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 relies on ad-hoc Wifi over the common 3G or 4G cellular network. The app utilizes the novel DIstributed Programming Layer Over Mobile Agents (DIPLOMA) programming abstraction 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 suggests that distributed ad-hoc Wifi network apps may outperform cellular-network-only apps 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

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 smart phones, 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: taking new photos and getting 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.

We conducted six experiments while continuing to improve the codebase. In the experiments, buttons were pressed to take or get pictures. Some experiments were conducted with volunteers walking around outdoors with the phones while pressing the buttons, simulating real-life situations. Other experiments were done in a static setting indoors, where the phones do not move. We recorded and analyzed the number of success of requests and the latency of the responses.

Chapter 2

Background on DIPLOMA

The DIstributed Programming Layer Over Mobile Agents (DIPLOMA) programming abstraction only runs on CameraDP. In order for the experimental section to make sense, a few aspects of DIPLOMA need to be addressed. Before doing so, keep in mind that a paper detailing DIPLOMA is, as of May 2012, in the process of being submitted to a conference. So we won't be able to reference it here, but feel free to look it up online for more details on DIPLOMA.

In DIPLOMA, the phones are assigned into regions based on their GPS location. Given a map of the entire area of interest, the map should be sectioned off into invisible regions so that the phones inside the same region are close to each other in physical distance. Theoretically there is no limit to the number of regions you can make, but it is limited by the number of participating phones. Ideally the length of time of regions being empty should be minimized. So more phones correspond to more regions. Since phones are mobile, if a phone walks out of a region it is assigned to a neighboring region. There should be no overlapping of regions or unassigned phones. Every phone in the region should be technically able to hear from every other phone in the region through ad-hoc Wifi.

Inside each region one of the phones is designated to be the LEADER of the region. All the other phones in the regions are NONLEADERS. If a new phone comes into the region from another region or because it's turned on inside the region, it will try to JOIN the region through an exchange with the LEADER. LEADERS inform the NONLEADERS of its continued existence by broadcasting periodic *I'm alive* heartbeat packets. The LEADER in the region saves photo data that the NONLEADERS (and itself) take. The LEADER is also responsible to communicating with other LEADERS to retrieve and relay a remote region's photo to a NONLEADER in the region. This LEADER-to-LEADER communication is called *multi-hop* because at any step, neighboring LEADERS relay the request, so the request moves from LEADER to LEADER until the destination LEADER is reached.

In addition to these these ad-hoc Wifi requests, LEADERs have the right and privilege to communicate with a cloud server via the cellular 3G or 4G network, just like in CameraCL, but with fewer cloud accesses. In CameraDP, the cloud server acts like a last resort for keeping a region's state consistent. For example, if the LEADER phone leaves the old region to go to a neighboring region, the NONLEADERs of the old region will detect that by the missed LEADER to NONLEADER heartbeat packets. Within a few seconds, the NONLEADERs will randomly choose a new LEADER among themselves. This potential new leader will try to send a packet to the old leader, from which the old leader will give the new leader its states. However if the old LEADER never hears back from the newly elected LEADER or if the old LEADER knows that there are no other phones in the region, then in this case there are no phones that the old LEADER can pass on the state of the region to. So the old LEADER uploads the region's state to the cloud server (much like what happens for every request in CameraCL). Whenever a new leader is formed in a region, under any circumstance, a cloud request is made to ask the cloud server to give it permission to become the new leader (preventing double leaders in a region). If the new leader is approved by the cloud, it also receives from the leader any old states of the region, so that data for that region can remain consistent.

LEADERs also send heartbeats to the cloud server to announce that they are still viable. This way in case that the old LEADER phone turned off or crashed, the cloud server can quickly grant leadership to another phone when it detects multiple skipped cloud heartbeats. Generally CamerDP should try to have as few cloud accesses as possible, to reduce latency. However, the cloud heartbeats should not be made too infrequently that potential new LEADERs of a region with a dead LEADER have to wait for a very long time, during which they would repeated leadership cloud requests. The length of the heartbeat period should be customized according to the app and the phones.

