



AMERICAN UNIVERSITY OF BEIRUT

Crowdsourcing for Mobile Security: Modelling,  
Psychological Bias, and Performance of  
Aggregation Methods

by

FARAH WALID SAAB

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

Beirut, Lebanon  
December 2018

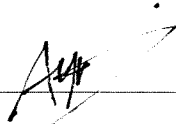
# AMERICAN UNIVERSITY OF BEIRUT

## Crowdsourcing for Mobile Security: Modelling, Psychological Bias, and Performance of Aggregation Methods

by  
FARAH WALID SAAB

Approved by:

\_\_\_\_\_  
Dr. Ayman Kayssi, Professor  
Electrical and Computer Engineering

  
Chairman

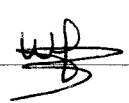
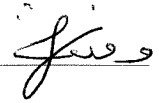
\_\_\_\_\_  
Dr. Imad Elhajj, Professor  
Electrical and Computer Engineering

  
Advisor

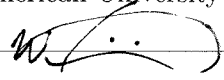
\_\_\_\_\_  
Dr. Ali Chehab, Professor  
Electrical and Computer Engineering

  
Co-Advisor

\_\_\_\_\_  
Dr. Wissam Fawaz, Associate Professor  
Electrical and Computer Engineering, Lebanese American University

   
Member of Committee

\_\_\_\_\_  
Dr. Wassim Itani, Associate Professor  
Electrical and Computer Engineering, Beirut Arab University

  
Member of Committee

Date of dissertation defense: September 21, 2018

# AMERICAN UNIVERSITY OF BEIRUT

## THESIS, DISSERTATION, PROJECT, RELEASE FORM

Student Name: Saab \_\_\_\_\_ Farah \_\_\_\_\_ Walid \_\_\_\_\_  
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

7-12-2018  
\_\_\_\_\_  
Date

# Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Imad El-hajj and my co-advisors Prof. Ayman Kayssi and Prof. Ali Chehab for their continuous support and motivation, their patience, and immense knowledge. Their guidance helped during my studies and in writing of this dissertation. I could not have imagined having better mentors throughout my graduate years.

I would also like to thank the rest of my committee: Prof. Wissam Fawaz and Prof. Wassim Itani, for their insightful comments and hard questions which led me to examine my research topic from many perspectives.

I thank my colleague and friend, Georgi Ajaeiya, for our research discussions and his tremendous help during the system implementation phase.

To my friends, thank you for listening, offering me advice, and supporting me throughout this entire process. Special thanks to Chadi Trad and Sarah Abdallah. Thank you for your thoughts and for being there whenever I needed a friend. And most importantly, thank you for all the editing advice ☺.

Last but not least, I would like to thank my family especially my parents and my sister for supporting me in any way they could throughout writing my dissertation. Finally, I thank my grandmother for always believing in me and inspiring me to be the best version of myself. Whatever I am today is due to the values she instilled in me.

# An Abstract of the Dissertation of

Farah Walid Saab for Doctor of Philosophy  
Major: Electrical and Computer Engineering

Title: Crowdsourcing for Mobile Security: Modelling, Psychological Bias, and Performance of Aggregation Methods

Since the introduction of the Android platform in October 2003, mobile developers have been creating apps for different purposes. This large variety of apps brought along considerable malware that has infected millions of Android devices throughout the years. Trojans, worms, viruses, and spyware have found their way onto user devices through mobile apps. In addition to possible malware infection, a good percentage of these apps have low utility and are generally not desirable by users. The first line of defense against malware was app store vetting. Then came user ratings and reviews which provided a better understanding of the general effectiveness of these mobile apps. The problem, however, is that reviews are not reliable and numerical ratings do not really describe the exact shortcomings, if any, within an app. A low app rating could be attributed to a variety of different reasons such as low utility or high resource consumption. Some users are interested in knowing whether an app is malicious. Others are interested in detecting poor app designs that reduce app utility. Therefore, depending on their expectations, users' numerical ratings will be different. In this dissertation, we provide a fine-grained analysis of apps that is not offered on app stores nowadays. We develop a system for composite rating for apps that includes an objective and a subjective score. The objective score does not depend on input from device users and only measures the utility and performance of an app. The subjective score, on the other hand, aggregates collected input from the crowd in an attempt to measure an app's suspiciousness level. Human input is central to assessing an app since the challenge in rating is in capturing context and user perspective. For example, when an app consumes a large amount of bandwidth, this could be due to malicious behavior or simply as a result of user

activities on the device (viewing a video, downloading a file, etc.). This “context” is very challenging to capture without user input. It is this input that we are crowdsourcing in order to transform subjective input into a subject-independent score. Of course, the employed aggregation technique when crowdsourcing user input has a major effect on the resulting score. To this end, we develop formal mathematical models for some of the most popular crowdsourcing aggregation techniques.

One of the challenges of aggregation is accounting for considerable bias emanating from human factors. One of the most critical human factors to consider when collecting input from a crowd is referred to as the Dunning-Kruger effect. It states that low-ability individuals suffer from the illusion that their abilities are higher than what they really are. This psychological bias has a significant effect on the performance of confidence-related aggregation techniques which are modelled in this dissertation. To this end, we formally model the Dunning-Kruger bias and study its effect on the confidence-weighted and the maximum confidence aggregation techniques as compared to plurality voting. The resulting modelled app rating system is implemented and tested on real user devices for a period of six months during which different aggregation techniques are employed and general trends in app and device usage are studied. Our main contributions in this dissertation are: (1) Designing and implementing a system that can provide users with a composite app score that is based on both utility of an app and maliciousness level, (2) Modelling the Dunning-Kruger psychological bias, (3) Modelling and comparing the performance of some of the most popular aggregation techniques (plurality, confidence-weighted, maximum confidence, and a newly derived competence-weighted approach) while taking into consideration the effect of the demonstrated Dunning-Kruger bias on output performance, (4) Studying general trends in app and device usage on both mobile phones and tablets at different times of the day and different days of the week.

# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contributions . . . . .	8
1.2 Dissertation Structure . . . . .	9
<b>2 Literature Survey</b>	<b>11</b>
2.1 Crowdsourcing Applications . . . . .	11
2.1.1 Crowdsourcing Location . . . . .	13
2.1.2 Crowdsourcing for Network Mapping . . . . .	16
2.1.3 Crowdsourcing Health Care . . . . .	17
2.2 Different Crowdsourcing Task Types . . . . .	19
2.3 Crowdsourcing for Security . . . . .	22
2.4 Crowdsourcing-Based IDS with Game Theory . . . . .	28
2.5 Crowdsourcing for Recommender Systems with Expert Detection Techniques . . . . .	34
2.6 Quality Assurance in Crowdsourcing Systems . . . . .	43
2.7 Modeling Cognitive Biases in Crowdsourcing Systems . . . . .	46



2.8	Comparative Analysis . . . . .	49
<b>3</b>	<b>Crowdsourced Game-Theoretic Rating System</b>	<b>54</b>
3.1	Methodology . . . . .	57
3.1.1	Utility Score Computation . . . . .	58
3.1.2	Behavior Score Computation . . . . .	61
3.2	Mathematical Model . . . . .	71
3.3	Simulation Setup . . . . .	74
3.3.1	Utility Score Setup . . . . .	75
3.4	Design Results and Analysis . . . . .	90
3.4.1	Utility Score Results . . . . .	90
3.4.2	Behavior Score Results . . . . .	95
<b>4</b>	<b>Probabilistic Models for Aggregation in Voting Systems</b>	<b>112</b>
4.1	Modeling the Dunning-Kruger Effect . . . . .	113
4.1.1	Function Constraints . . . . .	115
4.1.2	Competence Model . . . . .	116
4.1.3	Notes . . . . .	118
4.2	Modeling Aggregation . . . . .	119
4.2.1	Plurality . . . . .	119
4.2.2	Confidence-Weighted . . . . .	122
4.2.3	Maximum Confidence . . . . .	124
4.2.4	Competence-Weighted . . . . .	128
4.3	Numerical Analysis . . . . .	130
4.3.1	Dataset . . . . .	130
4.3.2	Results and Discussions . . . . .	131
4.3.3	The Irregular Crowd . . . . .	137

4.4	Discussion . . . . .	139
<b>5</b>	<b>System Design and Implementation</b>	<b>142</b>
5.1	General Overview . . . . .	143
5.2	Client Side . . . . .	144
5.2.1	Method of Collecting and Storing Logs . . . . .	144
5.2.2	Method of Detecting Events . . . . .	147
5.2.3	Querying Process . . . . .	152
5.2.4	Implementing Feature Collection . . . . .	162
5.2.5	App Permissions . . . . .	165
5.3	Server Side . . . . .	166
5.3.1	MySQL Database . . . . .	166
5.4	System Performance . . . . .	168
<b>6</b>	<b>Results and Analysis</b>	<b>170</b>
6.1	Institutional Review Board . . . . .	171
6.2	Experiment Overview . . . . .	174
6.3	Collected Data . . . . .	175
6.4	Results of Aggregation Performance and App Classification . . . . .	179
6.4.1	Utility Scores . . . . .	179
6.4.2	Behavior Scores . . . . .	181
6.5	Discussion . . . . .	201
<b>7</b>	<b>Conclusions and Future Directions</b>	<b>202</b>
7.1	Conclusions Related to the Game-Theoretic Composite App Rating System . . . . .	203
7.2	Conclusions Related to Modelling of Aggregation Methods . . . . .	204

7.3	Conclusions Related to CrowdApp Rating Scores and Performance	206
7.4	Future Directions . . . . .	209
<b>A</b>	<b>Proofs of Theorems</b>	<b>214</b>
A.1	Theorem I - Plurality vs. Confidence-Weighting . . . . .	214
A.1.1	Case 1: $\Delta_k = 0.5$ . . . . .	215
A.1.2	Case 2: $\Delta_k > 0.5$ . . . . .	215
A.2	Theorem II - Plurality vs. Maximum Confidence . . . . .	216
A.3	Theorem III - Maximum Confidence vs. $\mathcal{P}$ . . . . .	219
A.4	Theorem IV - Confidence-Weighting vs. Maximum Confidence . .	219
A.4.1	Case 1: $\Delta_k = 0$ . . . . .	221
A.4.2	Case 2: $\Delta_k = 0.5$ . . . . .	221
A.4.3	Case 3: $\Delta_k = 1$ . . . . .	222
<b>B</b>	<b>Device and App Usage Stats</b>	<b>223</b>
B.1	Users and Device Type Detection . . . . .	224
B.2	Statistics Related to Apps and App Categories . . . . .	227
B.2.1	Categories Usage Statistics . . . . .	232
B.3	Discussion and Conclusions Related to Device and App Usage . .	241
	<b>Bibliography</b>	<b>243</b>

# List of Figures

3.1	System overview showing the crowd, CrowdApp on their devices, and the different engines on the server side . . . . .	56
3.2	App rating process describing how the overall rating of an app is based on utility and behavior scores . . . . .	58
3.3	Payoff matrix showing the gains, losses, and costs of both players for each of their actions . . . . .	67
3.4	The expected payoffs of SYS when its actions are to defend or not to defend . . . . .	69
3.5	Usage.txt and Period.txt sample output . . . . .	78
3.6	Pseudo-code of described method . . . . .	89
3.7	Running time . . . . .	97
3.8	The change in output values as the total number of users in the system increases . . . . .	98
3.9	The change in output values as the crowd expertise increases from 0.1 to 1 in steps of 0.1 . . . . .	101
3.10	The change in output values as the fraction of malicious apps increases from 0.1 to 1 . . . . .	103
3.11	The change in output values given selected differences between malicious and normal event rates . . . . .	104

3.12	The change in output values as SYS's defense cost increases from 0 to 6 in steps of 0.1 . . . . .	106
3.13	ROC curves of app and user classification . . . . .	108
4.1	Four sample DK plots . . . . .	114
4.2	Difference between success probability of plurality and confidence- weighted voting across all difficulty levels and for different values of <b>a</b> and <b>b</b> . . . . .	123
4.3	Success probabilities of plurality, confidence-weighted, and three maximum confidence approaches with different percentage values .	127
4.4	Success probabilities of different aggregation methods based on data from WWTBAM dataset . . . . .	132
4.5	Performance of different aggregation methods based on data from WWTBAM dataset . . . . .	133
5.1	Client-server architecture . . . . .	143
5.2	Sample CrowdApp v2.0 push notification . . . . .	158
5.3	Sample output of Linux top command . . . . .	163
5.4	Sample output of Linux netstat command . . . . .	164
6.1	Rates of Plurality experiment . . . . .	176
6.2	Rates of Confidence-Weighted experiment . . . . .	177
6.3	Rates of Maximum Confidence experiment . . . . .	177
6.4	Rates of Competence-Weighted experiment . . . . .	178
6.5	Rates of Surprisingly Popular experiment . . . . .	178
6.6	Behavior score convergence for selected apps given Plurality ag- gregation technique . . . . .	186

6.7	Behavior score convergence for selected apps given Confidence-Weighted aggregation technique . . . . .	187
6.8	Behavior score convergence for selected apps given Maximum Confidence aggregation technique . . . . .	187
6.9	Behavior score convergence for selected apps given Competence-Weighted aggregation technique . . . . .	188
6.10	Behavior score convergence for selected apps given Surprisingly Popular aggregation technique . . . . .	188
6.11	Percentage change in app reputation in the False Event Rate experiment . . . . .	192
6.12	Cumulative change in app reputation in the False Event Rate experiment . . . . .	193
6.13	Mirrored confidence-competence plot . . . . .	193
6.14	Main activity of MalApp . . . . .	196
6.15	CPU and memory consumption of MalApp - part 1 . . . . .	197
6.16	CPU and memory consumption of MalApp - part 2 . . . . .	197
6.17	CPU and memory consumption of MalApp - part 3 . . . . .	197
6.18	Generated memory-related question for MalApp . . . . .	198
6.19	Generated down bandwidth question for MalApp . . . . .	199
6.20	Generated up bandwidth question for MalApp . . . . .	200
B.1	A histogram of app download distribution . . . . .	228
B.2	Daily usage time of popular apps . . . . .	229
B.3	Hourly usage time of popular apps . . . . .	229
B.4	Average daily opening frequency of popular apps . . . . .	231
B.5	Normalized daily opening frequencies of popular apps . . . . .	231

B.6 Average values different features of popular apps . . . . . 232

B.7 Average daily usage time of phones versus tablets . . . . . 233

B.8 Daily usage time of selected categories on all devices . . . . . 234

B.9 Daily usage time of selected categories in phones . . . . . 234

B.10 Daily usage time of selected categories on tablets . . . . . 234

B.11 Average daily opening frequency on all devices . . . . . 236

B.12 Normalized average opening frequencies of selected categories on  
phones versus tablets . . . . . 238

B.13 Normalized daily opening frequencies of selected categories on all  
devices . . . . . 238

B.14 Average values of different features for selected categories on all  
devices . . . . . 239

B.15 Average values of different features for selected categories on phones 240

B.16 Average values of different features for selected categories on tablets 240

# List of Tables

3.1	Variables used in mathematical model . . . . .	71
3.2	User devices . . . . .	78
3.3	Base values . . . . .	79
3.4	Android download distribution . . . . .	80
3.5	Optimal weights . . . . .	90
3.6	User 1 collected data . . . . .	90
3.7	User 1 Google Play scores . . . . .	91
3.8	User 2 scores . . . . .	91
3.9	User 3 scores . . . . .	91
3.10	User 4 scores . . . . .	92
3.11	Correlation with Google Play . . . . .	92
3.12	User 2 scores . . . . .	93
3.13	User 3 scores . . . . .	94
3.14	User 4 scores . . . . .	94
3.15	Correlation with user ratings . . . . .	94
3.16	Suggested values for gains, costs, and losses . . . . .	96
3.17	Convergence rates . . . . .	100
3.18	Malicious and normal app event rates . . . . .	104



4.1	Error rate of competence-detection approach compared with approaches in the literature . . . . .	136
5.1	CrowdApp v2.0 performance specs . . . . .	169
6.1	Computed utility scores using CrowdApp v2.0 . . . . .	181
6.2	Average of traffic-related alarms . . . . .	183
6.3	Average of connection-related alarms . . . . .	183
6.4	CPU-related alarms . . . . .	184
6.5	Memory-related alarms . . . . .	185
6.6	Final app reputations given different aggregation techniques . . .	189

To my grandma.

# Chapter 1

## Introduction

Nowadays, 30 million mobile apps are downloaded every day and this trend is on the rise. These downloads do not include updates, but rather new apps by users [1]. This large variety of apps brought along considerable malware that has infected millions of Android devices throughout the years. Trojans, worms, viruses and spyware have found their way onto user devices through mobile apps [2]. For example, Brain Test is a malware that was masquerading as an Android app that supposedly tests users' IQ. It was available on Google Play until it was discovered in September 2015 [3]. In addition to possible malware infection, a good percentage of these apps have low utility and are generally not desirable by users. This large app count makes it increasingly difficult to keep track of all malicious apps, or simply provide a proper rating of all apps based on the level of user satisfaction.

In terms of malware, the first line of defense was app store vetting. Then came user ratings and reviews which provided a better understanding of the general effectiveness of mobile apps. The problem, however, is that written reviews on app markets are relatively limited and are not always reliable. On the other

hand, numerical ratings are easier to examine, but they do not really describe the shortcomings, if any, within an app. A low app rating could be attributed to a variety of different reasons such as low utility, too many crashes, or high resource consumption. Some users are interested in knowing whether an app is malicious. Others are interested in detecting poor app designs that reduce app utility. Therefore, depending on their expectations, users' provided numerical ratings for apps will be different. For example, if an app deals with monetary transactions, it is critical to have very high security levels. In this case, the user might be indifferent regarding the utility of the app assuming it is highly secure. On the other hand, if the app is a calorie counter, the user would be more interested in its utility since he or she might open it frequently. In this case, data security is of lower importance.

This fine-grained analysis of apps is not found on app stores nowadays. Also, based on our review, no one has attempted to provide a rating of mobile applications that is based on two separate measures which capture security and utility of apps. This was the initial motivation behind our research. Our system is designed to compute two sets of scores for every app on a user device. The first score measures the utility of an app, how easy and practical it is to use. This score is called the objective score since it does not depend on subjective input from the users, but rather on data collected from the devices that are related to the usage of the app itself (time it remains open, crash frequency, etc.). The second score measures the suspiciousness of an app, whether it is accessing sensitive data, sending data online to servers, or designed to consume CPU, bandwidth, or battery resources, etc. This score is called the subjective score since in some cases it depends on subjective input from the users. Human input is central in subjective score computation since the challenge is capturing context and user

perspective. For example, consider the case where an app is using a large amount of bandwidth. This can be considered suspicious but might also be due to user activities (e.g. watching a video). This “context” is not possible to capture without user input. It is this input that we are crowdsourcing in order to transform subjective input into a subject-independent score.

The concept of crowdsourcing is becoming common practice and it is an active research topic. The term was coined in 2005 by Jeff Howe and Mark Robinson from Wired magazine [4]. Simply stated, crowdsourcing is the process of enlisting the services of a crowd of people with the aim of reaching a solution to a proposed problem. In other words, it is a way of utilizing the unused processing power of human brains or skills. Its applications are endless. Some examples are Amazon’s Mechanical Turk (MTurk), an online market place for getting tasks done by workers [5], iStockPhoto that crowdsources stock photographs from amateurs and sells them at very competitive prices [6], InnoCentive that provides a framework for solving research problems [7], TopCoder that offers a platform for developers to build software [8], and GoFundMe that allows people to raise money for events through a crowdfunding platform [9].

Still, despite its many applications, crowdsourcing has not yet entirely penetrated the mobile space. However, it is evident that the wide use of always-connected smartphones will soon expose the full potential of crowdsourcing as a new problem-solving approach. Smartphones enable us to reach a large crowd of contributors, much larger than web-based crowdsourcing applications. In addition, they have plenty of sensing capabilities such as location, light, audio, vibration, etc. These capabilities offer new effective means to passively collect data from devices, resulting in a wide variety of new applications. Our approach is a step in this direction. We introduce crowdsourcing into the mobile space

in an attempt to deliver a system that is capable of objectively evaluating apps based on utility as well as providing a subjective rating that indicates their suspiciousness level.

Even though crowdsourcing is still on the rise and the term was only recently introduced, human beings have been working in crowds since the beginning of civilization. Man is a social being. He dislikes solitude and has always longed for a society beyond that of his own family. His evolutionary success is mainly due to his social nature and his tendency to collaborate with his tribe [10]. Human beings have collaborated to build asylums for the maimed and create vaccines for the sick. They have collectively exerted their skills to ensure the continuation of their species. Their propensity to collaborate is instinctive. Spikins et al. from the University of York discuss how a subtle change in our evolutionary history, thousands of years ago, has allowed individuals with autism to be integrated into society due to the emergence of a collaborative morality [11]. On the other hand, evolution has also brought upon selfish and spiteful behavior, which is manifested through an individual's occasional preference to work alone as opposed to working within a group [12]. This trade-off between collectivism and individualism in crowdsourcing was modeled by Guazzini et al. who show how dividing a population into subgroups influences the ability of these subgroups to solve problems of varying levels of difficulty. They show both computationally and analytically how smaller groups tend to collaborate more intensely whereas larger groups create a niche for free riders who selfishly withdraw from sharing their acquired knowledge [13].

Guazzini's model of collectivism versus individualism assumes one type of crowdsourcing tasks, mainly those that can be solved by an expert in the field. These are collaborative crowdsourcing tasks where consensus is achieved when the

crowd converges to an individual solution. The larger a random crowd, the higher the probability that one individual will provide a high quality solution to the proposed task. Collectivism in this case refers to utilizing the joint intelligence of a subgroup of individuals to develop one solution provided that the contributions of the participating individuals complement rather than possibly conflict with one another. The other category of crowdsourcing tasks includes those with a collective solution which is derived by means of aggregating the input of every individual in the crowd, whether this input is in agreement or disagreement with one another. Our focus in this dissertation is on the second type of crowdsourcing tasks, which are usually referred to as micro-tasks; those with a collective solution. Also, we do not assume a reward factor in our work. Financial rewards are the simplest ways to attract workers [14]. However, once monetary rewards are included, workers have more incentives to cheat to increase their overall pay. Removing the reward factor thereby rids our system of the notion of free riders.

Crowdsourcing tasks with a collective solution are similar in nature to the voting process in electoral systems. As mentioned by Conitzer et al., voting is a method of preference aggregation over a set of alternatives that can range from potential presidents to a ranked list of popular songs [15]. Every vote in a voting scheme corresponds to a noisy perception of the correct outcome. The aggregation technique employed by the voting platform should be carefully crafted to find a compromise candidate that maximizes the voters' combined well-being while inferring their reliability based on their noisy votes [16]. A good aggregation technique manages to extract a hidden objective ground truth that is external to human judgment given many problems such as varying expertise and task difficulty levels [17]. It can intelligently process crowdsourced data with the goal of returning maximum benefit to the crowd [18]. We focus on four of the most

popular voting methods. The difference among these methods is in the utilized aggregation technique. We begin with the simplest aggregation method which is based on plurality voting where the most voted for choice by the crowd is selected as the final solution to a task. Besides its simplicity, another advantage of plurality voting is its elimination of random and incompetent replies from users much like its exclusion of extremist party representation in electoral systems. However, this exclusion is a two-edged sword. Plurality voting systems suffer from the tyranny of the majority. They have strong disincentive to the emergence of a new party or idea. This means that experts in the field, if not in majority, will not have decisive influence on the final output of the crowdsourcing process [19]. To alleviate the disadvantage of unfair representation, weighted algorithms were introduced. One such example is the competence-weighted approach where more weight is given to replies from individuals with higher capabilities than others in the crowd. Giving more “competent” individuals an advantage in voting goes back to the first form of “democracy” in Athens around the sixth century B.C. [20]. Even though participatory democracy was practiced to broaden participation, only citizens, as defined by the state, had the right to vote. This naturally meant excluding women, slaves, convicts, foreigners, and children. It was commonly known that the Athenian assembly must be a sample of the citizenship. However, this sample was not even a random one [21]. Only individuals who were deemed worthy or competent enough were allowed to vote. Further evidence was given by Mogens Hansen who calculated the seating capacity of the Pnyx and estimated the space taken by an adult Athenian male to conclude that the Pnyx was able to hold at most 6,000 people which is not even close to a fair representation as defined by our modern standards of democracy [22].

The main limitation in competence-weighted approaches is the accuracy of the



competence detection method that is employed in the system. In many crowdsourcing frameworks, the competence or expertise of the participants is based on a detection algorithm whose results may not be accurate as it provides in most cases a mere estimate of the actual competence of the participants, which may never be known. In some cases, the competence of a participant is based on the agreement of his reply with the “correct” reply, which raises the question of detection accuracy in cases where the ground truth is not known a priori. This variation between the estimated and true competence of participants is largely responsible for the success or failure of competence-weighted approaches in crowdsourcing systems. To this end, another weighting approach surfaced, which is based on ground truth information collected from the participants themselves. Confidence-weighted aggregation techniques are very popular and are in some cases considered an adjusted approach to plurality voting. The idea is to give higher weights to the answers of participants who are more confident and lower weights otherwise. Another variation is to apply plurality voting on a subset of participants with maximum reported confidence. We argue that introducing confidence into the aggregation method should be studied more carefully. In most cases, people tend to overestimate or underestimate their mental capabilities and this might affect their perception of their own knowledge when answering a question or solving a task. This psychological phenomenon was in fact introduced in a very famous study by David Dunning and Justin Kruger in 1999 [23]. Modeling this psychological bias and incorporating the result into confidence-related aggregation methods is a major part of this dissertation. In a sense, we can draw an analogy between the error in competence detection in competence-weighted approaches and the psychological bias in assessing one’s own abilities in confidence-weighted approaches. The success or failure of each approach in

giving the correct answer depends on the error or bias size. According to [24], there are over 100 identified biases in human decision-making. If crowdsourcing systems do not begin to take these biases into consideration, results from these systems are bound to be sub-optimal.

Our work in modelling the Dunning-Kruger effect and incorporating this model in the evaluation of different aggregation methods is a step in the right direction. Our application of choice for verifying these developed models is the discussed composite app rating system for enhanced mobile security. We implemented and ran this system for a period of six months during which we collected data related to app usage trends and queried users regarding their usage behaviors. In terms of the described objective measure, we computed scores for all apps based on usage patterns and resource consumption. As for the subjective measure, we monitored abnormalities on devices and referred to users when needed. We tested the subjective score computation with each of the different aggregation techniques and compared our system's performance in each case while relating the results to the derived models.

## 1.1 Contributions

Our contributions in this dissertation can be summarized as follows:

- Designing and implementing an app-rating system that can provide users with a composite app score that is based on both an objective measure (utility of an app) and a subjective measure (maliciousness level). Our system is based on a game-theoretic approach.
- Modelling the Dunning-Kruger psychological bias that affects crowdsourcing-based systems where people of low ability have illusory superiority and

mistakenly assess their cognitive ability as greater than it is.

- Modelling and comparing the performance of some of the most popular aggregation techniques in crowdsourcing systems such as plurality voting, confidence-weighted voting, maximum confidence voting, and a newly derived competence-weighted approach. The model takes into consideration the effect of the demonstrated Dunning-Kruger bias on output performance and is applicable to any crowdsourcing system that employs one of the mentioned aggregation approaches.
- Studying general trends in app and device usage on both mobile phones and tablets at different times of the day and different days of the week.

## 1.2 Dissertation Structure

The rest of the dissertation is organized as follows: In Chapter 2, we present a literature survey on several of the topics in this document. We discuss types of crowdsourcing applications and services, quality assurance techniques in crowdsourcing systems, different cognitive biases, and so on. Then in Chapter 3, we describe our first goal in this work which is to model the interaction between our system and apps on user devices using concepts from game theory so as to maximize our system's profit regardless of the action chosen by the app in question. We also present a mathematical model of the objective and subjective score computation. In Chapter 4, we study four of the most basic and popular aggregation techniques that are used in crowdsourcing systems. We also shed light on one of the most well-known psychological biases related to human cognition and that is the Dunning-Kruger effect. We describe how this bias can affect the

performance of certain aggregation techniques in crowdsourcing systems and we present theorems to back our claim. In Chapter 5, the design of our crowdsourcing mobile app and the back-end server is described in detail. This is followed by a description of the experiment process and the results that we collected over a period of six months in Chapter 6. Finally, we present our conclusions and shed light on future directions in Chapter 7.

# Chapter 2

## Literature Survey

We divide our literature survey into several subsections. We begin with a study of the different crowdsourcing applications in the literature and the different task types in crowdsourcing systems on the market nowadays. We discuss crowdsourcing for location, health care, security, intrusion detection systems, and recommender systems. This is followed by a review of quality assurance methods in the literature along with some of the work already done in the area of cognitive biases and human factors in crowdsourcing. Finally, we give a comparative analysis between our approach and discussed approaches.

### 2.1 Crowdsourcing Applications

The classification of crowdsourcing apps based on passive and active contribution is part of the taxonomy given by the authors in [25] in the field of mobile crowdsourcing. They discuss the difference between participatory crowdsourcing where users are actively participating in the process of data collection, and opportunistic crowdsourcing where the tasks are transparent to users and the apps

are usually running in the background collecting data from mobile sensors such as location, camera, audio, vibration, and so on. They further classify applications based on human skills in the case of participatory involvement and incentives be it monetary, ethical, entertainment, service, etc. An example of a passive crowdsourcing scheme is the popular app Duolingo [26] which provides users with a free language-learning service. While users practice new languages, their feedback is used to create a translation engine for these languages. The app is in the form of a game where users collect points every time they properly translate a sentence. When the creators came up with the idea at first, their vision was that if one million users use the app, the entire Spanish Wikipedia could be translated in 80 hours. Today, there are more than 10 million downloads of the app. The developers provided the users with a free service, and without knowing it, the users' feedback resulted in a translation engine for many languages. DuoLingo is an example of a successful passive crowdsourcing scheme that managed to accomplish what it was designed for. Users are actively providing information, yet passively contributing to a larger cause. LogicCrowd, on the other hand, is an active crowdsourcing approach that integrates logic programming into crowdsourcing middle-ware to offer a declarative programming paradigm for improving the knowledge of people through crowdsourcing [27]. It was designed with the intention to give recommendations to users based on datasets, rank the quality of products, and get feedback from the crowd. It combines the power of the crowd in social media networks with conventional machine computation. A prototype of LogicCrowd was executed on an Android platform. It is composed of a Prolog system that contains an interpreter and a knowledge base for user profiles, a social API to interact with the social media network, and a mediator between Prolog and the API to register and execute queries and handle results from the crowd.

Three scenarios were studied in [27]. In the first one, a user is looking for the best Thai restaurant in a chosen location. In the second scenario, a user is asking the crowd to rank handbag brands based on their popularity. And in the third scenario, a user is requesting the name of a place in a posted image. The three tasks were posted on Facebook and results from the crowd were returned to the main program after the expiration time that was set by the users. The prototype allowed programmers to create their rules through an event-driven approach that is combined with social media networks for the use of crowdsourced data from within logic programs.

### **2.1.1 Crowdsourcing Location**

Several crowdsourcing applications have been designed to address the issue of location. An example is SmartTrace+ [28], a crowdsourcing app that asks smartphone users to classify given trajectories based on their popularities. This app proves to be useful, for example, when determining bus routes that are usually chosen based on the number of users that take the routes at a certain time of day. Users passively participate in the service without any privacy concerns. A similar location-based passive crowdsourcing app is Crowdcast proposed in [29]. This app allows its users to continuously find their real-time k-nearest neighbors. The obtained information can be used to save people's lives in life-threatening situations by sending SOS signals to them. It can also be used to improve the efficiency of public emergency services. SmartP2P is another app that uses users' locations to provide them with a search service [30]. When a user issues a query to search for a location, possibly through an image, SmartP2P sends the query to another user who is currently living in the indicated location, rather than sending it to someone who lives far away from it, thus increasing the probability of

getting directions to the specified location. In addition, if two users live in the indicated location, the app will choose to query the one who is spatially closer to the querying user in order to save on resources. PotHole is a crowdsourcing app that helps users to identify holes in streets by collecting vibration and location data from mobile sensors [31]. Waze [32] and Vtrack [33] are two crowdsourcing apps that handle traffic monitoring and delay estimation. CrowdOut [34] is an Android crowdsourcing application that allows users to report traffic offenses in their cities with the aim of improving road safety. The app benefits citizens who will be able to identify dangerous locations near them and adjust their trajectories accordingly. It can also be used by urban designers to enhance infrastructures and improve city roads. An offense reported by a user will be shared to the community in real time. The user can specify the offense type, take a picture, and add a short description of the violation. CrowdOut uses GPS coordinates of users to indicate the events on a map. The Android app displays a non-exhaustive list of icons that represent possible infractions as witnessed by citizens (traffic light problem, damaged road, illegal parking, etc.). In addition to the type of infraction, every time an event is reported by a citizen, other information such as date, time, user ID, photo, etc. is sent to the server. On the sever side, a histogram shows which infractions are the most frequent and where they are located in order for admins and urban planners to properly deal with them. Some experiments were conducted in the urban community of Grand Nancy in France and CrowdOut was found beneficial to users and offered satisfying results. However, some issues need to be addressed such as user privacy and anonymity, reliability of user reporting, and the scalability of the provided service. The location of lost objects was also addressed in SecureFind [35]. The loss and recovery of objects around the world is significant. An object can refer to anything valuable or a



person. It could be a phone, a child, an elderly, etc. The most common recovery method for lost objects is the lost-and-found service. Aside from the fact that it is a slow process that is not practical, most lost objects are not found or turned in and sometimes the object owner does not know in which lost-and-found place to look. With the very low cost of Bluetooth tags especially compared to the value of the lost object, their ability to communicate with mobile devices, and their long communication range and battery life, they can be very useful in revolutionizing the lost-and-found service. When an owner loses an object, he can use his mobile device to search for it. When the object tag is queried, it can report back its location. There are commercial products that use Bluetooth tags to find lost objects but their main drawback is that the searching device must be close enough to the lost object to be able to locate it. This proves to be useless in most scenarios. SecureFind offered a solution to this limited range problem by incorporating crowdsourcing into its functionality. Such a solution is feasible due to the huge number of mobile devices in the world. When an object is lost, its owner can issue a request in the form of a tag query to the proposed service, which will in turn, forward the query to a set of mobile devices. Every device then broadcasts the query until the tag on the lost object responds with its location, which gets sent back to the object owner. Every mobile detector can be slightly rewarded and the object owner will pay a small amount, but in return, will get back the location of his lost object with very high probability. To provide object security in SecureFind, some mobile detectors issue dummy tags that are indistinguishable from the real tag response so that only the object owner can identify it. Moreover, to ensure location privacy, only the object owner is informed of the location under a dynamic pseudonym. SecureFind depends on a framed slotted ALOHA protocol which is assumed to support Bluetooth tags. The basic scheme

will guarantee that the owner gets an approximate location as long as there is at least one mobile detector in the range of the object. Simulations showed that SecureFind was efficiently locating lost objects while protecting their security as well as the privacy of the mobile detectors.

### **2.1.2 Crowdsourcing for Network Mapping**

In addition to location, crowdsourcing can be used to collect other types of information. Portolan in [36] is a crowdsourcing-based system that uses smartphones as mobile measuring elements. It is able to build signal coverage maps and generate a graph of the Internet at the AS level. Measuring large networks has many challenges. Due to their size, collecting information about networks requires a lot of effort and resources and is beyond the possibilities of a single institution. A possible solution is crowdsourcing using many devices scattered over a large geographic area. End users who install Portolan collect local measures and the aggregated results are assembled to build a detailed map of the network. In order to achieve scalability, there are proxies in Portolan. Every proxy receives micro-tasks from the server as well as information from smartphones (location coordinates, IP address, sensing capabilities, etc.) based on which it assigns the suitable devices for every micro-task. The authors tested the effectiveness of Portolan. Their results showed that if a user changes his position by only a few dozen meters, the signal quality might change significantly. This can be very beneficial to network operators who can then fine-tune their infrastructure. The results also showed how the RSS changes with different operators. This can be useful for users of the app who will then choose their operator based on where they spend most of their time. At the AS abstraction level, Portolan's performance was shown to exceed that of CAIDA [37]. Another work that tackles mobile-based crowd-

sourcing of network properties was introduced in [38]. The measuring technique introduced follows a bottom-up approach by using smartphones as monitoring nodes in order to obtain maps through crowdsourcing. The authors envisioned a system whereby mobile nodes inject a tiny quantity of traffic into the network which represents small-range measurements out of which the final Internet graph is obtained. On the telecom level, Nokia Siemens Networks in cooperation with Ciqua announced four years ago that they are working on the Mobile Quality Analyzer [39], a client that users download to their devices. It continuously checks network connectivity and sends feedback to operators who will then have a real-time overview of the network status. The client also periodically asks the user some questions related to the quality of service.

### **2.1.3 Crowdsourcing Health Care**

The area of health care has its share of crowdsourcing applications as well. The authors in [40] demonstrate how crowdsourcing can be used to distribute health care support for patients by creating a service for prospective donors to contribute. They discuss some of the main issues faced by the health care system in the Philippines, mainly distribution, availability, and accessibility. They argue that a crowdsourcing platform can be used to connect between people in need of proper health care and people who are willing to donate to help others. They point out two basic mechanisms that can be used in a crowdfunding system which are return rule and direct donations, how each one works, and in which projects it should be applied. They present some ways in which the crowd can be motivated to donate to a crowdfunding project, for example, advertising causes on social media, emphasizing the importance of helping others and the satisfaction of being part of a group. In terms of donations, they analyze donor behavior and

how the social status of people tends to affect their donation preferences. They emphasize the importance of a clear donation management scheme for proper resource allocation and the importance of micro-donations where small amounts add up and make a difference. In terms of security, they argue that in a crowdsourcing platform, privacy and confidentiality must be protected just like in any patient-doctor relationship. However, sometimes there is a tradeoff between this security and the number of donors or amount of donations. All of these crowdsourcing design problems were addressed in WeSave where the crowdsourcer is a group of social workers who search for donations from the crowd which is a group of prospective donors. Their ideas to attract donors include promoting causes on social media and emphasizing the importance of communal unity and cooperation. In WeSave, beneficiaries are assessed by social workers in order to receive proper care. As for donors, they have the option to register as individuals or groups, or even donate without registration. When it comes to monetary donations, PayPal, or any other electronic money transfer system can be used. WeSave also allows general donations that are not for a specific cause. These donations might be given to causes that still do not have donors or they can be used to help maintain and develop the service. The service supports the use of passwords and certificates to assure donors that it is a legitimate crowdsourcing service. WeSave suffers from some drawbacks such as possible security breaches, the possibility of unfunded campaigns, and the abuse of the system by users. The authors suggest possible mitigation techniques for each drawback. Crowd-Help is another health care crowdsourcing application presented by the authors in [41]. It provides real-time patient assessment through mobile triaging. The application can be installed on computers, as well as smartphones and tablets. It collects information related to the medical condition of the user along with other

data emanating from the device’s sensors. Based on the collected data, CrowdHelp aids emergency personnel in responding to users who are in need of help. Emergency responses are based on a medical triage scheme which categorizes users based on the severity of their conditions and divides resources accordingly. CrowdHelp can be used by several users when reporting an event (natural disaster, terrorist attack, etc.) or by a single user to give information related to his medical condition (symptoms, injured areas, etc.). It provides users with causes for their reported symptoms along with a list of possible locations that are capable of treating these symptoms. This information is stored as part of a user’s profile on the server side where data analysis and machine learning are used to cluster all user inputs based on medical urgency, physical proximity to dangerous events, or proximity to neighboring entries. Security is achieved by separating users into guests, operators, and administrators who are the only ones capable of performing advanced managerial functions. Numerous tests of different sizes were performed, all of which displayed a clear mapping of users’ reported entries, thus leading to a fast understanding and visualization of large amounts of data by emergency experts.

## **2.2 Different Crowdsourcing Task Types**

In our model, we only consider crowdsourcing tasks with a collective solution derived by means of aggregating the input of all individuals in the crowd. These tasks are similar to voting schemes where a relatively large number of workers have to participate simultaneously. Also, they are not open-ended in nature but rather have a definite answer that a group of individuals can agree on. It is worth noting, however, that there are other crowdsourcing tasks that do not fall under

the model presented in this work but are worthy of mention. One example is the classical ROVER algorithm proposed in [42]. The NIST Recognizer Output Voting Error Reduction system combines the output generated by multiple Automatic Speech Recognition systems resulting in a lower error rate than any of the individual systems. First, the outputs from multiple ASR systems are aligned to generate a single word transition network. Then a voting scheme is applied to select the highest scoring word. Three benchmark evaluation submissions were used to test ROVER. Each output word was provided along with a confidence score ranging from 0 to 1. The authors investigated three voting schemes: majority, confidence-weighted, and maximum confidence. The best error reduction was achieved with the maximum confidence approach followed by the confidence-weighted approach and finally the majority approach. Of course this does not agree with the results in our work. ROVER combines system outputs of multiple recognition systems. The difference between ROVER’s voting scheme and that in crowdsourcing systems is that in the latter, the miscalibration in self-assessment that leads to inaccurate confidence scores is a result of human factors and it renders the majority score superior to confidence-weighted scores. The human factor does not apply in the ROVER case.

The authors in [43] use the ROVER tool to address the problem of cross-language transfer of domain-specific semantic annotation. There are several issues to address. The language of interest might be under-represented. This was addressed by using targeted crowdsourcing. Also, domain-specificity of the required annotation increases complexity of the task. This was coped with via priming with a list of source language concepts. Another issue is the evaluation of crowd-annotated data without reference target language annotation which was coped with by applying inter-annotator agreement. It considers both segmenta-

tion and labeling agreement measures meaning that annotators have to agree on both the label and its span. The authors apply the ROVER technique with a majority voting scheme to decide on the label and its span. Their results show acceptable annotation quality. In a more recent work, they study selection and aggregation techniques for this semantic annotation task [44]. Their goal is to select a word mapping that is closest to the source language or to aggregate all mappings into a single one that best represents the meaning of an utterance. The baseline for selection is randomly picking one of the mappings whereas that for aggregation is using majority voting while randomly breaking ties. Both these approaches are not a good choice considering the varying expertise levels of crowd annotators. Language Models (LM) based on the maximum likelihood were used to estimate the reliability of crowd annotators. Their results show that majority-voted ROVER provides a strong baseline. However, weighing each hypothesis with respect to other annotations proves to be the best weighing scheme with an increase of 0.4 in the F-measure. The authors' baseline evaluation in [45] is the random re-sampling approach where the precision of one randomly selected judgment is computed and results are averaged after repeating the procedure 1000 times. Their results show that the performance of majority voting is higher than the baseline thus proving how combining the "power of the crowd" with computational methods improves annotation quality.

In addition to what was mentioned, there are crowdsourcing tasks that have one solution only and these are modeled differently than tasks that aggregate worker solutions. In these types of tasks, it is up to the requester to determine the winning solution based on the amount of money he is willing to pay and the minimum required solution quality. In addition, there are crowdsourcing tasks with open-ended questions. For example, Wikipedia [46] is a crowdsourced

encyclopedia. Several contributors can participate in creating Wiki pages and there is no one solution or output format for the pages. These types of tasks are modeled differently and are beyond the scope of this dissertation.

## 2.3 Crowdsourcing for Security

The main issue in crowdsourcing is how we are expected to trust data contributed by a crowd of anonymous users. The authors in [47] give several examples of challenges in crowdsourcing systems. For example, self-policing is not applicable in many crowdsourcing services such as social media where news is instantly acted on. Made-up stories still make it to the media via dishonest journalists. Anonymity in several crowdsourcing systems can result in Sybil attacks. These are only a few examples of the issues faced in such systems. There are possible solutions to these issues; some are already implemented, while others are approaches that are worth looking into. For example, statistical analysis can be applied to a data stream to separate the good data from the bad. However, this technique's effectiveness depends on the type of collected data and the kind of service collecting it. The authors give examples of other approaches along with their limitations. In [48], the authors discuss the main challenges in mobile crowdsourcing networks (MCNs) where human involvement offers unprecedented sensing and transmission opportunities. However, security and privacy issues are more critical in MCNs than in traditional networks due to several considerations. First, due to the fact that humans are involved in crowdsourcing sensing and computing, there is a risk of exposing sensitive information on the devices or even private information related to the user himself. In addition, the device could be controlled by a malicious human to launch an attack. The second con-



sideration is task crowdsourcing, which raises a big security question since tasks themselves might contain sensitive information. In addition, MCNs have a dynamic topology since new users might join or others might leave the network and since users choose tasks based on their device capabilities and their interests and they are mobile most of the time. All of this makes addressing security and privacy concerns even more challenging. There is also the fact that these networks can be heterogeneous in nature (WLANs, Bluetooth, WiFi, VANETs, etc.) and mobile devices are heterogeneous as well. This complicates the process of securing MCNs. And if there is no guarantee of security and privacy in MCNs, users will have no incentive to participate. All these threats were summarized by the authors into privacy, reliability, and availability threats. To deal with reliability and availability threats, crowdsourcing participants must be authenticated before they join the MCN. A threshold-based distributed authentication mechanism is desirable where group authentication of a new mobile device is required. To protect the task privacy of end users and the subscription privacy of mobile users, attribute-based encryption can be used where the ciphertext can be decrypted only with specific attributes of the decryptor. As for computing results, a trusted third party can be assigned to verify the integrity of these results. The authors formulated many other research problems that might lead to future research directions in the area of security and privacy in MCNs.

Crowdsourcing in itself can be used to improve the security of systems. Tal Eisner from cVidya talks about crowdsourcing in telecom fraud and security management through a mobile app for Android called FraudView CyberHub [49]. Its purpose is to detect, block, and report premium rate fraud numbers that are often the result of malware and malicious app infection to the mobile device. The crowd intelligence is supplied by the mobile users themselves who report actual

and suspected malicious numbers and apps. Their reports go to a cloud-based server in which automatic algorithms backed up by experts analyze reports and distribute the info to all FraudView CyberHub users. They completed most of the research needed to launch it and many operators are intrigued with the idea including AT&T and Telstra. Peer production was combined with crowdsourcing in [50] in order to improve network security. Instead of relying on the software adoption of individual users, the authors argue that a better approach would be to focus on the community as a whole. The premise was that peer production could address incentives such that participating individuals value contribution and hence improve the level of security for the entire community. As a proof of their concept, they provided a high-level instantiation on mitigating the insider threat. They presented the system design conditions and provided an example that still needs to be tested. Crowdroid [51] is a downloadable client for Android devices that uses collected information from a crowd to detect malware. The framework relies on three components. The first is data acquisition where Crowdroid uses Strace to monitor applications on users' devices. The second is data manipulation where information from Strace is collected, analyzed, and system call traces are processed to produce the feature vectors that will be used for clustering. The third is malware analysis and detection where the feature vectors are clustered using k-means clustering in order to create the normality model and detect anomalous behavior in apps. The authors mainly targeted Trojan applications. They tested Crowdroid on self-written malware and real malware. For the real malware, they used normal and infected versions of the Steamy Window and Monkey Jump 2 apps. For the first app, they got a detection rate of 100% and for the second 85%. The authors in [52] argue that there are thousands of mobile apps that are being downloaded on a daily basis and most of them use

sensitive information such as addresses or credit card numbers. According to them, previous methods were able to detect that sensitive information was being used, but could not tell whether this use of information was legitimate. That is why human expert intervention is required. However, with the large number of apps in markets nowadays, manual inspection is no longer feasible. Hence, they propose AppScanner, which builds on crowdsourcing, automation and virtualization to enable large-scale analysis of mobile apps. It provides end-users with more understandable information regarding what mobile apps are really doing on their devices. It relies on the use of automation and traditional security techniques to learn application behavior with fine granularity.

Gander et al. argue that in recent years, the spreading of malware has changed from traditional channels such as E-mail to a new channel which is websites [53]. They propose Croft, a monitoring tool that gathers data related to users and visited websites in a crowdsourcing fashion. Their goal is to decrease the risk of malware infection while increasing the awareness of users on the Internet. Users who install Croft are fully aware of how the monitoring method works. In addition, their data is not made available to third parties. This is ensured by applying end-to-end encryption on the data. Croft was designed with three goals in mind: studying the probability model of users getting infected, studying the spreading characteristics of malware, and studying the performance of antivirus software. It is constantly running in the background. When a user visits a website, the tuple (URL, timestamp) is stored in Croft. Meanwhile, the antivirus software is trying to detect if the URL is malicious. If it is not, then after a certain threshold, the tuple is replaced with new entries. However, if the activity is malicious, then the antivirus will detect a malware. In that case, data is forwarded from Croft to the back-end which includes a timestamp, the list of

URLs from open tabs, a keyword check (e.g. AVG Malware Alert), the OS configuration, and the user's information. This data is encoded as JSON and the transmission channel uses SSL/TLS. A MySQL database is used and all the information on the database is moved twice every day to a local storage that is not connected to the Internet and the database is wiped. This will add extra protection to the users' collected information. Croft is implemented using C# and the .NET framework since it targets windows users. Its components are Croft client and Croft Web application. After the data is sent by Croft, the next step is false-positive reduction. This is still a work in progress. However, expert information in the form of manual analysis may be used to detect these Type I errors. After false-positive reduction, three forms of statistical analyses are performed. In malware statistics, malicious URLs are ranked based on several categories (number and type of malware, rate of infection, etc.). User statistics include information related to user status and behavior (browsing profiles, number of infected users, etc.). In antivirus statistics, information will mostly relate to the performance of different antivirus software. The authors performed an experiment in cooperation with a European antivirus product. Twenty-nine users were asked to volunteer in the experiment with the incentive of receiving free licenses of the product. The experiment was conducted over a period of one month. Evaluation showed that Croft works as expected. In addition, feedback from the users showed that they trust that the system cares for the privacy and confidentiality of their submitted information, and that they believe in the usability and reliability of the tool. In [54], the authors discuss the problem with worms and how easily targets can get infected. They propose crowdsourcing as a possible solution to identify attacks and act accordingly. Internet worms have had an intense impact over the last few years and the currently offered

solutions lack in several areas such as scalability and flexibility in handling new threats. To solve this problem, the authors developed NetBuckler, a Microsoft windows application based on the JXTA framework that employs the collective intelligence of a crowd to detect and avoid worms by monitoring the change in traffic in a P2P network. The designed application builds visual graphs to identify variations in traffic and provides security measures to protect the host it is installed on. These variations in traffic rates are collected locally by making decisions based on two threshold values. If the average rate is below the lower threshold, the security level is decreased. If it is anywhere between the lower and upper thresholds, no action is taken. And if it is above the upper threshold, the security level is increased. The resulting information is exchanged between peers in the network. NetBuckler is based on Java programming language which means it can be easily modified to make it compatible with operating systems other than Microsoft. The authors simulated attacks to test their application. These attacks triggered the response of the system. However, the experiments do not result in False Positives which means that a more advanced testing approach is needed. Although NetBuckler tackles many drawbacks in existing IDSs such as efficiency, scalability, and single point of failure, it suffers from drawbacks itself. The authors plan to build a secure environment that provides authentication, non-repudiation, privacy, integrity, and solves the peer dishonesty problem.

The above was an example on crowdsourcing with intrusion detection systems which is currently an active research topic. In what follows, we present a sample of the work on crowdsourcing-based game-theoretic IDSs.

## 2.4 Crowdsourcing-Based IDS with Game Theory

Most of the work in the area of network security focuses on improving intrusion prevention and detection techniques. Intrusion response remains a manual process performed by an administrator in response to an alert from a detection system. This manual process introduces a delay during which an attacker can cause serious damage to the system. This raises the need for an automated intrusion response scheme and it was the motivation behind the authors' Response and Recovery Engine (RRE) in [55]. RRE models the interactions between itself and an attacker as a two-player nonzero-sum sequential Stackelberg stochastic game where RRE is the leader and the attacker is the follower. The authors introduce the attack-response tree (ART) structure to describe the security of a system based on possible intrusions for the attacker and responses for the engine. RRE automatically transforms ARTs into observable competitive Markov decision processes (POCMDP). Once solved, the optimal response against an attacker can be found such that the damage caused to the system later in the game is minimized. RRE has a two-layer architecture composed of local and global engines. Local engines are found on host computers whereas the global engine is on the RRE server. When the system is not recoverable by the local engines, the global engine selects an appropriate global action. This architecture enhances the scalability and performance of RRE so that it can be applied in large-scale networks. The authors implemented RRE on top of Snort 2.7 [56]. Their results showed that for large-scale networks with 330,000 hosts, RRE was able to generate the Markov model within 24 ms only and it was able to generate optimal actions for ARTs with more than 900 nodes in less than 40 seconds which means that it can protect

large-scale computer networks.

