

國立臺灣大學電機資訊學院資訊網路與多媒體研究所

碩士論文

Department of Graduate Institute of Networking and Multimedia

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis



即時通訊軟體在 Android 平台上的省電策略

Power-Saving Strategies of Mobile Instant-Messaging

Application on the Android Platform

陳政柏

Cheng-Po Chen

指導教授：蘇雅韻博士

Advisor: Ya-Yunn Su, Ph.D.

中華民國 103 年 7 月

July, 2014



## 摘要

隨著智慧型手機的普及，其應用軟體也隨之快速增長，然而即時通訊軟體也為其一。本篇論文旨在探討搭載安卓系統的智慧型手機在安裝多個即時通訊軟體下，對其電量帶來過量耗損的情形，並提出一套程式設計架構，以解決此一現象。

### 關鍵字

行動計算，即時通訊，安卓系統，耗電量，保持活動



# Abstract

With the development of mobile devices, more and more applications has been added to its application store by developers and installed by users in their own devices. However, one main concern for mobile users is that whether an application drains a device's battery. Therefore, improving battery life always is a popular issue in recent years because of limited battery. One popular application on the market is a mobile instant messaging (MIM) application. MIM applications allow mobile users owning devices with different platforms (e.g., Android, iOS) to communicate through text, voice, or images over Wi-Fi or cellular networks. This thesis targets to reduce power consumption on MIM applications and examines the application and energy behavior of users who install multiple MIM applications on their phones. We designed and implemented Consultant, a light-weight platform to optimize energy consumption for MIM applications. Consultant analyzes the data provided by different MIM applications and help applications to better use phone resource. In our evaluation, we replayed the synthetic real trace collected from the users and experimental results showed that Consultant can save 52% MIM energy with 3 installed MIM applications during a day.

## KEYWORDS

Mobile Computing, Instant-Messaging, Android, Power Consumption, Keep-alive



# Contents

摘要	i
Abstract	ii
<b>1 Introduction</b>	<b>1</b>
<b>2 Problem</b>	<b>3</b>
2.1 Mobile Instant Messaging . . . . .	3
2.2 Radio Power in 3G Cellular Network . . . . .	4
<b>3 Mobile Instant Messaging Application Behavior</b>	<b>7</b>
3.1 Measurement and Detecting . . . . .	7
3.2 Evaluate Detecting Flow . . . . .	10
3.3 Measurement Results . . . . .	10
3.4 Instant or Non-instant . . . . .	11
<b>4 Power-Saving Strategies</b>	<b>15</b>
4.1 Keep-alive Mechanism . . . . .	15
4.2 Message-Delayed Delivery . . . . .	16
4.3 APIs . . . . .	17
<b>5 Implementation</b>	<b>20</b>
5.1 Client-side . . . . .	20
5.2 Server-side . . . . .	20

<b>6</b>	<b>Evaluation</b>	<b>22</b>
6.1	Methodology . . . . .	22
6.2	Consultant for Keep-alive . . . . .	23
6.3	Trace Replay . . . . .	24
<b>7</b>	<b>Related Work</b>	<b>29</b>
<b>8</b>	<b>Conclusion</b>	<b>31</b>
	<b>Bibliography</b>	<b>32</b>





## List of Figures

2.1	MIM Application Architecture . . . . .	5
2.2	Poll and Push Sequence . . . . .	5
2.3	GCM Service[4] . . . . .	6
2.4	RRC State Machine[13] . . . . .	6
3.1	Network Statistics Collector Architecture . . . . .	12
3.2	Example of Traffic Volume Variation Trace . . . . .	12
3.3	Detecting Periodic Transfer Flow . . . . .	12
3.4	Session Flow . . . . .	12
3.5	Periodic Offset . . . . .	13
3.6	Proportion of Periodic Sessions . . . . .	13
3.7	Power Consumption of Receiving . . . . .	13
4.1	Consultant Architecture . . . . .	18
4.2	Different Approaches for Keep-alive Mechanism . . . . .	18
4.3	Consultant for Keep-alive . . . . .	19
4.4	Message-Delayed Delivery Flow . . . . .	19
4.5	Consultant for Message-Delayed Delivery . . . . .	19
4.6	Consultant APIs . . . . .	19
6.1	Scenarios . . . . .	26
6.2	Separating Example . . . . .	27
6.3	Trace Replay Flow . . . . .	27



# List of Tables

2.1	Comparison between XMPP and MQTT . . . . .	6
3.1	Detecting Result . . . . .	14
3.2	Keep-alive Interval of Applications . . . . .	14
6.1	Energy saving on keep-alive messages with different number of MIM applications . . . . .	27
6.2	Result of Trace Replay . . . . .	28



# Chapter 1

## Introduction

As mobile devices are becoming more powerful, for example multi-core CPU, faster network speed, and higher camera pixel, many emerging applications are driven with these phone improvements to provide better experience. With many diverse applications installed, one common issue is that these applications would consume significant battery power and thus shorten phone's usage time. Therefore, improving energy efficient on phone device has been a popular research topic.

This thesis targets to reduce energy consumption on one type of applications services, called mobile instant messaging (MIM). MIM applications is one of the most common installed applications on mobile phone and allow mobile users owning devices with different platforms (e.g., Android, iOS) to communicate through text, voice, or images over Wi-Fi or cellular networks.

To understand the energy impact of the MIM application, we designed a lightweight application to collect user's network usage and behavior with low battery and performance overhead. Through experiments, we make the following observations: 1) Unlike typical data transfer, MIM periodic transfer which is usually short in duration and small in size [20]. These small amounts of transmissions with very little overhead can cause serious tail-energy effect. 2) Most of periodic transfer belongs to instant messaging applications and those are mainly used keep-alive mechanism between a mobile client and server. 3) Such as in an office meeting or driving a car, the user does not want to be disrupted by notification but the server always sends every message instantly.



Owing to previous findings, we propose several power saving strategies to reduce energy overhead. We designed and implemented *Consultant*, a light-weight platform to optimize energy consumption for MIM. The design of Consultant is based on *collaborated concept*. Consultant analyzes the data provided by different MIM applications and help applications to better use phone resource. We argue that a simple sharing mechanism among applications perhaps makes user more profitable without much influence. We will show the evidence in the following chapters.

In our evaluation, we propose replay editing technique to accelerate the process of replaying the real trace collected from the users as fast as possible and still can quantify accurate power usage. In order to acquire more precise result of power saving, we split the measure part and the estimate part instead of merely using the power model. Then, we observed that it can saves about 52% MIM energy with 3 installed MIM applications and about 37% MIM energy with 2 installed MIM applications during a day after optimizing.

In summary, we make the following contributions:

- We found that periodic transfer of MIM application can cause serious energy overhead.
- We propose several power saving strategies to reduce energy overhead and design Consultant which is light-weight easy to use and portable with those strategies.
- We make MIM applications intelligently determine whether it is appropriate to receive messages and it is an opportunity to save additional energy.
- We use mixed approach to accelerate the trace replay and able to get more exact result compared to estimate merely.
- Experimental results show that Consultant can save 52% MIM energy with 3 installed MIM applications during a day.

The following chapters are organized as follows: We describe background information in Chapter 2. Chapter 3 shows the MIM application behavior which we observed and we present our power saving strategies in Chapter 4. The implementation and evaluation are characterized in Chapter 5 and Chapter 6. We summarize related work which we have surveyed in Chapter 7 and conclude in Chapter 8.



## Chapter 2

# Problem

### 2.1 Mobile Instant Messaging

In recent years, Mobile instant messaging applications (MIM) such as WhatsApp, LINE and Hangout have grown rapidly. For example, an official report from LINE shows that the company has reached 400 million registered users since first launch on June 23, 2011 and 1 million users were registered in last 5 months [5]. MIM applications allow the user not only to send and receive text messages but also multimedia containing images, video and audio for free over internet. Many of these applications even start integrating social network functions like photo/video sharing, meeting nearby friends and gaming to attract more new users. Users expect MIM applications to deliver messages with minimal delay so that they can enjoy the instantaneous communications. On the other hand, mobile phone users share the concern of mobile phone batter life. As more carriers stop offering unlimited data plan, data usage charge also become a constraint for users.