Another aspect of DIPLOMA that is important to the CloudDP app is its regions. As mentioned before, each region have to small enough so that every phone in the region are in range with each other, but also big so that more phones can be contained in one region. Recall that leaders of neighboring regions must be able to communicate with each other. If there is any one region that is missing a LEADER or have an out-of-range LEADER along a linear path of multi-hop, then the DIPLOMA request is broken and the request cannot be completed successfully. In other words, if there is a chain of regions, all regions must have a leader an that leader must be in range with its neighboring leaders. This implies that we must make the region size small enough that two phones anywhere inside two adjacent regions, not one, could hear each other. But how wide should a regions to be? If the region widths are set exactly 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 innate 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. (We found in Experiment 3 that setting the region width down from 20 meters in width to 10 meters in width did not improve the rate of success at all.)

Hysteresis was used around region boundaries in the first two experiments but was not used due to its complications in smaller regions of 20 meters or fewer in Experiment 3 onwards. In the first two experiments where the region widths were 52 meters, we used reserved 10 meters around each region boundary as the hysteresis buffer zone where the region on a phone cannot be changed. At that time we were worried that the GPS would be too imprecise that phones standing near region boundaries might flicker between the two regions very quickly, constantly having to JOIN. Hysteresis buffer zones worked well for large regions. However, having hysteresis at smaller regions, such as 20 meters instead of 52 meters, brought more confusion than benefits. At a majority of points on the grounds of the experiment, there would be some phones assigned to inside a hysteresis region, i.e. there would be about five meters where all phones agree on a region. This relatively large area of hysteresis, caused by GPS inaccuracies and different internal GPS offsets on each phone, led to phones next to each other getting assigned to different regions, sometimes even two regions apart, which would break region monotonicity. Before Experiment 3, we added a hysteresis selector button that would choose different width for hysteresis, but we just set the hysteresis to 0.

Chapter 3

User Interface, Functionality Common to Both Camera Apps

CameraDP and CameraCL, unlike traditional photo sharing apps where each phone functions individually, assigns the phones into different regions based on their GPS locations and a region's leader collectively saves the newest photos for all the phones in the region (it's easy to change the code to save more than one phones). This implies: a) a new photo is saved on its phone's region, not the phone itself and b) a phone can only request the newest photo of a region, not of another individual phone.

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, by pressing the "Take Photo" button and request to see the latest photos taken by other phones by pressing the "Get X Photo" button where "X" correspond to the desired region number. The "Take Photo" button press triggers a TAKE request in CameraDP and the "Get X Photo" button press triggers a GET request. These are the two main requests crucial to our experiments.

The rest of the UI are add-ons to help debugging in the process. Log messages are displayed in the middle. Success rates of TAKEs and GETs are displayed on the bottom of the screen, along with request latency information. The textfield is for setting a new region width. Remember when changing the region width on one phone, all the other phones involved must have the region width changed as well to keep the region assignments consistent among all phones.

The last button is a switch for hysteresis, allowing you to pick different percentages of the region width you want to be applied to be the hysteresis buffer region. If hysteresis is set, the region of the phone cannot be changed inside the hysteresis region, the few meters (based on the hysteresis percentage chosen) around the boundaries of the regions. Hysteresis was set to 0 after Experiment 2 due to its complications discussed in the previous chapter.

For all experiments after the first, after a user presses a TAKE request or GET request button the UI is frozen until the request is finished, preventing double clicking a button and double-sending a request. A double button click and request may cause the camera to be in an inconsistent state, causing the app to crash. There are two levels of disabling the UI, a ProgressDialog and a boolean flag. The ProgressDialog darkens the screen and shows a popup of a spinner, literally freezing the entire UI. It is dismissed when the request is finished. The boolean flag, 'areButtonsEnabled', is independent from the ProgressDialog, serving as another line of defense agains double clicks. Whenever the user clicks on a request button, the global 'areButtonsEnabled' flag is checked and the request only proceeds if the flag is true. As soon as it's determined that the request can proceed, the flag is immediately set to false so that any subsequent button clicks cannot proceed. The flag is set back to true at the completion of the request. The completion of the request could either be receiving the request reply or reaching a timeout.