In [57], the authors study the security of mobile AdHoc networks. Due to the characteristics of MANETs such as frequent changes in network topology along with many system constraints, they are inherently vulnerable to security attacks. In addition, these networks have no centralized management unit, no certification authority, and connectivity within them is intermittent. All of these reasons make it difficult to secure MANETs and motivate the need for an IDS. A node in the network with an IDS deployed on it will constantly monitor and evaluate neighboring nodes, based on which it will cooperate with neighbors that it trusts and dismiss any requests from suspicious neighbors. The authors use game theory to model the interaction between a node in a MANET and its neighbor. The aim of their work is to provide optimal strategies for both malicious and regular nodes using a game-theoretic approach. They model their problem as a dynamic two-player non-cooperative game with incomplete information. In an interaction between a node and its neighbor, the node does not know the type of its neighbor which could be regular or malicious. Nonetheless, it has a belief regarding the type. Nodes in the network are divided into clusters. Every cluster will elect a leader using some clustering algorithm. The leader uses a hierarchical IDS whereas the cluster members use a standalone IDS. There are three modules in the IDS deployed on regular nodes: The monitoring unit, the decision-making unit, and the secure communication unit. The monitoring unit uses game theory to compute thresholds that will be used to update the belief of a neighbor's type. If the belief is that the neighbor is regular, the packet is accepted and sent to the secure communication unit where it is encrypted before being forwarded. Otherwise, the node will accept its neighbor with a certain probability. If an intrusion is detected, the decision-making unit will request the cluster head to

further analyze the data. The cluster head will then consult all the cluster heads. If more than half of them agree that the node is malicious, an alert will be raised. Otherwise, the neighboring node will be trusted. The difference between a cluster member and a cluster head is that the former uses an independent decision making system while the latter uses a collaborative decision making system by forwarding messages to other nodes and waiting for their replies. Payoffs are awarded based on the actions chosen by the two players. The authors analyzed the Bayesian Nash equilibrium of the game and computed the probabilities of both a node to accept its neighbor and an attacker to attack that are required to maintain this equilibrium. They conducted simulations on a randomly generated MANET with 100 nodes in a  $1500\text{m} \times 1500\text{m}$  region that is evenly divided into 15 clusters. They studied three parameters: The percentage of detected attacks, the percentage of false alerts, and the throughput. As the number of malicious nodes in the network increased, both the throughput and percentage of detected attacks decreased while the percentage of false alerts increased. Their proposed method was found to perform better than existing methods in the case of throughput and attacks detected, but slightly worse in the case of false alerts.

In [58], the authors study the trade off between the level of security enforced by an IDS and system performance. Network assets as well as the overall performance of an IDS should be optimized. This was addressed by the authors using game theory. They modeled the problem as a stochastic multi-player non-cooperative non-zero-sum game and demonstrated the existence of a stationary Nash equilibrium. In the basic formulation of the game, the network is composed of defending machines connected by a graph, and malicious attackers. Every machine can be in a finite number of states and can have a finite number of detection libraries. On the other hand, every attacker can be in a finite number of states



which reflect the energy level of the attacker. It can choose its action from a set of actions where each one denotes a different type of attack which causes damage to the machine and costs to the attacker. For every library, there is a scope of detection which comprises the types of attacks that can be detected by it. Also, every machine has a coverage factor which represents the set of attacks that will surely be detected by this machine. In every time instance, the action of the defender is to configure a set of libraries. The action of the attacker is to choose its attacks from its available attack types. The authors extended this model to take into consideration heterogeneous networks. They assumed that every machine protects a network asset and every asset has a security asset value which denotes the importance of the node to the network. Since assets in a network are inter-connected, when one asset is attacked, other assets might be compromised. This is referred to as asset inter-dependency. They categorized the assets in a heterogeneous network based on the CIA triad (confidentiality, integrity, and availability). They also defined an impact factor for each attack type on each asset. A defender will attempt to maximize its utility function which takes into consideration both direct and indirect (incurred by adjacent compromised nodes) damages to a machine. Similarly, a rational attacker will attempt to maximize its utility by maximizing the damage it inflicts on its targets. The authors proved the existence of a stationary Nash equilibrium for their game based on the theorem which states that every non-zero-sum finite discounted stochastic game will have at least one stationary equilibrium point. This equilibrium can be used by defenders to find their optimal library configuration in different states. The tradeoff between security and performance was also studied by the authors in [59] for Host Intrusion Detection Systems (HIDS). They argue that appropriate optimization is required for efficient detection and suggest game theory as a technique

for optimization and efficiency in HIDS. They begin by modeling the network as a graph where the nodes are HIDS (defender nodes) and attacker nodes. They present a three-step proposal. In the first step, the local reputation between every two defending nodes in the network is calculated. This is the direct evaluation of behaviors. A trust algorithm based on the Perron-Frobenius theorem is then applied to compute the global reputation vector. The algorithm keeps on iterating until the difference between two consecutive trust values becomes less than a predefined threshold. This threshold is set by the leader node and can be adapted depending on the precision required by the application. In the second step, the leader is elected. The election is based on two factors: The leader must have a high trust value and the minimum resources required to monitor. The elected leader will calculate global reputations, play the game, and decide when to activate HIDS. In the third step, a two-player non-zero-sum non-cooperative iterated game is played between an attacker and HIDS. The authors give the payoff matrix of the attacker and defender in strategic form. Then they define the utility functions of both players. The solution to their game is a mixed strategy Nash equilibrium where the best response of HIDS is to monitor when the belief that a node is attacking is higher than a computed threshold, and the best response of the attacker is to attack when its belief that HIDS is defending is lower than a computed threshold. The authors simulated their experiments using MATLAB. They studied two scenarios. In the first one, higher priority is placed on trust (e.g. cloud computing). In the second one, higher priority is placed on monitoring and attack costs (e.g. WLAN where defenders have limited resources). Their results showed major improvements in both resource consumption and detection rate when game theory was introduced into their detection system.

Game theory is also used when designing reputation protocols and incentive

mechanisms. The authors in [60] study a typical crowdsourcing system where requesters post tasks with rewards and workers solve these tasks and provide their solutions to the requesters, who in turn select the best solution and grant the rewards to the respective workers. Requesters are motivated to use the system when the provided solutions by the workers are of high quality and when the reward they have to pay is as small as possible. On the other hand, the larger the reward, the higher is the incentive for workers to join and provide good solutions. This tradeoff is one of the challenges when designing reputation and incentive mechanisms. The authors use a probabilistic model to describe the interaction between a worker and a task. They begin by proposing an incentive mechanism to make sure that experts will join the system and will provide high quality solutions. They describe their binary rating system where a requester categorizes every provided solution as satisfactory or not satisfactory. The reward is then divided by the administrator over all the workers whose solutions were satisfactory. If there are no satisfactory solutions, then the requester's reward is divided evenly over all workers. This is done to prevent denial of payment by the requester where he benefits from one of the solutions but provides a false rating. Since it is possible for all workers to earn rewards even when their efforts are low, the authors formulated their incentive model as a game. They showed that when there is only one worker in the system, it is impossible to give him the incentive to provide a high quality solution. The solution to this problem is simple: Always make sure that there are at least two workers for every proposed task. With more than two workers in the system, the authors prove that their incentive mechanism can guarantee that expert workers provide high effort solutions. On the other hand, it was shown that regardless of the reward size, novice workers will always free ride. The authors derived the minimum number

of workers and the corresponding required reward that will ensure that at least one high quality solution provided by an expert is returned to the requester with very high probability. They proposed a reputation system in order to prevent the problem of novice free riding by penalizing workers when their reputation is poor. They showed that their repeated game has a unique sub-game perfect equilibrium where the novices refuse tasks voluntarily and the experts provide high quality solutions. A byproduct of their proposed reputation scheme was that it reduced reward payments for requesters.

## **2.5 Crowdsourcing for Recommender Systems with Expert Detection Techniques**

The authors in [61] study several online recommender systems by analyzing their input parameters, effectiveness, and drawbacks so that they can assess how they can be used in crowdsourcing systems. Their aim was to derive the best practices from existing recommendation systems in order to model an effective system. To this end, they presented a critical review of existing works. The papers they reviewed covered recommender systems and crowdsourcing in these systems. The review describes the methodology as well as the technological aspect of the works. In addition to their provided summaries of every work, they evaluated how it addresses some questions that they formulated (how the problem is solved, how scalable it is, what parameters are used in the problem formulation, what are some limitations of the work, etc.).

In the Android framework, the resources given to an app are isolated from those given to all other apps. Upon installation of an app from the market, a user has to accept all the resource access requests before he is allowed to use the app.

However, it was found that more than 70% of smartphone apps request access to resources that are irrelevant to the main function of the app [62]. In addition, only 3% of users pay attention to these requests and make correct judgments regarding whether or not to grant permission [63]. To this end, the authors in [64] propose RecDroid. A crowdsourcing-based participatory framework gives recommendations to its users on how to control their privacy and respond to these permission requests. RecDroid gives users the option to install an app in two modes. The first is a trusted mode where all requested resources are granted to the app. The second is a probation mode where real-time resource granting is implemented while the apps are running. RecDroid also gives recommendations to the user regarding the type of installation to choose based on previous selections made by experts for this particular app. It starts with a set of seed expert users and returns its initial recommendations based on their responses. The authors propose a spanning algorithm to identify external expert users based on the similarity of their responses to those of the seed experts whose responses are taken as ground truth. RecDroid collects responses to resource requests from users, analyzes them to remove any biases, and infers the security and privacy risk levels of apps based on its collected data. When an app in probation mode requests a resource, RecDroid will provide the user with a recommended response. If the app was covered by seed experts, their response is suggested to the user. Otherwise, an aggregated response from other users based on weighted voting is recommended provided its confidence level is high. Simulations were conducted to measure RecDroid’s accuracy and effectiveness. The authors set up 100 RecDroid user profiles with expertise levels varying between low, medium, and high. They ran their simulations on MATLAB and repeated every experiment 100 times with different random seeds. Their results showed that the accuracy of the system decreases as

the coverage increases which is expected since higher accuracy means less votes will be qualified for recommendation. However, when parameters were carefully set, RecDroid was shown to achieve high accuracy with good coverage. Results also showed that RecDroid only needs a small seed value of experts to bootstrap the system, and it was shown to be effective and feasible when implemented on Android phones.

The authors in [65] study expert detection and motivation techniques in knowledge sharing sites which depend on human expertise. They focus on the case of StackOverflow where participation requires a high level of understanding of the specific domain. The site uses a detailed reputation system that rewards users who ask good questions, give good answers, or rate the quality of other questions and answers. This reputation scheme can be used to easily recognize experts in the system. It proves to be beneficial since there is a need to differentiate between different levels of users and to motivate and reward experts. The authors used data from the StackOverflow website which included around 3 million questions and 7 million answers from over 1 million users. They proposed that early participation can indicate which users might end up giving helpful answers in the end since new users to the system tend to follow different activity patterns which lead to a change in the reputation gain patterns. They plotted the mean cumulative user contribution in several measures from both high and low reputation users. The results agreed with their proposal that the activity of a new user is indicative of his long-term contribution. They performed SVD and PageRank analysis and concluded that they are both useful in detecting extreme cases of influential users. To try to recognize the experts based on their early activity on the site, the authors modeled this issue as a classification problem where users are classified into one of two classes: experts or non-experts. They

compared the performance of their classifier to that of Pal et al. [66]. Their classifier was shown to have lower precision than Pal but higher recall and F-measure. In addition, their classifier results showed that one month of site usage is enough for reliable identification of experts in the system.

In [67], the authors discuss the importance of user modeling in recommender systems, which allows to link users with common interests. Their proposal was to model the network of rating users as a graph that changes with time. New users can join at any time and existing ones can rate new items. The aim behind their work was to find out whether regardless of the network size, it will always reach a stable form and whether there is a pattern in the network. They formulated the recommendation problem using usefulness matrices which have users as rows and rated items as columns. In collaborative filtering, recommendation was formulated using similarity matrices. Since the authors represented their network as a graph, the nodes are the users and an edge between any two users is an indication of enough similarity between them. This means that the degree of a node will represent the number of its similar users. The Hamming distance was used to calculate the similarity between two users. The data set they used was MovieLens. In the first data set, there were 100,000 evaluations on 1,682 movies by 943 users. In the second data set, there were 1,000,000 evaluations on 3,952 movies by 6,040 users. In order to observe the time evolution of the network, the data was split into four time periods. Evaluations of movies were normalized to 0 or 1 to represent approval or disapproval of a movie by a user. Contrary to what they expected, the authors did not discover a power-law. Instead, they discovered a decreasing linear relationship between a node's ranking and degree.

The authors in [68] discuss Question Answering (QA) communities where social interactions between users enable sharing of knowledge in different fields.

In these communities, some users are generally more knowledgeable than others are. This core group of users is referred to as experts who constitute a small percentage of the overall community, but are responsible for a large percentage of the answers. They are the drivers of such communities and it is very important to detect them at an early stage. The authors used a dataset from the TurboTax Live Community (TTLC) from July 2006 until April 2009. This dataset has 83 users that were explicitly categorized by Intuit [69] employees as super-users or experts. This human evaluation process has two main limitations. The first is that humans usually evaluate the expertise of users based on long-time contribution which means that users with high potential that have just joined the community are not taken into consideration. The second is that an evaluation process of this kind generally takes a long time which is a problem since there is a risk of high-potential users choosing to opt out of the community due to the lack of credit for their efforts. These limitations emphasized the importance of designing a tool that is capable of quickly going through thousands of users and recommending high-potential ones to human evaluators. According to the authors, a potential expert must be highly motivated to help others and should have the ability to answer questions correctly. Based on these two properties, the authors came up with a list of user qualities that could be used to identify potential experts. They used both SVM and DTree over these features to try to find the potential experts in their selected dataset. Performance was measured using precision, recall, and f-measure. SVM was shown to have better precision than DTree but a lower recall and f-measure. The lower precision of the DTree could indicate that it recognizes more potential experts. This made it a good method of choice for further analysis. In one of their experiments, their DTree model detected 75 potential experts out of which 40 were actually categorized by Intuit as super-



users. The remaining 35 were then recommended to Intuit and it was found that 27 out of them were almost ready to be promoted to super-user. The purpose behind this early detection of experts is to encourage and mentor them in order to improve the quality of answers in the community. However, due to cost and time constraints, this mentoring cannot be provided to all users. It would be desirable to provide it to the top ranked users only. To this end, the authors proposed a ranking mechanism in order to rank users based on their potential. Their ranking algorithm found several potential experts which were missed by the DTree algorithm. The authors concluded that a combination of the two algorithms is the most effective way to find the potential experts in a QA community. These identified users could be upgraded to an intermediate status to give them an incentive to continue to participate in the community while Intuit employees take their time in evaluating their potential. The case of identifying authoritative users in Yahoo! Answers was studied in [70]. In such a QA community, an asker prefers to get answers from users who are considered experts on the specific subject. That is why, it is very important to be able to identify these authoritative actors. The authors propose to discriminate authoritative and non-authoritative users. They represent the interaction between the asker and the best answerer as a weighted directed graph where the link magnitude between any two users corresponds to how often one of them chooses the other as the best answerer. To provide authority scores for the nodes in the graph, link analysis can be applied. The authors discussed some of the most common link analysis techniques and concluded that PageRank, HITS, and Z-score are all not appropriate for their application. They adopted the simple InDegree technique where the authority of a node is measured based on the sum of the weights of the links that point to it. They propose a probabilistic method based on a mixture model. The

normalized InDegree values of all users in every category were first estimated and after analyzing them, the authors found that they are well fitted by two gamma components, one of which contains large InDegree values and therefore represents authoritative users. They performed experiments on datasets from Yahoo! Answers for six different categories over one year. They first attempted to manually evaluate the answers of the authoritative users that were identified by their approach. They found that these users always provided detailed answers with good quality, and some of them were even teachers, graduate students, engineers, etc. Then, they used the approach proposed in [71] to identify high-quality content. The experiments showed that the average quality score of the identified authoritative users was above 0.7 which shows that their approach significantly results in high-quality content in Yahoo! Answers.

Similar to QA communities, in personalized learning, students benefit from the shared knowledge of users. The aim of personalized learning is to adapt the learning process to the knowledge and preferences of every student. The problem is that traditional personalized learning is not scalable. As the number of students and the variability in their preferences and backgrounds increases, it becomes more difficult to establish effective personalization schemes. That is why the authors in [72] describe a new approach to combine crowdsourcing with social networks to improve personalized learning. They develop SALT (Self-Adaptive Learning through Teaching) where students participate by both learning and teaching through social interactions. SALT was implemented as a large-scale Online Social Network (OSN) similar to Facebook. It achieves its goal of personalized educational knowledge by exploiting the combined intelligence of its users who can take the role of both students and teachers and interact by sharing learning ideas in the form of small lessons (lesslets). Each lesslet has a

name, a small explanation of the concept, an example, and a test to evaluate a student's understanding of the concept explained in the lesslet. When more than one lesslet are used together, the result is a learning pathway that is carried out by a student. Creating lesslets will result in students being more involved in the work and gaining a deeper learning experience. SALT was designed as an adaptive system. It automatically orders lesslets based on their estimated difficulty. It also keeps a profile for every user based on his performance. It recommends pathways to users based on similarities with other users. This adaptive approach gives the most productive pathways and results in a positive experience for users. SALT was implemented as a web application. It is currently available for registration by invitation only. When users enter the system, they can click on a lesslet name which gives them access to the different components of the lesslet. After users learn a lesslet and take the test, they are given a score along with recommended lesslets that they can take. A user also has the option to request a lesslet by posting a wish (similar to QA systems) and when the requested lesslet is created, the user will be notified. Users on SALT can friend each other and participate in online chats. SALT was evaluated in undergraduate and graduate classes where 260 students created around 300 lesslets. Results suggested that users with similar learning patterns can be identified and their intra-group similarities used to achieve personalization. Evaluations also studied how different groups of users follow different pathways and how collaborative filtering algorithms perform in SALT.

Another work that discusses the problem of qualifying workers in a crowdsourcing project was presented in [73]. The aim of the work was to find skilled workers on Amazon's Mechanical Turk. The crowdsourcing task that the authors considered was determining the geolocation of videos on social media. The

difficulty of this task is in selecting the workers who would execute it reliably. Workers were asked to look for traces in the videos that will help them make a decision regarding the location. However, this task depends on the workers' personal judgment which raises the issue of dishonest workers who attempt to fool the system for rewards. In the experiments, a dataset of Creative-Commons licensed Flickr videos was used. The designed web interface had a set of instructions, a progress bar, and an instance of Google Maps for users to geo-locate the videos. After workers tag their locations, the Haversine formula is used to compare the distance between the workers' estimated coordinates and the true ones. In the next phase of the experiment, workers were provided with a tutorial explaining how to find clues and geo-locate videos. After the addition of the tutorial, qualified workers gave better results than both internal testers and machines. The work was summarized with a list of steps to follow to properly qualify workers. These steps include testing with trusted experts, obtaining feedback, and developing and refining tutorials until the results of workers match those of predefined experts.

In [74], the authors discuss Tourist Spot Recommender Systems (TSRS) and the possibility of enriching their recommended locations by providing extra features that will benefit users who are requesting information from the service. Earlier research in TSRS does not focus on conditions that should be updated in real time. The authors suggest gathering updated contextual information about any location from users who are currently in that location for improving the quality of recommendations. This information enrichment can include safety alerts, traffic and weather conditions, crowdedness, construction in progress, etc. Their designed system consists of four parts. The first part is the mobile client which sends the current user location to the TSRS either manually or by GPS and gets

back a list of recommended spots with the extra information. The second part is the TSRS that takes the location of the user and gets a list of recommended spots that are nearest to it. It sends this list to the third part of the system, the crowdsourcing platform, which in turn sends an aggregated result to the user. The fourth part of the system is the crowd resources which is the set of users registered on the system. The authors implemented a prototype of the system with client-server architecture. The crowdsourcing platform was simulated using Android apps. In their experiments, the crowd resource was composed of 76 registered users who were randomly divided into smaller sets with each set located in a popular tourist spot in Delhi. For every user request, responses from volunteers were stored in the platform database. At the end of their experiments, feedback from 29 users of the service was recorded. 19 users labeled the service useful, 6 labeled it very useful, 3 labeled it somewhat useful, and only 1 user considered it not useful.

## **2.6 Quality Assurance in Crowdsourcing Systems**

The benefits of crowdsourcing are realized when a large number of workers participate in solving small tasks. However, these contributing workers may try to cheat the system especially in the presence of monetary reward or they may make mistakes due to personal bias or different experience levels with the subject matter. One approach to detect such workers would be to manually verify the output quality. The only problem is that manually verifying the quality of the submitted results is hard and negates many of the advantages of crowdsourcing. Verifying every submitted solution has the same cost and time as performing the

task itself. We need algorithms that will accurately estimate the quality of the submitted work for maximum benefit.

There is a plethora of quality assurance (QA) techniques designed for crowdsourcing systems that can be categorized along two main dimensions: design-time and runtime approaches [75]. Worker selection based on pre-specified reputation levels or pre-specified credentials are two examples of design-time approaches. The authors in [76] estimate reputations of experts based on link structure and periodically updated trust relations that capture any changes in preferences and maintain skill evolvment. In some cases, domain experts check the contribution quality. The Wikipedia encyclopedia employs an expert review approach for quality control [14].

Dawid and Skene use the expectation maximization algorithm to estimate both the quality of the workers and the correct answers for each task [77]. The authors in [78] argue that the inherent value of a worker cannot be measured from the error rate alone. Workers may be careful to avoid error but their solutions might suffer from bias, the effects of which can be reversed. They expand on the work of Dawid and Skene and present an algorithm that separates this bias of potential high-quality workers from other unrecoverable errors of low-quality workers. Their approach leads to better treatment of workers and allows for better quality estimation.

Amazon's Mechanical Turk is an example of a crowdsourcing marketplace that incorporates several QA methods such as reputation systems, majority consensus, contributor evaluation [79, 80], and ground truth where a small number of tasks with an available gold standard are mixed in with other tasks so as to identify malicious workers who are deliberately attempting to sabotage the system [79, 81]. Requesters on MTurk can also design defensive tasks which are

more difficult to cheat on than properly solve. Another QA method it provides is based on redundancy where each task is performed by several workers and high-quality solutions are identified based on some voting scheme. The redundancy QA approach is also incorporated in the famous reCAPTCHA protection service [82]. In case of any discrepancy among received answers, a word is sent to several other workers and the answer that has the highest votes is selected.

Two popular and somewhat similar approaches to QA are output agreement and input agreement [83]. They are commonly used for task labelling. In output agreement, two or more randomly chosen workers are given the same input and are required to produce output based on this input. Matching output is selected as the winning label. Since workers are chosen randomly, the quality of the output is verified considering that it is based on agreement from two largely independent sources. One very popular example of a game that uses this approach is the ESP game [84]. In input agreement, the two or more randomly selected workers are given input. The workers do not know if they are given the same or different input. They provide labels and can observe other player's labels as well. Based on all labels, the workers decide if they have the same input or not. If they correctly determine whether or not they have the same input, their labels are taken into consideration. Quality of labels is maintained by discouraging random guesses via strong penalties. A popular example is another Game With A Purpose (GWAP) which is TagATune [85]. Similar to output agreement, another QA approach is the multilevel review. However, unlike in output agreement, the work is not done in parallel. A group of workers first perform a task. Then a different group assesses its quality. An example of this approach is the Find-Fix-Verify crowd programming pattern proposed by the authors in [86].

## 2.7 Modeling Cognitive Biases in Crowdsourcing Systems

The Dunning-Kruger effect is one of the many cognitive biases that human beings tend to fall prey to. Eric Bonabeau points out to several human biases that can be observed while making decisions. Examples of biases while generating solutions are the self-serving bias where humans tend to search for information that confirms their existing assumptions, anchoring where they tend to heavily rely on one piece of information, and stimulation bias where they only recognize a solution when they see it. Examples of biases in the evaluation phase of potential solutions are pattern obsession where humans realize patterns that do not exist, and framing where evaluation is influenced by how the solution is presented [87]. In [24], the authors discuss how the phenomenon of cognitive bias has been explored in psychology since the mid-seventies, but has only recently gained attention in the area of information systems. According to them, interest in cognitive bias research is increasing in information systems considering how it revolves around human decision-making. They define cognitive biases in humans as systematic errors in the decision making process that result in suboptimal outcomes. They identify 120 cognitive biases in their analysis which was larger than the number of relevant papers at the time (84). They categorize the biases into perception (e.g. negativity bias), pattern recognition (e.g. confirmation bias), memory (e.g. reference point dependency), decision (e.g. cognitive dissonance), action-oriented (e.g. overconfidence), stability (e.g. anchoring), social (e.g. cultural bias), and interest (e.g. self-justification).

In [88], the authors discuss a main limitation in crowdsourcing systems nowadays where inadequate representation of the uncertainty resulting from human



factors results in suboptimal system design. The human factors that they discuss are mainly related to the availability of workers (workers may leave unexpectedly), the wage that they may request at any point in time, and their varying skill levels. They propose SmartCrowd, an interactive crowdsourcing system that takes into account the dynamic and uncertain nature of crowdsourcing environments. The authors in [89] discuss how human computation is susceptible to systematic biases that cannot be corrected by simply aggregating multiple answers. They study the case of Amazon’s Mechanical Turk. The difference between an error and a bias is that the latter can be mistaken for the true value and is therefore harder to detect and cannot be treated similarly. Their examples include the anchoring effect, common beliefs, and recurring answer bias, all of which cannot be eliminated by simply increasing worker count or any of the common methods for quality control. To this end, they propose the Peer Truth Serum, a game-theoretic incentive scheme that evaluates how scaling bonuses can overcome biases in worker answers resulting in significant improvement in system accuracy. Eickhoff studies the effect of several cognitive biases in document relevance assessment tasks in [90]. He demonstrates how the literature commonly assumes that noisy label submissions are due to three main reasons: unethical spammers, unqualified workers, and malicious workers. There is no consideration to cognitive biases that are systematic deviation patterns from the ground truth. The first step in countering the effects of such biases in crowdsourcing systems is to recognize them. He demonstrates how common QA methods are not enough to overcome this source of noise. To prove the effect of cognitive biases on a system, he studies the ambiguity effect which occurs when missing information in a question makes it appear more difficult and ultimately less desirable to solve. The missing information was chosen so as to have very little relevance with the

document to be labelled. He showed that even when the missing information was not informative, it negatively affected workers' outcome. He then designed a two-phase experiment. In the first phase, workers are presented with information not relevant to the document, and then at a later stage, relevant information becomes available. This is known as the anchoring effect and he shows how it considerably reduced label accuracy. To study the bandwagon effect in another experiment, he discloses to the workers the prior vote statistics resulting in a drop in accuracy. Finally, he looks into the decoy effect, which is a very popular effect in the advertisement discipline where a worker's preference between two options changes with the introduction of a third option (the decoy). He shows the high risk of unintentionally suffering from this effect when several options are provided for relative ranking resulting in degraded label quality.

Gadiraju et al. present a similar work to ours in [91]. They demonstrate how to use worker self-assessments to derive competence levels that can be used along with their performance in the pre-screening phase to achieve better results in crowdsourcing micro-tasks. They start off by describing the Dunning-Kruger effect and investigating through several studies whether it can be observed in paid micro-task crowdsourcing systems which are different in many ways when compared to the controlled experiments performed in the original study by Dunning and Kruger. In their first experiment, they wanted to study crowd workers' self-assessment. They deployed eight tasks on CrowdFlower with varying difficulty levels. After solving the tasks, workers are asked questions related to their perceived test scores and those of others as well as their perceived abilities. They observe that across all tasks of varying difficulty levels, least-competent workers were the ones to significantly overestimate their abilities and scores whereas competent workers underestimate their abilities. In addition, competent work-

ers overestimate other workers' performances more than incompetent workers do. In another experiment, the authors wanted to study the effects of competence on a crowdsourcing task such as tagging images. They were able to show that competent workers outperformed least-competent ones by providing better quality tags that are more diverse. Based on the results from this experiment, they suggested to employ worker self-assessments in the pre-screening phase for better worker selection. They performed another experiment where one group of workers was pre-screened in the traditional manner that considers only their performance whereas another group was pre-screened based on both worker performance and self-assessment. The crowdsourcing task for both groups was sentiment analysis. Their results show that including self-assessment in the pre-screening phase provides a better representation of actual worker competence which results in improved output quality. This work represents a starting point for research related to worker self-assessment in crowdsourcing tasks.

## 2.8 Comparative Analysis

Different methods have been used to classify mobile apps into malware and benign ones. In [92] the authors propose Drebin which extracts permissions, APIs and IP addresses and uses SVM to detect malware. However, due to its large number of extracted features (545,000), the classifier needs a lot of time to be built. Another classifier, DroidMat, uses kNN instead [93]. The problem with this classifier is that its recall value is significantly lower than its precision value. DroidAPIMiner in [94] uses several classifiers based on API features. Its accuracy is high but this might be due to the fact that the testing data used has far more benign than malicious apps. Keeping efficiency in mind, the authors in [95] use an SVM-

based approach to detect Android malware. They use similarity scores (based on suspicious API calls) between benign and malicious apps as features in their feature vector. They additionally train their classifier based on risky permission requests. Using only dangerous API calls, their classifier’s accuracy reached 81% and increased to 86% when adding risky permission requests to the feature vector. Compared to other works however, the accuracy is not very good.

In [96], the authors propose a system to detect Android malware using static analysis, but they focus on a type of attack known as SMiShing where SMS or MMS messages are sent including a URL that prompts users to install a malicious app through some form of social engineering. The work is limited to one type of attack and one type of analysis (static). In their system, a decision is made after parsing messages, comparing against an ID/URL blacklist, collecting the APK, comparing against known APK black and whitelists, analyzing APKs, and comparing against a database of signatures. They perform experiments on several APK families and reach 100% detection. In [97], the authors argue that conventional machine learning classifiers are prone to countermeasures by attackers who are aware of their use. A classifier can be reverse-engineered and training data can be polluted with well-crafted data. To this end, they propose the self-adaptive learning enhancing system, KuafuDet, which includes an offline and an online training phase, that when acting together, improve the detection of adversarial attackers. They evaluated three types of attackers with varying levels of knowledge and influence on the classifiers’ (SVM, RF, and kNN) training data. They show that KuafuDet reduces false negatives and boosts accuracy by 15% with respect to each of the classifiers.

In many cases, decompiling an APK file does not give any useful information. To solve this problem, the authors in [98] propose a hybrid malware detecting

scheme for Android apps. If an app can be decompiled, its API and required permissions can be extracted through static analysis to create a feature vector of 3413 dimensions. Otherwise, the app's system calls will be extracted through dynamic analysis. In static analysis, SVM gave the best results with an accuracy of 99.28%. As for dynamic analysis, Naïve Bayes gave the best results with an accuracy of 90%. The advantage of their approach is that a hybrid scheme of detecting malware works whether or not an app can be decompiled. However, the FPR rate in the dynamic scheme was relatively high (10.85%). The authors in [99] also employ static and dynamic analyses to extract 202 features from every app. Required permissions and sensitive APIs are extracted through static analysis while dynamic behavior is extracted through dynamic analysis. They used a deep learning approach for malware detection, which resulted in 96.5% accuracy and was shown to be better than other machine learning models such as SVM and Naïve Bayes.

In [100], the authors propose DroidClone which exposes code clones to detect Android malware. Their work is extended to detect malware based on Android native code clones as well. They use control flow with patterns to reduce obfuscation effects. They compare their work against NiCad [101] which works at the Java source code level and DroidSim [102] which uses component-based control flow graphs of Android APIs to detect code reuse in malware variants. The accuracy of NiCad, DroidSim, and DroidClone was found to be 83.11%, 89.62%, and 97.85% respectively. Even though the accuracy of DroidClone is superior when compared to other clone detection techniques, the approach is quite complex.

Machine learning was used in [103] to detect mobile malware by classifying images instead of apps. As a first step, every APK file is converted to four selected image formats (Greyscale, RGB, CMYK, and HSL) based on its byte stream. The

GIST feature is then extracted for each image format. Three machine learning algorithms were used (DT, RF, and kNN). RF with the Greyscale format gave the best results with 91% accuracy in detecting malware.

A major drawback in many of the mentioned approaches has to do with decompiling every app's APK file in order to extract the required permissions and generate Smali code for feature vectors. Another drawback is testing every app centrally, which is complicated and not scalable. Finally, all of the stated methods classify apps based on maliciousness and do not study utility. In our work, crowdsourcing was the chosen method to classify apps. Rather than using machine intelligence to classify apps as malicious or not, our system uses the intelligence of the crowd to reach the ground truth regarding the status of an app (high utility, low utility, suspicious, etc.). This detailed app rating along with direct observation from users is what sets our system apart from other classifiers and results in very high classification accuracy of apps as the discrimination threshold increases. Game theory was introduced to model the interaction between our system and apps on user devices. The decision of whether or not to use crowdsourcing which is at the core of our system is based on maximizing our system's profit according to the devised game model.

In terms of quality assurance, most QA methods attempted to correctly identify worker characteristics for better solution quality. Even in the absence of the ground truth, these mechanisms can be used to estimate the competence of workers to a very good degree. We develop a novel approach to detect competence based on reported worker confidence. This approach complements the work in the literature on quality control via proper user identification. In a sense, it can be categorized as a QA method that can be applied with other QA methods (ground truth seeding, contributor evaluation, etc.) for a better representation of

user abilities that takes into consideration incentives, technical skills, experience with a topic, along with any personal and cognitive biases.

Finally, our work in this dissertation complements that of Gadiraju et al. In terms of detecting worker competence through self-assessment, we show similar results. In addition to the common pre-screening method, we also provide a formal model of this cognitive bias which allows us to study several self-assessment-related aggregation techniques. Correctly estimating workers' actual competence levels is one of the several methods to perform quality control in crowdsourcing systems as described before. The problem is that most of these methods do not focus on the human factor in crowdsourcing, and those that do, focus on trying to detect personal bias patterns and reversing their effects on the overall system output. The number of biases, however, is not small, and there is definitely plenty of room for research in this area. In the specific case of the Dunning-Kruger psychological bias, our work and that of Gadiraju et al. provide a starting point.

## Chapter 3

# Crowdsourced Game-Theoretic Rating System

We begin this section with a high-level overview of our system while introducing the general idea behind our composite app rating mechanism. We then proceed to formally define the two rating components and describe the different experiments and simulations that we performed to examine their benefits when compared to a traditional app rating system.

Our system consists of several components as illustrated in Figure 3.1.

- CrowdApp is installed on users' devices to collect objective measures and user input.
- The Utility Score Engine manages extracted features from devices such as crash and pop-up frequencies to compute a utility score for every app.
- The Behavior Score Engine tracks events issued from users' devices. An event is defined as a suspicious incident that takes place on a device while a user is using one of his installed apps. Example events are calls to unknown



phone numbers, sudden increase in CPU usage or bandwidth, etc. Both normal and malicious apps can issue events.

- The User Status Engine separates the crowd into “honest” users and “dishonest” ones and then further sorts the honest ones into “authoritative” and “unreliable”. The importance of this last step is to guarantee that the subjective input provided to our system is coming from users that are well aware of how their devices operate and how their installed apps should function. Such an input is therefore considered accurate.
- The App Reputation Engine continuously updates app reputations.
- The Overall Score Engine presents for every app a detailed score or rating that is composed of two parts: A utility score and a behavior score. The two scores are offered separately since they represent different views of the ratings, and so merging them will hide the insight provided.

The steps of the app rating process are detailed below:

1. CrowdApp extracts features and monitors events issued by apps on all user devices. These datasets are continuously updated and sent to the server that manages incoming data from user devices.
2. The Utility Score Engine obtains a copy of the extracted features and uses it to compute a utility score for all user-installed apps. The Behavior Score Engine obtains a copy of the recorded events and uses it to compute a behavior score for a portion of user-installed apps.
3. The Behavior Score Engine refers to the User Status Engine and the App Reputation Engine for fresh copies of user and app reputations that will

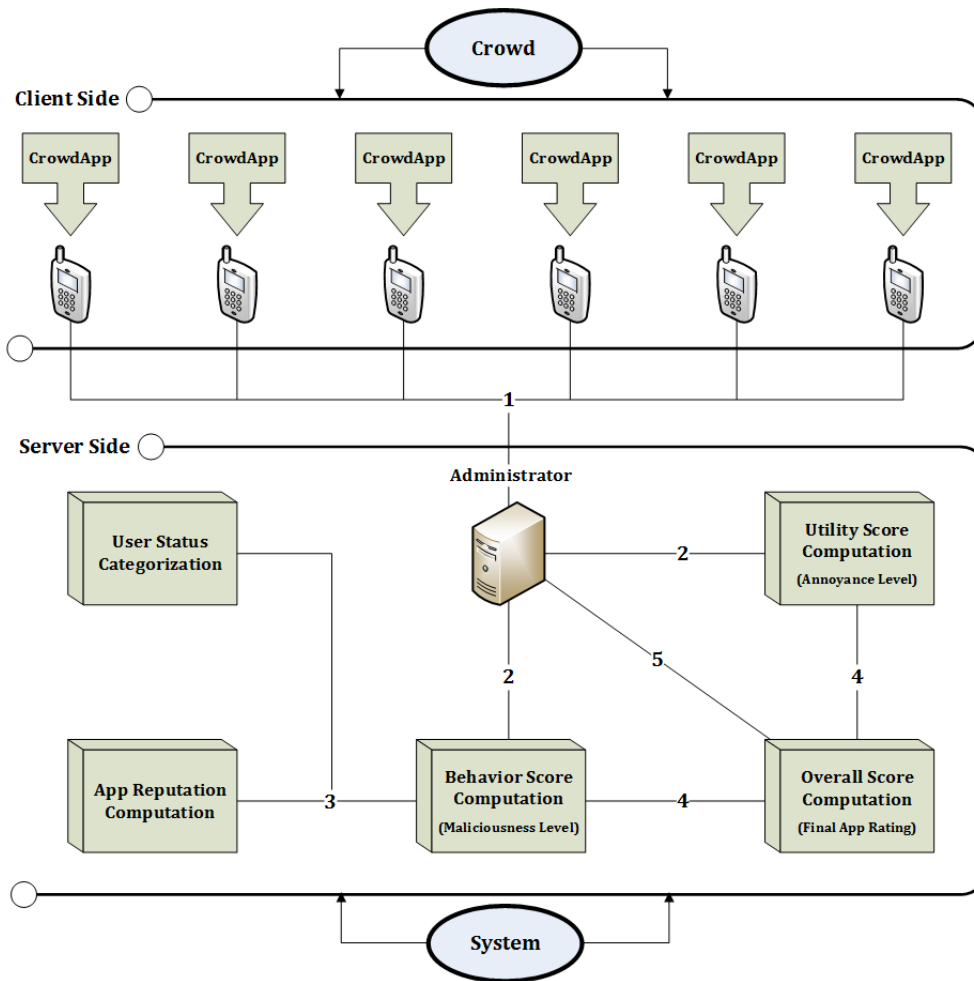


Figure 3.1: System overview showing the crowd, CrowdApp on their devices, and the different engines on the server side

be needed when querying users and updating app reputations. When this engine updates the status of an app, for example by flagging it, the reputations of this app and of the users whose feedback was used in the querying process, are both updated.

4. Utility and Behavior Score Engines send their lists of updated scores to the Overall Score Engine, which will prepare a detailed score for every user-installed app.
5. Upon request, the Overall Score Engine sends a copy of the updated detailed scores to a user via CrowdApp.

CrowdApp records the events issued from every app on all devices and sends this information to the server. When the events counter of an app exceeds its threshold, a game (in a game-theoretic sense) is played between our system and the app that resulted in the irregularity. Our system considers the app's reputation as well as the overall expertise of the crowd. Based on these values, it decides whether or not to refer to the crowd for their subjective input. Based on the input from the crowd, the system will categorize the app as malicious or normal. This, in turn, will result in updating the users' categorization which affects the overall crowd expertise. It will also result in updating the app's reputation. If the app is normal, its reputation is slightly improved. If it is malicious, its reputation is decreased.

### **3.1 Methodology**

As shown in Figure 3.2, the overall rating of an app is a combination of two scores. The first is a utility score that measures the annoyance level versus the

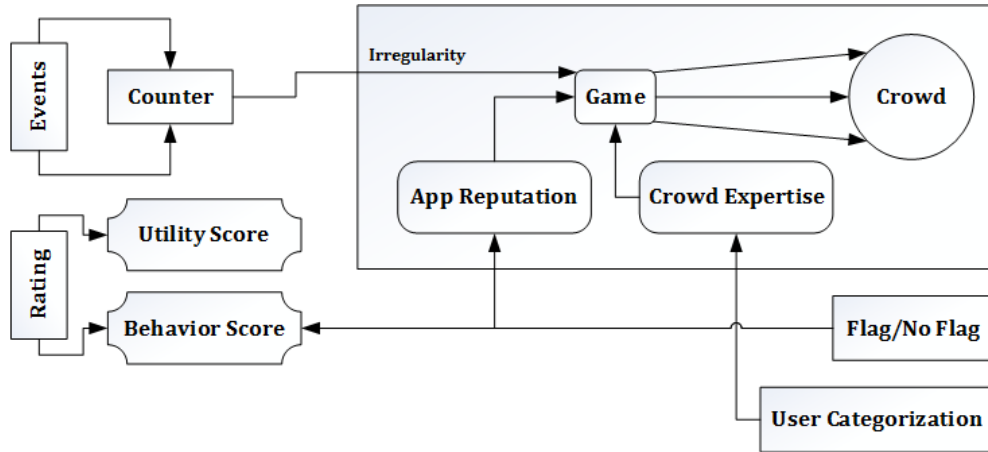


Figure 3.2: App rating process describing how the overall rating of an app is based on utility and behavior scores

utility of an installed app. This score is computed for all apps installed on user devices and is constantly updated. The second is a behavior score that provides a measure of the maliciousness/suspiciousness of an app. Depending on several factors, most important of which being the average rate of events issued by an app, this score is computed. This means that the behavior score is not necessarily computed for all apps. An app that issues a low number of events will only have its utility score in its final rating.

### 3.1.1 Utility Score Computation

The utility score of any app will be based on a set of non-private data that is collected transparently from users' devices. For every user, information is recorded for every app installed on the device. The data collected is: (1) the install period which is the time an app remains on a user's device, (2) the usage time of the app, (3) the number of times an app was opened, (4) its crash record, and (5) the frequency of pop-ups it generates. To collect these records, no explicit participation from the user is required. Nevertheless, in a realistic setup, all

information must be obtained with the users' consent.

### **Install Period**

This is the amount of time that a user keeps an application installed before removing it. The number of installs that is usually shown on the market does not specify how many times the app is instantly removed from the device. If one million users download an app and nearly half of them remove it after a couple of minutes, what is interesting to us is not the one million downloads, but the average install time on the device. In this example, the app clearly should have a low rating; however, seeing that one million users have downloaded it might give a wrong impression. In our scheme, we assume that any app which lasts on a device for more than a month is generally satisfying. Apps that are uninstalled after periods of less than a month are generally assumed to be of low quality. The utility score of these apps is directly proportional to the install period.

### **Average Usage Time**

This feature monitors the time that an app remains open on a device. Same as before, if an app has one million downloads as shown on the market, but on average, the amount of time that it is opened by users is less than a couple of minutes per week, this means that the app, even though it is popular, has a low utility. Of course, the usage time depends on the app itself. The messaging app WhatsApp, for example, is opened several times per day; however, a trip advisor app is probably opened once or a few times per year when the user is planning a trip. This feature alone does not give us clear information regarding the usability of the app, but when combined with other features, it is very useful in determining the usability of the app and it helps in determining the app's final

score. The utility score is directly proportional to the average usage time.

### **Opening Frequency**

This feature records the number of times that a user opens an app. In some cases, a user might open an app very frequently and only use it for few seconds each time. Instant messaging apps are an example. The utility score is directly proportional to this feature.

### **Crash Records**

This feature stores the number of times that an app is force-closed or crashes. This is a clear sign that an app is not reliable and is disruptive to the user. The utility score is inversely proportional to the number of times an app crashes or is force-closed.

### **Pop-Up Frequency**

The frequency of pop ups in an app affects its overall score. An app that features an ad popping up every couple of minutes is generally annoying. And the higher the frequency of the pop ups, the more likely a user will eventually stop using the app. The utility score is inversely proportional to the frequency of pop ups. Of course, sometimes pop ups within an app are legitimate, for example, a pop up that asks the user whether or not he really wants to delete something or exit from the app. It is crucial to differentiate between legitimate pop ups and ads. The former should not reduce an app's utility score.

## Rating Algorithm

After gathering the data from the devices, the recorded values ( $RV$ ) are normalized to numbers between 1 and 5 which is the typical app rating range.

Let  $U_L$  be the largest number in the set of values and let  $U_S$  be the smallest number in the set of values, then the normalized value is defined as:

$$RV = \frac{(5 - 1) \times (OldValue - U_S)}{(U_L - U_S)} + 1 \quad (3.1)$$

The utility score is defined as:

$$US = a \times UT + b \times IP + c \times OF + d \times CF + e \times PF \quad (3.2)$$

Where  $UT$ ,  $IP$ ,  $OF$ ,  $CF$ , and  $PF$  stand for normalized usage time, install period, opening frequency, crash frequency, and pop-up frequency, respectively. The coefficients  $a$ ,  $b$ ,  $c$ ,  $d$ , and  $e$  are scaling factors, which are determined using data from the most active users. The most active users are those who rank in the top 25% in terms of overall activity rate. We calculate the optimal values of the feature weights that will result in the highest correlation between CrowdApp utility scores and Google Play scores, by using the least squares method to solve the over-determined system with  $N$  equations and five unknowns, where  $N$  is the number of different apps of the most active users.

### 3.1.2 Behavior Score Computation

In addition to the utility measure indicated by an apps' utility score, our system also computes a behavior score that reflects the level of suspiciousness of apps. A suspicious app is defined as one that results in specific events that might be affil-

iated with malicious behavior. CrowdApp is constantly monitoring these events and reporting back to the server as shown in Figure 3.1. Here, average values that indicate satisfaction levels are of no interest. What matters is suspicious behavior that indicates malicious implementations.

Some of the recorded events by CrowdApp are:

- Calls and text messages made to phone numbers that are not stored in the address book
- Unidentified increase in the average CPU and RAM usage
- Unidentified increase in bytes of data received by and transmitted from the device
- The screen status at the time of feature collection

Examples of events that are instantly recorded are unknown calls or unknown text messages while the screen is off. Such a behavior is suspicious. Once it happens, it is recorded by CrowdApp with a timestamp. On the other hand, bytes sent and received while the screen is off, are not as indicative of malicious behavior. For example, PopcornTime is an app where users can stream or download movies and TV shows. The device can be in sleep mode while a movie is being downloaded. But this does not mean that all apps that send or receive a significant number of bytes have the consent of the user.

Records of events with timestamps are stored per app. Unlike the utility score case where features are continuously updated and sent only periodically or upon request to the server, in the behavior score computation, events are sent to the server immediately. Every user-installed app will have two values stored on the server, an Event Counter which gets incremented every time an app issues



an event on any user device, and an Event Threshold which gets updated after every interaction between our system and the app. When the Event Counter of an app exceeds its respective Event Threshold, an alarm is fired and a game (in a game-theoretic sense) is played between the system and the app that fired the alarm.

Events issued are not indicative of an attack. Normal apps might also issue events and that does not mean they are “attacking” the system. Hence, Event Counters and Event Thresholds are not used by our system to decide if an app is normal or malicious. They are used to decide whether or not to play the game against the app. If a game is played, this means that events issued by the app in question had to be many, but this app can be normal or malicious with a certain belief value that is reflected as an app reputation.

The Behavior Score Engine gets the app’s reputation from the App Reputation Engine. If the app has a very high reputation, the system does not defend against it. It also raises the event threshold of this app slightly since it assumes that this number of events is considered normal behavior for this app. However, if the reputation is not high, then the system’s best action will be to defend against the app that fired the alarm. To do this, it needs to query users. The Behavior Score Engine gets the list of users whose devices have issued events for this specific app. It then gets the status of these users from the User Status Engine. From this data, it can then choose from the set of users those that were categorized as experts. However, it does not choose all experts, only a small fraction from them. The Behavior Score Engine then contacts CrowdApp on the selected user devices only. Based on the type of event, CrowdApp then queries the device’s owner. The query is a Yes/No/I Don’t Know question. After getting replies back from the users, CrowdApp sends them to the server. Based on these replies, the

Behavior Score Engine will determine whether or not this large number of events was in fact indicative of malicious behavior. Accordingly, the status of the app is set, its reputation is updated, and the status of the users who were queried is also updated.

In terms of the behavior score, there are three possible outcomes: (1) An app that never issues enough events for it to fire an alarm and will therefore not have a behavior score, (2) an app that issues enough events and fires an alarm, however, when choosing the number of users to query with the requirement that these users' devices issued events for this particular app, this number turns out to be small. For user replies to be accurate and for us to consider that the system is based on crowdsourcing, we require a statistically significant number of users to query in every game round. This app will also not have a behavior score but is classified differently than the first category, and (3) an app that issues a lot of events, fires an alarm, and has a statistically significant number of devices that have issued these events. It will have a behavior score that clearly indicates its maliciousness level.

## **Game Model**

In our game model, the players are our system (SYS) and an app (APP) that can belong to one of two types, normal or malicious. Our system's goal is to detect malicious apps and blacklist them so as to ensure accurate ratings throughout the platform. Since the system is unaware of the type of app it is interacting with in every round, our game is one of incomplete information where players do not know some information about the other players such as their type, strategies, payoffs, or preferences. The game is defined as follows:

- Players - There are two players in every game round. Player 1 is SYS.

Player 2 is APP.

- States - There are two possible states for the players, normal (N) and malicious (M). Player 1 can only be normal, while Player 2 can be both.
- Actions - The actions available to Player 1 are to defend by referring to the crowd for their subjective input  $\mathbf{U}$ , or not to defend by accepting the app's utility score as is  $\bar{\mathbf{U}}$ . The actions available to Player 2 are to attack by being a malicious app  $\mathbf{A}$ , or not to attack by being a normal app  $\bar{\mathbf{A}}$ .
- Signals - Player 1 has the same signal for both states of Player 2 ( $\tau_1(\mathbf{NM}) = \tau_1(\mathbf{NN}) = \mathbf{N}_1$ ). Player 2 has different signals for the same state of Player 1. Its issued signal when it is normal is ( $\tau_2(\mathbf{NN}) = \mathbf{N}_2$ ) and when it is malicious is ( $\tau_2(\mathbf{NM}) = \mathbf{M}_2$ ).
- Beliefs - Given the reputation,  $\Omega$ , of Player 2, when Player 1 receives the signal  $\mathbf{N}_1$ , it believes with probability  $\Omega$  that Player 2 is malicious and probability  $(1 - \Omega)$  that Player 2 is normal. On the other hand, when Player 2 receives the signal  $\mathbf{N}_2$ , it believes with probability 0 that Player 1 is normal and Player 2 is malicious, and with probability 1 that Player 1 is normal and Player 2 is normal. When Player 2 receives the signal  $\mathbf{M}_2$ , it believes with probability 1 that Player 1 is normal and Player 2 is malicious, and with probability 0 that Player 1 is normal and Player 2 is normal.
- Payoffs - They are updated in every game round depending on actions of both players.

The game described is an example of a game with incomplete information since the system does not know the type of app it is interacting with in every round of the game. SYS will refer to the app's reputation,  $\Omega$  (that is initially

assumed to be neutral at 0.5), to decide whether or not to defend against this app. If the reputation is lower than a threshold, SYS's best action is to defend and it does so by referring to a fraction of the crowd. If this fraction categorizes the app as normal, SYS will improve the app's reputation. If it categorizes the app as malicious, SYS will decrease the reputation and eventually flag the app.

If Player 2 is a malicious app that is attacking and SYS is defending by referring to the users, then the profits of both players will depend on the probability of correct user detection,  $\beta$ , which is related to the overall expertise of the crowd. Assuming correct detection, the gain incurred by SYS ( $G_M$ ) represents the gain from decreasing the reputation of a malicious app and the loss incurred by the malicious app ( $L_A$ ) is the loss from having its reputation decreased. Assuming incorrect detection, the loss incurred by SYS ( $L_M$ ) is the loss from increasing the reputation of a malicious app and the gain incurred by the malicious app ( $G_A$ ) is the gain from having its reputation increased. Both players incur a cost, one for defending ( $C_D$ ) and one for attacking ( $C_A$ ).

