



AMERICAN UNIVERSITY OF BEIRUT

NETWORK TRAFFIC CLASSIFICATION AND NOVELTY  
DETECTION FOR MOBILE APPS

by  
GEORGI ABDULLAH AJAEIYA

A thesis  
submitted in partial fulfillment of the requirements  
for the degree of Master of Engineering  
to the Department of Electrical and Computer Engineering  
of the Faculty of Engineering and Architecture  
at the American University of Beirut

Beirut, Lebanon  
April 2017

AMERICAN UNIVERSITY OF BEIRUT

NETWORK TRAFFIC CLASSIFICATION AND NOVELTY  
DETECTION FOR MOBILE APPS

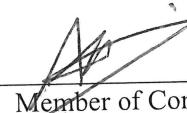
by  
GEORGI ABDULLAH AJAEIYA

Approved by:


Prof. Ali Chehab, Professor  
Electrical and Computer Engineering

  
Advisor

Prof. Ayman Kayssi, Professor  
Electrical and Computer Engineering

  
Member of Committee

Prof. Imad H. Elhajj, Associate Professor  
Electrical and Computer Engineering

  
Member of Committee

Date of thesis defense: April 11, 2017

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name:

Ajaeiya \_\_\_\_\_ Georgi \_\_\_\_\_ Abdullah \_\_\_\_\_  
Last First Middle

Master's Thesis       Master's Project       Doctoral Dissertation

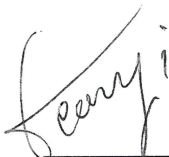
I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

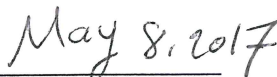
I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes  
after:

**One --- year from the date of submission of my thesis, dissertation, or project.**

**Two --- years from the date of submission of my thesis, dissertation, or project.**

**Three --- years from the date of submission of my thesis, dissertation, or project.**





Signature

Date

## ACKNOWLEDGMENTS

First, I would like to thank my advisors for their support and patience that they've shown during my study. I would like to thank Prof. Ali Chehab for giving me the opportunity to join AUB and be a part of the magnificent research group. A great thanks for Prof. Ayman Kayssi for his advice and experience which he provided to support my work. I would like to express my gratitude and acknowledgment for Prof. Imad Elhajj who helped me and provided key ideas for my work.

I would like to thank my family, fiancé Julie and my friends from Aleppo who inspired me and supported me during my masters in Beirut despite the circumstances which our beloved city and country are going through.

I would like to thank Accad Institute, manager and staff, who have a great favor for their endless support.

A great thanks to all my new friends in Lebanon for their compassion, support and lovely spirit which relieved me during the difficult times.

# AN ABSTRACT OF THE THESIS OF

Georgi Abdullah Ajaeiya for Master of Engineering  
Major: Electrical and Computer Engineering

Title: Network Traffic Classification and Novelty Detection for Mobile Apps

Network operators and mobile carriers are facing serious security and QoS challenges caused by an increasing number of services provided by smartphone apps. For example, Android OS has more than 2 million apps in stores. Hence, network administrators tend to adopt strict policies to secure their infrastructure. At the same time these policies have to ensure and maintain an excellent user experience. Our aim in this study is to propose an efficient classification and novelty detection mechanism for mobile apps' network-flows based on traffic analysis. The aim of this mechanism is to classify network flows based on a predefined set of classes which reflect user actions in each app. Such a mechanism can help network administrators to set QoS parameters according to traffic types over multiple network segments. Additionally, it helps in detecting new types of traffic that might be abnormal or malicious which can affect network performance. The mechanism differs from other proposed studies by focusing on identifying apps traffic from a network perspective without introducing additional clients on users' smartphones, or requiring special privileges. It involves a technique for pre-possessing network flows to acquire a set of feature vectors (samples) that are used to build a classification model using supervised machine learning algorithms. The study includes a parameter tuning phase, and a performance comparison phase to assess multiple machine learning models at their best parameter values. It is revealed that classification ensembles called Random Forests outperform other types of supervised classifiers such as Multi-Class SVMs. The classification model is used in an outlier detection process that employs Bayesian Inference. The process uses a confidence score metric produced by the classification model to detect novel samples. We reached a high detection accuracy for novel samples at 97% for benign apps and 92% for malicious apps with a low false alarms rate at 3%.

# CONTENTS

ACKNOWLEDGMENTS .....	v
AN ABSTRACT OF THE THESIS .....	vi
CONTENTS .....	vii
ILLUSTRATIONS .....	ix
TABLES .....	x
Chapter	
1. INTRODUCTION .....	1
2. LITERATURE REVIEW .....	6
2.1 COMPOSITE SOLUTIONS.....	6
2.2 NETWORK SIDE SOLUTIONS .....	8
3. PROBLEM DESCRIPTION.....	12
4. PROPOSED CLASSIFICATION AND NOVELTY DETECTION TECHNIQUE .....	16
4.1 CLASSIFICATION AND NOVELTY DETECTION PROCESS .....	17
4.1.1 Process Design Blocks and Parameters .....	21
4.2 PROPOSED APPROACH OF BUILDING THE CLASSIFICATION MODEL .....	24
4.2.1 Bayesian Optimization of Supervised Machine Learning Classifiers .....	26
4.2.2 Supervised Machine Learning Classifiers .....	27
4.2.3 Feature Selection.....	32
4.3 FLOW EXTRACTION AND FEATURE AGGREGATION APPROACH.....	36
5. EXPERIMENTAL SETUP AND DATA COLLECTION RESULTS.....	40
5.1 EXPERIMENTAL SETUP.....	40
5.2 APPS .....	43
5.3 USERS .....	44
5.4 DATA DESCRIPTION .....	45

5.4.1	Flow Extraction.....	45
5.4.2	Feature Extraction.....	49
6.	RESULTS AND ANALYSIS.....	55
6.1	COMPARATIVE ANALYSIS OF SUPERVISED MACHINE LEARNING CLASSIFIERS ....	55
6.1.1	Parameter Tuning and Evaluation Results.....	56
6.1.2	Feature Selection and Time Performance.....	58
6.2	CLASSIFICATION AND NOVELTY DETECTION PROCESS RESULTS.....	62
6.2.1	Detecting Novel Samples of Benign Classes.....	65
6.2.2	Detecting Novel Samples of Malicious Classes.....	67
7.	CONCLUSION.....	70
8.	BIBLIOGRAPHY.....	73



# ILLUSTRATIONS

Figure	Page
1.1 AN ILLUSTRATION OF THE PROPOSED METHODOLOGY.....	5
3.1 CLASSIFICATION MODEL BUILDING STEPS .....	13
4.1 NOVELTY DETECTION PROCESS FLOW CHART.....	16
4.2 HYPER-PARAMETER TUNING FOR SUPERVISED MACHINE LEARNING CLASSIFIERS .....	25
5.1 EXPERIMENTAL SETUP .....	40
5.2 ERD OF CLIENT MONITOR DB .....	42
5.3 FLOW EXTRACTION .....	45
5.4 FIRST DATA COLLECTION SET FLOW COUNT .....	46
5.5 FIRST DATA COLLECTION SET FLOW LENGTH BOXPLOT .....	47
5.6 SECOND DATA COLLECTION SET FLOW COUNT .....	47
5.7 SECOND DATA COLLECTION SET FLOW LENGTH.....	48
5.8 MALICIOUS FLOWS COUNT .....	49
5.9 MALICIOUS FLOWS LENGTH BOXPLOT .....	49
5.10 FEATURE VECTORS EXTRACTION AND AGGREGATION .....	50
5.11 ACCURACY OF THE INTERMEDIATE CLASSIFIER AT EACH SAMPLING RATE TIME INTERVAL.....	51
5.12 PACKET BURSTS INTER-ARRIVAL TIME SCATTER PLOT .....	52
5.13 CLASSIFIABLE SAMPLES IN FIRST DATA COLLECTION SET.....	53
5.14 CLASSIFIABLE SAMPLES IN SECOND DATA COLLECTION SET.....	53
6.1 F1 SCORE AND FPR OF ALL CLASSIFIERS.....	58
6.2 F1 SCORE VS. FEATURE VECTOR SIZE.....	60
6.3 TRAINING TIME VS. FEATURE VECTOR SIZE .....	61
6.4 ACCURATELY CLASSIFIED SAMPLES VOTE PDF.....	63
6.5 NOVELTY DETECTION RESULTS OF BENIGN CLASSES .....	67
6.6 NOVELTY DETECTION RESULTS OF MALICIOUS CLASSES .....	69

# TABLES

Table	Page
4.2 TUNABLE PARAMETERS OF BAGGED TREES .....	28
4.3 TUNABLE PARAMETERS OF RF .....	29
4.4 TUNABLE PARAMETERS OF MULTI-CLASS SVM .....	30
4.5 TUNABLE PARAMETERS OF KNN WITH RANDOM SUBSPACE.....	31
4.6 PACKET ATTRIBUTES .....	37
4.7 EXTRACTED FEATURE VECTOR.....	38
5.1 PACKET INFORMATION .....	41
5.2 CONNECTION INFORMATION .....	42
5.3 EXPERIMENT'S APPS.....	44
5.4 EXPERIMENT'S USERS AND SMARTPHONES .....	44
6.1 PARAMETERS' RANGES AND OPTIMIZED VALUES.....	57
6.2 EVALUATION RESULTS .....	58
6.3 FEATURES' RANKING .....	59
4.4 PROCESS PARAMETERS.....	63
6.4 PROCESS PARAMETERS VALUES (DETECTING NOVEL SAMPLES OF BENIGN CLASSES ) .	66
6.5 PROCESS PARAMETERS VALUES (DETECTING NOVEL SAMPLES OF MALICIOUS CLASSES ) .....	68

# CHAPTER 1

## INTRODUCTION

According to IDC [1], smartphones market worldwide share grew 13% over the second quarter of 2015 indicating a remarkable increase. Android OS dominated the market with 86.8% share and 7.2% growth over the four quarters of 2015. This indicates an increase in the number of smartphone users. In 2015, the percentage of users who accessed the web using their smartphones only was 52.7%. As a natural result for this increase, the number of smartphone apps and services is growing quickly to satisfy users' needs. According to Statista [2], as of June 2016, Google play store apps have reached 2.2 million, and Apple App store contained 2 million apps.

Serious challenges are facing today's computer and smartphone networks to cope with this vast number of apps and services. For example, maintaining a certain QoS level for some apps' functionalities (e.g. media streaming, video conferencing), and detecting abnormal network flows that may be consuming network's resources and affecting service quality for other users. These abnormal flows could be resulting from network attacks that aim to downgrade the availability of the network (e.g. Distributed Denial of Service DDoS using botnets [3]).

These challenges are affecting every public network that serves smartphones as a part of its infrastructure (e.g. mobile operators). Additionally, private networks have raised concerns after Bring your Own Device (BYOD) technology has emerged [4]. This technology endues users to bring their smartphones and tablets to work instead of using machines provided by the workplace. Users have full control over their devices and they

can easily consume workplace's network resources inappropriately. Therefore, most companies tend to use Mobile Device Management (MDM) systems [5] to gain partial control over the staff personal devices and protect the scarce resources of the workplace (e.g. bandwidth, servers). Access control solutions have been adopted in BYOD and implemented through MDM. These solutions act at the app level by placing restrictions on apps usage, or at the file level by using containers to limit the scope of company's data leakage to other apps on the same smartphone. Usually, MDM solutions require installing a light weight client on users' smartphones which raises privacy concerns. Additionally, MDM clients may not always be a lightweight service and may consume smartphone's resources (e.g. Battery, Memory, Computational power). Therefore, most users tend to avoid installing such clients on their smartphones. MDM solutions are ineffective in public networks, and they are suited better for private networks since they operate in a client-controlled environment.

Many solutions are proposed to protect networks' resources against inappropriate use and possible attacks. Most solutions use gathered information from smartphones or probes over the network to identify abnormal activities. Some solutions provide feedback to the users and warn them about the misbehaving apps. In general, many proposed works in the literature are based on a client-server architecture e.g. [6]. This architecture requires the installation of a client app on the smartphone just like MDM solutions. However, the client's role is to warn the user about their misuse or a misbehaving app without having any control over the device. On the other hand, the literature includes network-side solutions which concentrate on protecting network resources only. Such solutions include modeling apps behavior from processing and network perspectives [7], payload inspection by applying Deep Packet Inspection (DPI) [8], using machine learning to

classify apps behavior [9], and traffic analysis through statistically analyzing apps network behavior [10].

As mentioned, the majority of the proposed work employ gathered information from the smartphone itself, or from distributed probes over the network to build a knowledge base. This knowledge base is used to detect abnormal activities over the process or network level. Thus, we can categorize existing solutions based on the source of collected information.

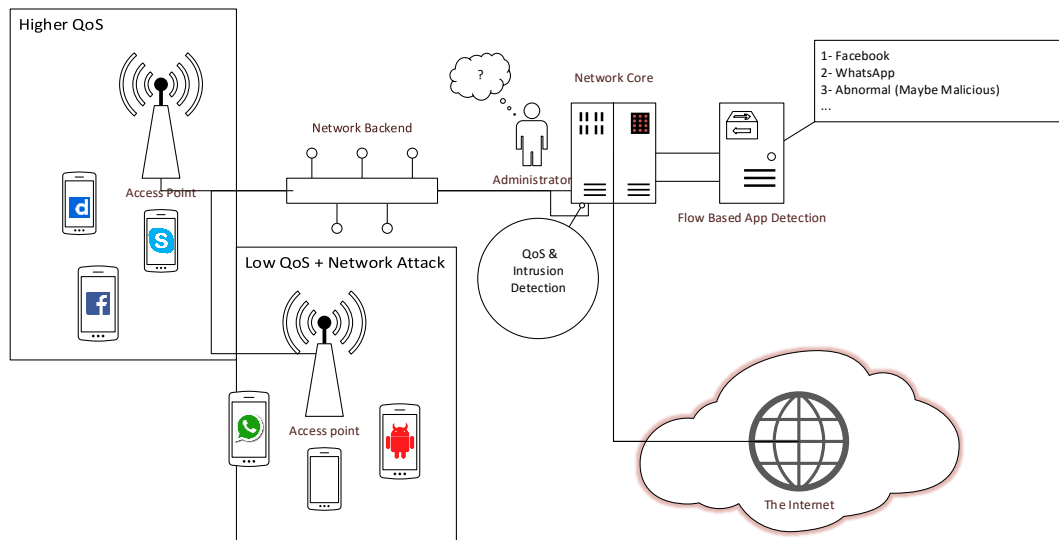
Data mining and pattern recognition techniques have dominated state of the art methodologies to build such knowledge bases. A knowledge base is used to detect and fingerprint apps' process and network behavior by comparing the detected behavior against it. The type of detected behavior relies on a set of measurements (Features) which are gathered from multiple sources and levels. Methodologies that concentrate on process behavior of apps use features that are linked to CPU and Memory consumption. At a deeper level, such methodologies inspect messages from inter-process communication that take place on the kernel level in case of Android OS. This requires installing client apps to collect such features, and sometimes these apps require root and super-user access [11]. Such methodologies have more disadvantages compared to their advantages. For example, installing clients that consume smartphones' resources due to their complexity is impractical. Additionally, the nature of the collected features makes them unable to detect apps' network behavior. Super-user privilege and root access may not be available on most smartphones as most users are not security experts. Therefore, most methodologies tend to collect multi-level features which are able to represent apps' behavior more generally [12]. Again, these require installing clients which act as data gathering and information collection agents. Due to the introduced complexity of using

multi-level features, data processing is moved to take place on a backend server. This introduced communication overhead between the distributed agents and the main server. In addition, it requires buffering the collected info which can be useless if not delivered at the time when an abnormal incident took place.

Finally, most recent state of the art methodologies [13][14] have been oriented towards network-side solutions. Such solutions focus on protecting network resources and pay less attention to users' smartphones. From a network operator, or a network administrator perspective, these solutions are more practical and introduce more advantages compared to the used solutions in the latter two types of approaches. Such solutions are located in the network core, and don't require installing a client or an agent on users' smartphones. This makes them secured against intruders, user misbehavior, and manipulation. They outperform client-server and client only solutions since they are completely deployed in the network core which theoretically has unlimited computational power and negligible communication overhead. The down time of such solutions is limited since the features are collected from the network which links the availability of the solution to the availability of the network.

The proposed methodology in this thesis follows a network-side solution to classify apps encrypted traffic. We focus on apps' network behavior since our aim is to protect network resources against improper usage and unwanted traffic types. Additionally, our aim is to utilize a classification technique to detect traffic types which helps in tuning QoS parameters to deliver best user experience. We focus on Android OS since it dominates the market share and has more users worldwide. Another reason for picking Android OS is the easiness of app development and familiarity with Linux operating system and shell commands which form the Android OS kernel. Figure 1.1

shows an illustration of how the proposed methodology could help in solving the described challenges.



**Figure 1.1** An illustration of the proposed methodology

The rest of the thesis is organized as follows. CHAPTER 2 describes the related work. In CHAPTER 3, we state the problem and the proposed technique to solve the problem. CHAPTER 4 describes the proposed technique for app traffic classification and novelty detection in details. CHAPTER 5 demonstrates the experimental setup and the data collection results. In CHAPTER 6, we show the results of classification and novelty detection and their analysis. Finally, conclusion and future work are covered in CHAPTER 7.

## CHAPTER 2

### LITERATURE REVIEW

Most proposed solutions in the literature which concentrate on app identification are related to security. As stated earlier we can classify previous work into the following categories.

#### **2.1 Composite solutions**

Some solutions are related to profiling and app fingerprinting such as [12], where the authors presented a multilayer system for profiling Android apps. The system includes four layers: Static, User interaction, OS, and Network layers. The main idea is taking advantage of cross-layer analysis to detect invisible abnormalities from a single layer perspective. However, root privileges are used in order to analyze apps' events and network behavior.