The camera and photo taking interface is provided by our own CameraSurfaceView class. At the beginning of development, we used the built-in camera image capturing intent, a much simpler way of retrieving pictures. When a user wants to take a picture, the phone is redirected to the Android camera snapshot mode, filling the entire phone screen with a photo preview. After the user takes a picture and is satisfied the phone goes back to the CameraDP or CameraCL app, with the picture shown at the top of the app. However this simple solution only worked on the Nexus S phones. On Galaxy Notes, a *Cannot open socket ... Address already in use* error comes up and causes the app to crash. Somehow, the built-in camera interface works differently on Galaxy Notes by leaving the original CameraDP or CameraCL app in a different state when the phone switches to the snapshot mode. After switching to CameraSurfaceView, we no longer see the error, because the camera preview and photo taking process is directly integrated into the CameraDP or CameraCL app itself, so we never have to leave the app to take a picture. It provided a friendlier UI because the users can see a preview of the picture at any point, directly in the CameraDP or CameraCL app. Since CameraSurfaceView works on both types of phones, we used this solution for both.

The pictures that are generated from TAKEs are both downsampled and compressed in the JPEG format to between 2000 and 6000 bytes before sent to the local LEADERs to be saved. Since Wifi connection is weak, packets containing larger photos are more prone to get dropped. Even though CameraCL does not use Wifi, the images are resized in the same way for fair comparison.

Chapter 4

CameraDP Android Application

CameraDP runs DIPLOMA in the background, which means that in each region there is a LEADER phone while the rest of the phones are NONLEADERs. The communication between the LEADERs and its NONLEADERs are through sending simple UDP packets through Wifi. The communication among LEADERs is done in DIPLOMA, where custom UDP packets are sent through Wifi. A LEADER takes care of all the requests coming from all the NONLEADERs in its region. When a user takes a new photo, the phone broadcasts the photo data to its region's LEADER, where the photo is saved. When a user requests a photo from a remote region, this request is also sent to its leader, which in turn uses DIPLOMA to contact the remote leader.

Besides the LEADER and NONLEADER states of DIPLOMA, a phone can be in other states when it's transitioning between regions. However, TAKE request and GET request buttons are disabled unless the phone has a LEADER or NONLEADER state due to complications of keeping consistency during region transitions, since initially when a phone steps into a new region the phone does not know the node ID/IP address of the new region's LEADER, or if a LEADER exists at all.

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 are tweaked to support CameraDP

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 TAKE request button triggers that buttons listener to retrieve the photo information from the CameraSurfaceView. 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 (the first leg).

Similarly, pressing a GET request button, requesting the newest photo from a remote region, 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 (the first leg).

Through the ad-hoc Wifi, the TAKE or GET request reaches its leader, which is the original leader of the request. The original leaders UserApp.java:handleClientRequest() processes the UDP packet by the type of request. In both *UploadPhoto* and *DownloadPhoto*, the original leader sends a DIPLOMA request, along with the additional information from the UDP packet, to the remote leader (the second leg). In the *UploadPhoto* case, the remote leader is the same as the original leader, since new photos are processed locally. In the *DownloadPhoto* case, the remote leader is the leader of the region of interest. The remote leader's UserApp.java:handleDSMRequest() processes the DIPLOMA request from the original leader. For *UploadPhoto*, the leader saves the new photo's byte array as the first element of the photo array list on the region's DIPLOMA memory. (For the experiment, we only saved the newest photo by overwriting the first element of the ArrayList every time. But it is easy to edit the code to save multiple photos in a region.) The reverse occurs for *DownloadPhoto*, where the remote leader retrieves the newest photo from the photo array in its DIPLOMA memory. In both cases, the remote leader sends a reply back to the original leader (the third leg), arriving at the original leaders UserApp.java:handleDSMReply(). A DIPLOMA request could time out if the original leader does not get a reply from the remote leader within a certain time, in which case the original leader will send a fake self reply with a timedOut field flag switched on. The handleDSMReply() function sends a UDP packet, containing the timed out flag and in the case of *DownloadPhoto* the photo data, back to the original phone that made the request (the fourth leg).

Finally the original phone's StatusActivity.java handler receives the UDP reply from its leader and logs the reply information, including whether DIPLOMA timed out. If the request was a GET, the remote regions newest photo is displayed. For TAKE requests, the leader displays the newest uploaded photo.