If Player 2 is a malicious app that is attacking and SYS is not defending by referring to the users, then SYS will incur a small loss ( $\ell_m$ ). Since the main goal behind playing the game is flagging suspicious apps, then any round against a malicious app that does not result in decreasing the reputation of the app will result in a loss to the system. The value of this loss will vary depending on whether the app's reputation remains unchanged or it is actually increased. In this case, the reputation remained the same when it should have decreased, so the loss incurred by SYS is minor ( $\ell_m \leq L_M$ ). On the other hand, the malicious app will gain since its reputation remained the same when ideally it should have decreased. The gain ( $g_a$ ) is less than the gain of a malicious app when its reputation increases instead of being decreased ( $g_a \leq G_A$ ). Since SYS did not

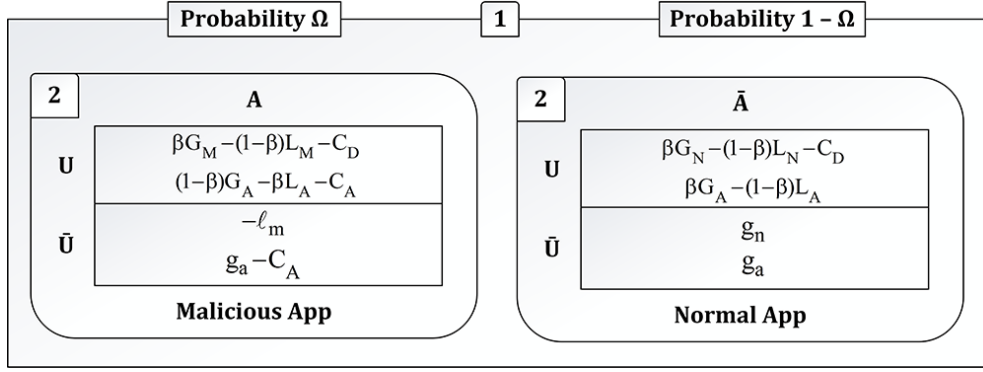


Figure 3.3: Payoff matrix showing the gains, losses, and costs of both players for each of their actions

defend, it has no costs. The attacking app suffers the attacking cost ( $C_A$ ).

If Player 2 is a normal app that is not attacking and SYS is defending, then the profits of both players will depend on the probability of correct user detection,  $\beta$ , which is related to the overall crowd expertise. Assuming correct detection, the gains incurred by SYS ( $G_N$ ) and the app ( $G_A$ ) represent the gains from increasing the reputation of a normal app. Assuming incorrect detection, the losses incurred by SYS ( $L_N$ ) and the app ( $L_A$ ) represent the losses from decreasing the reputation of a normal app. SYS also suffers the cost of defending ( $C_D$ ).

If Player 2 is a normal app that is not attacking and SYS is not defending, both players will incur small gains. Since SYS did not defend, its belief of the normal app is that it is in fact normal (the normal app already has a high enough reputation). By keeping the high reputation as is, both players gain. The gains incurred here are smaller than those incurred when SYS increases the reputation of a normal app ( $g_n \leq G_N$  and  $g_a \leq G_A$ ). The payoff matrix is shown in Figure 3.3.

Regardless of its type, an app will gain when its reputation is increased or remains unchanged. When an app attacks, its cost must be lower than its gain when its reputation increases, otherwise it will have no incentive to attack.

When SYS decreases the reputation of a malicious app, it gains  $G_M$ . However, the value it gains is significantly smaller than the value that a malicious app gains when its reputation is wrongfully increased. This is because SYS requires several instances of a reputation change to benefit as much as one app benefits from an increase in reputation ( $G_M \leq G_A$ ). Similarly, when SYS increases the reputation of a normal app, it gains  $G_N$  which has a smaller value than what a normal app gains when its reputation is increased ( $G_N \leq G_A$ ). It is worth noting that SYS's gain when decreasing the reputation of a malicious app is slightly higher than its gain when increasing the reputation of a normal app ( $G_N \leq G_M$ ). This is because SYS's main aim is to flag malicious apps, so more weight is given to identifying malicious rather than normal apps. When SYS defends, its cost must be lower than its gain when it decreases the reputation of a malicious app, otherwise it will have no incentive to defend.

### **Best Response Analysis**

Defending is SYS's best response when its expected payoff is higher than that of not defending. The expected payoffs of SYS are shown in Figure 3.4. Setting the upper value higher than the lower value results in a lower limit on the belief as illustrated in equation 3.3. The value of this lower limit will be defined as the Reputation Threshold based on which SYS forms its decision. When SYS's belief of an app being malicious is high (app has low reputation), its best response is to defend. When this belief is low (app has high reputation), its best response is not to defend. The value of this belief above which SYS should defend depends on the probability of correct user detection (crowd expertise) as shown in equation 3.3.

Expected Payoff	
U	$\Omega \left[ \beta(G_M + L_M - G_N - L_N) - (L_M - L_N) \right] + \left[ \beta(G_N + L_N) - (L_N + C_D) \right]$
$\bar{U}$	$g_n - \Omega(\ell_m + g_n)$

Figure 3.4: The expected payoffs of SYS when its actions are to defend or not to defend

$$\Omega > \frac{(g_n + L_N + C_D) - \beta(G_N + L_N)}{(g_n + L_N + \ell_m - L_M) - \beta(G_N + L_N - G_M - L_M)} \quad (3.3)$$

### Reputation Updates

When an app fires an alarm, the Behavior Score Engine gets the app's reputation from the App Reputation Engine. If the app's reputation is higher than the computed Reputation Threshold, SYS does not defend against it. It also raises the Event Threshold of this app slightly since it assumes that this number of events is considered normal behavior for this app. However, if the reputation is lower than the Reputation Threshold, then SYS's best action is to defend against this app by referring to the crowd. As such, the Behavior Score Engine requests the list of users who have the app installed and whose devices have issued events for this app. Also, it gets the status of these users from the User Status Engine. Then, it can choose from the set of users those that were categorized as authoritative by SYS. However, it only chooses a small random fraction of these users. The size of the chosen fraction depends on how long CrowdApp has been installed on devices. At the beginning, the information available to

SYS regarding the status of users is still not very accurate and so the larger the fraction of users to query, the more accurate the joint replies. After a while, the user status information becomes more accurate, which allows SYS to query a smaller number of users most of which are authoritative with high confidence. The Behavior Score Engine then informs the admin to contact CrowdApp on the selected user devices only. Based on the type of event, CrowdApp queries the device's owner. The query is a "Yes/No/I don't know" question. For example, if the issued events are related to a sudden increase in bytes received by the device, CrowdApp asks the user whether or not she is aware that this app is consuming bandwidth. After getting replies from the crowd, CrowdApp sends them to the server. Based on these replies, SYS determines whether the selected app is malicious or normal based on an aggregated vote. If the resulting vote agrees that the app is suspicious, it is flagged. Otherwise, it is classified as normal. If an app is flagged, its reputation is reduced by a ratio equal to the difference in replies (Users classifying app as suspicious - Users classifying app as normal) divided by the total number of queried users. If an app is classified as normal, its reputation is increased by that same ratio and its Event Threshold is increased by a product equal to this ratio plus one.

Accordingly, the reputations of both the queried users and the app are updated and sent to the User Status and App Reputation Engines, respectively. Reputations of users whose replies agree with the aggregated vote will increase, whereas reputations of users whose replies don't agree with the aggregated vote will decrease.



Symbol	Description
$US_{(i,j)}$	Utility score of app $i$ in round $j$
$BS_{(i)}$	Behavior score of app $i$
$\gamma_{1(i)}$	Binary value for 1 <sup>st</sup> game condition
$\gamma_{2(i)}$	Binary value for 2 <sup>nd</sup> game condition
$\gamma_{3(i)}$	Binary value for 3 <sup>rd</sup> game condition
$ET$	Base value of Event Threshold
$ET_{(i,j)}$	Event Threshold of app $i$ in round $j$
$E_{(i)}$	Events issued from app $i$
$R_{(i,j)}$	Reputation of app $i$ in round $j$
$\Omega_{(i)}$	Belief of maliciousness of app $i$
$\beta_{(j)}$	Probability of correct user detection in round $j$
$UQ_{(i)}$	Users available to query regarding app $i$
$CS_{min}$	Minimum crowd size required to query
$ctr_N$	Number of replies indicating normal app behavior
$ctr_M$	Number of replies indicating malicious app behavior
$U$	Total number of users with CrowdApp
$\Psi$	Fraction of malicious apps
$\delta_M$	Event rate of malicious apps
$\delta_N$	Event rate of normal apps
$\Delta_{(i)}$	Devices with extracted features associated with app $i$

Table 3.1: Variables used in mathematical model

## 3.2 Mathematical Model

Three conditions are defined for an app to be subjectively tested by SYS:

- The app issues a number of events larger than its Event Threshold value.
- The belief of maliciousness of the app is larger than the threshold in equation 3.3.
- The number of users available to be queried by SYS is larger than the minimum required number of users for the replies to be statistically significant.

Table 3.1 shows all the variables used in our proposed mathematical model along with a description of each term.

Equation 3.4 defines the minimum allowed number of users that are providing their subjective input for these replies to be considered statistically reliable. The minimum number is 10% of the total number of users who have CrowdApp installed or 50 if the total number of users is small. If the number of users who have replied to a query is less than  $CS_{\min}$ , then the app will have no reputation.

$$CS_{\min} = \max(50, \text{round}(0.1 \times U)) \quad (3.4)$$

Equation 3.5 represents the first condition for the app to be subjectively tested by SYS which is related to the number of events issued by an app from the start of the event timeout for event  $k$ , ( $t_{\text{ETO}(k)}$ ), up until the current time, ( $t_C$ ).

$$\gamma_{1(i)} = \left\lfloor \frac{\text{sgn}\left(\sum_{t=t_{\text{ETO}(k)}}^{t_C} E_{(i)} - ET_{(i,j)}\right) + 1}{2} \right\rfloor \quad (3.5)$$

Equation 3.6 represents the second condition for the app to be subjectively tested by SYS which is related to the app's reputation versus the Reputation Threshold.

$$\gamma_{2(i)} = \left\lfloor \frac{\text{sgn}\left(\Omega_{(i)} - \left(\frac{(g_n + L_N + C_D) - \beta_{(j)} \times (G_N + L_N)}{(g_n + L_N + \ell_m - L_M) - \beta_{(j)} \times (G_N + L_N - G_M - L_M)}\right)\right) + 1}{2} \right\rfloor \quad (3.6)$$

Equation 3.7 represents the third condition for the app to be subjectively tested by SYS. It is related to the minimum available users to query which was defined in equation 3.4.

$$\gamma_{3(i)} = \left\lfloor \frac{\text{sgn}(\text{UQ}_{(i)} - \text{CS}_{\min}) + 1}{2} \right\rfloor \quad (3.7)$$

Equation 3.8 combines the three conditions (binary values) for an app to be subjectively tested. If only one of these conditions is not satisfied, the new reputation (behavior score) is the same as the previous value of the reputation. Otherwise, the reputation is updated.

$$\text{BS}_{(i)} = 1 + 4 \times \left\{ \text{R}_{(i,j-1)} + \left[ \gamma_{1(i)} \times \gamma_{2(i)} \times \gamma_{3(i)} \times \left( \frac{\text{ctr}_N - \text{ctr}_M}{\text{UQ}_{(i)}} \right) \right] \right\} \quad (3.8)$$

Equation 3.9 defines the initial value of the Event Threshold that is common for all apps. Equation 3.10 shows how this threshold is updated every time an app is subjectively tested, in a manner similar to the update of an apps' behavior score.

$$\text{ET} = \frac{(\Psi \times \delta_M) + ((1 - \Psi) \times \delta_N)}{100} \quad (3.9)$$

$$\text{ET}_{(i,j)} = \text{ET}_{(i,j-1)} \times \left( 1 + \left[ \gamma_{1(i)} \times \gamma_{2(i)} \times \gamma_{3(i)} \times \left( \frac{\text{ctr}_N - \text{ctr}_M}{\text{UQ}_{(i)}} \right) \right] \right) \quad (3.10)$$

Equations 3.11 and 3.12 describe the utility score computation. The format is similar to the rating function in 3.2. The difference is that 3.11 defines an old value of the utility score and 3.12 defines the new computed value.

$$\begin{aligned} \mathbf{US}_{(i,j-1)} &= (0.246 \times \mathbf{UT}_{(i,j-1)}) + (0.239 \times \mathbf{IP}_{(i,j-1)}) + (0.362 \times \mathbf{OF}_{(i,j-1)}) \\ &\quad + (0.155 \times \mathbf{CF}_{(i,j-1)}) + (0.286 \times \mathbf{PF}_{(i,j-1)}) \end{aligned} \quad (3.11)$$

$$\begin{aligned} \mathbf{US}_{(i,j)} &= (0.246 \times \mathbf{UT}_{(i,j)}) + (0.239 \times \mathbf{IP}_{(i,j)}) + (0.362 \times \mathbf{OF}_{(i,j)}) \\ &\quad + (0.155 \times \mathbf{CF}_{(i,j)}) + (0.286 \times \mathbf{PF}_{(i,j)}) \end{aligned} \quad (3.12)$$

Equation 3.13 shows how the previous and current values of the utility score are combined. A higher weight is given to the score value that resulted from data emanating from a larger number of devices since it is a better depiction of the app's utility.  $t_{R(j)}$  is the time at the start of round  $j$ .

$$\mathbf{US}_{(i,j)} = \left(1 - \frac{\sum_{t=t_{R(j)}}^{t_C} \Delta(i)}{\sum_{t=t_{R(j-1)}}^{t_C} \Delta(i)}\right) \times \mathbf{US}_{(i,j-1)} + \left(\frac{\sum_{t=t_{R(j)}}^{t_C} \Delta(i)}{\sum_{t=t_{R(j-1)}}^{t_C} \Delta(i)}\right) \times \mathbf{US}_{(i,j)} \quad (3.13)$$

### 3.3 Simulation Setup

We now describe the setups for both the utility score and the behavior score computation. Validation of the utility score was carried out via our first implemented design of CrowdApp (version 1.0), and the experiment was carried out with four users. As for the behavior score, the design was validated via MATLAB simulations as shown in the following sections. The adopted aggregation method is a variant of plurality voting.

### 3.3.1 Utility Score Setup

To collect the required data, we implemented CrowdApp v1.0, which is a client that runs on rooted Android devices. It has four services that are constantly running in the background and collecting user data.

The first service calculates the install period for every user-installed app on the device. When a user uninstalls an app from his device, the install period is calculated. Once and if an app is uninstalled, all its records are sent to the server and are deleted from local storage. The install periods for all apps are stored in a file called “Period.txt” on the device.

The second service calculates the number of times an app is opened by the user. A messaging app such as WhatsApp is typically opened several times during the day. Another app such as Duolingo is opened several times a week. A constant opening frequency might be a good sign of user satisfaction whereas a decrease in frequency might suggest a decrease in satisfaction. In order to differentiate between different types of apps that are used differently by users, one approach would be to consider the fluctuation of the frequency rather than its absolute value. The opening frequency per app is recorded for a long period of time up until we reach a constant average over a predetermined timespan. For example, WhatsApp’s opening frequency is recorded over an entire month during which averages are taken over different timespans (one hour, one day, one week, etc.). An app such as WhatsApp will have a constant average if the chosen timespan is a day but a fluctuating one if it is an hour. The app is likely to be opened more at noon than after midnight. Some apps on the other hand such as TripAdvisor are opened several times per year when the user is planning a trip. Such apps would have to be monitored for a longer period of time in order to determine their average opening frequency. Consider the following scenario: A user installs

two new apps on his device. The first one is a Sudoku game and the other a TripAdvisor app. The user plays Sudoku the first two days and then gets bored and stops playing the game without uninstalling it from the device. He also uses the TripAdvisor app to plan a trip. Throughout the year, he is no longer playing the game and has no other trips to plan. The following year, he does not play the game but plans two trips. In this case, we cannot deduce a lot of information from the first year since both apps showed similar opening frequency. However, the second year tells us that the TripAdvisor app is still being used and a proper timespan to consider for an average value of its opening frequency is one year. Whereas the game was only opened once at the beginning and throughout the next couple of years, was left installed on the phone without being used.

The third service calculates the average usage time for every user-installed app on the device and takes into consideration apps that are installed after the service has started. This feature depends on the opening frequency of the app. With time, the timespan to consider for computing an average usage time is defined based on which the average value can be constantly updated. Upon initiating our app on a user device and until the proper timespan for the app-under-test is detected, the total usage time is logged with timestamps of when an app is opened. The recorded usage times are stored in a file called "Usage.txt" on the device.

The fourth service calculates the number of times an app crashes based on the onCreate, onPause, onResume, and onDestroy events called by an activity. Apps that crash or are forcefully closed by the user do not call the onDestroy method after being created or resumed. On the other hand, apps that exit normally call onDestroy. In order to intercept these methods, this service was implemented as an Xposed module which is a framework for modules that can change the behavior

of the system and apps without modifying APKs. The created Xposed module runs like any Android service but requires a rooted device. Upon initiating our app and until detecting the proper timespan for an app-under-test, crash records are stored in absolute value along with timestamps of the logged crash. When a timespan is set, an average value can be used. The timespans for the average opening frequency and average usage time do not necessarily have to be the same as the timespan for crash records. For example, for WhatsApp it would make sense to compute the average times it is opened by a user per day and the average number of times it crashes per month. Again with crash/forceful closing, all records are stored in a separate file called “Crash.txt” on the device.

The fifth service is also implemented as an Xposed module. It monitors the onCreate of alert dialog boxes inside every installed app on the device. By keeping track of opened dialog boxes for apps, the relative frequency of pop-ups per app can be estimated. This feature also requires a separate timespan. For example, WhatsApp’s pop-up frequency could be averaged over a week. The recorded features from this service are stored in a file called “Popup.txt” on the device.

In order to detect the proper timespan for every feature, CrowdApp has to monitor all other apps for a significant period of time. In order to get more accurate results regarding the types of apps installed on each device, we need to consider the change in app statistics over long periods of time. The recommended period is one year. An app that is used once per year only is still considered a satisfactory app as long as it is used at least once every year. Another upside of having our client on several devices is that after monitoring an app on a portion of devices and inferring its proper timespans, it can use these values on other devices.

In Figure 3.5, we show a sample output from the two text files, “Usage.txt”

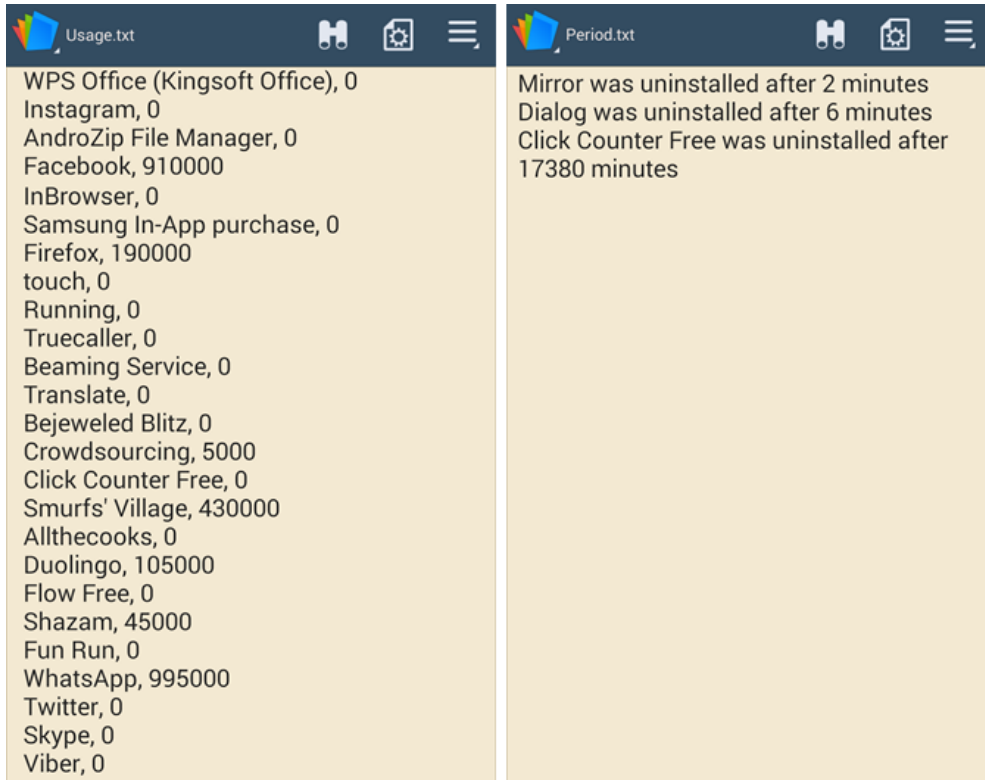


Figure 3.5: Usage.txt and Period.txt sample output

User	User 1	User 2	User 3	User 4
<b>Device</b>	Samsung Galaxy S4	Samsung Galaxy S3	Samsung Galaxy S3	Samsung Galaxy S3
<b>OS version</b>	4.4.2	4.3	4.3	4.1.2

Table 3.2: User devices

and “Period.txt”. In the first file, Usage.txt, every user-installed application is shown and its usage time in milliseconds is recorded next to it. In the second file, Period.txt, every application that was removed from the device is listed and the time it remained on the device is shown next to it in minutes. The application Mirror was uninstalled after only two minutes of being installed, Dialog after six, and Click Counter Free after 17,380 minutes (about twelve days).

CrowdApp was installed on four rooted Android phones, with their specifications shown in Table 3.2.



Variable	Description	Base Value
T	Total rounds in simulation	1440
TB	Duration of one round in seconds	30
TNA	Total number of apps	1000
FMA	Fraction of malicious apps	0.25
MER	Malicious event rate	50
NER	Normal event rate	50
TNU	Total number of users	200
AR	Activity rate of users	13
APU	Number of apps on user device	27
CE	Crowd expertise	0.7

Table 3.3: Base values

The four users that installed CrowdApp were aware that it is a service that will be running in the background at all times while collecting and logging data. However, they were not told why this data was being collected so as not to affect their usage patterns. The testing was carried out over a period of one week.

### Behavior Score Setup

The Behavior Score Engine was simulated in MATLAB. The aim was to show the performance of the engine in classifying apps and users in different scenarios. These scenarios are defined by the inputs to every experiment. Table 3.3 lists the different input variables that were varied in the experiments with base values for every variable. Unless otherwise stated, base values were used.

Table 3.4 shows the Android app distribution up until 2016 [104]. A similar app distribution was adopted in our experiments. The malicious apps were assigned disregarding the popularity of apps, meaning that any malicious app had the same probability of belonging to any one of the seven app categories. After randomly assigning apps to their respective download categories based on the below percentages, the apps installed by users on their devices were assigned. The choice of apps is not entirely random; it takes into account the popularity

Times downloaded	Number of apps	% of total app count
< 100	679,887	35
100 – 1,000	539,731	28
1,000 – 10,000	386,975	20
10,000 – 100,000	223,739	12
100,000 – 500,000	61,623	3
500,000 – 1,000,000	12,391	1
> 1,000,000	17,139	1
<b>Total</b>	1,921,485	100

Table 3.4: Android download distribution

of the chosen apps. Apps from the category in row six are more likely to be chosen than those from the category in row one. Popular apps are chosen more frequently than others.

According to [105], the fraction of malicious Android apps currently in the market is around 0.25. Accordingly, the base value of FMA was set to 0.25. The base value of the timeout is 1,440. This means that simulations stop after iterating for 1,440 rounds. Since the base value of a round is 30 seconds, every simulation will represent 12 hours in real life. According to [106], U.S. smartphone users accessed 26.7 apps per month in the fourth quarter of 2014. In our experiments, this value is the App Count per User. The number of apps installed on devices is then defined per user based on a normal distribution with a rounded mean of 27.

According to [107], in 2015, U.S. smartphone and tablet users used mobile apps for around three hours and five minutes every day. This means that at any time during the day, the probability of a user being active is  $\left(\frac{3 \times 60 + 5}{2460}\right) \times 100\% \approx 13\%$ . So the base value of the Activity Rate was set to 13.

The process proceeds as a Time Block of 30 seconds whereby a user uses his device according to a probability based on a normal distribution around the Activity Rate. If a user is using his device in this time block, it is assumed that

on average, he will be using one app. A random app from his set of installed apps is chosen. This app will then issue events according to a probability based on the Malicious/Normal Event Rate. The Event Counters of apps are constantly updated and at any point where the Event Counter of an app exceeds its personal Event Threshold, a game is played. CrowdApp checks the reputation of the app based on which it chooses its best action.

In all game rounds, profits of SYS, normal, and malicious apps are updated based on actions chosen by players. The system converges once all malicious apps are detected and flagged.

For different ranges of input values, two receiver operating characteristic (ROC) curves are of interest, one for classifying apps and one for classifying users. The ROC curve is a plot that shows the performance of a binary classifier as its discrimination threshold is varied. It is a plot of the true positive rate (TPR), also known as the sensitivity index, against the false positive rate (FPR), which can be calculated as  $(1 - \text{specificity})$ , at different thresholds. ROC curves are plotted to select optimal threshold values and discard sub-optimal ones. In the case of apps, our population is the set of rated apps, i.e. category 3 in subsection 3.1.2. From the set of rated apps, the total number of normal apps is Apps Total Negatives, and the total number of malicious apps is Apps Total Positives. In the case of users, the population is the set of users that were queried at least once. From this set, the total number of authoritative users is Users Total Positives, and the total number of unreliable users is Users Total Negatives.

An experiment proceeds as a Time Block of 30 seconds. In this block, a user uses his device according to a probability based on a normal distribution around the Activity Rate with the base value 13. If a user is using his device in a block, we assume that on average, he will be using one app. A random app from his

set of installed apps is chosen. If the chosen app is normal, it will issue events according to a probability based on the Normal Event Rate with base value 50. If the chosen app is malicious, it will issue events according to a probability based on the Malicious Event Rate with base value 50. The event counters of apps are constantly updated and at any point where the event counter of an app exceeds its personal event threshold, a game is played. The system checks the reputation of the app based on which it chooses its best action. If the reputation is lower than a computed Reputation Threshold, then SYS's best response is to defend. It will defend by referring to a fraction of the users. This fraction of users is equal to the product of Fraction Users Query and the number of users who have the app and whose devices issued events for this app. The fraction will be chosen randomly from this subset. The variable Fraction Users Query depends on where we are in the running experiment. It is equal to  $\max(0.1, 1 - \text{Counter}/\text{Timeout})$  where Counter indicates in which iteration of the experiment we are and Timeout is the total length of the experiment which is set at the beginning. This means that when we first start the experiment, the Counter value is still very small, so Fraction Users Query is large. As the experiment proceeds, this value keeps on decreasing until it reaches its minimum value of 0.1. This is because at the beginning of an experiment, the information available to our system regarding that status of users is still not very accurate and so the larger the fraction of users to query, the more accurate the joint replies. As the experiment proceeds, the user status information becomes more and more accurate. This better accuracy allows the system to query a smaller number of users most of which are experts with high confidence.

Based on the replies from the chosen users, SYS determines whether the selected app is malicious or normal. If more queried users agree that the app

is malicious, then it is flagged. Otherwise, it is classified as normal. If an app is flagged, its reputation is reduced by a ratio equal to the difference in replies (Users classifying app as malicious - Users classifying app as normal) divided by the total number of queried users. If an app is classified as normal, its reputation is increased by that same ratio. The event threshold of an app classified as normal is also increased by a product equal to this ratio plus one.

Every app has an Event Timeout which is the time that an event issued by the app continues to take place. For example, an event issued by app 230 might last for one hour on all user devices whereas an event issued by app 10 might last for an entire day. All users are queried about events that were issued on their devices no longer than 30 seconds ago (assuming the chosen Time Block is 30 seconds). However, all user replies within an event's timeout are aggregated to get the result. The Event Timeout of apps is a random number between ten minutes and one day.

Based on the Crowd Expertise value, experts are chosen randomly from the set of all users. When assigning expertise values for users, we first start by assigning a value of 50 for all users. This means that all their replies will be 'correct' 50% of the times. In other words, their replies are always random. Then the chosen experts are assigned an expertise value ranging between 90 and 100. In other words, replies of randomly chosen experts will be 'correct' 90% to 100% of the times and only rarely are they 'incorrect'.

These assigned expertise values are the 'actual' expertise levels of the users in any chosen scenario. In reality, our system does not have access to this information at first. It tries to learn the expertise of users with time. We define Users Detected Expertise, to represent the detected expertise values by SYS. It is constantly updated for every user.

Users Correct Detection Counter is the number of times every user's reply agrees with the crowd's reply which we assume to be the 'correct' reply if we are to trust the wisdom of the crowd. The value Users InCorrect Detection Counter, on the other hand, is the number of times every user's reply does not agree with the crowd's reply.

Every user's rating of every app he has installed is computed taking into consideration his expertise and the type of the app (normal/malicious). We span all the users and for each one we get the list of apps downloaded on his device. Then each app from this list is assigned a rating as given by the user. If an app is normal (or malicious) and the user is an expert then there is a very high chance that he will rate it as normal (or malicious). If, on the other hand, the user is a non-expert meaning that his expertise is always 50, then it is equivalent to saying that regardless of the type of app, his rating is always random. Here when we say rating, we do not actually mean a rating between 1 and 5 that is provided by users. It is only a term used in the simulations. It is actually a representation of the correctness of the replies when the admin informs CrowdApp to query users based on specific events.

Every round in our experiments is set to the defined Time Block variable. For every user, we first check to see whether or not he is active and this is based on his Activity Rate. If the user is in fact active then one of his apps is chosen randomly as the app being used in this Time Block. Then we check whether or not this chosen app issued any events in the Time Block and this is based on the Event Rate of the app. If an event was issued by this app during this Time Block on this user's device then we increment Apps Event Counter per Round matrix for this specific user and this app. This inner loop ends after spanning all users once. After spanning all the users once, we get the total events per app

by adding the new values from Apps Event Counter per Round matrix to the Apps Event Counter Total array. At the start of every round the Apps Event Counter per Round matrix is reset to zero and recalculated. The Apps Event Counter Average array stores the average Event Rate of every app which is the Apps Event Counter Total array divided by the Apps Number Times Opened array. Apps that fire an alarm are those whose average Event Rate is larger than their Events Threshold. They are stored in the Apps Check array in order to be verified separately.

All the apps that issued events at any point will have their value of Events Boolean set to 1. In every round every app with this value set to 1 will have its Events Timeout InProgress decreased by 30 which is the duration of the round. The Events Timeout InProgress value is the value which is changed between the rounds. Once this value reaches 0, the Events Boolean value of the app is set back to zero, its entry in Users Events is cleared and its Events Timeout InProgress is set back to its initial value of Events Timeout. Also, the app's two values of event counters (total and average), are set back to 0. Basically, for the app in question, everything is set back to its initial value. Even if an app did not fire an alarm in the round we are currently in, if its Events Boolean value is 1, then 30 seconds must be removed from its Events Timeout InProgress value.

For the apps stored in the Apps Check array, there are two cases to consider: (1) In the first case, the app's Events Timeout InProgress value is still greater than 0 which means that replies from users with events in the last 30 seconds still count and they have to be considered. So, these new users must be added to the crowd, and this is done by calling the function Get Users with Events which updates the Users Events matrix, after which the game is played by calling the Game function, and (2) in the second case, the Events Timeout InProgress

value has reached 0 which means that the new replies from users with events from the last 30 seconds do not count since this event's timeout has ended. The new users are not taken into consideration, and the previous value of the Users Events matrix is used as an input in the Game function.

After going through the apps that fired an alarm in the current round and getting the updated Apps and Users matrices, the expertise levels of all the users are updated based on a normalized formula that takes into consideration the answer that resulted from the crowd as well as the number of 'correct' and 'incorrect' replies by all users.

If an app hasn't been rated before, or there was an attempt to rate it but the number of users was less than the minimum crowd size, or if it is any number lower than the Reputation Threshold, then the game will take place. Replies from all users in Users Events are considered. These replies are used to determine Count Normal and Count Malicious. For every app, the larger value between the two will determine whether the app is normal or malicious. The crowd's answer is assumed correct. Individual replies from users are then compared against this 'correct' answer.

In addition to updating the reputations of the apps, the Events Threshold values of the apps that were checked and found to be normal are increased by a value that depends on Change which in turn depends on the difference between the malicious and normal votes of the app in question. Also, depending on the answer of every user, his Users Correct Detection Counter and Users InCorrect Detection Counter values are updated. Finally, if the minimum crowd size is not satisfied, an app's reputation is set to **X** so as to differentiate between different cases.

App reputations at the end of an experiment are divided into four categories:



- NaN - There were no events (or a small number of events) issued by the app which means that it wasn't subjectively tested. In this case, our system will only return the utility score of this app.
- X - These are the apps that issued a significant number of events but there weren't enough users to query for a statistically significant result. The minimum crowd size defined by our system is  $\min(50, \text{round}(0.1 \times \text{TotalUserCount}))$ . This means that unless the number of users whose devices issued events for this specific app is at least equal to 10% of the total number of users (or 50 if the number of users is relatively small), then the minimum crowd size requirement is not achieved. In this case, our system will only return the utility score to users but will also notify them of potential malicious behavior due to an unknown behavior score value.
- Reputation  $< 0.5$  - Apps with a reputation below 0.5 are those that issued enough events, fired an alarm, fulfilled the minimum crowd size requirement, were behaviorally tested by our system, and were found to be malicious to a certain extent.
- Reputation  $> 0.5$  - Apps with a reputation above 0.5 are those that issued enough events, fired an alarm, fulfilled the minimum crowd size requirement, were behaviorally tested by our system, and were found to be normal to a certain extent.

After classifying an app, the users whose replies agree with the combined result will have their Correct Detection Counter incremented by 1 and those whose replies do not agree with the combined result will have their InCorrect Detection Counter incremented by 1. These counters are used by the system when updating the expertise of queried users.

The value of  $\beta$  in Figure 3.3 is defined as the probability of correct user detection. In our case, if the users categorize an app correctly, then  $\beta$  is 1, and if they don't then  $\beta$  is 0. We know whether or not the categorization is correct because we know the list of 'true' malicious apps (the ground truth).

In all iterations of the game, the profits of SYS, a normal app, and a malicious app are being updated based on the actions chosen by the two players. The system converges when all the malicious apps have been flagged by SYS.

The ROC curve of app classification is affected as follows: If an app is in fact normal and SYS classifies it as malicious, the Apps False Positives value is incremented. On the other hand, if an app is in fact malicious, and SYS classifies it as malicious, the Apps True Positives value is incremented. True positive and false positive rates of apps are computed in equations (14) respectively.

The ROC curve of user classification is affected as follows: If a user is in fact an expert and SYS categorizes him as an expert, the Users True Positives value is incremented. On the other hand, if a user is in fact a non-expert and SYS categorizes him as an expert, the Users False Positives value is incremented. True positive and false positive rates of users are computed in equations 3.14 respectively. Additionally, pseudo-code of the described method is given in Figure 3.6.

$$\begin{aligned}
 \text{TPR}_{\text{Apps}} &= \frac{\text{TruePositives}_{\text{Apps}}}{\text{TotalPositives}_{\text{Apps}}} \\
 \text{FPR}_{\text{Apps}} &= \frac{\text{FalsePositives}_{\text{Apps}}}{\text{TotalNegatives}_{\text{Apps}}} \\
 \text{TPR}_{\text{Users}} &= \frac{\text{TruePositives}_{\text{Users}}}{\text{TotalPositives}_{\text{Users}}} \\
 \text{FPR}_{\text{Users}} &= \frac{\text{FalsePositives}_{\text{Users}}}{\text{TotalNegatives}_{\text{Users}}}
 \end{aligned} \tag{3.14}$$

```

while not Timeout
  for all Users
    compute ActivityRate
    select ActiveApp
    select EventRate
    Random  $\leftarrow$  flip coin
    if Random less than or equal to EventRate
      | increment Events
    end if
  close for

  add Events to TotalEvents
  compute AverageEvents from TotalEvents

  if AverageEvents greater than or equal to EventsThreshold
    | if EventsBoolean TRUE
    | | if EventsTimeout greater than 0
    | | | run Game and return AppReputation, CorrectDetection, IncorrectDetection
    | | else
    | | | run Game and return AppReputation, CorrectDetection, IncorrectDetection
    | | | reset EventsTimeout
    | | | EventsBoolean  $\leftarrow$  FALSE
    | | | TotalEvents  $\leftarrow$  0
    | | end if
    | end if
  end if

  Expertise  $\leftarrow$  difference (CorrectDetection, IncorrectDetection)

close while

AppsFPR  $\leftarrow$  ratio(AppsFalsePositives, AppsTotalNegatives)
AppsTPR  $\leftarrow$  ratio(AppsTruePositives, AppsTotalPositives)
UsersFPR  $\leftarrow$  ratio(UsersFalsePositives, UsersTotalNegatives)
UsersTPR  $\leftarrow$  ratio(UsersTruePositives, UsersTotalPositives)

```

Figure 3.6: Pseudo-code of described method

<b>Feature</b>	UT	IP	OF	CF	PF
<b>Weight</b>	0.246	0.0239	0.362	0.155	0.286

Table 3.5: Optimal weights

<b>App</b>	<b>UT</b>	<b>IP</b>	<b>OF</b>	<b>CF</b>	<b>PF</b>
Smart Voice Recorder	1.01	2.23	1	5	5
Facebook	5	5	2.25	1	4.95
InBrowser	1.87	4.74	1.35	5	4.59
Smurfs' Village	3.9	4.89	1.27	5	4.73
Duolingo	1.5	2.56	1.06	5	5
Shazam	1	2.79	1	5	5
Goal.com	1.21	1.01	1.05	5	5
WhatsApp	4.61	5	5	3.74	4.69
Viber	2.04	5	1.2	5	4.97
PowerTutor	1.02	1	1.02	5	1

Table 3.6: User 1 collected data

## 3.4 Design Results and Analysis

In this section, we first present the results of the utility score experiment with CrowdApp v1.0 with a brief discussion of the implications behind these results. In the following subsection, we show the results of the MATLAB simulations and discuss what these results mean in the context of a behavior score rating system.

### 3.4.1 Utility Score Results

Using the data from the most active user (User 1), we calculated the optimal values of the feature weights as described previously. The weights are shown in Table 3.5. The normalized data from User 1's device is shown in Table 3.6.

For the above set of user-installed apps for User 1, the Google Play scores are shown in Table 3.7. A Google Play score is computed based on both the rating and the number of downloads of an app. The number of downloads for the set of apps per user is normalized to a range between 1 and 5 as described above.

<b>App</b>	<b>Google Play Score</b>
Smart Voice Recorder	2.8
Facebook	4.5
InBrowser	2.6
Smurfs' Village	2.85
Duolingo	2.9
Shazam	3.7
Goal.com	2.51
WhatsApp	4.7
Viber	3.65
PowerTutor	2.5

Table 3.7: User 1 Google Play scores

<b>App</b>	<b>CrowdApp Score</b>	<b>Google Play Score</b>
Clash of Clans	3.60	3
Lebfiles	2.10	2.4
Metro	2.44	2.1
No Emoji Ninja Dies	1.77	2.4
WhatsApp	4.20	4.7

Table 3.8: User 2 scores

A Google Play score is then computed by averaging the download score with an app's market rating.

In order to test if the rest of the collected data show correlation with Google Play scores, we used the above weights to calculate the CrowdApp score per user per app. CrowdApp and Google Play scores are listed per app for each user in Tables 3.8, 3.9, and 3.10.

Let  $n$  be the number of apps per user, and vectors  $\mathbf{X}$  and  $\mathbf{Y}$  be the CrowdApp

<b>App</b>	<b>CrowdApp Score</b>	<b>Google Play Score</b>
Instagram	3.04	2.8
Facebook	3.57	4.5
WhatsApp	3.60	4.7
Messenger	2.60	2.6

Table 3.9: User 3 scores

<b>App</b>	<b>CrowdApp Score</b>	<b>Google Play Score</b>
Alfa	1.79	2.55
Viber	4.15	3.65
WhatsApp	4.03	4.7
Opera Mini	2.99	3.7

Table 3.10: User 4 scores

<b>User</b>	<b>Correlation Coefficient - r</b>
User 2	0.87
User 3	0.95
User 4	0.83

Table 3.11: Correlation with Google Play

scores and the Google Play scores of the users' apps, respectively. The correlation coefficient is then defined as:

$$r = \frac{n(\Sigma XY) - (\Sigma X)(\Sigma Y)}{\sqrt{[n\Sigma X^2 - (\Sigma X)^2][n\Sigma Y^2 - (\Sigma Y)^2]}} \quad (3.15)$$

For every user, we calculated the correlation coefficient based on their scores. The results are shown in Table 3.11.

The correlation coefficient is at least 0.83 in all cases, which means that there is a very strong positive relationship between CrowdApp scores and Google Play scores.

Interviews were conducted after collecting the data from the users. Users were asked to rate their apps on a scale from 1 to 5. User 2 did not provide a rating for Lebfiles stating that he rarely uses it so it wouldn't be fair to rate it. This shows the importance of CrowdApp in that it takes the usage time of an application into consideration when determining its overall score. So in this case, even though User 2 was not able to provide what he believed was a proper

<b>App</b>	<b>CrowdApp Score</b>	<b>Subjective Rating</b>
Clash of Clans	3.60	5
Lebfiles	2.10	-
Metro	2.44	5
No Emoji Ninja Dies	1.77	2
WhatsApp	4.20	5

Table 3.12: User 2 scores

rating for Lebfiles, CrowdApp managed to do so. The user was pleased with all the other apps except with No Emoji Ninja Dies. According to him, it is a very nice app, however, there are plenty of ads that keep popping, and this worsened his experience with the app. He said a proper rating would be 2 out of 5 in the best case. CrowdApp gave No Emoji Ninja Dies a score of 1.77.

User 1 commented that WhatsApp crashed twice while he was using it. User 3 also said that Facebook Messenger crashed once, and in general, he was not pleased with the app. He therefore gave it a rating of 1 out of 5. In this case, both Users 1 and 3 happened to be experts in this field. However, in most cases, users don't recognize when an app crashes, especially when it is not clearly mentioned in a dialog box that says: "Unfortunately, the app has stopped". In these two cases, CrowdApp detected that WhatsApp crashed twice with User 1 and Messenger crashed once with User 3.

As for User 4, he said that he would give all his apps 5 stars except for Alfa because it is "very bad". He only uses it to send free SMS messages, and that is why he opens it occasionally.

In Tables 3.12, 3.13, and 3.14 we show CrowdApp scores along with the users' subjective ratings for each of their apps.

For every user, we calculated a correlation coefficient again, this time between CrowdApp scores and the user's subjective opinions. Results are in Table 3.15.

<b>App</b>	<b>CrowdApp Score</b>	<b>Subjective Rating</b>
Instagram	3.04	3
Facebook	3.57	4
WhatsApp	3.60	2.5
Messenger	2.60	1

Table 3.13: User 3 scores

<b>App</b>	<b>CrowdApp Score</b>	<b>Subjective Rating</b>
Alfa	1.79	2
Viber	4.15	5
WhatsApp	4.03	5
Opera Mini	2.99	5

Table 3.14: User 4 scores

The correlation coefficient is at least 0.75 in all three cases. This means that there is also a very strong positive relationship between CrowdApp scores and users' own subjective opinions.

## Discussion

When downloading an app from the market, the first thing that most users consider is the app's rating, and some of them even check the reviews. However, for these ratings and reviews to be fair and accurate, they must be numerous for statistical significance, and they both require user input. CrowdApp's utility score, on the other hand, does not involve user input and can be computed quickly. We have shown that with only a few data points, we are able to get utility scores that correlate highly with the market ratings and number of downloads. This

<b>User</b>	<b>Correlation Coefficient - r</b>
User 2	0.75
User 3	0.77
User 4	0.88

Table 3.15: Correlation with user ratings



objective rating can be continuously updated, and the more users use an app or the more time it is used, the more accurate is the given rating. In a sense, what CrowdApp is doing is crowdsourcing the “opinions” of the different users without their direct participation.

The results that we obtained are promising. After using the optimal weights calculated from the data of User 1, the CrowdApp utility scores of the three other users showed a correlation with Google Play scores that is higher than 83%. This means that there is a very strong positive relationship between the two sets of scores. In addition to these results, the users who installed CrowdApp on their devices were interviewed at the end of the experiment period. Their comments on the performance of the apps installed on their phones agreed with the results that we collected using CrowdApp. The correlation in this case was higher than 75%. In conclusion, CrowdApp v1.0 was shown to rate apps based on actual user experience without any direct participation from the users themselves.

### **3.4.2 Behavior Score Results**

In all iterations of the game, the profits of SYS, a normal app, and a malicious app are updated based on actions chosen by players. For different ranges in input values, the below output values are of interest:

- Percentage of convergence of apps that were correctly flagged by SYS in a specific period (PCA)
- Percentage of convergence of authoritative users that were correctly identified by SYS in a specific period (PCU)
- Average profit of SYS per round ( $\mathcal{AS}$ )

Variable	$C_A$	$g_a$	$G_A$	$L_A$	$C_D$	$g_n$	$G_N$	$L_N$	$\ell_m$	$G_M$	$L_M$
Suggested Value	4	5	10	10	2	6	6	6	8	8	8

Table 3.16: Suggested values for gains, costs, and losses

- Average profit of a normal app per round ( $\Lambda_N$ )
- Average profit of a malicious app per round ( $\Lambda_M$ )
- Percentage of apps that were tested (PA)
- Percentage of users that were queried (PU)
- ROC curves for app and user classification

In all the experiments that follow, the base values from Table 3.3 are used. Any use of values other than the base values is clearly stated in each section. In addition, the values for the gains, costs, and losses that were used in the simulations are shown in Table 3.16. These values abide by the conditions defined in the game model in subsection 3.1.2. Unless otherwise stated, every experiment was repeated 50 times and the results were averaged over the repetitions.

### Running Time

In this experiment, we wanted to study the relationship between the number of rounds and the output values. We varied the number of rounds from 10 (5 minutes) to 2880 (24 hours) in steps of 10. The rest of the variables were kept at their base values in Table 3.3. The experiment was performed only once. The results are shown in Figure 3.7.

We notice that the average profits of SYS and normal apps are positive. The profit of a normal app is stable at 3 and that of SYS stabilizes at around 1.4.

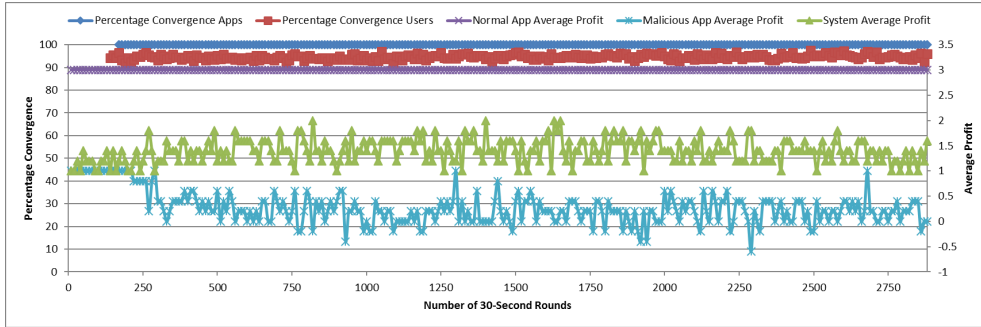


Figure 3.7: Running time

The average profit of malicious apps is less than that of SYS and normal apps. It stabilizes at around 0.4. Slightly more than two hours (250 rounds) were enough for the average profit of a malicious app to become smaller than that of SYS. This tells us that from the first couple of hours after installing CrowdApp and starting the system, on average, malicious apps are benefiting less than normal apps and the system. It is worth noting that these profit values are not indicative in the absolute but their relation to one another is important.

As for convergence values, they both stabilize after almost 150 rounds (1+ hour). The percentage convergence of apps stabilizes at 100% whereas that of users stabilizes at around 95%. This tells us that about an hour is enough for our system to correctly detect the malicious apps it checks and accurately identify users it queries based on their input.

### Apps vs. Users

In this experiment, we wanted to study the effect of increasing the number of users relative to the total number of apps. We varied the number of users with CrowdApp on their devices from 10 to 500. The number of apps was set to 1000. Other variables were also kept at their base values in Table 3.3. Results are shown in Figure 3.8.

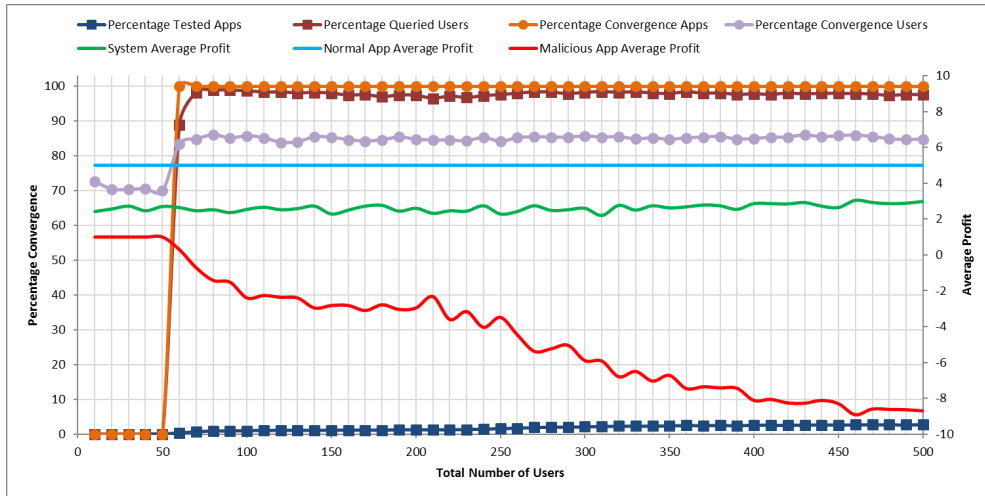


Figure 3.8: The change in output values as the total number of users in the system increases

First, we notice that for all values of the total number of users less than 50, there are no values of PCA and PCU. This is because of the minimum crowd size limit that we imposed on the system. For values less than 50, we notice that the average profit of SYS and that of a malicious app are almost the same which is not desirable. As the total number of users increases, we notice that this difference between the average profits of SYS and a malicious app is continuously increasing. For a total number of users greater than 200, the average profit of a malicious app decreases to the extent that it becomes negative. For our chosen base value of 200, the average profit of SYS and a normal app is much higher than that of a malicious app. This tells us that the system does not require a very large number of users to ensure that malicious apps are not gaining as much as the system and normal apps.

As for the convergence values, starting from 50 users only, PCA stabilizes at 100% and PCU stabilizes at around 85%. The results from this experiment show that the system does not require a very large number of users that are giving their input in order for it to properly detect apps. This is a promising result. It

means that our system can be applied in many scenarios with a small number of participants. For example, it can be applied in a small company with 200 employees and still be expected to give good results.

It is clear that as TNU increases, PA increases as well. It eventually reaches around 2.7%. Even though PA is very small, as shown in Table 3.4, the percentage of apps downloaded more than 500,000 times on Google Play is only 2%. Assuming that these apps are the ones that are being tested by our system every time, which is a very reasonable assumption, then even though a large percentage of apps is not being tested, however these apps are found on a small number of user devices and therefore their damage (if it exists) will not be as significant as the small portion of apps in the higher 2%. This increase in PA is expected since as TNU increases, the number of downloaded apps will increase and these apps will issue events and when they do, more of them will fire an alarm and end up being tested since there are more users who have installed them. As for PU, it instantly stabilizes at around 98% starting from TNU greater than 50 which is the  $CS_{\min}$  value.

The above results are for 1000 apps. We cannot assume that if 50 users are enough to reach convergence in the case of 1000 apps, then they will be enough to reach convergence when the number of apps increases to 10,000 for example. We performed another experiment. Our aim in it was to obtain the needed number of users in order to reach stability in both app and user convergence for a varying number of apps. The number of apps was varied from 1000 to 10,000. The number of users was varied from 50 to 10,000. The experiment was repeated 5 times. Average results are shown in Table 3.17. The second column shows the average required user count (RUC) to reach high convergence rates for apps (> 95%).

<b>Apps</b>	<b>RUC</b>	<b>Users/Apps Ratio</b>	
1000	60	0.06	
2000	60	0.03	
4000	80	0.02	
6000	110	0.02	
8000	150	0.02	
10000	180	0.02	
<b>Apps</b>	<b>EUC</b>	<b>RUC</b>	<b>Error</b>
20000	314	320	2%

Table 3.17: Convergence rates

The trend line based on the computed values from the top half of the table is  $RUC = 0.014 \times Apps + 34$ . If this value is used to estimate RUC when the number of apps doubles (20,000), the estimated user count (EUC) will be 314 as shown in the second column. In the second phase of this simulation, the experiment was repeated for 20,000 apps to validate the extrapolated result. The result is shown in the third column. The fourth column shows the percentage error between the extrapolated and computed values of the required number of users for 20,000 apps. It is around 2% which is reasonable. We notice that as TNA increases, the number of required users to reach convergence increases. In general, the number of users needed to converge is relatively small. And based on the trend line, it can be assumed that the ratio of users to apps is usually around 0.02. So given 100,000 apps, 2,000 users should be enough to reach convergence. This supports our conclusion that our system can be used in small settings irrespective of app count.

### **Crowd Expertise**

In this experiment, we wanted to study the relationship between crowd expertise and the output values. We varied the expertise from 0 to 1 in steps of 0.1. Other variables were kept at their base values in Table 3.3. Results are in Figure 3.9.