The authors of [8] proposed app fingerprinting based on DPI. As stated it helps network operators to expect traffic loads, quality of service, and discover network abnormalities. However, their approach supposes that 70% of apps don't use HTTPS which is not the case for current apps in stores. Additionally, applying deep packet analysis locally consumes smartphone's resources.

Other solutions are related to behavioral analysis. By concentrating on the analyzed behavior, we can split the proposed solutions to general behavior analysis and network behavior analysis.



Andromaly [11] and Crowddroid [15] are solutions based on general behavior analysis. Andromaly is a behavioral malware detection framework that consists of real-time monitoring, collection, pre-processing, and analysis of Android system metrics (Features). These features are extracted from both kernel and app levels using a client running on the device. They capture aspects such as network activities, resource consumption (e.g. memory and CPU), and event occurrence (e.g. touch screen and keyboard). Crowddroid uses a crowdsourcing dynamic analysis method to detect Android-platform malware. The system collects samples of execution traces for the running apps. These traces help in differentiating between benign and malicious apps. However, both solutions require rooted devices in order to collect some of the features.

Network Behavior analysis solutions concentrate more on features related to network activity. The authors of [16] presented a hybrid behavioral based anomaly detection system, which has a client-server architecture. The system is designed to protect mobile users and network infrastructure by detecting deviations in apps' network behavior. Models are generated to represent the normal network behavior of each installed app. The models are based on network related features, which are collected using a client running on the smartphone.

The authors of [17] proposed a system for network behavior detection of android malware, which consists of three parts: monitoring, anomaly analyzing, and cloud storage model. The system monitors apps' network behavior in real time, and does not need to parse the content of exchanged packets, which protects users' privacy. It depends on network behavior features only such as Bytes in, Bytes out, connection length, and connections log of the running processes. However, the results show low accuracy rates.

## 2.2 Network Side Solutions

These kinds of solutions are being adopted in the latest state-of-the-art work since they introduce more advantages than disadvantages over client-server solutions. Most network-side solutions employ traffic analysis since it is the only source of information on a network level. Latest works include profiling and modeling apps network traffic using machine learning algorithms and statistics.

The authors of [18] presented detailed analysis of Android smartphones traffic to show the effect on power consumption and throughput. They analyzed transfer sizes of TCP Flows and Round Trip Times, in addition to retransmission rates. The authors of [7] concentrated on modeling network traffic produced by users' behavior. They introduced a session concept that represents the needed flows to complete a single task. Multiple session characteristics are extracted and modeled using Hidden Markov Models (HMM). The authors introduced session types, which differ in packet lengths and traffic exchange duration. Each type of traffic, such as media streaming or browsing, defined a set of values for these variables. However, the values were based on assumptions and collected statistics. The models are used in profiling apps' patterns. However, models are not tested in a real environment.

The authors of [19] investigated background traffic generated by Android apps. They confirmed traffic characteristics' diversity by conducting a detailed experiment to analyze the traffic. The authors also studied Persistent TCP based apps, which require periodic message exchange in order to keep connections alive. However, the study was restricted to background traffic only. They suggested using DPI in order to identify running apps, which is not practical due to traffic encryption.

In order to identify apps, the authors of [9] suggested an adaptive algorithm that automatically recognizes traffic by relying on Classification Rules. They introduced a traffic classifier as a collection of rules that defines each type of traffic. The system architecture is composed of three sections: data collection, a flow capturing mechanism, and a classifier generation algorithm. The information generated from flow capturing and payload information of non-encrypted flows are used to generate the classifiers. However, their method had drawbacks because it depends on known destination IPs and ports of apps' servers. Such information is not determined in peer to peer communication. In addition, the method used non-encrypted flows to build classifiers, which is not an effective method in case of encrypted traffic.

The authors of [20] proposed traffic anomaly recognition using SVM classification algorithms. A detection model is built using collected features from the phone. Afterwards, they evaluated the detection model using real malware. The system used network features, and applied statistical classification in order to detect malicious apps. However, traffic features vectors are collected on the smartphone which can be done on network side to save resources.

The authors of [21] analyzed mobile traffic by comparing the URLs in apps' HTTP requests against a set of malicious domains. Requests are logged using Wireshark and Netstat, which requires deep packet inspection. This approach is limited because URLs are extracted from un-encrypted requests while most of apps' traffic is encrypted. The authors of [22] focused on analyzing encrypted traffic to build usage profiles to understand users' actions. The proposed framework analyzes TCP/IP packets, and extracts information about network flows. The authors used Dynamic Time Wrapping algorithm [23] to find alignments between incoming and outgoing packets, which turned

out to be unique for each app. Their approach handled encrypted traffic and identified users' actions with high accuracy; it did not involve a client installation on the device and it did not require any rooting privileges. However, it was affected by noisy packets such as TCP retransmission packets, and required reading TCP flags to set the start and end of the flow.

In another work [24], the same authors introduced an automatic app fingerprinting framework to identify apps behavior. The framework analyzes encrypted HTTPS/TLS flows collected from a client running on the phone. The authors applied various preprocessing techniques before extracting the features. The features expressed training samples used to train supervised machine learning classifiers. The authors introduced multiple training and testing scenarios and evaluated the performance of the used classifiers. However, their dataset is generated using specific conditions on a single phone.

The authors of [14] combined statistical-based and behavioral-based detection. Network traffic attributes were represented by Graphlets and packet sizes as distributions. They studied apps' background and foreground traffic separately. The authors achieved high accuracy in detecting apps' traffic patterns using a feature vector of 59 features. However, their experiment was limited were they studied traffic of a single device for a user, and they also used TCP flags to detect the flows which add more overhead.

In summary, the proposed solutions that employ traffic analysis are mostly related to network security. Only few have tried to classify normal traffic of multiple apps. The solutions which attempted to perform this kind of classification mostly involve installing a client on the smartphone, or require information that may expose users' privacy or may

not be available and increase the overall overhead of the proposed solution. Additionally, the majority of proposed works use limited datasets generated using a single testing device. This may limit the scope of the solution to a particular device or a user.

In this thesis, we propose a methodology to classify apps' traffic patterns using supervised machine learning algorithms and network collected features. The proposed technique doesn't require a client-app installation and ensures users' privacy. The overhead is moved to the network core since the computational resources are theoretically unlimited. Novel traffic patterns detection forms the second part of our contribution. We employ a specific type of machine learning classifiers called classification ensembles in an algorithm to detect novel flows using Bayesian Inference. The classification model and the detection process are validated using two datasets generated by multiple users using different smartphone models.

## CHAPTER 3

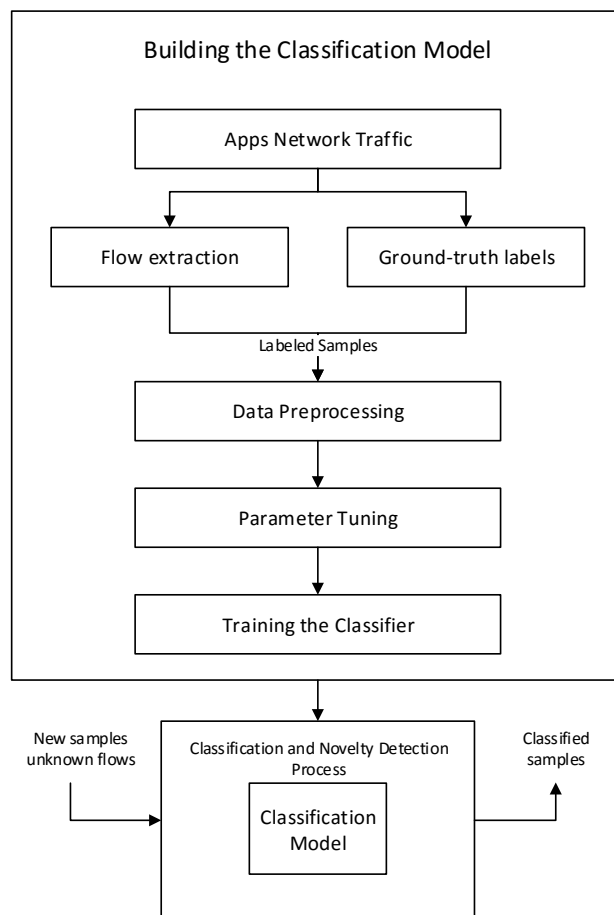
### PROBLEM DESCRIPTION

The aim of this study is to provide a technique for mobile apps traffic classification, and novel flows detection. The technique uses network-side features and doesn't introduce processing overhead to the smartphone. The technique is non-invasive for users' and transparent for intruders over the network. Since the proposed process operates and depends entirely on the network core, and doesn't use information from probes or client-apps that are can exposed to users it should be protected against manipulation and its security is tied to the core's security. Flows classification can take place on pseudo real-time basis as will be explained in CHAPTER 4.

There are two main components which the technique consists of. First, the classification model, which is represented by a supervised machine learning classifier. Building the classification model is described in CHAPTER 4. A comparative analysis is performed to asses multiple supervised machine learning classifiers. Assessment results are used to choose the best classification model which will form the main part of the novelty detection process. Second, novel flows detection process, which is based on Bayesian Inference. The process is designed to utilize Bayesian decision theory in detecting novel flows. A confidence metric is used in the process which is produced by the classification model. The confidence metric is assigned to each sample in the network flow and the patterns that has a relatively large number of samples with a low confidence are detected. Apps' flows are sampled during the lifetime of the flow and each sample is represented by a feature vector  $X$ . Forming the feature vector is an integral part of the

proposed methodology because it has a great effect on the performance of the classification model. Many proposed feature extraction attributes can be found in the literature [25] and they are implemented in this thesis.

The classification model is trained off-line before deployment. Since supervised machine learning models are used, the training phase should provide the ground-truth of the training data samples. We ensure the ground-truth labels for the training samples in the flow extraction phase which is a part of building the classification model.



**Figure 3.1 Classification model building steps**

As shown in Figure 3.1 training the classifier is the last phase of building the classification model. Important phases precede the training which are data preprocessing and feature extraction and aggregation. These phases enable the model to deliver accurate classification results depending on the nature of the extracted features.

Thereafter, the classification model can be used in a generalized classification process which is able to detect novel samples. The process algorithm employs Bayesian Inference to make decisions about the testing samples. Bayesian Inference helps in avoiding threshold based decisions which require continuous updates to preserve the detection performance.

In support of designing and implementing the process we have to answer the questions below.

- 1- how to design the classification and novelty detection process and what metrics are used to help in detecting novel flow samples?
- 2- What are the process parameters which minimize novelty detection error?
- 3- What type of supervised machine learning classifier should be used and how is it validated?
- 4- What features should be extracted in the feature extraction and aggregation phase?
- 5- Can we preserve same classification accuracy by reducing the feature vector dimension?
- 6- How to implement the flow extraction phase and what are the characteristics that represent a flow?

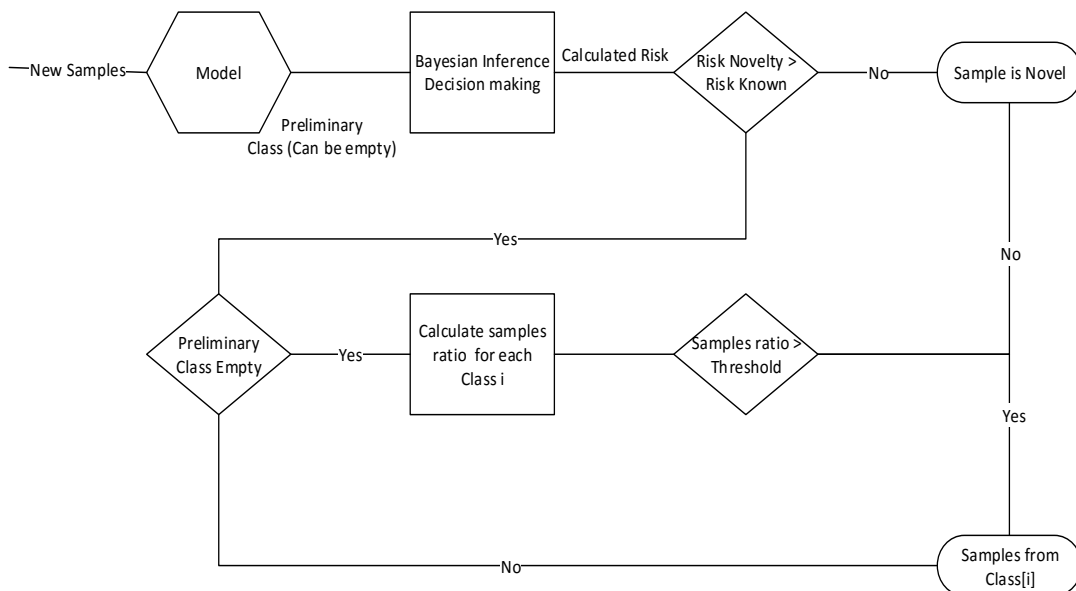


These questions are addressed through the thesis to build the methodology process of the proposed solution. Then, to validate the results of classification and novelty detection.

## CHAPTER 4

### PROPOSED CLASSIFICATION AND NOVELTY DETECTION TECHNIQUE

In this chapter, we address the problem described in CHAPTER 3 for the case of real traffic collected from real smartphones and regular users. First, we describe the novelty detection process algorithm and blocks. Then, in section 4.2 we describe our approach of building the classification model which used in the novelty detection process. Finally, in section 4.3 we describe the flow extraction and feature aggregation approaches which are used to extract classifiable samples to train and validate the classification model and the detection process.



**Figure 4.1 Novelty detection process flow chart**

#### 4.1 Classification and Novelty Detection Process

As shown in Figure 4.1, the classification model is used as part of the general classification and novelty detection process. The process consists of sequential sub-processes to make a final decision about the class of the new sample. The sample's class can be one of the known classes in the supervised classification model, or it can be novel. Therefore, the generalized process adds the capability of detecting novel samples that are unknown by the classification model. The Bayesian Inference and decision making block of the process is shown in Algorithm 4.1. Bayesian decision is based on a calculated conditional risk. The conditional risk is calculated as shown in Equation (4.1).

$$R(\alpha_i|x) = \sum_1^c \lambda(\alpha_i|w_i)p(w_j|x) \quad (4.1)$$

Where  $\alpha_i$  is the decision made by the Decision Rule to classify samples within a specific window in class  $w_i$  (e.g. known),  $\lambda(\alpha_i|w_i)$  is a coefficient in the loss matrix which is controlled by the user.  $p(w_j|x)$  is the posterior probability of the other class  $w_j$  (e.g. novel). The loss matrix for a Decision Rule of two classes have the structure shown in Equation (4.2).

$$\lambda(\alpha_i|w_i) = \lambda_{ij} ; \text{where } \lambda_{ij} \text{ in } \begin{pmatrix} \lambda_{kk} & \lambda_{uk} \\ \lambda_{ku} & \lambda_{uu} \end{pmatrix} \quad (4.2)$$

Where  $\lambda_{kk}$ , and  $\lambda_{uu}$  represent the loss of classifying known samples as known, and novel samples as novel respectively. Thus, these two values are set to zero usually because there is no penalty on making correct decisions.

Consequently, the decision rule for classifying incoming samples in our process would be an inequality between the conditional risk of classifying samples as known, and the conditional risk of classifying samples as novel as shown in Equation (4.3).

$$R(\alpha_k|x) >< R(\alpha_u|x) \quad (4.3)$$

Where

$$R(\alpha_k|x) = \lambda_{kk} p(w_k|x) + \lambda_{ku} p(w_u|x) \quad (4.4)$$

$$R(\alpha_u|x) = \lambda_{uk} p(w_k|x) + \lambda_{uu} p(w_u|x) \quad (4.5)$$

$R(\alpha_k|x)$  is the conditional of classifying samples as known, and  $\lambda_{ku}$  is the loss coefficient of classifying known samples as novel.  $R(\alpha_u|x)$  is conditional risk of classifying samples as novel, and  $\lambda_{uk}$  is the loss coefficient of classifying novel samples as known. Algorithm 4.1 describes the Bayesian decision concept in our approach.

The preliminary class score (confidence score)  $S_{k_{c_i}}$  is calculated based on the type of the classifier which used in the generalized process. The class score will be null if none of the known classes has a relatively high confidence score to classify the sample. If the score is not null it means that one of the known classes has a relatively high confidence score. When the class score is not null we increase the vote count for class  $V_{c_i}$ . Then, we split Bayesian decision risk calculation into two phases.

---

**ALGORITHM. BAYESIAN INFERENCE & DECISION MAKING**

---

```
while (new samples)
   $S_{k_{c_i}}$  = get preliminary class of sample  $k$  from the classification model;
  if ( $S_{k_{c_i}}$  is not null)
     $V_{c_i}++$ ;
    for  $l = x$  to  $k$  do | ( $x = 1$  if at the we are at the beginning of the flow else  $x = k - w$ ;  $w$  is the window size)
       $Risk_{uH} = V_{c_i} / \sum S * \lambda_h$ ;
       $Risk_{kH} = V_U / \sum S * (1 - \lambda_h)$ ;
    end

    for  $l = k$  to  $x$  do | ( $x = \text{end}$  if at the we are at the end of the flow else  $x = k + w$ ;  $w$  is the window size)
       $Risk_{uF} = V_{c_i} / \sum S * \lambda_f$ ;
       $Risk_{kF} = V_U / \sum S * (1 - \lambda_f)$ ;
    end

  else
     $V_U++$ ;
    for  $l = x$  to  $k$  do | ( $x = 1$  if at the we are at the beginning of the flow else  $x = k - w$ ;  $w$  is the window size)
       $Risk_{uH} = V_k / \sum S * \lambda_h$ ;
       $Risk_{kH} = V_U / \sum S * (1 - \lambda_h)$ ;
    end

    for  $l = k$  to  $x$  do | ( $x = \text{end}$  if at the we are at the end of the flow else  $x = k + w$ ;  $w$  is the window size)
       $Risk_{uF} = V_k / \sum S * \lambda_f$ ;
       $Risk_{kF} = V_U / \sum S * (1 - \lambda_f)$ ;
    end

  end
   $Risk_{novelty} = \alpha * Risk_{uH} + (1 - \alpha) * Risk_{uF}$ ;
   $Risk_{known} = \alpha * Risk_{kH} + (1 - \alpha) * Risk_{kF}$ ;
end
```

---

**Algorithm 4.1 Bayesian Inference and decision making**

First, calculate the risk of classifying the sample as known or unknown based on flow samples history (predecessors).  $Risk_{uH}$  and  $Risk_{kH}$  represent the risk of classifying the sample as novel or known respectively based on the predecessors.