Latency is obtained from time stamps taken right before the first leg and right after the fourth leg, it is the total time of all four legs. We also logged the time for just the DIPLOMA portion, legs two and three, but we did not analyze this data.

Chapter 5

CameraCL Android Application

In CameraCL, every request is sent to the cloud server. The cloud server keeps a dictionary linking each region to its newest photo. CameraCL only has one important file: CameraCloud.java that is analogous to DiplomaCameras StatusActivity.java, but instead of sending UDP packets, CloudCamera sends HTTP post requests. Latency is calculated from the difference of the time stamps surrounding the line that executes the http request.

The server returns a status for every request. For TAKE requests, this status indicates if the photo was saved successfully. For GET Requests, a status of failure does not distinguish between a null region or a region without any photo uploads.

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

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 of same type of phone

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 experiment 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

One of the biggest reason for the crashing, on both types of phones and on both CameraDP and CamerCL was due to double clicking a request button that causes an inconsistent state by the different requests triggered in parallel. We fixed this bug by using a ProgressDialog to freeze the UI when a request is still being processed after its button click. In addition, a boolean flag was introduced as a double check to ensure that requests are strictly sequential and buttons must be pressed one at a time.

The OutOfMemory error on Nexus S phones occur when multiple TAKEs were pressed one after the other, which could also have contributed to the frequent crashing during the experiment. 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 this check and let the regions to be updated to any new region, whether the regions might be next to each other or not. 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 of same type of phone

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

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%

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 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

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

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.

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.

The low CameraDP GET success, see Tables 6.1 and 6.2, was a concern and we decided to improve the setup and code for another experiment. Also note from the result tables that CameraDP had higher success rates and lower latencies with 3G than 4G.

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 was 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 multi-hop problems in CameraDP, 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.

We added acks for first and final legs of CameraDP, so that there are 4 chances to make the first leg or final leg succeed. However after the later experiments we found that this addition did not improve results drastically, possibly due to the 4 repeated sends occurred within 1 second, during which time the status of the Wifi would not likely to be changed. Note that these acks had a bug that was not fixed until Experiment 5.

6.3 Experiment 3

Location: 77 Massachusetts Avenue

Date: April 25, 2012

Weather: Sunny

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 of same type of phone

Regions: 6 linear regions each with width 20 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/e22605b1b644aa60aff54a08`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/experiment3_april_25_2011`

Results: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment3_april_25_2011/results.txt`

Table 6.5: Experiment 3 4G (Galaxy Notes) Results

	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	82	111	75	105
successes	22	83	17	58
percentage	26%	74%	22%	55%
latency mean	206 ms	651 ms	1033 ms	268 ms
latency stdv	455 ms	1450 ms	1048 ms	394 ms
latency median	93 ms	495 ms	92 ms	166 ms

Table 6.6: Experiment 3 3G (Nexus S) Results

	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	362	388	470	455
successes	251	388	131	438
percentage	69%	100%	27%	96%
latency mean	900 ms	3749 ms	1858 ms	2704 ms
latency stdv	1328 ms	4134 ms	1355 ms	3175 ms
latency median	259 ms	2567 ms	2169 ms	2264 ms

Table 6.7: Experiment 3 4G (Galaxy Notes) GET Hop Results

	Hop 0	Hop 1	Hop 2	Hop 3+
requests	23	28	21	3
success rate	65%	7%	0%	0%

Table 6.8: Experiment 3 3G (Nexus S) GET Hop Results

	Hop 0	Hop 1	Hop 2	Hop 3+
requests	126	210	92	42
success rate	86%	7%	6%	0%

The first region of this experiment started around the intersection of Amherst St and Mass Ave, the last region ended around 77 Mass Ave. The location is chosen due to its much smaller number of Wifi hotspots compared to busy location last time. MIT Building 5 was the only building on the same side of the street as the experiment. Opposite the street were an MIT undergraduate dorm Maseeh Hall and the MIT Chapel.

In order to have a more stable server, we used a laptop connected to the ethernet in one of the Building 5 classrooms. The connection was stable during the experiment, i.e. no server crashes occurred. One person was watching the server for the entire duration.