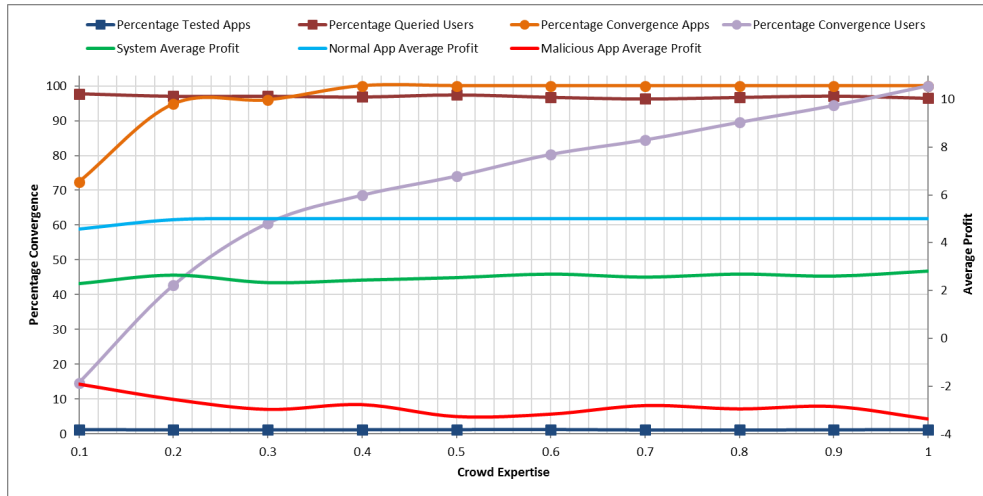


Figure 3.9: The change in output values as the crowd expertise increases from 0.1 to 1 in steps of 0.1

We notice that for very low values of the crowd expertise, the average profits of SYS and a malicious app are very close in value which is not desirable. But as the crowd expertise increases, the difference between the average profit values of SYS and a malicious app increases as well. This result is expected of course. The higher the number of experts in the crowd, the better is the performance of our system and the lower the profits of a malicious app. For all values of the crowd expertise that are higher than 0.1, the average profits of SYS and a normal app are higher than that of a malicious app.

We notice a similar pattern in the convergence plots. Both app and user convergence values increase with the increase in crowd expertise. For expertise values of 0.4 or higher, app convergence reaches 100% and user convergence reaches around 70% and this has a very good indication. It shows that our system can be deployed in many settings even with low expertise levels of the crowd. In reality, we expect the crowd expertise to be higher than 0.5 since the entire idea behind our system is that it can trust the wisdom of the crowd when collecting their input. A crowd where more than half of the participants are providing random

input is not the standard case. If we are to trust the wisdom of the crowd, we can safely assume that the expertise value is at least 0.7 which is the base value that we chose in Table 3.3.

PA is not changing with the change in expertise. It is stable at around 1.2%. PU on the other hand is slightly decreasing with the increase in crowd expertise. This agrees with our design requirement which states that when expertise values are still unknown, the system queries a larger portion of users. When expertise levels become known, the system will query a smaller portion of selected authoritative users. What the above plot shows is somewhat similar. When the crowd expertise is small, the system will have to query a larger portion of the crowd for it to trust their aggregated reply. When the overall expertise is high, it is sufficient for the system to query a smaller portion of the crowd (the authoritative users). This is reflected in the slight decrease in PU. Of course for all values of the crowd expertise, PU is very high ( $> 90\%$ ).

### **Fraction of Malicious Apps**

In this experiment, we wanted to study the relationship between the fraction of malicious apps and the output values. We varied the fraction from 0 to 1 in steps of 0.05. Other variables were kept at their base values in Table 3.3. Results are shown in Figure 3.10.

We notice that as the fraction of malicious apps increases, the percentage convergence of users increases significantly. It eventually reaches 99% which is very close to the stabilized convergence value of apps. This is because as the number of malicious apps increases, the probability of defending against an app also increases. This means that more users are being queried throughout the experiment. The more our system queries users, the more accurate is their identifi-



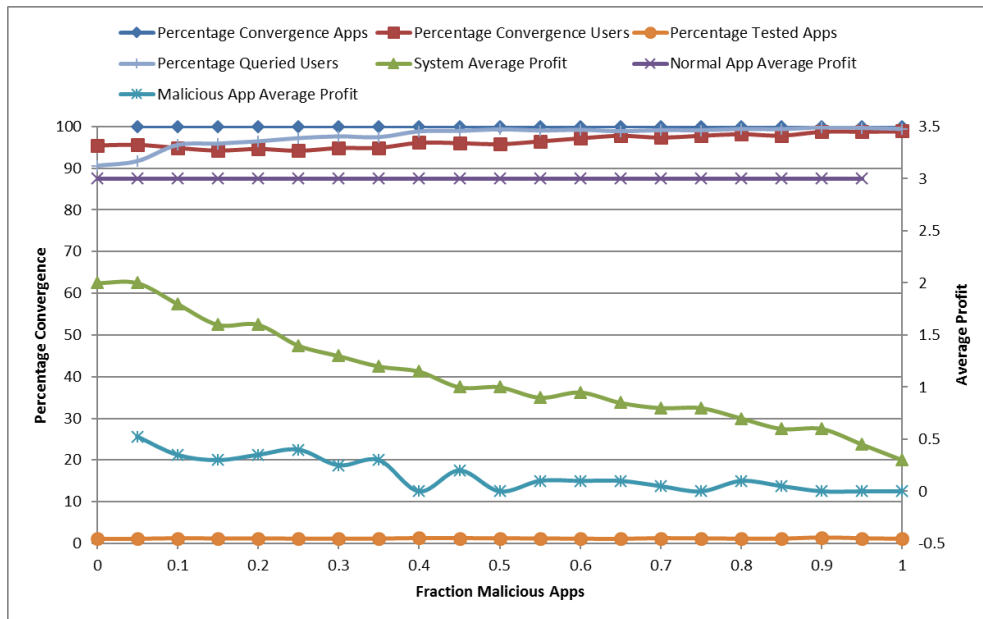


Figure 3.10: The change in output values as the fraction of malicious apps increases from 0.1 to 1

cation. However, for all values of the fraction of malicious apps, the convergence of both apps and users is very high.

This increase in the percentage convergence of users is also reflected in the average profit of a malicious app. As more experts are identified, the probability of detecting malicious apps increases which means that the losses of malicious apps increase as well. This can be seen in the plot. We notice that as the fraction of malicious apps increases, the average profit of a malicious app decreases until it stabilizes at 0. The average profit of the system also decreases with the increase in the fraction of malicious apps. However, for all values of the fraction, the average profits of both SYS and a normal app are always higher than that of a malicious app.

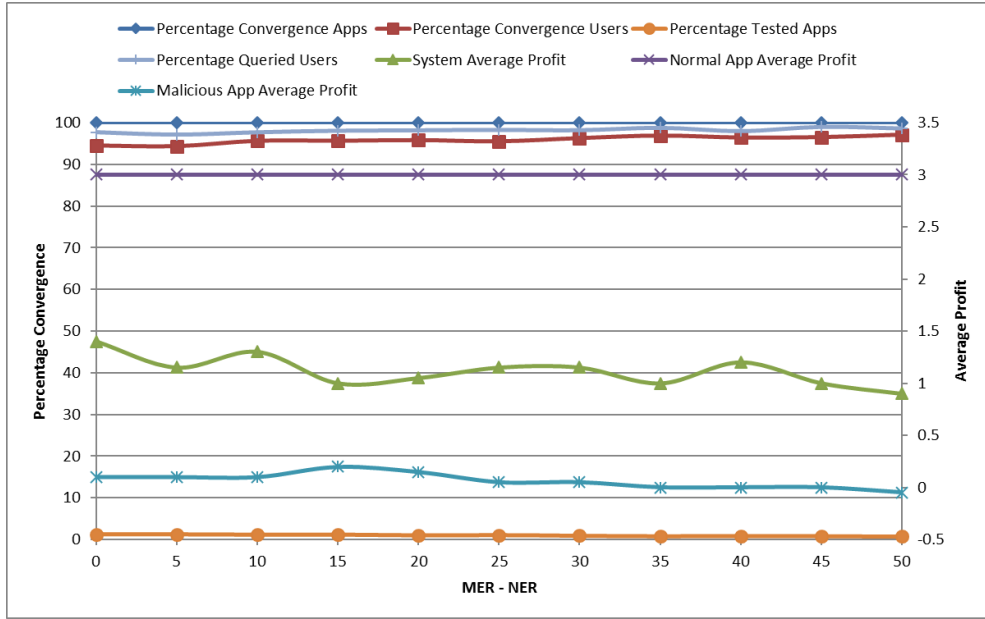


Figure 3.11: The change in output values given selected differences between malicious and normal event rates

<b>MER</b>	50	50	50	60	60	65	65	70	70	75	75
<b>NER</b>	50	45	40	45	40	40	35	35	30	30	25
<b>Difference</b>	0	5	10	15	20	25	30	35	40	45	50

Table 3.18: Malicious and normal app event rates

### Event Rate Analysis

In this experiment, we wanted to study the effect of changing the normal and malicious event rates on the output values. In the other experiments, we were giving both normal and malicious apps the same event rate so as not to differentiate between the two types of apps. This choice of event rates is considered a worst case scenario where normal apps issue events with the same frequency as malicious ones. The difference between malicious and normal event rates was increased from 0 to 50. The chosen rate values are shown in Table 3.18. The rest of the variables were kept at their base values in Table 3.3. The results are shown in Figure 3.11.

We notice that as the difference between the event rates of malicious and normal apps increases, the identification of users based on their expertise levels is slightly improved. The percentage convergence of users stabilizes at around 96%. This result is of course expected since when this difference increases, it means that malicious apps are on average issuing more events than normal apps. This means that our system will defend against them more often and will end up querying more users in the process. The percentage convergence of apps also stabilizes here at 100%.

The increase in the percentage convergence of users is also reflected here in the decrease in the average profit of a malicious app which stabilizes at 0. We notice that regardless of the difference in event rates, the average profits of SYS and a normal app are always higher than that of a malicious app. The plots of SYS and a malicious app are almost opposites. This is expected. We notice that with the increase in difference in event rates, the difference between these two plots slightly increases as well.

This is a very important result. It means that even if all the normal apps in the system issue the same number of events on average as the malicious apps, most of them will still not be misclassified. In all other experiments, we assumed both event rates to be equal so as to study the worst case scenario. Identification of both apps and users remains very good even in the worst case.

### **System Defense Cost**

The relationship between SYS's defense cost ( $C_D$ ) and the output values was also analyzed.  $C_D$  was varied from 0 to 6 (which is the base value of the gain from increasing the reputation of a normal app,  $G_N$ ) in steps of 0.1. The remaining variables were kept at their base values. Results are shown in Figure 3.12. We

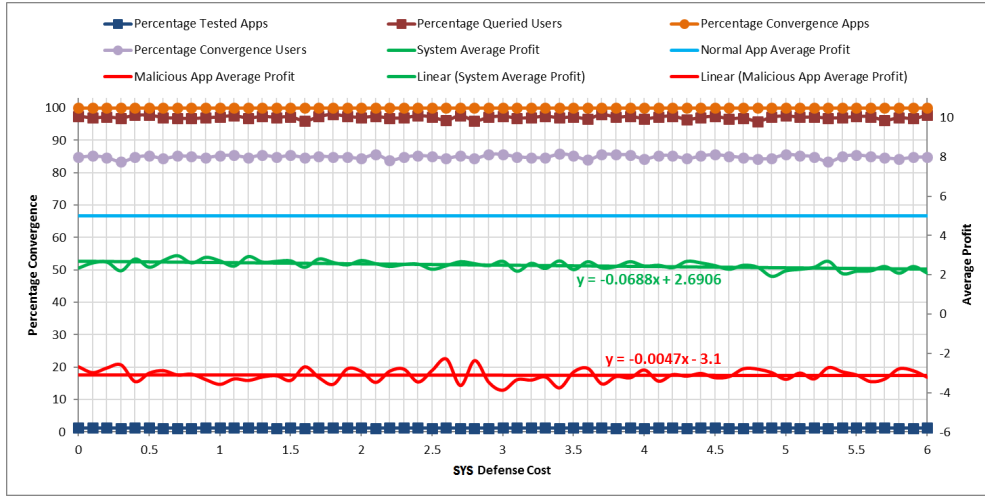


Figure 3.12: The change in output values as SYS's defense cost increases from 0 to 6 in steps of 0.1

notice that for all values of  $C_D$ , PA is around 1.2%, PU around 97%, PCA around 100%, PCU around 85%, and  $\Lambda_N$  is around 5. The output values that are changing with the change in cost are  $\Lambda_S$  and  $\Lambda_M$ . The profits of both SYS and a malicious app are decreasing. The rate of decrease of  $\Lambda_S$  however is larger than the rate of decrease of  $\Lambda_M$  as illustrated by the slopes of the trend lines shown in the figure.

If we consider the profit of SYS alone, it will become zero at the cost value where the trend line intersects with the x-axis ( $-0.0688 \times C_D + 2.6906 = 0 \Rightarrow C_D \approx 39$ ). Given the restriction that  $C_D$  should always be less than  $G_M$  for SYS to have an incentive to defend, we know that the case where SYS's profit becomes zero will not occur. To find the value of  $C_D$  at which the profits of SYS and a malicious app become equal, we simply set the equations of the trend lines for  $\Lambda_S$  and  $\Lambda_M$  equal to each other ( $-0.0688 \times C_D + 2.6906 = -0.0047 \times C_D - 3.1 \Rightarrow C_D \approx 90$ ). Similarly, with the constraint on  $C_D$  being at most equal to  $G_M$ , the above case where the profits of SYS and a malicious app become equal will not occur.

These results tell us that regardless of the value of our system's defense cost, as long as it is kept lower than the gain that it achieves when it decreases the reputation of a malicious app ( $G_M$ ), then its profit will always be larger than that of a malicious app given the rest of the base values that were considered in the experiments.

### ROC Curve Analysis

ROC curves of app and user classification are shown in Figure 3.13. The discrimination threshold is the crowd expertise, CE, which was varied from 0.05 to 0.95 in steps of 0.05. The circled values for apps are those farthest away from the random guess. We notice that for values of CE larger than 0.35, the classification of apps as normal and malicious becomes very good. In the case of user detection, the results are not similar. We notice that for values of CE around 0.6, the TPR of users increases and the FPR is still relatively small. However, as we further increase CE, the FPR of user detection starts to increase significantly. This result is reasonable. As CE increases, the number of unreliable users in the system decreases. A decrease in the total number of negatives will increase the value of FPR. For a very large value of CE (0.95), the total negatives become very small to the extent that FPR reaches around 0.62 which is quite large. It is worth noting that this large value of FPR does not necessarily imply a large value of false positives but rather a very small number of total negatives. In terms of user detection, our previous experiments have shown that the percentage of convergence of users reaches around 85% when the base values are used. This is an acceptable result keeping in mind that the aim behind the system is to classify apps rather than users. The detection of users is a by-product of the system. The results from this experiment agree with previous results where it was shown that

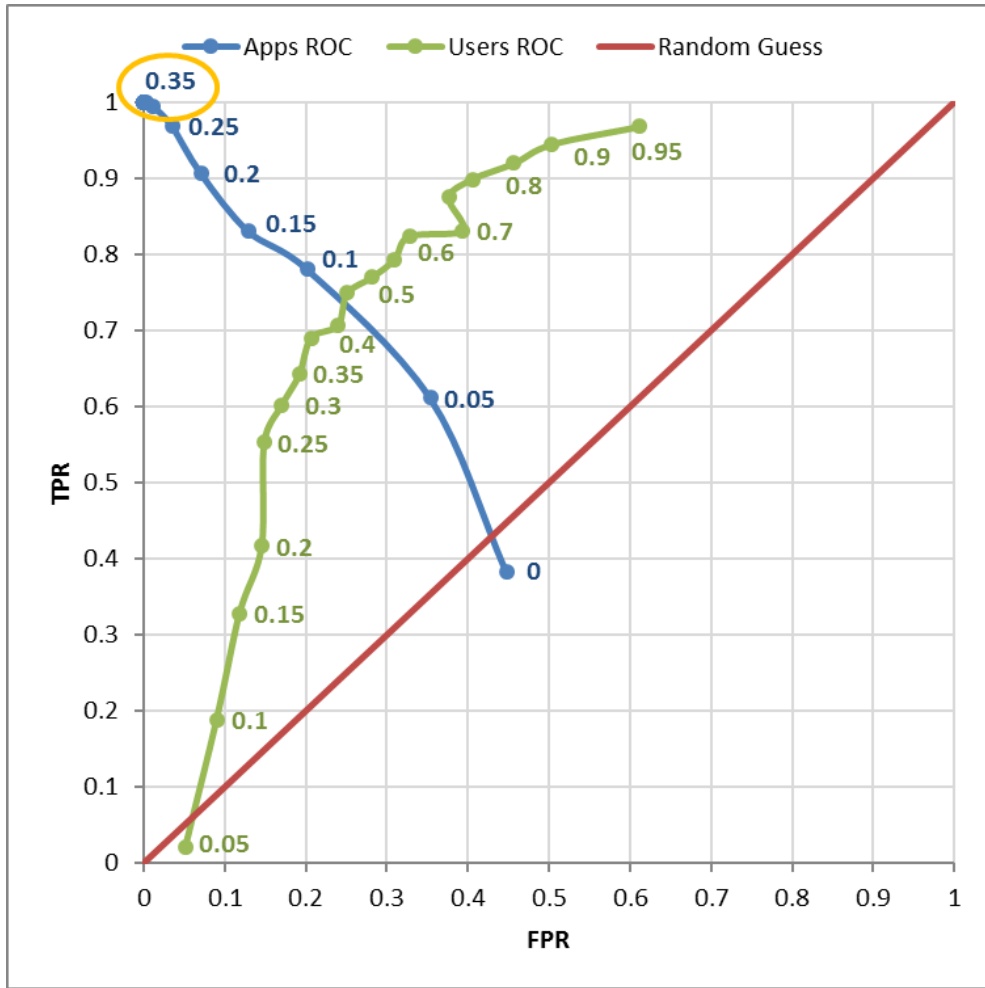


Figure 3.13: ROC curves of app and user classification

the classification of apps by our system is slightly better than that of users.

### Real-Life Scenario

According to [104], Android apps that have been downloaded over 50,000 times amount to 120,000+. The number of high-speed Internet subscribers of the Canadian operator TELUS is 1.5 million [108]. If only 1% of these subscribers install CrowdApp on their devices, TNU in this scenario will be 15,000 (more than TNA). As our results show in Table 17, a maximum ratio of 0.02 users per apps

is enough to reach PCA and PCU stabilization and to ensure that profits of SYS and normal apps are higher than those of malicious apps. Based on the computed trend line, required users are  $0.014 \times 120000 + 34 \approx 1714$ . It is hence safe to assume that 15,000 users are enough to classify 120,000+ apps.

As for CE, our assumption is that our system can trust the wisdom of the crowd, otherwise, the concept of crowdsourcing wouldn't have proven to be useful in other applications. This means that the CE value of 0.7 that was used in the simulations holds in this scenario. Our system can be employed in small settings such as within a portion of an operator subscriber base where a relatively small crowd size is enough for it to detect malicious apps, flag them, and classify users as authoritative and unreliable, thus leading to better system performance.

## **Discussion**

The base value of the number of rounds was set to 1,440 which are around 12 hours. However, our results show that in only a few hours, our system was able to detect most apps as well as most users that it queried. The base value we chose for the crowd expertise was 0.7. Our assumption is that we should be able to trust the wisdom of the crowd. A higher expertise value will give better and faster results as was shown in previous sections. If, for example, TELUS has its 40,000 employees install CrowdApp on their devices and chooses only 5,000 users from its subscriber base, then it can reach a very high overall expertise value (close to the chosen base value) assuming all of its employees are considered to be authoritative. This improves system performance significantly. Also, 45,000 users will be more than enough for the system of 120,000+ apps to converge.

The fraction of malicious apps was set to 0.25, which according to the literature is a correct estimation of malicious app distribution in the Android market.

In reality, this value is not an input that is controlled by an operator such as TELUS. A similar thing can be said regarding malicious and normal event rates.

In our experiments, on average, both types of apps were issuing events once every minute. This high rate of events might not be realistic but was set in order to speed up our simulations. In reality, an app might issue an event once every hour. If our results showed that our system converges after 12 hours, in reality it might reach convergence after a few days. Detecting all malicious apps after one month is still considered a very good achievement.

Generally, in all of our experiments, PA averaged around 1.2% whereas PU averaged around 97%. A small number of apps are tested in every experiment as a result of the download distribution of apps from the Android market. The most popular apps downloaded by users ( $> 500,000$  times) constitute only 2% of the total number of apps on the market as was discussed before. Therefore, an average PA around 1.2% is reasonable. PU on the other hand was always high. And from the queried users, the majority of them are being correctly classified by our system.

Our ROC curve analysis for both app and user classification supported the rest of the results by showing for what values of the crowd expertise our system achieves its best classification results. In the case of user detection, there is a trade-off when choosing the optimal value of crowd expertise. For higher values, the FPR of user detection increases. But since app classification is the main goal behind our system, we can conclude that our base value of 0.7 will result in very good app and user classification.

Having modeled the interaction between our system and a target app using a game-theoretic approach, we shift our focus to the process of aggregating received replies from users for the purpose of computing the behavior score. There are



several techniques that can be used to aggregate user replies and we discuss some of the most common ones in the following chapter. We also provide mathematical models for these techniques with the aim to compare their performance given different system variables.

## Chapter 4

# Probabilistic Models for Aggregation in Voting Systems

There is a large number of techniques that can be used to aggregate replies from users in the behavior score computation process. Some of these techniques involve reported user confidence. Such techniques cannot be modeled without considering the effect of the Dunning-Kruger bias on user replies. Therefore, we begin this chapter by modelling the Dunning-Kruger psychological bias that will be used throughout the rest of this dissertation. We then model the aggregation methods that were mentioned earlier: Plurality Voting (PR), Confidence-Weighted (CF), Maximum Confidence (MC), and Competence-Weighted (CP). Our main objective in this chapter is to provide the necessary conditions for the input variables in our system that will render one method superior to the others. We test our findings on a crowdsourced dataset and present a discussion of the main results and conclusions that can be drawn from this work.

## 4.1 Modeling the Dunning-Kruger Effect

The DunningKruger effect is a cognitive bias in which low-ability individuals suffer from the illusion that their abilities are higher than what they really are. This effect was presented in a renowned study in 1999 by psychologists David Dunning and Justin Kruger who attributed this illusion of superiority in low-ability individuals to their meta-cognitive weakness in accurately evaluating their own competence. In other words, they have the propensity to overvalue their abilities when solving a task or answering a question. The study also shows how high-ability individuals generally underestimate their competence [23].

With this idea in mind, we modeled the relationship between competence and confidence, as presented in their study entitled “Unskilled and Unaware of It: How Difficulties in Recognizing One’s Own Incompetence Lead to Inflated Self-Assessments”, as a continuous function. To the best of our knowledge, modelling this psychological effect has not been done before. The function is estimated by a quadratic equation that is concaved upwards. The choice of an upward concaved function matches properly the decrease-increase pattern in confidence levels as a respondent’s competence increases.

$$\mathbf{y}_r = \mathbf{a}x_r^2 + \mathbf{b}x_r + \mathbf{c} \quad (4.1)$$

The subscript  $r$  refers to the respondent,  $\mathbf{y}_r$  represents the respondent’s confidence level,  $x_r$  represents his competence level, and the coefficients  $\mathbf{a}$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  determine the resulting shape of the competence-confidence plot, which we will refer to from now on as the DK plot or DK function after Dunning and Kruger. Figure 4.1 shows several samples of the DK plot. Depending on the type of crowd,

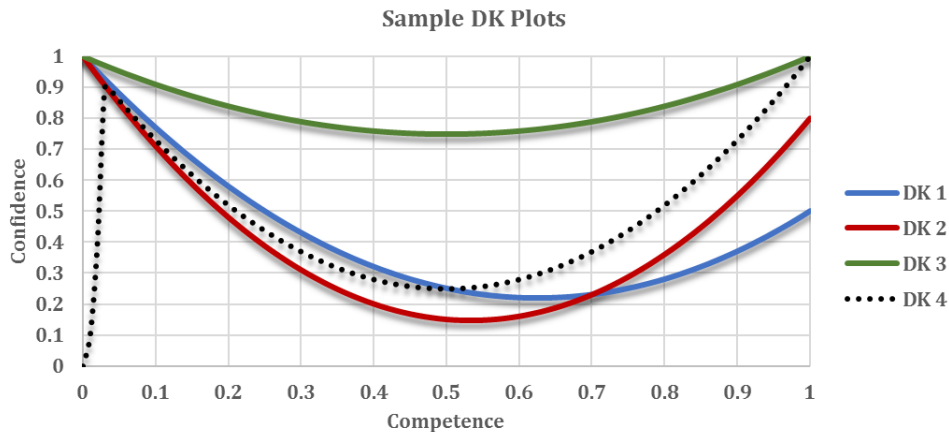


Figure 4.1: Four sample DK plots

one DK plot might be a better representation than another. Note that DK 4 is different from the other plots in that it starts with a straight line and continues with a parabola. This is another possible representation of the Dunning-Kruger effect. It shows how when a respondent knows nothing about a topic, his confidence level is very low. However, upon basic introduction to it, for example after skimming through a Wiki page, his competence increases slightly while his confidence explodes. Intuitively speaking, representing the Dunning Kruger effect as a piecewise function as in DK 4 is a more general representation. However, it complicates the mathematical analysis significantly and has minor effect on the obtained results. We choose to adopt a simplified general model of this effect as represented in DK 1, DK 2, and DK 3.

Choosing to model the DK effect with a polynomial function serves two purposes. First, as will be seen throughout the work, it will help us formally prove the superiority of certain aggregation techniques over others for different task difficulty levels. Second, it allows us to model a wide variety of crowd types. The DK effect was studied in settings where both competent and incompetent workers exist. There are scenarios where the crowd is mostly competent or mostly incom-

petent and in those cases, the shape of the polynomial will differ. Throughout this work, we will follow the assumption of a general crowd that has workers of all competence levels. However, it is important to keep in mind that DK1, DK2, and DK3 shown below do not accurately represent crowds in every case. That is where the role of the coefficients  $\mathbf{a}$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  comes into play.

It is worth noting that the Dunning-Kruger effect is at the heart of many studies on illusory superiority. People who noticeably lack in areas of logical reasoning, emotional intelligence, grammar, financial knowledge, math, chess, fairness, job skills, driving abilities and other disciplines have the tendency to rate their expertise almost as favorably as actual experts do [23, 109, 110, 111, 112, 113]. It is the prevalence of this effect throughout various disciplines that motivated us to formally model it in our work.

#### 4.1.1 Function Constraints

We define constraints on the shape of the DK function that are derived from characteristics of this psychological effect as described by Dunning and Kruger. Both confidence and competence values are in the range  $[0,1]$ :

- Respondents with the lowest competence have maximum confidence in their abilities, which means that the function starts at the point  $(0, 1)$ . From here on out, we will set the value of the coefficient  $\mathbf{c}$  to 1.
- Confidence levels decrease then increase as competence levels become higher, which means that the function is concaved upwards. From now on, we note that the coefficient  $\mathbf{a}$  should be positive.
- Confidence levels cannot be lower than 0, which means that the point of minimum confidence either lies on the x-axis or is above it. This is achieved

by setting a constraint on the function to have at most one real root.

$$b^2 - 4a \leq 0 \Rightarrow -2\sqrt{a} \leq b \leq 2\sqrt{a}$$

- On the other extremity of the function, we note that respondents with maximum competence should also have a confidence value anywhere between 0 and 1.

$$x_r = 1 \Rightarrow 0 \leq y_r \leq 1 \Rightarrow -a - 1 \leq b \leq -a$$

Combining the last two conditions gives us the range of allowed values for  $b$  given  $a$ , which only applies for  $0 \leq a \leq 4$ :

$$-2\sqrt{a} \leq b \leq -a$$

*Dunning-Kruger psychological effect model.* A quadratic function representing the relationship between a participant's confidence ( $y_r$ ) and competence ( $x_r$ ) where  $y_r = ax_r^2 + bx_r + 1$  given that  $0 \leq a \leq 4$  and  $-2\sqrt{a} \leq b \leq -a$ .

### 4.1.2 Competence Model

We define respondent  $r$ 's competence as the likelihood of him or her solving a task  $k$  correctly. It is given by the Rasch psychometric measurement model, which states that the competence  $x_r^k$  of a respondent  $r$  solving a task  $k$  is a function of the respondent's ability  $\theta_r$  and the task difficulty  $\Delta_k$  [114, 115].

$$x_r^k = f(\theta_r) = \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} \quad (4.2)$$

The competence increases with the increase in the respondent's ability and decreases with the increase in task difficulty. The ability  $\theta_r$  is a value between 0 and 1. We define this value to be normally distributed with mean  $\mu$  and standard deviation  $\sigma$ . The task difficulty  $\Delta_k$  is also a value between 0 and 1. However, in our model, we do not consider tasks with zero difficulty levels. We assume that even the easiest task will require some time to complete, i.e.  $\Delta_k \in (0, 1]$ .

Taking the Rasch competence model into account, the DK function can then be defined in terms of respondent's ability, task difficulty, and the two coefficients  $\mathbf{a}$  and  $\mathbf{b}$ .

$$\text{DK} = y_r^k = \mathbf{a} \left( \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} \right)^2 + \mathbf{b} \left( \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} \right) + 1 \quad (4.3)$$

**Expected Competence.** The expected value of a respondent's ability ( $\theta_r$ ) is the ability mean  $\mu$ . To get the expected value of the competence ( $x_r^k = f(\theta_r)$ ), we first refer to Jensen's inequality, which states that for a convex function  $f(\theta_r)$ ,  $\mathbb{E}[f(\theta_r)] \geq f(\mathbb{E}[\theta_r])$ , and for a concave function,  $\mathbb{E}[f(\theta_r)] \leq f(\mathbb{E}[\theta_r])$ . As defined above, the shape of the competence function according to the Rasch measurement model depends on the value of the task difficulty ( $\Delta_k$ ). For  $0 < \Delta_k \leq 0.5$ ,  $f(\theta_r)$  is concave, which gives us an upper limit  $f(\mathbb{E}[\theta_r])$  on the competence expected value. For  $0.5 \leq \Delta_k \leq 1$ ,  $f(\theta_r)$  is convex, which gives us a lower limit  $f(\mathbb{E}[\theta_r])$  on the competence expected value.

$$f(\mathbb{E}[\theta_r]) = \frac{\mu(1 - \Delta_k)}{\mu(1 - \Delta_k) + \Delta_k(1 - \mu)} \quad (4.4)$$

$$\begin{cases} 0 < \Delta_k \leq 0.5 & \mathbb{E}[f(\theta_r)] \leq f(\mathbb{E}[\theta_r]) \\ 0.5 \leq \Delta_k \leq 1 & \mathbb{E}[f(\theta_r)] \geq f(\mathbb{E}[\theta_r]) \end{cases}$$

The probability of one participant choosing the best option among several options is equal to the expected value of the competence. In a crowdsourcing context however, there are many participants. The probability of their aggregate answer being correct depends on the aggregation method. Modeling this probability is one of the main contributions in this dissertation.

### 4.1.3 Notes

Given the modelling constraints presented above, there are some observations that are worth mentioning:

- Setting both coefficients in the DK function equal to 0 is equivalent to aggregating by plurality.

$$\mathbf{a} = \mathbf{b} = 0 \Rightarrow \mathbf{y}_p = 1 \quad \forall \mathbf{x}_p$$

- An extreme case is when the confidence level strictly decreases as the competence increases.

$$\mathbf{a} = 1 \quad \text{and} \quad \mathbf{b} = -2\sqrt{\mathbf{a}} \quad \text{or} \quad \mathbf{a} = 0 \quad \text{and} \quad \mathbf{b} = -1$$

- When the two coefficients  $\mathbf{a}$  and  $\mathbf{b}$  are equal in absolute value to  $\mathbf{m}$ , the confidence ( $\mathbf{y}_r$ ) starts at 1 for minimum competence ( $\mathbf{x}_r = 0$ ) and ends at 1 for maximum competence ( $\mathbf{x}_r = 1$ ) i.e., the plot is symmetric about the vertical line  $\mathbf{x}_r = 0.5$ . In addition, as  $\mathbf{m}$  increases, the point of minimum



confidence on the plot shifts downwards.

- For the same values of  $\mathbf{a}$  and  $\mathbf{b}$ , as  $\Delta_k$  increases, the point of minimum confidence shifts towards the right.
- Increasing  $\mathbf{a}$  has the effect of increasing the confidence faster for respondents of higher competence.
- Decreasing  $\mathbf{b}$  has the effect of decreasing the confidence faster for respondents of lower competence.

## 4.2 Modeling Aggregation

In any crowdsourcing system, the method used to combine the large number of collected replies into a single output is crucial in determining the success of the crowdsourcing process. Depending on many factors, most notable of which are related to the size and characteristics of the crowd, one approach may perform better than the others. In this section of the chapter, we provide a general model for the most popular aggregation methods focusing on how aggregation is performed and how the probability of reaching a correct answer changes with the change in the defined system properties.

### 4.2.1 Plurality

In PR voting, one of the most popular and simplest aggregation methods is used. The method selects the most voted for choice by the crowd as the final answer. The aggregated reply  $\ddot{\mathbf{a}}_q^{\text{PR}} \rightarrow \operatorname{argmax}_{\ddot{\mathbf{a}} \in \mathbb{A}} \sum_{r \in \mathbb{R}_q} \delta_{\ddot{\mathbf{a}}_q^r \ddot{\mathbf{a}}}$  for a question  $\mathbf{q}$  is the one that agrees most with replies given by all participating respondents where  $\ddot{\mathbf{a}}$  is each of the possible answers from the set of answers  $\mathbb{A}$ ,  $\ddot{\mathbf{a}}_q^r$  the answer for question  $\mathbf{q}$

given by respondent  $r$ , and  $\delta_{\ddot{a}_q^r \ddot{a}}$  is the Kronecker delta which returns 1 when  $\ddot{a}_q^r$  matches  $\ddot{a}$  and 0 otherwise.

In a crowdsourcing scenario with  $N_q$  participating respondents, the probability of success of more than half of them is equivalent to independently repeating the experiment  $N_q$  number of times. PR with  $N_q$  independent respondent decisions gives an overall correct answer that follows the binomial formula (equation 4.5) where  $\ddot{a}_q^*$  is the correct answer to question  $q$  and  $P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1) = \mathbb{E}[f(\theta_r)]$ .

$$P_q^{\text{PR}} = P(\ddot{a}_q^{\text{PR}} = \ddot{a}_q^*) = \sum_{i=N_q/2}^{N_q} \binom{N_q}{i} \left( P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1) \right)^i \left( 1 - P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1) \right)^{N_q-i} \quad (4.5)$$

For a large  $N_q$ , as is the case in most crowdsourcing systems, we can use the Chernoff bound to get a sharp bound. Let  $X$  be the number of respondents who answer correctly, we obtain a lower bound on  $P_q^{\text{PR}}$  shown in equation 4.6 that is only valid for  $P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1)$  greater than 0.5 ( $\mathbb{E}[f(\theta_r)] > 0.5$ ).

$$P_q^{\text{PR}} = P\left(X > \frac{N_q}{2}\right) = 1 - P\left(X \leq \frac{N_q}{2}\right) \geq 1 - e^{-\left(\frac{N_q}{2P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1)}\right) \left(P(\delta_{\ddot{a}_q^r \ddot{a}_q^*} = 1) - \frac{1}{2}\right)^2} \quad (4.6)$$

Requiring the success probability of an average respondent to be greater than 0.5 reminds us of Condorcet's Jury Theorem on voter competence; a very popular theorem in the field of social choice theory despite its limitations which include

respondents facing a binary choice. Condorcet, who was an enthusiastic supporter of democracy, believed that having an independent probability of voting for the correct decision greater than 0.5 was sufficient for majority voting to succeed, and that adding more voters will increase the probability of reaching the correct decision [116].

Combining the constraints for low difficulty tasks where  $0 < \Delta_k < 0.5$ :

$$\begin{aligned}
0.5 < \mathbb{E}[f(\theta_r)] &\leq \frac{\mu(1 - \Delta_k)}{(\mu(1 - \Delta_k) + \Delta_k(1 - \mu))} \\
&\Rightarrow 0.5 \leq \frac{\mu(1 - \Delta_k)}{(\mu(1 - \Delta_k) + \Delta_k(1 - \mu))} \Rightarrow \mu \geq \Delta_k
\end{aligned} \tag{4.7}$$

Combining the constraints for higher difficulty tasks where  $0.5 < \Delta_k < 1$ :

$$\mathbb{E}[f(\theta_r)] \geq \left\{ \frac{\mu(1 - \Delta_k)}{(\mu(1 - \Delta_k) + \Delta_k(1 - \mu))} \middle| 0.5 \right\} \Rightarrow \mu \geq \Delta_k \tag{4.8}$$

To summarize, for aggregation in PR to work, the expected value of the crowd's ability should be at least equal to the difficulty of the proposed task. This constraint on the ability mean ensures that the expected value of the competence is at least equal to half, which agrees with the assumption of voter competence. If so, the probability of success in PR will have a lower bound and will depend on the number of respondents who attempt to solve the task. As the number of respondents grows in size, the probability of success converges at a faster rate.

In order to compute the average probability of success versus task difficulty in PR, we take the integral of the competence function over the entire range of crowd ability resulting in equation 4.9. The result is in terms of task difficulty.

$$\begin{aligned}
P_{\text{PR}} &= \int_0^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r = F(\Delta_k) \\
&= \frac{(\Delta_k - 1)(2\Delta_k + 2\Delta_k \tanh^{-1}[1 - 2\Delta_k] - 1)}{(1 - 2\Delta_k)^2}
\end{aligned} \tag{4.9}$$

### 4.2.2 Confidence-Weighted

In the CF method, the notion is to give higher weights to the replies of respondents who are more confident and lower weights otherwise.

$$\ddot{\mathbf{a}}_q^{\text{CF}} \rightarrow \underset{\ddot{\mathbf{a}} \in \mathbb{A}}{\text{argmax}} \sum_{r \in \mathbb{R}_q} \mathbf{y}_r^k \cdot \delta_{\ddot{\mathbf{a}}_r \ddot{\mathbf{a}}}$$

The weights are the reported confidence values per respondent  $\mathbf{y}_r^k \rightarrow \mathbf{a}(\mathbf{x}_r^k)^2 + \mathbf{b}(\mathbf{x}_r^k) + 1$  which are modeled based on our proposed DK function.

We compute the expected probability of success in CF by taking the integral of the confidence-weighted competence over the range of ability values as shown in equation 4.10. The result is in terms of task difficulty and the coefficients  $\mathbf{a}$  and  $\mathbf{b}$  of our modeled DK function. The full equation is provided in the Appendix.

$$P_{\text{CF}} = \frac{\int_0^1 \left( (\mathbf{a}(\mathbf{x}_r^k)^2 + \mathbf{b}(\mathbf{x}_r^k) + 1) (\mathbf{x}_r^k) \right) d\theta_r}{\int_0^1 \left( \mathbf{a}(\mathbf{x}_r^k)^2 + \mathbf{b}(\mathbf{x}_r^k) + 1 \right) d\theta_r} = F(\Delta_k, \mathbf{a}, \mathbf{b}) \tag{4.10}$$

We define a general crowd as one that includes all classes of respondents from the low end of the spectrum (non-experts) to the high end (experts). Assuming we have a general crowd, and taking into consideration the psychological bias that accompanies respondents, which is represented in the modeled Dunning-Kruger

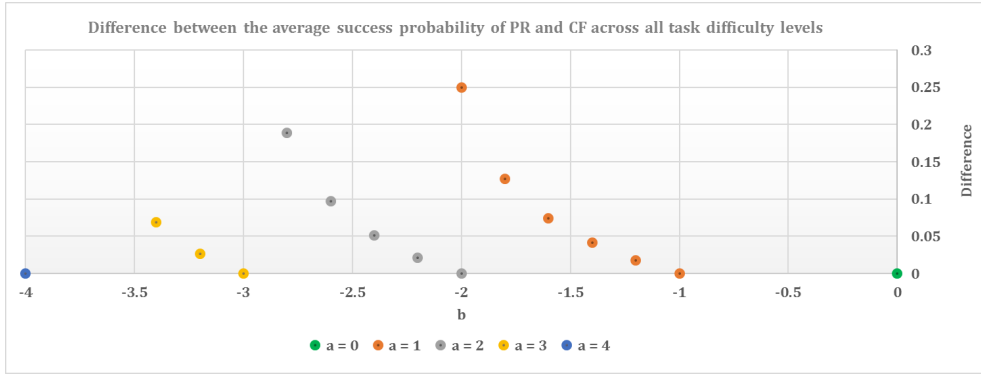


Figure 4.2: Difference between success probability of plurality and confidence-weighted voting across all difficulty levels and for different values of  $\mathbf{a}$  and  $\mathbf{b}$

function as a generally decreasing relationship between reported confidence and competence, confidence-weighted approaches will in most cases perform worse than the simple plurality approach. If the least competent are reporting the highest confidence and the most competent are not, then weighing by the reported confidence values will negatively affect the aggregation outcome.

**Theorem I** Given any DK plot, PR will perform better than CF for tasks of higher difficulty ( $\Delta_k \geq 0.5$ ).

The full proof of Theorem I is found in Appendix A.

For different samples of the DK plot as shown by the values of the coefficients  $\mathbf{a}$  and  $\mathbf{b}$  in Figure 4.2, we computed the difference between the probability of success of PR and that of CF. As expected, based on the model alone, PR outperforms CF for a wide range of possible DK plots. The difference between the two success probabilities is never below zero. What is interesting however, is the relationship between this difference and the coefficients  $\mathbf{a}$  and  $\mathbf{b}$ .

As the value of  $\mathbf{a}$  increases, the difference is decreasing, which means that CF's performance is improving compared to PR. This agrees with our previous

note on how increasing  $\mathbf{a}$  increases the confidence faster for respondents of higher competence. The result of increasing  $\mathbf{a}$  is an increased confidence value for the more competent.

As the value of  $\mathbf{b}$  decreases, we notice a similar pattern; the difference decreases in value for all values of  $\mathbf{a}$ . This observation agrees with another previous note on how decreasing  $\mathbf{b}$  has the effect of decreasing the confidence faster for respondents of lower competence. The result of decreasing  $\mathbf{b}$  is a decreased confidence values for the less competent.

### 4.2.3 Maximum Confidence

In this approach, PR is applied to a subset of the population with maximum confidence.

$$\ddot{\mathbf{a}}_q^{\text{MC}} = \underset{\ddot{\mathbf{a}} \in \mathbf{A}}{\operatorname{argmax}} \sum_{\mathbf{r} \in \mathbf{R}_q^{\text{MC}}} \delta_{\ddot{\mathbf{a}}_q \ddot{\mathbf{a}}} \quad (4.11)$$

We compute the expected probability of success in MC similarly to that in PR but by taking different integral boundaries, which depend on the number of defined confidence levels. Or, if the confidence is a continuous value, the choice of respondents is those in the top  $\mathcal{P}$  percentage in terms of confidence level. Given a DK plot and a percentage  $\mathcal{P}$ , boundary competence values are computed as follows. A horizontal line based on the selected  $\mathcal{P}$  is drawn on the DK plot. The line will intersect the DK function at two points. The x-values of these two intersection points are the resulting competence values that are used as boundaries for the integral that is used to compute the MC success probability. We define  $x_{\text{min}}^{\mathcal{P}}$  and  $x_{\text{max}}^{\mathcal{P}}$  as the intersection points between our modeled DK function and the horizontal line drawn at  $y^{\mathcal{P}} = 1 - \frac{\mathcal{P}}{100}$ .

The minimum point on the DK plot is defined as the point where the first derivative equals 0. Let  $\nabla$  denote the point of minimum confidence. Its coordinates are shown below.

$$\nabla = (\nabla_x, \nabla_y) = \left( -\frac{b}{2a}, 1 - \frac{b^2}{4a} \right)$$

There are two cases to consider; in the first case, the point  $y^{\mathcal{P}} = 1 - \frac{\mathcal{P}}{100} \leq \nabla_y$ , which means that the percentage covers the entire range of confidence values. In this case, applying MC has the same effect as applying PR and the two boundary points are in fact one point.

$$y^{\mathcal{P}} \leq \nabla_y \Rightarrow x_{\min}^{\mathcal{P}} = x_{\max}^{\mathcal{P}}$$

In the second case, the point  $y^{\mathcal{P}} = 1 - \frac{\mathcal{P}}{100} > \nabla_y$ , which means that if we are to apply the MC method, then we should only consider the answers of the respondents whose competence levels are less than  $x_{\min}^{\mathcal{P}}$  or greater than  $x_{\max}^{\mathcal{P}}$  where  $x_{\min}^{\mathcal{P}}$  and  $x_{\max}^{\mathcal{P}}$  are computed as shown below.

$$y^{\mathcal{P}} > \nabla \Rightarrow \mathcal{P} < \frac{25b^2}{a} \text{ and } x_{\min}^{\mathcal{P}} = \frac{-b - \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a} \text{ and } x_{\max}^{\mathcal{P}} = \frac{-b + \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a}$$

We compute the expected probability of success in MC by taking the integral of the competence over the range of ability values as shown in equation 4.12. The result is in terms of the boundary points and the task difficulty. The full equation

of  $P_{MC}$  is provided in the Appendix.

$$\begin{aligned}
P_{MC} &= \int_0^{x_{\min}^{\mathcal{P}}} \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r + \int_{x_{\max}^{\mathcal{P}}}^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r \\
&= F(\Delta_k, x_{\min}^{\mathcal{P}}, x_{\max}^{\mathcal{P}})
\end{aligned} \tag{4.12}$$

Assuming we have a general crowd and taking into consideration the psychological bias that accompanies respondents, which is represented in the modeled Dunning-Kruger function as a generally decreasing relationship between reported confidence and competence, selecting the most confident replies only will affect the aggregation outcome negatively. If the least competent respondents are reporting the highest confidence values, a maximum confidence approach will result in selecting a majority composed of the least competent individuals in the crowd. The outcome in MC is affected more severely than that in CF. In this method, we are considering a sub-crowd of respondents who were the most extreme when reporting their confidence levels. The MC method focuses on the least competent respondents and disregards respondents whose reported confidence values were moderate, which we argue are the majority of the most competent respondents. Accordingly, the higher the value of  $\mathcal{P}$ , the better the performance of MC when compared to CF and PR. For a value of  $\mathcal{P} = 100$ , the MC approach is the same as PR voting.

**Theorem II** Given any DK plot, any task difficulty, and any value of the percentage  $\mathcal{P}$ , PR performs better than MC.

**Theorem III** The probability function based on MC is increasing in  $\mathcal{P}$ .

**Theorem IV** At  $\Delta_k = 0.5$ , CF performs better than MC for all values of



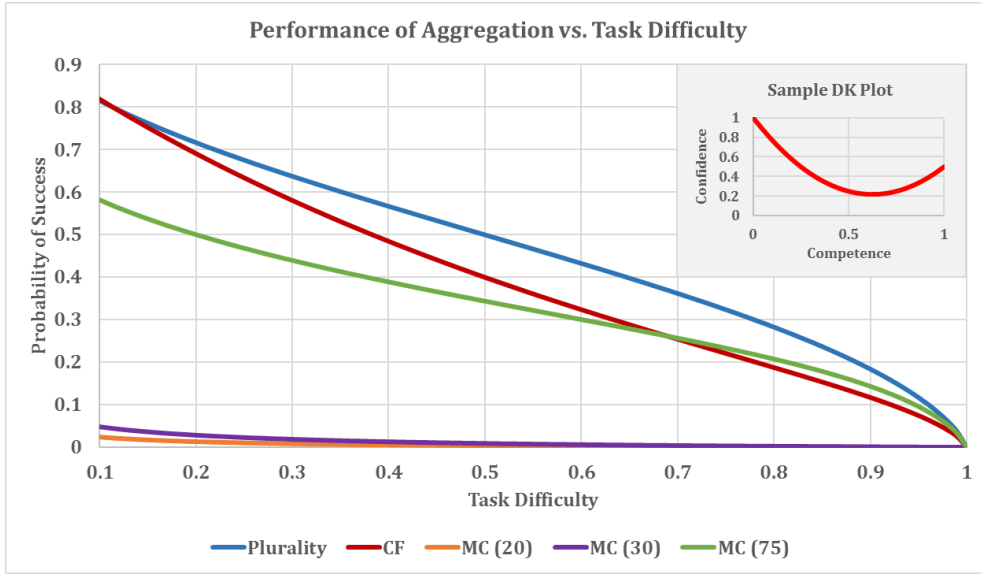


Figure 4.3: Success probabilities of plurality, confidence-weighted, and three maximum confidence approaches with different percentage values

$\mathcal{P} \leq \frac{25b^2}{a} - 10$ . As  $\mathcal{P}$  decreases, the point at which MC outperforms CF shifts further to the right at higher difficulty levels.

Proofs of Theorems II, III, and IV are found in the Appendix.

We show a sample DK plot for a general crowd where  $\mathbf{a} = 2$  and  $\mathbf{b} = -2.5$  in the top right corner of Figure 4.3. Using this DK function, we plot the success probabilities of PR, CF, and MC for three values of  $\mathcal{P}$  (20, 30, and 75) and for values of task difficulty ranging from 0.1 to 1. These plots are based on our modeled values of PR, CF, and MC.

A first look at the figure shows how PR outperforms all other methods for difficulty levels higher than 0.5. However, for smaller values of the task difficulty, the performances of PR and CF are comparable. For the smaller values of  $\mathcal{P}$  (20 and 30), the performance of MC is very poor. For a higher value of  $\mathcal{P}$  (75), the performance of MC becomes comparable to that of CF when task difficulty is around 0.7 and even surpasses it for the harder tasks. However, its performance

remains worse than PR throughout.

Note that for the chosen  $\mathbf{a}$  and  $\mathbf{b}$  of 2 and -2.5, the percentage limit that was defined in Theorem IV evaluates to  $(\frac{25\mathbf{b}^2}{\mathbf{a}} - 10) \approx 68$ . Based on Theorem IV, this means that for  $\mathcal{P} = 20$  and  $\mathcal{P} = 30$ , at  $\Delta_k = 0.5$ ,  $P_{CF} \geq P_{MC}$  which is true. And as  $\mathcal{P}$  decreases,  $P_{CF}$  and  $P_{MC}$  intersect further to the right at values of  $\Delta_k > 0.5$  which is shown. At  $\mathcal{P} = 75$ , the two curves intersect at  $\Delta_k \approx 0.7$  and as  $\mathcal{P}$  decreases to 20% and 30%, the intersection takes place at  $\Delta_k = 1$ .

#### 4.2.4 Competence-Weighted

Another weighted approach worth discussing is the competence-weighted approach. In CP, rather than weighing the replies by the reported confidence of the respondents, the method weighs them by the estimated competence of the respondents. The real competence of a respondent is an unknown value. Additionally, it is affected by many factors such as his well-being and ability to focus at the time of solving the task. Instead of getting the real competence values of respondents, CP methods try to estimate these values. The literature is filled with findings related to competence detection techniques [117, 118, 119, 120, 121, 122]. Some techniques are a function of the time it takes a respondent to solve a task [123]. Some are based on the performance of a respondent in previous tasks [124]. A wide variety of techniques even make use of Bayes' theorem to try to detect the competence of respondents when the ground truth of the proposed questions is not available a priori [125, 126, 127]. Even though our focus in this work is not on the deployed competence detection method, we stress that it is one of the most important steps when designing a crowdsourcing platform. Many techniques have been derived with the aim to estimate workers' competence and reliability when attempting to solve a task. The better the detection technique,

the higher the chances of reaching a correct aggregate answer. In fact, detecting the competence or reliability of workers is one of the many techniques of quality control that are employed by the system to improve performance. We provided a survey of quality control mechanisms in the literature review section where we showed that estimating the real competence of respondents is not a trivial task. If the estimated and the real values of the competence are very far off indicating a primitive detection method, then naturally, a competence-weighted aggregation approach will perform poorly.

Instead of focusing on detecting the competence, we make use of our modeled DK function to derive the competence of respondents based on their reported confidence values. Getting the reported confidence from a respondent is a very easy task. And this reported confidence value is an actual value rather than an estimated one. Assuming a general crowd, we can give lower competence values for highly confident respondents and higher competence values for the moderately confident ones. In other words, we can use our model of the DK function as a competence detection technique. The performance of this detection technique can be evaluated based on the performance of the competence-weighted aggregation method when using the estimated competence values derived from our model of the DK function.

***Hypothesis I*** Given a general crowd, competence-weighted approaches using competence values estimated from the modeled DK function will outperform approaches based on plurality voting.

We argue that the choice of the DK function coefficients does not significantly affect the performance of the competence detection method. This, of course, only holds in the case of a general crowd where the overall confidence level decreases as the competence increases and where more competent individuals are never as

confident as the least competent ones.

## 4.3 Numerical Analysis

To validate our model of the different aggregation techniques, we performed numerical analyses on crowdsourced data. We begin by describing the dataset that was used. Then, we discuss the two main experiments that we performed and the results that were drawn from each experiment.

