

Semester Thesis - Fall Term 2012/2013

Improved Offline Operation of Twimight



Fengrui Shi
fshi@student.ethz.ch
SA-2012-36

March 15, 2013

Tutor:	Prof. Dr. Bernhard Plattner	plattner@tik.ee.ethz.ch
Supervisor:	Dr. Theus Hossmann	hossmann@tik.ee.ethz.ch
	Paolo Carta	paolo.carta@tik.ee.ethz.ch

Abstract

Twimight, in comparison with other Twitter clients, has demonstrated unique off-line functionalities, which are able to mitigate communication outages due to failing or congested infrastructure during natural disasters with the help of opportunistic message spreading via Bluetooth. However, as a Twitter client, Twimight does not support some basic functions of Twitter such as attaching photos when posting a Tweet. As web pages and videos convey more information, users of Twitter are showing more interests in the media attached to the a Tweet such as web pages and videos than Tweet itself. As a result of current situation, some improvements have to be made by adding new features to Twimight such as allowing users to send Tweets along with photos in disaster mode and to view linked web pages even without Internet access. In addition, since mobile devices such as smart phones have limited amount of battery power and wireless communication consumes a large amount of energy, once electrical infrastructure goes down in a natural disaster, the efficiency of scanning and finding peers will become a fatal issue for Twimight which sends disaster messages opportunistically via Bluetooth scanning and pairing. The scanning frequency has to be optimized in order to save energy and at the same time, not to lose too much useful information from other peers, i.e. to attain good accuracy. However, at present, only several simple scanning policies have been used to explore the trade-off between energy and accuracy which are not good solutions to the current problem of Twimight.

In this semester thesis, new functionalities for Twimight supporting photos as well as web pages in both normal and disaster mode have been implemented. Design principles of these modules are illustrated and performance evaluations and some statistics of newly added functions are briefly discussed. Then, new adaptive scanning policies aimed to solve energy problem of opportunistic communication are proposed. Performance of these algorithms are simulated using mobility traces data. Simulation results show that although the existing models have already achieved good energy saving level (with 20% – 40%), the algorithm proposed by us can further improve energy saving by another 10%

– 30% given different mobility traces.

Preface

I want to express my sincere gratitude to everybody who made this thesis possible, especially to

- *Professor Bernhard Plattner*, for providing the opportunity to do this thesis in his group and gives some valuable advice on this thesis.
- *Dr. Theus Hossmann*, for his continuous help and feedback during the whole period of this thesis.
- *Paolo Carta*, for his time and efforts in instructing and helping me on coding issues.
- *Anonymity*, for sharing their knowledge and contributing their codes online.
- *My family and my girl friend*, for their support and understanding.

Zürich, 15.03.2013

Fengrui Shi

Task Description

Introduction

Twimight (see [1]) is an open source Twitter client for Android mobile phones, developed at the Communication Systems Group at ETH Zurich. Additionally to normal Twitter functionality, it supports a fallback mode of operation called "disaster mode" which the user can enable in case of loss of connectivity (e.g., due to Internet outage in natural disasters). In disaster mode Tweets spread epidemically from phone to phone (using Bluetooth or WLAN-Opp) until they reach a phone with connectivity, which will upload the Tweets to Twitter.

This offline operation is one of the key features that distinguishes Twimight from other Twitter clients and is already implemented for the normal Twitter functionality. Yet, the interesting content is often not in the Tweets itself but in content (websites, pictures, videos), which is linked from Tweets. For this linked content, Twimight currently does not support offline operation.

The goal of this semester thesis is thus to extend Twimight with new features that allow offline consumption of linked content like pictures, websites or videos.

Tasks

1. Familiarization: The student familiarizes with the Twimight code and the literature related to it (e.g., [2], [3] and references therein).
2. Photos: As a first implementation task, the Twimight client should be extended with the functionality for photos. In the interface, where a tweet is sent, an option to take a photo from the camera or library and add it to the Tweet. If connectivity is available, the photo should be uploaded to the Twitter API, if not, it should be stored locally for later upload.

-
3. Websites: Android allows storing entire websites as webarchives for offline viewing [4]. The Twimight interface should be extended with a button that allows the user to download all linked websites for later offline consumption.
 4. Predictive prefetching: The download of websites should be extended with functionality to automatically pre-fetch linked websites when connected to WiFi to reduce 3G data traffic.
 5. Report and presentation: At the end of the project, the student will hand in a scientific report. Further, the student will give a presentation describing the project and ideally demonstrating the implemented offline capabilities.

Contents

Abstract	I
Preface	III
Task Description	V
Table of Contents	VII
List of Figures	IX
List of Tables	XI
1 Introduction	1
1.1 Thesis Outline	3
2 Background	5
2.1 Twitter	5
2.2 Android OS	7
2.3 Twimight	8
2.4 Delay Tolerant Networks	8
2.4.1 Energy Issues	9
2.4.2 Related Works	9

3	Media in Offline Mode	11
3.1	Overview	11
3.2	Normal Mode	11
3.2.1	Photo	13
3.2.2	Off-line Web View	15
3.3	Disaster Mode	22
3.3.1	Data Structure	22
3.4	Evaluation	24
4	Adaptive Scanning	27
4.1	Performance metrics	27
4.2	Existing Model	28
4.3	Adaptive Scanning	29
4.3.1	Probabilistic scanning	29
4.3.2	Linear Model	29
4.4	Proposed Model	31
4.5	Results and Analysis	33
4.5.1	Simulation Results	33
4.5.2	Analysis	36
5	Conclusions and Outlook	37
5.1	Conclusions	37
5.2	Outlook	37
A	User Know-how	39
A.1	Requirements	39
A.2	Instructions	39
A.2.1	Normal Mode	39
A.2.2	Disaster Mode	41
B	Developer Know-how	43
C	Time Schedule	44
	Bibliography	47

List of Figures

2.1	Tweets per day [7]	6
2.2	Growth of Average Web Page Size and Number of Objects [9]	6
2.3	Distribution of platform versions [12]	7
2.4	Historical distribution of platform versions [12]	8
2.5	Scan interval and peer numbers over time[15]	10
3.1	Post a Tweet with photo	12
3.3	Flow chart of photo part	14
3.4	Database structure of web pages	15
3.5	Flow chart of automatic web download	17
3.7	Flow chart of manual web download with Wi-Fi access	19
3.8	Flow chart of manual web download with mobile network (3G) access	21
3.10	Web page database table	23
3.11	Photo packet structure	23
3.12	Web page packet structure	24
3.13	Energy consumption in normal mode	25
4.1	CDF of the number of Bluetooth devices seen in periodic scans from 150 Nokia N95 mobile phones. This data covers five months in 2008 [15]	28
4.2	Simulated energy consumption based on ETH trace, w.r.t different initial probabilities with 1 min scan interval	34
4.3	Simulated energy consumption based on Huggle trace, w.r.t different initial probabilities with 1 min scan interval	34

4.4	Simulated energy consumption based on MIT trace, w.r.t different initial probabilities with 30 min scan interval	35
A.1	Turning on automatic web download	40
A.2	Manual web download	40
A.3	Download web page for a single Tweet	41
A.4	Enable disaster mode	42
A.5	Enable web share	42
B.1	Online repository	43
C.1	Time schedule(planned)	45

List of Tables

3.1	Statistics about web pages	25
4.1	Energy saving with 80% accuracy	35
4.2	Energy saving with 85% accuracy	35
4.3	Energy saving with 90% accuracy	35
4.4	Energy saving with 95% accuracy	36

1

Introduction

As a Twitter client, Twimight distinguishes itself by enabling a function called disaster mode [3]. Although both Twimight and other Twitter clients support basic functions for social networking, unlike common Twitter clients which need access to wired or wireless network facilities, Twimight can still provide certain useful functionalities in order to spread the message and to exchange information even when network infrastructures are down. This goal can be realized with the help of opportunistic communication mechanism. In principle, each device will try to discover other devices nearby and establish communications to them. Messages will thus be relayed opportunistically from one node to another via these P2P links. In recent years, this is not rarely the case when networks are not working and in the meanwhile, people need network to communicate with each other and to spread information more than ever. For example, during the earthquake and consequent tsunami happened in Mar. 2011 in Japan, the network overload occurred and especially mobile traffic increased in volume over those days [5]. By operating in disaster mode, Twimight can help solve the problem of heavy dependency on network infrastructure in emergencies. Users can communicate with each other and share Tweets opportunistically. Twimight, in this point, demonstrated a great idea of combining existing social platforms and opportunistic communication technique to solve the problem. However, before Twimight can truly become a well-versed application and gain a large amount of user base, there are still two major challenges having to be overcome.

The first challenge is how to make Twimight more useful in daily life and thus gain better user experiences. On one hand, besides disaster functions, current versions of Twimight can only support functions such as posting, re-tweeting, replying, favouriting or searching a Tweet and following other Twitter users. These are quite limited functionalities compared to other Twitter clients which better explores the functions

that Twitter API offers, for instance, users can post a Tweet together with a picture via Twitter API. On the other hand, Twitter also supports other forms of media like websites by supporting adding a link together with a Tweet. Some statistics of Twitter indicated the importance of links for carry information. However, functions involved with links in most Twitter clients have not been well exploited so far.

In order to solve the first issue, new functionalities have been proposed and implemented for Twimight. Newly added functions support photos such that users can attach photos to Tweets and post the Tweets along with photos to Twitter. Twimight can therefore meet all the basic requirements of Twitter clients users. Besides, in order to cater the demand of better exploiting links of Tweets, a so-called “Off-line Web View” function is implemented in Twimight. After enabling this function, when the phone has access to Internet via Wi-Fi or 3G, this function downloads the web pages linked to the Tweets beforehand such that even when there are no Internet connections available, such as in a subway, the users can still browse the content of the links off-line.

The other challenge faced by Twimight is the energy issue. With the emergence of smart phones, energy consumption has been always the most tricky part of application designers. The same situation applies to Twimight. In disaster mode, as Twimight enables opportunistic communication via Bluetooth, battery energy can be drained really fast. Often, when disaster happens, electrical infrastructure can be malfunctioning as well. Therefore, the energy issue will be a decisive factor for Twimight to better support disaster mode. Intuitively, if a device wants to establish communication with others, because it does not know anything about its surroundings, this device has to scan periodically, usually with a fixed time interval, in order to discover nearby active devices. During this process, this device first opens Bluetooth, starts scanning and if it receives positive results (active devices are around), it will try to establish communication with those active devices. This mechanism does not guarantee discovering devices each time it scans, what’s worse, over half of such scans find nothing. If the phone repeats this process quite often, large amount of energy will be consumed by doing nothing.

