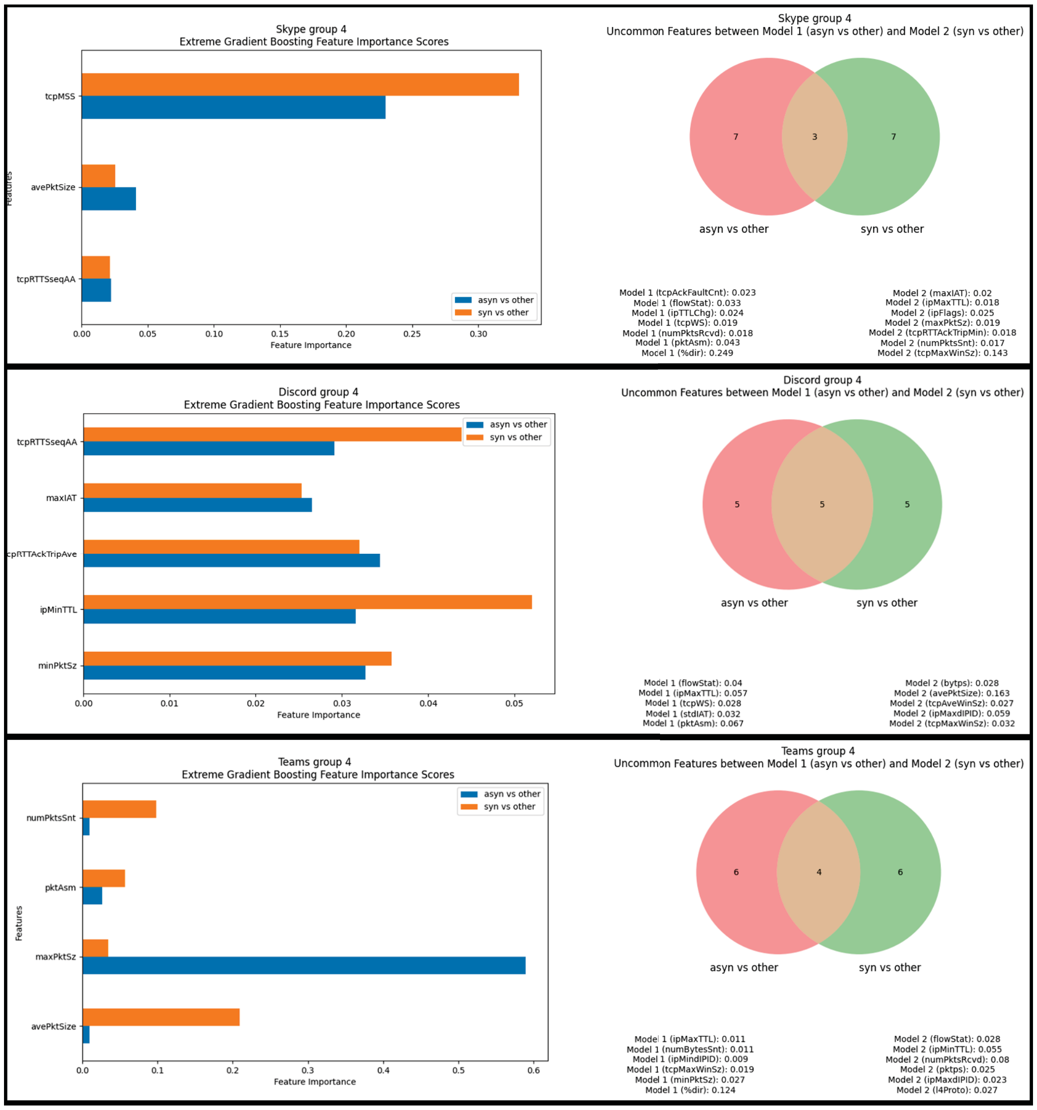

Figure C.3: Comparing Extreme Gradient Boosting Group 3 user feature importance for Skype, discord, and Teams