### 4.3.1 Dataset

Aydin et al. developed and deployed a crowdsourcing system for playing the popular “Who Wants to Be a Millionaire?” television game show [128]. They created an Android app that allowed respondents to play the game at the same time as the show was being broadcasted. The app was downloaded over 300,000 times over a period of 9 months. The data in it is based on 1908 live game show questions from a total of 80 broadcasted episodes. Respondents gave a total of 214,658 answers to these questions. Every game show had an average respondent count of 733. The average number of answers per question was around 100. On the server side, project administrators instantly type the questions and answers as they appear on the game show. On the respondent side, every participant selects his answer to each question along with his confidence for every submitted answer. The confidence values to choose from are 1 for “no idea”, 2 for “guessing”, and 3 for “certain”. Every game has 12 questions ranked by increasing difficulty. Questions 11 and 12 are the most difficult questions. Consequently, the number of times these questions were asked was of no statistical significance and so we do not include them in our analysis. Taking into consideration the large number

of respondents who are fans of the “Who Wants to Be a Millionaire?” game show and who can lie anywhere on the spectrum of respondent competence, we assume a general crowd.

### 4.3.2 Results and Discussions

In the first experiment, we started by ranking the questions based on their difficulty level. The aggregated answer for every question in the case of PR was the most voted for choice. In the case of CF, the aggregated answer was the most voted for choice after weighing all the choices by their respective confidence values. In the case of MC, we considered two cases. In the first case, we applied plurality voting on the subset of respondents whose reported confidence was 3. In the second case, the subset of respondents included those whose reported confidence was either 2 or 3. Then, we compared all the aggregated answers to the gold label of every question. A question whose aggregated answer matches with the gold label gets a score of 1, otherwise, it gets a score 0. After getting the total scores for every method, we computed the average score per difficulty level and plotted the results, as shown in Figure 4.4.

The first observation is related to PR. It outperforms the confidence-weighted approach ( $p\text{-value} < 0.0005$ ) and the maximum confidence approach ( $p\text{-value} < 0.0001$ ). This validates Theorem I which states that plurality outperforms confidence-weighted approaches for tasks with difficulty higher than 0.5. It also validates Theorem II which states that PR will always outperform MC aggregation approaches. Also note that for the easier tasks, the performance of CF is better than that of plurality voting. The point at which PR outperforms CF depends on the values of the coefficients  $\mathbf{a}$  and  $\mathbf{b}$  of the DK plot.

These results agree with the conclusions of Li et al. who study an M-ary clas-

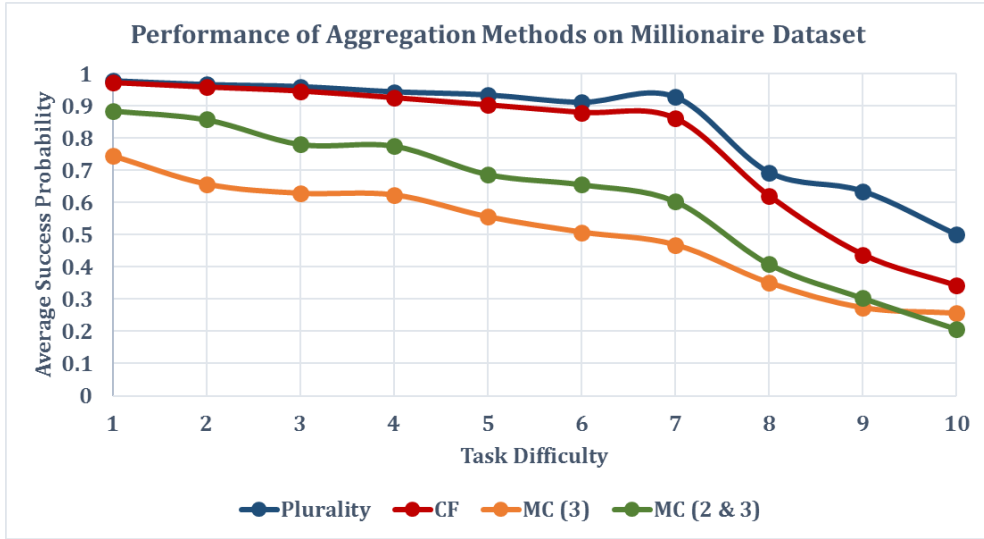


Figure 4.4: Success probabilities of different aggregation methods based on data from WWTBAM dataset

sification task via crowdsourcing where the workers report quantized confidence scores [129]. They consider a crowd that is only composed of honest workers who answer questions and report confidence in good faith. Their simulation results demonstrate how the performance of the crowdsourcing task does not improve when incorporating workers' confidence scores.

The second observation is that  $P_{MC}$  decreases with the decrease in  $\mathcal{P}$  except for the point at question 10, the reason of which could be due to data that is insignificant for this question. We notice that when only replies with confidence level 3 were considered, the probability of success was lower than that when confidence levels 2 and 3 were considered ( $p\text{-value} < 0.0001$ ). This validates Theorem III. CF outperforms both instances of MC regardless of the task difficulty. This does not contradict with Theorem IV since there is no intersection whatsoever. The theorem would be contradicted had there been an intersection at a relatively small difficulty level.

The aim of the second experiment was to test our competence detection idea

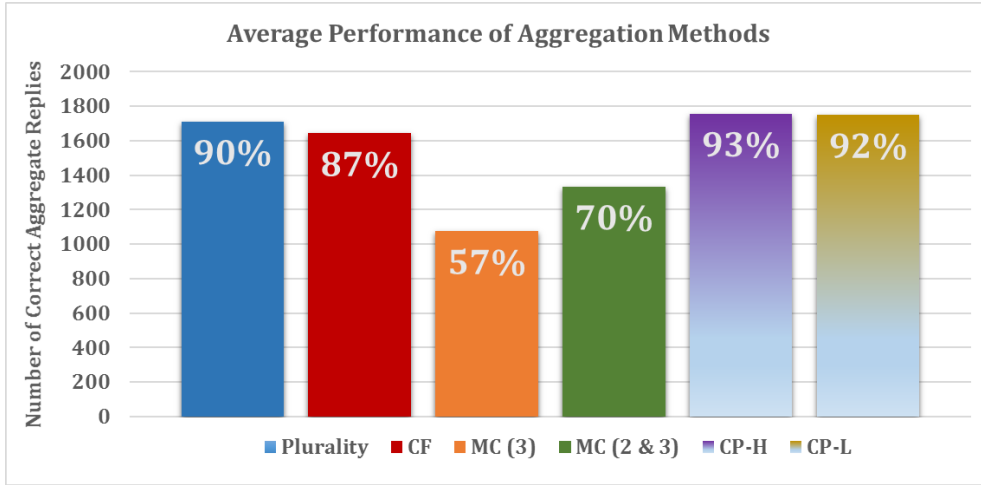


Figure 4.5: Performance of different aggregation methods based on data from WWTBAM dataset

based on our modeled DK function. We selected three random DK functions ( $\alpha = 2$  and  $\beta = -2.75$  |  $\alpha = 2.55$  and  $\beta = -3.15$  |  $\alpha = 3$  and  $\beta = -3.45$ ). We derived the competence value for every user from his reported confidence using each of the three DK functions. For some confidence values however, there are two possible competence values considering the general shape of our modeled DK plot. For these cases, we generated two sets of weights. In the first set, we selected the lower competence value (CP-L) and in the second, we selected the higher one (CP-H). We then weighed the answers using the derived competence values and computed the scores of CP-L and CP-H after comparing each set of aggregated answers to the gold label and taking an average score over the 3 DK functions. We present the results in terms of a bar chart in Figure 4.5. Inside the bars, we show the percentage of questions out of the total questions that were correctly aggregated by each method.

Our first observation is that, as in the previous experiment, MC (3) performed worse than MC (2 and 3) and both performed worse than CF which in turn lagged behind PR. What is interesting in this experiment, however, is that both CP-

L and CP-H performed better than PR ( $p - \text{value} < 0.01$ ). This indicates that the competence values that we derived using our DK model were in fact good estimates of the real competence values. This validates Hypothesis I above, which states that given a general crowd, competence-weighted approaches using competence values estimated from the modeled DK function will outperform approaches based on PR.

Another interesting observation is that CP-H performed slightly better than CP-L, which is expected. For high values of the reported confidence, there is only one value of the competence based on the general shape of the DK function. It is for the lower confidence values that we encounter two competence values, a low one and a high one. One of the main conclusions given by Dunning and Kruger is that the more competent individuals are those who report lower confidence values as they generally underestimate their competence. This means that giving a higher competence value to these low-confidence respondents will result in a better overall competence estimation and therefore better aggregation performance.

Note that the CP scores before averaging across the three different DK functions were very close and surpassed all other approaches. This validates how the choice of the DK coefficients  $\mathbf{a}$  and  $\mathbf{b}$  does not significantly affect the performance of the competence detection method. The chosen DK plots in this example revealed a pattern where the most incompetent respondents showed the greatest miscalibration in assessing their skills.

Qi et al. discuss the issue of long-tail phenomenon in crowdsourcing tasks where most workers only provide answers to a few tasks and only a few workers provide answers to plenty of tasks [130]. They argue that existing crowdsourcing approaches clearly overlook this phenomenon which causes problems in estimat-



ing worker reliability. They propose to consider both the estimate of worker reliability along with its confidence interval in order to accurately reflect reliability levels of workers with different degrees of participation. This results in reducing the effect of less active workers who do not solve many tasks. They perform experiments on four real world crowdsourcing tasks, one of which is the previously described “Who Wants to Be a Millionaire” dataset. Results demonstrate how their proposed Confidence-Aware Truth Discovery (CATD) method, which takes into consideration the long-tail phenomenon, outperforms existing approaches [131, 132, 133, 134, 135].

More recently, Fenglong et al. addressed the issue of topic diversity in crowdsourcing systems [136]. They argue that most existing systems assume that a worker will have the same reliability when answering any question. Existing approaches ignore the fact that a worker’s reliability may vary significantly depending on the topic. To this end, they propose FaitCrowd which probabilistically models question content and answer generation in an attempt to assign topics to questions, learn the ground truth, and estimate workers’ topic-specific expertise simultaneously. They test their method on real world datasets including the “Who Wants to Be a Millionaire” dataset. Their results demonstrate how FaitCrowd reduces the error rate when compared with other approaches [137, 130, 131, 132, 133, 134, 135, 138].

Table 4.1 compares our competence-weighted aggregation technique (CWAT) to CATD and FaitCrowd, among other methods, in terms of the Error Rate. We computed the error rate per task difficulty as well as the overall error rate. The highlighted cells are the ones that have a lower error rate than our proposed method. CWAT outperforms TruthFinder and Investment across all difficulty levels. For the more challenging tasks, it outperforms all methods except CATD

Level	1	2	3	4	5	6	7	8	9	10	Overall
CWAT	<b>0.0161</b>	<b>0.0341</b>	<b>0.0346</b>	<b>0.0434</b>	<b>0.0442</b>	<b>0.0585</b>	<b>0.0343</b>	<b>0.2124</b>	<b>0.2472</b>	<b>0.3</b>	<b>0.0641</b>
CATD	0.0132	0.0271	0.0276	0.029	0.0435	0.0596	0.0481	0.1304	0.1414	0.2045	0.0485
FaitCrowd	0.0132	0.0271	0.0241	0.0254	0.0395	0.055	0.0481	0.087	0.101	0.1136	0.0399
TruthFinder	0.0693	0.0915	0.1241	0.0942	0.1581	0.2294	0.2674	0.3913	0.5455	0.5455	0.1816
AccuSim	0.0264	0.0305	0.0345	0.0507	0.0632	0.0963	0.0909	0.2826	0.3636	0.5	0.0913
Investment	0.033	0.0407	0.0586	0.0761	0.087	0.1239	0.1283	0.3406	0.3838	0.5455	0.1151
3-Estimates	0.0264	0.0305	0.031	0.0507	0.0672	0.1055	0.0963	0.2971	0.3737	0.5	0.0942
CRH	0.0264	0.0271	0.0345	0.0435	0.0593	0.0872	0.0856	0.2609	0.3535	0.4545	0.0866
D&S	0.0297	0.0305	0.0483	0.0507	0.0672	0.1101	0.0963	0.2971	0.3636	0.5227	0.0975
ZenCrowd	0.033	0.0305	0.0345	0.0471	0.0593	0.0872	0.0856	0.2754	0.3636	0.5227	0.0899

Table 4.1: Error rate of competence-detection approach compared with approaches in the literature

and FaitCrowd. The overall performance of CATD and FaitCrowd is better than that of our method except in tasks of difficulty levels 6 and 7 where our method performs better. It is worth mentioning that the error rates of the other methods are based on the results presented by Qi and Fenglong in their papers and are not based on our own simulations of these methods.

Even though the performance of CATD was better than ours given the “Who Wants to Be a Millionaire” dataset, it suffers from a major drawback. The approach by Qi et al. offers no advantage over existing approaches in crowdsourcing tasks where the long-tail phenomenon is not present. Even though this phenomenon should not be overlooked when estimating worker reliability, however it is not always present in crowdsourcing systems. For example, workers might not have the option of solving a subset of the questions even if they do not know the answer to some questions. Moreover, there are cases where there is only one question that workers attempt to answer. These are two crowdsourcing examples where the long-tail phenomenon does not exist and will have no effect on the overall performance of the system. This renders other methods such as ours that work regardless of the participation level to be more useful for better system performance. Of course in specific cases where the long-tail phenomenon is present, the estimation of worker reliability will be affected by the total num-

ber of claims that he or she makes. A worker that makes one claim to one task will either be categorized as extremely reliable or highly unreliable based on the correctness of his single claim. We believe that this negatively affects worker reliability estimation as well as confidence reporting. Addressing this phenomenon in both reliability detection and confidence reporting is a necessary step that we believe will result in error rates lower than both our approach and CATD.

FaitCrowd by Fenglong et al. suffers from a similar drawback. Even though the approach detects per-topic expertise of workers for better system performance, the assumption of different topics in proposed questions is not always valid. There are many examples where all proposed questions fall into the same area. When this is the case, FaitCrowd will show no advantage over other approaches. This also applies in cases where the crowd is voting for the best answer for a single question. Even though our overall error rate is slightly higher than FaitCrowd's, our method is more versatile in that it considers all types of crowds and any blend of question topics. FaitCrowd is superior only in cases where the topics of proposed questions vary which is not always the case.

We believe that an aggregation technique that aims to estimate topic-specific worker expertise (when applicable) taking into consideration the effects of the long-tail phenomenon (if present) and the cognitive biases of the workers involved in the crowdsourcing process should improve the truth discovery process considerably.

### **4.3.3 The Irregular Crowd**

If we do not consider a general crowd, the relationship between confidence and competence will no longer abide by the characteristics of the DK plot. For example, if we have a crowd of experts in the field, a sample competence-confidence

relationship might be a straight line starting from 0 with a slope of 1. This is, by far, the best representation of confidence versus competence since the crowd consists of experts who have the ability to correctly estimate their cognitive skills on either side of the spectrum. In this case, CF approaches will perform better than simple PR voting due to the fact that the reported confidence values are not biased.

In fact, this is illustrated in one of the experiments by Prelec et al. who describe a new and interesting aggregation method commonly referred to as the Surprisingly Popular algorithm [139]. The authors argue that this algorithm can help extract the correct answer from a crowd even when the majority replies incorrectly. To accomplish this, participants are queried twice. They are asked to answer the question and to predict what the majority will answer as well. The “correct” answer is based on the variation between the given answers and the predicted ones. In their paper, they describe how they performed the same crowdsourcing experiment twice. The experiment asked respondents to judge the price of 90 reproductions of modern 20th century artwork. In the first part of the experiment, the crowd was composed of people working with art in galleries or museums. In the second part, the crowd was composed of MIT master’s and doctoral students who have not taken any courses on art or art history. What is interesting is that both the confidence-weighted and the maximum confidence approaches outperformed the majority vote in the case of the art professionals. On the other hand, they both performed worse than the majority vote in the case of the MIT students who fit the description of a general crowd. The maximum confidence in the second part even performed worse than the confidence-weighted approach, which agrees with our results in this dissertation. The surprisingly popular algorithm has received considerable attention recently and is already

referenced in works related to voting and crowdsourcing [140, 141]. We did not include this algorithm in our “Who Wants to Be a Millionaire” study due to the lack of data. However, we compare its performance with other approaches in the experiment phase of the work that is described in Chapters 5 and 6.

## 4.4 Discussion

There are many factors that affect the choice of the aggregation method to adopt in crowdsourcing. Our conclusion is that the type of crowd is the most important of these factors. In the case of an incompetent crowd, no aggregation method will have a good performance. This includes simple methods such as plurality voting which requires an average respondent to have a success probability above 0.5 in accordance with Condorcet’s jury theorem of voter competence. When knowledge within a crowd surpasses a given threshold, the choice of aggregation technique matters as we have shown.

The main finding in this chapter is related to the Dunning-Kruger effect. We presented a formal model of this psychological bias in terms of a quadratic relationship between respondent competence and confidence. Constraints related to our modeled function were derived from the characteristics of this effect as described by Dunning and Kruger.

Using our model of the Dunning-Kruger effect as the core of our work, we then went on to model the performance of different aggregation methods. Due to the general shape of the DK function, we were able to validate that for a general crowd of respondents, a plurality aggregation method will, in most cases, outperform methods based on the confidence of respondents such as the confidence-weighted and maximum confidence approaches. We also showed that the maximum con-

confidence approach generally lags behind the confidence-weighted approach since it focuses on the least competent respondents and disregards respondents whose reported confidence values were moderate, which we argue are the majority of the most competent respondents. We also showed how the size of the subgroup of respondents in the maximum confidence approach affects the overall performance of the method for a general crowd.

We then went on to use the modeled DK function as a competence detection technique. There are many detection techniques in the literature that attempt to estimate the competence of respondents in a crowdsourcing system with or without the presence of the ground truth. These techniques are complicated and in some cases, they do not give a good estimate of the real competence values of respondents. Our argument is that it is fairly easy to get the reported confidence of respondents when solving a task and we showed how we were able to use these reported values to estimate, to a good degree, the competence of respondents. Using our estimated values of the competence, we managed to get the best performance using a competence-weighted approach compared to other approaches.

According to Dunning and Kruger, perhaps the best illustration of inflated self-appraisals of the incompetent is the tendency of the average person to rate her skills as above average, which defies the logic of statistics. On the other hand, the most competent individuals usually suffer from the false-consensus effect [142] where they assume that because they have performed well on a task, then others must have performed well likewise, which results in these individuals underestimating their relative abilities. These two findings validate the general shape of the DK function that we adopted in our model. Dunning and Kruger also observed that it was the most incompetent individuals who showed the greatest

miscalibration in assessing their skills, which validates our choice of having the DK function start at the point  $(0, 1)$  indicating the highest confidence miscalibration for the least competent.

# Chapter 5

## System Design and Implementation

Having represented the interaction between our system and a target app and modelled several aggregation methods that can be used when computing the behavior score while accounting for possible biases in user replies, we proceed to implement an app rating system that will allow us to verify our developed models. We present the overall design and implementation of both the backend server and the updated version of CrowdApp (version 2.0). We describe the process of storing feature logs that are used for utility score computation, and the process of detecting events that are used for behavior score computation. We discuss concepts of user selection and reputation threshold and describe how the aforementioned aggregation methods were implemented. Backend specifications along with performance results are also provided.



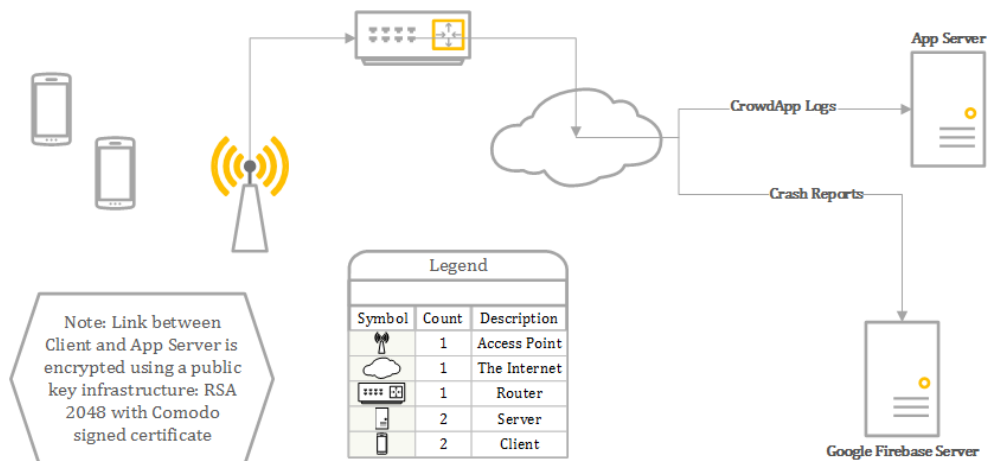


Figure 5.1: Client-server architecture

## 5.1 General Overview

The general client-server architecture is shown in Figure 5.1. The client is CrowdApp v2.0 which is installed on Android devices running API 21, 22, and 23. CrowdApp v2.0 collects different types of data from a device and sends it to the server that manages this incoming data and responds accordingly. The figure also shows Google’s Firebase Server [143]. We use Firebase for crash reporting and notification services. Upon installing CrowdApp on a device, a user is automatically registered. A unique Google Firebase token is generated for every user. The hash of this token is the username that will be stored on the server. The token is also used to send push notifications to users. This process ensures the anonymity of users in our system. CrowdApp v2.0 also uses a hard coded password along with the username to insert and update logs on the server. The client-server link is encrypted using Public Key Infrastructure (PKI). We used a certificate signed by Comodo [144]. The certificate uses RSA with a 2048 bit key. It is included in the Assets folder of CrowdApp v2.0.

## 5.2 Client Side

CrowdApp v2.0 collects two types of data from every device: logs and events. Logs are basically statistics of all the monitored activities that are taking place on the device. These dumps are first stored in log files on the device's external storage and are later uploaded to the server. Events are certain actions that take place on the device. Once these events take place, they are instantly sent to the server side. In the following, we will discuss, in detail, the differences between stored logs and detected events. In the first subsection, we describe the method used to collect and store logs for utility score computation. In the second subsection, we describe the method of behavior score computation which includes the game process, querying process, and rating process.

### 5.2.1 Method of Collecting and Storing Logs

The stored data in these log files is categorized into:

- Connection Stats (CxN) - These logs contain information related to all types of connections that are opened by all third-party apps on the device. The information per app  $\hat{A}$  is stored as follows:
  - Number of connections made by  $\hat{A}$  at each  $T_{CxN}$  seconds
  - Age of every connection made by  $\hat{A}$  at each  $T_{CxN}$  seconds
  - Destination IP and port number, source IP and port number, and protocol, all combined as one HEX string
  - Timestamp for every stored log entry
- Traffic Stats (TR) - These logs contain information related to the traffic sent

and received by all third-party apps on the device. The traffic is stored in bytes as well as in packets. The information per app  $\hat{A}$  is stored as follows:

- Sent bytes from  $\hat{A}$  at each  $T_{TR}$  seconds - TxBytes
  - Received bytes by  $\hat{A}$  at each  $T_{TR}$  seconds - RxBytes
  - Sent packets from  $\hat{A}$  at each  $T_{TR}$  seconds - TxPackets
  - Received packets by  $\hat{A}$  at each  $T_{TR}$  seconds - RxPackets
  - Timestamp for every stored log entry
- CPU and Memory Stats (CM) - These logs contain information related to the CPU and memory consumption of all third-party apps on the device. The memory consumption is given in terms of the Resident Set Size (RSS) memory consumption. The information per app  $\hat{A}$  is stored as follows:
    - The percentage of CPU time of  $\hat{A}$  at each  $T_{CM}$  seconds
    - The RSS memory consumption of  $\hat{A}$  at each  $T_{CM}$  seconds
    - Timestamp for every stored log entry
- App Usage Stats (AUsg) - These logs contain information related to the usage of all third-party apps on the device. The information per app  $\hat{A}$  is stored as follows:
    - Average time  $\hat{A}$  is used on the device
    - Number of times  $\hat{A}$  is opened on the device
    - The overall install period of  $\hat{A}$  on the device

The above logs are collected at their respective collection intervals (every  $T_{CN}$ ,  $T_{TR}$ ,  $T_{CM}$  seconds) and the log files are constantly updated on the device's

storage. Once the size of the log files reaches a set limit, CrowdApp will attempt to upload the log files to the server. The limits for the log files depend on the type of log. For the connection stats, traffic stats, and CPU and memory stats, the limit for the log file size is set to  $L_L$ . For the app usage stats, the limit for the log file size is set to  $L_S$ . Before beginning the uploading process, CrowdApp will check the type of connection on the device. If the user is using his mobile data, the upload will not take place to ensure that the data plan of the user is not affected. Once the user switches from mobile data to WiFi, CrowdApp will detect the change in network type and will start (or resume if it had already started) the uploading process. When a log file reaches its limit and is set to be uploaded to the server, a new log file is automatically created in the device's external storage and new logs are inserted in the new log file. Assuming the server was reachable, after it successfully receives an uploaded file from CrowdApp, it sends an acknowledgement back to the user's device. Upon receiving this acknowledgement, the uploaded log file is deleted from external storage.

All log files are stored on the device encrypted using AES 128. Before uploading to the server, they are decrypted and converted to JSON strings. The resulting data is very small which means that the time to upload a log file will be insignificant and the resulting bandwidth consumption by CrowdApp v2.0 in the process is minor. On the server side, the JSON strings are processed to extract the necessary information which are inserted in their respective tables in a database.

## 5.2.2 Method of Detecting Events

An event is defined as any behavior of a third-party app on the device that is considered suspicious. The types of events that can be issued by a third-party app  $\hat{A}$  on a device are those related to:

- $\hat{A}$  has high CPU usage -  $E_{CPU}$
- $\hat{A}$  has high RAM usage -  $E_{RAM}$
- $\hat{A}$  has high bandwidth consumption -  $E_{TR}$
- $\hat{A}$  is opening long-term connections -  $E_{CA}$
- $\hat{A}$  is opening a relatively large number of connections -  $E_{CC}$

As previously discussed, readings take place on the device at specific time intervals ( $T_{CN}$ ,  $T_{TR}$ , and  $T_{CM}$ ) where each interval is related to one of the monitored features on the device. Every reading for app  $\hat{A}$  that is collected by CrowdApp v2.0 at any time is compared to a feature threshold. The defined feature thresholds are:

- CPU feature threshold -  $FTh_{CPU}$
- RSS feature threshold -  $FTh_{RSS}$
- Bytes sent feature threshold -  $FTh_{TxB}$
- Bytes received feature threshold -  $FTh_{RxB}$
- Packets sent feature threshold -  $FTh_{TxP}$
- Packets received feature threshold -  $FTh_{RXP}$

- Connection age feature threshold -  $FTh_{CA}$
- Connection count feature threshold -  $FTh_{CC}$

The above feature thresholds are computed by the server based on incoming data from all user devices. They are stored in the database. A feature threshold is computed as a tuple: [average value per feature per app which is then averaged over all apps, standard deviation over all apps]. It is updated on a regular basis. The computation takes place as follows. The server is constantly receiving logs of different categories for all third-party apps on all user devices. It will compute a feature threshold for each category of collected data for each third-party app. For example, to compute the average value of  $FTh_{CPU}$  for WhatsApp, it takes the average value of the CPU readings of WhatsApp from all user devices. This average reading per feature (e.g. CPU) per app (e.g. WhatsApp) is averaged with other readings for the same feature (CPU), but for different apps (e.g. Facebook, Dailymotion). The resulting average is the first tuple in the feature threshold. Of course, since data related to the CPU consumption of apps is constantly being sent from user devices, the value of this average will keep changing in the database on the server side. To make sure that all CrowdApp clients have an updated value of this feature threshold as well as all other feature thresholds, a control bit is added to CrowdApp v2.0. This control bit is read every time CrowdApp uploads data to the server. If the values of the feature thresholds have been updated since the last time CrowdApp uploaded data to the server, the new values will be downloaded to the user's device. This process ensures that at any point in time, all CrowdApp users have updated values of the feature thresholds stored on their devices. And they receive the updated values when a connection has already been set up with the server to upload logs. This way, there is no need to open

a connection specifically for the purpose of downloading new feature threshold values, and thus saving the lifetime of the device's battery.

Having updated feature threshold values, every time CrowdApp collects a new reading for one of the features of third-party apps, it compares this reading to the respective feature threshold at a specific time interval  $T_{\text{Comparison}}$ . The comparison is done by calculating the z-score, REV, of the reading with respect to the average and standard deviation of the feature threshold. A pseudorandom number is generated in Python using the basic `random()` function which generates a random float uniformly in the semi-open range  $[0.0, 1.0)$ . Python uses the Mersenne Twister as the core pseudorandom number generator which produces 53-bit precision floats and has a period of  $2^{19937} - 1$ . It is one of the most extensively tested random number generators [145]. The resulting random number is compared against REV. If REV is higher than the generated random number, an event is triggered on the device. The event is defined by the ID of the third-party app that triggered it, the feature being considered (CPU, RSS, TxBytes, etc.), the ID of the user whose device issued the event, and a timestamp of when the event was triggered. When an event is triggered, CrowdApp v2.0 will instantly attempt to send it to the server regardless of the network type. Sending an event to the server has low bandwidth consumption. Since it is a JSON string with 4 fields, the event payload is less than 1 KB which means that there is no need to set the constraint that events can only be sent when the network type is WiFi. Also, events are indicative of suspicious behavior on the device and should therefore be sent directly to the server assuming that the server is reachable. A flag that the server is reachable is set at each connection change.

The process of comparing new readings to feature thresholds is handled using a flag which is controlled by the server. The server can start or stop the process

of feature comparison by CrowdApp at any time by simply setting this flag which will, in turn, get automatically updated at the client side whenever an upload process is started.

On the server side, an event rate per feature per app is computed and updated every time a new event is triggered. The event rate is computed as the ratio of the total number of events received per feature per app divided by the total possible number of events that can be received in a specific time. There are two cases to consider. The first case is when an app has not been rated before. The time period to consider is the difference between the time when the last event was received and the time when the first event was received. The second case is when an app has been rated before. In that case, the time period to consider is the difference between the time when the last event was received and the time when the first event after the last rating of the app was received. After computing the time period, the total possible number of events is defined as follows:

$$\sum \text{Events}_{\text{Possible}} = \sum \text{Users}_{\hat{A}} \times \frac{T_{\text{Total}}}{T_{\text{Comparison}}} \quad (5.1)$$

The time period is divided by the comparison interval for the feature in consideration. This interval is set to be same for all apps. Assuming an event is triggered every time a comparison takes place, then this ratio will give us the total possible events that can take place on one user device. If we multiply this value by the total number of users who have the app in consideration installed on their device, the result will be the total possible events that can be issued in the computed time period. The event rate is then computed as follows:



$$\text{EventRate} = \frac{\sum \text{Events}}{\sum \text{Events}_{\text{Possible}}} \quad (5.2)$$

After computing the Event Rate, another pseudorandom number is generated using Python's `random()` function. If the value of the computed Event Rate is higher than the value of the random number, an alarm will be fired and the game process will be triggered by the server. In the next subsection, we describe the game process.

### Game Process

When playing a game versus app  $\hat{A}$ , the reputation of  $\hat{A}$  and the variables of the game model are first fetched from the database. Then, the average reputation of all the users who issued events for  $\hat{A}$  is computed from the database and stored as  $\beta$ . The reputation threshold is then defined as in Chapter 3 as follows:

$$\text{RT} = \frac{(g_n + L_N + C_D) - \beta(G_N + L_N)}{(g_n + L_N + \ell_m - L_M) - \beta(G_N + L_N - G_M - L_M)} \quad (5.3)$$

The server compares  $\hat{A}$ 's reputation to its reputation threshold. If  $\hat{A}$  was not rated before or if  $\hat{A}$ 's reputation is lower than its reputation threshold, the system defends against it. It selects from the database the particular question that is related to the feature which raised the alarm. The `Start_Asking_Questions` flag is checked. If it is 1, the `Questions` table in the database is updated by inserting the Question ID, App ID, Feature ID, and a timestamp, and the querying process is triggered.

On the other hand, if  $\hat{A}$  was rated before and its reputation is higher than

its reputation threshold, its `Reputation_Counter` value is incremented. The `Reputation_Counter` is a value that is stored in the database for every application. When an app fires an alarm but its reputation is higher than its threshold, the value of this counter increases. The ratio of the `Reputation_Counter` to the maximum allowable counter value (which is also stored in the database) is computed. A pseudorandom number is generated again using the `random()` function in Python. If the resulting value is lower than the computed ratio, the system will defend against the app even though its reputation is high enough (the same process takes place as in the case when the reputation is lower than the reputation threshold). After defending against it, it will reset its `Reputation_Counter` value back to 0. This is to ensure that if an app is behaving suspiciously on the device and it just happens that its reputation was wrongfully increased at some point in time, there is a chance for it to be detected later on. If we don't include this counter value, then it will be impossible to detect a suspicious app once and if its reputation increases to a certain value.

In the next subsection, we will discuss the querying process in detail which is the system's approach of defending against a suspicious app by referring to a group of respondents from the crowd.

### **5.2.3 Querying Process**

When the rating process is triggered on the server side, the first action that takes place is selecting the users to query. The selection process is based on many factors. First, only the users who have the app installed on their devices are taken into consideration. From this subset of users, those whose devices did not issue events related to the app and feature in question are filtered out. From the remaining subset of users, a user is chosen to be queried if he meets the below

conditions:

- The user should be active. If, at any point, a notification is sent to a user's device and the Firebase server indicated that the user token is inactive, the user is considered no longer active.
- The user hasn't been asked the same question regarding the same feature for the same app in the past 24 hours.
- The user hasn't been asked any question in the past hour.
- The total number of times that the user has been asked during that day does not exceed the maximum allowable number of times that a user may be asked in one day which is a value that is stored in the database and updated as desired.

Another condition to consider before querying users is that the current time should be between 10 am and 8 pm. This last condition was added to ensure that CrowdApp does not send notifications to users during times which are considered inappropriate for it will affect the quality of experience of CrowdApp v2.0 itself.

Note that if the app has not been rated before, then the initial set of users includes all users who have the app installed and whose devices issued events. However, if the app has been rated before, then the initial set of users will include the users who have the app installed and whose devices issued events that took place after the timestamp of the last time the app was rated.

After selecting the users to query, the server sends a push notification to their devices. This push notification will contain three simple questions related to the usage of the app when the event was triggered. The first question is related to the behavior of this app on a user's device. The second question asks for the user's

confidence in the answer he gave to the first question. This reported confidence is needed for the confidence-related aggregation techniques. Finally, the third question asks the user what he believes the majority of users will reply to the first question. The answer to this question is needed to apply the Surprisingly Popular aggregation approach that was proposed by the authors in [139]. In summary, the types of questions related to app  $\hat{A}$  that might be included in the notification are:

- $\hat{A}$  has relatively high CPU usage. The average value of its CPU time consumption on your device is X which is Y times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?
- $\hat{A}$  has relatively high RAM usage. The average value of its RAM consumption on your device is X which is Y times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?
- $\hat{A}$  is consuming a lot of down bandwidth. The average value of its bandwidth consumption on your device is X which is Y times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?
- $\hat{A}$  is consuming lot of up bandwidth. The average value of its bandwidth consumption on your device is X which is Y times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?
- $\hat{A}$  is opening long-term connections. The average value of the connection

age opened by  $\hat{A}$  on your device is  $X$  which is  $Y$  times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?

- $\hat{A}$  is opening a relatively large number of connections. The average value of the number of connections opening by  $\hat{A}$  on your device is  $X$  which is  $Y$  times the norm. Is this behavior considered normal? How confident are you of this answer? What do you think the majority will reply?

The average value of the feature in the above six questions is the mean of all the readings recorded by CrowdApp related to this feature and this app in the time period considered for the targeted user. The norm is the mean of all the readings recorded by CrowdApp related to this feature and this app on all user devices. Thus, in addition to providing the user with his average value of the recorded feature that triggered the event, CrowdApp v2.0 will also display the average value across all user devices so that the user is more informed before replying.

An example of the push notification that is sent to selected users is shown in Figure 5.2. The user's reply to the first question is Yes, No, or I don't know. His reply to the confidence question is any value between 0 and 5 in steps of 0.25. And his reply to the third question is Yes, No, The majority will reply with "I don't know", or I don't know what the majority will reply. In addition, another link button labeled "I don't understand the question" will provide the users with detailed explanations in case they do not understand the question. Explanations for the possible questions regarding app  $\hat{A}$  are:

- The CPU (Core Processing Unit) is the brain of your device. CPU usage refers to your device's processor and how much work it is doing. Every

app on your device will use a percentage of your CPU for a period of time. Some apps can use up a large amount of CPU without you even knowing it. This usually results in slow performance, freezing, and sudden shutdowns of your device. CrowdApp detected that  $\hat{A}$  is consuming a relatively large amount of your device's CPU. If you believe this is a normal behavior for  $\hat{A}$ , click on 'Yes'. Otherwise, click on 'No'. If you still don't understand the question or you're simply not sure of the answer, click on 'I don't know'.

- When your device runs any application, it stores temporary files in RAM (Random Access Memory). All running applications on your device consume varying quantities of RAM. Clearing the RAM on a device improves performance by freeing up space so that the device can run faster.  $\hat{A}$  has a relatively high memory usage meaning that it is taking up a large portion of your device's RAM. This means that  $\hat{A}$  might be responsible in part for degrading the performance of your device and causing it to run slower. If you believe this is a normal behavior for  $\hat{A}$ , click on 'Yes'. Otherwise, click on 'No'. If you still don't understand the question or you're simply not sure of the answer, click on 'I don't know'.
- $\hat{A}$  is receiving a suspiciously large number of packets/bytes. This means that data is being downloaded to your device. By data, we mean images, videos, music files, programs, etc. This may result in draining your data plan. If you believe this is a normal behavior for  $\hat{A}$ , click on 'Yes'. Otherwise, click on 'No'. If you still don't understand the question or you're simply not sure of the answer, click on 'I don't know'.
- $\hat{A}$  is sending a suspiciously large number of packets/bytes over the Internet. This means that data is being uploaded from your device. By data, we mean

images, videos, music files, etc. This may result in draining your data plan. If you believe this is a normal behavior for  $\hat{A}$ , click on ‘Yes’. Otherwise, click on ‘No’. If you still don’t understand the question or you’re simply not sure of the answer, click on ‘I don’t know’.

- Apps running on your device can open connections with several IP addresses at a time. CrowdApp is constantly monitoring these opened network connections for every app. This is useful to detect when hidden apps are connecting to remote servers or when an app has a suspiciously large number of opened connections. We have detected that  $\hat{A}$  is one of those apps with a large number of opened connections. If you believe this is a normal behavior for  $\hat{A}$ , click on ‘Yes’. Otherwise, click on ‘No’. If you still don’t understand the question or you’re simply not sure of the answer, click on ‘I don’t know’.
- Apps running on your device can open connections with several IP addresses at a time. CrowdApp is constantly monitoring these opened network connections for every app. This is useful to detect when hidden apps are connecting to remote servers or when an app has suspiciously long-term opened connections. We have detected that  $\hat{A}$  is one of those apps with long-term opened connections. If you believe this is a normal behavior for  $\hat{A}$ , click on ‘Yes’. Otherwise, click on ‘No’. If you still don’t understand the question or you’re simply not sure of the answer, click on ‘I don’t know’.

When the user provides a reply, it will be sent directly to the server. If no Internet connection is available, the answer will be saved locally and sent once an Internet connection becomes available.

**CrowdApp Monitor**

**aub.edu.lb.malapp is consuming a lot of down bandwidth. Your average value is 75 MB which is 2.37 of the norm**

Is this behavior normal?

Yes  
 No  
 I don't know

How Confident are you from this answer ?

★ ★ ★ ★ ★

What do you think the majority will reply?

Yes  
 No  
 The majority will reply with "I don't know"  
 I don't know what the majority will reply

**SUBMIT**

[I don't understand the question](#)

Figure 5.2: Sample CrowdApp v2.0 push notification



## Rating Process

After sending push notifications to the selected users, the server waits for a sleep period which is stored in the database. This sleep period (e.g. one hour) is set to give a chance for most users to send their reply back to the server. At the end of the sleep period, the rating process of app  $\hat{A}$  takes place.

The server selects all the user replies that were collected during the sleep period. It computes the number of users who replied No,  $C_{No}$ , and the number of users who replied Yes,  $C_{Yes}$ . The server discards all “I don’t know” answers which relate to the current asked question. It also computes average weights of the users who answered No in this round,  $AW_{No}$ , and average weights of the users who answered Yes in this round,  $AW_{Yes}$ . These values are stored in a specific table in the database. Depending on the aggregation technique that will be employed by the server, these weight values will differ.

In the case of Plurality, the weights of all users will be set to 1 giving the same advantage to all user replies.

In the case of the Confidence-Weighted approach, the weights will be the reported confidence scores of the users whose replies we are taking into consideration. In this case, we are weighing the reply of every user by the normalized value of his reported confidence score thus giving higher weights to users who are more confident in their replies.

In the case of the Maximum Confidence approach, the weights of the selected users will be set to one. However, the selected users will be those who reported a maximum confidence value. In this case, we are selecting a subset of users who are most confident of their replies and aggregating the replies from this subset in the same way as in Plurality voting.

In the case of the Competence-Weighted approach, the weights will be the

derived competence scores from an assumed pre-selected Dunning-Kruger curve that represents our crowd as was described in Chapter 4. Based on the reported confidence score of every user, we derive his competence score from the DK plot and then use this competence value to weigh his reply thus giving more competent users higher weights.

To compute the new value of  $\hat{A}$ 's reputation, the following takes place:

- $\sum CY$  is computed - This is the summation of the multiplication of the number of users who replied Yes by their average weights in all previous rounds from the beginning of time.

$$\sum CY = \sum (C_{Yes} \times AW_{Yes}) \quad (5.4)$$

- $\sum CN$  is computed - This is the summation of the multiplication of the number of users who replied No by their average weights in all previous rounds from the beginning of time.

$$\sum CN = \sum (C_{No} \times AW_{No}) \quad (5.5)$$

- $R_{\hat{A}}$  is computed - This is the updated value of  $\hat{A}$ 's reputation for a selected feature. It is based on all previous rounds where user replies were collected. This value is not normalized.

$$R_{\hat{A}} = \frac{\sum CY - \sum CN}{\sum CY + \sum CN} \quad (5.6)$$

- $R_{\hat{A}}$  is stored in the database as the reputation of the app for the selected feature.

- An overall reputation,  $OR_{\hat{A}}$ , is computed as the average reputation of the app over all its feature reputations.
- Normalized reputations for all apps in the database are updated every time an app's reputation is updated. The normalization takes place as shown in the equation below where  $NR_{\hat{A}}$  is the normalized reputation of app  $\hat{A}$ .  $R_{\min}$  is the smallest reputation value in the database, and  $R_{\max}$  is the largest reputation value in the database.

$$NR_{\hat{A}} = \frac{OR_{\hat{A}} - R_{\min}}{R_{\max} - R_{\min}} \quad (5.7)$$

- Normalized reputations are updated in the database Apps table.

The expertise of the users who sent replies to the server is also updated. Users whose replies agree with the global majority will have their Correct\_Detection\_Counter (CDC) incremented. And users whose replies did not agree with the global majority will have their InCorrect\_Detection\_Counter (IDC) incremented. When the CDC or IDC value of any user is updated, normalized values of all user expertise are updated as well. First, we compute the sum of CDC and IDC for all the users and record it as total\_DIFF. DIFF per user is then calculated as follows:

$$DIFF = (CDC_{\hat{u}} - IDC_{\hat{u}}) \times \frac{CDC_{\hat{u}} + IDC_{\hat{u}}}{total\_DIFF} \quad (5.8)$$

$DIFF_{\max}$  and  $DIFF_{\min}$  are selected from the resulting DIFF values of all users. The normalized user expertise of user  $\hat{U}$  is computed as shown below and stored in the database:

$$\text{NCE}_{\hat{u}} = \frac{\text{DIFF}_{\hat{u}} - \text{DIFF}_{\min}}{\text{DIFF}_{\max} - \text{DIFF}_{\min}} \quad (5.9)$$

## 5.2.4 Implementing Feature Collection

In the following subsections, we describe the process of collecting different features that are related to specifications of third-party apps on devices as well as their usage statistics.

### CPU, RAM, and Usage Stats

To collect CPU and RSS readings, the Linux `top` command is run on a user's device every  $T_{CM}$  seconds. A sample output is shown in Figure 5.3. The output is then parsed as follows. Only the tasks that are owned by the effective user (UIDs that begin with `-u0_` in the figure) are considered since our concern is with third-party applications installed by the user. The values for CPU and RSS are extracted for each running task (identified by the value under `Name` in Figure 5.3) and stored in the respective log files on the device. `PCY` is either `fg` (task is in foreground) or `bg` (task is in background). If one of the third-party apps is running in the foreground, its usage time value is updated by adding  $T_{CM}$  seconds to it. Additionally, if this third-party app wasn't previously in the foreground, its opening frequency value is incremented by 1.

### Traffic Stats

The `TrafficStats` class in Android is used to collect `TxBytes`, `RxBytes`, `TxPackets`, and `RxPackets` for every third-party app (defined by its UID) on a user's device. Since traffic stats are cumulative, every  $T_{TR}$  seconds, the four previous traffic

```

u0_a201@ja3g:/ $ top -n 1 -d 0

User 7%, System 23%, IOW 0%, IRQ 0%
User 3 + Nice 0 + Sys 9 + Idle 27 + IOW 0 + IRQ 0 + SIRQ 0 = 39

  PID PR  CPU% S   #THR   VSS     RSS PCY UID      Name
27159 3   17% R     1   1352K   488K fg u0_a201 top
 1393 2    2% S     1     0K     0K  root  irq/539-SSP_Int
 6686 2    2% S    31 949316K 11868K bg u0_a200 com.nike.plusgps:fullpower
 1488 1    2% S     1     0K     0K  root  kswitcher_1
    7 0    0% S     1     0K     0K  root  watchdog/0
   20 1    0% S     1     0K     0K  root  khelper
  449 0    0% S     1     0K     0K  root  sync_supers
  451 1    0% S     1     0K     0K  root  bdi-default
  453 1    0% S     1     0K     0K  root  kblockd
  468 0    0% S     1     0K     0K  root  spi2
  471 1    0% S     1     0K     0K  root  spi3
  479 2    0% S     1     0K     0K  root  khubd
  497 0    0% S     1     0K     0K  root  irq/524-max7780
  524 0    0% S     1     0K     0K  root  irq/519-sec-pmi
  609 2    0% S     1     0K     0K  root  cfg80211
  710 1    0% S     1     0K     0K  root  khungtaskd

```

Figure 5.3: Sample output of Linux top command

values of every app are subtracted from the four newly collected traffic values. The differences in TxBytes, RxBytes, TxPackets, and RxPackets are logged in their respective binary files on the device.

### Connection Stats

To collect connection stats, the Linux netstat command is run every  $T_{CxN}$  seconds. A sample output is shown in Figure 5.4. The output is then parsed to get the list of protocols that are being used. In the example shown in Figure 5.4, tcp and tcp6 are used. Then for each available protocol in the netstat output, CrowdApp v2.0 reads the corresponding file at /proc/net and extracts connection information. The result is further parsed to get the source IP, source port number, destination IP, and destination port number. The UID is also extracted which gives us the corresponding app name. This process repeats every  $T_{CxN}$  seconds. If the same connection (defined by the app name, source IP, source port number, destination IP, and destination port number) persists after  $T_{CxN}$

```

u0_a201@ja3g:/ $ netstat -a
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.168.43.107:52517    149.154.167.91:443     ESTABLISHED
tcp6       0      24 :::ffff:10.168.43.107:60515 :::ffff:216.58.210.36:443 LAST_ACK
tcp6       1      0 :::ffff:10.168.43.107:44773 :::ffff:216.58.205.106:443 CLOSE_WAIT
tcp6       1      37 :::ffff:10.168.43.107:52499 :::ffff:216.58.205.106:443 CLOSE_WAIT
tcp6       1      37 :::ffff:10.168.43.107:33171 :::ffff:216.58.210.238:443 CLOSE_WAIT
tcp6       1      0 :::ffff:10.168.43.107:50908 :::ffff:216.58.205.202:443 CLOSE_WAIT
tcp6       1      1 :::ffff:10.168.43.107:58817 :::ffff:216.58.198.3:443 LAST_ACK
tcp6       1      0 :::ffff:10.168.43.107:52754 :::ffff:216.58.210.194:443 CLOSE_WAIT
tcp6       1      37 :::ffff:10.168.43.107:48624 :::ffff:216.58.210.238:443 CLOSE_WAIT
tcp6       1      0 :::ffff:10.168.43.107:44698 :::ffff:216.58.210.238:443 CLOSE_WAIT
tcp6      38      0 :::ffff:10.168.43.107:50931 :::ffff:92.123.229.6:443 CLOSE_WAIT
tcp6       1      0 :::ffff:10.168.43.107:60457 :::ffff:216.58.198.164:443 CLOSE_WAIT
tcp6      38      0 :::ffff:10.168.43.107:52253 :::ffff:54.69.228.107:443 CLOSE_WAIT
tcp6       0      24 :::ffff:10.168.43.107:60696 :::ffff:216.58.205.138:443 LAST_ACK
tcp6       1      0 :::ffff:10.168.43.107:46273 :::ffff:216.58.198.14:80 CLOSE_WAIT
tcp6       1      0 :::ffff:10.168.43.107:44682 :::ffff:216.58.210.194:443 CLOSE_WAIT
tcp6       1      37 :::ffff:10.168.43.107:40742 :::ffff:216.58.210.238:443 CLOSE_WAIT
tcp6       0      0 :::ffff:10.168.43.107:42712 :::ffff:54.192.217.178:80 ESTABLISHED
tcp6       1      1 :::ffff:10.168.43.107:55907 :::ffff:216.58.205.138:443 LAST_ACK
tcp6       0      0 :::ffff:10.168.43.107:43194 :::ffff:54.77.125.154:5223 ESTABLISHED
tcp6       1      0 :::ffff:10.168.43.107:53418 :::ffff:216.58.210.194:443 CLOSE_WAIT
tcp6       1      1 :::ffff:10.168.43.107:51075 :::ffff:216.58.198.14:443 LAST_ACK
tcp6       0      0 :::ffff:10.168.43.107:59271 :::ffff:31.13.73.34:443 ESTABLISHED

```

Figure 5.4: Sample output of Linux netstat command

seconds, its Age value is updated by adding  $T_{C \times N}$  seconds to it. Only when a connection ends, it is logged in the proper binary file on the device to ensure that there is no redundancy in logging the connection stats. For every third-party app, the total number of opened connections is counted where two different connections will differ in at least one of the defining elements (source IP, source port number, destination IP, destination port number, and protocol).

### Screen Activity

To monitor the screen activity, a broadcast receiver is used to detect when the screen is switched on or off. The receiver calls a method to log the screen activity to binary backup files.

### Install Period

Upon installation of CrowdApp v2.0, the install time of all currently installed third-party apps is collected using the PackageManager class in Android which

retrieves information related to packages on the device. Additionally, a broadcast receiver is used to detect when a new package is installed on the device and record the time it was added. Another broadcast receiver is used to detect when a package is uninstalled and record the time it was removed. The install and uninstall time (if any) of all third-party apps are sent to the server. The install period of every app on a user device can then be computed as the difference between these two values.

Even though reading stats from the device takes place every  $T_{CM}$ ,  $T_{TR}$ , or  $T_{C \times N}$  seconds, however, the process includes running Linux commands and parsing their outputs, reading stats from files on the device, and listening to broadcast receivers. The resulting processing is minimal and this is shown in CrowdApp's power consumption results in a following subsection.

### 5.2.5 App Permissions

To achieve the above functionalities, the permissions that are requested by CrowdApp are:

- `WRITE_EXTERNAL_STORAGE` to store collected logs on the device
- `READ_EXTERNAL_STORAGE` to read logs stored on the device
- `MODIFY_EXTERNAL_STORAGE` to modify logs stored on the device
- `RECEIVE_BOOT_COMPLETED` to restart the service whenever the device is booted
- `BATTERY_STATS` to hold off sending logs when the device's battery is low
- `INTERNET` to receive questions and send replies and log files to the server

- ACCESS\_NETWORK\_STATE to detect when the connection on the device changes
- ACCESS\_WIFI\_STATE to send log files only when the device is connected to WiFi

## 5.3 Server Side

For testing purposes, the backend was operating on a virtual machine with the following specifications:

- OS - Ubuntu 14.04 64-bit
- RAM - 2048 MB
- CPU - Single Thread Core i5 @ 2.67 GHz
- Web Server - Apache 2.4.7
- PHP Version - 5.5.49
- MySQL Version - 5.5.52

### 5.3.1 MySQL Database

Every user's ID, his generated username and Firebase token, his device's OS version, reported IT knowledge, his computed reputation, along with other entries related to threshold and interval updates are all stored in Users table.

Apps' names, package names, final reputation, final normalized reputation (which is shown to users of CrowdApp v2.0), and the reputation counter are stored in Apps table in the database. In app\_ratings table, we store every instance



of an app rating, that eventually averages out to the final reputation, along with a timestamp of when the rating took place. Every instance is in fact the result of aggregating user replies to a question. Note that every app will have a separate app rating per feature and the final app reputation is the average of the final values of the per-feature app ratings.