As for the energy issue of Bluetooth scanning, a new adaptive scanning model is proposed. The idea is to choose a probability p_{n+1} for next scan $n + 1$ based on the scanning results obtained in both last $n - 1$ and current scans n . Unlike commonly used scanning policies, which increase the probability p_n “linearly” by a fixed number Δp to get p_{n+1} if the peers found are different between two consecutive scans and decrease by an amount otherwise, proposed policy chooses a Δp based on the number of peers found during both last scan and current scan and thus increase or decrease the probability non-linearly. By simulating proposed algorithm based on the data of several mobility traces

(ETH, MIT, Huggle), results show that with the same accuracy (accuracy = captured events/total events, it means the proportion of interesting events the algorithm can capture over all interesting events), energy saving is improved by an amount ranging from 10% to 30% compared to the commonly used linear adaptive scanning algorithm.

1.1 Thesis Outline

This thesis is organized as follows: in next chapter, some essential backgrounds are presented. It briefly introduces Twitter and Android OS to restate the necessities and feasibilities of implementing new functions. Besides, backgrounds of energy issues in Delayed Tolerant Networks and related work are discussed. Chapter 3 shows in detail the structure and design of new functionalities—photos and off-line webpages respectively. Some evaluation results of Twimight are also presented in this chapter. The proposed scanning algorithm will be discussed in chapter 4, by showing the algorithm itself and simulation results of mobility traces. Summary and conclusion are presented in the end in chapter 5.

2

Background

This chapter presents some backgrounds of the project. Section 2.1 gives some points of the benefits to build a Twitter client as well as the importance of links in Twitter by showing some statistics about Twitter and websites. Sections 2.2 discusses the feasibility of implementing fore-mentioned functions on Android platform. Some status quo regarding old versions of Twimight and the importance of new functions for Twimight are described in the next section. In the end, facts about Delay Tolerant Network (DTN) are presented and the energy issue of it is raised and briefly discussed along with introduction of related works.

2.1 Twitter

The fact that Twitter had huge growth rate at the time when the first prototype of Twimight was developed is one of the most important reason of choosing Twitter as the platform [6]. The strong growth of Twitter, both the number of Twitter users and number of Tweets posted, justifies the reason why to develop an application based on Twitter. It has seen almost exponential growth of both figures in recent years. In addition, Twitter provides a convenient and easy-to-deployed API for implementing 3rd party clients.

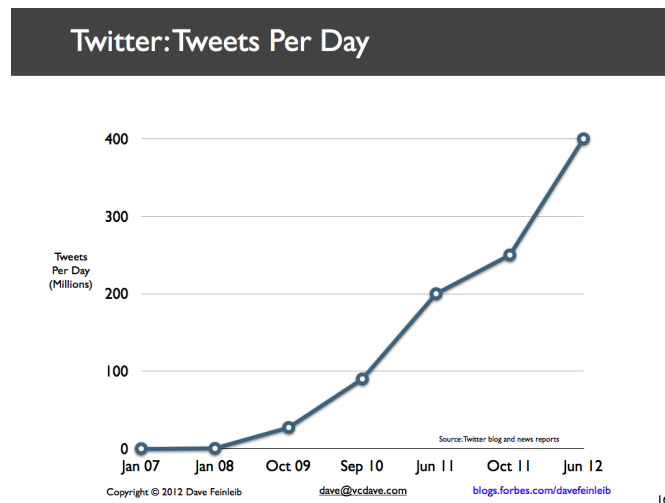


Figure 2.1: Tweets per day [7]

Moreover, two years ago, some statistics revealed by Evan Williams, co-founder of Twitter, showed that 25 % of Tweets contain links to resources like blogs, news, photos and videos [8]. Considering the 140 character limit of a Tweet and the amount of content in a webpage, it is clear that information contained in a webpage far exceeds the information of a plain-text Tweet. Thus, special functionalities must be available on any successful Twitter client to access and view the resources linked in Tweets.

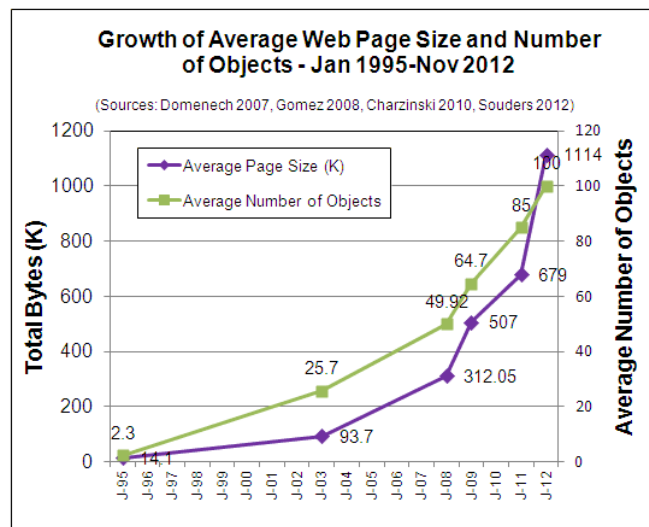


Figure 2.2: Growth of Average Web Page Size and Number of Objects [9]

2.2 Android OS

Android OS is the platform on which Twimight runs. It offers a variety of developing tools to make the development much easier. For example, some API functions can be easily used to download and save web pages as the format of XML. Later if the contents of the saved pages need to be shown in a browser, Android also provides some parsers to extract the information of XML files and display them as HTML text. The “WebView” class on Android [10] is the core for achieving above mentioned functions. This class offers several convenient methods for developing web applications on Android such as building a simple web browser application to display the content of web page. The support from “WebView” class and other packages make the development of the new functions feasible and easier.

Among the methods of the “WebView” class, `saveWebArchive()` is the core method that we used to achieve saving web pages locally [11]. However, this method is only added after Android API level 11, as a consequence, Twimight now only supports phones with an Android OS higher than version 3.1 (Honeycomb). Although this seems to restrict the user base of Twimight, some facts can still justify the choice. First, as the number of Android users is still growing really fast and people change their phones every one or two years or even shorter time, and in the meanwhile, Google is releasing new Android versions every several months. As a result, the number of users for new Android versions will increase very fast. Although at the time of writing, the users of version 3.1 or higher still account for only less than 50 %, the growth rate will increase this number larger soon.

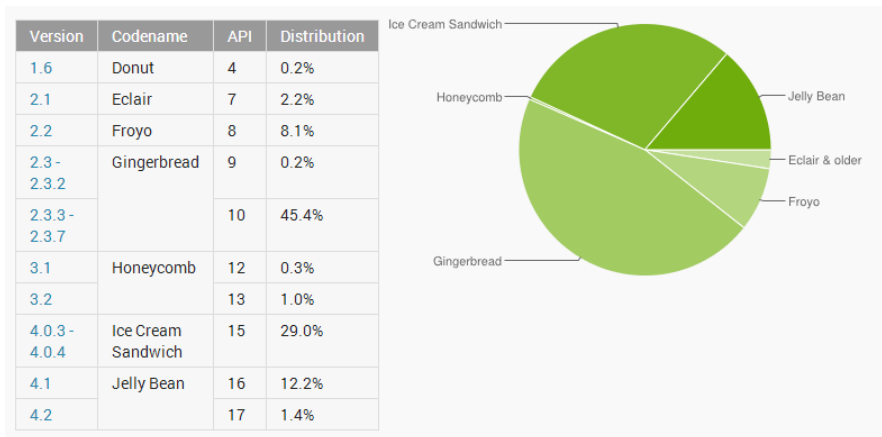


Figure 2.3: Distribution of platform versions [12]

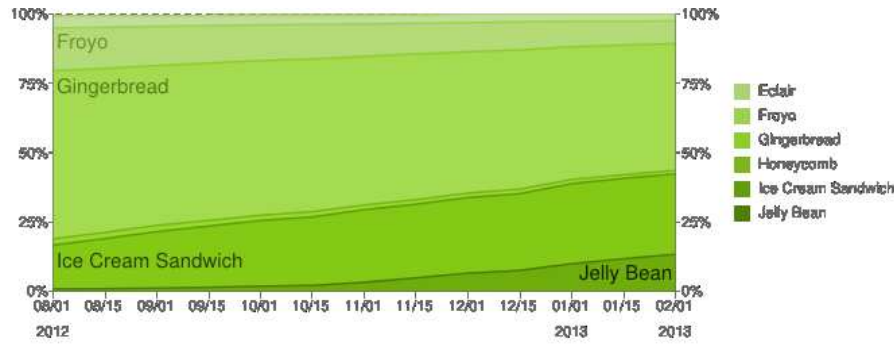


Figure 2.4: Historical distribution of platform versions [12]

2.3 Twimight

There are a number of Twitter clients available in the market and most of them offers full support of functions as the web version of Twitter [13]. Considering functionalities, Twimight has its unique disaster mode which can demonstrate powerful functions when disaster happens, in order to survive and compete with its counterparts after being released on Google Play, Twimight has to offer more support of basic functions of Twitter and thus enhancing user experiences. In general, the lack of support for posting photos to Twitter is one of the major shortcoming of current Twimight versions. That is why the support for photos has to be implemented in Twimight.

As for the support of web pages, in general, available clients (including current versions of Twimight) can only display the links of a Tweet and when users click on the links, the clients will redirect the UI to open the website in a default browser of the operating system. Considering the importance of links to spread information, better way to support links should be added as well, for example to pre-fetch the content for offline viewing, which could greatly enhance the functionality of Twitter clients.

2.4 Delay Tolerant Networks

The traditional wired or wireless networks both operate poorly under the condition with very long delay paths and frequent network partitions [14]. Therefore, once there is no continuous network connectivities in some extreme environments, a special communication strategy is required. For this purpose, the paradigm of Delay Tolerant Network (DTN) was proposed. The introduce of DTN helps to solve the problems of communication such as when a disaster happens and network facilities are down. The principle of

disaster mode is to relay messages among nodes (mobile phones), without using existing network infrastructure.

This message relay has to be achieved using wireless peer to peer communication via Wi-Fi or Bluetooth. However, wireless communication of smart phones consumes a large amount of battery energy, and thus causes the battery to drain soon. Therefore, to give the battery enough life-time is a challenge when using DTN to forward messages.

2.4.1 Energy Issues

In Twilight, the messages are transmitted via Bluetooth. The device has to find some other peers within the range of connectivity before establishing a connection to transmit messages. There are two operations involved in Bluetooth communication - scanning and transmitting. In the old versions of Twilight, scanning operations are performed with a fixed time interval of 2 minutes. However, it is not efficient to scan like this without taking the surroundings into consideration. For example, when the users are walking around in the ruins or in an open field, it is not possible that they will always meet other peers every 2 minutes. As a result, some of the scanning operations are useless and will cause unnecessary energy consumption, which could then result in faster draining of battery power. To improve this, we propose some new and energy efficient scanning algorithms.