Second, calculate the risk of classifying the sample as known or unknown based on the upcoming samples (successors).  $Risk_{uF}$  and  $Risk_{kF}$  represent the risk of classifying the sample as novel or known respectively based on the successors.

The samples that are taken into consideration whether as part of flow history or flow upcoming samples fall within a specific window. The moving window technique is used to include latest flow samples in the decision making process. Hence, when the process is making a decision about a specific sample it waits for the moving window to have all needed samples.

$V_U$  is the count of samples which are classified as novel, and  $\sum S$  is the total number of samples.  $\lambda_h$  and  $\lambda_f$  are weights of the loss matrix which are used in making weighted decisions. If the confidence score is null, the same process repeats with one difference:  $Risk_{kH}$  and  $Risk_{kF}$  will be calculated using the vote count of all known classes  $V_k$  instead of a specific class  $V_{c_i}$ .

Finally, the total risk for classifying the sample as novel  **$Risk_{novelty}$** , or as known  **$Risk_{known}$**  is calculated using a weighted sum of  $Risk_{u(H|F)}$  and  $Risk_{k(H|F)}$  using the parameter  $\alpha$ . The posterior probability for being known or being novel which is used in calculating the conditional risk is represented by the vote count ratio of each class. After calculating the conditional risks, we classify the sample according to the lowest risk. However, if the risk of classifying the sample as known was lower and the preliminary class was null; we calculate samples' count ratio for each known class, and if any class ratio exceeds a predefined penalty threshold, the sample is classified in that class.

#### 4.1.1 Process Design Blocks and Parameters

We revisit the novelty detection process flow chart shown in Figure 4.1. After deciding on which classification model will be used in the process, we have to define the method of assigning the preliminary class by the model. We suggest using Random Forest (RF) in the classification model component as it will be shown that it outperforms the other supervised classifiers in terms of the classification precision and sensitivity. Thus, RF will form the classification model part in the detection process.

RF is used in the literature for outlier and intrusion detection in computer networks [26]. The authors used a proximity score generated by the RF model to assign a confidence score for each network pattern. The proximity score shown in Equation (4.6) is the average fraction of trees in the RF model for which the instance lands on the same leaf versus other instances in the same class.

$$P^-(n) = \sum_{k \in \text{class } j} (\text{prox}^2(n, k)) \div N \quad (4.6)$$

Where  $k$  is a known instance from class  $j$ ,  $n$  is the testing instance, and  $N$  is the number of trees in the RF ensemble. In other words, the outlier score of an instance is the inverse of its proximity towards a specific class in the RF model (see Equation (4.7)).

$$O^-(n) = \frac{N}{P^-(n)} \quad (4.7)$$

Then, they compared the assigned scores for each network pattern to detect the outliers which represent network intrusions.

In another work [27], the authors proposed two approaches for novelty detection using RF. The first approach relies on the proximity score that is mentioned earlier.

However, the second approach relies on the voting mechanism that RF uses to classify samples. The authors use the votes given by each tree in the RF ensemble to create a confidence score for each class as shown in Equation (4.8).

$$F_i = \frac{v_i}{N} ; \sum_1^k F_i = 1 \quad (4.8)$$

Where  $F_i$  is the confidence score of the  $i$ th class in the sample,  $v_i$  is the votes sum for the  $i$ th class and  $k$  is the number of trees in the RF ensemble. Thus, the confidence of the RF classifier towards each class regarding a particular sample is represented by the vote distribution for each class. Grouping the confidence scores of each class in the same sample defines a vote vector as shown in Equation (4.9).

$$vote = [F_1 F_2 \dots F_k] \quad (4.9)$$

The confidence score is used to differentiate between two cases. First, the case of a sample coming from a class known by the RF ensemble. Here, the vote distribution should bias towards the known class  $F$  score in the vote vector. Second, the case of a sample coming from an unknown class. Here, the voting distribution should not be biased as in the case of the sample coming from a known class. As mentioned by [27], this approach provides a metric to measure the proximity between a sample and class relying on the characteristics of that class.

In our technique, we use the confidence score defined in Equation (4.8) to assign a preliminary class for the testing samples. First, the process defines a Probability Density Function (PDF) for each class voting distribution. Only correctly classified samples are included to form the voting PDF of each class. The distribution defines the region of vote



counts when the sample is correctly classified in that class. Then, for each new sample from the testing set, the sum of the votes  $v_i$  for each class  $c_i$  is calculated. Thereafter, the process calculates the normalized  $Z_{score}$  for each  $v_i$  as shown in Equation (4.10).

$$Z_{v_i} = \frac{v_i - \mu(PDF_{c_i})}{\sigma(PDF_{c_i})} \quad (4.10)$$

The process assumes that the voting distribution for each known class follows a Gaussian distribution with a mean  $\mu$  and standard deviation  $\sigma$ . The assumption is based on analyzing the vote counts for all classes' samples as will be shown in CHAPTER 6.

After calculating the  $Z_{score}$  for each  $v_i$ , the process calculates the  $P_{value}$  of the normalized score to produce a probability score for each class vote sum between 0 and 1. The produced probability is compared against a probability threshold which represents a decision making edge. The testing sample is assigned to a preliminary class if the probability score of that class crosses that edge. We note that only a single preliminary class can be assigned to each sample since the probability scores of classes in the same sample sum up to 1. The steps for assigning the preliminary class are summarized in Algorithm 4.2.

---

**ALGORITHM. ASSIGNING A PRELIMINARY CLASS**

---

**Initialization:** **foreach** class  $c_i$  calculate voting distribution  $PDF_{c_i}$ ;  
**while** ( $S$ )  
  **foreach** class  $c_i$   
     $Z_{v_i} = \frac{v_i - \mu(PDF_{c_i})}{\sigma(PDF_{c_i})}$ ;  
     $P_{v_i} = P_{value}(Z_{v_i})$ ; | one tailed Gaussian distribution  
  **end**  
   $P = [P_{v_1} \dots P_{v_k}]$ ;  
  **if any**(  $(P_v \text{ in } P) > P_{threshold}$  )  
     $S_c = Indexof(P_v)$ ;  
  **else**  
     $S_c = \text{null}$ ;  
  **end**  
**end**

---

**Algorithm 4.2 Preliminary class assigning**

Where  $S$  is a new sample,  $P_{v_i}$  is the  $P_{value}$  of the normalized Z score for the testing sample,  $P$  is the class probability vector, and  $S_c$  is the assigned preliminary class for that sample.

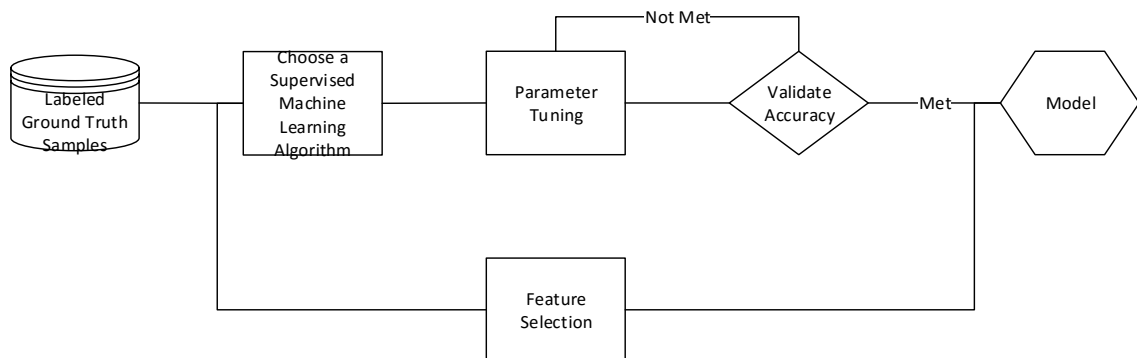
After assigning the preliminary class the process continues to assign the final class for the testing sample by calculating the Bayesian risk as shown in Algorithm 4.1.

#### **4.2 Proposed Approach of Building the Classification Model**

Classification is a form of supervised or unsupervised machine learning implemented using data mining techniques [28]. In supervised learning there are three datasets, one for training the second for evaluation and tuning, and the third is for testing. Every sample in both datasets is represented by a set of features that may be continuous, categorical, or binary [28] and have known labels or classes. Before building the classification model, there are three important phases that have to be implemented. These phases guarantee correct interpretation to achieve high detection accuracy. Data Pre-processing is the first phase. Training datasets may contain missing values that need to

be filled, or noisy points that need to be cleaned [28]. Therefore, extracted samples should go through a filtering phase. The second phase is feature selection, which is the process of removing redundant and useless features that may affect learning either by increasing the learning period, or decreasing the accuracy of classification. In most cases, classifiers use aggregated features from a set of basic features. The feature selection phase implementation depends on the type of the selection method e.g. Filtering or Wrappers. The final phase is choosing a suitable learning algorithm to meet the objectives of the study. Supervised classification has a large number of techniques that use data mining, e.g. Decision Trees [29], and some use statistical approaches, e.g. Bayesian classifiers [30].

As stated, we use supervised machine learning algorithms to build the classification model. In the literature, each proposed solution uses a specific model to classify patterns in apps' behavior. We decided to conduct a comparison among several models and pick the best in terms of classification accuracy and time performance



**Figure 4.2 Hyper-parameter tuning for supervised machine learning classifiers**

To perform the comparison, each selected classifier goes through a parameter tuning phase as shown in Figure 4.2. This phase assigns best values for each hyper-parameter. We use Bayesian optimization [31] for the parameter tuning phase.

#### 4.2.1 *Bayesian Optimization of Supervised Machine Learning Classifiers*

Seeks best hyper-parameter values that minimize the generalization performance function of a machine learning classifier. In general, training a machine learning classifier requires some high-level choices about the values of its hyper-parameters. Most supervised machine learning models have a set of hyper-parameters which have their values assigned during a parameter tuning phase. The aim is to use Bayesian optimization, which outperforms other state of the art optimization algorithms in this case [32], to determine the best values for the set of hyper-parameters of a specific model. In general, Bayesian optimization assumes that the generalization function is sampled from a Gaussian Process (GP) [33]. Therefore, it maintains a posterior distribution for the outcome of that function at each evaluation. The outcome of the generalization function in the case of a machine learning classifier is represented by the generalization error under different hyper-parameter values.

In order to pick the next value for a hyper-parameter, Bayesian optimization optimizes the Expected Improvement (EI) [31]. It is desirable to invest the computational power in making better choices about where to find best hyper-parameters instead of making expensive function evaluations e.g. training a classifier. The improvement is compared to the current best result of the generalization function.

Bayesian optimization constructs a probabilistic model for the generalization function  $f(x)$ . Then, it exploits the model to find values of  $x$  which is a subset of  $R^D$  to evaluate the function. It uses all available information from previous evaluations of  $f(x)$  and do not rely only on local gradient and Hessian. Thus, the procedure can minimize a non-convex function such as  $f(x)$  in a few iterations. This kind of optimization is

preferred in the case of expensive evaluation of the generalization function e.g. training a machine learning classifier.

While using Bayesian optimization one has to make choices about the Gaussian process prior and type of Acquisition Function (AF) [34]. The Gaussian process prior is used to make assumptions about the target function. While the AF is used in constructing a utility function for the model posterior which allows to determine the next evaluation point.

#### ***4.2.2 Supervised Machine Learning Classifiers***

In this study a comparative analysis is performed to compare four of the most known supervised machine learning classifiers.

1. Bagged Ensembles of Trees (Bagged)
2. Random Forest (RF)
3. Multi-Class SVM
4. KNN with Random Subspace

Each algorithm has a different set of parameters that has to be tuned for training. Each algorithm represents an ensemble of classification models of the same type. For example, Random Forest is formed of an ensemble of Decision Trees. Multi-Class SVM contains multiple binary classification SVMs. In this section, we briefly describe each algorithm and identify the different parameters used in the tuning phase.

##### ***4.2.2.1 Bagged Trees***

The Bagging Algorithm, introduced by Breiman [35], creates a single classifier from multiple weaker classifiers, which are Decision Trees. These classifiers generate their votes from multiple Bootstrap samples [30], by uniformly sampling  $n$  instances from a single training dataset and replacing them [36]. Afterwards,  $T$  Bootstrap samples are

used in building  $T$  classifiers in parallel using the C4.5 Algorithm [37]. Each Classifier,  $C_i$ , is learned by Bootstrap sample,  $B_i$ . The output of the final classifier  $C$  is the class that is most voted for by  $C_1, C_2, \dots, C_T$  as shown in Algorithm 4.3.

---

**ALGORITHM. THE BAGGING ALGORITHM**

---

**Input:** set  $S$ , Inducer  $I$ , integer  $T$  (number of bootstrap samples)

**Output:** Classifier  $C^*$

**for**  $i = 1$  to  $T$  **do**

$S' =$  bootstrap sample from  $S$  (i.i.d. sample with replacement);

$C_i = I(S')$ ;

**End**

$C^*(x) = \min_{y \in Y} \sum_{i: C_i(x)=y} \mathbf{1}$  (the most often predicted variable  $y$ );

---

**Algorithm 4.3 The Bagging Algorithm**

Each instance in the training set has a probability of  $1 - (1 - \frac{1}{n})^n$  to be selected in one of the  $n$  times instances picked from the single training set [38]. This technique decreases the error probability since we are including most instances in different combinations to build the classifier. In addition, it prevents creating an over fitted model of the system. The set of tunable parameters in this study are shown in Table 4.1.

**Table 4.1 Tunable Parameters of Bagged Trees**

Parameter	Description
N	No. of bootstrap samples to generate the classifiers
Learner type	Classification or Regression
Min. Leaf size	The minimum No. of observations to create a leaf

The learner type depends on whether the class is continuous or categorized in order to be considered as regression or classification, respectively. The leaf is the terminal node in the Decision Tree that assigns the class for an instance.

#### 4.2.2.2 Random Forest

Random Forest Classifier (RF) consists of a collection of Decision Tree classifiers, each built using an independent set of random training vectors  $\{\theta_k\}$ , which is identical to the rest of the sets. Each Tree classifier casts a vote for the most popular class for the input vector [39]. Therefore, the RF classifier uses a similar approach to the Bagged Trees classifier with one difference; the random vectors used in training each Tree in RF are identical in size, but each uses a different combination of features to structure the training set. For example, the Bagging Algorithm generates the random vectors in each training set by randomly selecting  $N$  instances from  $N$  sets where  $N$  is size of the training set. However, in RF, each random vector  $\theta_k$  is generated independently of the former vectors, but using the same distribution. The random split selection builds training vectors by permuting a specified number of features between 1 and  $K$ , where  $K$  is the maximum size of a training vector. The tunable parameters in this study are shown in Table 4.2.

**Table 4.2 Tunable parameters of RF**

<b>Parameter</b>	<b>Description</b>
N	No. of bootstrap samples to generate the classifiers
K	Number of Predictors
Learner type	Classification or Regression
Min. Leaf size	The minimum No. of observations to create a leaf

$k$  is fixed for all the training vectors in all the training sets, but each vector has a different random permutation of features in each training set.

#### 4.2.2.3 Multi-Class SVM

Support Vector Machines (SVM) was introduced by E. Osuna [40]. It builds a supervised classification model that can perform classification and regression analysis on linear and non-linear data. SVM models separate known classes by mapping their

instances on a hyperplane using a kernel, then it tries to minimize the generalization error between classes. SVM is a binary model that can separate between two classes at a time. Therefore, the employed implementation uses an Error-correcting Output Code Multiclass Model (ECOC) that reduces the problem of Multi-class Classification to a set of binary classifiers. The Coding Design in ECOC determines the classes that the binary learners should train on. For example choosing one versus one (OVO) coding design results in  $K(K-1)/2$  classifiers. The decoding scheme determines how the predictions of the multiple binary classifiers are interpreted. For example, in OVO coding a binary classifier prediction can be one of three possible choices. The sample is in Class 1, or in Class 2, or in none of the two classes. ECOC uses the loss  $g$  to determine the class that each new sample should be assigned as shown in Equation (4.11).

$$\hat{k} = \min_k \frac{\sum_{l=1}^L |m_{kl}| g(m_{kl}, s_l)}{\sum_{l=1}^L |m_{kl}|} \quad (4.11)$$

$m_{kl}$  is an element in the design matrix  $M$  representing class  $k$  and binary learner  $l$ .  $s_l$  is the predicted classification score for the positive class in learner  $l$ . Using this approach can improve the accuracy compared to other classification models. The tunable parameters in this study are shown in Table 4.3, depend on the coding and decoding scheme, and the type of the used kernel.

**Table 4.3 Tunable parameters of Multi-Class SVM**

<b>Parameter</b>	<b>Description</b>
Coding design	Type of coding used to separate classes, e.g. OVO, OVA.
Kernel function	Type of kernel used in the binary classifier e.g. Gaussian, Polynomial
Kernel scale	The learning rate parameter
Polynomial order	If the kernel is Polynomial, define polynomial order
Soft Margin	Allows misclassifications at the cost of a penalty factor $c$ .



#### 4.2.2.4 KNN with Random Subspace

The  $K$  Nearest Neighbor Algorithm was introduced by [41], and it is a non-parametric method used for classification and regression. The sample is classified according to the superior class dominating over the  $k$  neighbors of the sample in the feature space. KNN assigns weights to the contribution of each neighbor, where nearer neighbors contribute more in the classification decision. Training vectors are separated over a multidimensional feature space, each with a different class label.  $K$  and the distance metric are user defined constants.