There were 4 runs, with the later runs of people concentrated in the first two regions. One trial with Nexus S set the region width to 10 meters instead of 20 meters, but the success rate of GETs did not improve (23%). (see the chapter on DIPLOMA to learn about complications with smaller region sizes.)

The results of the 4 trials are congregated into 6.5 and 6.6. Again, CameraDP had higher success rates on 3G than 4G. CameraDP TAKE failures came from time outs, i.e. requests that do not respond within 6 seconds, which was caused by weak Wifi conditions. For the Nexus S results, 58% of CameraDP GET requests failed in DIPLOMA, due to the leader unable to get a response from the requested remote leader. There are two causes for DIPLOMA level failures, either the leaders were not in range with each other or at least one region in the multi-hop path were absent of a leader. The rest of the CameraDP GET requests failed due to the 6-second time out just like the case in TAKEs. For Galaxy Notes, only 22% of CameraDP GET failures were cause by DIPLOMA.

This is the last experiment where we used multi-hop. The Wifi conditions were not good enough to yield good multi-hop results, see Tables 6.7 and 6.8, which was why we stopped using multi-hop.

Due to bad Wifi connectivity outdoors, the future experiments were run indoors in a much smaller area.

6.4 Experiment 4

Location: Inside Stata, in the lounge closest to the Vassar/Main St intersection

Date: April 30, 2012

Weather: Sunny

Phones: 20 Galaxy Notes: with 10 running CameraDP, 10 running CameraCL

People: 10 People: each held 1 CameraDP and 1 CameraCL of same type of phone

Regions: 6 2x3 or 4 2x2 regions each with width of around 5 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/892b9793536613366b5293ee`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/experiment4_april30`

Results: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment4_april30/results.txt`

This is an indoors experiment with volunteers walking around different 5mx5m regions marked on the ground, manually pressing a button to change their region whenever a new region is entered (GPS turned off on all phones). In the 5 runs, only Run 2 used 3G (from a 3G/4G switch app).

We used 6 regions only in Run 0, else we used a 2x2 4-region setup. No DIPLOMA multi-hops were used, so that means every phone must be in range of every other

Table 6.9: Experiment 4 Run 0 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	87	87	160	159
successes	56	87	61	158
percentage	64%	100%	38%	99%
latency mean	362 ms	871 ms	853 ms	395 ms
latency stdv	652 ms	334 ms	1163 ms	432 ms
latency median	102 ms	831 ms	344 ms	346 ms

Table 6.10: Experiment 4 Run 1 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	154	150	238	351
successes	124	150	166	349
percentage	80%	100%	47%	99%
latency mean	526 ms	909 ms	830 ms	366 ms
latency stdv	965 ms	566ms	909 ms	288 ms
latency median	183 ms	835 ms	638 ms	339 ms

phone, regardless of the region. Since it Run 0 had the largest area of experiment, we would expect its success rates to be lower, but that's only the case for CameraDP TAKEs, and its CameraDP GETs result is the second lowest, only 2% better than the lowest (6.9). In the only 3G run CameraDP had both TAKE and GET success rates above 50% (Table 6.11).

Similar to the previous experiment, TAKE failures were mostly due to timeouts and GET failures were mostly due to a DIPLOMA failure of unable to contact remote regions. This should not have been the case since there were always at least 1 person in each region during the experiment, and leader transitions did not take very long. The first explanation is that Wifi still did not work consistently. Indeed, at one point

Table 6.11: Experiment 4 Run 2 Results

3G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	131	136	279	286
successes	103	136	192	280
percentage	78%	100%	68%	97%
latency mean	364 ms	2302 ms	857 ms	1215 ms
latency stdv	718 ms	762 ms	939 ms	755 ms
latency median	214 ms	2171 ms	599 ms	1080 ms

Table 6.12: Experiment 4 Run 3 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	153	152	189	168
successes	124	152	69	168
percentage	81%	100%	36%	100%
latency mean	772 ms	726 ms	774 ms	347 ms
latency stdv	1172 ms	235 ms	757 ms	338 ms
latency median	163 ms	716 ms	483 ms	298 ms

Table 6.13: Experiment 4 Run 4 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	271	272	370	355
successes	202	272	147	354
percentage	74%	100%	39%	99%
latency mean	695 ms	769 ms	816 ms	361 ms
latency stdv	1188 ms	311 ms	924 ms	316 ms
latency median	146 ms	734 ms	444 ms	324 ms

when users were reporting low success rates, we tried *pinging* between two phones but failed. Another possibility could be the ack bug I introduced pre-Experiment 3.

