Peer-to-Peer Content Distribution Using Smartphones

ECSE 476 - GR6_SP1

Patrick Desmarais (260329253)

Iain Macdonald (260270134)

Guillaume Viger (260309396)

Supervisor: Professor Michael Rabbat

Due December 6, 2011

Abstract

The popularity of mobile devices has caused applications and infrastructure to advance quickly. Many mobile devices require a persistent network connection; content distribution on mobile phones is contingent on Internet access, making peer-to-peer mobile networking a compelling access mechanism. The first phase of our project is a feasibility investigation and the architecture of an application for sharing content between mobile devices. The application must manage temporally changing network topologies, use network protocols, and be user-friendly. Further phases will focus on optimizations of battery life and file transfer rate. In this report we present an architecture, and discuss the underlying technologies.

Table of Contents

5 Introduction
7 Background
9 Use Cases and Architecture
9 Use Cases
9 Use Case 1
11 Use Case 2
12 High Level Architecture
13 Low Level Architecture
13 Activities
19 Services
21 Content Providers / Local Storage
24 Broadcast Receivers / Emitters
26 Overview
28 Network Protocol
28 NetworkAdapter Interface
30 BluetoothAdapter Class
34 Extensions
37 Development Stages
40 Validation
42 Environmental/Social Impact
44 Conclusion
45 References
49 Appendix
49 Prototype Code
49 Speed Calculation

Table of Figures

Figure 1	Page 12	A high-level architecture of the application, based on the model-view-adapter design pattern.
Figure 2	Page 14	The activities used by the application, which include the “My Files” activity, and the “Network Files” activity.
Figure 3	Page 15	A prototype version of the “My Files” activity.
Figure 4	Page 16	Prototype versions of the “Network Files” activity, which includes browsing a list of network files, selecting a file, and filtering the list of files.
Figure 5	Page 19	A low-level diagram of the services required by the application.
Figure 6	Page 22	A diagram of the content providers in the application, including the content providers, and contextual usage information.
Figure 7	Page 24	A sample storage schema for the file information content provider.
Figure 8	Page 25	A diagram of the broadcast emitters in the application.
Figure 9	Page 27	A diagram of the complete low-level architecture of the application.
Figure 10	Page 31	A diagram of the Bluetooth network topology (piconets).
Figure 11	Page 38	A diagram of the development of user stories.
Figure 12	Page 39	A diagram of the user stories and associated tasks.
Figure 13	Page 41	An high-level outline of the intended validation scenarios for the application.
Figure 14	Page 51	A calculation of the transfer times of several files over Bluetooth.

Introduction

The world is becoming a more highly connected place. New technology like the Internet and cell phones supply people with a constant stream of information. As individuals migrate their social lives to Facebook, their movie rentals to Netflix, and their office work to Google Docs, many of these same users are upgrading their cellular telephones to smartphones. The smartphone market has exploded in size, as smartphone sales continue to accelerate through the end of 2011 [1]. Accompanying this explosion in hardware sales is an equivalent boom in software sales, with a transition in pricing models from one-time large purchases to many smaller purchases through an application store. The network infrastructure which supports these devices is also expanding rapidly, from 2G to 3G to 4G. However, there are still some places where wireless access is not available, and where these devices experience limited capabilities. Improvements in mobile routing suggest that some features of the Internet may be possible without infrastructure by using opportunistic social routing techniques [2]. Our project involves an investigation into file transfer in the absence of wireless infrastructure using peer-to-peer networks between mobile devices.

There are two common cases when a smartphone user might be left without a mobile broadband connection: when they are in a dead zone, and when the provider is experiencing a service outage. While coverage is improving

in many areas, especially cities, there are still places, such as underground malls and subway stations, where mobile network connectivity is limited [3]. Additionally, service outages can affect millions of users [4].