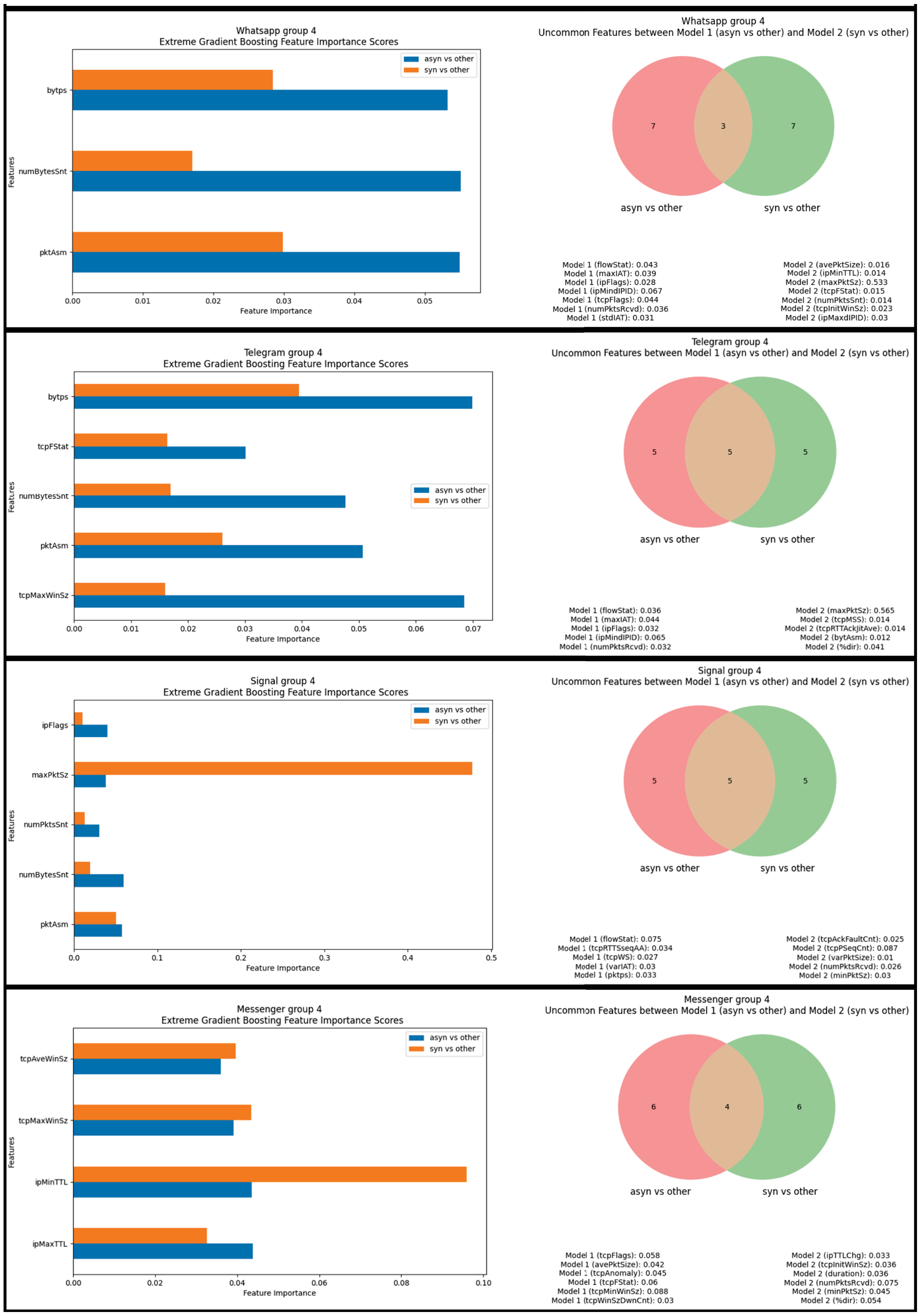

Figure C.4: Comparing Extreme Gradient Boosting Group 3 user feature importance for WhatsApp, telegram, signal, and Messenger

