The Design and Implementation of a Distributed Photo Sharing Android Application Over Ad-Hoc Wireless

by

HaoQi Li

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2012

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2012

Certified by .....
Li-Shiuan Peh
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor May 21, 2012

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

The Design and Implementation of a Distributed Photo Sharing Android Application Over Ad-Hoc Wireless

by

HaoQi Li

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2012, in partial fulfillment of the
requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We present a distributed photo-sharing Android application, CameraDP, that uses ad-hoc Wifi connections in addition to the common 3G or 4G cellular network connections. The app utilizes the novel DIstributed Programming Layer Over Mobile Agents (DIPLOMA) programming layer to provide a consistent shared memory over a large distributed system of android phones. The success rate and latency of photo saves and photo gets on CameraDP were compared to the numbers generated from CameraCL, a 3G or 4G-only version of the same user interface as CameraDP. Under near-ideal Wifi conditions and only a 1.4% sacrifice in success rate, a 10-phone CameraDP system yielded a 2.6x improvement over a 10 CameraCL phones running on 4G, and the CameraDP system yielded a 16x improvement over CameraCL running on 3G. The methods and results of this research suggest that this new way of designing phone apps using a distributed Wifi network may be very useful when Wifi becomes more robust and smart phone Wifi ranges increase.

Thesis Supervisor: Li-Shiuan Peh
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my thesis advisor Li-Shiuan Peh for giving me this valuable opportunity to learn, my labmates Anirudh Sivaraman and Jason Gao for their clear explanations and debugging assistance, and all the volunteers in the experiments for their endless patience.

Contents

1 Introduction and Motivation **9**

2 Background on DIPLOMA **11**

3 User Interface and Functionality of both Camera Apps **13**

4 CameraDP Android Application **15**

5 CameraCL Android Application **19**

6 Experiments and Code Improvements **21**

6.1 Experiment 1: The discovery of many fatal issues on multiple phones 23

6.1.1 Improvements 24

6.2 Experiment 2 25

6.3 Experiment 3 27

6.4 Experiment 4 27

6.5 Experiment 5 27

6.6 Experiment 6 27

7 Discussion and Conclusion **29**

Chapter 1

Introduction and Motivation

Smart phones rely heavily on the Cloud to carry out extensive computations or get access to abundant storage. Frequent communication with the Cloud via the 3G (HSPA) or 4G (LTE) cellular networks, especially in an area dense with smartphones, can cause an increased latency time, to the immediate frustration of the users. A solution to this problem is to set the phones in a distributed shared-memory network. Given reliable and strong ad-hoc Wifi conditions, requests to nearby phones should be faster on average than 3G or 4G requests to the cloud, improving the user experience.

This is an area of active research and we utilize a recently built consistent shared-memory system over ad-hoc Wifi, DIPLOMA, to test the feasibility of a popular photo-sharing app, Paranomio, on a distributed memory system relying mostly on the ad-hoc WiFi.

We created a stripped-down version of Paranamio that only have two functions: TAKE new photos and GET other photos. In order to quantify the advantage of ad-hoc WiFi, we built two functionally identical apps: CameraDP and CameraCL. CameraDP uses DIPLOMA in the background while CameraCL is the control case where each request is independently sent to the cloud through a 3G or 4G connection.

Chapter 2

Background on DIPLOMA

* DIPLOMA hop cutting (all regions in a chain have to be populated

* width thing

— * server region leader timeout for new leader

* ergion width stuff At this time, we failed to realize the region size should be small enough so that 2 phones anywhere inside 2 adjacent regions could hear each other, not 1 region. Since the leaders must be able to talk to adjancent leaders. If the region widths are set as the limiting range of the phones (20 meters in our case), the only way a DIPLOMA multi-hop would work is if the leaders are exactly 20 meters from each other. If one leaders just moves a little bit, it will fall out of range of the farther neighboring leader and thus breaking DIPLOMA multi-hops. Even though technically the region width should be half of the phone range, with the GPS inaccuracy of the phones, setting the region width to 10 meters is not ideal. If the regions are too small, and the phone's inate GPS inaccuracies varies a lot, we could end up with insensible region allocations. In the worst case, region monotonicity may be broken, e.g. a phone could be erroneously assigned to region 2, between a region 3 and region 4 phone. Without region monotonicity, DIPLOMA multi-hop would not be concrete. When we realized this, we made it possible to *change the region's width* from the UI. The default is still 20 meters.