2.4.2 Related Works

In order to solve the energy issue of mobile devices, several algorithms have been proposed for regulating Bluetooth scanning. For example, Bluetooth scan will be performed opportunistically with a probability $p_n \in [0, 1]$ for the n_{th} scan. Before each scan, a number q is chosen randomly within range $[0,1]$ (uniform distribution) and it will compare this number q with p_n , if $q < p_n$, then this scan is performed, or discarded otherwise. Besides, each p_n is determined adaptively, which means that p_{n+1} depends on both the result of n_{th} scan and the value of p_n .

Another possible solution is to determine the time interval between two consecutive scans adaptively, for example, a scanning policy has been proposed for using Bluetooth to determine a person's social context [15]. This scanning policy makes decisions on the value of the subsequent scan interval based on the information of topology of a device's peers. If the degree of topology change is large (over a threshold value), then the scan interval will be increased by a fixed value (e.g 10 seconds) and if the degree of topology

change is smaller than the threshold value, the interval will be halved, thus achieving adaptive scanning.

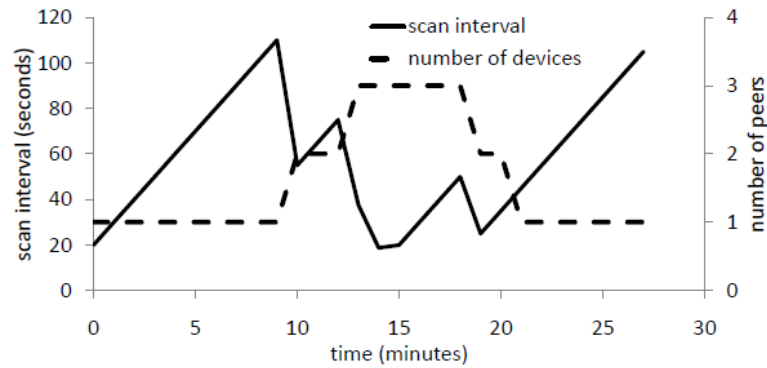


Figure 2.5: Scan interval and peer numbers over time[15]

Figure 2.5 shows an example of the Bluetooth scan interval for a device over time. Initially, it had only one peer device in the vicinity. After 10 minutes, a second peer device is introduced, and at 12 minutes a third peer device. At 18 minutes one device is removed, and then a second one at 20 minutes. The scanning intervals are changing with respect to the change of topology of peers.

3

Media in Offline Mode

This chapter describes the details, including the design and architecture of two new features — photo and off-line web view. At first, an overview of the new functions are presented and then, specific design decisions and implementation details are discussed respectively by showing how they operates in both normal mode and disaster mode. Some fundamental evaluation results and statistics of the new functionalities are shown in the last section.

3.1 Overview

We have implemented two new features for Twimight: photos and off-line web view. As for photos, users can upload a photo along with a Tweet and post both of them on Twitter. In disaster mode, a photo can be attached to a Tweet and sent along with the Tweet to Bluetooth peers. Off-line web page is aimed to enable off-line view of web pages. This function will download web pages when the devices have Internet connection and when connection is lost, the content of web pages can be still viewed using a simple browser built in Twimight. The users can also share the web pages in disaster mode to other peers after enabling “Web Share Function”.

3.2 Normal Mode

In normal mode, the application communicates with Twitter server. Photos attached will be uploaded to Twitter server using the Twitter API along with the Tweet and posted online. The main User Interface for photo function consists of five buttons. The

“Photo” button is used to display or hide the photo panel as shown in Figure 3.1a. On the panel from left to right are “Gallery”, “Camera”, “Preview” and “Delete” buttons. Users can choose a photo for uploading either from the local image gallery or by taking a new photo using the camera. Once a photo is selected Figure 3.1b, users can take a preview of the photo or delete it if they don’t like the photo Figure 3.1c. After a Tweet is posted successfully along with a photo, the photo is displayed below the text when viewing the details of the Tweet.

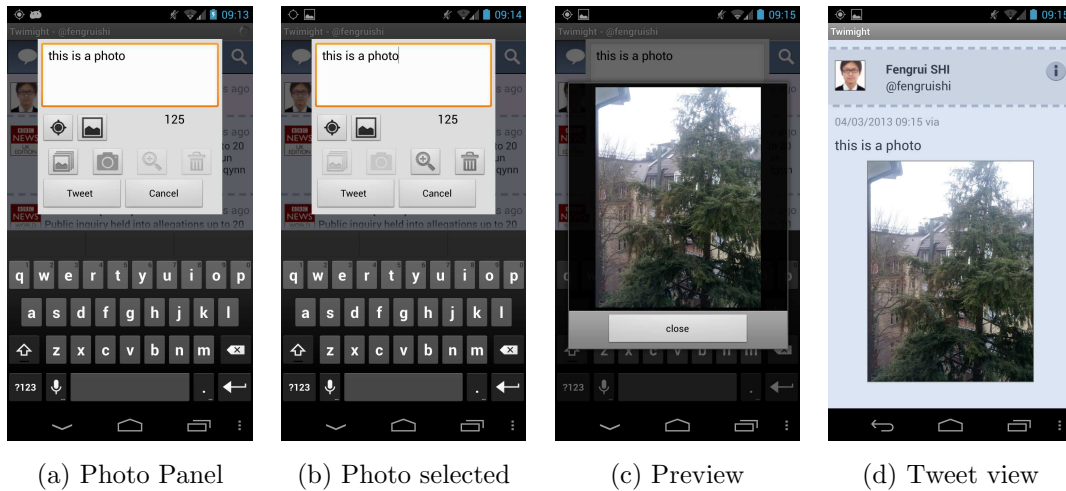
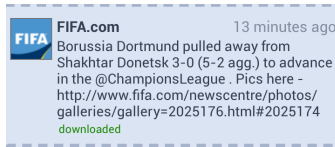
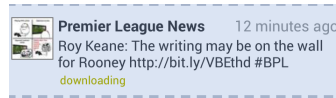


Figure 3.1: Post a Tweet with photo

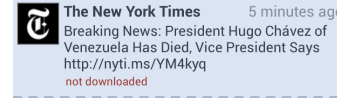
As for web pages, there are two cases in normal mode, namely with Internet connection and without connection. If devices are connected to the Internet, a built-in process of Twimight will download web pages associated with the links in the Tweet. The user interface shows indications of whether a web page has been successfully downloaded Figure 3.2a, in being downloaded Figure 3.2b or has not yet tried to download Figure 3.2c. Design decisions have been made separately to distinguish the situations when Wi-Fi connection is available and mobile data connection such as 3G is available. The idea is to let the user choose whether to download the web pages or not when only mobile data connection is available because mobile network is considered to be more expensive and slower than Wi-Fi network.



(a) Downloaded



(b) Downloading



(c) Undownloaded

3.2.1 Photo

There are several possible operations related with the photo function. When users write a new Tweet, they can choose whether to attach a photo to the Tweet or not, either by selecting a photo in the image gallery of the phone, or by capturing a new photo using the phone camera. The user can also preview the photo before posting the Tweet or delete the photo if the user is not satisfied with the current one and wants to re-take one.

When a photo is selected, either from gallery or camera, a temporary copy of the photo is stored in the “temp” folder in the SD card. When deleting the photo or cancelling the operation of writing a new Tweet, the temporary copy will be deleted as well. If the user confirms posting the Tweet with the photo, the temporary copy will be copied to a final location in SD card with folders corresponding to the user’s Twitter account ID (Figure 3.3).

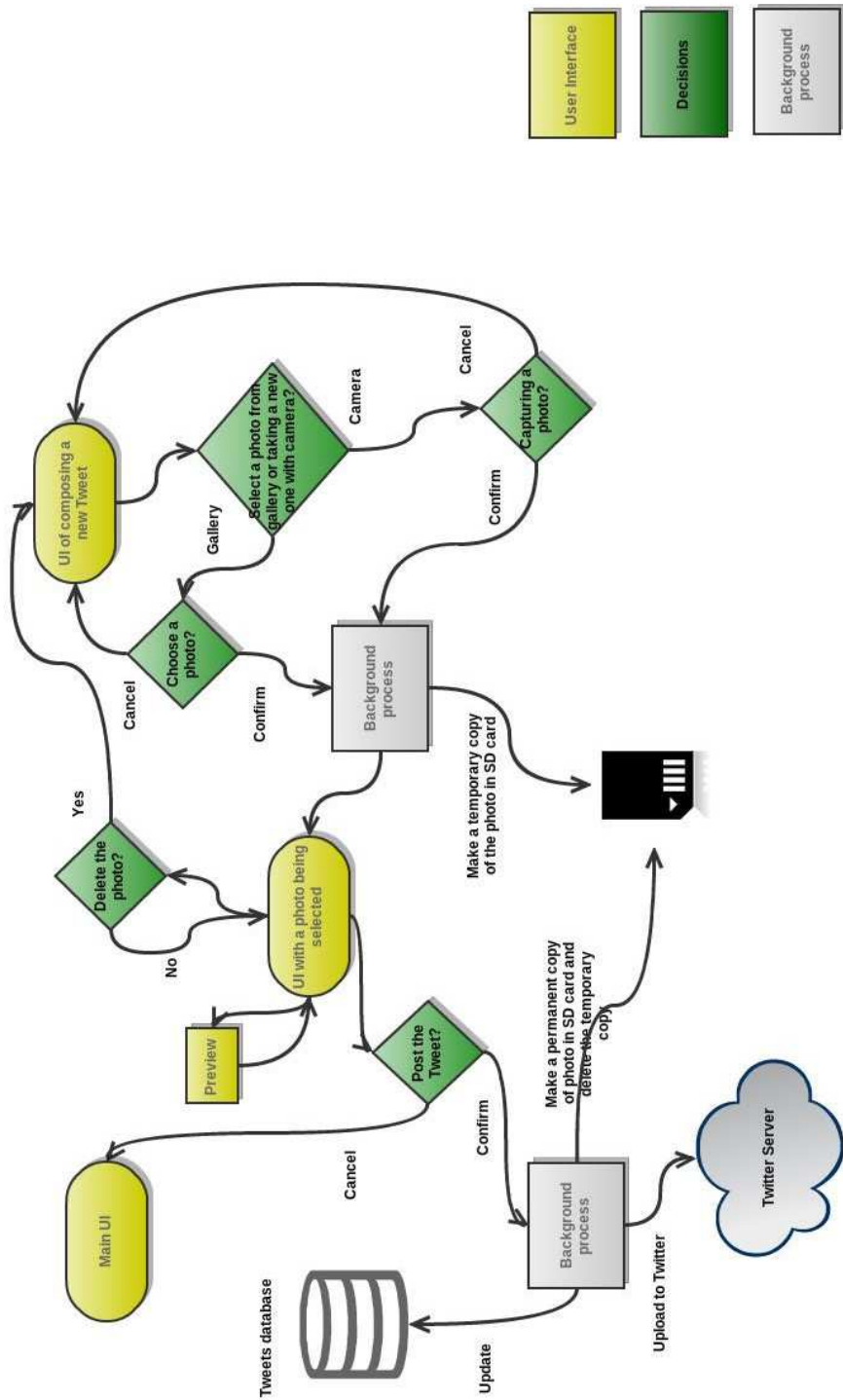


Figure 3.3: Flow chart of photo part

3.2.2 Off-line Web View

For off-line web view function, things get much more complicated compared to the photo part, because the application has to distinguish between different scenarios of network connectivity. Basically, there are three cases of network connection in normal mode of off-line web view function: Wi-Fi connection, 3G or mobile data connection and no connection. Different operations are used among three conditions. The objective that web pages can be both downloaded automatically when the so-called “Automatic Web Download” function is enabled and downloaded manually will make the above cases even more difficult. Details will be discussed later.