**Table 4.4 Tunable parameters of KNN with random subspace**

<b>Parameter</b>	<b>Description</b>
K	Number of neighbors
Distance metric	The used method for measuring the distance between neighbors e.g. Euclidian, Hamming
Distance weight	A function applied on the distance metric e.g. inverse, squared inverse, equal.
Tie break	When two neighbors have the same weighted distance, KNN breaks the tie by locating: nearest same neighbor from other points or randomly picks the class.
M	Number of subspace dimensions
N	Number of learners in the ensemble

Typically, Euclidian distance is used for continuous features, while Hamming distance is used for discrete ones. Using KNN algorithm with Random Subspace selection [42] can improve the accuracy when an ensemble of  $n$  classifiers is created using  $n$  training sets. Each sample in the training set contains  $m$  random features selected from the original feature space that has a dimension  $d$ . Therefore, there are additional tunable parameters along with the number of neighbors  $k$ , which are shown in Table 4.4.

After performing the comparison, we compare the ability of each classifier to reduce the features vector size while partially preserving the same classification accuracy within an acceptable ratio. This phase is called feature selection, where it is used to increase the classification model performance time-wise, and reduce the noise in training data.

### **4.2.3 Feature Selection**

There are two approaches to perform this selection. The first approach is to score the subset of features using a classification model. In other words, choose subsets of features which increase the classification accuracy of the used model. This type of feature selection methods is called Wrapper methods [43][44][45][46].

The second approach is to analyze dataset properties and extract relationships among features. For example, calculating the correlation between two features in the feature space to eliminate one. This type of feature selection methods is called Filter methods. The first approach requires optimizing the classification model before using it in features ranking. On the other hand, Filter methods rely on the characteristics of the dataset to select features independently of the used classifier. In our approach, we implement both Wrapper and Filter methods to validate each classifier after completing the parameter tuning phase.

Feature selection aims to reduce the number of features in the feature space to increase the performance e.g. training time, and remove the noise in decision making caused by irrelevant features. However, reducing the feature vector size shouldn't have a great negative impact on the classification accuracy. As mentioned, we concentrate on two types of feature selection methods. First, filter methods which focus on the statistical

properties of the feature vectors and filter-out the ones which deliver less information e.g. Correlation Score. In our implementation, we use the correlation score analysis to rank the features that will be extracted later on. The computed rank is the correlation between the feature column and the change in the feature vector class with respect to all samples in the dataset. The correlation score is a real number between -1 and +1 that measures the degree of association between two data vectors. Whether it is a negative or a positive score, as long as it is closer to -1 or +1, the correlation is considered more significant.

Second, Wrapper methods which use a machine learning classifier to score the importance of each feature. We used four algorithms (Information Gain [44], OOB predictor Importance [47], RELIEFF [45], and Hold-out Support Vector Machines (HO-SVM) [43]) to calculate the importance and rank the extracted features later on. Each of the used algorithms is associated with a specific classifier. For example, OOB Importance and Information Gain are associated with DT classifiers which are used in the Bagging ensembles and RF. HO-SVM obviously uses SVM, while the RELIEFF algorithm uses KNN.

After ranking the features using both methods we re-evaluate the four classification models using the top  $n$  ranked features.

#### 4.2.3.1 OOB Predictor Importance

This method uses out-of-bag (OOB) estimation [47] in Trees ensembles to calculate two metrics, which express the importance of a feature.

1. Delta Error: which is the increase in the prediction error if the values of the specified feature are permuted across the OOB observations.
2. Delta Mean Margin: which is the decrease in the classification margin if the values of the specified feature are permuted across the OOB observations.

OOB observations are the omitted samples that form  $1/e$  or 37% of the total samples for each Decision Tree in the ensemble [46]. The samples are omitted when performing drawing with no replacement over  $N$  samples to form  $T$  bootstrap samples. The classification margin is the difference between the classification score of the positive class and the maximum score obtained for the false class in the same sample [48].

#### 4.2.3.2 RELIEFF

The algorithm [45] estimates features' strength according to the change in their values in nearby neighbors. The algorithm searches for a nearest neighbor from the same class, which is called a near hit, and  $k$  nearest neighbors for other classes, which are called near misses selected from  $n$  random training instances. Then, it calculates the difference between the features of the sample, the near hit, and the near misses. The difference for discrete features is either 0, which indicates equality, or 1, which indicates a difference. For continuous features, the difference is calculated using the Euclidian distance and normalized between  $[0, 1]$  to guarantee that all features' weights are within  $[-1, 1]$ . The algorithm averages the contribution of  $k$  samples to update the weight of the feature. A good feature has the same value for instances in the same class and should differ in instances from different classes.

#### 4.2.3.3 Information Gain

Information Gain [44] is based on the change of the entropy value for a feature after splitting the dataset using one of the features. The same method that DT uses when raking the importance of a feature to be selected for the next dataset split. There are two kinds of entropies, the first is calculated using the frequency table of a class attribute  $c$ , where the frequency  $p_i$  is the count of the distinct values of that attribute, as shown in Equation (4.12).

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i \quad (4.12)$$

The second entropy value is calculated using the frequency table of a feature against the class attribute  $c$ , as shown in Equation (4.13).

$$E(T, X) = \sum_{c \in X} P(c)E(c) \quad (4.13)$$

Finally, the gain of each feature is calculated using both entropies, and the feature with the largest gain is chosen to split the dataset based on its values, as shown in Equation (4.14).

$$Gain(T, X) = E(T) - E(T, X) \quad (4.14)$$

#### 4.2.3.4 HO-SVM

HO-SVM Algorithm is summarized as shown in Algorithm 4.4.

---

#### **ALGORITHM.** HO-SVM

---

(1) Model Selection

(2) Initialization

**while** smallest value of  $E_{(-p)}(\alpha, \sigma) < E(\alpha, \sigma)$  **do**

(a) Extract a random split of the dataset

(b) Train SVM using the specified parameters

(c) **foreach** feature  $p$  with  $\sigma_p = 1$ , **do** determine  $E_{(-p)}(\alpha, \sigma)$

(d) remove feature  $j$  with the smallest value of  $E_{(-p)}(\alpha, \sigma)$

**end**

---

#### **Algorithm 4.4 HO-SVM**

Where  $E_{(-p)}(\alpha, \sigma)$  is the number of classification errors when feature  $p$  is removed.  $E(\alpha, \sigma)$  is the number of errors in the validation set using all features as indicated by the current feature vector  $\sigma$ . The Model selection step is to determine the parameters of the SVM model e.g. kernel type. The initialization step indicates training

and validating the model using all features for the first time. We note that the previous method is used in the case of binary classification only. Thus, it is applied in each binary classifier in the multi-class SVM classifier.

SVM does not determine the importance of used features such as Bagged Trees and RF. The method starts with all available features in the dataset. Then, for each feature, the method determines the contribution in the SVM classifier. The feature that has the least impact on the classification accuracy will be removed at each iteration until a certain criterion is met.

In our technique, we validated the classification accuracy at the removal of each feature alone to rank the features independently of each other. We didn't set a stopping criteria because our aim is to rank all features.

In the following section we suggest our approach of flow and feature extraction and aggregation. The extracted features will be used to build the classification model and will go through the feature selection process as will be shown later.

### **4.3 Flow Extraction and Feature Aggregation Approach**

We followed two approaches while extracting the flows. In the first approach we considered all exchanged packets between two network IPs using the same ports and protocol to represent a flow and it is called the 5 tuples approach. In the second approach, we dropped the ports and considered all exchanged packets between two network IPs using the same protocol to represent a flow and it is called the 3 tuples approach. The flow is terminated when traffic exchange is idle for a specified amount of time and no more packets from this flow appear. Features are extracted from network flows generated

by a running app. The flows consist of TCP or UDP packets. Each flow packet is defined by a set of attributes as shown in Table 4.5. As we will see later, the proposed flow extraction approach does not require reading any information that may put users' privacy at risk. Some proposed methods in the literature require reading part of the payload. Others, read TCP flags to have inquiries about the flow state.

**Table 4.5 Packet Attributes**

Packet Number	Timestamp	Packet Length	Inter arrival Time	Direction (In/Out)	App Label
---------------	-----------	---------------	--------------------	--------------------	-----------

As shown in Table 4.5, timestamp is the time at which the packet has arrived. The packet length is the payload size in Bytes. The inter-arrival time is the time elapsed since the last packet was received in the flow. Packet direction states if the packet was outgoing or incoming relative to the smartphone network IP. Finally, the label represents the app which produced these packets by executing specific actions.

Feature extraction and aggregation is implemented to get useful information that describe the traffic generated by apps. The traffic is the set of extracted flows and packets exchanged while executing certain actions on the app. It is an important phase which has a great impact on the classification model accuracy and performance. We used some of the proposed feature groups listed in [25].

**Table 4.6** Extracted feature vector

#	Feature
1	Packets Out Count
2	Packets In Count
3	Packets Out / Packets In ratio
4	Bytes Out Count
5	Bytes In Count
6	Bytes Out / Bytes In ratio
7	Average difference of Inter-arrival time of incoming packets
8	Average difference of Lengths of incoming packets
9	Average difference of Inter-arrival time of outgoing packets
10	Average difference of Lengths of outgoing packets
11	Median of Inter-arrival time of incoming packets
12	Median of Lengths of incoming packets
13	Median of Inter-arrival time of outgoing packets
14	Median of Lengths of outgoing packets
15	Variance of difference of Inter-arrival time of incoming packets
16	Variance of difference of Lengths of incoming packets
17	Variance of difference of Inter-arrival time of outgoing packets
18	Variance of difference of Lengths of outgoing packets
19	Average of Inter-arrival time of incoming packets
20	Average of Length of incoming packets
21	Average of Inter-arrival time of outgoing packets
22	Average of Length of outgoing packets
23	Variance of Inter-arrival time of incoming packets
24	Variance of Lengths of incoming packets
25	Variance of Inter-arrival time of outgoing packets
26	Variance of Lengths of outgoing packets
27	Inter-arrival Time between packets bursts incoming
28	Inter-arrival Time between packets bursts outgoing

The features shown in Table 4.6 are extracted for each flow at a fixed time interval. Each flow may have more than one sample generated in this phase. An advantage of this technique is having multiple measurements for the same flow at formal intervals. These measurements can be used in real time detection of abnormalities in flow behavior, or in detecting novel samples.

There are important parameters to extract the flows which we can aggregate the features from. Flow idle timeout and sampling time interval are two important parameters



of our system. There are no standards that suggest values for these parameters. Therefore, they are set empirically in such a way to get best achievable results as we describe later.

In the following chapter we will show the outcomes of the flow extraction and feature aggregation phases. Classification model building through the comparative analysis approach, along with novelty detection results are described in CHAPTER 6.

## CHAPTER 5

### EXPERIMENTAL SETUP AND DATA COLLECTION RESULTS

In this chapter we describe the data collection experiment which is performed to form the experimental dataset. During the experiment we went through different stages from packet dump and network connections info collection, to flow and feature extraction and aggregation. Finally, the dataset is filtered to pull useful training and evaluation samples. The experimental dataset is used in our methodology implementation to build, design and evaluate the classification model and the generalized classification and novelty detection process.

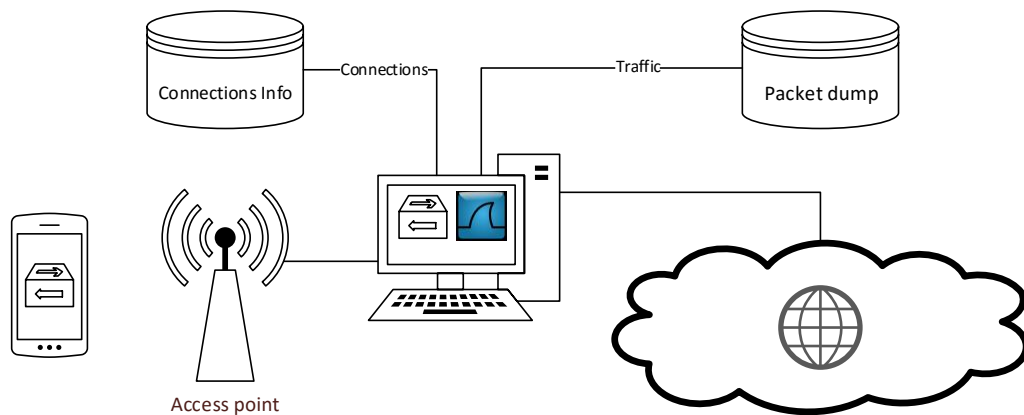


Figure 5.1 Experimental setup

#### 5.1 Experimental Setup

We can notice from Figure 5.1 that the experimental setup follows a client-server architecture. To collect packet dumps, we create a virtual access point attached to a terminal. The terminal is running Wireshark [49] to sniff and store packets in promiscuous mode. Packet dumps are stored as Pcap files that hold information about

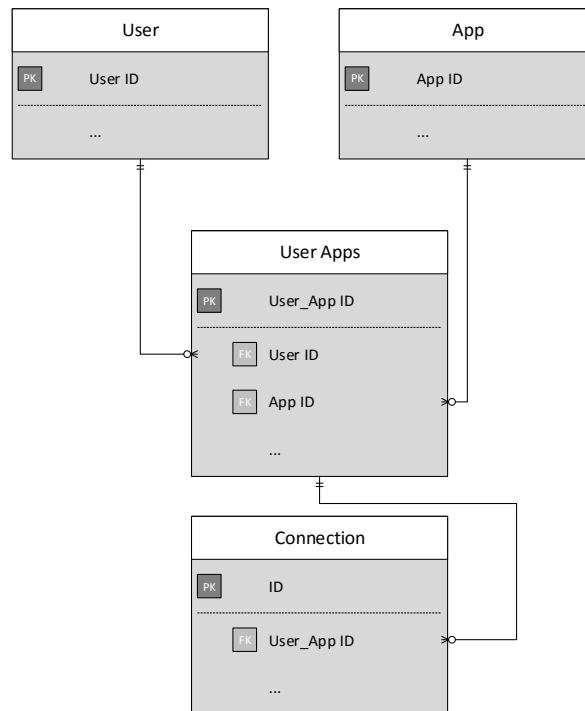
each sniffed packet as shown in Table 5.1. On the other hand, the smartphone runs a background service that stores network connections information established or received by every running app.

**Table 5.1 Packet information**

<b>Attribute</b>	<b>Definition</b>
No.	Packet sequential number as sniffed from the network.
Time	A Unix timestamp which indicates the packet sniffing time.
Source	Source IP address as stated in the packet header
Destination	Destination IP address as stated in the packet header
Protocol	Layer 3 protocol as stated in the packet header
Length	Packet (Frame) Payload size
Src. Port	Layer 3 Source port as stated in the packet header
Dst. Port	Layer 3 Destination port as stated in the packet header.
Delta Time	Time difference in milliseconds between current packet and last received packet from a specific source.
Time Sum	Cumulative addition of timestamps in sequential packets from a specific source.

The client-app extracts connections' info by frequently by running a simple "Netstat" command at fixed intervals. The service checks each established connection against a runtime list and stores new connections as entries. At each interval iteration, if the same connection still exists, the service updates its entry by updating the connection age attribute (see Table 5.2). The background service stores the information as temporary encrypted binary files on the external storage of the phone. Then, the service starts uploading the binary files to the server through a secure link over the internal network once these files exceed a specified file size. The secure link is established from the client to the server using a Public Key Infrastructure (PKI). For this setup we use a self-signed certificate which uses a 2048 RSA key generated by the server. The server side, which is implemented using PHP, receives the encrypted connections information as JSON arrays

and saves these arrays into a database structure. The Entity Relationship Diagram (ERD) of the database structure is shown in Figure 5.2.



**Figure 5.2 ERD of client monitor DB**

**Table 5.2 Connection Information**

<b>Attribute</b>	<b>Definition</b>
ID	Connection unique number
User App ID	A unique ID for the app which generated this connection which runs on a unique user smartphone.
Source IP	Layer 3 IP of the smartphone running the app which generated this connection
Destination IP	Layer 3 IP as shown when the service collected connection's info
Source Port	Layer 3 source port of this connection
Destination Port	Layer 3 destination port of this connection
Age	Connection's lifetime in seconds until the file was uploaded to the server

We can notice that the database structure can save all connections information for all apps running on different smartphones independently. Each inserted connection information is represented by several attributes shown in Table 5.2.

Connections information are used in filtering packet dumps and extracting traffic flows generated by each user app. We followed two approaches in flow extraction. In the first approach, we used the 5 tuples structure (Source IP, Destination IP, Source Port, Destination Port, Protocol) to group packets under the same flow. In the second approach, we used the 3 tuples structure (Source IP, Destination IP, Protocol). Therefore, by matching the specified tuple of a single connection in the database structure with the packet information saved in the packet dump file we can extract apps' flows. The User\_App ID is used to label each extracted flow by the unique ID of the app which runs on a unique smartphone.

We note that the client service is used to label the extracted flows. However, whenever the model is deployed along with the process, users do not have to install this client on their smartphones.

## **5.2 Apps**

A total of 14 Apps are used in the experiment as shown in Table 5.3. Each app is categorized according to the type of user actions executed while performing the experiment.

A set of user actions is specified for each app as shown in Table 5.3 to narrow down the classification task. The users were recommended to perform this set of actions only. We also used two advanced apps to generate malicious traffic. The first app is Nmap for Android [50] where it was used to perform network, service, and port scan attacks. The second app is Packet Generator [51], it was used to generate large number of dummy packets to perform a DoS attack on the server.