Chapter 3

User Interface and Functionality of both Camera Apps

From the user's point of view, CameraDP and CameraCL are identical. Both apps allow users to share photos among themselves using their Android phones. The users can take new photos on their phones and request to see the latest photos taken by other phones.

For DIPLOMA But unlike traditional photo sharing apps where each phone functions individually, our experiment assigns the phones into different regions based on their GPS locations and a regions phones collectively save their photos. This implies: a) a new photo is saved on its phones region, not the phone itself and b) a phone can only request the newest photo of a region, not of another individual phone.

In our experiment, six square consecutive regions (0 to 5) about 30 meters wide were created along a stretch of busy road. Ideally, there is no limit to the number of regions as long as a phone can only belong to one region at any time. The size of the region should be as big as possible, but still small enough that phones from any points in the region are within WIFI ranges from each other. The users walked around and pressed buttons on the apps to take new photos or request photos from different regions. With two apps on two different phones, the users synchronized the button presses so that sequences of events for the two app types are similar.

The two different apps are: DiplomaCamera and CloudCamera. Even though

their UI are identical, DiplomaCamera handles the pictures using DIPLOMA while CloudCamera is the control of the experiment, handling all requests with the cloud.

Chapter 4

CameraDP Android Application

* LEADER/JOIN/NONLEADER * discussion on hysteresis * Region width - leader spacing — where is that? * Use CameraSurfaceView The reason for this change was because the *intent/sd card solution only works on Nexus S phones, not Galaxy Notes* phones. In Galaxy Notes, the Mux is killed and restarted before and after the camera intent (because StatusActivity is paused), causing a "Cannot open socketAddress already in use" error: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/8de606e548b4854807ea91a4822b7638a250843c/logcats/galaxy_note_camera_crash.txt#L470` CameraSurfaceView fixes this problem because StatusActivity never has to be paused when taking a photo. Since it works on both types of phones, we used this solution for both.

The DIPLOMA design introduces a leader in each of the regions. The leaders take care of all the requests coming from all phones (clients) in the region. When a phone takes a new photo, it broadcasts the photo data to its leader, where the photo is saved. When a phone requests a photo from Region X, this request is broadcasted to its leader, which in turn uses DIPLOMA to relay the request and receive Region Xs photo data from Xs leader.

The code is divided into three big components:

1. StatusActivity.java for UI and client processing 2. UserApp.java for leader and remote leader functions 3. The DIPLOMA java files: Mux.java, VCoreDaemon.java, DSMLayer.java are unchanged.

StatusActivity.java contains listeners for the button presses that send requests to its region leader and a handler that processes replies from the region leader. Each phone has a unique id based on its IP address that can help a regions leader distinguish the non-leader phones in its region.

Pressing the button that takes a picture triggers that buttons listener to retrieve the photo information from the Camera SurfaceView. The photo data is then put into a packet along with the phones ID, the phones region number, and type of request (UploadPhoto). This packet is serialized inside StatusActivity.java into a UDP broadcast that reaches the leader of the region.

Similarly, pressing a button that requests the newest photo from region X triggers the request buttons listener to get information on the target region number that the user is requesting. A UDP packet consisting of the phones ID, the phones region number, the target region number, and the type of request (DownloadPhoto). Again, StatusActivity.java broadcasts this packet to the leader of the region.

Let Original Leader (OL) be the leader of the phone that made the request.