As for determining which pages are to be downloaded, a buffer for database has been used here. All web pages which are tagged as “can be downloaded” but “has not been downloaded” are put in this buffer. Each time downloading operation is performed, the application will get the 10 most recently inserted entries from this buffer, try downloading them and then update these entries based on download result. Once an entry is marked as “successfully downloaded”, it will be untagged from this buffer. For limiting the required storage space, this buffer will not contain any entries older than 24 hours, which is to say that even if some pages have not been downloaded yet, they will be discarded once 24 hours have passed since first receiving them.

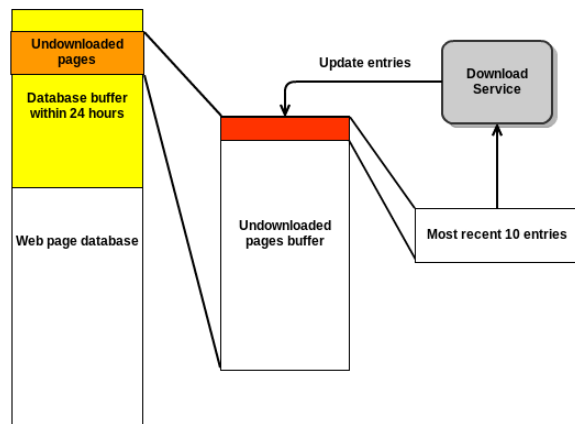


Figure 3.4: Database structure of web pages

We now explain the different operations for different network conditions. Automatic web download can only be valid when the device has Wi-Fi connections. This design is based on the fact that normally Wi-Fi will not consume on the data plan and users

don't have to pay more if they consume more as opposed to mobile data connection scenarios. Two cases may trigger the application to download web pages: 1) the alarm which periodically run some functions and 2) when user is scrolling up the list. Figure 3.5

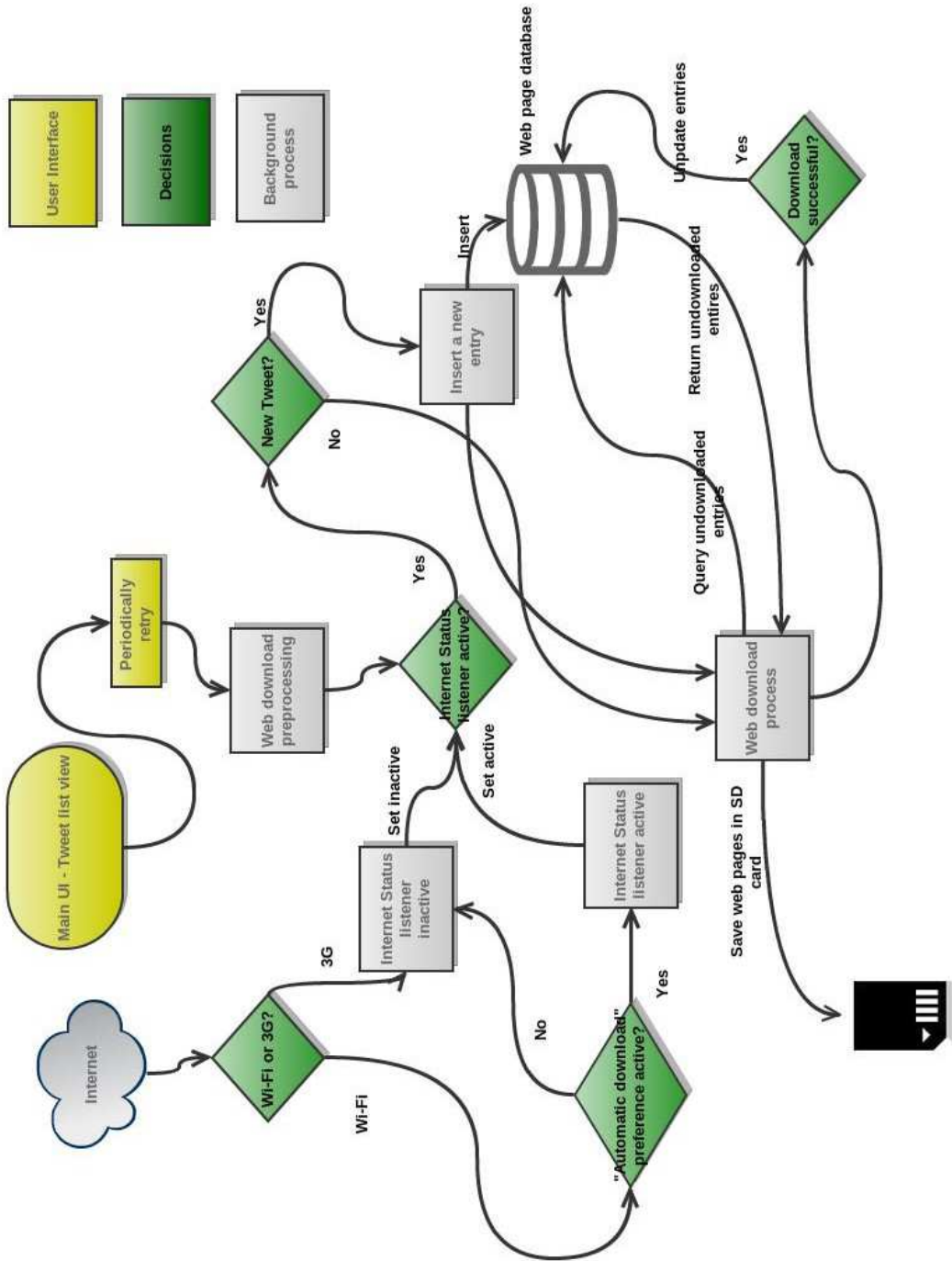
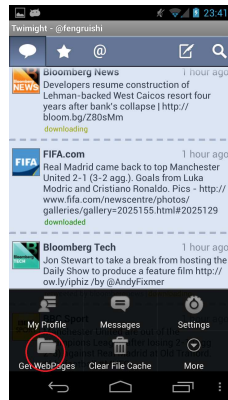
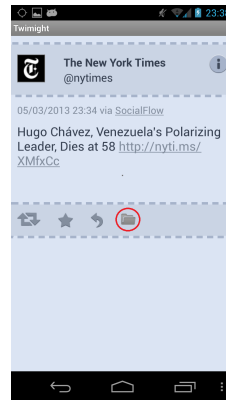


Figure 3.5: Flow chart of automatic web download

Under Wi-Fi connection situation, downloading operation can also be achieved manually. Two ways can accommodate this. In the main user interface of the application, a button called “Get Webpages” Figure 3.6a can be clicked to download the 10 most recently received web pages in the buffer as mentioned before, or user can click a similar button Figure 3.6b when viewing the details of a Tweet and download only the page for a single Tweet.



(a) Get Web pages
Button



(b) Get web page
for a single Tweet

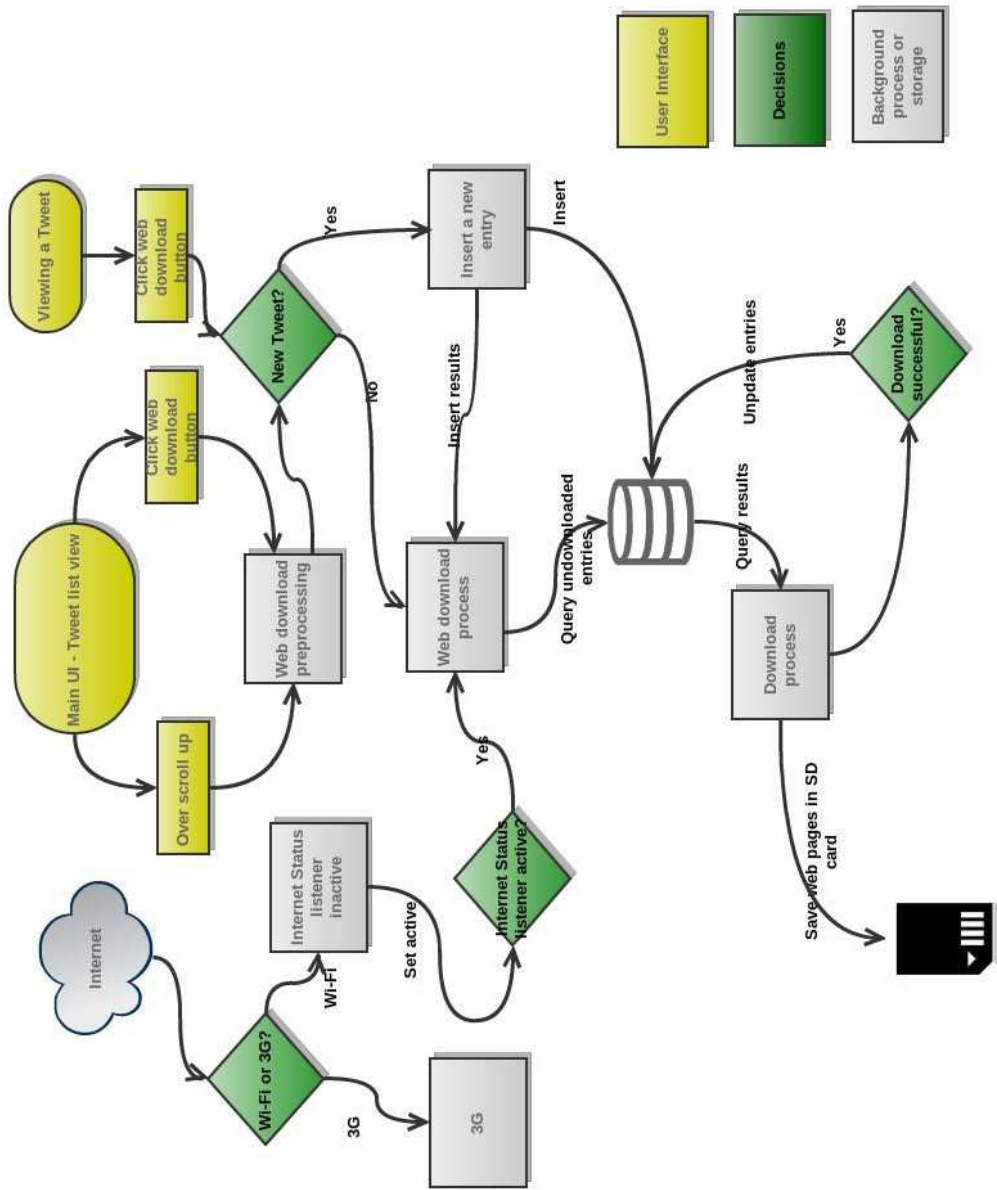


Figure 3.7: Flow chart of manual web download with Wi-Fi access

Manual download can also be performed when only mobile data connection is available. With mobile data connection, the data flow of data packets is usually confined to a certain amount, e.g. 2G per day. In order to avoid consuming large data flow without even consciously knowing why, automatic download cannot be performed here. However, considering the failure rate of downloading web pages, another mechanism has been added to allow automatically download pages tagged with “forced”, which is a case when user determines to download a page linked to a Tweet by clicking the downloading button for a specific Tweet when there is only mobile data connection available for the device. Figure 3.8