**Table 5.3 Experiment's apps**

#	App Name	Version	Category	Actions
1	Facebook	64.0.0	Interactive browsing	Browse, Comment, Like, Post
2	LinkedIn	4.0.62	Interactive browsing	Browse, Comment, Like, Post
3	8 ball pool	3.5.0	Game	Play
4	Clash Royal	1.8	Game	Play
5	Skype	6.22.0	Video Call	Video Call
6	Viber	5.8.0	VOIP	Voice Call
7	WhatsApp	2.12.45	Messaging	Texting, Media sharing
8	Instagram	10.3.2	Media Browsing	Browse, Like
8	Telegram	3.17.1	Messaging	Texting
9	Sound Cloud	2016.10.19	Music streaming	Listen to Music
10	Anghami	2.2.5	Music streaming	Listen to Music
11	YouTube	11.04.56	Video streaming	Play Videos
12	Daily Motion	9422	Video streaming	Play Videos
13	N map	N/A	Malicious	Service and Port Scan
14	Packet Generator	N/A	Malicious	DoS

### 5.3 Users

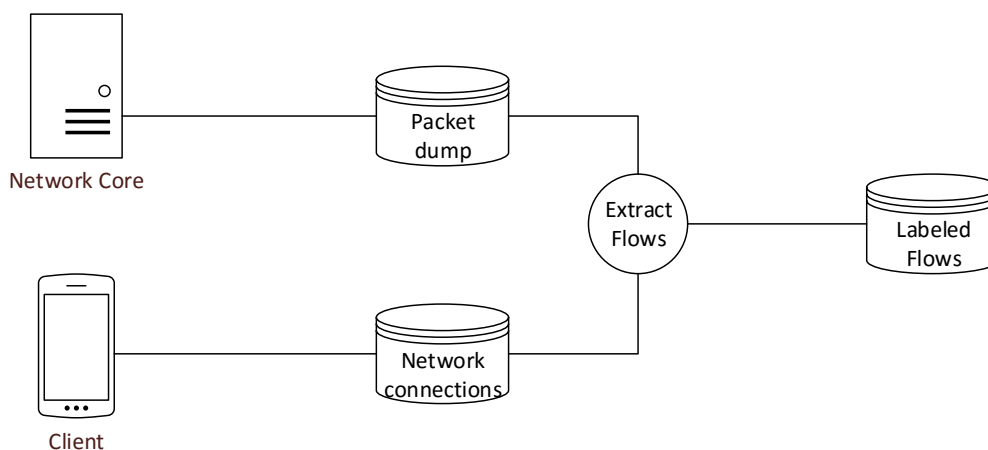
Two groups of users participated in the first round of the experiment. The first group contained 4 users, and the second group contained 5. In the second round, we repeated the experiment to collect more data with a single user. Table 5.4 shows a list of smartphones used in the experiment and the users' count associated with each smartphone.

**Table 5.4 Experiment's users and smartphones**

#	Smartphone	Model No.	Android OS Version	Users
1	Samsung Galaxy S3	GT-I9300	Lollipop 5.0	4
2	Samsung Galaxy Tab 4	SM-T230	Marshmallow 6.0.1	1
3	Sony Xperia M2	D2302	Lollipop 5.1	1
4	Sony Xperia Z3	D6633	Marshmallow 6.0.1	1
5	LG G3	D855	KitKat 4.4	1
6	HTC Desire 826	D826W	Lollipop 5.1	1

## 5.4 Data Description

In this section we describe the results of flow extraction and feature aggregation phases. Additionally, we show some statistics about the dataset size and extracted samples.



**Figure 5.3** Flow extraction

### 5.4.1 Flow Extraction

Figure 5.3 shows the proposed method for extracting the flows. We note that flows are extracted by filtering the packet dumps which are collected from the network core using connections' information collected from a client running on the smartphone. The client is used only to get labeled flows where each label represents an app. However, when the classification model is deployed in a real environment, the users do not have to install any client on their phones. The client is used to get the ground-truth of the extracted flows which is the type of the app or action that generated the flow. The proposed approach produces ground truth labels for the flows that results from specific actions executed within a specific app.

Two data collection sets resulted from two rounds of the experiment. The first data collection contains packet dumps and connection information from 6 different apps

(Facebook, 8 Ball Pool, Skype, Viber, WhatsApp, YouTube) produced by 9 users. The second data collection contains packet dumps and connection information from 9 different apps produced by a single user. We applied the 5 tuples flow preprocessing approach on the first data collection. On the other hand, we applied both flow preprocessing approaches on the second data collection. Figure 5.4 shows flow count statistics for the first data collection set per app and per user. Figure 5.5 shows the box plot for flow length in seconds for the same set. We can notice that Skype and WhatsApp have a relatively higher flow length compared to the other apps. We also note that some apps generated a larger number of flows due to its behavior in opening random network ports while being used.

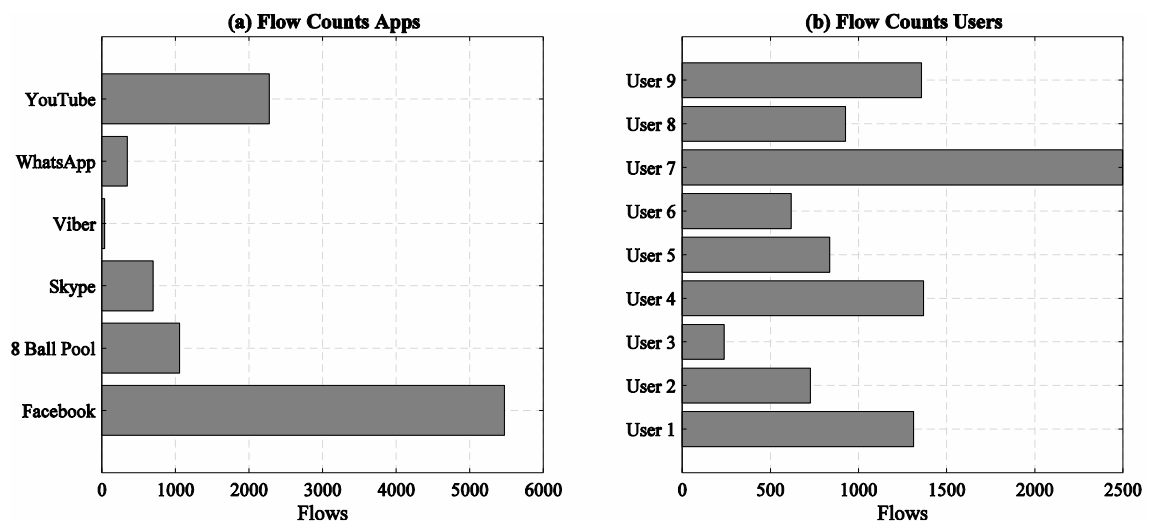


Figure 5.4 First data collection set flow count



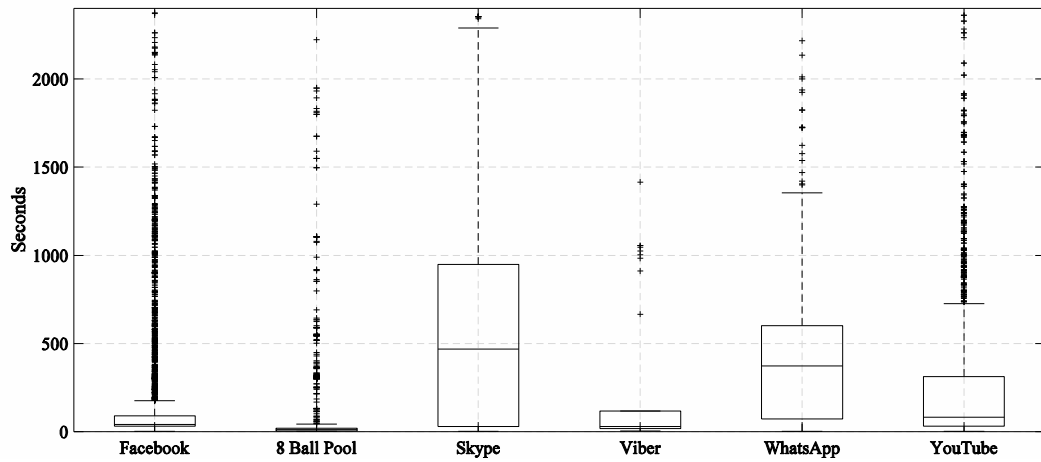


Figure 5.5 First data collection set flow length boxplot

Figure 5.6 shows flow count statistics for the second data collection using 5 tuples preprocessing approach and 3 tuples preprocessing approach. We can notice that the flow count drops significantly using the second preprocessing approach because many apps open multiple parallel connections to exchange data on multiple ports simultaneously. Therefore, when we treat all packets between two IPs as a single flow, we are aggregating all ports' traffic in that flow.

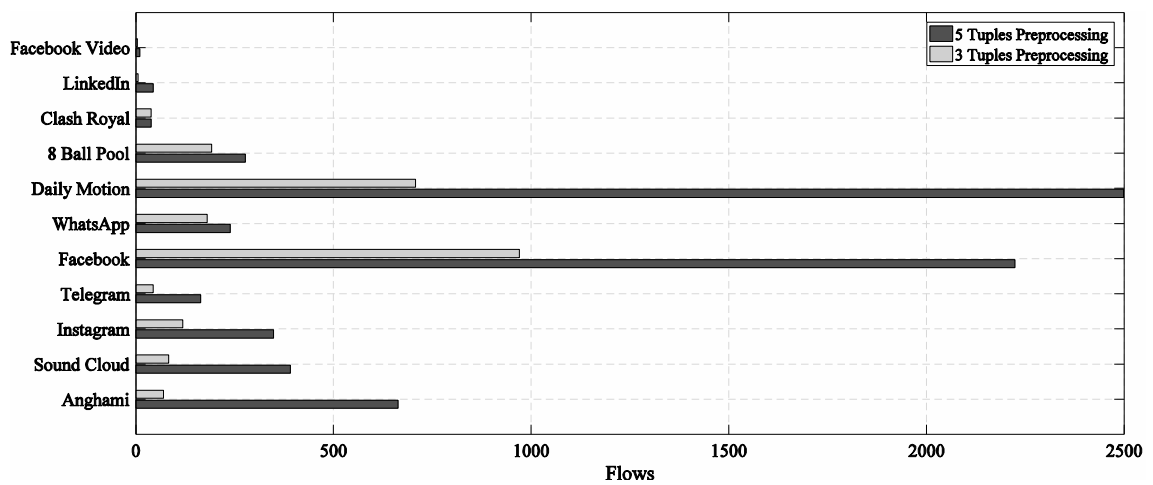


Figure 5.6 Second data collection set flow count

Figure 5.7 shows the box plot for flow length in seconds for the second data collection. By comparing flow lengths resulting from the 5 tuples preprocessing approach against flow lengths resulting from the 3 tuples preprocessing approach, we can notice that the flow length has increased significantly in the case of some apps when following the 3 tuples preprocessing approach. It is a result of aggregating all randomly opened ports for all app's connections as a single flow. This indicates that some apps change the randomly opened ports occasionally while the app is running.

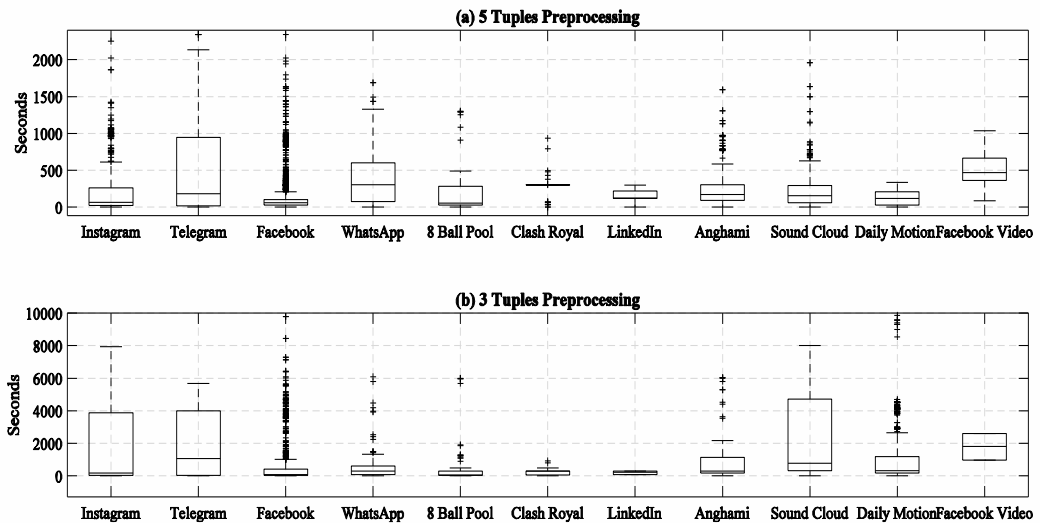


Figure 5.7 Second data collection set flow length

Additionally, we generated flows of malicious traffic using nMap and Packet Generator. Three different types of attacks are performed: Service Scan, Port Scan, and DoS. Figure 5.8 shows statistics on flow counts generated by these attacks which are extracted using the 3 tuples preprocessing approaches. We note that the 5 tuples preprocessing approach produced insignificant flows with less than two packets exchanged on the same flow due to the nature of these network attacks. Thus, we will rely on the 3 tuples preprocessing approach to extract attack feature vectors.

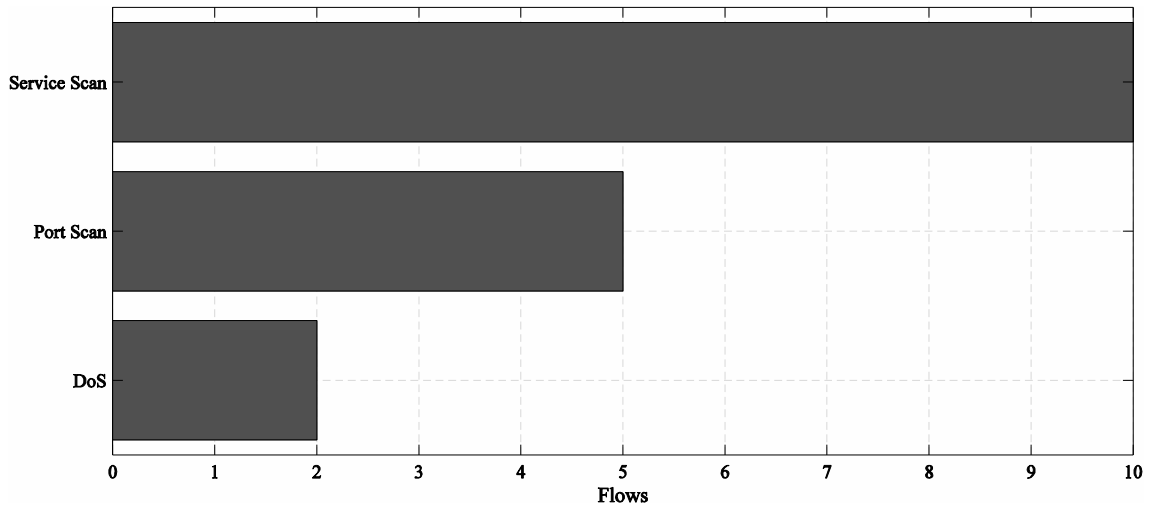


Figure 5.8 Malicious flows count

Figure 5.9 shows flow length statistics for the three types of attacks following the 3 tuples preprocessing approach. The extracted flows represent the network attack traffic exchanged between two IPs irrelevant of the port.

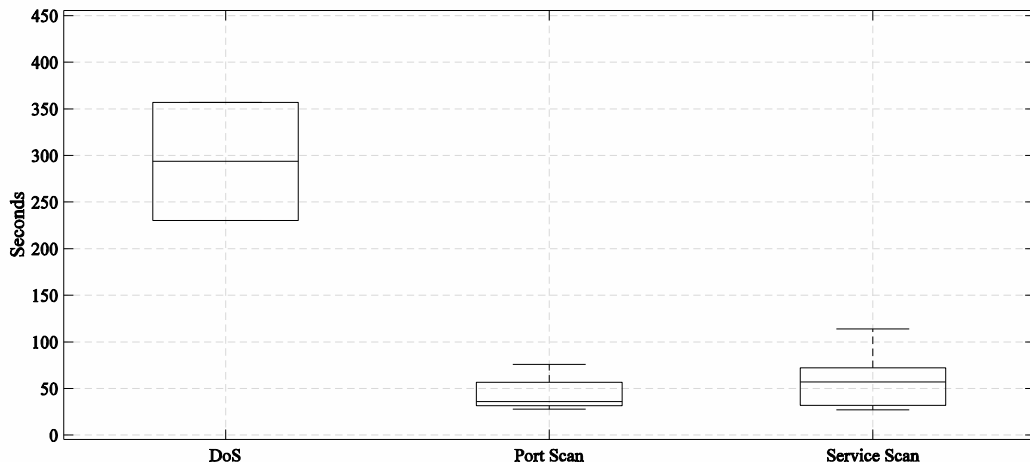
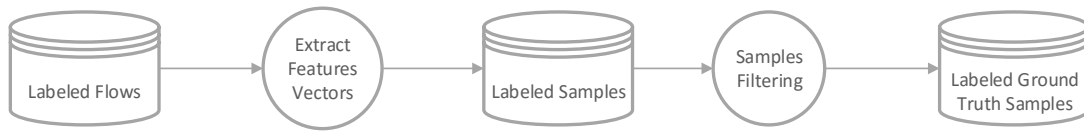


Figure 5.9 Malicious flows length boxplot

#### 5.4.2 Feature Extraction

As shown in Figure 5.10 the labeled flows are used to get labeled samples by feature extraction which are used to build the classification model.