The success rates were still too low, so we thought maybe a table-top experiment would improve the percentage of success.

6.4.1 Improvements

We fixed the bug that caused an entire region to not be able to GET and TAKE for a period of time. This was due to an error in the ack counter, where I set the reply counter independently from the request counter when in fact the standard correct practice is for the reply counter to be the same as its request counter or at least based on it. My erroneous ack counter was based on a counter in UserApp. During every UserApp reset, the counter would be reinitialized to 0, potentially resending the same counter to the same client of a previous reply. When this client received the reply, it checks against the queue of received reply counters and finds a match, which causes the client to just ignore the reply, thinking that reply were a duplicate.

I made this mistake because it did not occur to me that UserApp does not continue from region to region. The fix was simply changing the construction of the leader reply counter to be based on the request counter. Since all request counters are unique, reply counters would also be unique.

6.5 Experiment 5

Location: Inside Stata, in the lounge closest to the Vassar/Main St intersection

Date: May 6, 2012

Weather: Sunny

Phones: 19 Galaxy Notes: with 10 running CameraDP, 9 running CameraCL

People: 2, controlled experiment Regions: 6 2x3 regions each with width of around 5 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/aeb358fc5a8f887c4193d761`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/experiment5_may6_indoors`

Results: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment5_may6_indoors/results.txt`

Table 6.14: Experiment 5 Run 0 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	55	48	409	378
successes	55	48	404	378
percentage	100%	100%	98%	100%
latency mean	131 ms	515 ms	180 ms	267 ms
latency stdv	61 ms	85 ms	165 ms	142 ms
latency median	91 ms	525 ms	146 ms	215 ms

In this experiment, we placed the phones on the ground near vertically, supported by plastic phone holders on the back. The GPS were turned off again. The phones were placed in a 2x3 regions arrangement with each region the size of 5mx5m. There were either 2, 3, or 4 phones in each region.

Table 6.15: Experiment 5 Run 1 Results

3G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	41	36	180	171
successes	41	36	180	171
percentage	100%	100%	100%	100%
latency mean	132 ms	1960 ms	208 ms	717 ms
latency stdv	61 ms	793 ms	260 ms	727 ms
latency median	104 ms	2362 ms	161 ms	398 ms

In the two runs we would press the same buttons on all phones before moving onto a new button press on all phones. We also switched regions a few times when no other phones are pressing buttons, so that did not have any affect on the results.

We noticed the average latencies for CameraCL requests were under a second when we watched many requests taking a few seconds. During Experiment 6 we would discover the reason for this peculiarity.

6.5.1 Improvements

We added a latency information display on the screen so that in the next experiment we could observe in real time the average latency, median latency, and the newest request's latency. This UI addition helped us find the last bit of information that would finally produce our desired results.

6.6 Experiment 6

Location: Inside Stata, in the lounge closest to the Vassar/Main St intersection

Date: May 6, 2012

Weather: Sunny

Phones: 19 Galaxy Notes: with 10 running CameraDP, 9 running CameraCL

People: 2, controlled experiment Regions: 6 2x3 regions each with width of around 5 meters

Files:

Code version: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/7df1600531f730d03cc8249`

Phone logs and scripts: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/tree/master/experiment6_may12_indoors`

Results: `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment6_may12_indoors/results_diploma.txt`

and `https://github.com/haoqili/Android_DIPLOMA_CAMERA/blob/master/experiment6_may12_indoors/results_cloud.txt`

Table 6.16: Experiment 6 Run 1 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	40	41	242	241
successes	40	41	242	241
percentage	100%	100%	100%	100%
latency mean	146 ms	551 ms	190 ms	254 ms
latency stdv	61 ms	90 ms	144 ms	95 ms
latency median	148 ms	530 ms	162 ms	226 ms