OLs UserApp.java:handleClientRequest processes the UDP packet by the type of request. In both cases, OL sends a DIPLOMA DSM atom request, along with the additional information from the UDP packet, to the Remote Leader (RL). In the UploadPhoto case, RL is the same as the OL, since new photos are processed locally. In the DownloadPhoto case, RL is the leader of the target region. RLs UserApp.java:handleDSMRequest processes the DSM atom request from the OL. For UploadPhoto, the photo info is saved in RLs DIPLOMA memory as the first element of an ArrayList. (For the experiment, we only save one photo at a time. But theoretically, there is no limit to the number of photos that can be saved.) The reverse happens for DownloadPhoto, where RL retrieves the newest photo from its DIPLOMA memory. In both cases, RL sends a reply back to the OL, arriving at OLs UserApp.java:handleDSMReply which sends a UDP packet containing DIPLOMA latency and a success boolean, and also the photo data in the case of DownloadPhoto, back to the original non-leader.

Finally back to the original phone, its StatusActivity.java handler gets the UDP

reply from OL and logs the replies. In the case of DownloadPhoto, the remote regions newest photo is displayed.

* I dont think the UI of leader showing new photos from nonleaders is important to write about

Leader transitions (its covered by other parts of the paper, right?) ———————— * leader go out of region TODO: add handing-off state? * leader is dead (cloud wait 100 secs to allow new leader) * leader heartbeat to the cloud and to non-leaders

Other things ———————— To avoid confusion and inconsistency of the region numbers, phone buttons only work if the phone is in a LEADER or NONLEADER state. To avoid double-sending a request (and possibly crash the camera surface view), a ProgressDialog is shown until the client has received a leader reply or until a timeout.

Chapter 5

CameraCL Android Application

In CloudCamera, every request is sent to the cloud server. The cloud server keeps a dictionary linking each region to its newest photo. Codewise, CloudCamera only has one file: CameraCloud.java that is analogous to DiplomaCameras StatusActivity.java, but instead of sending UDP packets, CloudCamera sends HTTP post requests.

Chapter 6

Experiments and Code Improvements

We performed a total of 6 data-collection experiments in a span of almost 2 months. Through time, the apps had fewer bugs and more robust code bases. However, it was impossible to fix the most critical issue – the Wifi range and consistency of the phones. The interference of 20 phones carried by 10 people moving simultaneously and randomly made collecting meaningful data infeasible with the current Wifi abilities of the phones. In the final 2 experiments, we resorted to a controlled indoors experiment with minimal Wifi interference and obtained more expected results.

Two pre-experiments were conducted.

Pre-experiment 1: Test DIPLOMA multi-hop and phone WiFi range

Three people, each held a Galaxy Note phone, conducted the experiment outside northeastern entrance of the Stata Center. One person stood at the corner of the entrance while the other two people each stood along a different wall. The phones were held vertically, the outer phones faced the middle phone. There were no obstructions in the path of transmission. We would later find out that the range from this test would be too optimistic for multi-user experiments where users moved around and obstructed each other all the time. By first disabling CameraDP on the middle phone,

we increased the distance between the middle phone to the two outer phones until the outer phones could not consistently complete GET requests, i.e. they were out of each other's WiFi range. This distance was about 20 meters for each leg. We then turned on CameraDP on the middle phone and observed that GET requests between the two outer phones worked again, demonstrating that DIPLOMA multi-hop at least works for three phones.

While outside, we also conducted a 2-phone range test on an open field, where Phone A was stationary and Phone B moved away. When Phone B took a new picture, a hand gesture was shown and Phone A would try to get this newest picture. The GET requests did not work if the two phones stood more than 20 meters apart. However, when we used *ping*, the range of success increased to at least 25 meters.

Pre-experiment 2: Test phone WiFi range at 436 Mass Ave

Two people holding two Galaxy Note phones walked near 436 Mass Ave using CameraDP. Even though all future outdoor experiments were conducted strictly on the eastern sidewalk of Mass Ave, this experiment was also run on both sidewalks. The phones successfully got each other's pictures at opposite ends of Mass Ave.

Outdoor experiment setup:

The volunteers holding the phones were instructed to walk around independently and freely in the valid regions, pressing buttons to take and get pictures at their own will and pace. During runs where volunteers to hold a phone running CameraDP in one hand and a phone running CameraCL in the other hand, the volunteers were instructed to press buttons in the same sequence on both phones.