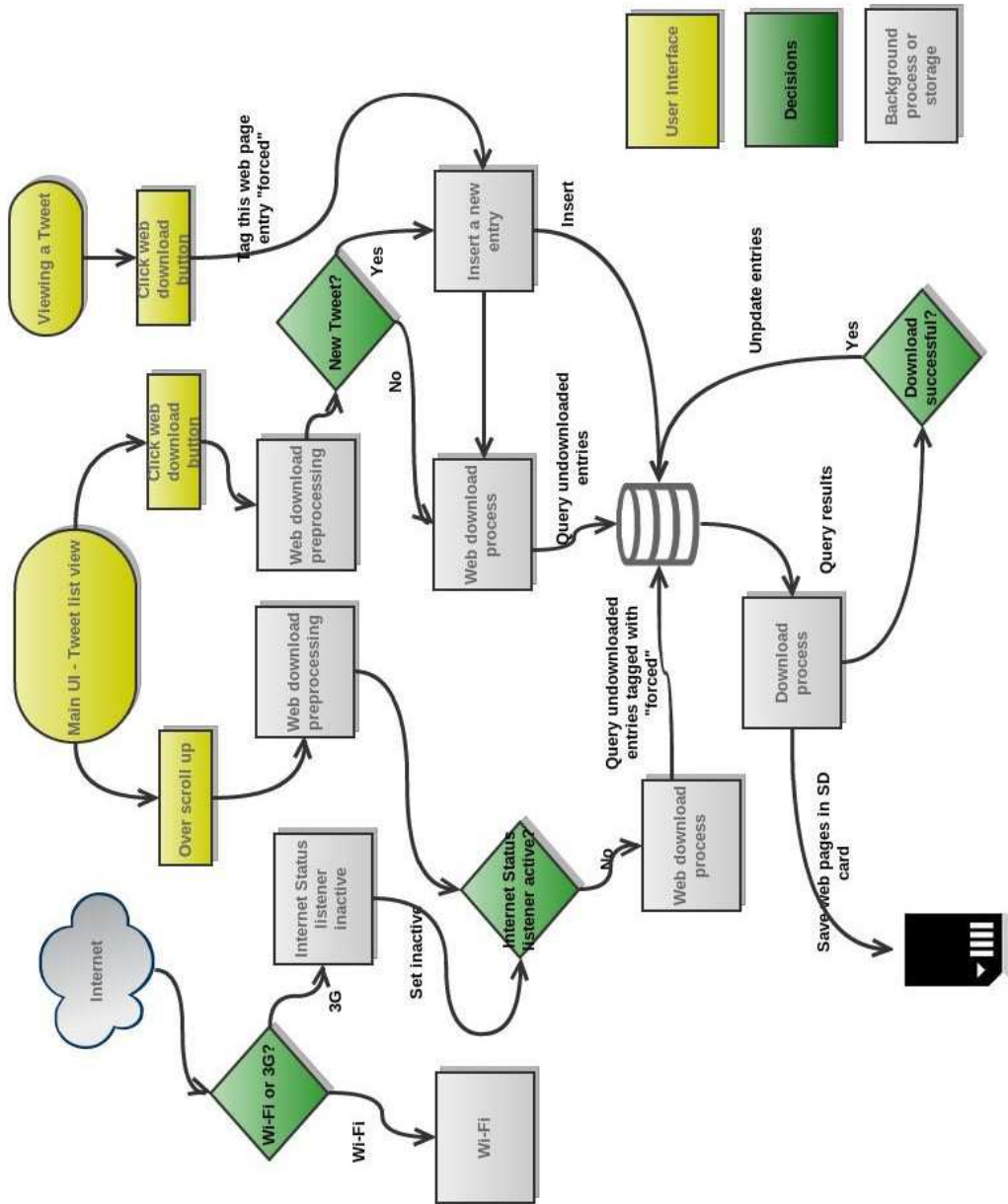


Figure 3.8: Flow chart of manual web download with mobile network (3G) access

3.3 Disaster Mode

Disaster mode operations for both functions are much simpler. Basically, two processes are executed in disaster mode, which are illustrated below. The device scans for nearby peers Figure 3.9a Figure3.9b and send required data to the newly found peers Figure 3.9c.

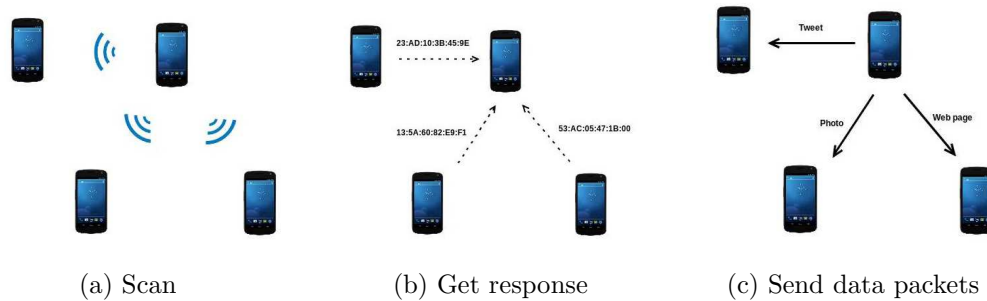


Photo transmission in disaster is by default enabled, while web page share function is not default. This distinction is made as a result of different scenarios of using these functions. Photo transmission is a useful function in disaster mode which can benefit the users in a variety of ways, such as finding a family member or spreading information about a place by taking photos. Web page share mode is useful in disasters in a sense that some connected users can re-tweet important Tweets as well as linked pages so as to spread the web page (e.g. a news article) to other unconnected users. This model is also implemented for the convenience of users who want to share a specific web page with their friends and we made such distinction to make sure that users know exactly what they need and what they are doing.

3.3.1 Data Structure

Database Tables In order to accommodate new functionalities, one more table is added and some entries of old tables are modified. The one newly added table “HTML Page” table is shown in Figure 3.10.

HTML Page	
Id	Integer
User ID	Text
Tweet ID	Text
Url	Text
Filename	Text
Downloaded	Integer
Forced	Integer
Tries	Integer

Figure 3.10: Web page database table

When constructing an entry, besides the primary key *Id* and *Filename* of the web page, other fields all have default values which are 0s. The *User Id* and *Tweet Id* are used to make sure that the reference of this entry to a file is unique globally in a user’s phone. *Downloaded* field is a binary field to indicate if the page has been downloaded or not. *Forced* is used when the device has only mobile data connection and if *Forced* = 1, the app will download the web page in any cases. *Tries* is used to indicate how many attempts have been made to download the web page. Because of unstable connections or unavailable CPU resources, the download can fail, however, in order not to overuse the phone resource, there is a limit number of tries which can be made before the entry is marked as “failed”. “failed” status is not a final state, because the user can reset this “*Tries*” counter to zero by manually re-download the web page marked as “failed” if the user regards the page as a “worthwhile to download” one.

Data packets Data packets for transmitting photos and web pages are illustrated in Figure 3.11. The photo packet has 4 fields.



Figure 3.11: Photo packet structure

Type is used for the receiver to identify which kind of information it receives and process the packets accordingly. *Filename* and the field “Media” in the disaster Tweet packet have same value so as to associate the photo to a specific Tweet. The Twitter account ID of the owner of this Tweet is sent as well. Both the facts that the value in “Filename”, which is the timestamp when the owner of this Tweet creates the Tweet and the photo, is unique in the Tweet owner’s phone and owner’s Twitter account ID is globally unique make sure that there are no collisions of naming issues in receiver’s phone. The last field is the payload of photo file which is a byte array.

The Data packet format of web pages has 6 fields as shown in Figure 3.12. Besides *Type*, *Filename* and *User ID*, which have same functions as in photo packet, *URL* and *Tweet ID* fields are used to associate this web page entry to a specific Tweet. Different from a photo packet, a single Tweet could have several links and therefore, only by combining both *URL* of the link and the ID of the Tweet, one can tell if a page belongs to this Tweet or not. Of course, if all links to a Tweet are the same, *URL* and *Tweet ID* cannot distinguish them but it does not matter in a sense that proper content of the web page can still be fetched anyway.

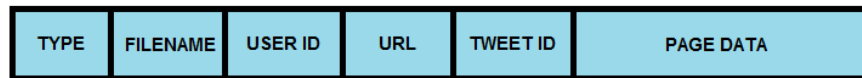
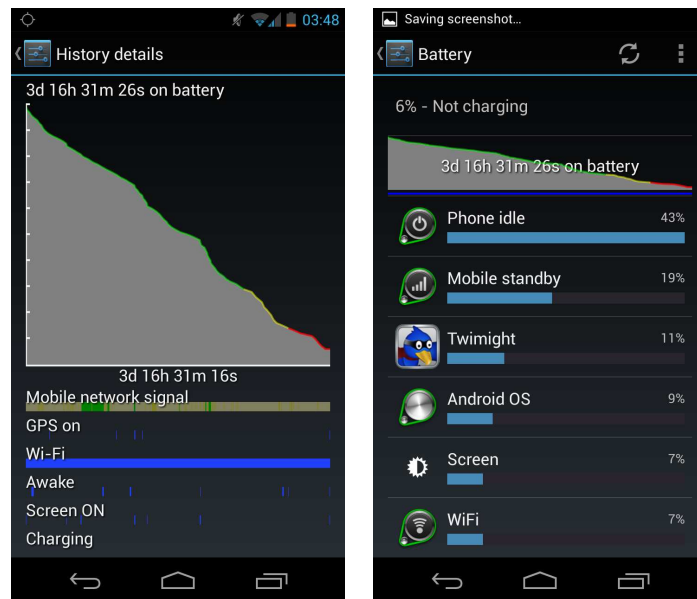


Figure 3.12: Web page packet structure

3.4 Evaluation

The energy consumption in normal mode is measured using Sumsung Galaxy Nexus 2 phone. Wi-Fi connection is always available during the measurement and the automatic web download function of Twimight is enabled. According to the result, the phone can still achieve long stand-by time for more than 3 days.