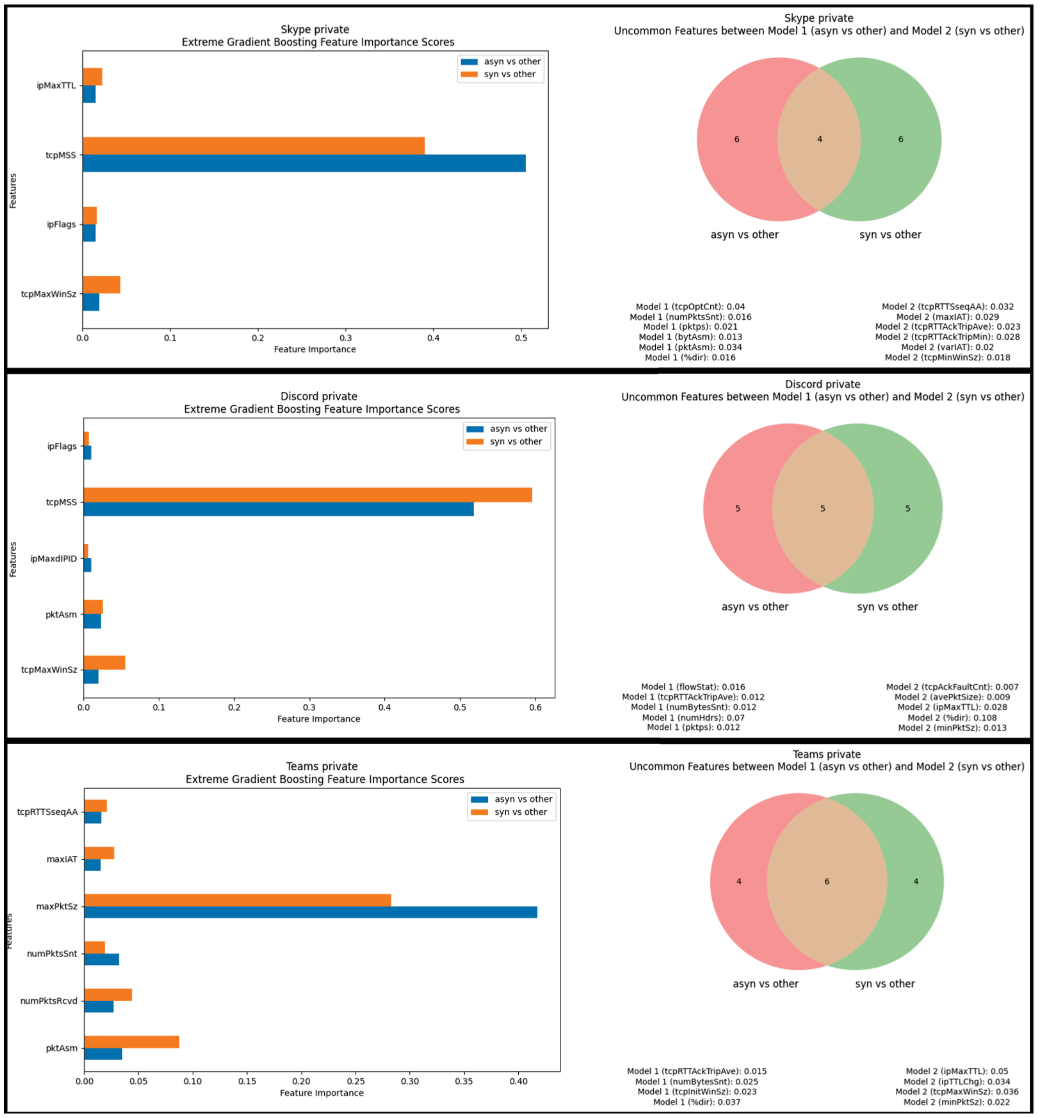

Figure C.5: Comparing Extreme Gradient Boosting Private user feature importance for Skype, discord, and Teams