Many mobile applications rely on network connectivity, and in the absence of a connection they become unusable. More specifically, the content distribution mechanisms these applications rely on are unavailable offline. Recent research in the area of infrastructure-less mobile routing suggests that peer-to-peer mobile networks can be used to distribute content between mobile phones [2]. These methods can achieve reasonable bandwidth, but suffer from high latency. Our project seeks to provide a means to connect users through ad-hoc networks for the purpose of exchanging content between mobile devices without relying on a network infrastructure. The proposed application will provide a content distribution mechanism in the absence of network connectivity.

Background

In this first stage of our project, we researched existing technologies related to this problem, looked at similar topics, and studied software APIs and hardware specifications for the technologies we considered using in our application.

Some of the existing applications we looked at include Bump™, PhotoSync™, and QwikCards™. Bump™ is an application for sharing content between mobile devices over a wireless network [5]. PhotoSync™ is an application for wireless synchronization of an image library between multiple mobile devices, such as the iPhone, iPad, and iPod Touch [6]. QwikCards™ is an application which attempts to mimic index cards, with the ability to synchronize a user's cards over Bluetooth on Apple devices [7]. We investigated what these applications do, and tried to deduce their architecture, implementation details, and network protocols based on documentation available on the project websites. After a thorough review of available applications, we were not able to find any applications which perform the task we are attempting to implement.

Investigating the topic, we found a few helpful papers looking at routing traffic through temporally changing mobile networks. We consulted three of these papers in depth. Hui et.al. outline an approach for routing traffic through temporally changing networks, called BUBBLE [2]. Ioannidis et. al. demonstrate a scalable, bandwidth optimal content dissemination mechanism over peer-to-peer

connections in mobile networks [8]. Ioannidis et. al. also studied the practice of limiting bandwidth on mobile social networks, while maintaining performance [9]. We also looked at several other academic papers which are cited throughout this text.

In looking at candidate hardware and software platforms, we consulted the Apple iOS Dev Center, and the Android development website [10], [11]. We chose to develop on the Android platform because of its cross-platform availability and our access to Android phones. We then spent more time looking at Android resources, including the Android API and the 'Dev Guide' [12], [13]. The details of Android development are incorporated into the Architecture section of our report. Outside of the Android API, we looked at information about wireless data transfer technologies such as Wi-Fi Direct, and Bluetooth [14], [15]. We also consulted a few books for more information on Bluetooth [16] [17] [18] [19].

Use Cases and Architecture

The implementation strategy we adopted was motivated by an investigation of the use cases of our application. Our goal in developing these use cases was to make them as trivial as possible, with the primary goal being to permit transfer of files from one user to another. We concluded that some privacy mechanism was necessary to preclude a user's ability to download any file on any other user's phone. We also realized that each device in the created network would need to advertise its list of available files to other devices which led us to formulate an application layer protocol for our software.

Use Cases

These use cases focus on the actual usage of the application, and assume that the users have already launched the application and that the device's network transceivers are enabled. In this communication model, only public files may be exchanged between peers to protect each user's privacy.

Use Case 1 (two peers)

This first use case ignores the fact that there may be multiple peers on the network, and focuses only on messages sent between two peers: User 1 and User 2. Square brackets are used to indicate network protocol events i.e. events that

are dealt with by the application and not the device owner. The ellipses are used to indicate that the user may continue to use the application before shutting it down.

User 1 **(Downloader)**	**User 2** **(Uploader)**
Confirms Bluetooth activation at startup	
[Discovers nearby devices]	
[Requests a connection]	
	[Accepts connection]
[Requests list of public files]	
	[Returns list of public files]
Browses list	
Requests file F	
	[Sends file F]
Receives file F	
...	…
[Closes connection]	
	[Closes connection]

Use Case 2 (per user)

The second use case focuses on a single user's interaction with the system, and the user's ability to change the privacy setting of a specific file.

User
Browses list of local files Changes the availability of a file to public or private

These two brief use cases summarize the basic idea behind our application clearly and concisely. The application maintains a list of files on the mobile device. For each of these files, the user may indicate if the file should be shared publicly, or not. A list of all available files is distributed to each member of the ad-hoc network. Other users download files available on the created network by having their phone download it from the appropriate device.

Analysis of these two use cases, and a review of the capabilities and features of the Android operating system shaped the high-level architecture of our application.