This study examines the application and energy behavior of users who install multiple MIM applications on their phones. Figure 2.1 shows the typical architecture of an MIM application. MIM application transfers messages by data plan and installed software on smartphone. Normally, an app developer would choose one of the following MIM data exchange approaches for exchanging data. Poll: HTTP polling; Push: Extensible Messaging and Presence Protocol (XMPP) [3] and MQ Telemetry Transport (MQTT) [7]. They provide services or standards to developer. Figure 2.2 shows poll and push sequences

and Table 2.1 shows the comparison between XMPP and MQTT. In addition, Google Inc. also provides Google Cloud Messaging Service (GCM) [4] for MIM data exchanging and Figure 2.3 shows the architecture of GCM Service.

One common behavior in MIM application is that application developer uses keep-alive (KA) mechanism [27] by sending a ping message to the remote server periodically to maintain a persistent connection with the server. This is due to mobile devices are hidden behind a NATs (Network Address Translation) so that the server is not able to reach (or sending message) directly to its clients. Although KA mechanism helps to establish a connection between a mobile client and the server, it also incurs many unnecessary transmissions and wastes energy consumption. Inspired by this unique phenomenon, this thesis aims to propose several power saving strategies to reduce energy overhead by KA mechanism. On the other hand, this KA mechanism would also incur much unnecessary energy usage.

## 2.2 Radio Power in 3G Cellular Network

3G network using WCDMA technology is operating commercially in many countries. In 3G network, the Radio Resource Control (RRC) is used to handles the control plane to signal mobile clients for utilizing limited radio resources. There are four primary RRC states in 3G network: IDLE, CELL\_PCH (Cell Paging Channel), CELL\_FACH (Cell Forward Access Channel) and CELL\_DCH (Cell Dedicated Channel) [13]. Figure 2.4 shows RRC state machine. Below we briefly review the energy impact on these RRC states. IDLE and CELL\_PCH state consume nearly the same and as low as 5 mA, which is negligible to network energy consumption. During CELL\_DCH state, the mobile device is allocated a dedicated transport channel in the downlink and in the uplink along with a requisite number of physical channels. It consumes the most network resources, including RNC processing and air interface resources. It is the most power hungry state in RRC and would cost nearly 200 mA for a typical phone device. During the CELL\_FACH state, a mobile client uses lower data rate to transmit small bits of data and it would cost medium amount of power ( $\approx 100$  mA).

An RRC state promotion takes up to 2 seconds since it requires the authentication and the exchange of signal messages. In contrast, state demotion finishes much faster but incurs tail times to match the inactivity timer value that cause significant waste of resources called tail-energy [11, 18, 19]. In 3G network, a large fraction (nearly 60%) of the energy, referred to as the tail-energy, is wasted in high-power states after the completion of a typical transfer.

Because of tail-energy effect, the overhead of small data transfer is inefficient obviously. Thus, aggregating or reducing small data is important issue of the power consumption on smartphones in 3G network particularly.

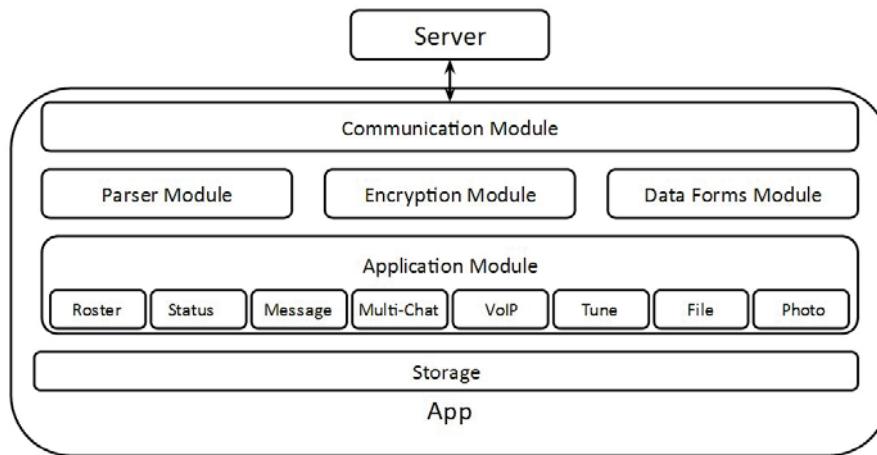


Figure 2.1: MIM Application Architecture

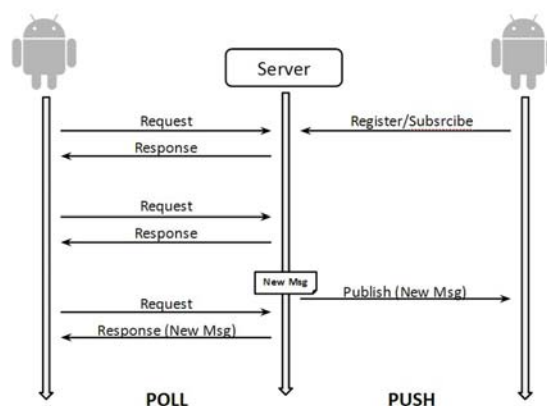


Figure 2.2: Poll and Push Sequence

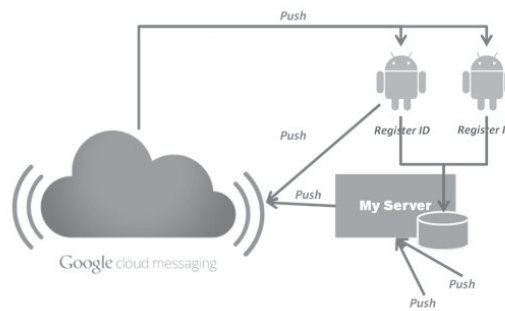


Figure 2.3: GCM Service[4]

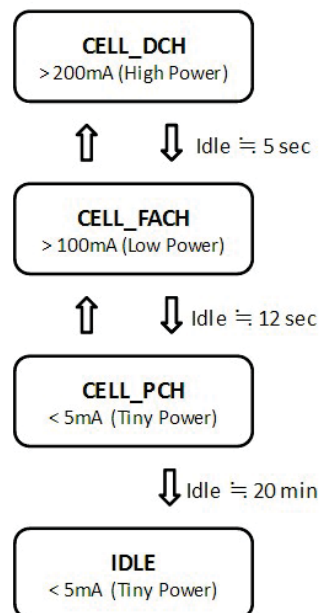


Figure 2.4: RRC State Machine[13]

Protocol	XMPP	MQTT
Character	Standard Decentralized Flexible	Simplicity Zero Administration
Scalability	Large	Small
Implementation	Hard	Easy
Power Consumption	High	Low
Network Traffic	High	Low

Table 2.1: Comparison between XMPP and MQTT



## Chapter 3

# Mobile Instant Messaging Application Behavior

### 3.1 Measurement and Detecting

In order to understand the energy usage when multiple MIM applications run a mobile phone, this work starts with a study to answer the following questions.

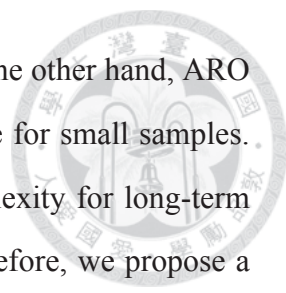
- How much energy consumption is used by MIM applications? Do they exhibit a simple summation or the sum is larger or less than the sum of each application?
- How prevalent is keep-alive mechanism in an MIM application? How many MIM application use keep alive mechanism? What is the interval of keep-alive messages?
- How much energy can be attributed to keep-alive messages? Does having a longer or shorter interval affect the energy consumption?

To understand user's network usage and behavior, we design Network Statistics Collector to be low battery and performance overhead. Ideally, we need to obtain root permission on the phone to obtain the network usage in depth like packet headers for analysis by running libpcap-like tools. But rooting the phones is not acceptable for every user. Instead, to make the deployment more practical and easier without rooting the phone and we use traffic volume variation and TCP connection information to represent usage pat-

tern. On the other hand, we do not require rooting a device so that we can collect more user traces. It costs less than 5% energy overhead per day and only causes a tiny effect in user experience. Traces of nine graduate students studying in computer science department were collected for more than 7 days. These volunteers were told to use their phones as usual and they reported no difference in phone response time and battery life during the trace collection period. Figure 3.1 shows application architecture. Network Statistic Collector running as background service to record user's traffic volume every second on smartphones and uploads compressed traces to the server automatically once a day.