(a) Battery energy consumption (b) Energy consumption of different tasks

Figure 3.13: Energy consumption in normal mode

Statistics about web pages downloaded is collected using a Twitter account which follows users such as BBC News, The New York Times, The Economist, TIME.com, FIFA.com, Premier League News, TechCrunch, The Washington Post, etc. These users have common characteristics that 90 % of Tweets posted by them have links. Moreover, the size and layout of web pages from the links of different users vary largely, which can prove the adaptivity of the function to different kinds of web pages.

Table 3.1: Statistics about web pages

Stats	Numbers
# of total web pages	124
% of web pages download failed	7.3%
average # of tries	3.5
average size of web pages	1028050 bytes
maximum web page size	4436556 bytes
minimum web page size	42048 bytes

According to the results, the web pages download works pretty stable with small fail

rate and number of tries. Besides, web pages are normally large file with average size of 1MB and maximum size 4MB. This could justify the reason why we design storage buffers to prevent over-exhausting phone resources and storage spaces.

4

Adaptive Scanning

We propose an adaptive Bluetooth scanning algorithm and compare it to the existing scanning strategy as well as to state-of-art linear adaptive scanning. The performance of these two algorithms are compared by simulating these two scanning algorithms based on three mobility traces [16] [17] [18]. The results show that, in all the three mobility cases, the nonlinear algorithm proposed by us clearly outperforms the existing linear one.

4.1 Performance metrics

Accuracy: The overall accuracy of scanning is the ratio of two numbers within a time frame, i.e. “number of interesting events the device can detect via scanning” over “total interesting events happened within the scanning range of the device”. Detected events are the events noticed by the device, which means the phone performs scanning when a new event has happened but not ended yet. For example, there are 2 peers around the user’s device currently, which is an event by definition ($\#$ of total events = 1), if the user scans at the moment, this event is called “observed” by the user and the number of observed events = 1. However, if the user does not scan after a new event happens, for instance, one more peer is added around the user’s device. The number of total events will become 2 and the observed events is still 1 unless the user’s device observes current event with 3 peers before further changes.

Energy: Energy consumption in this work is represented by a ratio between “the number of scans the device has performed” and “maximum number of scans” device

could perform. It is an evaluation of energy saving of a scanning algorithm because less scans the device performs, less energy it consumes. For example, if the device scans with a constant time interval 1 minutes, then in 10 minutes' time, the maximum number of scans is 10, however, as adaptive scanning algorithm is used, it is possible that the device decides not to scan at some time instance and the total number of scans performed could be lower than 10, e.g. 5. The value $1 - \frac{5}{10}$ is an indication of how much scanning energy has been saved.

4.2 Existing Model

At the moment, in disaster mode operation, Twimight has a very simple Bluetooth scanning algorithm, which simply scans every 2 minutes. However, a data set collected using 150 Nokia N95 phones (Figure 4.1) shows that, 41% of scans will not see any peers around the scanning device [15].

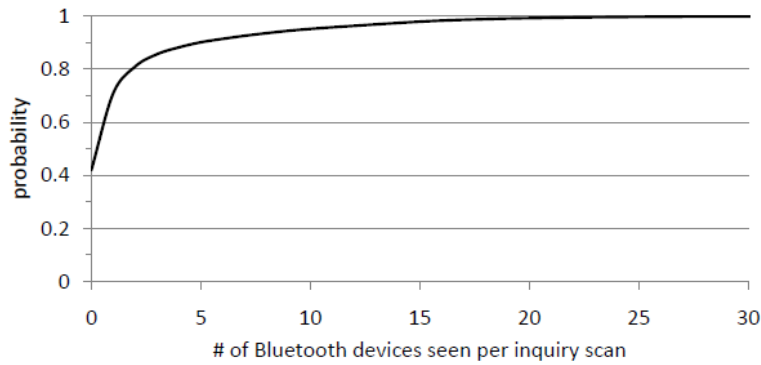


Figure 4.1: CDF of the number of Bluetooth devices seen in periodic scans from 150 Nokia N95 mobile phones. This data covers five months in 2008 [15]

In this case, scanning periodically all the time could waste a large amount of energy, while on the other hand, not scanning will lower the accuracy of the peer discovering process because it could miss a large number of information. There is clearly a trade-off between energy and accuracy. As a result, a scanning algorithm, which can lower the energy consumption and at the same time, does not lose too much accuracy is the ultimate goal of this algorithm design.

4.3 Adaptive Scanning

4.3.1 Probabilistic scanning

Probabilistic scanning is an adaptive scanning method which saves some scanning actions by scanning with a probability. Each time the device wants to scan, it randomly choose a number between 0 and 1 to see if the number is bigger than the probability. If the result is yes (random number is smaller than the scanning probability), then it performs scan, while it chooses to not scan if answer is no. Compared to the method which always triggers scans with a fixed time interval, if the scanning probability for each scan is set to be 0.5, then statistically, only 50% of scans will be performed and 50% of scanning energy of the device will be saved. Some terms are defined below in order to better present the algorithms.

p_{max}: *p_{max}* is the maximum probability a device will scan at a certain time points. It is set to 1 in normal cases.

p_{min}: *p_{min}* is the minimum probability a device will scan at a certain time points. This value is adjusted according to the algorithm. Usually it is a small value like 0.1 or 0.2.

Δp : Δp is the value to increase or decrease scanning probability. New probability for next scan is the current probability plus or minus Δp .

As mentioned before, there is always a trade-off between energy saving and accuracy loss. In order to better solve this issue of Bluetooth scanning, we proposed an adaptive scanning technique in this thesis and compared its performance, i.e. energy saving and accuracy, with an existing adaptive scanning model.

4.3.2 Linear Model

Before us, some adaptive scanning algorithms have already been proposed and they share a common characteristic: linear model which change the scanning probability or scanning interval linearly. For example, in social sensing applications, an adaptive scanning algorithm, which adjust scanning interval between two consecutive scans based on scan results, is used to detect the social context of users [15]. Besides, algorithm which adjusts scanning probability of each scan rather than scanning interval has been proposed to perform social sensing [19].

However, both algorithms make decisions by only considering results of current scan and then increasing scanning probability or scanning interval if result is positive and decreasing the probability if result is negative and the increase or decrease is a linear function of the scanning results. Therefore, they fail to fully utilize the information obtained by previous scans. Besides, as these models have different applications compared with Twimight, these models may not well directly adopted to Twimight. However, based on the idea of the above models, one can easily devise an adaptive scanning algorithm which adjusts the scanning probability by the information collected from each scan which can be suitable to the application for Twimight. The basic idea of this algorithm is, when the scanning result, which is a set of peers found, is different between two consecutive scans, the scanning probability will be increased, otherwise, it will be decreased. The amount of increase or decrease is a fixed value named Δp and is within the range $[0, 1]$. The probability can therefore alter from p_{min} to p_{max} adaptively.

Algorithm 1 Adaptive Scan

$p_{max} \geq p_i \geq p_{min}$ $\{p_i : \text{probability to perform } i_{th} \text{ scan, } i = (0 \dots n)\}$
 $\Delta p > 0$ $\{\Delta p : \text{change of the probability in } i_{th} \text{ scan, } i = (0 \dots n)\}$
 L_i $\{L_i : \text{list of devices found during } i_{th} \text{ scan, } i = (0 \dots n)\}$
 t_i : time stamp of i_{th} scan, $i = (0 \dots n)$
 Input : $p_0 \leftarrow 0.5$, $p_{min} \leftarrow 0.1$, $p_{max} \leftarrow 1$, $L_0 \leftarrow \text{no device}$, $\Delta p = 0.1$
 Start : $i \leftarrow 0$
for $i \leq n$ **do**
 determine with a probability of p_i at t_i if perform scan or not
 if scan at t_i **then**
 $L_i \leftarrow$ list of devices found during this scan
 compare two lists L_i and L_{i-1}
 if $L_i \neq L_{i-1}$ **then**
 $p_{i+1} = \min\{p_i + \Delta p, p_{max}\}$
 else
 $p_{i+1} = \max\{p_i - \Delta p, p_{min}\}$
 end if
 else
 $p_{i+1} = p_i$, $L_i = L_{i-1}$
 end if
 $i \leftarrow i + 1$
end for

4.3.3 Proposed Model

Compared to the fore-mentioned linear model, we propose a non-linear model which not only make decisions given the information from past scans, but also try to make reasonable predictions based on these information. Human mobility follows some pattern and various research works have tried to discover the pattern [20] [21]. These patterns could give some hints on how people move around and along with the information collected in previous scans, it is easier to make correct decisions on the next scans. However, as the mobility patterns are very complex and not each to manipulate. We made our own simplified pattern and designed the following algorithm based on that.

Algorithm 2 Adaptive Scan

$p_{max} \geq p_i \geq p_{min}$ $\{p_i : \text{probability to perform } i_{th} \text{ scan, } i = (0 \dots n)\}$
 $\Delta p > 0$ $\{\Delta p : \text{change of the probability in } i_{th} \text{ scan, } i = (0 \dots n)\}$
 L_i $\{L_i : \text{list of devices found during } i_{th} \text{ scan, } i = (0 \dots n)\}$
 $N_i \geq 0$ $\{N_i : \# \text{ of devices found during } i_{th} \text{ scan, } i = (0 \dots n)\}$
 t_i : time stamp of i_{th} scan, $i = (0 \dots n)$
Input : $p_0 \leftarrow 0.5$, $p_{min} \leftarrow 0.1$, $p_{max} \leftarrow 1$, $L_0 \leftarrow \text{no device}$
Start : $i \leftarrow 0$
for $i \leq n$ **do**
 determine with a probability of p_i at t_i if perform scan or not
 if scan at t_i **then**
 $L_i \leftarrow$ list of devices found during this scan
 N_i^{new} : # of new devices in i_{th} scan
 N_i^{old} : # of old devices remains in i_{th} scan
 $N_i = N_i^{new} + N_i^{old}$
 if $N_i \neq 0$ **then**
 $\Delta p = (p_{max} - p_i) \times \left\{ \left(\frac{N_i^{new}}{N_i} \right) + \left(1 - \frac{N_i^{old}}{N_{i-1}} \right) \right\} \left\{ \left(\frac{N_i^{new}}{N_i} \right) : \text{degree of new devices} \right.$
 increase, $\left. \left(1 - \frac{N_i^{old}}{N_{i-1}} \right) : \text{degree of old devices decrease} \right\}$
 $p_{i+1} = \min\{p_i + \Delta p, p_{max}\}$
 else
 $p_{i+1} = \max\{p_i/2, p_{min}\}$
 end if
 else
 $p_{i+1} = p_i$, $L_i = L_{i-1}$
 end if
 $i \leftarrow i + 1$
end for

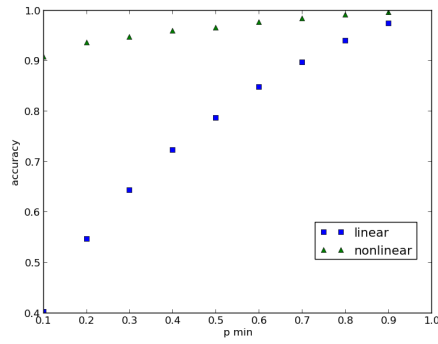
The mobility pattern we are using here is simple. It calculates how much the results of two consecutive scans differs. In the peer list found in every scan, there are three kinds of peers: those already appeared in the list for last scan and appears in the current list again, those appeared in the last list but not in the current list and those new in the current list. Given these numbers, we can find out how much the list has changed and thus change the scanning probability based on that.