The volunteers did not know the details of DIPLOMA other than the fact that regions exist. However from the second experiment onwards, the UI improved so that unfavorable circumstances would prevent GET and TAKE buttons from working. Examples of unfavorable circumstances include: walking out of the valid regions, phones in a state other than LEADER or NONLEADER.

If an app hangs for a certain period of time, the Android operating system would

prompt a message saying "xxx is not responding. Would you like to close it? 'Wait' 'Okay'". We instructed the volunteers that they must press "Wait", not "Okay". If 'Okay' is pressed, the CameraDP might crash.

6.1 Experiment 1: The discovery of many fatal issues on multiple phones

Location: 77 Massachusetts Avenue

Date: March 15, 2012

Weather: Drizzling and cold

Phones: 20 Nexus S: 10 running CameraDP, 10 running CameraCL

People: 10 People: each held 1 CameraDP and 1 CameraCL

Regions: 6 linear regions each with width 52 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/81e87e790c13ed3c8c4cd45703528e5216f04ec4`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/camera_diploma_exp1_data`

CameraDP notes: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/camera_diploma_exp1_data/diploma_notes.md`

CloudDP notes: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/camera_diploma_exp1_data/cloud_notes.md`

Before walking to 77 Mass Ave, the servers and the apps were started with Region 0 located at the intersection of Amherst St and Mass Ave and the regions increment northwestwards.

No usable quantitative data was extracted from this experiment due to the frequent crashes on both the CameraDP app and CameraCL. Insufficient and inadequate stress testing beforehand meant that these problems were not discovered until the ex-

periment started. Later analysis revealed that the crashes were mainly due to two reasons: double pressing the TAKE button and an OutOfMemory error caused by the camera interface using up too much VM heap.

The region width was too large, preventing successful communication even for phones in the same region. Compounded to this was a bug that forced users to walk to region 0 whenever the apps crashed. The region assignment based on GPS was observed to be robust. There were no requests generated from outside of region 3.

6.1.1 Improvements

We prevented users from double clicking by using a ProgressDialog to freeze the UI. A boolean flag was introduced to double check that no additional buttons are clicked during the processing time of a previous button.

The OutOfMemory error on Nexus S phones occur when multiple TAKEs were pressed one after the other. The first few TAKEs would behave normally and complete successfully. However around the third to sixth TAKE, the app would crash at the line 'BitmapFactory.decodeByteArray()', which converts the byte array of the image into a bitmap object to be displayed to the user. To work around this problem, we added an additional parameter into the decodeByteArray function so that the byte array is downsampled once every 12th pixel, greatly reducing the memory requirement. In addition, we manually placed system garbage collection calls before the memory-intensive functions. After these two workarounds were coded, we tested the phone by continuously pressing the TAKEs over 100 times, multiple times, and did not observe any crashes.

The Region 0 bug

The bug that causes users to reset from region 0 after every crash was fixed. The bug came about from the logic to prevent inaccuracies in the GPS location. From pre experiment GPS testing, we observed some rare cases where GPS was greatly off for a few seconds. In this case, the the region assignment would unrealistically jump multiple regions. So we put in the logic that unless a new region differs from the old

region by 1, the old region remains the same. The code initializes the region to be -1. During Experiment 1, our logic backfired if the app crashes inside regions 1 or above. Since the app would restart and be set to -1 and GPS would indicate the new region should be 1 or above, the logic prevents the old region to be changed unless the user walks back to region 0, the only region that is 1 away from -1.

After Experiment 1, we removed the GPS check and let the regions to be updated to any new region of any distance away from the old region. Even though we very occasionally notice that phones would jump to an insensible region, the GPS glitch would only last a few seconds, not long enough to cause any concern.