The collector records Cumulative Received Bytes (RXB) and Cumulative Transmit Bytes (TXB) every second from each process of the application according to system information. We then process the time series to calculate the difference in TXB and RXB between two consecutive seconds. We define the difference in RXB and TXB as *traffic volume variation per second* of all processes in every second. Traffic volume variation per second must be greater than or equal to zero. The application network behavior is represented as a time series of RXB and TXB so a user running three MIM has three time series of how many bytes were sent or received during each second. Figure 3.2 shows the example of traffic volume variation trace.

Since we only have traffic volume variation information, we do not have direct access to data that can determine the keep-alive interval, e.g. source code. We observe that network traffic sent by MIM application can be human-triggered or application-triggered keep-alive messages. We expect human-triggered messages to be larger and do not happen in a fixed interval. Keep-alive messages are usually based on some pre-determined timeout value so we expect them to be small in size, usually less than 1K bytes, and occur every fixed interval. Compared with human-triggered data transfer, periodic transfer is usually short in duration and small in size [20]. How to detect these small amounts of transmissions with very little overhead is an issue. Many works have been proposed to solve this problem [20]. However, those are not directly applicable in our scenario. One previous work directly seeks regular spacing between packets by applying DFT [17] or auto-correlation [25]. This method works well, but do not fit our scenarios. This is because



we have much less samples due to Android platform limitation. On the other hand, ARO [1] and Periodic Transfer [20] provide simple approaches to analyze for small samples. But these methods have very low accuracy and unacceptable complexity for long-term logging because we can analyze the trace of each application. Therefore, we propose a modified approach to effectively detect periodic transfers in our scenario. Now, we describe how to detect periodic transfer in details. The idea of our approach is that the data size sent by applications for periodic transfers is fixed. By finding the periodic data size difference, we can then find the interval by subtracting when two sending of this data size transfer. Finding the data size is easier in our traces because we already know the variation. Our approach takes three steps. First, we extract sessions from each trace. A session is defined as Transfer Sessions. The session consists of continuously increasing cumulative traffic volume of received bytes or sent bytes in period of time and symbolizes a complete transfer of one application. It also means that traffic volume variation per second is always positive during this period. Each session can be uniquely identified by application, e.g. WeChat or Line, and process information (Session Details, e.g. Application ID, Process ID), traffic volume variation (Session Size), start time and end time (Session Length). Then, we categorize sessions based on a range of sizes and types of sessions from Session Details into groups and filter session groups with oversize. Our target is the group with small data size because periodic transfer usually contains small session size which are almost identical for syncing with server purpose. Finally, we search periodic transfer in each session group and find candidate applications that show signs of periodic transfer. We assume each application has only one or two intervals configured for periodic transfers. We enumerate all possible intervals of time series in each group, and find most common intervals and mark those sessions as **Periodic session**. If both the proportion and frequency of periodic sessions are higher than threshold, the application informed from session details will be viewed as a suspicious application. Figure 3.3 shows the detecting flow.



## 3.2 Evaluate Detecting Flow

Since this is heuristic, traces do not follow perfectly our assumptions. To quantify the noise, we define two parameters called Session flaw and Periodic offset, we extract strictly increasing traffic volume from the trace and look upon those periods of the trace as sessions. Sometimes we observe some periods of the traffic volume increasing non-continuously in the session and we called it *Session flaw*. Session flaw prevent fragments of session because of unstable network environment. Furthermore, sometimes we observe several common intervals but our assumption is that only one or very few intervals are the norm. Therefore, we define *Periodic offset* to be the tolerance range when we find most common intervals from sessions. Figure 3.4 shows an example of the session flaw and Figure 3.5 shows an example of the periodic offset.

Then, we randomly select 2-3 traces and calculate average proportion of periodic sessions with different session flaw length and periodic offset. Figure 3.6 shows the result. We observed that the proportion eventually converge. But, in order to prevent oversize session because of larger session flaw length and loose detecting because of larger periodic offset. Therefore, we choose (Session flaw Length, Periodic offset) = (5, 10) which is reasonable from our study.

It is difficult to evaluate detecting flow because we lack ground truth. Therefore, we generate a trace with 500 periodic sessions and 100 non periodic sessions and use our algorithm to detect periodic transfer. The overall false positive ratio is 2-3% with our detecting algorithm.

## 3.3 Measurement Results

Table 3.1 shows the results and we can make the following observations: 1) Average proportion of periodic sessions of all traces is 46%. 2) Most of periodic sessions belong to the background<sub>1</sub> or service<sub>2</sub> process. 3) Most of periodic sessions belong to instant messaging applications. Therefore, we decide to explore the periodic transfer behavior of running multiple instant messaging applications.

We observed that the periodic sessions are mainly used keep-alive mechanism between a mobile client and server. A keep-alive mechanism is sending a message by one device to another to check the link between the two devices and keep-alive signal is often sent at predefined intervals. Table 3.2 shows the keep-alive predefined interval of popular MIM applications on Google Play Market. We use Monsoon power monitor [6] to measure the power consumption of a prevalent MIM application called LINE [5], and discover that it utilize 386.20  $\mu$ Ah and cost 16 second for keep-alive once on Galaxy Nexus with Android 4.3.

### 3.4 Instant or Non-instant

In additional, we find another problem from the trace is that the server always sends every messages instantly even if user is in some unsuitable moment (e.g., meeting) when user forgot to turn off notifications or similar functions. This could disrupt the user's activity. In other word, an MIM application should be able to determine whether it is appropriate to receive messages. It is not only for improving user experience but also reducing power consumption because of the tail-energy effect. We measure the power consumption of LINE for receiving simple messages separately with each incurring tail-energy and combined into close proximity into one tail energy on Galaxy Nexus with Android 4.3. Figure 3.7 shows the result. Not surprisingly, comparing the energy of LINE receiving each message with a pause in between and receiving multiple messages close in time, receiving ten messages close to each other can save 65.4% of energy consumption.

---

<sub>1</sub> The process contains services that should remain running. [13]

<sub>2</sub> The process contains background code that is expendable. [13]