High-Level Architecture

The high-level architecture follows the model-view-adapter (MVA) pattern also known as mediating-controller MVC, which is similar to the model-view-

controller (MVC) pattern with the added restriction that there is no interaction between the *view* module and the *model* module. The *view* handles the display of information to the user. The *model* handles interactions with files stored on the device and on remote devices on the network. The mediating *controller* manages interactions between the *model* and the *view*. Development on the Android platform lends itself to this design pattern, due to the APIs and tools made available to programmers.

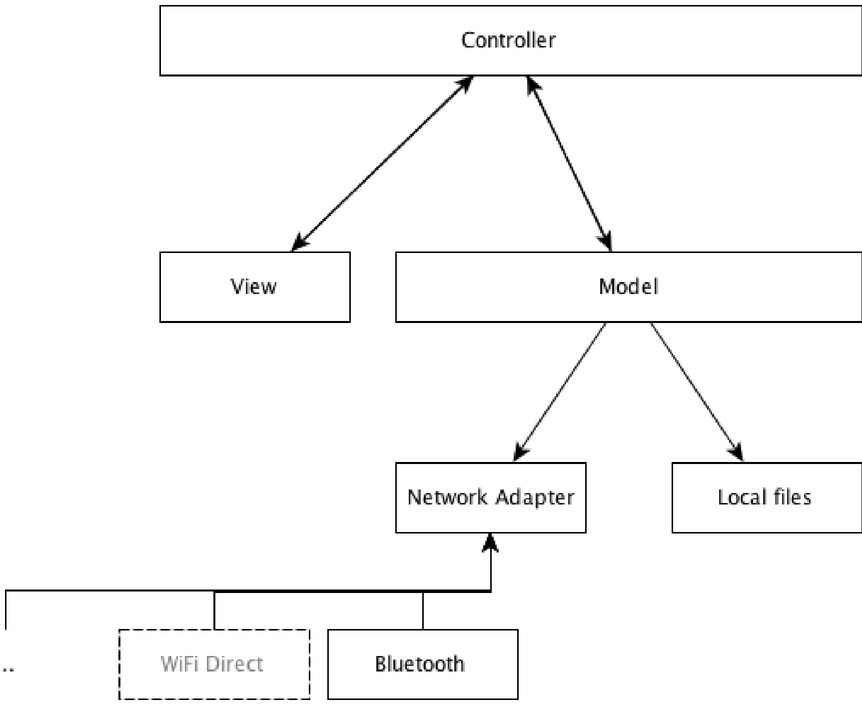

Figure 1: High-level architecture based on the model-view-adapter pattern.

The network adapter layer in the architecture is used to abstract away the underlying network technology used. This abstraction will allow the application to easily switch between BlueTooth, WiFi Direct, or any other data

communication protocol.

With the high-level architecture laid out, a further exploration of each module in the system is possible. In our description of the low-level architecture, we follow patterns established by the Android platform, and highlight the mechanisms available for developing applications on Android.

Low-Level Architecture

The description of our low-level architecture follows the four Android application fundamentals - activities, services, content providers, and broadcast receivers/emitters. Information about the specific implementation for our application accompanies the description of each of these fundamental components.

Activities

An activity is a screen with accompanying user interfaces [20]. An application typically consists of multiple activities. Each activity offers a user interface for certain functionality and this user interface is usually defined in a separate XML file. The activities represent part of the *controller* in the MVA architecture as well as the *view* module.

Our application contains two main activities, which are enclosed in dashed lines in Figure 2. Within each of these activities, there are user interfaces, represented by yellow circles.