The table `user_apps` stores the ID of every user along with the IDs of all of his installed third-party apps. In addition, every app is stored with two timestamps, one for its installation data, and another for its uninstallation data, if any.

The table `app_usage` stores features related to the usage time and opening frequency of all third-party apps. The table `connection_count_stats` stores connection count per app and `connection_stats` stores the age. CPU and memory stats are stored in `cpu_memory_stats`. Every entry in the above tables is stored with a timestamp. All entries related to traffic are stored in the table `traffic_stats` along with their respective timestamps. Finally, every time the screen on a user device changes its status between ON and OFF, an entry is added in the `screen_activity` table along with the `user_id` and a timestamp.

Feature thresholds are stored in the `features` table and the different intervals (when to collect data, when to upload files, etc.) are stored in the `intervals` table. Every fired alarm along with `user_id`, `app_id`, `feature_id`, timestamp, difference between current time and previous alarm, value of the feature that triggered the alarm, and its relation to the norm are stored as entries in the `alarms` table. The table `questions` stores every question that is sent out to the users along with the respective `app_id` of the app that originally fired the alarm and the `feature_id` of the feature under consideration. Every question is stored with a timestamp. Another table called `user_questions` stores in every entry the `question_id` and the `user_id` of every user that the question was sent to along with a timestamp of

when the user question was sent. Replies from users are stored in `user_replies` along with the `user_id`, the `question_id`, the ID of the reply (Yes, No, etc.), the ID of the aggregated reply, the reported confidence of the user, the user’s answer to the surprisingly popular question, and a timestamp of when the user submitted his reply.

## 5.4 System Performance

After monitoring the machine for an hour, for a single user communicating with the system, the backend process was consuming around 0.46% of CPU and 2.9% of RAM. Measurements were taken using the `top` command where we monitored both the Apache server user ‘`www-data`’ and the MySQL user “`mysql`”.

As for CrowdApp v2.0, we conducted an experiment where it was installed on 4 devices (Samsung Galaxy S5, Samsung Galaxy S7, Samsung Galaxy Tab A, and Sony Xperia Z3) for a period of 3 hours. Logs were collected from the devices, notifications sent, and replies gathered from the users. At the end of the experiment, average values related to the performance of CrowdApp were computed and are shown in Table 5.1. The power usage over 3 hours averaged around 14 mAh (which is less than 1% of battery capacity on modern devices). Average RAM usage was 32 MB, average data usage was 0.325 MB, the app’s average storage size was around 30 MB, and its average file size was around 375 KB. These values demonstrate the feasibility of our approach in collecting data from devices and gathering input from the crowd which will ensure that users will not be discouraged to install the app on their devices.

Device	Model Number	Average RAM Usage (MB)	Maximum RAM Usage (MB)	Data Usage (MB)	File Size (KB)	Storage (MB)	Consumed Battery %	Power Used (mAh)
Samsung Galaxy S7	SM-G935FD	54	107	0	807	31.98	1	22
Samsung Galaxy Tab A	SM-P550	6.6	26	0.39	126	29.02	0	6
Sony Xperia Z3	D6633	32	62	0	155.3	29.09	0	9
Samsung Galaxy S5	SM-G900F	36	66	0.91	412	29.02	1	19
Average		32	65.25	0.325	375	30	0.5	14

Table 5.1: CrowdApp v2.0 performance specs

# Chapter 6

## Results and Analysis

Once CrowdApp v2.0 was implemented, we were ready to begin the experiment and have users download the application to their devices. However, considering that the application asks users questions related to the behavior of apps on their devices, our experiment is considered one which involves human subject testing. In addition, CrowdApp v2.0 collects user data from devices. Even if the collected data does not violate any privacy requirements, it is still a major concern to users. Before contacting possible participants, we had to get approval from the Institutional Review Board at AUB.

The approval process is described in the following section. Then, we briefly describe the experiment and present a list of the collected data. Finally, we present our results related to CrowdApp's performance in classifying apps given different aggregation techniques.

## 6.1 Institutional Review Board

The role of the Institutional Review Board at AUB is to safeguard the rights and welfare of human subjects who participate in research activities conducted under the support of AUB and AUBMC. Since our designed application queries participants regarding apps on their devices, we submitted a request to IRB to be able to proceed with our experiment. We requested approval to send an email to the student body in the Faculty of Engineering and Architecture. The email invitation script is shown below:

I am inviting you to participate in a research study about detecting mobile malware on Android phones through crowdsourcing.

You will be asked to install an app on your Android device. The app will be running transparently in the background. Throughout the period of the experiment, you will occasionally receive notifications from the app in the form of Yes/No questions. These questions will be related to the behavior of other apps that are installed on your device and whether or not you think this behavior is considered normal for the app in question. Note that if your device is not compatible, you will be notified upon installation and would not be able to be part of the study.

Your participation should take approximately two months. Please understand your participation is entirely on a voluntary basis and you have the right to withdraw your consent or discontinue participation at any time without penalty. You are eligible for this study if you are aged over 18.

The research is conducted online and is hosted on an AUB server.

A draw on a smartphone will be performed at the end of the experiment.

If you would like to participate, please follow this [link](#) before November 3 and fill it out accordingly.

If you have any questions about this study, you may contact:

Farah Saab                      fws02@mail.aub.edu                      +961 70 904231

Prof. Imad Elhajj      ie05@aub.edu.lb

In addition, we were required by the IRB to submit a consent form that users have to agree to before participating in the experiment. The consent form should clearly explain to users what the experiment is intended for and how results from it will be used. The body of the consent form is shown next:

Hello. My name is Farah Saab. I am a graduate student in the Department of Electrical and Computer Engineering at AUB. I would like to invite you to participate in a research study about detecting mobile malware on Android phones through crowdsourcing. The recruitment of research participants will be done via an email sent by the research team through the ACPS. Our targeted sample size is around 200 participants. Only students aged 18 and above are eligible for this study.

Before we begin, I would like to take a few minutes to explain why I am inviting you to participate and what will be done with the information you provide. You will be asked to install an app on your Android device. The app will be running transparently in the background. Throughout the period of the experiment, you will occasionally receive notifications from the app in the form of Yes/No questions. These questions will be related to the behavior of other apps that are installed on your device and whether or not you think this behavior is considered normal for the app in question. You are free to send us any questions about the experiment.

I am doing this experiment as part of my studies at AUB. The application that you will install will be collecting non-private data from your devices such as the CPU and memory consumption of your third-party apps. The collected data will be used to monitor and detect abnormal app behavior on Android devices. The results from this experiment will constitute a major part of my dissertation. I may also use this information in articles that might be published, as well as in academic presentations. Your individual privacy and confidentiality of the information you provide will be maintained in all published and written data analysis resulting from the study. Please note that our developed app does not have access to any private data on your device (e.g. messages, media files, etc.). It does not collect any private information related to the user of the device. When you install the app, you will be registered on our server using the hash of a generated Google Firebase token to ensure your anonymity. We will not even know to whom the data on our server belongs to. Users who opt to participate in our experiment will be anonymous to us as well. We do not get their names, age, gender, or any other identification information related to them. The non-private data that we collect will also be encrypted before being uploaded to the server. Also note that all collected data will be hosted on an AUB server.

Your participation should take approximately two months. Please understand your participation is entirely on a voluntary basis and you have the right to withdraw your consent or discontinue participation at any time without penalty. Refusal to participate or deciding to withdraw from the study will involve no penalty or loss of benefits to which you are otherwise

entitled and neither will it affect your relationship with AUB. Also note that the app was designed to consume an insignificant amount of your device's resources. Its CPU, memory, battery consumption, data consumption, and file storage size are minimal. As payment for your participation, a draw on a smartphone will be performed at the end of the experiment period.

Please note that we will only keep the provided phone numbers till the end of the experiment period, after which they will be deleted. During the experiment period, we will only use the phone numbers for occasional notifications related to updating the app or uninstalling it from your device. The draw on the smartphone will be done using your phone numbers. The participant of the selected number will then be contacted to receive the phone.

If you have any questions regarding the Android application or the study in general, you may contact me at [fws02@mail.aub.edu](mailto:fws02@mail.aub.edu) or by telephone (70-904231). You may also contact Prof. Imad Elhajj at [ie05@aub.edu.lb](mailto:ie05@aub.edu.lb) for any of your concerns.

If you have questions about your rights as a participant in this research, you can contact the IRB office at AUB at [irb@aub.edu.lb](mailto:irb@aub.edu.lb), by telephone (01-350000 - 5445) or by fax (000961 1 738025).

## 6.2 Experiment Overview

After getting approval from the Institutional Review Board, an invitation email was set to around 1,400 students from the Faculty of Engineering and Architecture. Fifty participants signed up to participate in the experiment by agreeing to



the consent form and providing their phone numbers. After distributing CrowdApp v2.0 via WhatsApp, 33 users out of 50 installed it on their devices. The server specs are shown below:

- VMware Virtual Platform
- OS - Ubuntu Linux 16.04 kernel: 4.4.0
- CPU - x64 Intel(R) Xeon(R) CPU X5650 @ 2.67GHz
- Memory - 4GiB
- Disk - SCSI Disk 110GiB

We ran the experiment for a period of six months during which CrowdApp v2.0 was collecting all features related to app and device usage while monitoring events generated by apps, asking questions, gathering replies and confidence scores, and computing app reputations based on five different aggregation techniques. During this six-month period, each process corresponding to one of five aggregation techniques was kept running for around ten days which was enough for most app reputations to converge.

## 6.3 Collected Data

Throughout the ten days of every process, the number of events varied and the resulting questions asked varied as well. The average event count per day across the five aggregation techniques was around 8,000. The average number of questions asked was around 70 and the average reply count was 40 giving an overall reply rate of slightly over 50%. We show the resulting event count, question count,

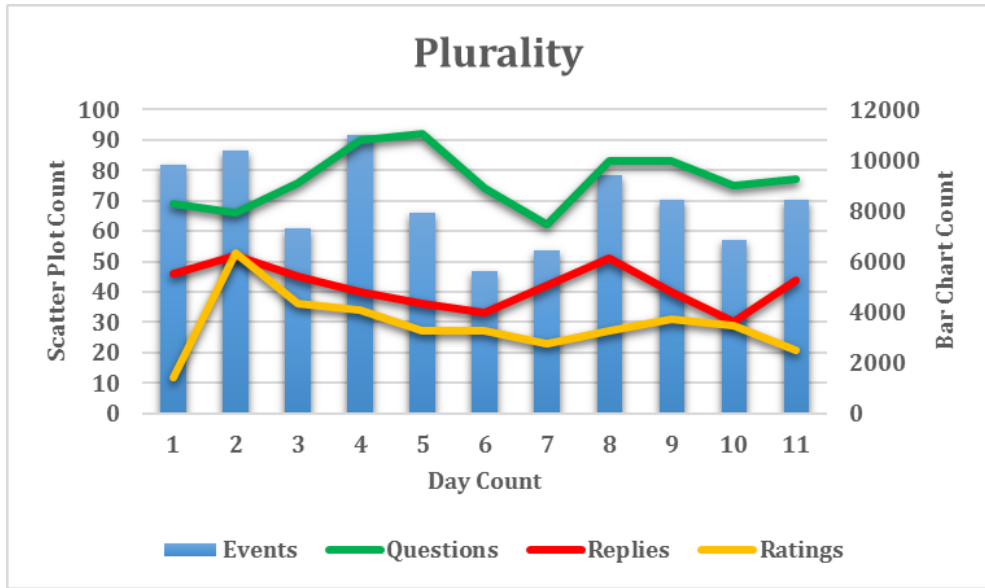


Figure 6.1: Rates of Plurality experiment

reply count, and rating count in each of the five experiments for the different aggregation techniques in Figures 6.1, 6.2, 6.3, 6.4, and 6.5, respectively.

In addition, all other app features were collected throughout the entire six-month experiment period. These features included traffic stats, CPU and memory stats, usage time and opening frequency of apps, the screen activity of a user’s device, etc. The size of the resulting database was over 2.5 gigabytes. Some of the tables were kept throughout the experiment period. Others were reset every time a new aggregation technique was tested. Tables that were reset at the beginning of every process’s experiment were Apps that has the final apps’ reputations relative to the current process, Users that has user’s reputations, alarms, questions, and replies tables.

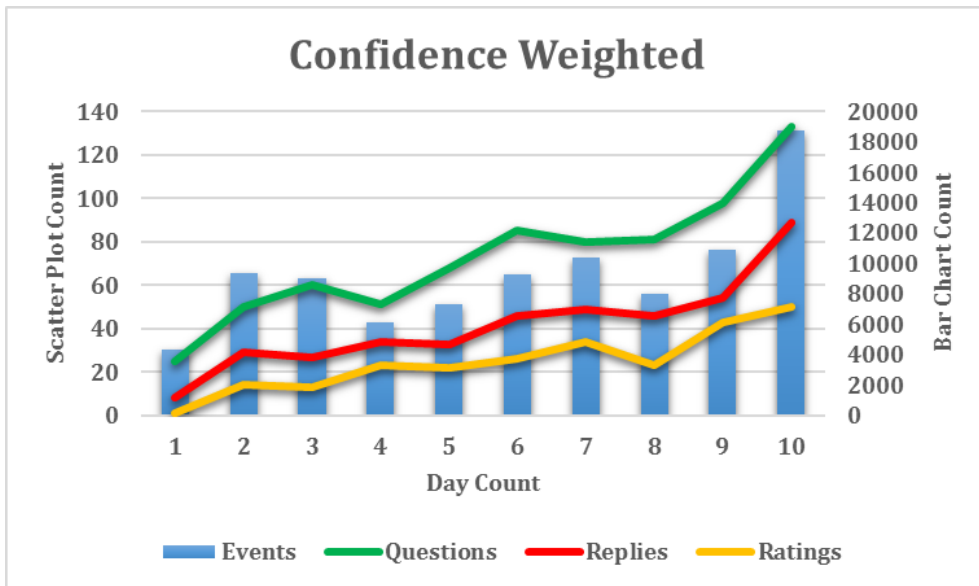


Figure 6.2: Rates of Confidence-Weighted experiment

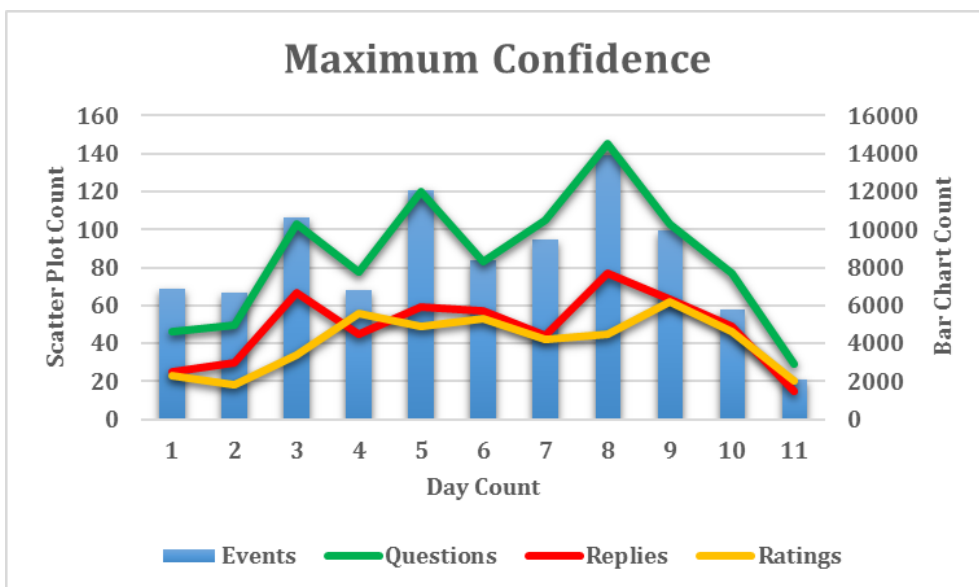


Figure 6.3: Rates of Maximum Confidence experiment

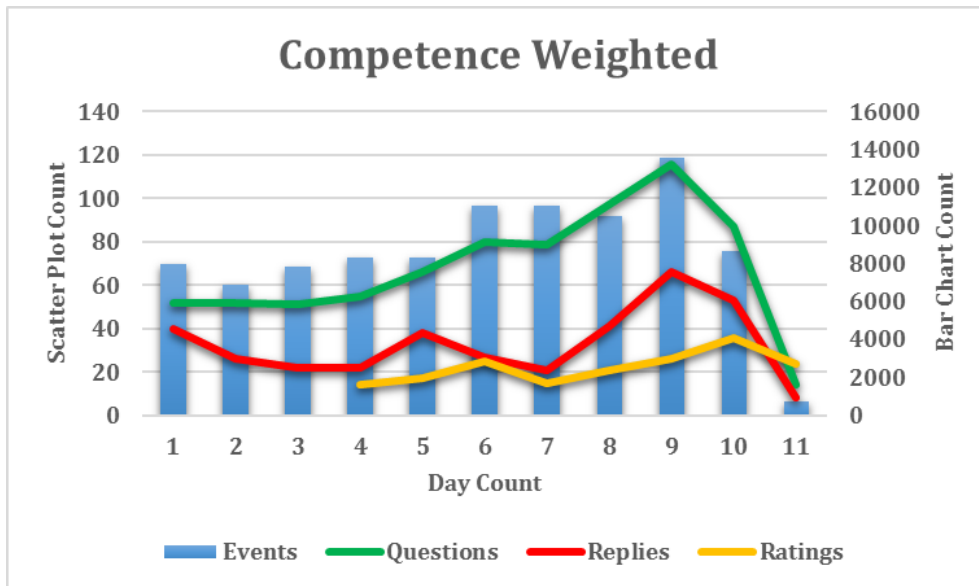


Figure 6.4: Rates of Competence-Weighted experiment

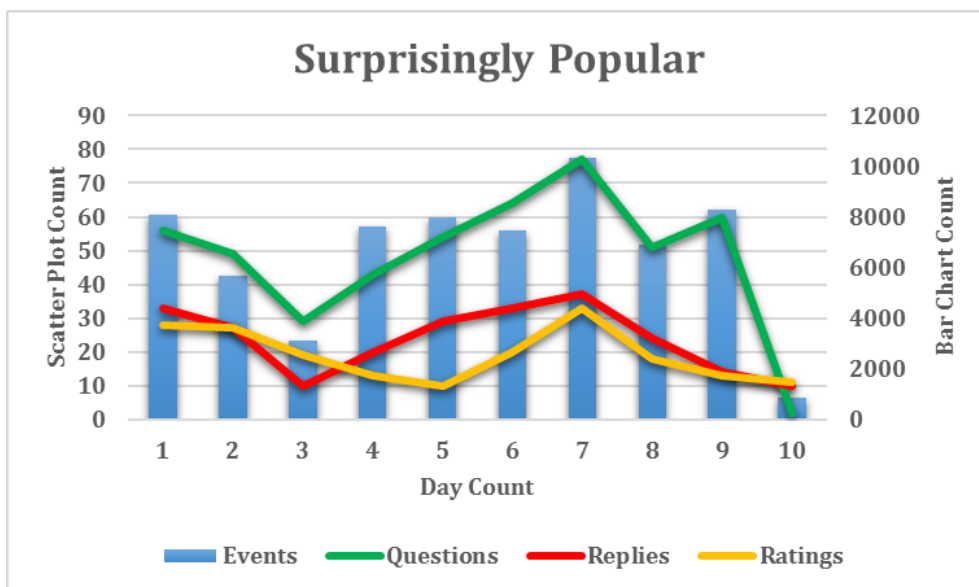


Figure 6.5: Rates of Surprisingly Popular experiment

## 6.4 Results of Aggregation Performance and App Classification

The two main objectives behind our experiment are related to the rating process. First, we want to study the performance of the different aggregation techniques when used in our system and compare the results to those in Chapter 4. Second, we want to study the composite ratings of apps that are provided by CrowdApp v2.0 at the end of the experiment period. We begin this section by describing how we computed the utility scores of apps offline and discussing what more needs to be done in this area. We then discuss the computed behavior scores of apps and compare the different scores resulting from each of the aggregation techniques. We further analyze the performance of these techniques with another controlled experiment in a following subsection.

### 6.4.1 Utility Scores

We computed the utility scores for apps that have been installed on at least four devices. We excluded apps that are from unknown sources since they do not have Google Play scores that we can use for comparison with our method's scores. We were left with 29 apps in total. For each app, we computed five scores.

The first score is the average opening frequency. It is computed by taking the total opening frequency per app per user and dividing it by the installation period of this app on this user's device. The result is then averaged over all users who have this app. After getting this score for all apps, we normalize it to the range  $[0, 5]$ .

The second score is the average usage time per opening frequency which is computed by simply dividing the total usage time per user per app over the total

opening frequency per user per app and then finally averaging over all users who have this app. After getting this score for all apps, we normalize to the range [0 5].

The third score is the install count per app which is simply the number of users in our experiment who have this app installed on their device. After getting this score for all apps, we normalize to the range [0 5].

The fourth score is the utility score of the app which is the average value of the first three scores. It is a score in the range of [0 5] that takes into consideration the average opening frequency of the app, the average usage time per opening frequency, and the relative install count on user devices. It is a good representation of the overall utility of the apps given the information at our disposal.

The fifth score is the Google Play score of all apps. To get this score, we first get the ratings of apps as shown on Google PlayStore. Then for every app, we also get the number of reviews that resulted in its rating. The review count for apps is then normalized to the range [0 1] and the resulting values are added to the rating of the app. The resulting value is the Google Play score. This is done as a means of weighing this app rating by considering the strength of the rating, i.e. how many users agree on the overall rating of this app.

In Table 6.1, we show the five computed scores for the 29 apps. The first four scores were computed from data collected in our experiment and the fifth score was computed based on data collected from Google PlayStore. The Pearson correlation between the utility scores and the Google Play scores is 0.72 which means there is a strong positive correlation and our technique can be used to infer the general utility of apps based on data collected from the devices related to app usage. It is worth noting that the data for utility score computation that was used here with CrowdApp v2.0 is different than the data used in Chapter 3

App	Score 1	Score 2	Score 3	Utility Score	Google Play Score
WhatsApp	2.65	0.11	5	2.59	5.28546
Clash of Clans	0.22	5	0.83	2.02	5.17438
DU Recorder	5	0.02	0.83	1.95	4.83331
Facebook	2.21	0.19	3.33	1.91	5.1
Anghami	1.2	1.82	2.08	1.7	4.50938
Candy Crush Saga	0.1	3.64	1.46	1.73	4.68706
Messenger	1.24	0.09	3.33	1.55	4.72533
Truecaller	2	0.2	2.29	1.5	4.60018
Instagram	0.09	0.09	2.5	0.89	5.35229
Telegram	0.98	0.05	0.83	0.62	4.43998
Skype	0	0.04	1.67	0.57	4.23406
Translate	0	0.04	1.46	0.5	4.47344
Zomato	0	0.26	1.25	0.5	4.30653
Dropbox	0	0.46	1.04	0.5	4.42374
Telly	0	0	1.25	0.42	4.00043
HP Print Service Plugin	0	0	1.25	0.42	4.41876
OLX Arabia	0	0.07	1.25	0.44	4.50268
Touch	0	0.12	1.04	0.39	4.3
Google Play Games	0	0	1.04	0.35	4.39171
Outlook	0.02	0.05	1.04	0.37	4.34155
AliExpress	0.14	0.05	0.83	0.34	4.67573
Perfect Piano	0	0.09	0.83	0.31	4.21053
Twitter	0.03	0.02	0.83	0.29	4.4492
Photos	0.09	0.02	0.83	0.31	4.63897
Sheets	0	0.05	0.83	0.29	4.30628
Pinterest	0.08	0.01	0.83	0.31	4.65506
Viber	0	0.02	0.83	0.28	4.44502
Adobe Acrobat	0	0.01	0.83	0.28	4.33852
Snapchat	0	0	0.83	0.28	4.21772

Table 6.1: Computed utility scores using CrowdApp v2.0

with CrowdApp v1.0. The previous version of CrowdApp was designed for rooted Android devices and therefore had access to several other app behaviors that are not accessible with CrowdApp v2.0 that was not designed with the requirement of a rooted device in mind. To this end, the utility score computation differed slightly, however the approach is similar to what was done before.

## 6.4.2 Behavior Scores

In this section, we focus our attention on apps that were rated by CrowdApp v2.0 ten times or more. We do not show results for apps that were rated less than

that. Our assumption is that if we are to consider the resulting rating accurate, it needs to be rated by a crowd of people. In this case, we consider rating instances equivalent to crowd members and proceed accordingly. First, we show the alarm distribution of some of the highest rated apps and comment on that. Then, we consider 8 of these apps that were rated in the five different processes with different aggregation techniques. We show their reputation convergence results in each process.

Since it is not possible for us to know which aggregation method is giving the most accurate results in the absence of the ground truth, we need to consider other approaches. To this end, we describe the False Event Rate controlled experiment that we performed in order to measure the performance of the different aggregation techniques.

### **Alarm Type Distribution**

In this section, we look at 9 of the highest rated apps throughout the entire experiment process. These apps are WhatsApp, Facebook, Messenger, Telegram, Pedometer, Candy Crush Saga, Clash of Clans, MiFit, and Popcorn Time. As mentioned previously, alarms that can be generated are those related to connection age and connection count, high CPU and memory consumption, and a high number of transmitted and received packets and bytes of data. We present the results in four tables.

In the first table, we get the average of all the alarms that are related to traffic (transmitted packets, received packets, transmitted bytes, and received bytes). We sort the apps in order of decreasing traffic alarm count as shown in Table 6.2.

We notice that the three COMMUNICATION apps, Telegram, WhatsApp, and Messenger, are resulting in the highest traffic alarm count. This is an



Sort by Traffic	
Application	Average
Telegram	546
WhatsApp	516
Messenger	380
Facebook	341
Clash of Clans	148
Candy Crush Saga	41
Popcorn Time	26
Pedometer	9
MiFit	0

Table 6.2: Average of traffic-related alarms

Sort by CxN	
Application	Average
Popcorn Time	8061
WhatsApp	3677
MiFit	3550
Messenger	3451
Facebook	2325
Candy Crush Saga	1404
Telegram	1121
Clash of Clans	763
Pedometer	333

Table 6.3: Average of connection-related alarms

expected result considering that these apps are also used for media sharing. HEALTH\_AND\_FITNESS apps such as Pedometer and MiFit have the lowest rate of traffic alarms. Clash of Clans has more traffic alarms than Candy Crush Saga since the game can only be played online versus Candy Crush Saga which has an offline option.

In the second table, we get the average of all the alarms that are related to connections (connection age and connection count). We sort the apps in order of decreasing connection alarm count as shown in Table 6.3.

We notice that Popcorn Time has the highest share of connection alarms.

Sort by CPU	
Application	Average
Candy Crush Saga	681
Clash of Clans	351
WhatsApp	267
Facebook	149
Messenger	98
Telegram	90
Pedometer	12
Popcorn Time	4
MiFit	1

Table 6.4: CPU-related alarms

This is expected since it is an app for P2P streaming and downloading of movies and series and so it will open connections with several IP addresses at a time and possibly for long periods. WhatsApp and Messenger have a higher connection alarm rate than Telegram and MiFit has a higher rate than Pedometer which might be due to issues such as syncing.

In the third table, we list alarms that are related to high CPU consumption and sort the apps in order of decreasing CPU alarm count as shown in Table 6.4.

It is not surprising that the two games in our list are on top of the chart in terms of CPU alarm count. Games are known to have high CPU consumption and result in quick battery drainage. Pedometer and MiFit have the lowest rate. WhatsApp has a higher rate when compared with Telegram and Messenger.

Finally, in the fourth table, we list alarms that are related to high memory consumption and sort the apps in order of decreasing alarm count as shown in Table 6.5.

As shown in the table, communication apps are generally consuming large portions of devices' RAMs which explains their high alarm rate. WhatsApp has the highest share of these alarms. These apps are usually opened frequently

Sort by RAM	
Application	Average
WhatsApp	4052
Messenger	3165
Facebook	2382
Telegram	1436
Pedometer	1393
Candy Crush Saga	1250
Clash of Clans	922
MiFit	794
Popcorn Time	580

Table 6.5: Memory-related alarms

throughout the day and when the user exists, they remain opened in the background. Unless the user explicitly frees up memory on his device by forcefully exiting, they will remain open.

Given these alarms and the replies that were collected from the users, we present different rating scores in the next section.

### Behavior Scores and Reputation Convergence

We provide five different behavior scores for every app from the list of highest rated apps. Each score is equivalent to one of five aggregation techniques. App reputations and alarm rates were reset in the database at the end of every experiment and they were recomputed every time. The results from the five experiments are completely independent. We kept every experiment running until app reputations stopped changing significantly (percentage change between consecutive rating instances ranged between 1% and 3%). The period was around 7 weeks on average for all experiments. The DK plot that was used to derive the competence scores of the users in our experiment from their reported confidence scores is  $y = 2x^2 - 2.5x + 1$ . We assumed a general crowd.

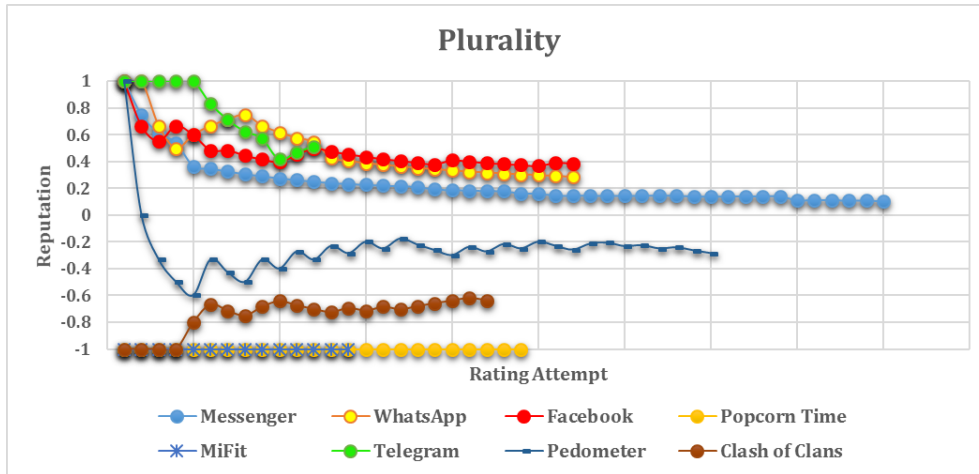


Figure 6.6: Behavior score convergence for selected apps given Plurality aggregation technique

The behavior score is a value between -1 and 1 for every app. We present in Figures 6.6, 6.7, 6.8, 6.9, and 6.10 five different behavior score results for each of the five experiments and for the highest rated and most popular apps. In every plot, the y-axis shows the reputation of the app and the x-axis shows the rating attempt number.

We can see how at the end of every experiment, most app reputations have converged to a certain value beyond which we expect it remains constant. However, the main result that we can take out from these figures is that every aggregation technique results in a different rating for some or all of the apps. If we consider Facebook for example, its resulting behavior score was 0.386 when the aggregation technique was plurality, 0.694 when it was confidence-weighted, 0.451 when it was maximum confidence, 0.167 when it was competence-weighted, and 0.196 when it was surprisingly popular. Note that not all apps have the same number of rating instances within each process and among different processes. The number of times an app gets rated depends on several factors, most important of which being the event rate which in turn is affected by the usage time on

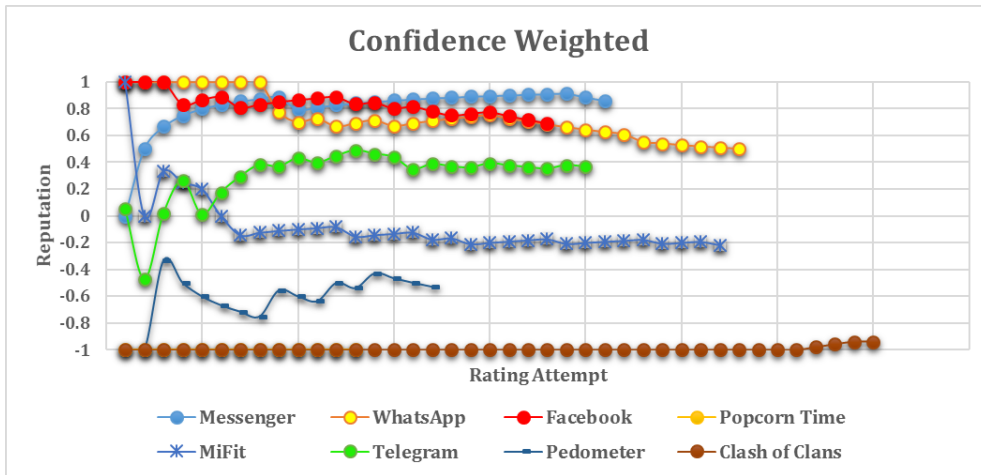


Figure 6.7: Behavior score convergence for selected apps given Confidence-Weighted aggregation technique

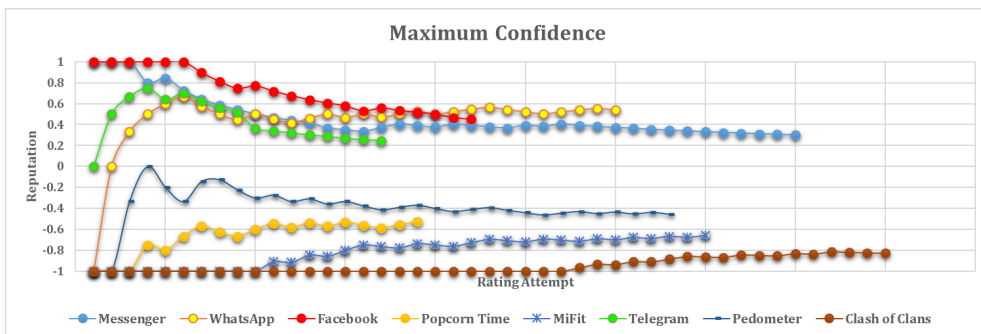


Figure 6.8: Behavior score convergence for selected apps given Maximum Confidence aggregation technique



	WhatsApp	Telegram	Messenger	Facebook	Clash of Clans	Popcorn Time	MiFit	Pedometer
PR	0.284	0.514	0.106	0.386	-0.636	-1	-1	-0.286
CF	0.499	0.365	0.861	0.694	-0.942	-1	-0.219	-0.529
MC	0.538	0.247	0.299	0.451	-0.829	-0.526	-0.657	-0.455
CP	-0.213	-0.034	0.395	0.167	-0.84	-0.909	0	0
SP	0.909	0.757	0.628	0.196	-1	-1	-1	0

Table 6.6: Final app reputations given different aggregation techniques

every device. Final reputations of the rest of the apps are shown in Table 6.6 for reference.

There is some agreement between the five techniques in certain cases. For example, using any of the five aggregation techniques will result in a negative rating for PopcornTime. However, in most cases, there is no agreement. What we can conclude from this is that the aggregation technique that is used will have a major effect on the resulting behavior scores of the apps being rated. And given that the ground truth is not available in our scenario, what we have available up to this point is not enough to assess the performance of the different methods. We provide a solution to this issue in the following subsections.

### False Event Rate Analysis

The objective behind our second experiment was two-fold. First, we wanted to test whether or not our system is able to detect a sudden increase in event rate which can result in cases where apps suddenly start to behave abnormally. In case our system detects this change, we are also interested to measure how long before such an app is detected. Second, we wanted to come up with a metric that will allow us to judge which aggregation technique had a better performance.

The False Event Rate experiment was designed as follows: Alarms and app reputations were once again reset in the database. For each aggregation technique, we chose one of the apps that it previously rated whose behavior score

was relatively high. The chosen apps were Airdroid which was given a score of 0.895 in the case of aggregating by plurality, Facebook which was given a score of 0.694 in the case of confidence-weighted aggregation, WhatsApp which was given a score of 0.538 in the case of maximum confidence aggregation, Messenger which was given a score of 0.395 in the case of competence-weighted aggregation, and Telegram which was given a score of 0.757 in the case of aggregating by the surprisingly popular approach.

We repeated the five experiments, this time only for a period of two days. We chose this small period of time because ideally we wanted to test whether our system will be able to quickly detect any abnormal behavior. In every experiment, we submitted false events for the app in question. To test the plurality aggregation technique, we repeated the experiment exactly as the first time, but we started submitting false events for the Airdroid app. We computed its average event rate per hour from the previous experiment. We began the second experiment with this average event rate as a base value and then doubled the false events every hour. This resulted in an exponentially increasing event rate for the Airdroid app under the plurality aggregation technique. The other experiments were repeated similarly, but for every experiment, we only faked event rates of the app that was chosen with an initial high reputation and the event rates of other apps were left as they are.

In an ideal scenario, given that the user is using an app in the same manner, these fake events are indicative of malicious behavior and should therefore alarm our system and the user. We expect that by increasing this event rate, the behavior scores resulting from a “good” aggregation technique should decrease tremendously. It is worth noting that the generated false events in each experiment were chosen randomly among the list of possible generated events (CPU,



memory, traffic, etc.).

We show in Figures 6.11 and 6.12 the change in the reputations of the five target apps, both in percentage and absolute format. The x-axis represents rating attempts with time. Our first observation is that the competence-weighted approach did not result in a steady decrease in the reputation of the apps as should be the ideal case. One possible reasoning behind this is that the DK plot that we chose to model our crowd is that of a general crowd. However, in our case, the crowd is mostly comprised of students from the faculty of engineering at AUB. This means that the majority of our users have the necessary background in order to properly assess app behavior on mobile devices. Representing our crowd with the assumption that most users will overestimate their competence might not have been the best approach and this resulted in sub-optimal performance of the competence-weighted approach. In order to study this effect on the performance of our CP method, we went back to the “Who Wants to Be a Millionaire?” dataset and repeated the competence-weighted approach, but this time with a different competence-confidence plot. Our chosen plot is shown in Figure 6.13. It is the mirrored function of the original DK plot that was used to compute the competence scores in Chapter 4. It is clear that this plot does not represent a general crowd whose members exhibit signs of the Dunning-Kruger bias. After computing competence scores based on this plot, and weighing user replies with these scores, the performance of this new competence-weighted approach was 85% (1591 correctly detected answers out of a total of 1881). This performance is worse than both plurality at 90% and confidence-weighted aggregation at 87% which agrees with the results in our CrowdApp v2.0 experiment. In the case of the “Who Wants to Be a Millionaire?” dataset, the users were from diverse backgrounds, and therefore, the crowd can be considered a general

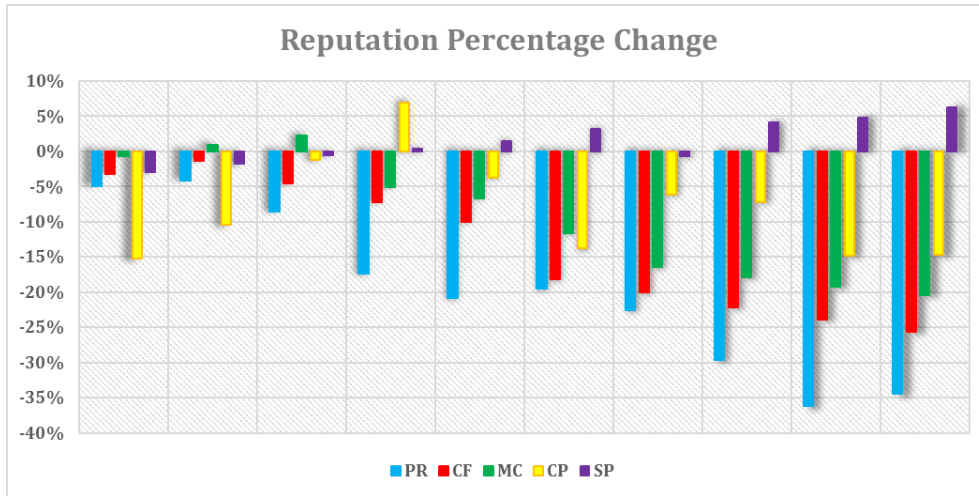


Figure 6.11: Percentage change in app reputation in the False Event Rate experiment

crowd. In our case, most users have a similar background that makes them good candidates to answer the questions asked by CrowdApp v2.0, and therefore, they cannot be considered a general crowd. A competence-confidence plot that works for one type of crowd does not have to work for other types.

Another observation is related to the surprisingly popular approach. Not only did the reputation not decrease, in later rating attempts, there is an increase in the reputation of the target app. Another conclusion that can be made is that the surprisingly popular aggregation approach does not have a good performance compared to other approaches.

The best performance was that of the plurality aggregation technique. It shows the largest decrease in the behavior score of its target app. The decrease even follows an exponential trend similar to the trend in submitting false events for the target app.

Next in line after the plurality approach is the confidence-weighted approach which also gave good results. However, its performance lags behind that of plurality aggregation. Similarly, the performance of the maximum confidence approach

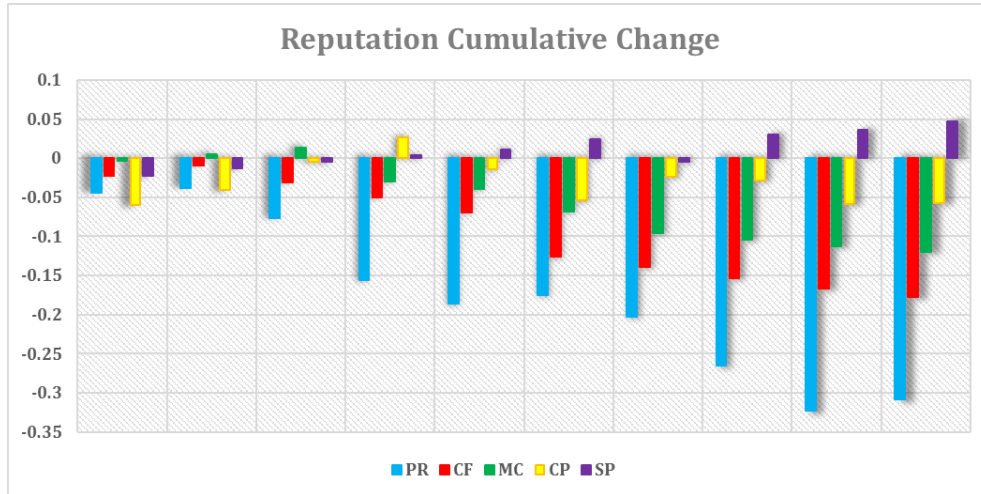


Figure 6.12: Cumulative change in app reputation in the False Event Rate experiment

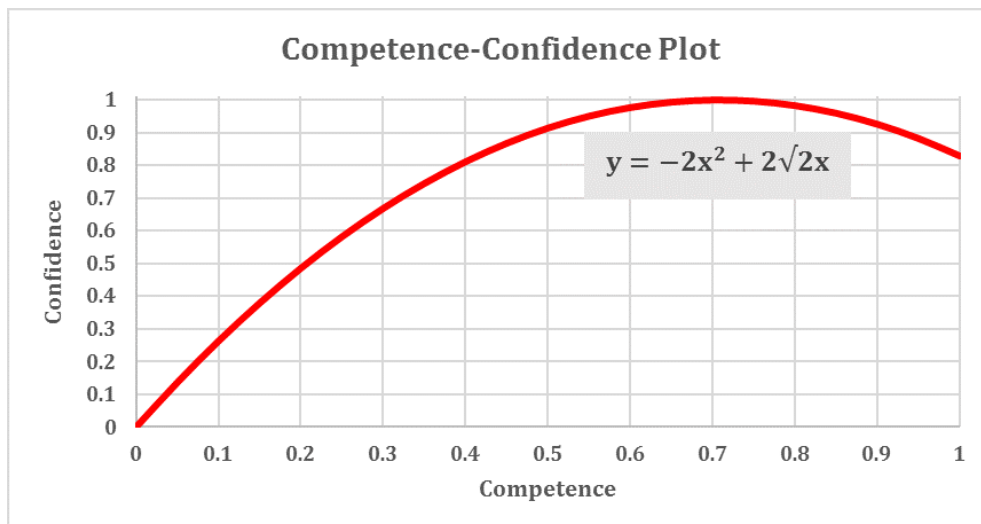


Figure 6.13: Mirrored confidence-competence plot

lags behind the performance of both plurality and confidence-weighted. These results agree with theorems in Chapter 4. We elaborate more on these results in the discussion section.

### **Designed Malware App**

Another experiment that we performed was to test whether our system can detect a newly introduced malicious app after all feature thresholds and reputations have converged. We designed and implemented a malware app, MalApp, and performed a controlled experiment where only a portion of the crowd (4 users) installed this app on their devices. MalApp generates abnormal activity in terms of CPU usage, memory consumption, network connections, and bandwidth consumption as follows:

- CPU stressing - We run 10 threads for 120 seconds. The threads are created and called by the main thread on MalApp. During this 120-second period, each thread keeps compiling a long regular expression.
- Memory stressing - We run a background service called by MalApp's main thread. The background service creates a new byte array  $[1024 \times 1024 \times 5]$  and fills it with random values. For each creation of a new byte array, the service allocates a memory block for the array. The service keeps allocating new blocks as long as there is available memory. The allocation repeats every 500 milliseconds until the memory is filled with random bytes.
- Network connections stressing - MalApp runs an AsyncTask to open N connections to a predefined IP and keeps them open. The connections are simple HTTP connections with a GET command. The number of connections is defined by the tester.

- Upload bandwidth stressing - In an AsyncTask, MalApp creates a text file with random strings. The text file size is around 10 MB. Then, MalApp uploads this file N number of times to a predefined IP. The number of uploads is set by the tester.
- Download bandwidth stressing - MalApp runs an AsyncTask to download a dummy file from a predefined IP. The file size is set by the tester. In our experiment, it was around 100 MB.

The main activity of MalApp is shown in Figure 6.14. In Figures 6.15, 6.16, and 6.17, we show the CPU and memory consumption of MalApp as monitored on one of the user devices during the experiment.

After running the experiment for only a couple of hours on four randomly selected user devices, several events were submitted by MalApp and questions were sent to these four devices. Users of these devices were asked to participate in this experiment over a period of a few hours and the employed aggregation technique was plurality voting. We show in Figures 6.18, 6.19, and 6.20 screenshots of some of the questions that were sent.

Users were asked to reply to these questions with the assumption that MalApp was a flashlight app instead of a designed malware app. Our main objective from this experiment was to check whether our system will manage to detect newly introduced malicious apps that are installed on a portion of the devices. We were able to show this through the questions that were being asked. However, we were also interested in the users' input to these questions assuming that this app was not a controlled app designed by us, but rather a normal app that is supposed to have a very basic functionality on their device such as a flashlight app. Of course, all users agreed that such a behavior demonstrated by the questions that were

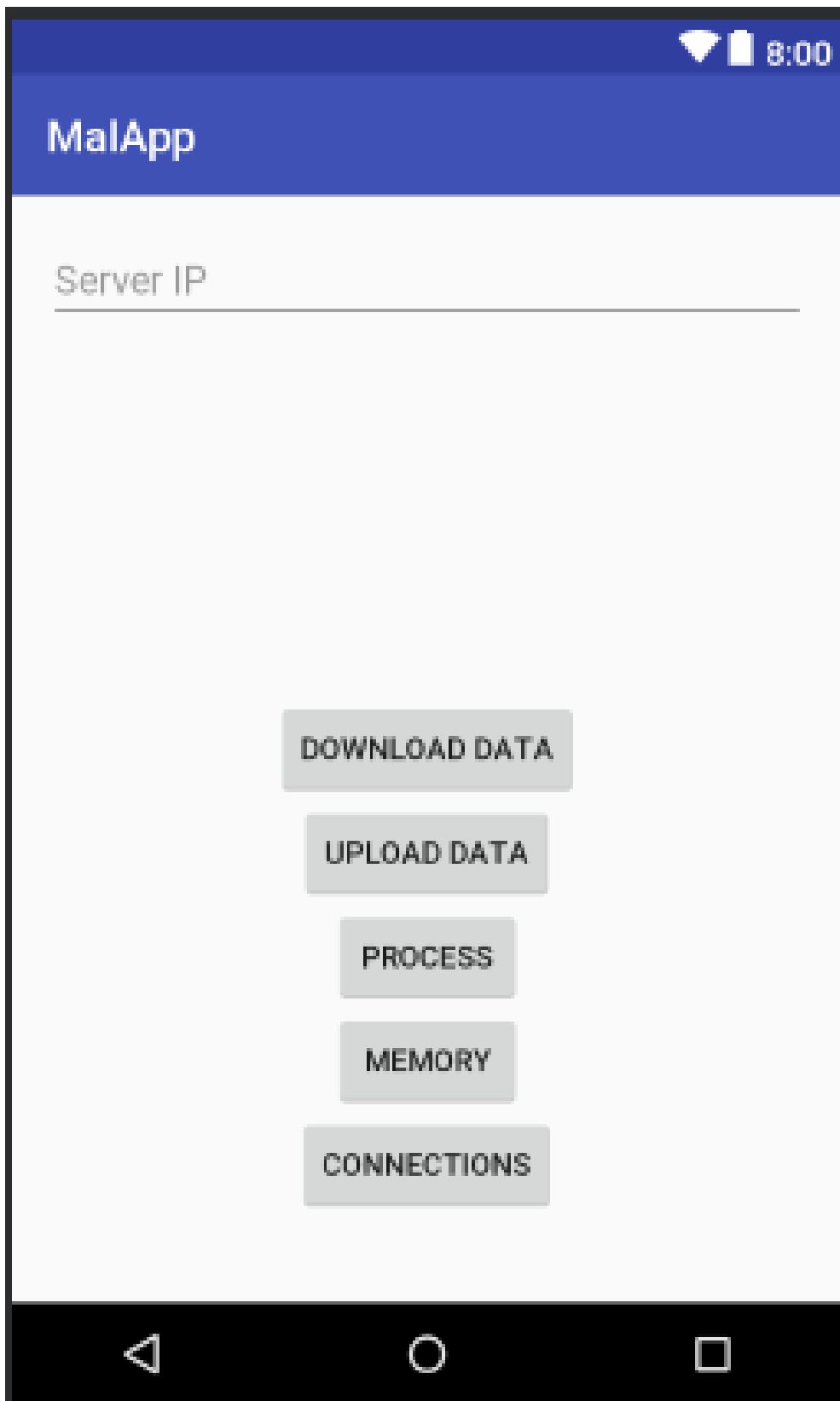


Figure 6.14: Main activity of MalApp  
196

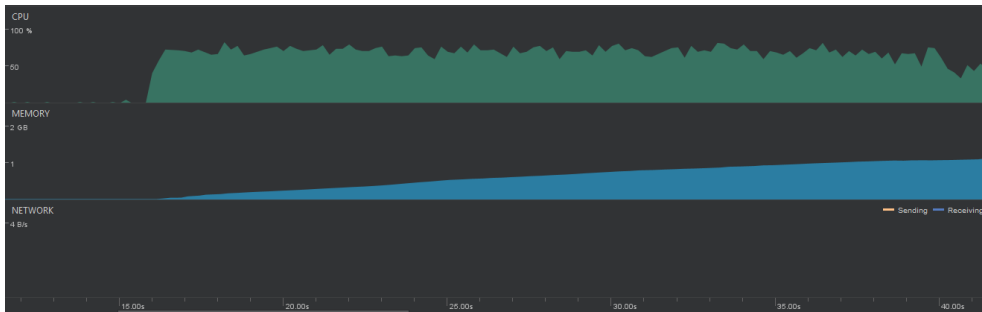


Figure 6.15: CPU and memory consumption of MalApp - part 1

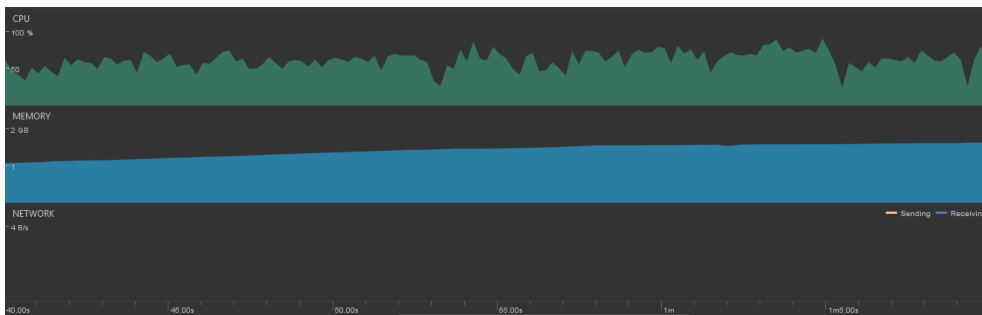


Figure 6.16: CPU and memory consumption of MalApp - part 2

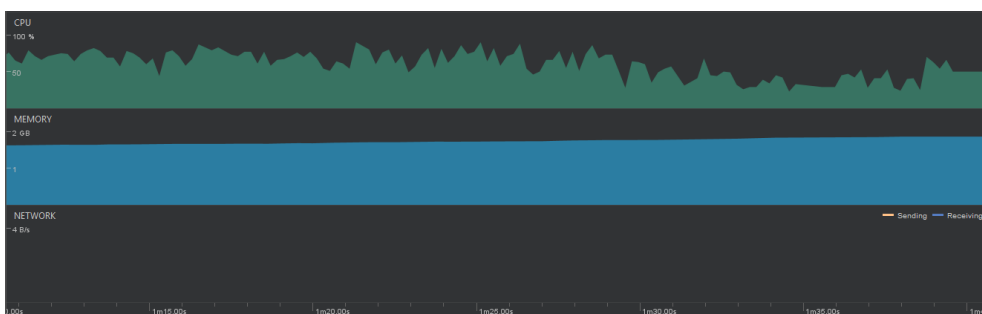


Figure 6.17: CPU and memory consumption of MalApp - part 3

CrowdApp Monitor

**aub.edu.lb.malapp has high RAM usage. Your average value is 70 MB which is 13.22 of the norm**

Is this behavior normal?