Table 6.17: Experiment 6 Run 2 Results

3G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	20	20	111	94
successes	20	20	105	94
percentage	100%	100%	94%	100%
latency mean	168 ms	2580 ms	225 ms	813 ms
latency stdv	146 ms	539 ms	268 ms	758 ms
latency median	111 ms	2464 ms	161 ms	415 ms

Table 6.18: Experiment 6 Run 3 Results

3G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	40	40	249	242
successes	39	40	242	242
percentage	97%	100%	97%	100%
latency mean	144 ms	2558 ms	217 ms	2279 ms
latency stdv	69 ms	408 ms	261 ms	285 ms
latency median	109 ms	2465 ms	161 ms	2229 ms

In this experiment, the regions were set up similarly as before, each of area 5mx5m. The two inner regions had two phones each, one running CameraDP and the other running CameraCL. The outer regions had four phones each, two running CameraDP and two running CameraCL. The phones this time were placed flat on the stools.

Table 6.19: Experiment 6 Run 4 Results

4G	TAKEs CameraDP	TAKEs CameraCL	GETs CameraDP	GETs CameraCL
total clicks	44	42	240	240
successes	44	42	240	240
percentage	100%	100%	100%	100%
latency mean	144 ms	546 ms	178 ms	469 ms
latency stdv	84 ms	75 ms	116 ms	51 ms
latency median	107 ms	534 ms	159 ms	469 ms

We forgot to turn off the GPS at the beginning and one of the phones during run 2 got a GPS fix, messing up the results. Then we proceeded to turn off all the phone's GPS.

The cloud accesses consisted of leader to cloud server heartbeats and the few initial leadership grants from the cloud. In run 3, there are 83 cloud accesses, corresponding to 1 cloud access per 3.5 TAKE or GET requests. In run 4 there are 62 cloud accesses, corresponding to 1 cloud access per 4.6 TAKE or GET requests. The cloud heartbeats were made once every 2 minutes on every LEADER.

The Warm-Up Effect: The sequence of button presses for the first two runs were as follows: TAKE pictures on every phone one by one, then on each phone GET pictures from all the regions (0-5). We were pressing the 6 GET requests on each phone within a second of each other. As we moved from phone to phone, we observed the strange behavior that the first GET request on each phone would be many times slower than the rest of the GET requests, i.e. the GET request latency decreased drastically after the first GET of a batch of GETs. At the end of run 2, we realized that if wait a while between GET requests, the decreased latency effect was not observed. This is a warm-up effect perhaps due to some component(s) in the phone not having to restart on latter GET presses, because the component(s) are already warmed-up.

So for runs 3 and 4 we avoided the warm-up effect by pressing buttons in this sequence: first TAKE pictures on every phone, then GET region 0 on all phones, one by one, then GET region 1 on all phones, etc. So between each addition GET request on a single phone, we'll have waited about a minute, more than enough to make the warm-up effect disappear. You can see the difference that the warm-up effect makes by comparing the decreased CameraCL GET latencies in Tables 6.16 and 6.17 to the normal latencies in Tables 6.18 and 6.19.

Since in the real world users would not be constantly making requests within seconds of each other, the more realistic data are from runs 3 and 4, which omit the warm-up effect.

Without the warm-up effect, our data results are even more promising, showing an average of a 2.6x improvement in 4G (6.19) with only an 1.4% decrease in success rate and a 16x improvement in latency over 3G (6.18) without any decrease in success rate!

The results are not surprising since

Chapter 7

Discussion and Conclusion

We discussed a photo app, CameraDP, that uses a distributed ad-hoc network abstraction to carry out user's requests and compared its success rate and latency times to identical app, CameraCL, that relies only on the 3G or 4G cellular network. In general the CameraDP app had much lower latencies than the CameraCL app but the success rates were not favorable when used outdoors.

The promising results of the indoors experiment shed light on how much a distributed ad-hoc app can improve the latency on all the phones without much change in success rates. Even though our outdoor experiments also showed better latency, the success rate suffered due to the current state of ad-hoc Wifi on phones. If one day the Wifi ranges on the smart phones are increased, the the strength of Wifi is increased, and smart phones become ubiquitous, then it is very logical for phones to rely on each other, relieving the traffic on the cellular networks.