6.2 Experiment 2

Location: 436 Massachusetts Avenue

Date: April 6, 2012

Weather: Sunny and cold

Phones: 20 Nexus S and 20 Galaxy Notes: each with 10 running CameraDP, 10 running CameraCL

People: 10 People: each held 1 CameraDP and 1 CameraCL

Regions: 6 linear regions each with width 52 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/b8a64242d4e6974c74d1c86a`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/experiment2_april_6`

Results: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment2_april_6/log_process_aniru_jason/0411c_meeting.txt`

The server was started in Stata on hermes5.csail.mit.edu (which we will later discover that this server periodically drops connections for security reasons). The experiment was conducted on the eastern sidewalk of 436 Mass Ave to 2 blocks northwestwards. This stretch of road is very busy, filled with restaurants and small

Table 6.1: Experiment 2 4G (Galaxy Notes) Results

	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	80	225	74	345
successes	54	202	15	314
percentage	67.5%	89.7%	20%	91%

Table 6.2: Experiment 2 3G (Nexus S) Results

	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	74	70	128	106
successes	73	62	39	95
percentage	99%	88.5%	30.4%	89.6%

businesses, which possibly caused a lot of Wifi interference with the large number of Wifi hotspots.

Run 1: We handed 2 Nexus S phones to each of the 10 people, 1 Nexus and 1 Galaxy note. When people started to press buttons, the Cloud phone request made the phone hung for over 2-3 minutes or some phones never stopped hanging. This can be seen in the large CameraCL latency numbers in Table 6.4, which are within a minute, but they are averaged over all the runs in this experiment. Still, these numbers are orders of magnitude larger than the rest of latencies in Table 6.4 and Table 6.3.

We decided to restart the servers by connecting a laptop to the strongest free Wifi in the area. Even though we were able to restart the server for run 2, we had to restart the server multiple times in the rest of the runs because the Wifi connection dropped frequently.

Table 6.3: Experiment 2 4G (Galaxy Notes) Latency

	CameraDP	CameraCL
mean	558 ms	837 ms
stdv	991 ms	769 ms
median	205 ms	479 ms

Table 6.4: Experiment 2 3G (Nexus S) Latency

	CameraDP	CameraCL
mean	263 ms	22546 ms
stdv	276 ms	20284 ms
median	205 ms	15557 ms

Run 2: With the server restarted, we started this run with Galaxy Notes phones instead of Nexus S phones and the exact setup.

The cloud requests did improve and were completed within 20 seconds. However, users complained about phones waiting for a long time to JOIN a region. People moved around a lot, sometimes forming occasional pairs or triples (to chat with each other). We do not know how much phones that were next to each interfered with each other's Wifi. It would not be significant since we rarely observed near-range interference indoors.

Run 3: Noticing that phones were stalling on JOIN, as if each time the server has to time out a region leader to let a new leader in, we decided to have stationary leaders. First we positioned individual people in the different regions and observed that they became leaders of their regions. After all the 6 leaders are set up, we had 2 non-leader phones as well as all the leaders pressing buttons (the other 2 people were monitoring the server, restarting it when necessary). The 2 non-leaders could walk around.

There were fewer JOIN request hangs in this run. Later we would discover bugs in DIPLOMA that fixed these hangs.

6.2.1 Improvements

The Wifi was not reliable during this experiment. Even testing pinging between two phones within an arm's distance would fail, most likely due to the Wifi hotspots interference. To fix this, the next experiment was moved back to 77 Mass Ave, a less

busier section of the street.

The region width of 52 meters were too big. So phones within a region could not hear each other (nonleaders and leaders) and leaders in adjacent regions could not hear each other either. In the next experiment the app the region width decreased to 20 meters and we made the UI possible to modify the width during the experiment.

The phones were not at their optimal arrangement for ad-hoc Wifi communication. The volunteers held the phones flat on their palms, which in Experiment 4 was discovered that this horizontal configuration reduced the Wifi range of the phones. In addition, people were facing different directions. Wifi range is reduced greatly behind a person's back. Since the regions were linear and not circular, transmitting through the back was inevitable.

Most of the time half of the regions were unpopulated, which would cause multihop problems in DIPLOMA, since Wifi hops in a chain would only work if there are leaders present in all the regions of the chain.

Since hermes5 would drop periodically, we switched to a more reliable server for future experiments.

6.3 Experiment 3

6.4 Experiment 4

6.5 Experiment 5

6.6 Experiment 6

Chapter 7

Discussion and Conclusion

Bibliography