Machine Problem 2: Let's *Share*

Post Date 10/06/2009 Due Date 11/06/2009

In MP2, we will build a P2P ***Share*** App for Android, where several android phones share files with each other using Internet over WiFi transparently. Transparency means that a user can download a file without knowing which phone he or she downloads from. **Share** app will find peers, who are online and holding copies, and select some peers to download the file on user's behalf. If there are multiple replicas of the same file, i.e., the same file resides in multiple peers, downloading peers can be selected randomly or based on some load balance rule. It will be left up to you. In order to accommodate different levels of android programming proficiency, we have designed MP2 with 2 parts, required part and bonus part. Balance your workload, and choose wisely whether you want bonus points from bonus part, which becomes useful in cases you are performing unsatisfactorily in homework and exams.

Please pick up G1 Dev phones after Oct. 20. See lease and pickup instructions on the course website.

Assignment Description – Required part

- Required part (100pt)

Implement an Android App **Share**, which enables mobile phones to share files over Internet from phone GUI. Programming and debugging can be done in emulator, while you will demo in real phones.

In **Share** App, we have both **Share** phone clients and the membership server (MS) as in MP1. Besides maintaining a directory of online members, MS also maintains the global directory of shared files from those peers. An *online* file is a file, whose copies can be found in at least one registered phone. Global file directory lists all online files with the following information: file name, and phone locations of replicas (IP address, port), etc. MS is only responsible for maintaining directory and performing directory-related operation, such as inserting and deleting an entry in the global file directory. Files themselves are kept by peers and transferred directly in peer-to-peer fashion among peers via sockets. This approach is similar to Napster and it reduces the workload on MS.

Required functions for each **share** phone client are **login**, **insert (file)**, **delete (file)**, **list (files)**, **download (file), search, logout and log**. The notation of each function has the following format: **function [parameters @ issuing phone/user]:** detailed comments follow.

Login [peername]: should register this phone and all current files to MS.
Insert [fileA @ user1]: should add fileA from user1's phone to global file directory. After this step, fileA is visible for other peers to view if other peers call 'list' function.
GetFileList [@user1]: should retrieve the list of online (active) files on all register peers from MS to user1.
Delete [fileA @ user1]: should delete fileA from user1's phone. Note that this function should also *update* the global file directory list on MS.
Download [fileA @ user1]: should download online fileA to user1's phone. Upon downloading completion, user1's phone has a replica of fileA.

Search [file name N @ user1]: should query for files with exact name N, ignoring the suffix. The result of the 'search' function will be a list of peers (and their IP/port information), which hold the searched file 'N'. If file 'N' is not found in the global directory, the 'search' function returns error message "No File 'N' Found".
Logout [peername]: logout from **Share**. Its removes this peer from membership list and removes all the files associated with this peer from the global file directory. *It also transfers the log file of the monitoring information back to MS.*
Log [@user1]: should log all the following events taking place in the system with ***timestamp*** on the background. Log function automatically starts after user1 logs in and stops right after user1 logs out. There is no need for user1 to explicitly call this function. MS maintains a log file per user.

Function	Related Parameters
Login	IP address, port number, The number of file inserted implicitly.
Insert	Filename, file size
GetFileList	Total number of file retrieved
Download	Filename, file size
	A list of Peer name (IP), number of bytes transferred, download status (completed or failed) with timestamp. A peer may appear several times in this list if it has been selected multiple times to download the same file
Search	Result (No file found; or the number of online peers with a replica)
Logout	The number of file removed from the global file directory, the size of log.

Comments about inserting process
There is an implicit inserting process when a phone registers with the MS after it starts or recovers from failure. It means the phone automatically inserts all its local files to the global directory, which have been either inserted or downloaded before.
Example: During login, do a) register the user with MS; b) register all the local files to MS's global file directory.

Comments about downloading process
To make the required part simple, a file A is downloaded from a remote phone, if a complete copy is located in the remote phone. After File A is completely downloaded, the global file directory is updated with the new holding peer for the file A. If the download fails in the middle due to peer leave or socket error, then the file should be re-downloaded from the same remote phone or another phone peer.