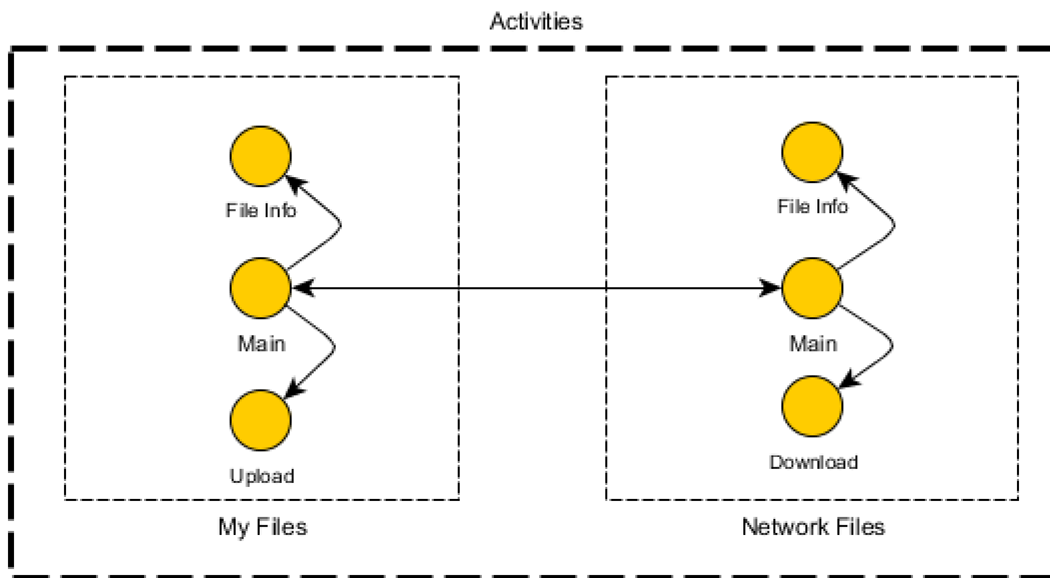

Figure 2: The activities in our application. There are two activities: “My Files” and “Network Files,” each with user interfaces, represented by yellow circles.

We outline the functionality of each of the activities, My Files, and Network Files, below.

1. My Files
    a. Browse by file name, file type and file size.
        i. View additional information about a file, including previous owner, original owner, and last modified date.
    b. Change privacy status of a file
    c. Search for files by file name

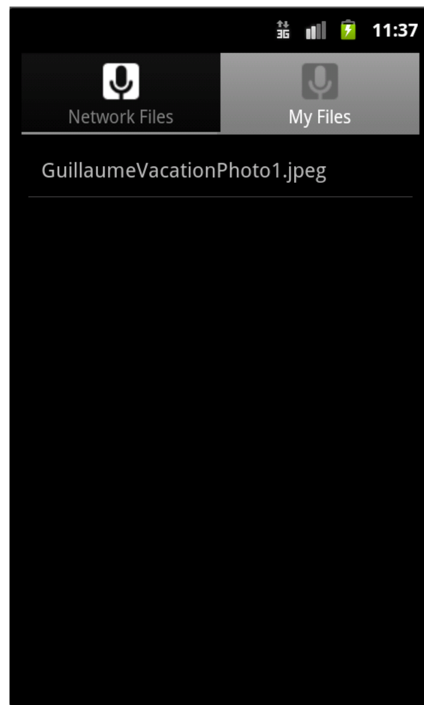

Figure 3: A prototype of the My Files activity.

2. Network Files
    a. Browse the list of files, sorting by name, type, or owner.
        i. View additional information about the file, including the original owner, the size, and if the file has already been downloaded.
        ii. Unique identification of the files using a Java-supported checksum (such as MD5, SHA-1, SHA-256, or others). [21]
    b. Search for files by file name
    c. Download files available in the local network
        i. Downloads proceed one file at a time.