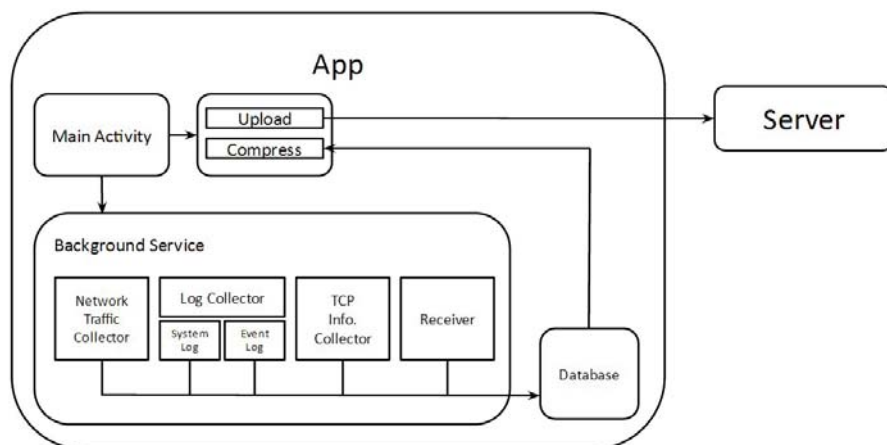


Figure 3.1: Network Statistics Collector Architecture

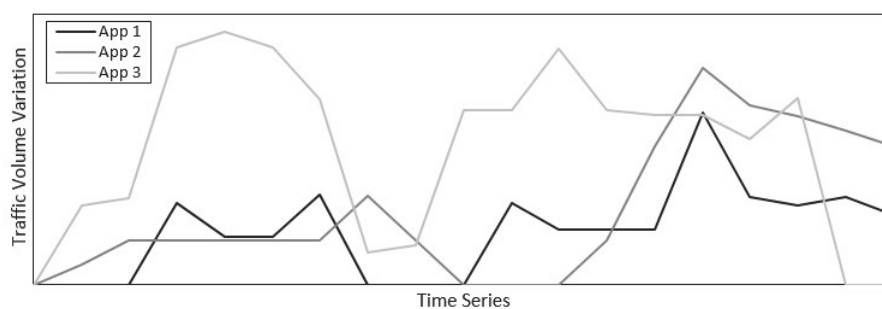


Figure 3.2: Example of Traffic Volume Variation Trace

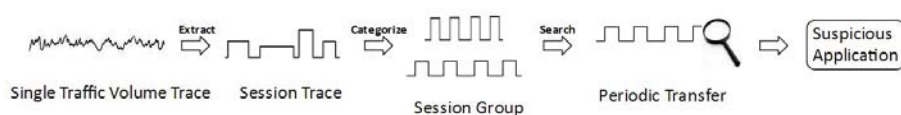


Figure 3.3: Detecting Periodic Transfer Flow

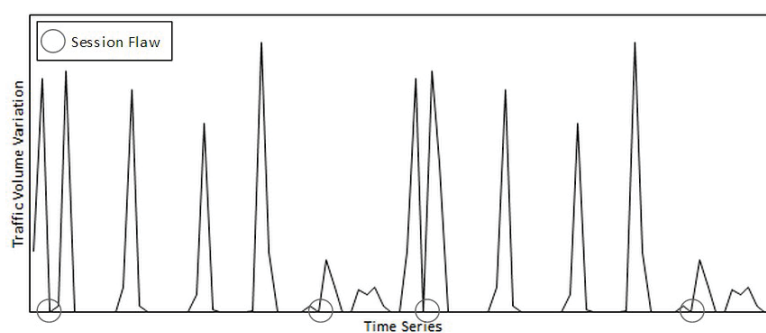


Figure 3.4: Session Flaw

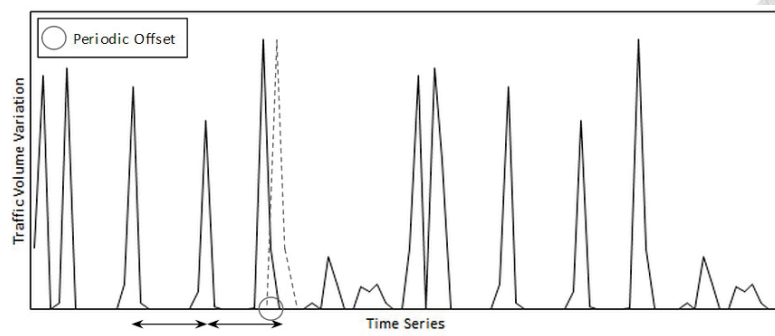


Figure 3.5: Periodic Offset

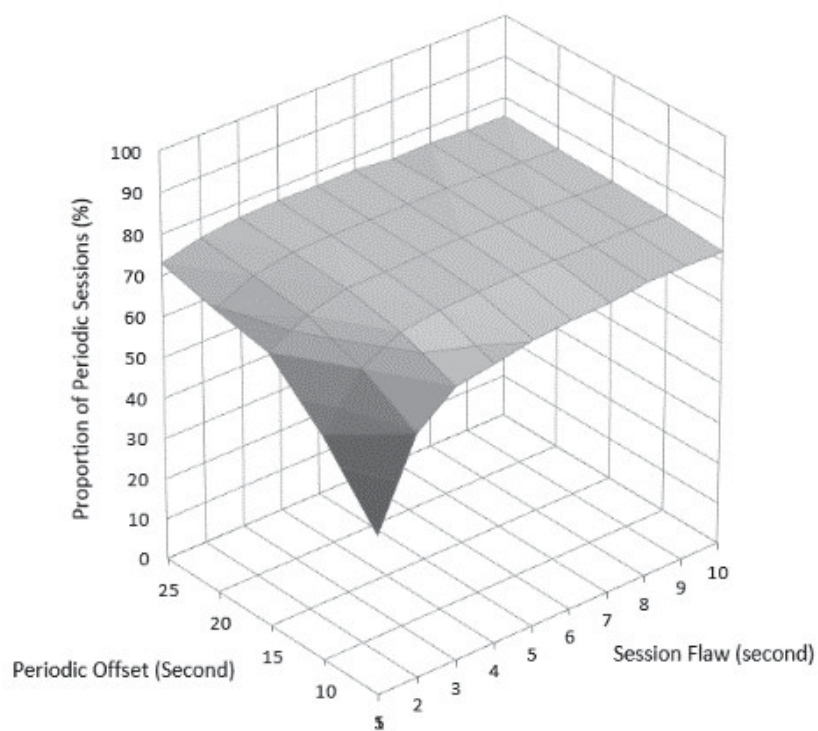


Figure 3.6: Proportion of Periodic Sessions

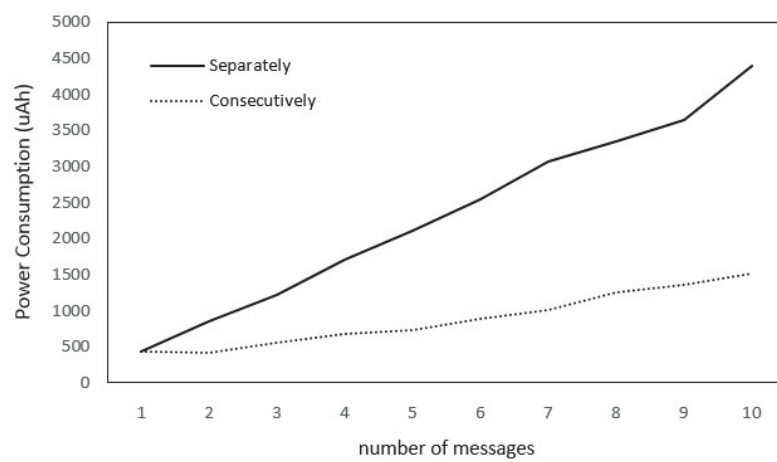


Figure 3.7: Power Consumption of Receiving



User	Periodic Sessions (%)	Suspicious Application (Suspicious/Total)	Top Application and Process Type
1	52%	10/24	WeChat(Service), LINE(Service), Facebook Messenger (Foreground UI)
2	54%	12/29	WeChat(Service), Facebook Messenger (Foreground UI), LINE(Service)
3	68%	6/17	Flippo(Perceptible), Facebook(Perceptible), Facebook Messenger (Background)
4	24%	3/11	LINE(Service)
5	54%	22/36	WeChat(Service), Facebook Messenger (Background), LINE(Service)
6	27%	8/21	LINE(Service)
7	14%	1/7	Facebook(Perceptible)
8	65%	2/7	Flippo(Perceptible)
9	52%	18/33	WeChat(Service), Facebook Messenger (Background), LINE(Service), Taobao (Service )

Table 3.1: Detecting Result

Application	Version	Downloads	Interval
Cubie	1.1.414	1,000,000 - 5,000,000	<1 min
Facebook Messenger	Depend on device	100,000,000 - 500,000,000	15 min
Hangout	2.0.303	100,000,000 - 500,000,000	15 min
imo messenger	4.4.5	5,000,000 - 10,000,000	5 min
Kakao Talk	4.3.6	100,000,000 - 500,000,000	10 min
LINE	4.1.2	100,000,000 - 500,000,000	3-5 min
QQ	4.6.1	5,000,000 - 10,000,000	5 min
Skype	Depend on device	100,000,000 - 500,000,000	5 min
Tango	Depend on device	100,000,000 - 500,000,000	10-20 min
Viber	4.2.1.1	100,000,000 - 500,000,000	10 min
WeChat	5.2	100,000,000 - 500,000,000	5 min
Weibo	4.2.6	5,000,000 - 10,000,000	10 min
Whatsapp	2.11.186	100,000,000 - 500,000,000	10-20 min
Yahoo Messenger	1.8.4	10,000,000 - 50,000,000	< 1 min

Table 3.2: Keep-alive Interval of Applications



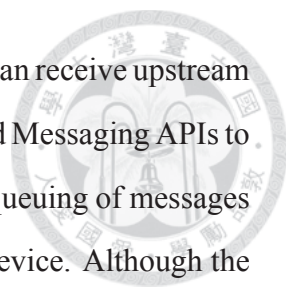
## Chapter 4

# Power-Saving Strategies

Based on our finding in Chapter 3, we provide power-saving strategies for MIM applications and how developers can use it. There are many kinds of MIM application on the market such as WhatsApp, LINE and Hangout and the user often installs multiple MIM application because of the network effect. Multiple MIM applications using keep-alive messages in an unsynchronized manner can cause serious tail-energy effect mentioned in previous chapters. We propose a Collaborate concept to solve this problem. On Android platform, each application is independent and it can produce many benefits, but we argue that a simple sharing perhaps makes user more profitable without any influence. We design Consultant which can help MIM applications to coordinate their keep-alive timing and save energy. We design Consultant to be light-weight easy to use and portable. Figure 4.1 shows Consultant Architecture. Applications can provide primary data to Consultant and Consultant will share useful information to all applications after analyzing.