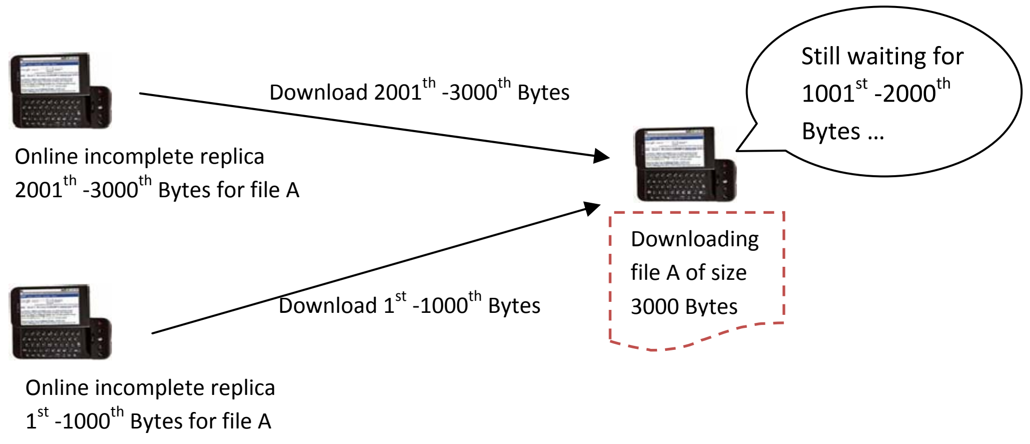
Comments about inserted contents
Inserted files must be images since we can easily verify that files are exchanged successfully and correctly among phones by displaying the pictures via `gallery' app of android phone. We will first use the camera application on the phone to capture images, then share the images among phone peers using **Share** app we program and finally use the gallery app to verify the images are transferred correctly. Since all the images the camera captures are stored in directory */sdcard/dcim/camera/,* users will insert files from this folder. You probably need to rename the image files with more meaningful names for the global file directory entry. This is because android media files, named by date and time, are hard to search. Downloaded files are placed in the app folder you create for **Share**, */sdcard/letusshare/,* for example, rather than the camera folder. This eliminates interference between **Share** and camera applications on the phones.

Example: 1) take picture with the phone and the image file is stored in */sdcard/dcim/camera/* with name 2009-10-01 21.44.55.jpg*;* 2) insert this file by copying it to directory /sdcard/letusshare/ and renaming it to fun-office.jpg; c) add this file entry to global file directory in MS.

- Bonus part (30pt)

For the bonus part, we will implement a more realistic P2P system, where both long and short files are shared and managed via chunks, fragments of information. It means a long file is segmented into multiple chunks. Furthermore, a user can select to download a file, even when all the active peers hold only part of chunks of this file at the time of selection. Modifications to the required part are amplified as follows.

A file is present with the following two states at a phone: (a) incomplete (being downloaded); (b) complete. A long file, segmented into chunks, is downloaded from multiple peers with incomplete or complete copies of the file. This compensates for peer churn where peers come up and down frequently. A long file is downloaded from multiple peers sequentially with missing segments being (re)downloaded to reserve communication resources. Pathological scenarios may happen that in the middle of downloading, all the peers leave the system. **Share** delays downloading process, until there is an online file replica, either complete or incomplete. You will demo downloading a file from multiple peers with complete or incomplete copies, due to peer leave and join.


Download 2001th -3000th Bytes
Online incomplete replica 2001th -3000th Bytes for file A
Still waiting for 1001st -2000th Bytes …
Downloading file A of size 3000 Bytes
Download 1st -1000th Bytes
Online incomplete replica 1st -1000th Bytes for file A

Suggested files are videos taken by camera application, which are located at */sdcard/dcim/camera/*. To the best of my knowledge, **the Dev phone with SDK1.1 has no support for video format** and it is awkward for us to check arbitrary files with large size are exchanged correctly. So, **you need to flash phone OS to SDK1.5 or 1.6 according to Comment (10)** in order to complete the bonus part.

Comments (Tips):

Use your android dev phones for the first time.

1) Register G1 dev phones without SIM card, when you first use your phone. Please refer to the post http://e.ntangled.net/content/howto-set-android-dev-phone-without-sim-card
2) Setup UIwpa2 network for G1 dev phones (root access is required)

Step 1: enable pull and push the system files

```
adb –d shell
su
cd /data/misc/
chmod 777 wifi
cd wifi/
chmod 777 wpa_supplicant.conf
```

Step 2: open another command terminal and pull the wifi configuration file into your desktop

```
adb pull /data/misc/wifi/wpa_supplicant.conf .
```

Step 3: modify wpa_supplicant.con with the information for UIwpa2. Right now we ignore security issue. Be careful not to let other see your AD password according to this instruction by protecting your phone well.

```
network={
	ssid="UIwpa2"
	key_mgmt=WPA-EAP
	pairwise=CCMP
	group=CCMP
	eap=PEAP
	phase2="auth=MSCHAPV2"
	identity="your netid"
	password="your ad password"
	priority=1			# position of this entry in the saved wifi list
}
```

Step 4: push the new configuration file to phones

```
adb push wpa_supplicant.conf /data/misc/wifi/wpa_supplicant.conf
```

Now you are able to use UIwpa2 without manually authenticating in UIUCnet webpage.