4.4 Results and Analysis

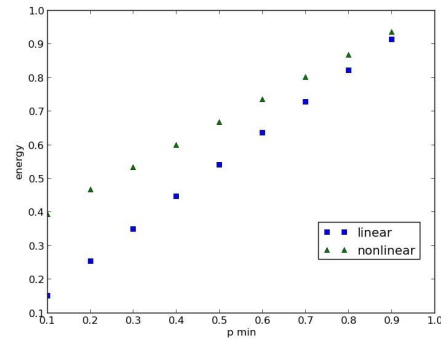
In this section, simulation results are presented and analysed. Three mobility traces are simulated with a simple simulator designed with Python by ourselves. The simulator itself performs the following tasks: read the trace data from the input files, perform the algorithms on the data and write the results in the output files. All the files are simple text files and the simulator is installed on a Linux workstation.

4.4.1 Simulation Results

Simulation results of both fore-mentioned algorithms are shown below. Figures of the results are shown first. For each figure, the performance (accuracy and energy) of two algorithms are compared over different p_{min} . Then several tables with more detailed numbers are presented. Each table illustrates relevant data given a certain scan accuracy. Energy saving is the portion of total energy saved during the simulation using respective algorithm. For example, in the third row of table 4.1, first column is the trace name. 48% means that the linear model can save 48% scanning energy with an accuracy of 80% and 60% is the energy saved by nonlinear algorithm. Improvement is the difference between two models which is $60\% - 48\% = 12\%$. The last column is the minimum scanning probability used in order to have an accuracy of 80%.

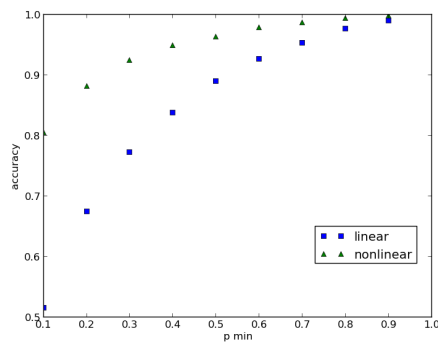


(a) accuracy

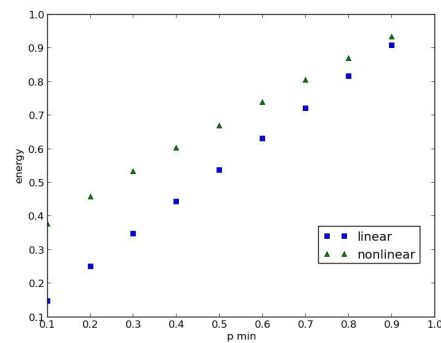


(b) energy

Figure 4.2: Simulated energy consumption based on ETH trace, w.r.t different initial probabilities with 1 min scan interval

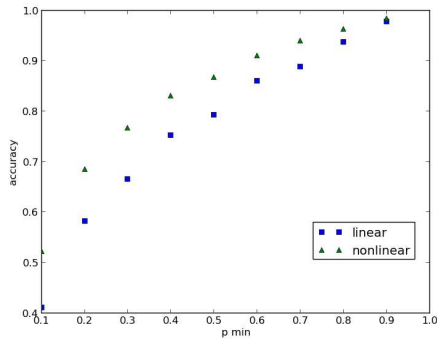


(a) accuracy

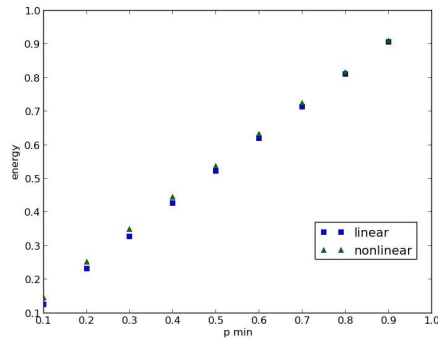


(b) energy

Figure 4.3: Simulated energy consumption based on Haggie trace, w.r.t different initial probabilities with 1 min scan interval



(a) accuracy



(b) energy

Figure 4.4: Simulated energy consumption based on MIT trace, w.r.t different initial probabilities with 30 min scan interval

Table 4.1: Energy saving with 80% accuracy

Traces	linear	nonlinear	improvement	p_{min}
ETH[16]	50%	N/A	N/A	0.1
Haggle[17]	60%	63%	2%	0.1
MIT[18]	48%	60%	12%	0.35

Table 4.2: Energy saving with 85% accuracy

Traces	linear	nonlinear	improvement	p_{min}
ETH	40%	N/A	N/A	0.1
Haggle	50%	60%	10%	0.15
MIT	37%	52%	15%	0.45

Table 4.3: Energy saving with 90% accuracy

Traces	linear	nonlinear	improvement	p_{min}
ETH	30%	60%	30%	0.1
Haggle	42%	52%	10%	0.2
MIT	26%	43%	17%	0.55

Table 4.4: Energy saving with 95% accuracy

Traces	linear	nonlinear	improvement	p_{min}
ETH	20%	40%	20%	0.3
Haggle	28%	40%	12%	0.4
MIT	18%	27%	9%	0.7

4.4.2 Analysis

The results clearly show that the proposed non-linear model well outperforms the linear model. The amount of energy saved varies from 10% to 30%. The reason for this improvement is that the nonlinear algorithm better explores the pattern of peer discovering and predicts future status based on that. According to the statistics given by [15], 41% of scans find no peers. Therefore, the status that there are some peers around is an unstable status compared to the status with no peers around. Based on this fact, the nonlinear algorithm increases the scanning probability more aggressively when the status changes from stable to unstable and decreases more aggressively when from unstable to stable, thus saving more unnecessary scanning operations. Besides, during unstable status, the scanning probability remains in a relatively high value, which can ensure not to miss many events because there can be more changes when in unstable status.

5

Conclusions and Outlook

This chapter concludes this thesis by giving a short overview of the work and the achieved results and presents some ideas for future work and following projects.

5.1 Conclusions

In this thesis, new functionalities are implemented to solve problems in old versions of Twimight and improve off-line operation of the application. With the photo function, users can upload and view photos and with off-line web page function, web pages can be downloaded and viewed off-line. Both functions are also supported in disaster mode, which means both photo files and web page files can be transmitted opportunistically. The design principle and structures are discussed and presented. Another contribution of this thesis is the proposal of an adaptive scanning algorithm for Twimight. By simulating the algorithms using several mobility traces and comparing the performance of these two algorithms, although the linear algorithm has already achieved energy saving with 20% to 30%, the nonlinear algorithm is able to add another 10% to 30% improvement (therefore 40% to 60% in total). By analysing the pattern of peer discovery, it is shown that this algorithm better utilizes the pattern and therefore finds a good compromise between energy and accuracy of Bluetooth scanning.

5.2 Outlook

There are several interesting possibilities of following projects and further research based on the presented work. For example:

Build a new HTML parser: As mentioned above, the web page function is achieved using WebView class and its method saveWebArchive() to download the pages. However, saveWebArchive() does not perform constantly well when downloading a large number of web pages in a short period of time. Therefore, to download the web page in a more primitive way and build a parser to extract the information will be likely to improve the performance of this function.

Mobility explanation of adaptive scanning algorithm: Although the adaptive scanning algorithm proposed in this thesis performs well, the theories for this good performance have not been well established. One potential explanation can be mobility pattern and therefore, if explanation can be found, the theoretical foundation of this algorithm can be justified and further improvement can be possibly made.

Putting contacts into context: Based on some information such as location and time, the algorithm can adapt accordingly to grasp this kind of information. It is possible that the algorithm can be further improved by considering the context of contacts.

Real life experiments: Performance of proposed algorithm is only simulated using mobility traces, however, neither the energy saving is quantified nor the algorithm is tested in real-life cases. Further experiments and tests should be carried on.

Appendix A

User Know-how

This chapter explains basic requirements of installing new version of Twimight as well as some instructions on how to use the functions properly.

A.1 Requirements

The application is aimed for Android phones. The basic requirements for the phone are:

- Android phones with API level at least 3.1(Honeycomb) in order to properly download web pages
- Support Bluetooth
- 3G or Wi-Fi during normal mode
- with an SD card or some emulated SD card storage
- 100 MB free space on SD card in order to store web pages off-line

A.2 Instructions

A.2.1 Normal Mode

After installation, the default status of automatic web download is false. Therefore, if user wants to enable automatic web download, the function has to be turned on manually in Menu/Settings/Automatic Web Download

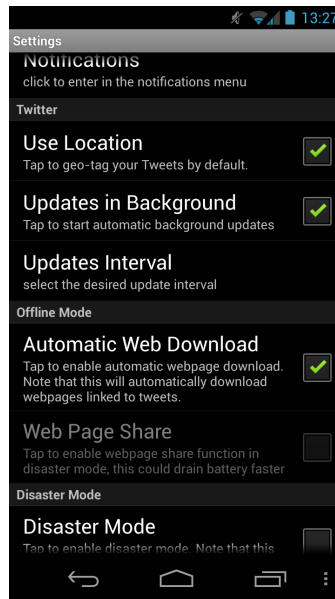


Figure A.1: Turning on automatic web download

For downloading web pages, either drag down the Tweet list or press the button Menu/Get Webpages.

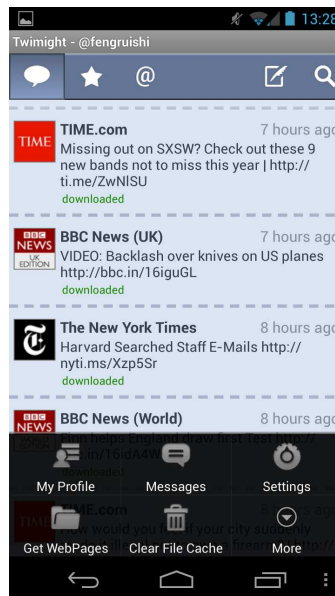


Figure A.2: Manual web download

For downloading web page for a single Tweet, click on the Tweet and press the button

below

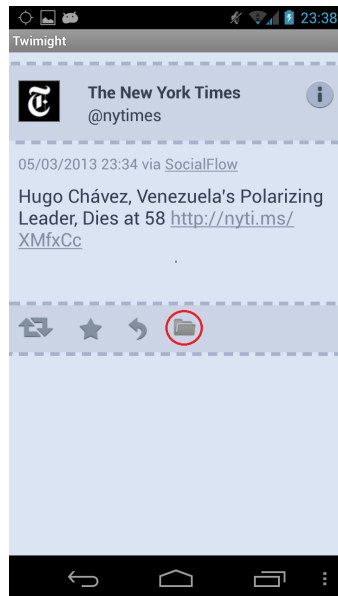


Figure A.3: Download web page for a single Tweet

Note that if 3G instead of Wi-Fi is available, automatic web download function will not be activated. However, user can still use the above two methods to download a web page. For the second case, the web page for that specific Tweet will be tagged as “forced” and can therefore be automatically downloaded even with only 3G connection.