### 4.1 Keep-alive Mechanism

Keep-alive (KA) mechanism is a common approach to establish a connection between the server and the mobile device. An app developer can design his own KA mechanism or simply use Google Cloud Messaging (GCM) Service [4] provided in Android platform. GCM Service for Android is a service that allows applications to send data from the server to the user's Android-powered device, and also to receive messages from devices on the



same connection. Using the GCM Cloud Connection Server, the user can receive upstream messages from the user's device. The developers use the Google Cloud Messaging APIs to design their applications and the GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device. Although the GCM service uses a keep-alive mechanism to send a heartbeat network packet about every 15 minutes on Wi-Fi environment and about every 30 minutes on 3G network, it is always unreliable for all users and lacking in flexibility for developers.

As an alternative, we design Consultant to suggest applications whether applications need to send a heartbeat or not. When an application sends a heartbeat successfully, Consultant will keep each heartbeat called *heartbeat record*. Then, whenever an application want to send a heartbeat, Consultant will give it the suggestion by tracking past heartbeat record. Figure 4.2 shows above three approaches. When an application sends a heartbeat successfully, Consultant preserves the IP address and the heartbeat record. A heartbeat record contains App ID and timestamp. While an application wants to send a heartbeat, Consultant will check the heartbeat record. If the time difference is less than predefined keep-alive interval, it means the heartbeat record is valid. Consultant will suggest the application **need not to send**. Otherwise, the app is free to send heartbeat beacon. As we known, the primary purpose of KA mechanism is to notify the server that the mobile device has moved. Therefore, whenever Consultant finds the IP address has changed after keep-alive successfully, it will clear all records and accept next request from applications. In addition, Consultant periodically accepts keep-alive request with fixed time for exception (e.g., an unstable network environment or the closed connection by the server) even though heartbeat record is valid. In other word, if Consultant discovers an application's keep-alive request has been rejected for 5 times, Consultant will suggest to send a heartbeat next time for exception.

## 4.2 Message-Delayed Delivery

Users complain that MIM applications require explicitly and do not have the intelligence determine whether it is appropriate to receive messages. In some scenarios, such as in an



office meeting or driving a car, the user does not want to be disrupted by MIM notification. We see this as an opportunity to save additional energy. One possible way is to provide developers with a mechanism for changing user status automatically instead of changing by the user. Our idea is as followed: if the application does not receive feedback (e.g., turn on the screen, unlock) after receiving a message from the user within timeout period (e.g., 2 minutes), the application will change user's status from online to busy and notify the server. Then, the server will block all messages and the user will not receive any messages until disabling delayed delivery. Figure 4.4 shows the message-delayed delivery flow.

Similarly, Consultant can give some helps for message-delayed delivery. Consultant will keep the status called *status record* whenever the status is changed. Therefore, if an application receives a message, Consultant will check the status record which shows whether the user is busy. If the user is busy, the application will enable delayed delivery without triggering the timer and it can also save additional energy. Figure 4.5 shows Consultant for message-delayed delivery.

In addition, we reject an aggressive approach for power-saving which could cause a negative user experience. In the message-delayed delivery flow, we turn the disabling delayed delivery decision over to the developers. We give developers the maximum flexibility to design their own application.

### 4.3 APIs

Applications interact with Consultant through the API of Consulting, Refreshing, and Controlling. Before an application sends a heartbeat packet, Consultant will give a suggestion to the application call Consulting. Developer can decides whether the application send the packet by the suggestion. After the application receives ACK from the server successfully, the application needs to call Refreshing to set valid timestamps. On the other hand, when the application receives a new message, it can call Controlling to enable Message-Delayed Delivery. However, disable delayed delivery decision depends upon developer's own implementations. In our case, when user unlocks the phone, it will disable delayed delivery.