Programming related comments

3) Exception handling
   Running in the real phones, applications may be paused and killed by Android to reserve system resources. Moreover, wireless connections are shut down frequently to save battery when the network communication is silent for a period of time. Therefore, a phone peer using WiFi may show a high churn rate. It is important and critical to handle various kinds of exception to deal with the peer uncertainty.
4) Setup your device for debugging procedure
   http://developer.android.com/guide/developing/device.html
5) By default, files created using the openFileOutput method are private to the calling application-a different application that tries to access these files will be denied access. The standard way to share a file between applications is to use a *Content Provider*. Alternatively, you can specify either Context.MODE_WORLD_READABLE or Context.MODE_WORLD_WRITEABLE when creating the output files to make them available in other applications:
6) File browser source code to manipulate files for your reference
   http://www.freeware4android.com/2008/08/10/android-file-browser-20.html

7) Download a free file browser/manger/explorer application from Google market. Using the tool, we can view the file directory of phones and check that your file operations are correctly implemented. Also we will use this tool to view images and videos under your app folder, which is supposed to be outside the camera app.
8) For those who have difficulty with MP1, please start with MP1 sample code from course website.
9) You can install applications into phone by either running or debugging the apps with phone device in Eclipse, or using adb as in Slide 30 of tutorial.pdf.
10) Our Google Dev phones are loaded with an early firmware, which means that we cannot compile our apps using Android SDK 1.5. If you do not want to flash the dev phones, just compile your apps using SDK 1.1. The required part of MP can be executed under the early firmware and you do not need to flash the dev phones. However, there are several reasons to flash the Dev phones with SDK 1.5 or 1.6 as early as possible. The first reason is that you desire to use advanced features of the later SDK in MP3 or other fun apps. The second is that you want to earn points from the bonus part. If you decide to flash your phone, follow the instructions carefully so that the phones you have loaned under your responsibility are not jeopardized. Also be sure you have the root access. The whole flashing process takes about 20 minutes.

What to submit for MP2

Similar to MP1, each group should:

- Place Java source code of your ***share*** program in mp2_submitted/ directory of your SVN. See MP1 for instructions.
  - Documentation of each major source code
  - Documentation of each function in your Java file as MP1
- Send Report (README file) **pdf** file to TA
- Group representative(s) comes to the lab 0216 at the scheduled time. Please sign-up at http://spreadsheets.google.com/ccc?key=0AmVHs8Rfyiz8dEs2MjFTUGpQYk43U1VEakNUMmFsQmc&hl=en for demo time between 4 and 6pm on Friday, Nov. 6th and show your phone demo of the required programs in 0216 Siebel Center.

The demonstration of the whole assignment for one group should take no more than 15 minutes. Set up applications on dev phones and be prepared to show us your source code when being asked.

Evaluation Scenarios of the Assignment (100+30pt)

Example of a sequence of functions being tested for the required part (100pt)

- **Login** 3 Peers; shoot picture X and picture Y from phone A and phone B separately
- **Insert** image X from phone A
- **Insert** image Y from phone B
- **Download** image X to phone B, until completion; show X
- **Download** image X and Y to phone C until completion; show X and Y on phone C
- **Delete** image Y from phone C
- **Download** image Y to phone A; show Y on Phone A
- **Disable** phone A;
- **Delete** image Y from phone B
- **Logout** B **, logout** C, **logout** A

- **Show logs in MS of all phones A, B, C**

Example application logistic for the bonus part (30pt), similar to requirement part but with video

- **In the middle of downloading** a video, we will ask to disable WiFi of phone A.

Please see the course website for detailed point assignment.

Requirement for REPORT

In the Documentation README file, you should answer the following questions. Email this document to TA at huang23@illinois.edu, and also store it in your group SVN directory. Write in the first page, your group number and the members of the group.

- Overview
  - Describe UI interface design at a peer node
- For Global File Directory management component
  - Describe the protocol design with message format and data structure
    - Insert; Delete; GetFileList; Search; login, logout
  - Describe the algorithm to deal with peer churn
- For file download component
  - Describe the download protocol design with message format
  - Describe the peer selection rule
  - For the bonus part - Describe the design of chunk-based downloading scheme.
  - For the bonus part - Describe the algorithm to deal with peer churn in terms of chunk management.

Feel free to post questions about any of these topics on the newsgroup or to come by TA office hours at Monday 2:00pm-3:00pm or Thursday 3:15pm-4:15pm in room 0207 SC.