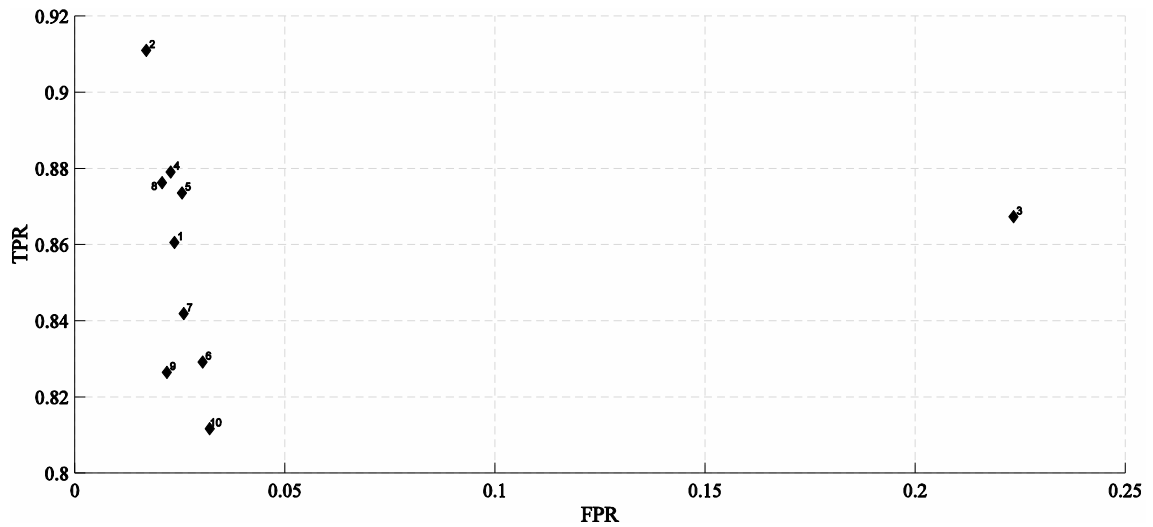


**Figure 5.10 Feature vectors extraction and aggregation**

Feature extraction phase requires setting two important parameters which are the flow idle time and the sampling interval. Since there is no criterion to set these parameters we followed a trial and error approach. The approach is to use an intermediate machine learning classifier which is Decision Tree (DT) to evaluate our choice of these two parameters. We chose this classifier because it is relatively fast to train and generates a classification model that is understandable by humans [29]. For each value for the idle time and the sampling rate ranging from 1 to 10 by increments of 1, we extracted samples defined by the feature vector shown in Table 4.6, and trained a DT classifier.

We used the flows resulting from the first data collection set to extract the samples. At each iteration we used the resulting samples at specific values for the idle time and the sampling rate to train the DT classifier. Then, we evaluated the accuracy of the classifier using 5 folds' cross validation expressed by TPR and FPR.

Figure 5.11 shows the ROC space of the intermediate DT classifier for each chosen set of parameters. We notice that a sampling rate of 2 seconds per sample returned the best evaluation accuracy for the intermediate classifier. Thus, we've chosen this value to extract the samples (feature vectors) from the first and second data collections using both pre-mentioned preprocessing approaches. The idle flow time parameter is set to 5 seconds where it has been shown that all TCP flow packets arrive at most within 4.43 seconds [52].

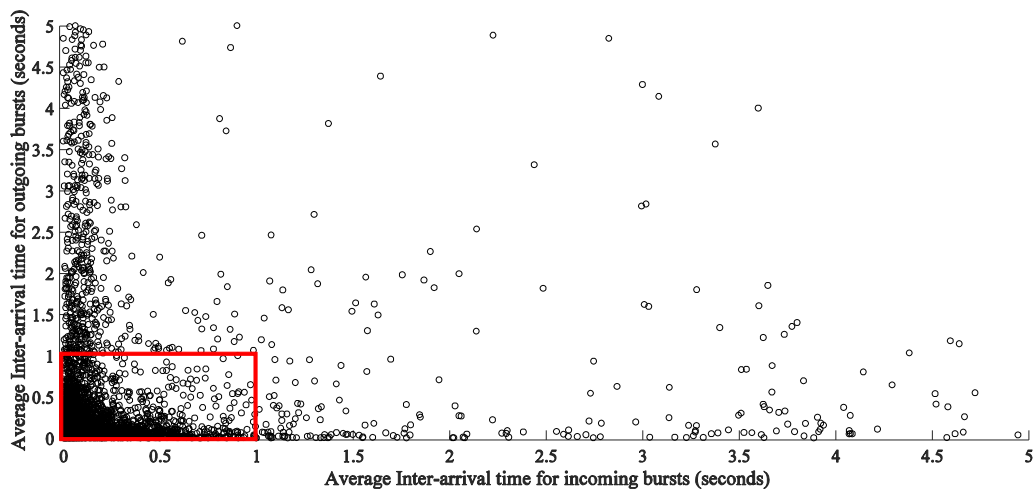


**Figure 5.11 Accuracy of the Intermediate classifier at each sampling rate time interval**

A set of rules is defined in a filtering phase to filter the samples (feature vectors) from the first and second data collection. This phase ensures that only statistically significant samples are taken into consideration to be used in building the classification model and later on, in the classification phase. Such samples represent traffic bursts that happen within the lifetime of the flow. A traffic burst is a relatively fast exchange of packets that occur when the user executes a specific action. Therefore, significant samples have a relatively low inter-arrival time, and a minimal count of exchanged packets. This phase ensures that we hold out unclassifiable samples which represent signaling and keep-alive packets. The set of rules are defined as following.

1. Each sample with zero incoming packets and less than 2 outgoing packet is discarded.
2. Each sample with zero outgoing packets and less than 2 incoming packet is discarded.
3. Each sample that have an average inter-arrival of more than 1 second for incoming and outgoing traffic bursts is discarded.

The limit on the inter-arrival time of packets is set by examining the samples which indicate action execution against samples that represent idle traffic. The aim of this filtering is to discard signaling packets and keep alive traffic which add noise to the dataset. Figure 5.12 shows plotting the inter-arrival time of incoming packet bursts against the inter-arrival time of outgoing packet bursts. We can notice that samples outside the 1 second box are less dense which indicates a lower packet burst rate. Therefore, we use this criterion to discard noisy samples that represent signaling data.



**Figure 5.12 Packet bursts inter-arrival time scatter plot**

Figure 5.13 shows the classifiable samples vs. total samples count before and after filtering for each app in the first data collection set. The total number of samples before filtering is 59385 which is reduced to 49942 after filtering.

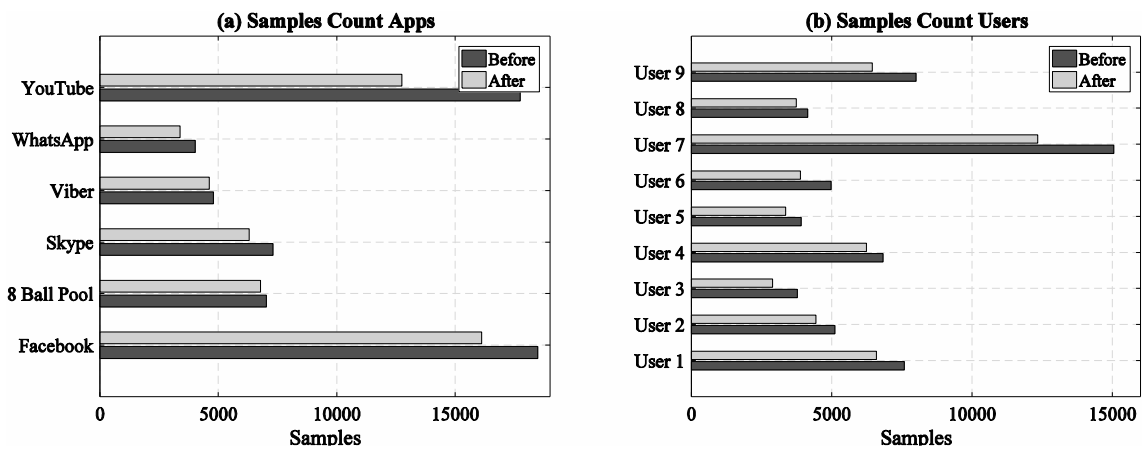


Figure 5.13 Classifiable samples in first data collection set

Figure 5.14 shows classifiable samples vs. total samples count in the second data collection set before and after filtering following the 5 tuples preprocessing approach, and the 3 tuples preprocessing approach. 14769 samples resulted from the 5 tuples preprocessing approach which is reduced to 12827 after filtering. The 3 tuples preprocessing approach produced 17893 samples which are reduced to 10989 after filtering.

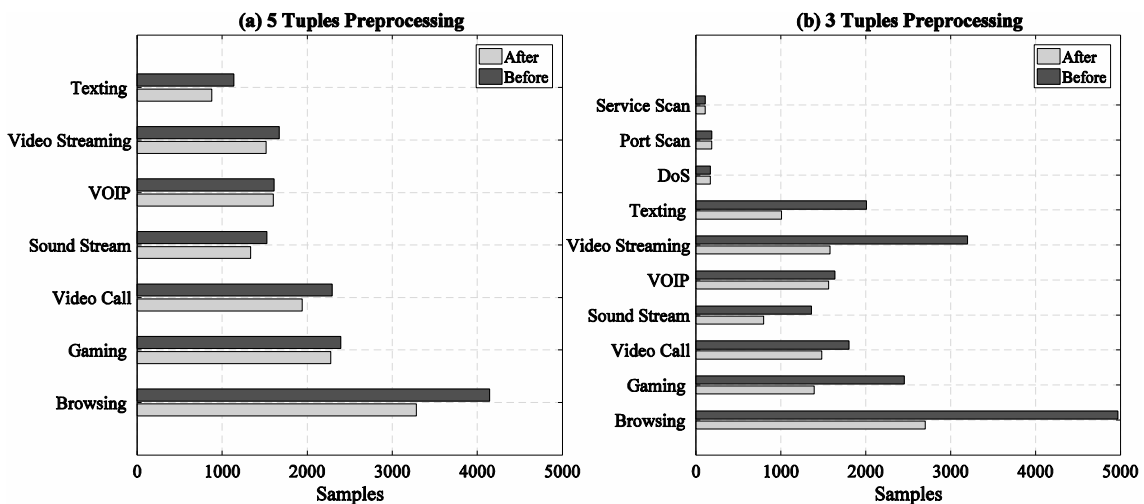


Figure 5.14 Classifiable samples in second data collection set

We note that in the second data collection set we grouped apps' samples based on their categories. The Browsing category contains Facebook, LinkedIn, and Instagram samples. The Gaming category contains 8 Ball Pool, and Clash Royal samples. The Video Call category contains Skype video call samples. The VOIP category contains Viber voice call samples. We grouped Daily Motion and Facebook video samples in the Video Stream category. Sound Stream category includes Sound Cloud and Anghami samples. Finally, the Texting category contains WhatsApp and Telegram samples.

In the 3 tuples preprocessing approach we notice a drop in the classifiable samples ratio for most apps. It varies according to the type of actions that can be executed in parallel and how the app handles parallel connections with multiple IPs for a single session. Actions such as voice calling and video calling nearly had the same ratio of classifiable samples in both preprocessing approaches. This is a result of sniffing packets for these actions while the calls were connected where packet bursts exchange is at its peak while transferring voice and video in real time. This means that there was no dragging in the packet exchange rate. The other apps have infrequent packet bursts from multiple parallel connections on different ports at the same time that result from executing a specific action. This makes the significant samples which indicate an action more significant by aggregating the exchanged traffic from all ports in these samples. Additionally, it produces more insignificant samples which indicates a lower traffic exchange rate while being idle.



## CHAPTER 6

### RESULTS AND ANALYSIS

This chapter demonstrates the results of our proposed method. Section 6.1 shows the results of the parameter tuning, feature selection, and time performance for the four supervised classification models. Section 6.2 shows the results of the classification and novelty detection process.

#### 6.1 Comparative Analysis of Supervised Machine Learning Classifiers

Bayesian optimization is used to select the values of hyper-parameters for each classifier. This type of optimization tends to minimize the generalization function  $f(x)$  which in our case is set to be  $1 - F1_{score}$  (see Equation (6.4)) of the evaluated classifier. Additionally, when using Bayesian optimization, we have to make a choice about the GP prior and the type of the Acquisition Function (AF). The implementation in Matlab called “bayesopt” [53] assumes that the GP prior has an added Gaussian noise and the prior distribution is assumed to be also a GP with a mean  $\mu(x, \theta)$  and a covariance kernel function  $k(x, x', \theta)$ , where  $\theta$  is a vector of kernel parameters. The kernel type is set to be “ARD Matern 5/2 kernel” shown in Equation (6.1).

$$k(x_i, x_j) = \sigma_f^2 \left( 1 + \frac{\sqrt{5}r}{\sigma_l} + \frac{5r^2}{3\sigma_l^2} \right) \exp\left(-\frac{\sqrt{5}r}{\sigma_l}\right) \quad (6.1)$$

Where

$$r = \sqrt{(x_i - x_j)^T (x_i - x_j)} \quad (6.2)$$

$X = x_i$  and  $X = x_j$  are the current and next set of points associated with the current and next value of the objective function  $F = f_i$  and  $F = f_j$  respectively. As for the type of the AF it is set to be “Expected Improvement” shown in Equation(6.3).

$$EI(x, Q) = E_Q[\max(0, \mu_Q(x_{best}) - f(x))] \quad (6.3)$$

Where  $x_{best}$  is the location of the lowest posterior mean and  $\mu_Q(x_{best})$  is the lowest value of the posterior mean.

For evaluation, we use the samples that resulted from the first data collection set that includes samples for 6 different apps and 9 different users. Forming the dataset is divided into three steps using the hold out technique. First 10% of the data produced by a user  $x$  is held out, then 65% of samples from the remaining users is grouped to form the training dataset which is used on each classifier iteratively with a different set of parameters. The parameters are picked by the Bayesian optimization process. Thereafter, the remaining 35% of samples is used to evaluate the trained classifier accuracy. Then, we use the 10% held out earlier as a testing set, which is not introduced in the training and evaluation phases to perform the final evaluation. The set of parameters that return best performance in terms of evaluation are used in the comparison.

### **6.1.1 Parameter Tuning and Evaluation Results**

The Bayesian optimization process repeats for 30 consecutive times, each time a different set of parameter values is picked according to the GP prior and AF. In order to enclose parameter values, we set lower and upper limits for each parameter of type numeric. As for the categorical parameters, we define a set of values that the Bayesian optimization process can pick from. The bounds were set based on some knowledge in

the field to avoid overfitting. At the end of the optimization process we get a set of parameter values referred as “X at min objective function”. Table 6.1 shows parameter values for each classifier at the located min objective value. Setting the bounds for each numerical parameter helps in preventing the classifier from overfitting the training data.

Each classifier is evaluated using the parameter values in Table 6.1. The evaluation is repeated on each 10% of data which was held out from each user. Then, the evaluation F1 score (see Equation (6.4)) is averaged over these folds for each app.

$$F1 = \frac{2 TP}{2 TP + FN + FP} \quad (6.4)$$

Where  $TP$  is the True positive,  $FP$  is the False Positive, and  $FN$  is the False Negative.  $F1_{score}$  measure which combines precision and recall of a classifier is referred as the classification accuracy or evaluation accuracy.

**Table 6.1 Parameters’ ranges and optimized values**

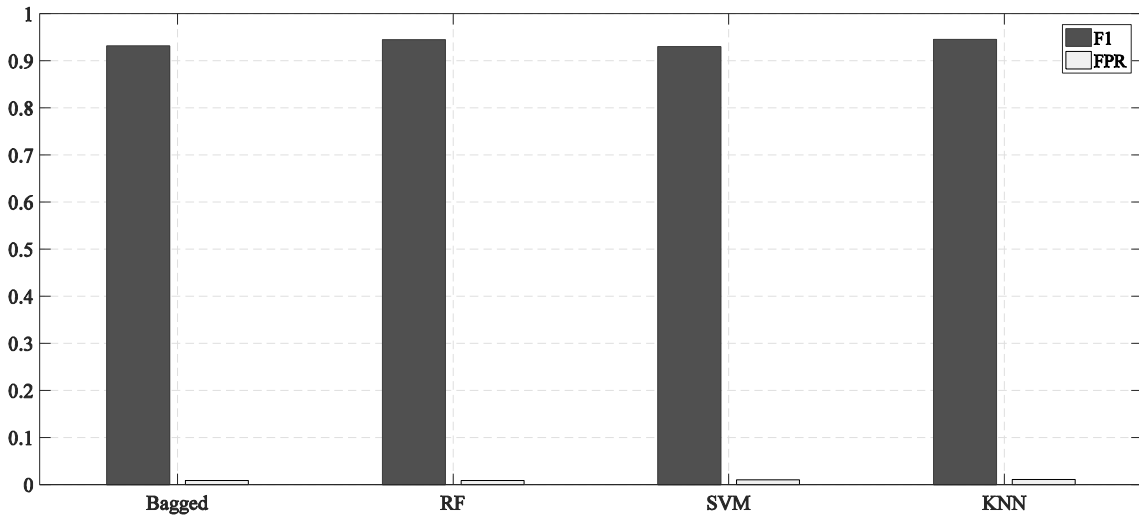
Classifier	Parameter	Values range	Value
Bagged Trees	N	[30 – 50]	50
	Min. Leaf Size	[1 – 5]	2
Random Forests	N	[30 – 50]	47
	K	[1 – 28]	9
	Min. Leaf Size	[1 - 5]	2
SVM	Kernel	{Polynomial – Gaussian – Linear}	Polynomial
	Coding Design	{OVA - OVO}	OVO
	Kernel Scale	[0 - 3]	N/A
	Polynomial Degree	[1 - 3]	2
	Soft Margin	[0 - 1]	1
KNN	K	[1 - 9]	5
	M	[1 - 28]	5
	N	[30 - 50]	42
	Distance Metric	{Euclidian – city block – cosine – hamming – mahalanobis – jaccard – minkowski}	city block
	Distance weight	{inverse – squared inverse – equal}	squared inverse
	Tie break	{nearest – smallest – random}	random

Table 6.2 shows the averaged  $F1_{score}$  over 9 users for each app in the dataset.