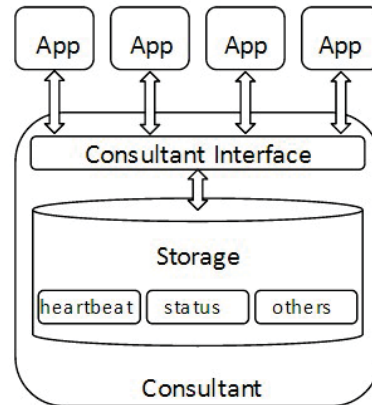


Figure 4.1: Consultant Architecture

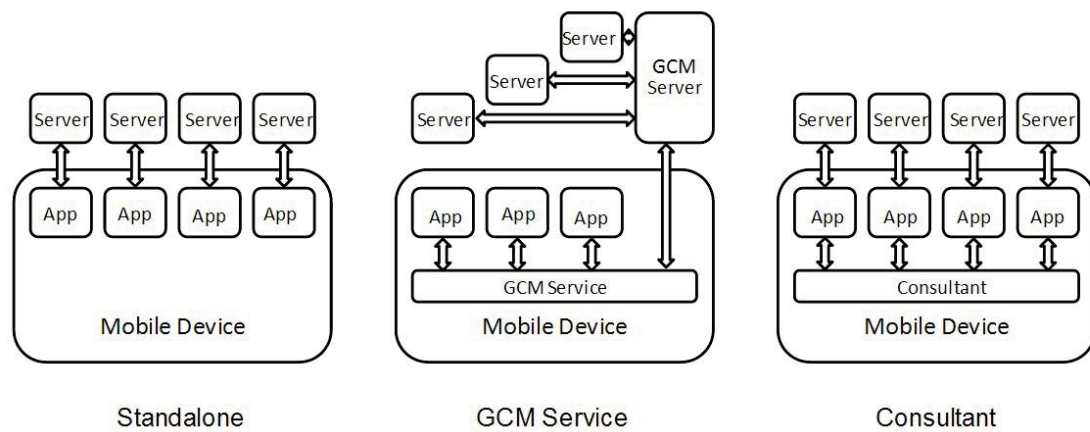


Figure 4.2: Different Approaches for Keep-alive Mechanism

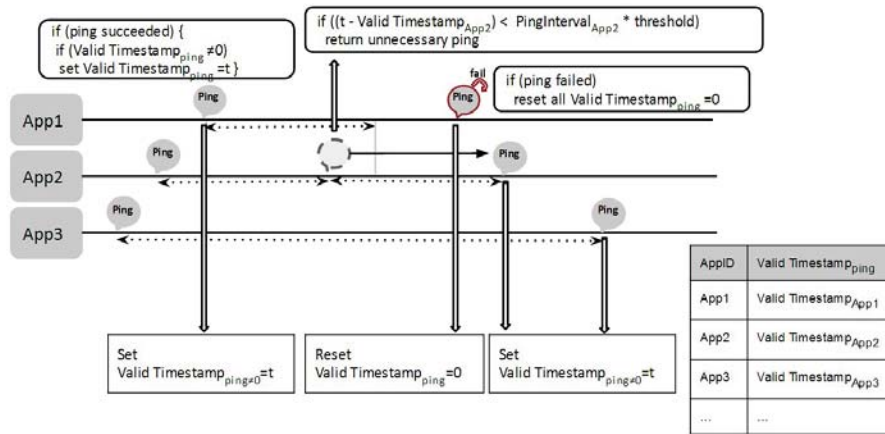


Figure 4.3: Consultant for Keep-alive

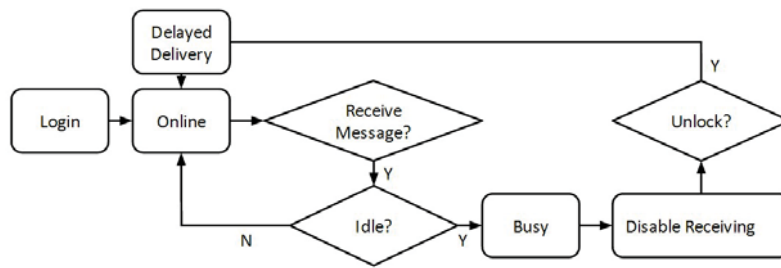


Figure 4.4: Message-Delayed Delivery Flow

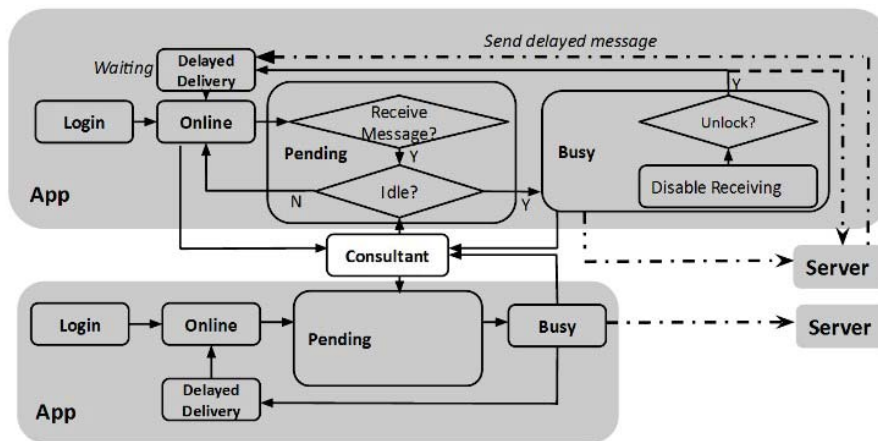


Figure 4.5: Consultant for Message-Delayed Delivery

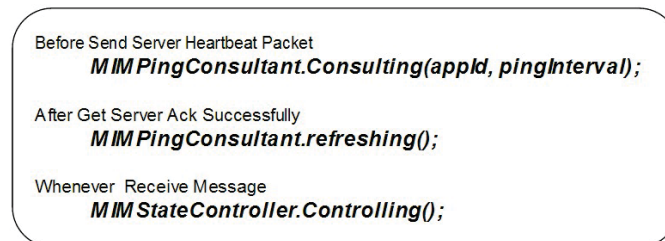


Figure 4.6: Consultant APIs



## Chapter 5

# Implementation

Implementing the strategies in previous chapters involves coordination between server and client side. Servers that support message-delayed delivery can support directly. Since the commercial MIM applications such as WhatsApp are not open source on Android and so is the corresponding server side software, our implementation is based on open source MIM application and messaging server. We augment XMPP protocols because their standards are publically-accessible and many existing server and client implementations are available [3].

### 5.1 Client-side

We modify an open source MIM application called Yet another XMPP Instant Messenger (Yaxim) [9]. We run multiple copies of Yaxim with different keep-alive configurations to represent multiple MIM applications to simulate real environment on the smartphone. We modify the BUSY status which means that the user is not appropriate to receive messages and the application will change to this status automatically by Consultant.

### 5.2 Server-side

On server side, we modify Erlang Jabber/XMPP daemon (ejabberd), which is a cross-platform, fault-tolerant and modular open source instant messaging server which is the

ejabberd instant messaging server allows two or more people to interact in real-time using typed text [2]. In order to implement and expand message-delayed delivery, we modify Session Manager and Client to Server Service (C2S) module in ejabberd to keep the delayed messages in a queue on the server and deliver them when client changes its state to active. In other words, if the C2S module receive that client change to busy status, it will notify Session Manager and the following messages which are sent to that client will be blocked by Session Manager.



## Chapter 6

# Evaluation

### 6.1 Methodology

To evaluate the effectiveness of delayed delivery, we ran experiments on Samsung Galaxy Nexus I9250 with Android 4.3 with Chunghwa Telecom 3G. All experiments were conducted in NTU CSIE building. The MIM energy consumption is measured using Monsoon power monitor [6]. Each experiment is repeated five times on real devices in a lab environment except built-in applications (e.g. Camera, Calendar, and Calculator) and the MIM energy consumption is the measurement deducted from base power. (We assume the base power to be 2.5μAh/second):

$$Energy_{MIM} = Energy_{Measure} - Power_{Base} \times Time_{Measure}$$

The questions we want to answer in the evaluations are:

- How much power can be saved on a multi-MIM application phone if keep-alive messages are coordinated?
- How much power can be saved if we deployed Consultant on a multi-MIM application phone in real trace?

The power saving is computed as

$$\text{Energy Saving Ratio} = (Energy_{MIMsOrigin} - Energy_{MIMsOptimization}) / Energy_{MIMsOrigin}$$

$$\left( \begin{array}{l} Energy_{MIMsOrigin} : \text{Without Consultant} \\ Energy_{MIMsOptimization} : \text{With Consultant} \end{array} \right)$$



We evaluate our strategies based on real-user traces described in Chapter 3. The main challenge in quantifying power-saving is due to time-dependent behavior, e.g. the tail energy effect in Radio Resource Management. One way to reproduce accurate power measurement is to replay minute-by-minute of the original trace and in the exact same network condition. It is time-consuming to replay the original trace and impossible to retain the same network condition. Therefore, to replay the real trace collected from the users as fast as possible and still collect meaningful power measurement, we adapt Drive-Thru [16] to automatically re-run traces on Android platform using Robotium. Robotium is an open-source Android GUI testing tool that has full support for native and hybrid applications [8].

## 6.2 Consultant for Keep-alive

The first factor we evaluate is the number of MIM applications running on the phone and their keep-alive intervals. According to our collected user traces in Chapter 3, we choose the keep-alive intervals of three popular MIM applications: LINE (5 minutes), WeChat (5 minutes), and Facebook Messenger (15 minutes) and create 2 scenarios. In the first scenario, we run two copies of the MIM applications with screen turned off and minimal activities on the phone. One application is configured to send keep-alive messages every 5 minutes and the other every 15 minutes and neither of them sends any messages. Similarly, in the second scenario, we repeat the same experiment with three copies of the applications and set keep-alive intervals as 5 minutes, 5 minutes and 15 minutes respectively. We

measure the power consumption for keep-alive during an hour. Since Consultant does not rely on keep-alive messages, the power spent on sending and receiving keep-alive messages is the power saved using Consultant. The power spent on keep-alive is calculated as follows:

$$Energy_{Keep\_alive} = Energy_{Measure} - Power_{Base} \times Time_{Keep\_alive}$$

Here Power Measure is the power measured by Monsoon power monitor during the one-hour. Table 6.1 shows the energy consumption of the two scenarios., First column shows the power spend on keep-alive and the second column (marked optimization) with Consultant so fewer keep alive message is sent. We observed that with Consultant installed on the phone, Consultant can save 45.3% MIM energy with 2 applications and 64.7% MIM energy with 3 applications during an hour. Since keep-alive messages cause tail-energy to occur, with fewer applications, we expect less energy spent on keep-alive messages.

### 6.3 Trace Replay

We quantify power saving with optimization described in Chapter 4 using synthetic traces based on real user traces. Instead of generating the random traces to evaluate our strategies, we decide to generate scenarios which are similar to user activities we observed from the real user traces we collected. However, because of limited user data, we simplify user behaviors and use replay editing to accelerate.