Yes  
 No  
 I don't know

How Confident are you from this answer ?

★ ★ ★ ★ ★

What do you think the majority will reply?

Yes  
 No  
 The majority will reply with "I don't know"  
 I don't know what the majority will reply

**SUBMIT**

[I don't understand the question](#)

Figure 6.18: Generated memory-related question for MalApp



CrowdApp Monitor

**aub.edu.lb.malapp is consuming a lot of down bandwidth. Your average value is 75 MB which is 2.37 of the norm**

Is this behavior normal?

Yes  
 No  
 I don't know

How Confident are you from this answer ?

★ ★ ★ ★ ★

What do you think the majority will reply?

Yes  
 No  
 The majority will reply with "I don't know"  
 I don't know what the majority will reply

**SUBMIT**

[I don't understand the question](#)

Figure 6.19: Generated down bandwidth question for MalApp

CrowdApp Monitor

**aub.edu.lb.malapp is consuming high up bandwidth. Your average value is 10 MB which is 1.92 of the norm**

Is this behavior normal?

Yes  
 No  
 I don't know

How Confident are you from this answer ?

★ ★ ★ ★ ★

What do you think the majority will reply?

Yes  
 No  
 The majority will reply with "I don't know"  
 I don't know what the majority will reply

**SUBMIT**

[I don't understand the question](#)

Figure 6.20: Generated up bandwidth question for MalApp

received was not a normal behavior for a basic app. The resulting reputation of MalApp decreased to -1.

## 6.5 Discussion

In terms of the behavior score, a major result that can be drawn from the experiment with CrowdApp v2.0 is related to the performance of plurality voting compared to other aggregation techniques. Plurality was shown to perform better than confidence-weighted and maximum confidence which agrees with the results in Chapter 4. We also reached a similar conclusion here concerning the better performance of confidence-weighted techniques compared to maximum confidence.

In terms of the competence-weighted approach, our main conclusion is that the representation of the crowd is one of the main issues that needs careful consideration. A competence-confidence plot that works for one crowd does not necessarily work for other types of crowds as was shown in the simulations in Chapter 4 versus the experiment results in this chapter. A proper crowd representation is one of the major requirements when using our competence-weighted technique.

Our system was shown to detect abnormal app behavior in a small amount of time. Plurality, confidence-weighted, and maximum confidence approaches resulted in decreased app reputations after only two days. This means that our system is able to detect abnormalities in the network. It also reflects how users in our crowd are aware of the functionality of their apps and are rating accordingly rather than just answering questions randomly.

# Chapter 7

## Conclusions and Future

### Directions

The large variety of mobile apps brought along considerable malware that has infected millions of Android devices throughout the years. Moreover, the number of written reviews which convey useful information regarding both utility and maliciousness of apps is fairly limited. And for these reviews to be reliable, a user will have to read a large number of them in order to form a basic idea regarding the status of an app. This, of course, is time-consuming and still not very reliable considering that users who write these reviews are not necessarily objective or experts. On the other hand, high or low numerical ratings of apps are easier to examine and are provided by a large number of users. However, they do not really describe the shortcomings, if any, within an app. A low app rating could be attributed to a variety of different reasons. And depending on user expectations, their rating of an app will be different.

There is no fine-grained analysis of apps on app stores nowadays. To the best of our knowledge, no one has attempted to provide a composite rating of mobile

applications which captures both security and utility. In this dissertation, we designed and implemented a system to compute such a composite score for every app on a user device. The challenge in our work was capturing context and user perspective which is not possible without crowdsourcing input from app users.

In the process of designing our system, we came across several other issues related to aggregation techniques and modelling the human factor in crowdsourcing systems. We summarize the contributions of this dissertation in the following subsections.

## **7.1 Conclusions Related to the Game-Theoretic Composite App Rating System**

We modeled the interaction between our system and abnormally behaving apps using concepts from game theory. The derived model maximizes the profits of both our system and benign apps while minimizing the profits of detected malicious apps. Through several real-life scenarios, we showed how our game-theoretic system manages to detect most malicious apps it plays against, flag them, and identify authoritative users from the set of queried users in order to ensure accurate app ratings for increased system security.

Our system was shown to classify apps based on two sets of scores. The first is a utility score that reflects utility and is derived from collected features related to app usage. Our computed utility scores showed a correlation with Google Play scores that is higher than 83% which demonstrates the fact that our system is able to rate apps based on actual user experience without any direct participation from the users themselves.

The second derived score is a behavior score that reflects suspiciousness and is

computed based on event rates of apps where an event is defined as any suspicious incident that takes place on a device while a user is using the app in question. Our simulation results showed that our system was able to detect most apps and classify most users in only a few hours. Having a crowd with an average expertise value of 0.7 was shown to have good effects on system performance and the higher this value, the faster the system converges. On average, the system was converging after 12 hours with the percentage of tested apps being around 1.2% and that of users around 97%.

Finally, ROC curve analysis for both app and user classification supported the rest of the results by showing for what values of the crowd expertise our system achieves its best classification results. In the case of user detection, there is a tradeoff when choosing the optimal value of crowd expertise. For higher values, the FPR of user detection increases. But since app classification is the main goal behind our system, we can conclude that our base value of 0.7 will result in very good app and user classification.

## **7.2 Conclusions Related to Modelling of Aggregation Methods**

Aggregating replies from users in the case of behavior score computation is a major research problem that we have attempted to study. There are many factors that affect the choice of the aggregation method to adopt in crowdsourcing. Our conclusion is that the type of crowd is the most important of these factors. When knowledge within a crowd surpasses a given threshold, the choice of aggregation technique matters as we have shown in this dissertation.

Our main finding in the area of aggregation modelling is related to the Dunning-

Kruger effect. We modeled this psychological bias as a quadratic relationship between respondent competence and confidence. We then incorporated this effect into the model of different aggregation algorithms. Due to the general shape of the derived Dunning-Kruger function, we were able to validate that for a general crowd of respondents, a plurality aggregation method will, in most cases, outperform methods based on the confidence of respondents such as the confidence-weighted and maximum confidence approaches. We also showed that the maximum confidence approach generally lags behind the confidence-weighted approach since it focuses on the least competent respondents and disregards respondents whose reported confidence values were moderate, which we argue are the majority of the most competent respondents. We also showed how the size of the subgroup of respondents in the maximum confidence approach affects the overall performance of the method for a general crowd.

Using our competence-confidence function, we derived a competence-detection algorithm for input aggregation. Detecting voter competence is not an easy task especially in the absence of a ground truth. On the other hand, deriving voter competence from his reported confidence is fairly easy. Our detected competence-weighted algorithm had the best performance compared to other approaches when applied to the “Who Wants to Be a Millionaire?” dataset. However, when we considered a general crowd in our app rating experiment, the performance of the competence-weighted approach was poor. The main conclusion that can be drawn from this result is related to the knowledge available to us regarding the crowd. If we are able to correctly model a crowd’s competence-confidence plot based on known information related to the users in the crowd, we would expect good results when applying this competence-weighted aggregation method. Having no information about the crowd, however, renders this approach a suboptimal one.

## 7.3 Conclusions Related to CrowdApp Rating Scores and Performance

With CrowdApp v2.0, utility scores were computed using only some of the features that were used with CrowdApp v1.0 considering our removed restriction of having only rooted devices in the crowd, which is not a very a reasonable assumption in reality. Features were mostly related to app popularity, usage time, and opening frequency. Results showed high correlation with Google Play scores which validate, once again, the ability of our system to detect app utility without the crowd's explicit participation.

We repeated the process of behavior score computation five times. The result was five sets of scores for every app, each score based on one of five discussed aggregation techniques. The scores of each technique were different with some similarities for certain apps. Considering that the ground truth was not available, we performed several experiments to validate the performance of each technique. One of the experiments was the false event rate experiment where we intentionally manipulated the event rates of selected apps that originally scored well in the first experiment. Ideally, this should decrease the reputation considering that the behavior of these selected apps on the devices did not change to explain this increase in event count. Out of the tested techniques, plurality voting resulted in the highest decrease in reputation for its selected app, followed by confidence-weighted then maximum confidence. This validated our theorems concerning the performance of plurality voting when compared to confidence-related approaches. The surprisingly popular approach performed poorly, sometimes resulting in an increase in app reputation. The competence-weighted approach that was implemented using a DK plot for a general crowd also performed poorly. One



possible explanation was the poor representation of the competence-confidence plot considering that our crowd does not have the qualities of a general crowd. Our main conclusion is again related to the importance of properly modeling the competence-confidence plot of our crowd before attempting to vote using the derived competence-weighted approach. A poor model of the crowd will result in a performance that is worse than confidence-related approaches as was shown in both our simulations and a real-life experiment.

In terms of system reliability, we argue that our system is more immune to possible attempts by users to tamper with app scores than is the case with traditional rating systems. There are several approaches with which users can cheat the system and they are different depending on the type of score. We consider our two scores separately. The utility score is computed based on data collected from devices and without reference to the users themselves. Deciding not to include users in the utility score computation has its pros and cons. Getting input from users regarding their apps' utility levels can prove to be beneficial in terms of improving the accuracy of these reported scores. Rather than just inferring the utility by analyzing collected features, we can refer to the ultimate utility judge (the user). On the other hand, deciding not to include the crowd in the utility score computation actually diminishes the likelihood of users attempting to wrongfully affect the resulting utility score of a target app. Additionally, collected features are encrypted on user devices and only decrypted before sending them to the server side in a secure manner. This ensures that all collected data cannot be manipulated by users and thus guarantees the accuracy of our reported utility score. In terms of the behavior score, a possible approach could be to estimate the event rate of other apps on the network so as to ensure that submitted events by a target app do not exceed the resulting event threshold and ultimately

fire an alarm. However, this information is difficult if not impossible to obtain. Still, users are able to cheat our system much like in the case of traditional rating systems. When asked a question regarding a target app, a group of users can agree that its behavior is normal even if that is not the case. This is similar to when users give an app a high rating on a store even though it suffers from several drawbacks. The advantage of our system, however, is that the effect of this type of “attack” is less severe since it only affects a part of the final score of an app. Having a composite score is one of the ways to diminish the effects of possible cheating attempts and is a benefit of our composite rating system when compared to traditional systems.

In summary, there are several contributions in this dissertation. We start off with the designed and implemented composite app-rating system that provides users with a score based on a utility and a behavior measure. Traditional app rating systems provide users with one score that does not really reflect the strengths or shortcomings within an app. They also provide written reviews submitted by other app users. However, a user will have to read many reviews to gain insight regarding an app, and this, of course, is time-consuming. Our app rating system solves this issue by providing users with more detailed information regarding their apps than any other system on the market. By transparently crowdsourcing data from devices, it provides users with a utility score that is representative of the utility and ease of use of an app. Similarly, by crowdsourcing input from users regarding abnormal behavior on their devices, it manages to provide them with a score that is representative of the suspiciousness level of the app. Then, depending on the type of app, the two scores, and the user’s personal preferences, the app can be judged accordingly. To our knowledge, no one has attempted to do this before.

Another main contribution in this dissertation is related to the derived model of the Dunning-Kruger psychological bias which is one of the most critical human factors that affect the performance of crowdsourcing systems. We modelled this bias so as to better understand its effect on confidence-related aggregation techniques. We went on to model and compare the performance of some of the most popular aggregation techniques in crowdsourcing systems such as plurality voting, confidence-weighted voting, maximum confidence voting, and a newly derived competence-weighted approach. The modelling took into consideration the effect of the demonstrated Dunning-Kruger bias on output performance. We showed how this general modelling can be applied to improve the performance of any crowdsourcing system that employs one of the aforementioned aggregation approaches.

## 7.4 Future Directions

In terms of utility score computation, several improvements can be made. In order to differentiate between different types of apps that are used differently by users, one approach would be to consider the fluctuation of the frequency of recorded features rather than its absolute value. For example, an app such as WhatsApp will have a constant Opening Frequency if the chosen timespan is a day but a fluctuating one if it is an hour. The app is likely to be opened more at noon than after midnight. Some apps on the other hand, such as TripAdvisor, are opened a few times per year. They would have to be monitored for a longer period of time in order to determine an average Opening Frequency. This example illustrates how our system's performance can be greatly improved when it manages to identify different app types. A survey can be conducted on the different types of apps

on the market (social media, video conferencing, messaging, gaming, etc.) and the suitable timespan for each combination of app type and extracted feature. Identifying the app type can be done in two methods. One approach would be to add a questionnaire where the user is asked to categorize her apps into a set of provided categories. Another approach can be to employ an app identification technique that is based on transparently monitoring patterns in network traffic.

In terms of behavior score computation, the list of events that was presented here is not exhaustive. Only a few examples were provided to give an idea of what is meant by an event. Further research is needed to identify all types of events that might be issued by apps on a device, how each event type relates to every app category, and how each event can be detected.

In terms of aggregation techniques, the number of techniques that can be modeled is extensive. The studied aggregation techniques in this dissertation are some of the most popular and basic techniques. There is a lot of work that can be done in the area of modeling cognitive biases in state-of-the-art crowdsourcing systems that use aggregation techniques other than the ones presented in this dissertation. Our work on modelling aggregation techniques will pave the way for better understanding of the psychological biases that accompany metacognitive skills of respondents in crowdsourcing systems and eventually lead to a more productive utilization of the wisdom of the crowd.

Techniques for accurately deriving a crowd's competence-confidence plot should also be studied. A possible approach is to request additional information from users of our system, such as educational background and experience in the topic at hand. In the behavior score computation with CrowdApp v2.0, the competence-weighted approach performed poorly since we did not consider that our crowd does not fit into the category of a general crowd. An inaccurate crowd model

will have drastic negative effects on overall performance. To this end, designing a well-crafted set of questions that can be used to better infer competence levels of users is one possible approach. Replies to these questions along with reported confidence values, provided context, and replies to queries related to app behavior can then be used to train an AI engine to learn this context at a later stage. Given a fixed number of app categories and an expected behavior for each category, we can use all collected data from the crowd to improve our engine's accuracy and attempt to guess whether or not a crowd believes an app's behavior is normal later on. As the accuracy of this engine improves with time, our system will choose to crowdsource less often. This hybrid approach that uses both crowdsourcing and artificial intelligence might prove to be most optimal since it has the advantages of both learning from users when no decision can be made while also decreasing the likelihood of possible cheating attempts later on or suffering from other biases resulting from human judgment.

A better understanding of the crowd will render crowdsourcing systems more optimal. However, understanding a crowd means tapping into their way of thought and assessing their several biases. Traditional crowdsourcing systems are assessed without any consideration to human factors and this ultimately leads to suboptimal results that we are still unaware of. A variant of our work with modelling bias can be applied in electoral systems. A majority of voters suffer from many personal biases that render their decision of a representative a poor one when it comes to the overall benefit of their state or country. Even though these biases can vary depending on many factors such as race or region, their effect on the election process is significant and they should be accounted for. Another example is testing students in exams with multiple-choice questions. The reported confidence of a student can play a major role when evaluating his understanding

of the topic. Incorporating confidence scores in multiple-choice exams is already in its experimental phase in certain schools and universities and should provide more insight regarding proper question format and overall student understanding. Another possible application of our system is rating different cloud services by having CrowdApp deployed as a service on the cloud. Cloud computing is no longer only limited to storage or emails. It is used for development of software as well as testing and deployment. Examples of cloud services that can be rated are intelligent chatbots (Siri [146], Alexa [147], Google Assistant [148], etc.) that provide personalized context-relevant interactions with customers by leveraging the abilities of the cloud. In terms of productivity, tools such as Google Docs [149] and Microsoft Office 365 [150] use cloud computing to allow users to work on their documents from anywhere at any time. Microsoft Azure [151], Google Cloud Platform [152], iCloud [153], and CloudSim [154] are other examples of cloud-based platforms and services. Enterprises are moving rapidly to the cloud. Almost everything nowadays is stored and managed by cloud computing. And since the cloud can affect the performance of an app, we can easily deploy our system to assess different cloud platforms by using CrowdApp itself as a service on the cloud.

In terms of app rating, our developed composite score system is a step in the right direction. Scores offered on app stores nowadays are no longer representative of an app's general performance. With the large number of apps, users are becoming more and more selective when it comes to downloading a new app on their device. A number between 1 and 5 does not begin to convey the necessary information regarding an app. We have provided one alternative where an app is assessed based on both utility and suspiciousness level. In the future, we envision app rating systems that assess applications based on three or more scores, each

representing a different aspect that might be of interest to the user.

Given these user interests and our collected data related to how users believe apps should behave on their devices, our system will be able to learn the proper behavior per category. It can learn the expected rate of submitted events for any app in a certain category. Based on this acquired knowledge, many lessons can be learned and recommendations can be offered to app developers. For example, it was shown that most games issue many events related to traffic, CPU, and memory. However, they were shown to issue very few events related to opened connections. If a game issues many events related to connections, this will result in many fired alarms when comparing its event rate for this feature to the overall event threshold. What this means is that this particular game is more likely to get rated frequently due to its suspicious activity and this might result in a decreased behavior score. This insight can be conveyed to developers of this app who can then enhance the game's design. A high event rate does not necessarily convey maliciousness. It can also be the result of poor app design. Sharing our gained insight with app developers is a step in solving this issue.

# Appendix A

## Proofs of Theorems

### A.1 Theorem I - Plurality vs. Confidence-Weighting

Given any DK plot, plurality voting will perform better than confidence-weighted voting for tasks of higher difficulty.

$$\boxed{\forall(\mathbf{a}, \mathbf{b}) \quad \Delta_k \geq 0.5 \Rightarrow P_{\text{PR}} \geq P_{\text{CF}}}$$

Evaluations of the formulas for the success probability of both plurality and confidence-weighted voting are given below.

$$\begin{aligned} P_{\text{PR}} &= \int_0^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r \\ &= \frac{(\Delta_k - 1)(2\Delta_k + 2\Delta_k \tanh^{-1}[1 - 2\Delta_k] - 1)}{(1 - 2\Delta_k)^2} = F(\Delta_k) \end{aligned}$$



$$\begin{aligned}
P_{CF} &= \frac{\int_0^1 \left( (a(x_r^k)^2 + b(x_r^k) + 1)(x_r^k) \right) d\theta_r}{\int_0^1 \left( a(x_r^k)^2 + b(x_r^k) + 1 \right) d\theta_r} = \frac{\int_0^1 \left( a \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right)^2 + b \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right) + 1 \right) \cdot \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right) d\theta_r}{\int_0^1 \left( a \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right)^2 + b \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right) + 1 \right) d\theta_r} \\
&= \frac{((\Delta_k-1)(2\Delta_k-1)^3(-2\Delta_k-1)(-2(a+b+1)+\Delta_k(8-a+4b)+4\Delta_k^2(a-2))+4\Delta_k(3a(\Delta_k-1)^2+(2\Delta_k-1)(-1+2b(\Delta_k-1)+2\Delta_k)) \tanh^{-1}[1-2\Delta_k])}{(2(1-2\Delta_k)^4((2\Delta_k-1)(1+a+b-\Delta_k(4+a+3b))+2\Delta_k^2(2+b))+2\Delta_k(\Delta_k-1)(-2a-b+2\Delta_k(a+b)) \tanh^{-1}[1-2\Delta_k])}
\end{aligned}$$

### A.1.1 Case 1: $\Delta_k = 0.5$

The limit of  $P_{CF}$  as  $\Delta_k$  tends to 0.5 is  $P_{CF}^{0.5} = \frac{6+3a+4b}{12+4a+6b}$

The value of  $P_{PR}$  at  $\Delta_k = 0.5$  is  $P_{PR}^{0.5} = 0.5$

Note that for  $b = -a$ , the two probabilities are equal at  $\Delta_k = 0.5$  ( $P_{CF} = P_{PR} = 0.5$ ). As  $b$  tends to  $-2\sqrt{a}$ , the two functions intersect further to the left at a smaller value of  $\Delta_k$ .

*Proof by contradiction.*

$$P_{PR}^{0.5} < P_{CF}^{0.5} \Leftrightarrow 0.5 < \frac{6+3a+4b}{12+4a+6b} \Leftrightarrow a+b > 0 \Leftrightarrow b > -a$$

Given that  $-2\sqrt{a} \leq b \leq -a$ , the condition that  $b > -a$  is a contradiction. This implies that the above condition is false for all  $a$  and  $b$  proving that for  $\Delta_k = 0.5$ ,  $P_{PR}^{0.5}$  is larger than  $P_{CF}^{0.5}$ . ■

### A.1.2 Case 2: $\Delta_k > 0.5$

Rewriting the numerator and denominator in the function  $P_{CF}$  as polynomials in  $a$  and  $b$ , the coefficients of the DK function, we get the below simplified expression.

$$P_{CF} = \frac{m_1 + m_2b + m_3a}{m_4 + m_5b + m_2a}$$

Where  $m_1, m_2, m_3, m_4$ , and  $m_5$  are functions of the task difficulty,  $\Delta_k$ :

$$m_1(\Delta_k) = 2(1 - 2\Delta_k)^2(\Delta_k - 1)(2\Delta_k + 2\Delta_k \tanh^{-1} [1 - 2\Delta_k] - 1)$$

$$m_2(\Delta_k) = 2(\Delta_k - 1)(2\Delta_k - 1)(1 - 2\Delta_k + 4(\Delta_k - 1)\Delta_k \tanh^{-1} [1 - 2\Delta_k])$$

$$m_3(\Delta_k) = (\Delta_k - 1)(-2 + 3\Delta_k + 6\Delta_k^2 - 8\Delta_k^3 + 12(\Delta_k - 1)^2\Delta_k \tanh^{-1} [1 - 2\Delta_k])$$

$$m_4(\Delta_k) = 2(1 - 2\Delta_k)^4$$

$$m_5(\Delta_k) = 2(2\Delta_k - 1)(1 - 3\Delta_k + 2\Delta_k^2)(2\Delta_k + 2\Delta_k \tanh^{-1} [1 - 2\Delta_k] - 1)$$

***Proof.***

$$P_{CF} > P_{PR}$$

$$\Leftrightarrow \frac{m_1 + m_2b + m_3a}{m_4 + m_5b + m_2a} > P_{PR}$$

$$\Leftrightarrow m_1 + m_2b + m_3a > m_4P_{PR} + m_5bP_{PR} + m_2aP_{PR}$$

$$\Leftrightarrow b(m_2 - m_5P_{PR}) > a(m_2P_{PR} - m_3)$$

$$\Leftrightarrow b > \left( \frac{m_2P_{PR} - m_3}{m_2 - m_5P_{PR}} \right) a = \xi a$$

Given the previous condition of  $b < -a$ , we conclude that  $P_{CF} > P_{PR} \Leftrightarrow \xi < -1$  which is true for values of the task difficulty,  $\Delta_k$ , that are strictly between 0 and 0.5. ■

## A.2 Theorem II - Plurality vs. Maximum Confidence

Given any DK plot, any task difficulty, and any value of the percentage  $\mathcal{P}$ , plurality voting will always perform better than maximum confidence voting. Put differently, the best possible performance of the maximum confidence approach

at any difficulty level is equal to that of plurality voting.

$$\boxed{\forall (\mathbf{a}, \mathbf{b}, \Delta_k, \mathcal{P}) \quad P_{PR} \geq P_{MC}}$$

**Proof.** Evaluations of the formulas for the success probability of both plurality and maximum confidence voting are given below.

$$P_{PR} = \int_0^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r = \frac{(\Delta_k - 1)(2\Delta_k + 2\Delta_k \tanh^{-1}[1 - 2\Delta_k] - 1)}{(1 - 2\Delta_k)^2}$$

$$P_{MC} = \int_0^{x_{\min}^{\mathcal{P}}} \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r + \int_{x_{\max}^{\mathcal{P}}}^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r$$

$$= \frac{(\Delta_k - 1)(x_{\min}^{\mathcal{P}}(2\Delta_k - 1) + \Delta_k \log [1 + x_{\min}^{\mathcal{P}}(\frac{1}{\Delta_k} - 2)])}{(1 - \Delta_k)^2} - \frac{(\Delta_k - 1)((x_{\max}^{\mathcal{P}} - 1)(2\Delta_k - 1) + \Delta_k \log [\frac{x_{\max}^{\mathcal{P}} + \Delta_k - 2x_{\max}^{\mathcal{P}}\Delta_k}{1 - \Delta_k}])}{(1 - \Delta_k)^2}$$

Note that for  $x_{\min}^{\mathcal{P}} = x_{\max}^{\mathcal{P}}$ , the two probabilities are the same for all values of  $\Delta_k$ . When the boundary points are equal, the selected percentage of the crowd is 100% which is equivalent to selecting the entire crowd. This is validated in the probability formulas which become equal when  $x_{\min}^{\mathcal{P}} = x_{\max}^{\mathcal{P}}$ .

The difference between the two probability functions is a product of two terms as shown below.

$$\begin{aligned} \Lambda &= P_{PR} - P_{MC} \\ &= \frac{1 - \Delta_k}{(1 - 2\Delta_k)^2} \left( (2\Delta_k - 1)(x_{\min}^{\mathcal{P}} - x_{\max}^{\mathcal{P}}) + \Delta_k \log \left[ \frac{x_{\min}^{\mathcal{P}} + \Delta_k - 2x_{\min}^{\mathcal{P}}\Delta_k}{x_{\max}^{\mathcal{P}} + \Delta_k - 2x_{\max}^{\mathcal{P}}\Delta_k} \right] \right) \\ &= \Omega\gamma \end{aligned}$$

The first term  $\Omega = \frac{1-\Delta_k}{(1-2\Delta_k)^2}$  is positive for all values of  $0 \leq \Delta_k \leq 1$ . Accordingly, the sign of  $\Lambda$  depends on the sign of  $\gamma$ .

$$P_{PR} > P_{MC}$$

$$\Leftrightarrow \Lambda > 0 \Leftrightarrow \gamma > 0$$

$$\Leftrightarrow (2\Delta_k - 1)(x_{\min}^{\mathcal{P}} - x_{\max}^{\mathcal{P}}) + \Delta_k \log \left[ \frac{x_{\min}^{\mathcal{P}} + \Delta_k - 2x_{\min}^{\mathcal{P}}\Delta_k}{x_{\max}^{\mathcal{P}} + \Delta_k - 2x_{\max}^{\mathcal{P}}\Delta_k} \right] > 0$$

$$\Leftrightarrow \log [x_{\min}^{\mathcal{P}}(1 - 2\Delta_k) + \Delta_k] - \log [x_{\max}^{\mathcal{P}}(1 - 2\Delta_k) + \Delta_k] > \left( \frac{2\Delta_k - 1}{\Delta_k} \right) (x_{\max}^{\mathcal{P}} - x_{\min}^{\mathcal{P}})$$

$$\Leftrightarrow \log [x_{\min}^{\mathcal{P}}(1 - 2\Delta_k) + \Delta_k] + \left( \frac{2\Delta_k - 1}{\Delta_k} \right) (x_{\min}^{\mathcal{P}}) > \log [x_{\max}^{\mathcal{P}}(1 - 2\Delta_k) + \Delta_k] + \left( \frac{2\Delta_k - 1}{\Delta_k} \right) (x_{\max}^{\mathcal{P}})$$

$$\Leftrightarrow \mathcal{J}(x_{\min}^{\mathcal{P}}) > \mathcal{J}(x_{\max}^{\mathcal{P}})$$

Given that  $x_{\min}^{\mathcal{P}} < x_{\max}^{\mathcal{P}}$ , this means that for  $\Lambda$  to be positive, the function  $\mathcal{J}(x)$  must be decreasing in  $x$ . The first derivative of  $\mathcal{J}(x)$  is shown below.

$$\mathcal{J}'(x) = \frac{1 - 2\Delta_k}{\Delta_k + (1 - 2\Delta_k)x} - \frac{1 - 2\Delta_k}{\Delta_k}$$

Given:  $0 \leq \Delta_k \leq 1$

$0 \leq \Delta_k \leq 1 \Rightarrow \mathcal{J}'(x) < 0 \Rightarrow \mathcal{J}(x)$  is strictly decreasing with  $x$

$\Rightarrow \gamma > 0 \Rightarrow \Lambda > 0 \Rightarrow \forall \Delta_k, P_{PR} > P_{MC}$ . ■

### A.3 Theorem III - Maximum Confidence vs. $\mathcal{P}$

The probability function based on maximum confidence voting is increasing in  $\mathcal{P}$ .

$$\boxed{P_{MC} \text{ is increasing in } \mathcal{P}}$$

*Proof.*

Evaluation of the formula for the success probability of maximum confidence voting is given below.

$$P_{MC} = \int_0^{x_{\min}^{\mathcal{P}}} \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r + \int_{x_{\max}^{\mathcal{P}}}^1 \frac{\theta_r(1 - \Delta_k)}{\theta_r(1 - \Delta_k) + \Delta_k(1 - \theta_r)} d\theta_r$$

$$= \frac{(\Delta_k - 1) \left( x_{\min}^{\mathcal{P}} (2\Delta_k - 1) + \Delta_k \log \left[ 1 + x_{\min}^{\mathcal{P}} \left( \frac{1}{\Delta_k} - 2 \right) \right] \right)}{(1 - \Delta_k)^2} - \frac{(\Delta_k - 1) \left( (x_{\max}^{\mathcal{P}} - 1) (2\Delta_k - 1) + \Delta_k \log \left[ \frac{x_{\max}^{\mathcal{P}} + \Delta_k - 2x_{\max}^{\mathcal{P}} \Delta_k}{1 - \Delta_k} \right] \right)}{(1 - \Delta_k)^2}$$

$$x_{\min}^{\mathcal{P}} = \frac{-b - \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a} \quad \text{and} \quad x_{\max}^{\mathcal{P}} = \frac{-b + \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a}$$

As  $\mathcal{P}$  increases, the lower integration bound  $x_{\min}^{\mathcal{P}}$  increases and the upper integration bound  $x_{\max}^{\mathcal{P}}$  decreases. This causes the integration limits to tend closer to each other until eventually  $\mathcal{P}$  reaches 100% and the two boundary points join resulting in  $P_{MC} = P_{PR}$ . ■

### A.4 Theorem IV - Confidence-Weighting vs. Maximum Confidence

At  $\Delta_k = 0$ , confidence-weighted voting performs better than maximum confidence voting for all values of  $\mathcal{P} < \frac{25b^2}{a}$ . For  $\mathcal{P} = \frac{25b^2}{a}$ ,  $x_{\min}^{\mathcal{P}} = x_{\max}^{\mathcal{P}}$  and

maximum confidence voting becomes equivalent to plurality voting resulting in  $P_{MC}^0 = P_{PR}^0 = P_{CF}^0 = 1$ .

At  $\Delta_k = 0.5$ , confidence-weighted voting performs better than maximum confidence voting for all values of  $\mathcal{P} \leq \frac{25b^2}{a} - 10$ . As  $\mathcal{P}$  decreases in value,  $P_{MC}$  decreases based on Theorem III above whereas  $P_{CF}$  stays the same. This shifts the point at which  $P_{MC}$  out performs  $P_{CF}$  further to the right.

At  $\Delta_k = 1$ , probabilities of both confidence-weighted and maximum confidence voting converge to 0.

Evaluations of the formulas for the success probability of both confidence-weighted and maximum confidence voting are given below.

$$\begin{aligned}
P_{CF} &= \frac{\int_0^1 \left( [a(x_r^k)^2 + b(x_r^k) + 1] \cdot [x_r^k] \right) d\theta_r}{\int_0^1 \left( a(x_r^k)^2 + b(x_r^k) + 1 \right) d\theta_r} \\
&= \frac{\int_0^1 \left( \left[ a \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right)^2 + b \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right) + 1 \right] \cdot \left[ \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right] \right) d\theta_r}{\int_0^1 \left( a \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right)^2 + b \left( \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} \right) + 1 \right) d\theta_r} \\
&= \frac{((\Delta_k-1)(2\Delta_k-1)^3(-2\Delta_k-1)(-2(a+b+1)+\Delta_k(8-a+4b)+4\Delta_k^2(a-2))+4\Delta_k(3a(\Delta_k-1)^2+(2\Delta_k-1)(-1+2b(\Delta_k-1)+2\Delta_k)) \tanh^{-1}[1-2\Delta_k])}{(2(1-2\Delta_k)^4((2\Delta_k-1)(1+a+b-\Delta_k(4+a+3b)+2\Delta_k^2(2+b))+2\Delta_k(\Delta_k-1)(-2a-b+2\Delta_k(a+b)) \tanh^{-1}[1-2\Delta_k])} \\
P_{MC} &= \int_0^{x_{\min}^{\mathcal{P}}} \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} d\theta_r + \int_{x_{\max}^{\mathcal{P}}}^1 \frac{\theta_r(1-\Delta_k)}{\theta_r(1-\Delta_k) + \Delta_k(1-\theta_r)} d\theta_r \\
&= \frac{(\Delta_k-1) \left( x_{\min}^{\mathcal{P}}(2\Delta_k-1) + \Delta_k \log \left[ 1 + x_{\min}^{\mathcal{P}} \left( \frac{1}{\Delta_k} - 2 \right) \right] \right)}{(1-\Delta_k)^2} - \frac{(\Delta_k-1) \left( (x_{\max}^{\mathcal{P}}-1)(2\Delta_k-1) + \Delta_k \log \left[ \frac{x_{\max}^{\mathcal{P}} + \Delta_k - 2x_{\max}^{\mathcal{P}}\Delta_k}{1-\Delta_k} \right] \right)}{(1-\Delta_k)^2} \\
&\quad x_{\min}^{\mathcal{P}} = \frac{-b - \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a} \quad \text{and} \quad x_{\max}^{\mathcal{P}} = \frac{-b + \sqrt{b^2 - \frac{a\mathcal{P}}{25}}}{2a}
\end{aligned}$$

#### A.4.1 Case 1: $\Delta_k = 0$

The limit of  $P_{CF}$  as  $\Delta_k$  tends to 0 is  $P_{CF}^0 = 1$

The limit of  $P_{MC}$  as  $\Delta_k$  tends to 0 is  $P_{MC}^0 = 1 - (x_{\max}^{\mathcal{P}} - x_{\min}^{\mathcal{P}})$

*Proof by contradiction.*

$$P_{MC}^0 > P_{CF}^0 \Leftrightarrow 1 - (x_{\max}^{\mathcal{P}} - x_{\min}^{\mathcal{P}}) > 1 \Leftrightarrow x_{\max}^{\mathcal{P}} - x_{\min}^{\mathcal{P}} < 0 \Leftrightarrow x_{\max}^{\mathcal{P}} < x_{\min}^{\mathcal{P}}$$

Given that  $x_{\max}^{\mathcal{P}} \geq x_{\min}^{\mathcal{P}}$ , the above condition is false. This proves that at

$\Delta_k = 0$ ,  $P_{CF}^0 \geq P_{MC}^0$ . The two probabilities are equal when  $x_{\max}^{\mathcal{P}} = x_{\min}^{\mathcal{P}}$  in

which case  $P_{MC}$  becomes the same as  $P_{PR}$ . ■

#### A.4.2 Case 2: $\Delta_k = 0.5$

The limit of  $P_{CF}$  as  $\Delta_k$  tends to 0.5 is  $P_{CF}^{0.5} = \frac{6+3a+4b}{12+4a+6b}$  where  $\max(P_{CF}^{0.5}) = 0.5$ .

The limit of  $P_{MC}$  as  $\Delta_k$  tends to 0.5 is  $P_{MC}^{0.5} = 0.5 + \frac{b\sqrt{25b^2 - a\mathcal{P}}}{10a^2}$  where  $\max(P_{MC}^{0.5}) =$

0.5.

*Proof by contradiction.*

$$P_{CF}^{0.5} < P_{MC}^{0.5}$$

$$\Leftrightarrow \frac{6 + 3a + 4b}{12 + 4a + 6b} < 0.5 + \frac{b\sqrt{25b^2 - a\mathcal{P}}}{10a^2}$$

$$\Leftrightarrow b\sqrt{25b^2 - a\mathcal{P}} > \frac{10a^3 + 10ba^2}{12 + 4a + 6b}$$

$$\Leftrightarrow \sqrt{25b^2 - a\mathcal{P}} < \frac{10a^3 + 10ba^2}{12b + 4ab + 6b^2}$$

$$\Leftrightarrow \mathcal{P} > \frac{25b^2}{a} - \frac{1}{a} \left( \frac{10a^3 + 10ba^2}{12b + 4ab + 6b^2} \right)^2 = \frac{25b^2}{a} - \mathcal{G}(a, b)$$

Given  $0 \leq \alpha \leq 4$  and  $-2\sqrt{\alpha} \leq \beta \leq -\alpha$ , the function  $\mathcal{G}(\alpha, \beta)$  is never greater than 10 for all values of  $\alpha$  and  $\beta$ . This proves that for  $\mathcal{P} \leq \frac{25\beta^2}{\alpha} - 10$ ,  $P_{MC}^{0.5}$  is never greater than  $P_{CF}^{0.5}$ . ■

### A.4.3 Case 3: $\Delta_k = 1$

The limit of  $P_{CF}$  as  $\Delta_k$  tends to 1 is  $P_{CF}^1 = 0$

The limit of  $P_{MC}$  as  $\Delta_k$  tends to 1 is  $P_{MC}^1 = 0$

For all values of  $\mathcal{P}$ , the two probabilities converge to 0 at maximum task difficulty. ■



# Appendix B

## Device and App Usage Stats

A byproduct of our experiment is related to app and device usage. We were interested in collecting statistics related to how users use their devices in general and how different apps and app categories are used at certain times of the day and certain days of the week. However, this was not an easy problem considering how usage statistics differ considerably between users on a primary device (which in most cases is a phone) and users on a secondary device (which in most cases is a tablet but can also sometimes be a phone). In our experiment, around 27% of the users had tablets and the remaining 73% were using their mobile phones. To this end, results in the following subsections are separated in terms of primary and secondary devices as well as results that were computed for all devices in general assuming that our distribution of devices is a fair representation of devices in the real world.

## B.1 Users and Device Type Detection

In the case of phones versus tablets, we already have the ground truth. We know which users were on a mobile phone and which were on tablets. Still, we approached this problem of device type classification with the assumption that we do not have the ground truth for the purpose of gaining insight into the different attributes.

As a first step, we created a table of all users. For each user, we inserted every possible attribute that was collected throughout the experiment process. Examples of attributes are the average usage time of apps in category GAMES in hour 16 of the day, or the average number of times the screen is turned on in day 6 of the week, or the average memory consumption of the apps in category COMMUNICATION, or the number of apps in category TOOLS, etc. Given the variety of both app categories and collected features, we ended up gathering over 200 distinct attributes per user.

We filtered this table by first removing all users whose average ON frequency (average number of times they turn on their device screen per day) was zero. We also removed all users who were active for less than 1 month. We were left with 21 users in total.

We used the Weka software to analyze these attributes. Weka is a suite of machine learning software written in Java. It contains algorithms for data analysis and modelling. We started out by including all initial attributes and applying random forest classification with 10-fold cross validation. We got a device type classification accuracy of 80% which is acceptable. However, we were more interested in the insight regarding the most frequently used attributes.

We studied different attribute selection techniques that are found within

Weka. There is correlation-based attribute selection which calculates the Pearson correlation between each attribute and the output variable and selects only those attributes that have a moderate-to-high positive or negative correlation. There is attribute selection based on entropy which selects attributes that contribute more information gain. Another approach is a learner-based attribute selection which evaluates classification performance based on different subsets of attributes and selects the subset that results in the best performance. Applying each of these methods gave us a different set of selected attributes with some intersections among the sets. We tried several combinations of the resulting attributes from the attribute selection phase before attempting to classify our data using the J48 decision tree. We noticed that the accuracy was always 75% to 80%.

The second approach used was filtering out the initial set of 200+ attributes into around 20 attributes which we believe conveyed the most information related to the device type. Examples of the filtered attributes are avg\_ON\_16 (the average number of times a user turns on his device at 4 pm everyday), UT\_OF (the average usage time per opening frequency), UT\_ON (the average usage time per device screen turn on), etc. After reducing the attribute set, we ran random forest again while printing out the resulting decision trees. We then got the number of times each attribute occurred in the set of decision trees.

We were interested in the attributes that occurred in 10+ decision trees in the random forest output. We selected these attributes and viewed their values. We noticed that if we can somehow combine only two attributes, we can reach very good classification results. These values were avg\_ON\_16 and avg\_RSS (the average memory consumption of apps on the device). The attribute avg\_ON\_16 has relatively high values on phones and low values on tablets. This is the average number of times that a user switches his device screen on at 4 pm every day. One

possible explanation as to why this attribute has high values on phones is that this is most likely an hour where users are leaving work (commuting home) which means that they are more likely to be carrying a phone versus a tablet. On the other hand, the attribute `avg_RSS` has relatively high values on tablets and low values on phones. This is because users tend to install more games and other memory-consuming applications on their tablets and they tend to use them more often considering the longer battery life of tablets versus phones.

After gaining this insight regarding these two features, we decided to generate a new attribute that is a combination of the above two attributes: `new_ATT = (1 - avg_ON_16) + avg_RSS`. We computed the values of this attribute for all users and updated the user table. Using this attribute alone and using a Naïve Bayes classifier, we managed to reach 100% device detection accuracy.

Of course, this classification result is not important to us but rather the insight that we gained from it. We learned that from over 200 attributes, a combination of only two of them was enough to separate devices into different types. This information can be used later on to aid in classifying new devices that install CrowdApp v2.0.

Another attribute that clearly separates phone users from tablet users that we decided not to include in our classification is related to the average time a user takes to reply to questions sent by CrowdApp v2.0. The reason we did not include it is because we wanted to study how other features that are not related to the rating process can be used to help classify device types.

It is worth noting that after splitting the data into data from phones and data from tablets, we noticed a very strong negative correlation between the average reply rate of users on tablets with their average reported confidence values which means that least confident users take their time before answering questions. We

also noticed a very strong positive correlation between the average reply rate and the number of apps in the SOCIAL category. This might be due to users constantly checking their device for notifications from social media platforms while answering any CrowdApp questions at the same time.

## **B.2 Statistics Related to Apps and App Categories**

We begin this section with an analysis of app distribution on user devices. Usage statistics of some of the most popular apps are studied. In addition to how users use their apps, we are interested in app categories in general. To this end, we categorized all the apps in our experiment into a set of categories as proposed on the Google PlayStore. In analyzing the usage statistics of app categories, we pay attention to the difference between mobile phones and tablets and how usage patterns and app downloads differ between the two types. We present our analyses accordingly.

### **Apps Usage Statistics**

Out of the 486 apps that the users in our experiment have installed on their devices, 365 apps are only installed on one device followed by 65 apps that are only installed on two devices. A very small number of apps can be found on the majority of user devices (7+ devices) as is shown in the histogram shown in Figure B.1. This agrees with the result in Table 3.4 which shows how apps that have been downloaded over 50,000 times constitute a minor percentage of the total number of apps on the market.

When analyzing the usage statistics of apps separately, we were only interested

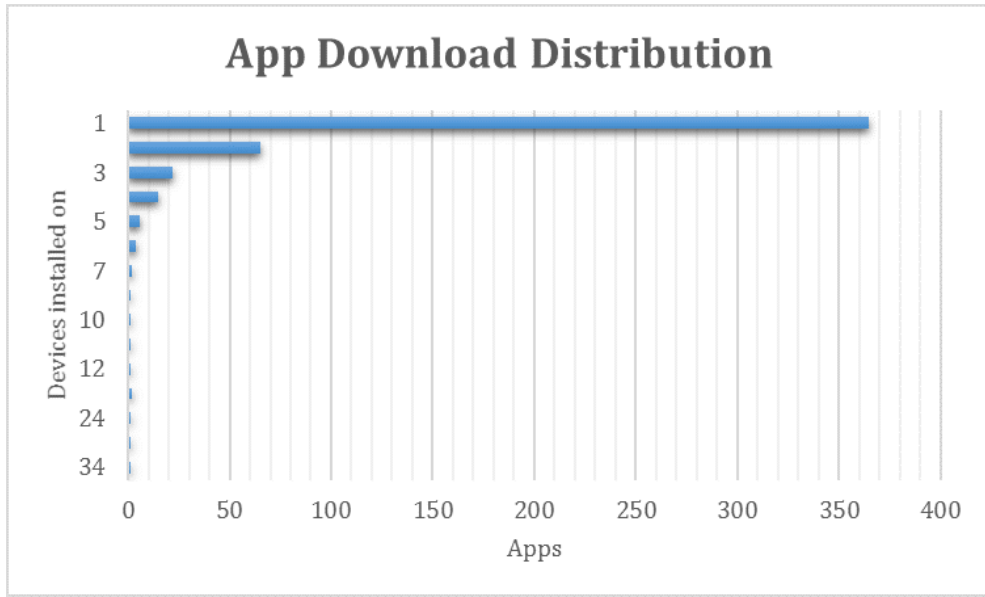


Figure B.1: A histogram of app download distribution

in the apps that are installed on at least 30% of the user devices. These apps are WhatsApp which was installed on 24 devices, Messenger and Facebook which were both installed on 16 devices, Instagram which was installed on 12 devices, a dialer app, Truecaller, which was installed on 11 devices, and a music-streaming app, Anghami, which was installed on 10 devices. We present results for selected apps in Figure B.2.

In terms of usage time, we notice that Anghami is used more during the weekends and less often in the middle of the week, especially on the first day after the weekend where it is barely used. Facebook is used least frequently on Fridays and Saturdays. We believe this is related to the fact that most people go out on these days and therefore spend less time on social media. Both Facebook and Instagram are mostly used on Wednesdays. This result agrees with the discussion in [155] where the author of the article advises his readers to post on these social media platforms on Wednesdays due to the significant activity on these days. Another observation is that WhatsApp is used more in the beginning of the

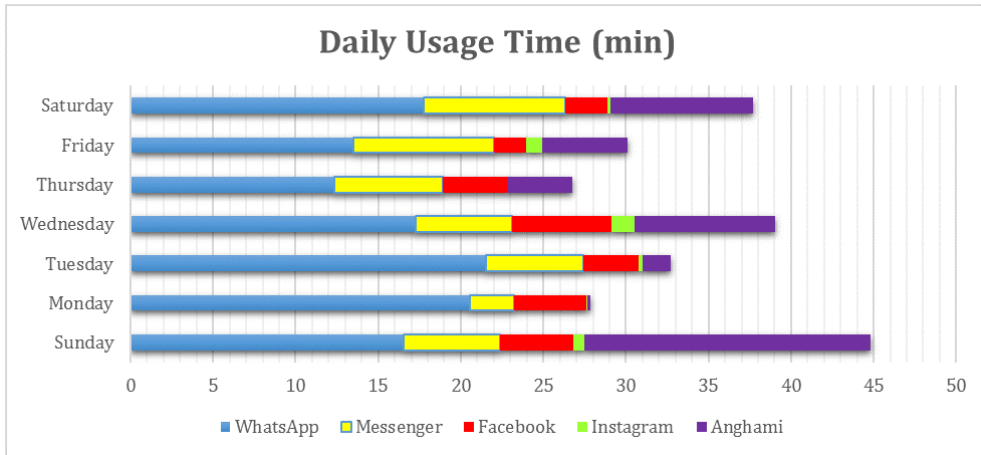


Figure B.2: Daily usage time of popular apps

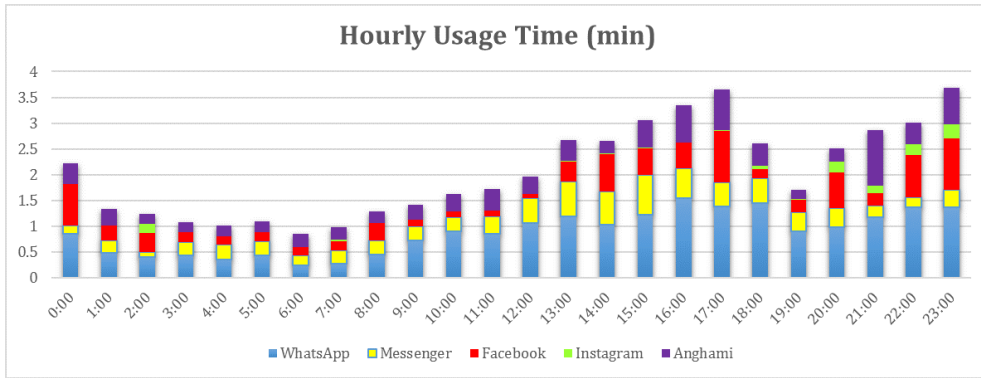


Figure B.3: Hourly usage time of popular apps

week versus Messenger which is used more towards the end of the week. Even though both applications fall under the COMMUNICATION category, however Messenger is used more frequently to send images, videos, and share Facebook links in general, which would probably explain its higher usage time closer to the weekend rather than at the start of a week.

When looking at Figure B.3, the first observation is that the overall usage time of these apps starts decreasing gradually after midnight and then increases again after 6 am. This result is expected of course, since most users are sleeping between these times and so they will be using their devices less.

Another observation is that the music-streaming app, Anghami, is used more at 4 and 5 in the afternoon which we suspect to be the times when many users are listening to music in their cars while commuting back home from work. In general, we notice that most apps are used more often at these times (between 4 and 5 pm) which is when most users get off work. Then at around 7 pm, there is a sudden drop in the overall usage time which could be when most users arrive home and are too busy to check their devices. However, the usage time gradually increases during the later hours of the night until midnight. Facebook is used mostly after working hours and before bedtime, whereas WhatsApp is used more constantly throughout.

In terms of opening frequency, the app with the highest value is WhatsApp. This result is expected since it is a communication app and these apps are usually used in short bursts rather than for longer periods of time. Even though Messenger is a communication app as well, however, given that it is mostly used to share Facebook posts in private, it is opened less frequently than both WhatsApp and Facebook as shown in Figure B.4.

We can also consider the daily opening frequency after normalizing it per app. The opening frequencies of every app over the seven days of the week were normalized to values between 0 and 1 as shown in Figure B.5. We notice that Messenger is opened most frequently on the weekend whereas WhatsApp is opened most frequently in the beginning of the week on Monday. Its frequency decreases gradually throughout the week. Anghami is also opened more frequently on the weekends. As for Facebook and Instagram, they are opened mostly in the middle of the week which agrees with [155]. In fact, if posting pictures on Instagram and Facebook on a Wednesday is known to result in more social interaction, then most users will post on these days and after posting will frequently open these



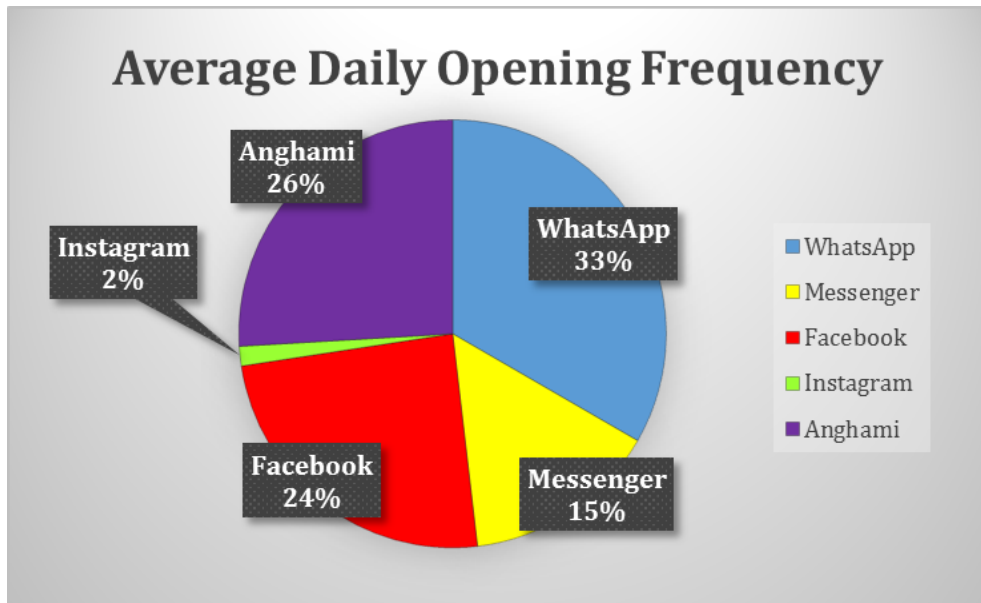


Figure B.4: Average daily opening frequency of popular apps

apps to check for any updates or notifications. This is one possible explanation for the popularity of these two apps on specific days of the week.