**Table 6.2 Evaluation results**

Classifier	Score	Class					
		Facebook	8 Ball Pool	Skype	Viber	WhatsApp	YouTube
Bagged Trees	FPR	0.02	0.004	0.001	0.001	0.021	0.005
	TPR	0.913	0.891	0.987	0.996	0.903	0.918
	F1	0.875	0.925	0.990	0.995	0.863	0.940
Random Forests	FPR	0.023	0.005	0.001	0.001	0.015	0.004
	TPR	0.920	0.954	0.985	0.997	0.890	0.910
	F1	0.878	0.961	0.989	0.995	0.889	0.945
SVM	FPR	0.014	0.006	0.001	0.001	0.030	0.008
	TPR	0.914	0.866	0.985	0.997	0.962	0.937
	F1	0.903	0.908	0.989	0.996	0.840	0.944
KNN	FPR	0.023	0.005	0.002	0.004	0.018	0.012
	TPR	0.864	0.939	0.974	0.998	0.896	0.916
	F1	0.847	0.954	0.982	0.991	0.979	0.920

We can notice that RF has the best overall averaged F1 score comparing the FPR rate for the 6 apps as shown in Figure 6.1.



**Figure 6.1 F1 score and FPR of all classifiers**

### 6.1.2 Feature Selection and Time Performance

Feature normalized rankings of each selection method are shown in Table 6.3.

The correlation method is implemented using all the samples combined in a single dataset.

As for the other wrapper methods, each method is implemented using the corresponding classifier with the best parameter values that we obtained from the parameter tuning phase. For example, for RELIEFF we've set the same distance metric, distance weighting, and number of neighbors. The same applies for all other classifiers regarding their corresponding parameters. The same dataset splitting technique that has been used for training and evaluating the classifiers in the parameter tuning phase is used. We highlight the top 10 features ranked by each method as shown in Table 6.3. As we can notice each method has ranked the features differently.

**Table 6.3 Features' ranking**

Feature #	Info Gain	Correlation	SVM	OOB Bagged	OOB RF	RELIEFF
1	0.511	<b>0.650</b>	0.229	<b>0.566</b>	<b>0.816</b>	<b>0.324</b>
2	0.596	<b>0.225</b>	0.342	<b>0.553</b>	<b>0.821</b>	0.220
3	0.545	0.217	0.394	0.242	<b>0.571</b>	0.015
4	<b>0.798</b>	<b>1.000</b>	0.486	<b>0.656</b>	<b>0.986</b>	0.174
5	<b>0.901</b>	<b>0.522</b>	<b>0.589</b>	0.142	<b>0.574</b>	0.201
6	<b>0.848</b>	0.180	<b>0.953</b>	<b>0.289</b>	0.347	0.000
7	0.142	0.000	<b>0.893</b>	0.191	0.183	0.171
8	0.079	0.114	0.429	0.075	0.196	<b>0.491</b>
9	0.085	0.058	0.411	0.107	0.165	0.058
10	0.038	0.168	0.143	0.086	0.205	0.240
11	0.497	0.064	<b>0.722</b>	0.236	0.427	0.083
12	<b>0.824</b>	<b>0.533</b>	<b>0.662</b>	<b>1.000</b>	<b>1.000</b>	<b>0.353</b>
13	0.203	0.125	0.369	<b>0.256</b>	<b>0.577</b>	0.093
14	0.553	0.130	0.575	0.137	0.216	0.111
15	0.577	0.068	0.489	<b>0.243</b>	0.391	0.133
16	<b>1.000</b>	0.210	<b>0.609</b>	<b>0.482</b>	0.491	<b>1.000</b>
17	0.151	0.013	0.376	0.222	0.425	0.046
18	<b>0.742</b>	<b>0.731</b>	0.400	0.231	0.529	0.254
19	0.335	0.090	<b>0.628</b>	0.242	<b>0.554</b>	0.162
20	<b>0.973</b>	<b>0.484</b>	0.381	<b>0.357</b>	<b>0.657</b>	<b>0.569</b>
21	0.250	0.087	0.195	0.126	0.531	0.209
22	<b>0.651</b>	<b>0.375</b>	<b>0.886</b>	0.063	0.260	<b>0.259</b>
23	0.632	0.081	0.530	0.081	0.426	0.149
24	<b>0.976</b>	0.046	<b>1.000</b>	0.223	0.542	<b>0.971</b>
25	0.192	0.019	0.458	0.196	0.411	0.066
26	<b>0.715</b>	<b>0.791</b>	0.303	<b>0.243</b>	<b>0.556</b>	<b>0.315</b>
27	0.169	<b>0.774</b>	<b>0.768</b>	0.000	0.016	<b>0.441</b>
28	0.000	0.128	0.000	0.031	0.000	<b>0.466</b>

To reevaluate the classifiers, we ordered the features in a descending order as ranked by each method. Then, we iterated over the features, and at each iteration we pulled out the least ranked feature from the training set and trained the classifier. For evaluation, the same feature is pulled out from the test set and the classifier is evaluated using the pre-mentioned technique. We report the averaged  $F1_{score}$  over the 9 user folds for the 6 apps. Additionally, we benchmark the training and evaluation time for each classifier when each feature was pulled out.

In Figure 6.2 we can notice the drop in the F1 score as we proceed with pulling the features from the training set according to the least ranked by each method. In the case of RF and Subspace KNN we stopped at 23 features because the random space and the subspace sizes in RF and KNN respectively are set to 5. KNN, RF, and Bagged classifiers reported a high evaluation score even when we pulled 75% of the features. However, SVM started reporting a lower score when 50% of the features were pulled. This indicates the significance of the remaining features in each set ranked by each method.

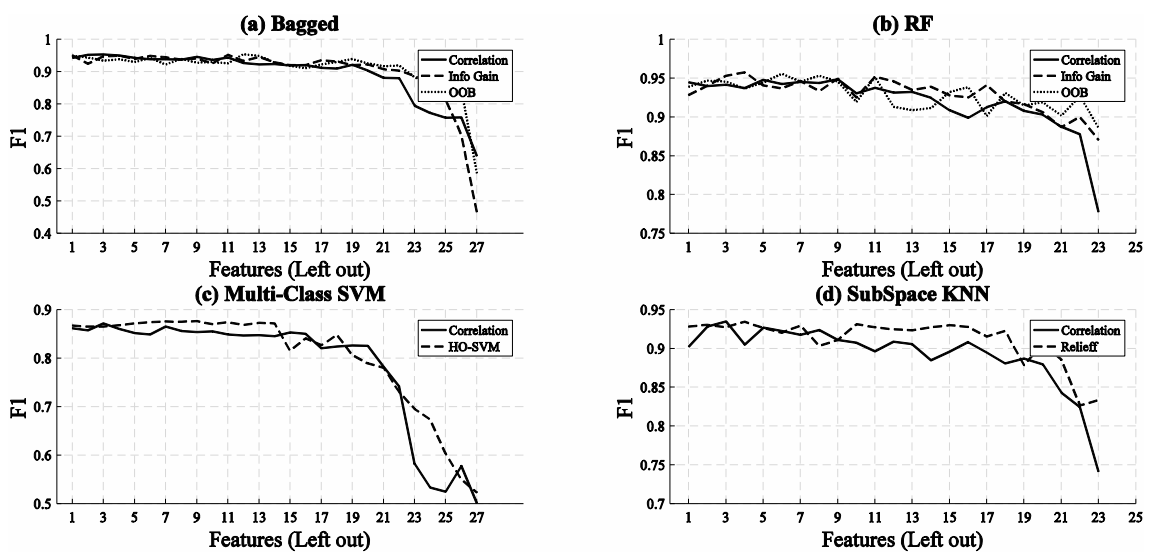


Figure 6.2 F1 score vs. feature vector size

Figure 6.3 shows the elapsed training time to train each classifier at each feature pull. We can notice that SVM has a relatively large training time compared to the other three classifiers. RF and Bagged reported a 50% decrease in the training time when pulling 75% of the features. The evaluation score stayed above 90% when the features are pulled with respect to the OOB predictor importance rank. Subspace KNN reported a relatively low training time compared to all other classifiers due to the decreased subspace size. However, the training time was not affected by pulling out the features. It is a result of executing the same distance measuring calculations using the same subspace size. Another reason is the nature of the KNN classifier which uses distance calculation instead of building a fixed a model to classify patterns.

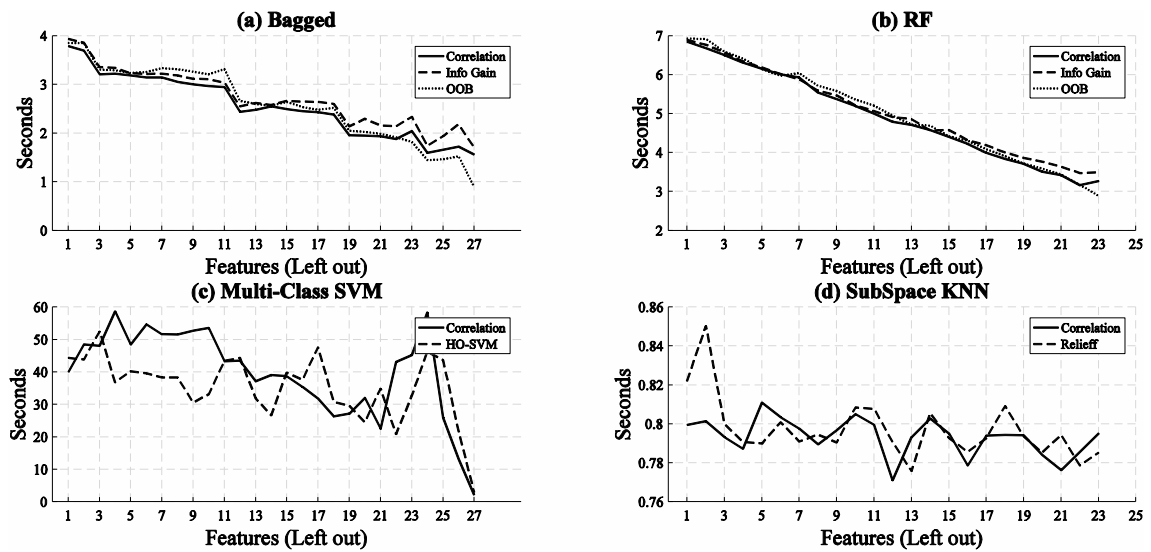


Figure 6.3 Training time vs. feature vector size

According to the results shown in Figure 6.2 and Figure 6.3, Bagged Trees and RF classifier have reported a better  $F1_{score}$  compared to the other two classifiers while pulling out the features. The random feature space technique which is used in RF helps

in defining significant random spaces to separate classes with higher accuracy [39]. Additionally, RF reports a reasonable training time with higher accuracy compared to SVM, Bagged, and KNN. Hence, RF is proven to be a better classifier. Thus, it is used in the classification model component in our implementation of the novelty detection process.

In all four models we used a parallel programming toolbox provided by Matlab for training and evaluation. The parallel programming toolbox reduces time consumption in terms of building classification ensembles such as RF, Bagged, and KNN. The same applies on SVM which consists of multiple binary SVM models that does binary classification between each pair of classes.

## **6.2 Classification and Novelty Detection Process Results**

We use the second data collection set to validate the novelty detection process. The process assumes that the voting distribution for each known class follows a Gaussian distribution with a mean  $\mu$  and standard deviation  $\sigma$ . The assumption is based on analyzing the vote counts for all classes' samples. Figure 6.4 shows the PDF of each class vote ratio over all correctly classified samples in the second data collection set when all classes are known.



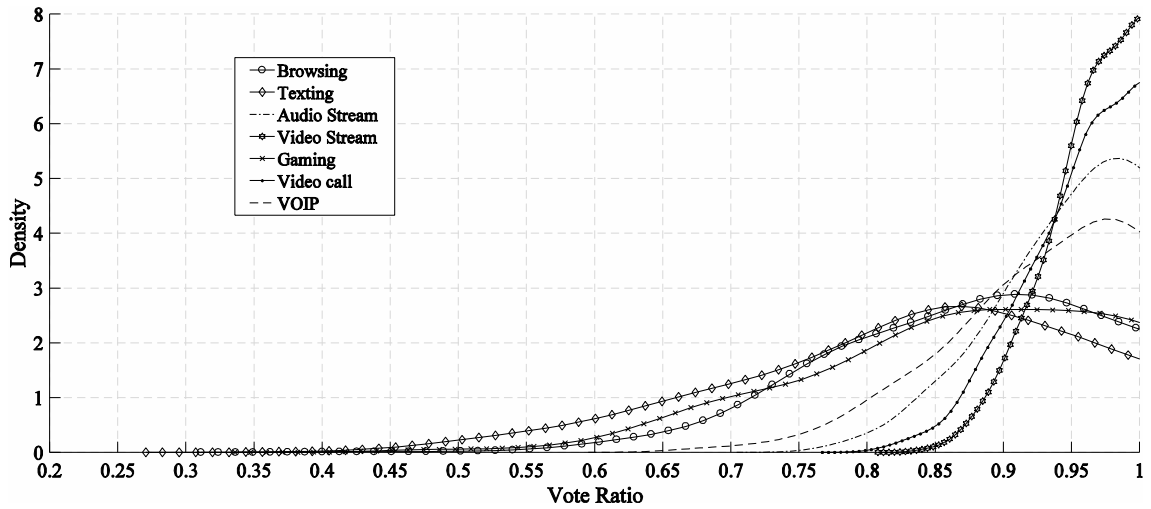


Figure 6.4 Accurately classified samples vote PDF

The process implementation requires setting important parameters shown in Table 6.4. Setting the correct values for these parameters is an essential step before validating the detection accuracy.

Table 6.4 Process parameters

#	Parameter	Value Range	Description
1	$P_{threshold}$	[0.33 - 1]	Probability threshold of assigning the preliminary class for the testing sample.
2	$Flow_{min}$	[1 - 5]	Minimum flow length measured in samples count that can be processed by the novelty detection process.
3	$w$	[1 - 5]	Size of the moving window in samples.
4	$\alpha$	[0.5 - 1]	Weighting attribute in calculating the decision risk from flow history and upcoming samples
5	$\lambda_H$	[0.5 - 1]	Bayesian decision loss matrix element which is a weight for assigning “Unknown” as class for the predecessors of the testing sample from the same flow.
6	$\lambda_F$	[0.5 - 1]	Bayesian decision loss matrix element which is a weight for assigning “Unknown” as a class for the successors of the testing sample from the same flow.
7	$Samples_{threshold}$	[0.3 - 1]	A penalty threshold which the process compares against when the preliminary class is empty and the sample has a high risk to be “Unknown”.

Values' ranges are set based on experimental knowledge to deliver best classification and detection accuracy in pseudo real-time approach as will be shown in the results section. For example, setting the upper limit of the moving window to 5 samples enables the process to detect novel samples with a maximum 10 seconds delay. Setting the minimum flow length to 5 samples enables the process to detect relatively short flows.

The validation is divided into two sections. In the first section, we validate the ability of the process to detect novel samples of benign classes e.g. browsing. Extracted samples from the second data collection set that resulted from the 5 tuples preprocessing approach are used in the first validation section. In the second section, we validate the ability of the process to detect novel samples of malicious classes e.g. DoS. We use the samples in the second data collection set that resulted from the 3 tuples preprocessing approach since the attacks samples are generated using this approach only. The classification and detection accuracies are represented by four indicators.

1. Recall or True Positive Rate (TPR): the number of correctly classified samples of a novel class to the total number of novel class samples, see Equation (6.5).

$$TPR = \frac{TP}{TP + FN} \quad (6.5)$$

2. Fall-out or False Positive Rate (FPR): the number of misclassified samples of a known class to the total number of known non-class samples, see Equation (6.6).

$$FPR = \frac{FP}{FP + TN} \quad (6.6)$$

3. Specificity or True Negative Rate (TNR): the number of correctly classified samples of a known class to the total number of known non-class samples, see Equation (6.7).

$$TNR = \frac{TN}{TN + FP} \quad (6.7)$$

4. False Negative Rate (FNR): the number of misclassified samples of a novel class to the total number of novel non-class samples, see Equation (6.8).

$$FNR = \frac{FN}{FN + TP} \quad (6.8)$$

### 6.2.1 *Detecting Novel Samples of Benign Classes*

In this validation section we iterate over each known benign class and pull its samples from the training dataset. For each iteration we validate the classification accuracy for the samples of known classes in addition to the detection accuracy of the novel samples coming from the unknown class. The evaluation mechanism divides the data set into a training set that forms 70% of the data, a validation set that forms 20% of the data, and a testing set which includes the rest of the remaining samples. Dividing the samples into these three sets is based on the flow ID. The training dataset includes flows from a class  $c_i$  which are not introduced in the validation set nor in the testing set. The same applies for the validation and testing sets.

We use the Bayesian optimization approach to optimize the novelty detection process parameters. Hyper-parameters of the RF ensemble are also included in the optimization process. We have set the generalization function  $f(x)$  to be  $(1 - TPR)$  of the detection accuracy for novel samples averaged over all unknown classes which were pulled one by one. However, to control the optimization process we've set a Coupled

Constraint [54] to be the averaged FPR value over all known classes to ensure a balanced optimization and decrease the false alarm rate. At each iteration a training set is extracted using the (70,20,10) mechanism then the samples of class  $c_i$  are removed. After training the RF ensemble, we use the validation set after removing the samples of class  $c_i$  to calculate  $\mu(PDF_{c_i})$  and  $\sigma(PDF_{c_i})$  for each known class. Finally, the testing set is used to validate the classification novelty detection process.

At the end of the optimization process we obtained the parameter values shown in Table 6.5 along with the validation results which represent the minimum observed objective function with respect to the optimization constraint.