A.2.2 Disaster Mode

To enable disaster mode, go to Menu/Settings/Disaster Mode

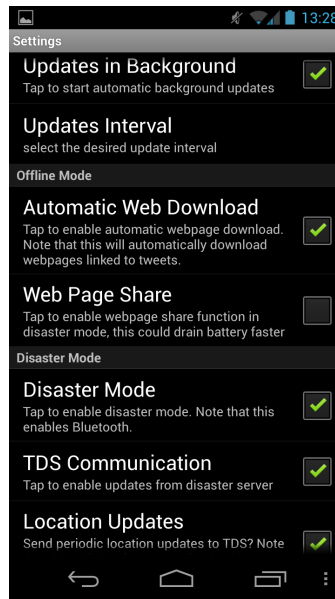


Figure A.4: Enable disaster mode

If web share function is to be enabled, one has to enable disaster mode first and then can choose if or not to enable web share function which will drain battery faster once enabled.

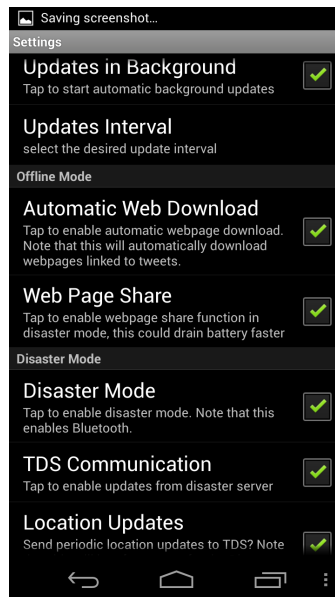
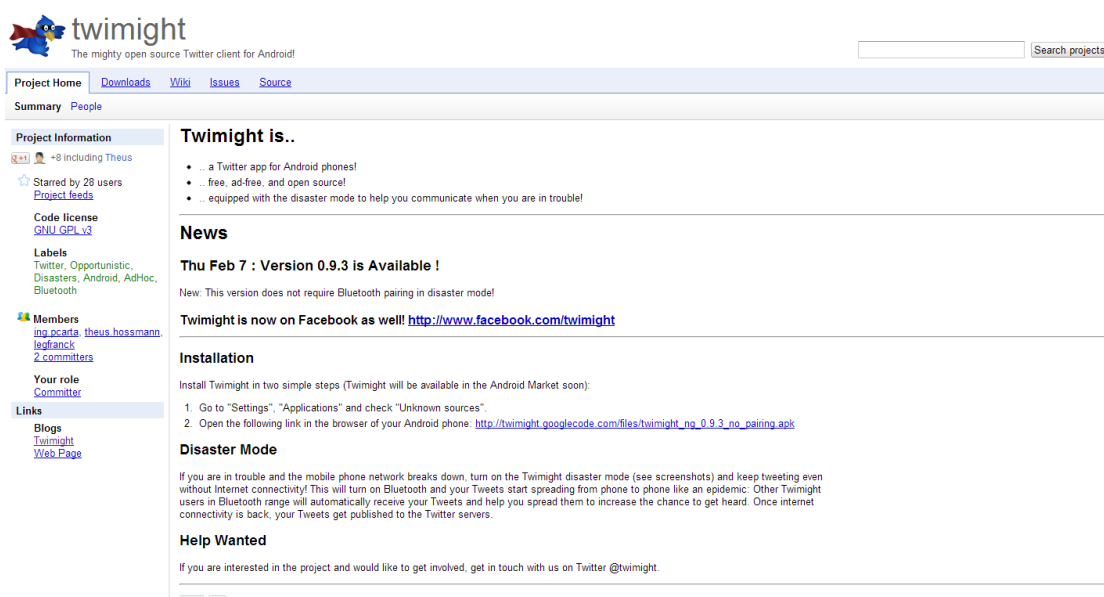


Figure A.5: Enable web share

Appendix B

Developer Know-how

Please check on-line repository <http://code.google.com/p/twimight/> and refer to corresponding instructions in Appendix B of the thesis *Implementation of a Disaster Mode to Maintain Twitter Communications in Times of Network Outages*[6].



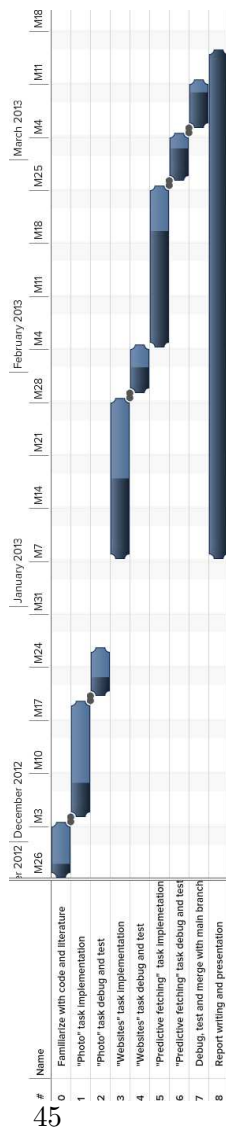
The screenshot shows the project page for Twimight on Code.google.com. The page has a blue header with the project name and a search bar. Below the header, there are navigation tabs for Project Home, Downloads, Wiki, Issues, and Source. The main content area is divided into several sections:

- Project Information:** Includes a star icon, a note that 8 users (including Theus) have starred the project, and a link to project feeds.
- Code license:** GNU GPL v3.
- Labels:** Twitter, Opportunistic, Disasters, Android, AdHoc, Bluetooth.
- Members:** Lists users including theus, hossmann, legfranc, and 2 committers.
- Your role:** Committer.
- Links:** Blogs, Twimight, Web Page.
- Twimight is..:** A list of features: a Twitter app for Android phones, free, ad-free, and open source, and equipped with a disaster mode.
- News:** A section titled "Thu Feb 7 : Version 0.9.3 is Available !" with a note that this version does not require Bluetooth pairing in disaster mode.
- Installation:** A section titled "Twimight is now on Facebook as well!" with a link to the Facebook page. It includes instructions to install Twimight in two simple steps.
- Disaster Mode:** A section explaining that if the mobile phone network breaks down, users can turn on the Twimight disaster mode and keep tweeting even without internet connectivity. Other Twimight users in Bluetooth range will automatically receive the tweets and help spread them.
- Help Wanted:** A section asking if anyone is interested in the project and would like to get involved, with a link to get in touch on Twitter.

Figure B.1: Online repository

Appendix C

Time Schedule



45

Figure C.1: Time schedule(planned)

Bibliography

- [1] *Twimight codes*:
<http://code.google.com/p/twimight/>
- [2] Theus Hossmann, Paolo Carta, Dominik Schatzmann, Franck Legendre, Per Gunningberg and Christian Rohner, *CoNext Special Workshop on the Internet and Disasters*, ACM, Tokyo, Japan, December 2011.
- [3] Theus Hossmann, Franck Legendre, Paolo Carta, Per Gunningberg and Christian Rohner. *Twitter in Disaster Mode: Opportunistic Communication and Distribution of Sensor Data in Emergencies*, ExtremeCom2011, September 26-30, 2011.
- [4] *Display Webarchive codes*: <http://stackoverflow.com/questions/12872043/android-webview-display-webarchive>
- [5] *Volume of internet traffic in Japan*:
<http://www.comscoredatamine.com/2011/03/mobile-internet-traffic-in-japan-spikes-in-response-to-the-tsunami>.
- [6] Paolo Carta, *Implementation of a Disaster Mode to Maintain Twitter Communications in Times of Network Outages*, Master Thesis, ETH Zurich, 2011
- [7] *Twitter: Tweets per day*, <http://www.forbes.com/sites/davefeinleib>
- [8] L. Rao, *Twitter Seeing 90 Million Tweets Per Day, 25 Percent Contain Links*, TechCrunch, Sept. 14, 2010.
- [9] *Average Web Page Size Triples Since 2008*:
<http://www.websiteoptimization.com/speed/tweak/average-web-page/>
- [10] *WebView*:
<http://developer.android.com/reference/android/webkit/WebView.html>

- [11] *saveWebArchive()*:
[http://developer.android.com/reference/android/webkit/WebView.html#saveWebArchive\(java.lang.String\)](http://developer.android.com/reference/android/webkit/WebView.html#saveWebArchive(java.lang.String))
- [12] *Android platform versions*:
<http://developer.android.com/about/dashboards/index.html>
- [13] *10 Best Twitter Clients for Android*:
<http://www.online-convert.com/result/3c05aef5ace48b361ff3d87b0135618d>
- [14] Kevin Fall, *A delay-tolerant network architecture for challenged internets*, In ACM SIGCOMM Conference, pages 27-34.
- [15] Banerjee N, Agarwal S, Bahl P, *Virtual compass: relative positioning to sense mobile social interactions*, Pervasive Computing, 2010: 1-21.
- [16] Lenders V, Wagner J, May M. *Measurements from an 802.11 b mobile ad hoc network*, Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks. IEEE Computer Society, 2006: 519-524.
- [17] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot and Augustin Chaintreau, *CRAWDAD trace cambridge/haggle/imote/infocom2006 (v. 2009-05-29)*, <http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/infocom2006>
- [18] Eagle, N., Pentland, A., *Reality mining: sensing complex social systems*, Pers. Ubiq. Comput. 10, 255268, 2006.
- [19] Rachuri K K, Mascolo C, Musolesi M, *Sociablesense: exploring the trade-offs of adaptive sampling and computation offloading for social sensing*, Proceedings of the 17th annual international conference on Mobile computing and networking. ACM, 2011: 73-84.
- [20] Thomas Karagiannis, Jean-Yves Le Boudec, and Milan Vojnovi, *Power law and exponential decay of inter contact times between mobile devices*, Proceedings of the 13th annual ACM international conference on Mobile computing and networking (MobiCom '07). ACM, New York, NY, USA, 183-194, 2007
- [21] Theus Hossmann, Thrasyvoulos Spyropoulos, and Franck Legendre, *Putting contacts into context: mobility modeling beyond inter-contact times*, Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '11). ACM, New York, NY, USA, 2011