First, we separate a day's trace into two states for each application: the application is either sending keep-alive messages to the server or actively sending or receiving messages in all extracted sessions. However, because Consult cannot save any energy on sending messages, we just consider receiving messages. Meanwhile, we reduce the size of receiving messages for simplifying and all receiving messages only contain one characters. Then, we scan through the messages in the trace to break the trace into either keep-alive period or actively messaging period. When a receive messages arrives on the phone and the user did not respond (e.g., turn on the screen) after receiving, it means that it was not

a suitable time for receiving the message. On the contrary, if the user responded immediately, we mark that *instant* as suitable to receive messages. Figure 6.1 uses circle to mark as time suitable to receive messages and a cross otherwise. We set the timeout period as 2 minutes for enabling delayed delivery, so we split the receiving period as a 2-minute slot after receiving a message. In other word, each slot normally contains one message but if the messages have been received within 2 minutes of each other, the slot will extend 2 minute after last message. After that, we aggregate those slots and mark them as Message-Replaying part, which we replay messages one-by-one with timing preserved and mark as remaining time slot as a Fast-Forwarding part. Figure 6.2 shows how the traces is reduced into *Message-Replaying* and *Fast-Forwarding* part. We run the message-replay part on real devices and set same keep-alive intervals as real applications and deduct the base power (2.5μAh/second):

$$Energy_{Message\_Replaying} = Energy_{Measure} - Power_{Base} \times Time_{Message\_Replaying}$$

The user's status will be changed to busy by Consultant so it will trigger delayed delivery automatically and we design a background service to disable delayed delivery according to the message attribute. In the fast-forward part, we estimate the energy saving by previous result:

$$Energy_{Fast\_Forwarding} = Energy_{Keep\_alive} \Bigg/ \frac{\text{Hour}}{\text{Hour}} \times Time_{Fast\_Forwarding}$$

However, because of limited user data, we assume that the user's location is stationary and make up for the limitation by exception mechanism in Chapter 4. Finally we sum up the power consumption:

$$Energy_{MIMs} = Energy_{Message\_Replaying} + Energy_{Fast\_Forwarding}$$

Figure 6.3 shows the flow of the trace replay and Table 6.2 shows the result of traces. We observed that it can saves about 52% MIM energy with 3 installed MIM applications



and about 37% MIM energy with 2 installed MIM applications during a day after optimiz-  
ing.

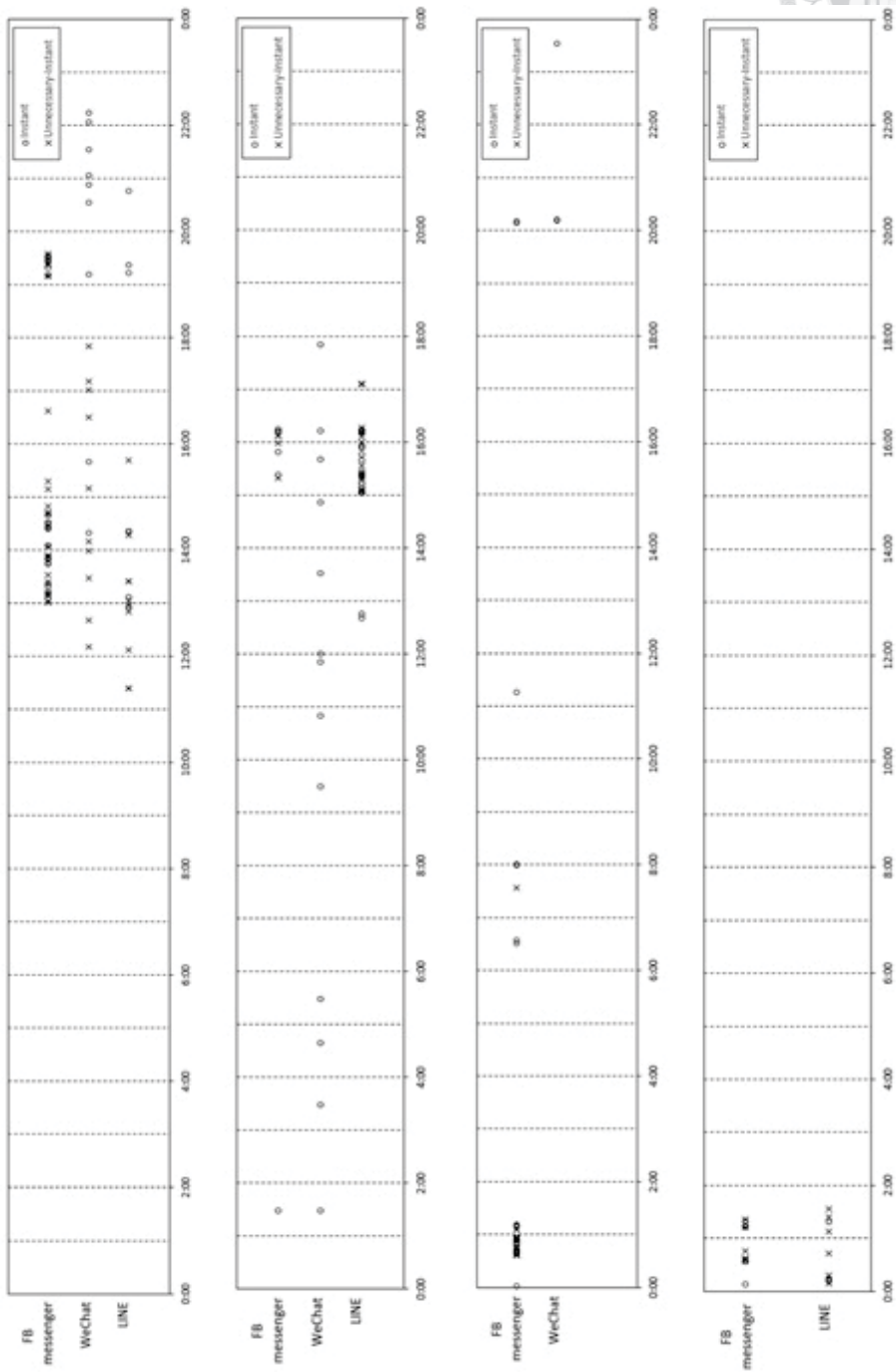


Figure 6.1: Scenarios

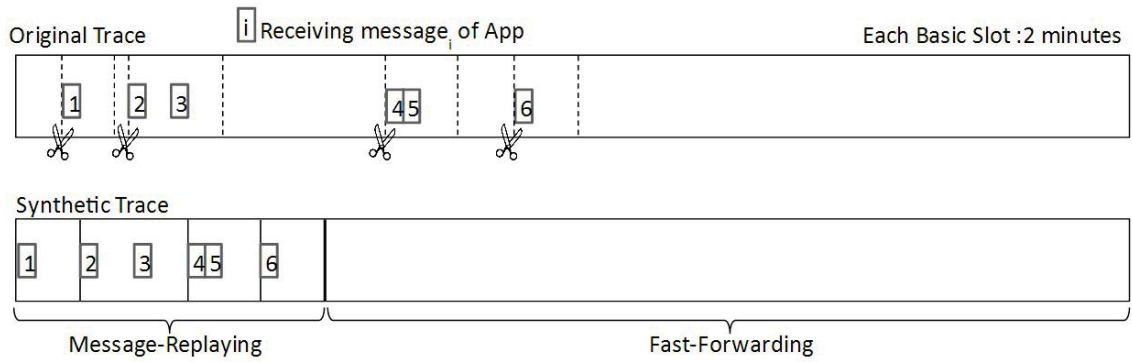


Figure 6.2: Separating Example

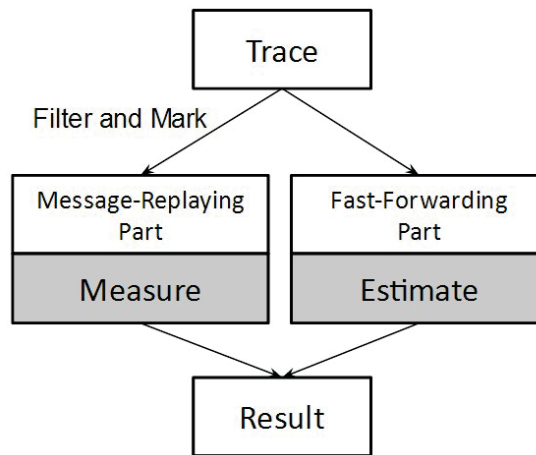


Figure 6.3: Trace Replay Flow

Scenario	Average Power Consumption (uAh)			Information
	Origin	Optimization	Saving	Keep-alive Interval (min)
1	5108.752	2795.008	<b>45.29%</b>	(5,15)
2	7357.228	2600.002	<b>64.66%</b>	(5,5,15)