**Table 6.5 Process parameters values (detecting novel samples of benign classes )**

#	Parameter	Value
1	$P_{threshold}$	0.39
2	$Flow_{min}$	3
3	$w$	5
4	$\alpha$	0.33
5	$\lambda_H$	0.98
6	$\lambda_F$	0.56
7	$Sample_{threshold}$	0.75
8	(RF) N	43
9	(RF) K	2
10	(RF) Min. Leaf Size	4

Figure 6.5 (a) shows the detection accuracy for each class when it was left out of the training set (TPR). We notice a good detection accuracy for all the classes. Figure 6.5

(b) shows the detection accuracy (TNR) and the false alarm rate (FPR) for each class when it was included in the training set averaged over the number of iterations. All evaluation iterations resulted a low false alarm rate for all the classes.

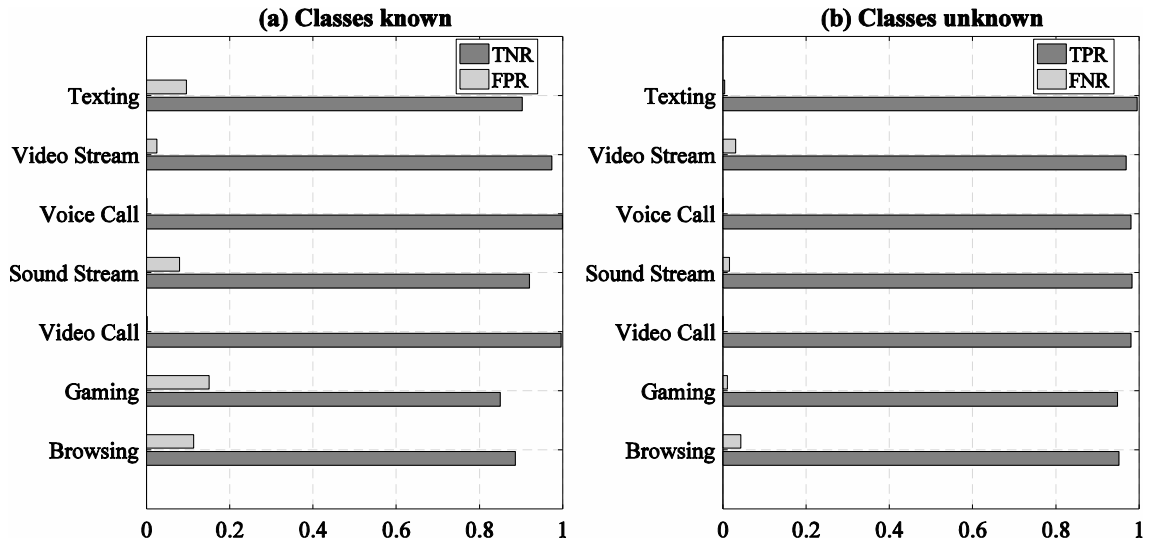


Figure 6.5 Novelty detection results of benign classes

### 6.2.2 Detecting Novel Samples of Malicious Classes

To validate the detection accuracy of novel malicious samples, the RF ensemble is trained using samples from benign classes only. Then, we use each malicious class samples to validate the novelty detection process. For each validation round, the RF ensemble will have knowledge about all the benign classes since it was trained using these classes. We use the same dataset dividing mechanism (70,20,10) which is mentioned in the latter section. However, the samples of the benign classes will stay as a part of the training and validation sets to calculate the  $\mu(PDF_{C_i})$  and  $\sigma(PDF_{C_i})$  for each known benign class. The testing set will contain samples from malicious classes only.

Similar to the former section we use the Bayesian optimization approach to set the novelty detection process parameters' and RF hyper-parameters values to achieve best

detection accuracy over the malicious classes.

Table 6.6 shows the parameters' values obtained at the end of the optimization process.

**Table 6.6 Process parameters values (detecting novel samples of malicious classes )**

#	Parameter	Value
1	$P_{threshold}$	0.34
2	$Flow_{min}$	5
3	$w$	4
4	$\alpha$	0.16
5	$\lambda_H$	0.97
6	$\lambda_F$	0.92
7	$Sample_{threshold}$	0.56
8	(RF) N	48
9	(RF) K	13
10	(RF) Min. Leaf Size	1

Figure 6.6 (a) shows the detection accuracy of novel malicious samples (TPR) along with the FNR. Figure 6.6 (b) shows the classification accuracy of samples coming from known classes along with the FPR. We can notice from the figures that the process is capable of detecting novel flows with high accuracy. Novel samples from benign flows detection accuracy is around 0.97 with a false alarm rate of 0.06. Novel samples of malicious flows detection accuracy reached 0.93 with a low false alarm rate of 0.03.

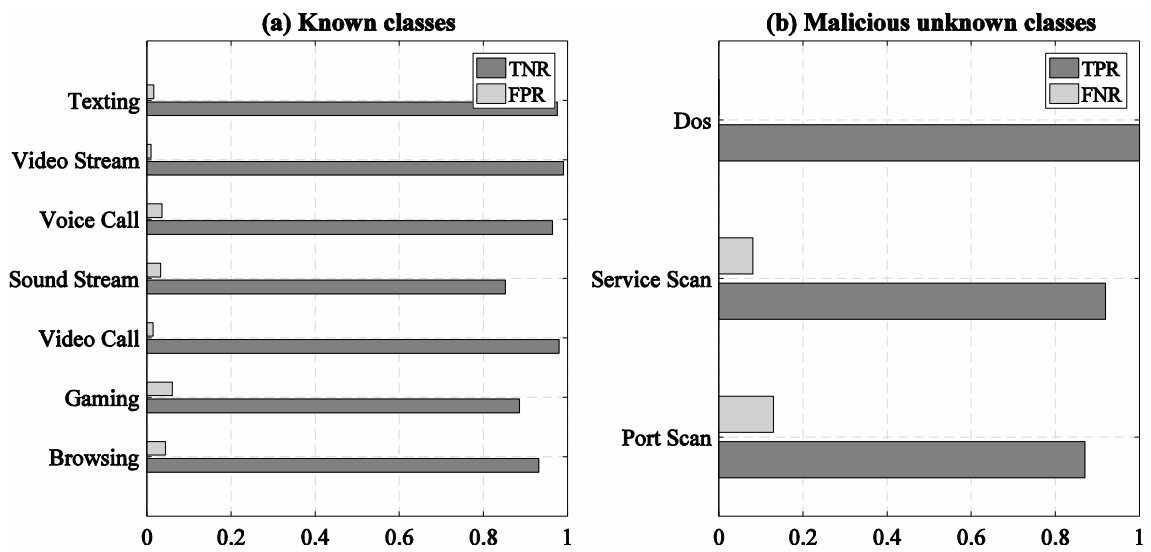


Figure 6.6 Novelty detection results of malicious classes

Both testing scenarios proved that the process can detect these flows regardless of their origin. We note that the type and count of the known classes plays a major role in detecting novel flows. As mentioned, the confidence score depends on the characteristic of the known class and the test sample. Accordingly, increasing the number of the unique known classes will enable the RF ensemble to assign the confidence score with more precision. In other words, the voting distribution is affected with the number of known classes and their uniqueness. The more classes we include in the training phase, the more distributed the votes will be. This is true if the characteristics of the novel test sample which are defined by the features' values tend to follow the characteristics of more than one class.

## CHAPTER 7

### CONCLUSION

In this thesis, we've successfully designed and implemented a classification and novelty detection process. The process is defined to classify and detect known and novel samples. These samples represent traffic flows generated by mobile apps. The flows are extracted by applying two preprocessing approaches known as the 5 tuples and the 3 tuples. Our contribution can be summarized in three main deliverables.

The first contribution is represented in the data collection experiment. Through our experimental setup, we were successful to collect packet dumps and connection flows of mobile apps running on Android devices. The connections' info is collected using a self-designed client-app that runs as a background service. Matching connections info with the collected packet dumps produced ground-truth labeled flows that represent mobile apps actions. Finally, the produced dataset of feature vectors (samples) represent the main part of the first deliverable.

The second contribution relies in the comparative analysis study of the classifiers. Using the extracted dataset, we trained and evaluated multiple supervised machine learning classifiers in terms of accuracy and time performance. The hyper-parameters of the classifiers were tuned using Bayesian optimization. In addition to tuning the hyper-parameters for evaluation, a feature selection phase was performed to increase the performance and identify significant features. Finally, the main part of this deliverable was identifying the best classifier to serve as a classification model in the novelty detection process.



The third and final contribution of this thesis is the design and implementation of a flow novelty detection process. The process uses Bayesian decision theory to detect novel flows. The process uses a confidence score produced by the classification model to infer samples classes whether known or novel. Using Bayesian decisions helped in avoiding threshold-based detection which requires instant updates and tweaking. As shown, the process delivered good detection results with a low false alarm rate.

In this study we faced an essential limitation presented by the lack of labeled datasets that represent mobile apps traffic. The produced dataset in this thesis was contributed by a limited number of users in a closed experimental environment. Additionally, the labeling approach was controlled by setting a limited set of actions for each app. Some users didn't comply to these actions which was a direct cause of producing noisy samples in some of the classes. We think that the latter limitation is out of this study's scope because it is practically difficult to assign labels to apps' actions accurately without producing noise samples. It is due to the way that Android OS, which is the used mobile platform in this experiment, treats established connections. However, an important part of the future work for this study is creating a larger dataset that contains more apps and users.

Another limitation in this study relies in the novelty detection process. As shown, the process can detect samples of novel flows with high accuracy. However, the process lacks the ability of identifying the type of the novel flow. We believe that such an action requires some interaction with the external world e.g. asking users about their latest activities on the phone. As such, the authors of [55] developed an app-rating system based on crowd interactions. Getting users feedback represent an external feedback loop which

can help in labeling detected novel flows in our proposed solution. Hence, designing and implementing this feedback link forms another major part of our future work.

## BIBLIOGRAPHY

- [1] “IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012.” [Online]. Available: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. [Accessed: 04-Apr-2017].
- [2] “Number of Apps in leading App stores 2016 | Statistic.” [Online]. Available: <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>. [Accessed: 01-Jan-2016].
- [3] H. Pieterse and M. S. Olivier, “Android botnets on the rise: Trends and characteristics,” in *Information Security for South Africa (ISSA), 2012*, 2012, pp. 1–5.
- [4] K. W. Miller, J. Voas, and G. F. Hurlburt, “BYOD: Security and privacy considerations,” *It Prof.*, no. 5, pp. 53–55, 2012.
- [5] M. Pierer, *Mobile Device Management (MDM)*. Springer, 2016.
- [6] A. Houmansadr, S. Zonouz, R. Berthier, and others, “A cloud-based intrusion detection and response system for mobile phones,” in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, 2011, pp. 31–32.
- [7] I. Tsompanidis, A. H. Zahran, and C. J. Sreenan, “Mobile network traffic: A user behavior model,” in *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*, 2014, pp. 1–8.
- [8] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, “Networkprofiler: Towards automatic fingerprinting of android apps,” in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 809–817.
- [9] Y. Choi, J. Y. Chung, B. Park, and J. W.-K. Hong, “Automated classifier generation for application-level mobile traffic identification,” in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 1075–1081.
- [10] M. Conti, L. V Mancini, R. Spolaor, and N. V. Verde, “Can’t you hear me knocking: Identification of user actions on Android apps via traffic analysis,” in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 297–304.
- [11] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, “‘Andromaly’: a behavioral malware detection framework for android devices,” *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.

- [12] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "ProfileDroid: multi-layer profiling of android applications," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012, pp. 137–148.
- [13] S. Rosen, Z. Qian, and Z. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," ... *Data Appl. Secur. Priv.*, pp. 221–232, 2013.
- [14] K. Mongkolluksamee, Sophon and Visoottiviseth, Vasaka and Fukuda, "Combining Communication Patterns & Traffic Patterns to Enhance Mobile Traffic Identification Performance," *J. Inf. Process.*, vol. 24, no. 2, pp. 247–254, 2016.
- [15] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.
- [16] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Comput. Secur.*, vol. 43, pp. 1–18, 2014.
- [17] Y. Qi, M. Cao, C. Zhang, and R. Wu, "A Design of Network Behavior-Based Malware Detection System for Android," in *Algorithms and Architectures for Parallel Processing*, Springer, 2014, pp. 590–600.
- [18] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 281–287.
- [19] S. K. Baghel, K. Keshav, and V. R. Manepalli, "An investigation into traffic analysis for diverse data applications on smartphones," in *Communications (NCC), 2012 National Conference on*, 2012, pp. 1–5.
- [20] J. Li, L. Zhai, X. Zhang, and D. Quan, "Research of android malware detection based on network traffic monitoring," in *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on*, 2014, pp. 1739–1744.
- [21] M. Zaman, T. Siddiqui, M. R. Amin, and M. S. Hossain, "Malware detection in Android by network traffic analysis," in *Networking Systems and Security (NSysS), 2015 International Conference on*, 2015, pp. 1–5.
- [22] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing Android Encrypted Network Traffic to Identify User Actions," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 1, pp. 114–125, 2016.
- [23] D. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *KDD*, 1994, vol. 398.

- [24] I. Taylor, Vincent F and Spolaor, Riccardo and Conti, Mauro and Martinovic, “Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic,” *2016 IEEE Eur. Symp. Secur. Priv.*, pp. 439–454, 2016.
- [25] A. Moore, D. Zuev, and M. Crogan, “Discriminators for use in flow-based classification,” *Queen Mary Westf. Coll. Dep. Comput. Sci.*, 2005.
- [26] M. Zulkernine and A. Haque, “Random-Forests-Based Network Intrusion Detection Systems,” *IEEE Trans. Syst. Man. Cybern.*, vol. 38, no. 5, pp. 649–659, 2008.
- [27] Q.-F. Zhou, H. Zhou, Y.-P. Ning, F. Yang, and T. Li, “Two approaches for novelty detection using random forest,” *Expert Syst. Appl.*, vol. 42, no. 10, pp. 4840–4850, 2015.
- [28] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” *Emerg. Artif. Intell. Appl. Comput. Eng. real word AI Syst. with Appl. eHealth, HCI, Inf. Retr. pervasive Technol.*, 2007.
- [29] J. R. Quinlan, “Induction of Decision Trees quinlan.pdf,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
- [30] I. Rish, “An empirical study of the naive Bayes classifier,” *IJCAI 2001 Work. Empir. methods Artif. Intell.*, vol. 22230, pp. 41–46, 2001.
- [31] J. Mockus, V. Tiesis, and A. Zilinskas, “The application of Bayesian methods for seeking the extremum,” in *Towards global optimisation. II*, 1978, pp. 117–129.
- [32] D. R. Jones, “A Taxonomy of Global Optimization Methods Based on Response Surfaces,” *J. Glob. Optim.*, vol. 21, pp. 345–383, 2001.
- [33] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning.*, vol. 14, no. 2. 2004.
- [34] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” *Adv. Neural Inf. Process. Syst. 25*, pp. 1–9, 2012.
- [35] L. Breiman, “Bagging predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.
- [36] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.
- [37] J. R. Quinlan, *C4.5: Programs for Machine Learning*, vol. 1, no. 3. 1992.
- [38] E. Bauer and R. Kohavi, “An empirical comparison of voting classification algorithms: Bagging, boosting, and variants,” *Mach. Learn.*, vol. 36, no. 1–2, pp. 105–139, 1999.

- [39] L. (University of C. Breiman, “Random forest,” *Mach. Learn.*, vol. 45, no. 5, pp. 1–35, 1999.
- [40] E. Osuna, R. Freund, and F. Girosi, “Support Vector Machines : Training and Applications,” *Massachusetts Inst. Technol.*, vol. 9217041, no. 1602, 1997.
- [41] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *Am. Stat.*, vol. 46, no. 3, pp. 175–185, 1992.
- [42] T. K. Ho, “The random subspace method for constructing decision forests,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, 1998.
- [43] S. Maldonado and R. Weber, “A wrapper method for feature selection using Support Vector Machines,” *Inf. Sci. (Ny).*, vol. 179, no. 13, pp. 2208–2217, 2009.
- [44] A. W. Moore, “Information Gain,” *Wwwcscmu.edu Awm*, vol. 34, no. 3, pp. 1–10, 2003.
- [45] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, “Overcoming the myopia of inductive learning algorithms with RELIEFF,” *Appl. Intell.*, vol. 7, no. 1, pp. 39–55, 1997.
- [46] Matlab, “OOB Observations.” [Online]. Available: <http://www.mathworks.com/help/stats/classificationbaggedensemble.oobloss.html>. [Accessed: 01-Jan-2016].
- [47] L. Breiman, “Out-Of-Bag Estimation,” Berkeley CA 94708, 1996.
- [48] Matlab, “Classification Margin.” [Online]. Available: <http://www.mathworks.com/help/stats/compactclassificationensemble.margin.html>. [Accessed: 01-Jan-2016].
- [49] “Wireshark.” [Online]. Available: <https://www.wireshark.org/>. [Accessed: 01-Apr-2017].
- [50] Kost, “Nmap for Android.” [Online]. Available: <https://github.com/kost/nmap-android>. [Accessed: 26-Mar-2017].
- [51] M3rgb, “Packet Generator for Android.” [Online]. Available: <https://forum.xda-developers.com/showthread.php?t=2647945>. [Accessed: 26-Mar-2017].
- [52] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, “Who do you sync you are?: smartphone fingerprinting via application behaviour,” in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, 2013, pp. 7–12.
- [53] Matlab, “bayesopt.” [Online]. Available: <https://www.mathworks.com/help/stats/bayesian-optimization-algorithm.html>. [Accessed: 28-Mar-2017].

- [54] Matlab, “Constraints in Bayesian Optimization.” [Online]. Available: <https://www.mathworks.com/help/stats/constraints-in-bayesian-optimization.html>. [Accessed: 01-Apr-2017].
- [55] F. Saab, I. Elhadj, A. Kayssi, and A. Chehab, “A crowdsourcing game-theoretic intrusion detection and rating system,” in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 622–625.