As for the other features, we present the results normalized per app as shown in Figure B.6. We notice that both Truecaller and Messenger have the largest share in terms of connection age. Considering that both apps can be used to make phone calls, this result is expected. WhatsApp and Instagram consume the

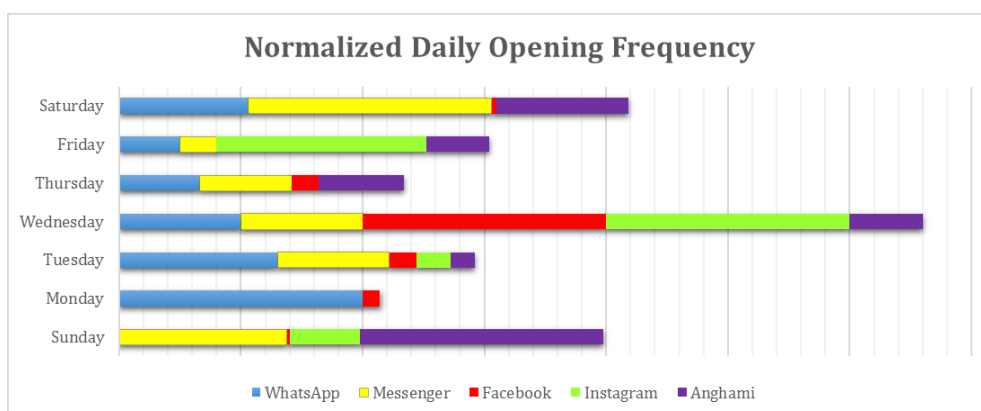


Figure B.5: Normalized daily opening frequencies of popular apps

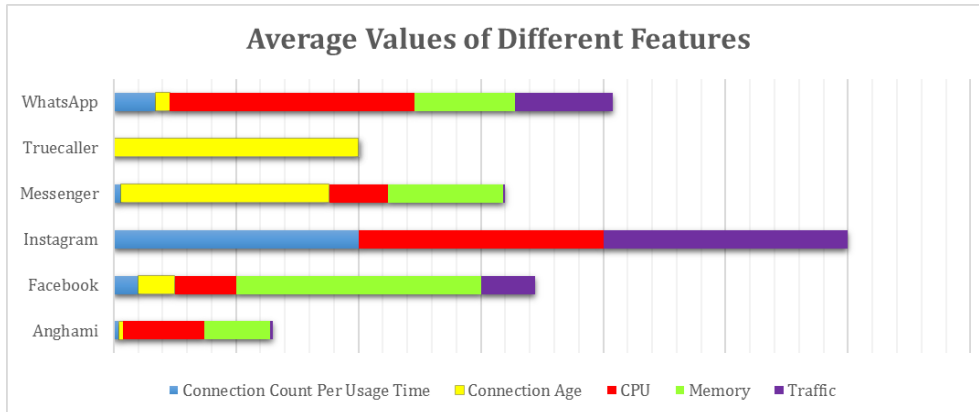


Figure B.6: Average values different features of popular apps

most CPU time, followed by Anghami and Facebook, which also consumes most memory. And finally, in terms of traffic, Instagram has the highest share considering that it is a social media platform specifically for sharing images and videos which means that opening the app is enough to download relatively significant data.

Some other results not shown in the plots are also worth noting. The Viber app was opened only once and used for one minute by one of the users in our crowd. The resulting total number of connections opened by this app was 928 ranging across the entire period of the experiment. Another application is the movie-streaming app, PopcornTime. The average number of opened connections per one minute of usage time was 5542. Clearly, both these apps are opening connections to IP addresses while in the background without users' knowledge.

### B.2.1 Categories Usage Statistics

We display three major results in this section which are the usage time, opening frequency, and results related to other app characteristics such as CPU, RAM, etc. In each case, the results are three-fold: Given that the percentage of tablets to the

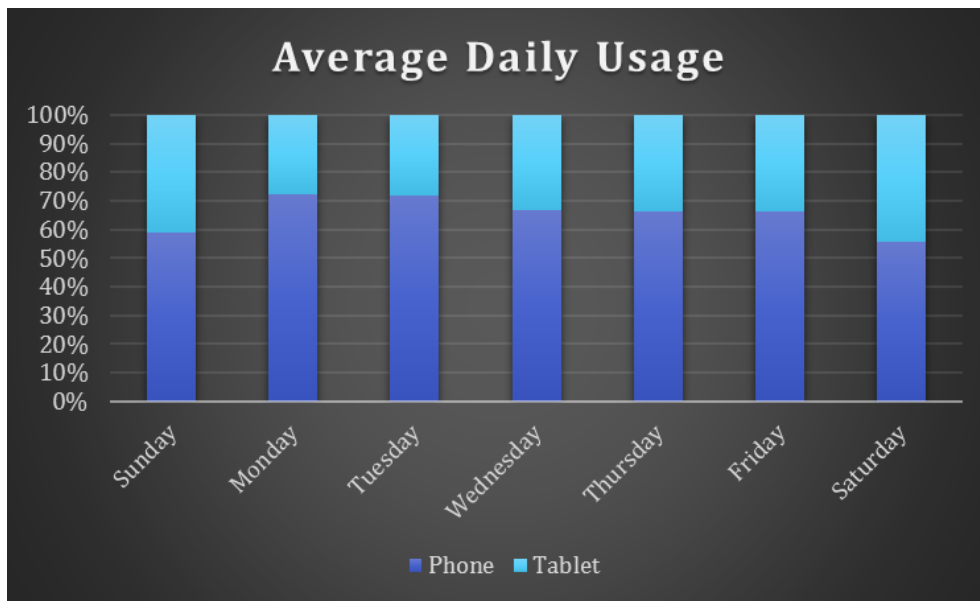


Figure B.7: Average daily usage time of phones versus tablets

total number of devices is a close estimate to that in a wider population, we first present in what follows statistical analyses related to app categories considering the entire set of devices in the experiment (both primary and secondary devices). We then present separate analyses related to phones and tablets.

Figure B.7 shows the distribution of the daily usage between phones and tablets. Phones are used more often than tablets since they are always carried with the user. However, on the weekends, the usage of phones and tablets becomes somewhat close since users might be home more often allowing them to be more active on their tablets.

### Usage Time

We begin with the daily usage time of a selection of app categories. We show in Figure B.8 the daily usage time of categories when combining both phones and tablets, in Figure B.9, the daily usage time is plotted considering phones only, and in Figure B.10, it is plotted considering tablets only.

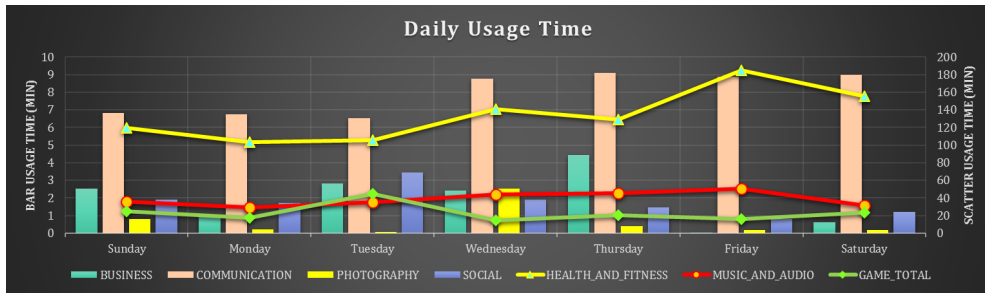


Figure B.8: Daily usage time of selected categories on all devices

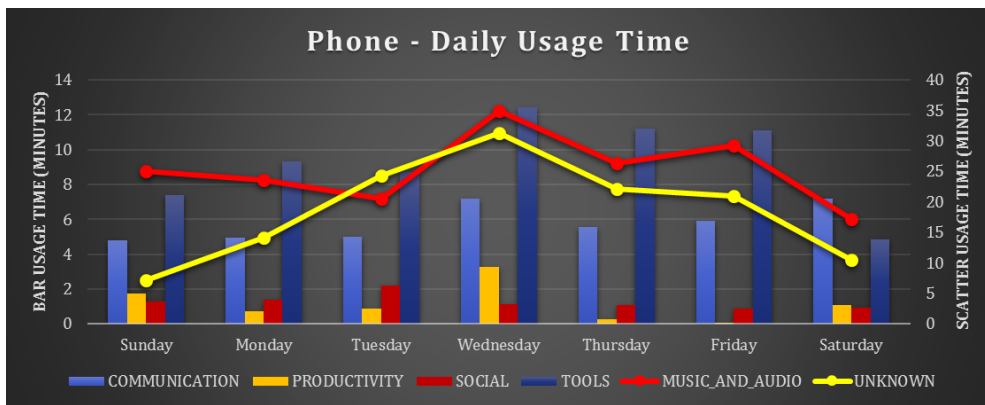


Figure B.9: Daily usage time of selected categories in phones

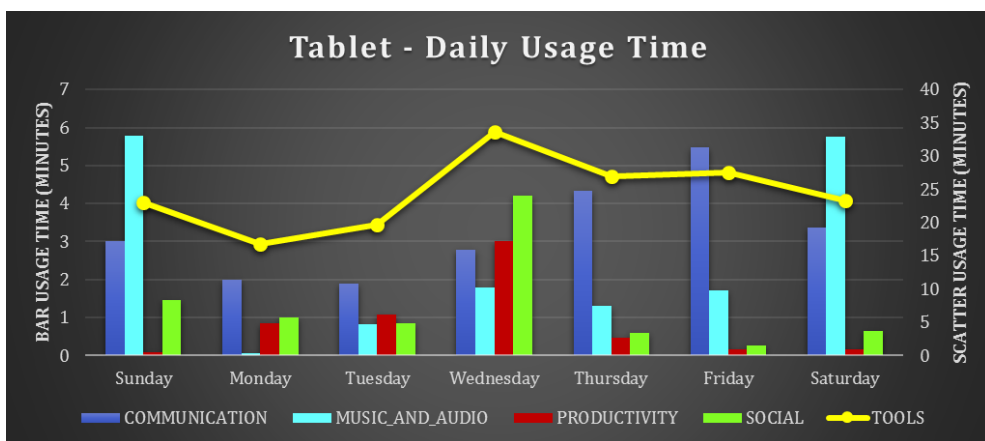


Figure B.10: Daily usage time of selected categories on tablets

Our first comment in Figure B.8 is related to the difference in usage time between all app categories and the HEALTH\_AND\_FITNESS, MUSIC\_AND\_AUDIO, and GAMES categories. One of the apps in the HEALTH\_AND\_FITNESS category is the pedometer which is used frequently and on a daily basis. GAMES and MUSIC\_AND\_AUDIO are also used for longer periods of time when compared to other categories such as COMMUNICATION. Apps in this category are usually opened a lot but used for shorter periods of time thus resulting in a lower overall usage time value. We also notice that apps in the COMMUNICATION category are used more often towards the end of the week than at the beginning.

Apps in the BUSINESS category are rarely used on the weekends (mainly on Fridays and Saturdays). As for apps in the SOCIAL category, they are most frequently used in the middle of the week and less frequently used on the weekends. This agrees with the discussion in [155].

PHOTOGRAPHY apps are mostly used in the middle of the week as well. This result is expected considering the correlation between PHOTOGRAPHY apps and SOCIAL apps. They are also used frequently on Sundays which could be explained given the amount of free time available to a device user on a Sunday.

Going now into the differences between phone and tablet usage, one major difference which can be observed in Figures B.9 and B.10 is related to the usage of apps in the MUSIC\_AND\_AUDIO category. These apps are used much more on phones in an almost consistent fashion (maybe more during the middle of the week). They are used less often on tablets and it can also be seen that if used on tablets, it is usually during the weekends. This is reasonable considering that phones are always with users even when they are commuting to and from work and listening to music in their cars. Tablets on the other hand are not always with users. They are sometimes kept at home and used more often on the weekends.

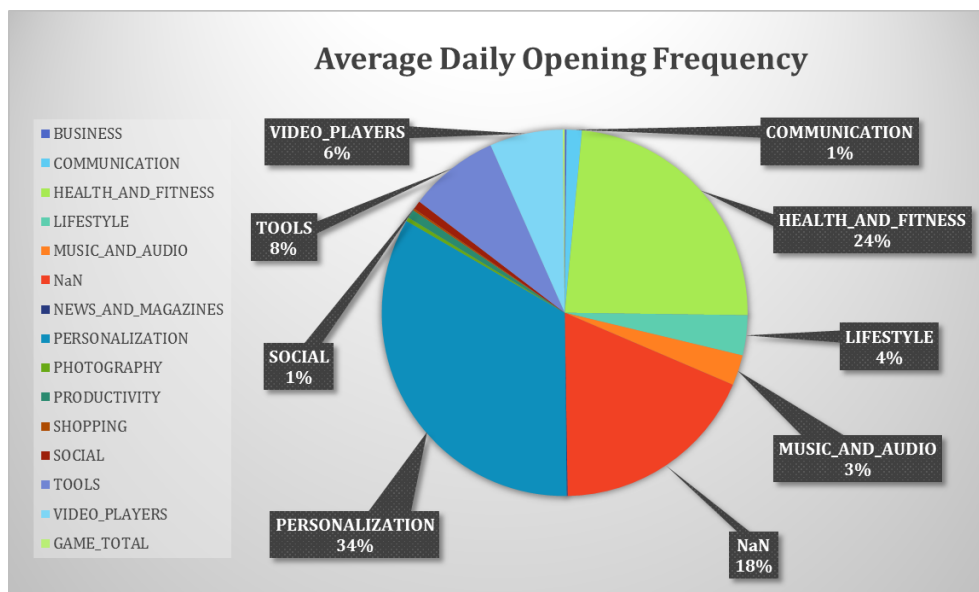


Figure B.11: Average daily opening frequency on all devices

Another observation is that apps in the TOOLS category (translate, ruler, file manager, QR reader, text converter, etc.) are used on phones more often during workdays than on the weekend, which is reasonable. However, these apps are used more often on tablets than they are on phones. This might be due to the larger memory size and the longer battery life on tablets which give users the luxury of downloading and using more of these apps more frequently.

### Opening Frequency

In terms of the average daily opening frequency, we begin by considering all devices in our experiment as shown in Figure B.11. We notice that apps in the PERSONALIZATION category have the highest share. One of the apps in this category is the Nova Launcher; a highly customizable home screen for Android devices that has over fifty million downloads on Google Play. Given that this app is for customizing the screen on a user device every time it launches, this very high opening frequency is expected.

Next, there is the HEALTH\_AND\_FITNESS category with its most common app which is the Pedometer. Users who have this app tend to open it frequently to check on the status of their step count. It is opened much more frequently than apps in the COMMUNICATION category for example.

Apps from unknown sources are also opened frequently. This result is expected considering that this category (NaN) includes apps from all categories such as music, games, personalization, social, etc.

Figure B.12 shows the distribution of the opening frequency of apps in different categories between phones and tablets. The general observation is that apps are opened more frequently on phones than on tablets. This is expected considering phones are always close to the user whereas it is not always practical to carry a tablet around. However, there are categories where apps are opened more frequently on tablets than on phones. One such example is SOCIAL apps. A possible explanation could be that these apps usually consume a lot of CPU, memory, and bandwidth. Considering the higher battery life and larger memory on tablets, users are more encouraged to check their SOCIAL apps on them. In addition, phones usually have a limited data plan unlike tablets which are in most cases connected to a WiFi network.

We also show the daily opening frequency of different categories normalized per category in Figure B.13. These results are combined for both phones and tablets. Apps in COMMUNICATION are opened most frequently on Thursdays and least frequently on Sundays. Apps in EDUCATION are opened mostly on Wednesdays and less frequently on other days of the week. An interesting result is for the apps in the HEALTH\_AND\_FITNESS category. They are used more often before the weekend and barely opened on Sundays. On the other hand, apps in MUSIC\_AND\_AUDIO are mostly opened on the weekends (Saturdays and Sun-

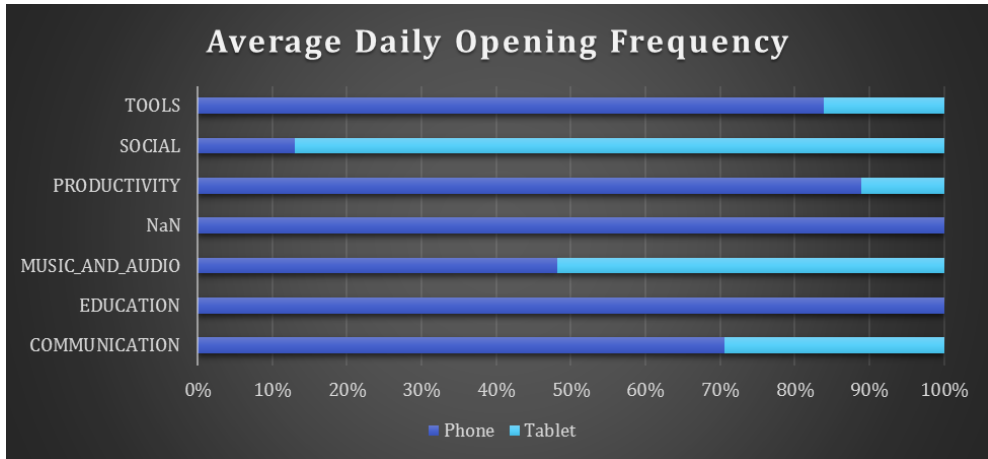


Figure B.12: Normalized average opening frequencies of selected categories on phones versus tablets

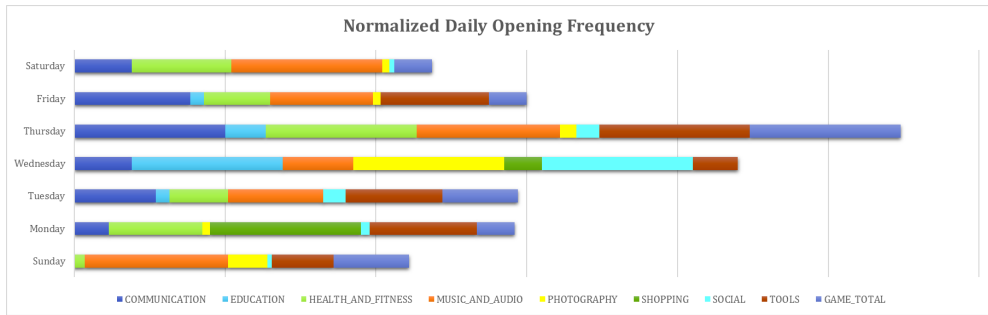


Figure B.13: Normalized daily opening frequencies of selected categories on all devices

days). PHOTOGRAPHY and SOCIAL apps are mostly opened on Wednesdays and SHOPPING apps on Mondays.

### Other Features

For the rest of the features, we present the results normalized per category. Figure B.14 shows the results combined for both phones and tablets. We notice that the highest connection count per usage time and the longest connections are generated by apps from an unknown source which is an expected result. One such app is PopcornTime which generates over 5,000 different connections per



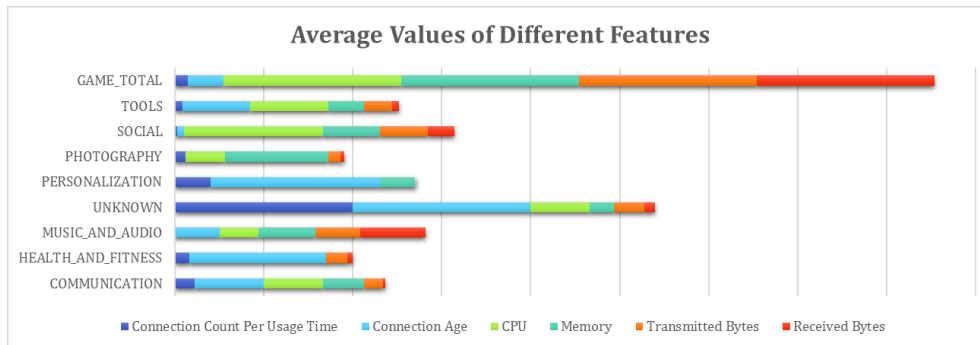


Figure B.14: Average values of different features for selected categories on all devices

minute.

Apps in the GAMES category have the highest CPU and memory consumption as well as the highest generated traffic both uploaded and downloaded. SOCIAL apps consume significant CPU time and HEALTH\_AND\_FITNESS apps open very long connections which is expected considering constantly having to sync data with servers.

When analyzing phones and tablets separately, the results are different. On phones, we notice that most categories have similar CPU consumption with SOCIAL taking a slightly bigger share than the rest of the categories as shown in Figure B.15. The highest CPU consumption on tablets, as shown in Figure B.16, is for apps of unknown sources which seem to be more common on tablets than on phones.

We notice a general trend in these results. Phones have an almost consistent consumption throughout different categories compared to tablets. This is due to the design of apps. Apps for phones are carefully designed to consume very little CPU, memory, and bandwidth. This requirement is relaxed when considering tablets due to their higher specs. This can also be due to a user's usage preferences especially considering a possible data plan. For example, apps in

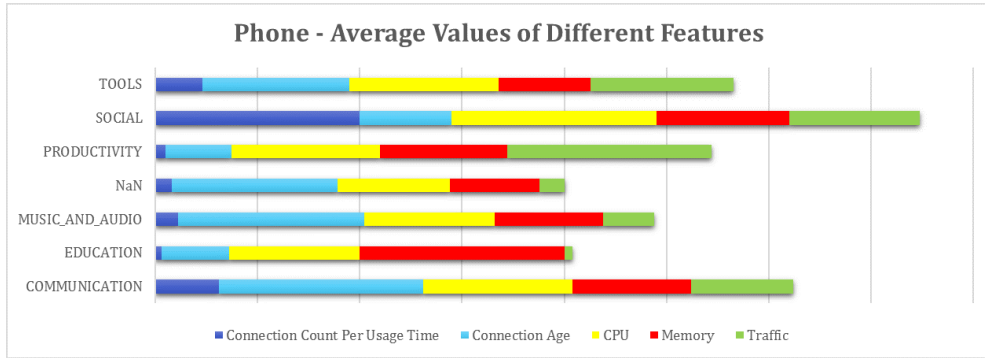


Figure B.15: Average values of different features for selected categories on phones

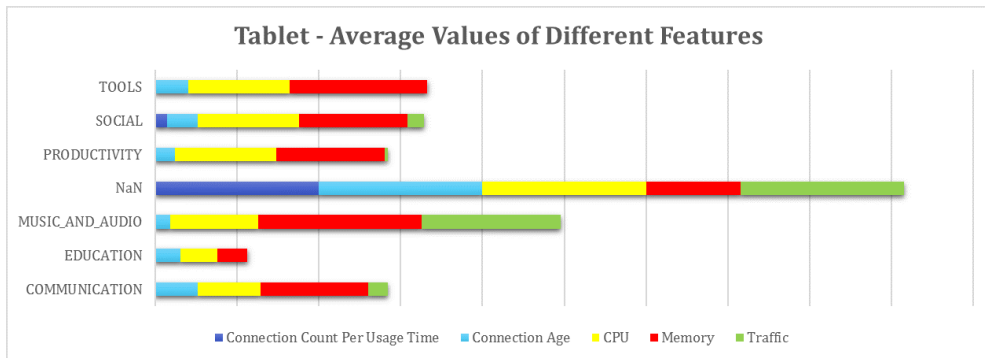


Figure B.16: Average values of different features for selected categories on tablets

MUSIC\_AND\_AUDIO consume more traffic on tablets than on phones which is expected since tablets are connected to WiFi more often than phones, and therefore, a user will be more encouraged to stream music on them than when on his phone.

### **B.3 Discussion and Conclusions Related to Device and App Usage**

App and device usage results can be helpful to detect proper timespans for utility score computations later on. We showed how different apps and app categories are used throughout hours of the day and days of the week. We also showed consumption trends of apps and categories on different device types as well as the insight given when attempting to classify devices into phones and tablets. Users tend to be more aggressive in terms of using memory and CPU consuming apps when on their tablets versus when on their phones. This comes as a result of generally better device specs and longer battery lives of tablets. On the other hand, a defining feature for phones was found to be related to the average number of times that a user switches his device on per hour. Tablets are not as easy to carry along at all times. During hours of the day when most users are assumed to be commuting back from work, the switching frequency on phones is much higher than that on tablets.

In addition to device defining features, another discovery was related to the usage of some of the most popular apps and app categories among our crowd. Games were shown to consume the highest CPU and memory. Messaging apps open long-term connections. Peer-to-peer apps such as PopcornTime open many connections. Social media apps are mostly used in the middle of the week versus

on the weekend. We also saw some correlation between the usage of Music apps on phones and the hours of the day when users are generally assumed to be off work and are commuting back home. General usage trends of different categories were highlighted throughout the Appendix.

# Bibliography

- [1] Bryan Wolfe, “The number of apps downloaded each day reaches 30 million.” <http://appadvice.com/appnn/2011/01/number-apps-downloaded-day-reaches-30-million>, 2011.
- [2] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A survey of mobile malware in the wild,” in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, (Chicago, Illinois, USA), pp. 3–14, 2011.
- [3] “Brain test.” Web Page, 2016. Accessed: 2017-02-10.
- [4] J. Howe, “The rise of crowdsourcing,” *Wired*, 2006.
- [5] M. Buhrmester, T. Kwang, and S. Gosling, “Amazon’s mechanical turk: A new source of inexpensive, yet high-quality, data?,” *Perspectives on Psychological Science*, vol. 6, pp. 3–5, Feb 2011.
- [6] K. McCurdy, “Crowdsourcing & istockphoto.”
- [7] A. K. Singh, “Innocentive for crowdsourcing,” *International Journal of Advanced Research in Computer Science & Technology*, vol. 2, no. 2, p. 303305, 2014.

- [8] K. R. Lakhani, D. A. Garvin, and E. Lonstein, “Topcoder (a): Developing software through crowdsourcing,” *Harvard Business School*, Jan 2010.
- [9] P. Belleflamme, T. Lambert, and A. Schwienbacher, “Crowdfunding: Tapping the right crowd,” *Journal of Business Venturing*, vol. 29, p. 585609, Jul 2013.
- [10] C. Darwin, *The Descent of Man*. John Murray, 1871.
- [11] P. Spikins, B. Wright, and D. Hodgson, “Are there alternative adaptive strategies to human pro-sociality? the role of collaborative morality in the emergence of personality variation and autistic traits,” *Time and Mind*, vol. 9, no. 4, pp. 289–313, 2016.
- [12] W. D. Hamilton, “Selfish and spiteful behaviour in an evolutionary model,” *Nature*, vol. 228, p. 12181220, Dec 1970.
- [13] A. Guazzini, D. Vilone, C. Donati, A. Nardi, and Z. Levnaji, “Modeling crowdsourcing as collective problem solving,” *Scientific Reports*, vol. 5, Nov 2015.
- [14] A. J. Quinn and B. B. Bederson, “Human computation: A survey and taxonomy of a growing field,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’11, (New York, NY, USA), pp. 1403–1412, ACM, 2011.
- [15] V. Conitzer and T. Sandholm, “Common voting rules as maximum likelihood estimators,” *CoRR*, vol. abs/1207.1368, 2012.
- [16] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, “Rank aggregation methods for the web,” in *Proceedings of the 10th International Conference*

- on World Wide Web*, WWW '01, (New York, NY, USA), pp. 613–622, ACM, 2001.
- [17] N. Quoc Viet Hung, N. T. Tam, L. N. Tran, and K. Aberer, “An evaluation of aggregation techniques in crowdsourcing,” in *Web Information Systems Engineering – WISE 2013* (X. Lin, Y. Manolopoulos, D. Srivastava, and G. Huang, eds.), (Berlin, Heidelberg), pp. 1–15, Springer Berlin Heidelberg, 2013.
- [18] G. Barbier, R. Zafarani, H. Gao, G. Fung, and H. Liu, “Maximizing benefits from crowdsourced data,” *Computational and Mathematical Organization Theory*, vol. 18, pp. 257–279, Sep 2012.
- [19] A. Lijphart, “Constitutional choices for new democracies,” *Journal of Democracy*, vol. 2, no. 1, p. 7284, 1991.
- [20] A. W. Saxonhouse, “Athenian democracy: Modern mythmakers and ancient theorists,” *American Political Science Association*, vol. 26, p. 486490, Sep 1993.
- [21] L. B. Carter, *The quiet Athenian*. Clarendon Press, 1986.
- [22] M. H. Hansen, *The Athenian Ecclesia*. Museum Tusulanum Press, 1983.
- [23] J. Kruger and D. Dunning, “Unskilled and unaware of it: How difficulties in recognizing ones own incompetence lead to inflated self-assessments.,” *Journal of Personality and Social Psychology*, vol. 77, no. 6, p. 11211134, 1999.
- [24] M. Fleischmann, M. Amirpur, A. Benlian, and T. Hess, “Cognitive biases in information systems research: a scientometric analysis,” in *ECIS*, 2014.

- [25] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, “Crowdsourcing with smartphones,” *IEEE Internet Computing*, vol. 16, pp. 36–44, June 2012.
- [26] “Duolingo.” <https://www.duolingo.com/>.
- [27] J. Phuttharak and S. W. Loke, “Logiccrowd: a declarative programming platform for mobile crowdsourcing,” in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2013)*, pp. 1323–1330, July 2013.
- [28] C. Costa, C. Laoudias, D. Zeinalipour-Yazti, and D. Gunopulos, “Smart-trace: Finding similar trajectories in smartphone networks without disclosing the traces,” in *IEEE 27th International Conference on Data Engineering (ICDE 2011)*, pp. 1288–1291, April 2011.
- [29] M. Constantinides, G. Constantinou, A. Panteli, T. Phokas, G. Chatzimilioudis, and D. Zeinalipour-Yazti, “Proximity interactions with crowdcast,” in *The 11th Hellenic Data Management Symposium*, 2012.
- [30] A. Konstantinidis, C. Aplitsiotis, and D. Zeinalipour-Yazti, “Smartp2p: A multiobjective framework for finding social content in p2p smartphone networks,” in *IEEE 13th International Conference on Mobile Data Management (MDM 2012)*, pp. 324–327, July 2012.
- [31] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, “The pothole patrol: Using a mobile sensor network for road surface monitoring,” in *The Sixth Annual International conference on Mobile Systems, Applications and Services (MobiSys 2008)*, June 2008.



- [32] “Waze.” <https://www.waze.com/>.
- [33] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, “Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones,” in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*, pp. 85–98, 2009.
- [34] E. Aubry, T. Silverston, A. Lahmadi, and O. Festor, “Crowdout: a mobile crowdsourcing service for road safety in digital cities,” in *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops 2014)*, pp. 86–91, March 2014.
- [35] J. Sun, R. Zhang, X. Jin, and Y. Zhang, “Securefind: Secure and privacy-preserving object finding via mobile crowdsourcing,” *IEEE Transactions on Wireless Communications*, vol. PP, pp. 1–1, October 2015.
- [36] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, “Smartphone-based crowdsourcing for network monitoring: Opportunities, challenges, and a case study,” *IEEE Communications Magazine*, vol. 52, pp. 106–113, January 2014.
- [37] “Caida.” <http://www.caida.org/home/>.
- [38] A. Faggiani, E. Gregori, L. Lenzini, S. Mainardi, and A. Vecchio, “On the feasibility of measuring the internet through smartphone-based crowdsourcing,” in *10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt 2012)*, pp. 318–323, May 2012.

- [39] “Operators can now crowdsource data on mobile broadband quality.” <http://networks.nokia.com/news-events/press-room/press-releases/operators-can-now-crowdsource-data-on-mobile-broadband-quality>, November 2010.
- [40] H. M. V. Go, J. C. B. Pabico, J. D. Caro, and M. L. Tee, “Crowdsourcing for healthcare resource allocation,” in *6th International Conference on Information, Intelligence, Systems and Applications (IISA 2015)*, pp. 1–6, July 2015.
- [41] L. I. Besaleva and A. C. Weaver, “Crowdhelp: m-health application for emergency response improvement through crowdsourced and sensor-detected information,” in *Wireless Telecommunications Symposium (WTS 2014)*, pp. 1–5, April 2014.
- [42] J. G. Fiscus, “A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover),” in *1997 IEEE Workshop on Automatic Speech Recognition and Understanding Proceedings*, pp. 347–354, Dec 1997.
- [43] S. Chowdhury, A. Ghosh, E. Stepanov, A. Orkan Bayer, G. Riccardi, and I. Klasinas, “Cross-language transfer of semantic annotation via targeted crowdsourcing,” in *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 09 2014.
- [44] S. A. Chowdhury, M. C. Lafarga, A. Ghosh, E. A. Stepanov, A. O. Bayer, G. Riccardi, F. García, and E. S. Arnal, “Selection and aggregation techniques for crowdsourced semantic annotation task,” in *INTERSPEECH*, 2015.

- [45] E. A. Stepanov, S. A. Chowdhury, A. O. Bayer, A. Ghosh, I. Klasinas, M. Calvo, E. Sanchis, and G. Riccardi, “Cross-language transfer of semantic annotation via targeted crowdsourcing: Task design and evaluation,” *Lang. Resour. Eval.*, vol. 52, pp. 341–364, Mar. 2018.
- [46] “Wikipedia.” <https://www.wikipedia.org/>.
- [47] L. P. Cox, “Truth in crowdsourcing,” *IEEE Security & Privacy*, vol. 9, pp. 74–76, September 2011.
- [48] K. Yang, K. Zhang, J. Ren, and X. Shen, “Security and privacy in mobile crowdsourcing networks: Challenges and opportunities,” *IEEE Communications Magazine*, vol. 53, pp. 75–81, August 2015.
- [49] T. Eisner, “Recruiting smartphone users as partners in telecom fraud & security control,” July 2013.
- [50] Z. Dong and L. J. Camp, “Peersec: Towards peer production and crowdsourcing for enhanced security,” in *Proceedings of the 7th USENIX Conference on Hot Topics in Security (HotSec 2012)*, 2012.
- [51] 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 (SPSM 2011)*, pp. 15–26, 2011.
- [52] S. Amini, J. Lin, J. Hong, J. Lindqvist, and J. Zhan, “Towards scalable evaluation of mobile applications through crowdsourcing and automation,” technical report, Carnegie Mellon CyLab, February 2012.

- [53] M. Gander, C. Sauerwein, and R. Breu, “Assessing real-time malware threats,” in *IEEE International Conference on Software Quality, Reliability and Security - Companion (QRS-C 2015)*, pp. 6–13, August 2015.
- [54] C. Christoforidis, V. Vlachos, and I. Androulidakis, “A crowdsourcing approach to protect against novel malware threats,” in *22nd Telecommunications Forum Telfor (TELFOR 2014)*, pp. 1063–1066, November 2014.
- [55] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, “Rre: A game-theoretic intrusion response and recovery engine,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 395–406, February 2014.
- [56] R. U. Rehman, *Intrusion Detection Systems with Snort: Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall PTR, 2003.
- [57] B. Paramasiva and K. M. Pitchai, “Modeling intrusion detection in mobile ad hoc networks as a non cooperative game,” in *International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME 2013)*, pp. 300–306, February 2013.
- [58] M. Ghorbani and M. R. Hashemi, “Networked ids configuration in heterogeneous networks - a game theory approach,” in *23rd Iranian Conference on Electrical Engineering (ICEE 2015)*, pp. 1000–1005, May 2015.
- [59] A. Bradai and H. Affi, “Game theoretic framework for reputation-based distributed intrusion detection,” in *International Conference on Social Computing (SocialCom 2013)*, pp. 558–563, September 2013.
- [60] H. Xie, J. C. Lui, J. W. Jiang, and W. Chen, “Incentive mechanism and protocol design for crowdsourcing systems,” in *52nd Annual Allerton Confer-*

- ence on Communication, Control, and Computing (Allerton 2014)*, pp. 140–147, September 2014.
- [61] E. Aldhahri, V. Shandilya, and S. Shiva, “Towards an effective crowdsourcing recommendation system: A survey of the state-of-the-art,” in *IEEE Symposium on Service-Oriented System Engineering (SOSE 2015)*, pp. 372–377, March 2015.
- [62] “What is the price of free?.” <http://www.cam.ac.uk/research/news/what-is-the-price-of-free>.
- [63] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, “Android permissions: User attention, comprehension, and behavior,” in *Proceedings of the Eighth Symposium on Usable Privacy and Security (SOUPS 2012)*, pp. 1–14, July 2012.
- [64] B. Rashidi and C. Fung, “A game-theoretic model for defending against malicious users in redroid,” in *IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, pp. 1339–1344, May 2015.
- [65] D. Movshovitz-Attias, Y. Movshovitz-Attias, P. Steenkiste, and C. Faloutsos, “Analysis of the reputation system and user contributions on a question answering website: Stackoverflow,” in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pp. 886–893, August 2013.
- [66] A. Pal, F. M. Harper, and J. A. Konstan, “Exploring question selection bias to identify experts and potential experts in community question answering,” *ACM Transactions on Information Systems (TOIS 2012)*, vol. 30, pp. 1–28, May 2012.

- [67] D. Vogiatzis and N. Tsapatsoulis, “Modeling user networks in recommender systems,” in *Third International Workshop on Semantic Media Adaptation and Personalization (SMAP 2008)*, pp. 106–111, December 2008.
- [68] A. Pal, R. Farzan, J. A. Konstan, and R. E. Kraut, “Early detection of potential experts in question answering communities,” in *Proceedings of the 19th International Conference on User Modeling, Adaption, and Personalization (UMAP 2011)*, pp. 231–242, July 2011.
- [69] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.,” technical report, Stanford InfoLab, November 1999.
- [70] M. Bouguessa, B. Dumoulin, and S. Wang, “Identifying authoritative actors in question-answering forums - the case of yahoo! answers,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*, pp. 866–874, August 2008.
- [71] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, “Finding high-quality content in social media,” in *Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM 2008)*, pp. 183–194, February 2008.
- [72] E. Karataev and V. Zadorozhny, “Adaptive social learning based on crowdsourcing,” *IEEE Transactions on Learning Technologies*, vol. PP, pp. 1–1, January 2016.
- [73] L. Gottlieb, G. Friedland, J. Choi, P. Kelm, and T. Sikora, “Creating experts from the crowd: Techniques for finding workers for difficult tasks,” *IEEE Transactions on Multimedia*, vol. 16, pp. 2075–2079, November 2014.

- [74] S. Tiwari and S. Kaushik, “Information enrichment for tourist spot recommender system using location aware crowdsourcing,” in *IEEE 15th International Conference on Mobile Data Management (MDM 2014)*, vol. 2, pp. 11–14, July 2014.
- [75] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. R. Motahari-Nezhad, E. Bertino, and S. Dustdar, “Quality control in crowdsourcing systems: Issues and directions,” *IEEE Internet Computing*, vol. 17, pp. 76–81, Mar. 2013.
- [76] D. Schall, F. Skopik, and S. Dustdar, “Expert discovery and interactions in mixed service-oriented systems,” *IEEE Trans. Serv. Comput.*, vol. 5, pp. 233–245, Jan. 2012.
- [77] A. P. Dawid and A. M. Skene, “Maximum likelihood estimation of observer error-rates using the em algorithm,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 20–28, 1979.
- [78] P. G. Ipeirotis, F. Provost, and J. Wang, “Quality management on amazon mechanical turk,” in *Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP ’10*, (New York, NY, USA), pp. 64–67, ACM, 2010.
- [79] A. Sorokin and D. Forsyth, “Utility data annotation with amazon mechanical turk,” in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, June 2008.
- [80] A. Kittur, E. H. Chi, and B. Suh, “Crowdsourcing user studies with mechanical turk,” in *Proceedings of the SIGCHI Conference on Human Factors*

- in Computing Systems*, CHI '08, (New York, NY, USA), pp. 453–456, ACM, 2008.
- [81] “Crowdflower.” Web Page. Accessed: 2018-08-18.
- [82] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, “re-captcha: Human-based character recognition via web security measures,” *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.
- [83] L. von Ahn and L. Dabbish, “Designing games with a purpose,” *Commun. ACM*, vol. 51, pp. 58–67, Aug. 2008.
- [84] L. von Ahn and L. Dabbish, “Labeling images with a computer game,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, (New York, NY, USA), pp. 319–326, ACM, 2004.
- [85] E. Law and L. von Ahn, “Input-agreement: A new mechanism for collecting data using human computation games,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, (New York, NY, USA), pp. 1197–1206, ACM, 2009.
- [86] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, “Soylent: A word processor with a crowd inside,” in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, (New York, NY, USA), pp. 313–322, ACM, 2010.
- [87] E. Bonabeau, “Decisions 2.0: The power of collective intelligence,” *MIT Sloan Management Review*, vol. 50, pp. 45–52, 12 2009.



- [88] S. Basu Roy, I. Lykourantzou, S. Thirumuruganathan, S. Amer-Yahia, and G. Das, “Crowds, not drones: Modeling human factors in interactive crowdsourcing,” in *DBCrowd 2013 - VLDB Workshop on Databases and Crowdsourcing* (R. Cheng, A. D. Sarma, S. Maniu, and P. Senellart, eds.), CEUR Workshop Proceedings, (Riva del Garda, Trento, Italy), pp. 39–42, CEUR-WS, Aug. 2013.
- [89] B. Faltings, R. Jurca, P. Pu, and B. D. Tran, “Incentives to counter bias in human computation,” in *HCOMP*, 2014.
- [90] C. Eickhoff, “Cognitive biases in crowdsourcing,” in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM '18*, (New York, NY, USA), pp. 162–170, ACM, 2018.
- [91] U. Gadiraju, B. Fetahu, R. Kawase, P. Siehndel, and S. Dietze, “Using worker self-assessments for competence-based pre-selection in crowdsourcing microtasks,” *ACM Trans. Comput.-Hum. Interact.*, vol. 24, pp. 30:1–30:26, Aug. 2017.
- [92] D. Arp, M. Spreitzenbarth, H. Gascon, and K. Rieck, “Drebin: Effective and explainable detection of android malware in your pocket,” in *2014 Network and Distributed System Security Symposium, NDSS '14*, (San Diego, CA, USA), February 2014.
- [93] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, “Droidmat: Android malware detection through manifest and api calls tracing,” in *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security, ASIAJCIS '12*, (Washington, DC, USA), pp. 62–69, 2012.

- [94] Y. Aafer, W. Du, and H. Yin, *DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android*, pp. 86–103. SecureComm '13, Sydney, NSW, Australia: Springer International Publishing, September 2013.
- [95] W. Li, J. Ge, and G. Dai, “Detecting malware for android platform: An svm-based approach,” in *2nd IEEE International Conference on Cyber Security and Cloud Computing*, CSCloud '15, (New York, USA), pp. 464–469, November 2015.
- [96] W. Park, S. Joong Kim, and W. Ryu, “Detecting malware with similarity to android applications,” in *6th International Conference on Information and Communication Technology Convergence*, ICTC '15, (Jeju Island, Korea), pp. 1249–1251, October 2015.
- [97] S. Chen, M. Xue, and L. Xu, “Towards adversarial detection of mobile malware: Poster,” in *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, (New York, NY, USA), pp. 415–416, 2016.
- [98] Y. Liu, Y. Zhang, H. Li, and X. Chen, “A hybrid malware detecting scheme for mobile android applications,” in *IEEE International Conference on Consumer Electronics*, ICCE '16, (Las Vegas, Nev, USA), pp. 155–156, January 2016.
- [99] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, “Droid-sec: Deep learning in android malware detection,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (Chicago, Illinois, USA), pp. 371–372, August 2014.

- [100] S. Alam, R. Riley, I. Sogukpinar, and N. Carkaci, “Droidclone: Detecting android malware variants by exposing code clones,” in *6th International Conference on Digital Information and Communication Technology and its Applications*, DICTAP '16, (Konya, Turkey), pp. 79–84, July 2016.
- [101] J. R. Cordy and C. K. Roy, “The nicad clone detector,” in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, (Washington, DC, USA), pp. 219–220, June 2011.
- [102] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, “Detecting code reuse in android applications using component-based control flow graph,” in *Proceedings of 29th International Conference on Systems Security and Privacy Protection*, SEC '14, (Marrakech, Morocco), pp. 142–155, June 2014.
- [103] A. Kumar, K. P. Sagar, K. S. Kuppusamy, and G. Aghila, “Machine learning based malware classification for android applications using multimodal image representations,” in *10th International Conference on Intelligent Systems and Control*, ISCO '16, (Coimbatore, Tamilnadu, India), pp. 1–6, January 2016.
- [104] “Download distribution of android apps.”  
<http://www.appbrain.com/stats/android-app-downloads>.
- [105] “Google play.” [https://en.wikipedia.org/wiki/Google\\_Play](https://en.wikipedia.org/wiki/Google_Play).
- [106] “So many apps, so much more time for entertainment.”  
<http://www.nielsen.com/us/en/insights/news/2015/so-many-apps-so-much-more-time-for-entertainment.html>.

- [107] “Growth of time spent on mobile devices slows.” <http://www.emarketer.com/Article/Growth-of-Time-Spent-on-Mobile-Devices-Slows/1013072>, Oct. 2015.
- [108] “Telus at a glance.” [https://about.telus.com/community/english/news\\_centre/company\\_overview/telus\\_at\\_a\\_glance](https://about.telus.com/community/english/news_centre/company_overview/telus_at_a_glance).
- [109] Y.-H. Kim, C.-Y. Chiu, and Z. Zou, “Know thyself: Misperceptions of actual performance undermine achievement motivation, future performance, and subjective well-being.,” *Journal of Personality and Social Psychology*, vol. 99, p. 395409, Sep 2010.
- [110] Y. J. Park and L. Santos-Pinto, “Overconfidence in tournaments: evidence from the field,” *Theory and Decision*, vol. 69, pp. 143–166, Jul 2010.
- [111] W. B. Liebrand, D. M. Messick, and F. J. Wolters, “Why we are fairer than others: A cross-cultural replication and extension,” *Journal of Experimental Social Psychology*, vol. 22, no. 6, p. 590604, 1986.
- [112] W. Poundstone, “”the dunning-kruger president”,” Jan 2017.
- [113] M. M. Roy and M. J. Liersch, “I am a better driver than you think: examining self-enhancement for driving ability,” *Journal of Applied Social Psychology*, vol. 43, p. 16481659, Aug 2013.
- [114] G. Rasch, *On General Laws and the Meaning of Measurement in Psychology*. Danmarks pdagogiske Institut, 1961.
- [115] J. McCoy and D. Prelec, “A statistical model for aggregating judgments by incorporating peer predictions,” *ArXiv e-prints*, Mar 2017.

- [116] D. Berend and J. Paroush, “When is condorcet’s jury theorem valid?” *Social Choice and Welfare*, vol. 15, no. 4, pp. 481–488, 1998.
- [117] A. Pal, R. Farzan, J. A. Konstan, and R. E. Kraut, “Early detection of potential experts in question answering communities,” in *Proceedings of the 19th International Conference on User Modeling, Adaption, and Personalization*, UMAP’11, (Berlin, Heidelberg), pp. 231–242, Springer-Verlag, 2011.
- [118] J. Zhang, M. S. Ackerman, and L. Adamic, “Expertise networks in online communities: Structure and algorithms,” in *Proceedings of the 16th International Conference on World Wide Web*, WWW ’07, (New York, NY, USA), pp. 221–230, ACM, 2007.
- [119] M. Bouguessa, B. Dumoulin, and S. Wang, “Identifying authoritative actors in question-answering forums: The case of yahoo! answers,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’08, (New York, NY, USA), pp. 866–874, ACM, 2008.
- [120] D. Attiaoui, A. Martin, and B. Ben Yaghlane, “Belief measure of expertise for experts detection in question answering communities: case study stack overflow,” *Procedia Computer Science*, vol. 112, pp. 622–631, 2017.
- [121] P. Welinder and P. Perona, “Online crowdsourcing: Rating annotators and obtaining cost-effective labels,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 25–32, Jun 2010.
- [122] J. Whitehill, T.-f. Wu, J. Bergsma, J. R. Movellan, and P. L. Ruvolo, “Whose vote should count more: Optimal integration of labels from label-

- ers of unknown expertise,” in *Advances in Neural Information Processing Systems 22*, pp. 2035–2043, Curran Associates, Inc., 2009.
- [123] P. C. Kyllonen and J. Zu, “Use of response time for measuring cognitive ability,” *Journal of Intelligence*, vol. 4, no. 4, 2016.
- [124] M. D. Lee, M. Steyvers, M. De Young, and B. Miller, “Inferring expertise in knowledge and prediction ranking tasks,” *Topics in Cognitive Science*, vol. 4, no. 1, p. 151163, 2012.
- [125] V. C. Raykar, S. Yu, L. H. Zhao, A. Jerebko, C. Florin, G. H. Valadez, L. Bogoni, and L. Moy, “Supervised learning from multiple experts: Whom to trust when everyone lies a bit,” in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, (New York, NY, USA), pp. 889–896, ACM, 2009.
- [126] Y. Bachrach, T. Graepel, T. Minka, and J. Guiver, “How to grade a test without knowing the answers - a bayesian graphical model for adaptive crowdsourcing and aptitude testing,” in *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, vol. 2, June 2012.
- [127] B. Lakshminarayanan and Y. Whye Teh, “Inferring ground truth from multi-annotator ordinal data: a probabilistic approach,” Apr 2013.
- [128] B. I. Aydin, Y. S. Yilmaz, and M. Demirbas, “A crowdsourced who wants to be a millionaire? player,” *Concurrency and Computation: Practice and Experience*.
- [129] Q. Li and P. K. Varshney, “Does confidence reporting from the crowd benefit crowdsourcing performance?,” *CoRR*, vol. abs/1704.00768, 2017.

- [130] Q. Li, Y. Li, J. Gao, L. Su, B. Zhao, M. Demirbas, W. Fan, and J. Han, “A confidence-aware approach for truth discovery on long-tail data,” *Proc. VLDB Endow.*, vol. 8, pp. 425–436, December 2014.
- [131] X. Yin, J. Han, and P. S. Yu, “Truth discovery with multiple conflicting information providers on the web,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, pp. 796–808, June 2008.
- [132] X. L. Dong, B. Saha, and D. Srivastava, “Less is more: selecting sources wisely for integration,” in *Proceedings of the 39th international conference on Very Large Data Bases, PVLDB’13*, pp. 37–48, VLDB Endowment, 2013.
- [133] J. Pasternack and D. Roth, “Knowing what to believe (when you already know something),” in *Proceedings of the 23rd International Conference on Computational Linguistics, COLING ’10*, (Stroudsburg, PA, USA), pp. 877–885, Association for Computational Linguistics, 2010.
- [134] A. Galland, S. Abiteboul, A. Marian, and P. Senellart, “Corroborating information from disagreeing views,” in *Proceedings of the Third ACM International Conference on Web Search and Data Mining, WSDM ’10*, (New York, NY, USA), pp. 131–140, ACM, 2010.
- [135] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han, “Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation,” in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD ’14*, (New York, NY, USA), pp. 1187–1198, ACM, 2014.

- [136] F. Ma, Y. Li, Q. Li, M. Qiu, J. Gao, S. Zhi, L. Su, B. Zhao, H. Ji, and J. Han, “Faitcrowd: Fine grained truth discovery for crowdsourced data aggregation,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’15, (New York, NY, USA), pp. 745–754, ACM, 2015.
- [137] A. P. Dawid and A. M. Skene, “Maximum likelihood estimation of observer error-rates using the em algorithm,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 20–28, 1979.
- [138] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux, “Zencrowd: Leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking,” in *Proceedings of the 21st International Conference on World Wide Web*, WWW ’12, (New York, NY, USA), pp. 469–478, ACM, 2012.
- [139] D. Prelec, H. S. Seung, and J. McCoy, “A solution to the single-question crowd wisdom problem,” *Nature*, vol. 541.
- [140] A. Laan, G. Madirolas, and G. Polavieja, “Rescuing collective wisdom when the average group opinion is wrong,” *Frontiers in Robotics and AI*, vol. 4, Nov 2017.
- [141] D. Bang and C. D. Frith, “Making better decisions in groups,” *Royal Society Open Science*, vol. 4, no. 8, 2017.
- [142] L. Ross, D. Greene, and P. House, “The false consensus effect: An egocentric bias in social perception and attribution processes,” *Journal of Experimental Social Psychology*, vol. 13, no. 3, p. 279301, 1977.
- [143] “Firebase.” <https://firebase.google.com/>.



- [144] “Comodo: Global leader in cyber security solutions.”  
<https://www.comodo.com/>.
- [145] M. Matsumoto and T. Nishimura, “Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Transactions on Modeling and Computer Simulation*, vol. 8, pp. 3–30, Jan 1998.
- [146] “Siri.” <https://www.apple.com/siri/>.
- [147] “Alexa.” <https://alexa.amazon.com/>.
- [148] “Google assistant.” <https://assistant.google.com/>.
- [149] “Google docs.” <https://www.google.com/docs/about/>.
- [150] “Microsoft office 365.” <https://www.office.com/>.
- [151] “Microsoft azure.” <https://azure.microsoft.com/en-us/>.
- [152] “Google cloud.” <https://cloud.google.com/>.
- [153] “icloud.” <https://www.icloud.com/>.
- [154] “Cloudsim.” <https://www.cloudsimapp.com/>.
- [155] A. York, “Best times to post on social media: 2018 industry research,” Jun 2018.