Table 6.1: Energy saving on keep-alive messages with different number of MIM applications

Trace	Average Power Consumption (uAh)										Information			
	Message-Replaying Part (Measure)		Fast-Forwarding Part (Estimate)			Total		Saving			Replay Time (min)	Delayed Messages	Status Update Times	Keep-alive Interval (min)
	Origin	Optimization	Origin	Optimization	Origin	Optimization	Replay Saving	Remaining Saving	Total Saving					
1	63935.45	49287.38	161245.91	56983.38	225181.37	106270.76	22.9%	64.7%	52.81%	125	18.60%	35	(5,5,15)	
2	31728.83	30000.85	167990.04	59366.71	199718.87	89367.56	5.4%	64.7%	55.25%	70	7.94%	28	(5,5,15)	
3	21709.28	23131.62	117501.30	64285.18	139210.57	87416.80	-6.6%	45.3%	37.21%	60	29.17%	18	(5,15)	
4	10413.58	9631.12	120055.67	65682.69	130469.25	75313.81	7.5%	45.3%	42.27%	30	27.27%	5	(5,15)	

Table 6.2: Result of Trace Replay



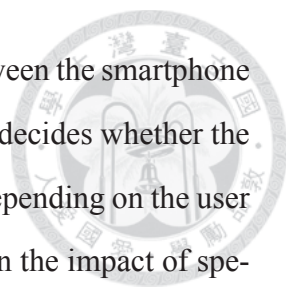


## Chapter 7

### Related Work

Energy efficient is a popular issue in mobile computing because of the limited battery. Several studies have designed many optimization policies for mobile power saving in term of the power management at primary components such as CPU, the screen and the network resource. Some researcher study how to reduce power consumption by improving the energy performance of mobile applications. Our work focuses on the energy impact of applications. Based on our survey results, we are the first to study the radio power overhead of multi-MIM applications on mobile devices and purpose energy saving strategies.

As we known, cellular radios consume more power in network resource. Previous work on reducing radio power provides many solutions in the whole system. RadioJockey [10] is a system that predict the end of communication by the system trace of the mobile device. It can reduce tail energy effect by effectively invoking fast dormancy and achieves 20-40% energy savings. Bartendr [24] is an energy-aware cellular data transfer system and the scheduling algorithm designed by measurements of the relationship between signal strength and power consumption. It can reduce 60% radio power by their algorithms. ARO [21], the mobile Application Resource Optimizer, the tool that detect radio resource and energy inefficiencies, such as content prefetching on the cross-layer. It benefits on several popular Android applications. LoadSense [12] can determine the suitable times for communication by the measurement result of the cellular load. On the other hand, there are some solutions in connection with some specific behavior (e.g. prefetching, periodic transfer) in applications. Feng Qian et al [20] revealed application traffic pattern



called periodic transfers which is the periodically communication between the smartphone and the corresponding server every fixed second. Procrastinator [22] decides whether the smartphone needs the fetching and reduce 4X energy consumption, depending on the user activities. Our work is different from previous studies, as we target on the impact of specific applications and study in more detail. Fengyuan Xu et al [26] derived general design principles for energy-efficient event for email sync on smartphones which is similar but materially different. In our evaluation, we proposed replay editing technique to quantify accurate power usage and evaluate the performance. Radhika Mittal et al [15] presented and implemented a tool to estimate the energy usage for their mobile applications and evaluate and compare it with real device energy measurements. AppInsight [23] can identify the critical path in user transactions by the instruments of mobile application binaries. It is lightweight, and helped developers improve the quality of their application.

Finally, Xuan Zhou et al [27] and Ling-San Meng et al [14] also investigated power consumption of MIM applications. They purposed a model to estimate the network performance of running MIM service [27] and presented a low power consumption solution about presence mechanism in MIM service [14]. Their work also reduced the energy overhead of MIM applications and we believe that much more power can be saved in our scenario if we combined their ideas.



## Chapter 8


### Conclusion

In this thesis, we examine the application and energy behavior of users who install multiple MIM applications on their phones and propose several power saving strategies to reduce energy overhead. First, we discuss periodic sessions by user's network usage and behavior and find that most of them belong to MIM applications. Second, such as in an office meeting or driving a car, the user does not want to be disrupted by MIM notification. We see those as an opportunity to save additional energy, so we propose Consultant which is light-weight easy to use and portable for optimizing. Based on our experiment, it can save about 52% MIM energy with 3 installed MIM applications during a day after optimizing. As we known, it is difficult to integrate MIM applications because users always choose different applications on their own will. Therefore, with the growth of MIM applications on the market, the user will install more MIM applications possibly and our strategies will become more and more useful.

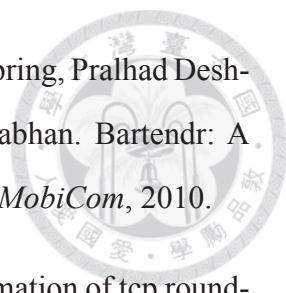


## Bibliography

- [1] Android developers. <http://developer.android.com/index.html>.
- [2] ejabberd. <http://www.ejabberd.im/>.
- [3] The extensible messaging and presence protocol. <http://xmpp.org/>.
- [4] Google cloud messaging for android. <http://developer.android.com/google/gcm/index.html>.
- [5] Line: Free calls & messages. <http://linecorp.com>.
- [6] Monsoon power monitor. <http://www.msoon.com/labequipment/powermonitor/>.
- [7] Mq telemetry transport. <http://mqtt.org/>.
- [8] Robotium. <https://code.google.com/p/robotium/>.
- [9] yaxim. <http://yaxim.org/>.
- [10] Pavan K. Athivarapu, Ranjita Bhagwan, Saikat Guha, Vishnu Navda, Ramachandran Ramjee, Dushyant Arora, Venkata N. Padmanabhan, and George Varghese. Radio-jockey: Mining program execution to optimize cellular radio usage. In *MobiCom*, 2012.
- [11] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *IMC*, 2009.
- [12] Abhijnan Chakraborty, Vishnu Navda, Venkata N. Padmanabhan, and Ramachandran Ramjee. Coordinating cellular background transfers using loadsense. In *MobiCom*, 2013.

- 
- [13] NSN Signals Research Group. Smartphones and a 3g network, 2010.
- [14] Ling-San Meng, Da-Shan Shiu, Ping-Cheng Yeh, Kuan-Chi Chen, and Hung-Yi Lo. Low power consumption solutions for mobile instant messaging. In *MobiCom*. IEEE Transaction on Mobile Computing, 2012.
- [15] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In *MobiCom*, 2012.
- [16] Daniel Peek and Jason Flinn. Drive-thru: Fast, accurate evaluation of storage power management. In *USENIX*, 2005.
- [17] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. Tcp revisited: A fresh look at tcp in the wild. In *IMC*, 2009.
- [18] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *IMC*, 2010.
- [19] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *ICNP*, 2010.
- [20] Feng Qian, Zhaoguang Wan, Yudong Gao, Junxian Huang, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Periodic transfers in mobile applications: Network-wide origin, impact, and optimization. In *WWW*, 2012.
- [21] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *MobiSys*, 2011.
- [22] Lenin Ravindranath, Sharad Agarwal, Jitendra Padhye, and Chris Riederer. Procrastinator: Pacing mobile apps' usage of the network. In *MobiSys*, 2014.
- [23] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. Appinsight: Mobile app performance monitoring in the wild. In *OSDI*, 2012.



- 
- [24] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *MobiCom*, 2010.
- [25] B. Veal, K. Li, and D. Lowenthal. New methods for passive estimation of tcp round-trip times. In *PAM*, 2005.
- [26] Fengyuan Xu, Yunxin Liu, Thomas Moscibroda, Ranveer Chandra, Long Jin, Yongguang Zhang, and Qun Li. Optimizing background email sync on smartphones. In *MobiSys*, 2014.
- [27] Xuan Zhou, Zhifeng Zhao, Rongpeng Li, Yifan Zhou, Jacques Palicot, and Honggang Zhang. Understanding the nature of social mobile instant messaging in cellular networks. In *IEEE Communications Letters*, 2014.