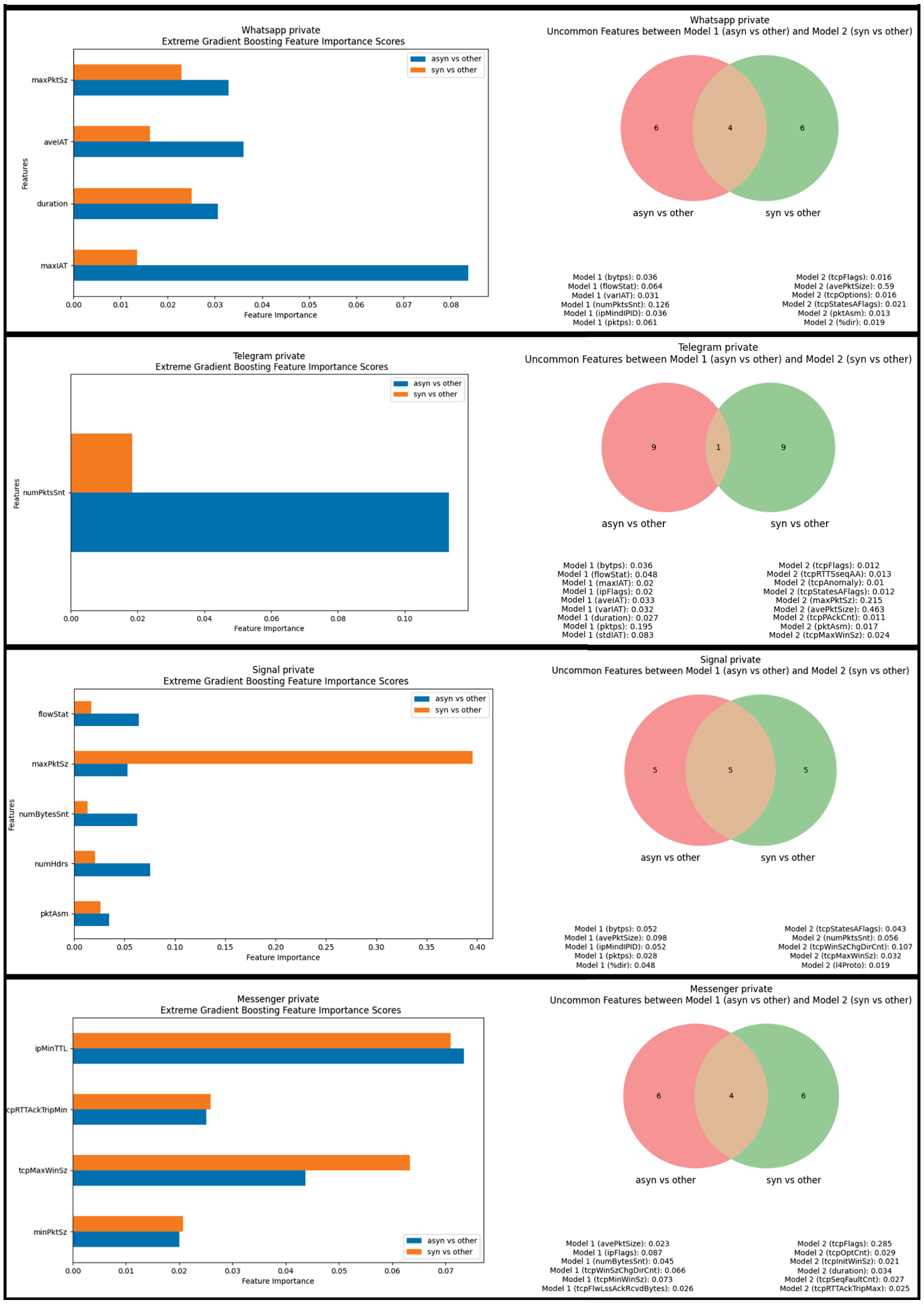

Figure C.6: Comparing Extreme Gradient Boosting Group 3 user feature importance for WhatsApp, telegram, signal, and Messenger