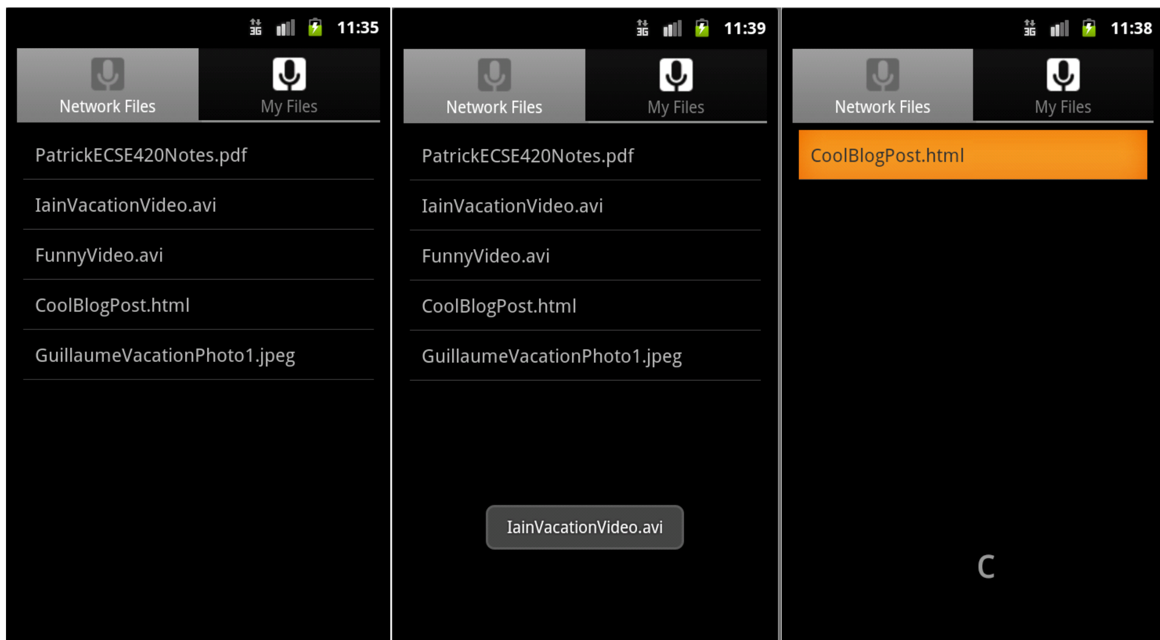

Figure 4: Prototype examples of the Network Files activity, including, from left to right, browsing, selecting a file, and filtering the list.

In later phases of the project, we would like to expand the functionality of each of these activities. Future expansions could include an extension of the privacy system to include more statuses than simply public and private. For example, a user could set certain files to be shareable only with one user, or with a group of users, such as the user's family. Using a BlueTooth ID to address book translation mechanism, such restrictions would be possible. In this enhanced privacy system, a user could configure the application to ask for permission each time a new, or unknown user requests a download from the uploading user. The system could remember these users for a specific period of time, so that the user is not prompted each time one of their friends requests a file.

In addition to these privacy settings, we have several other features we would like to build into the system, time permitting.

These more advanced features include:

1. Changing the sharing settings.
    a. Changing the secure connection level used in BlueTooth communication [15].
    b. Asking the user if operations should proceed when the battery level drops below a certain percentage.
    c. Swapping the network layer protocol in use.
        - BlueTooth
        - WiFi Direct
2. Searching through network file lists by expanding upon the existing search functionality.
    a. Searching by file category, and owner.
    b. Implementing a more advanced search function. The current search function is a simple rooted regular expression with no special characters. An improvement could implement something more powerful, searching for keys within the name of a file.
3. Improving the process of updating file privacy status.
    a. Enabling bulk privacy status updates by flagging files, or searching for files and setting all to a certain privacy status.
4. Managing current downloads
    a. Pausing and resuming downloads

b. Cancelling downloads
c. Retrieving the Internet location of the file, so download can be deferred until the user is connected to broadband.
d. If a download is interrupted, searching for new sources of the file on the peer network.

5. Maintaining and browsing upload/download statistics.
    a. Implementing a user ranking system similar to the one used in BitTorrent.
    b. Keeping track of common peers when choosing which to connect to in the network, such that you can see what's new in peers that you don't share with often as hinted by [8].
6. Pushing content from user to user, rather than only pulling content, if possible.

Services

In the Android platform, a service is a background component without a user interface [22]. Services are like daemons, with the exception that they need not be started on system boot. A service performs tasks that do not require user input, such as connecting to a network. The user can do other things while a service runs in the background.

Figure 5 shows a diagram of the services available in our application. Each service is represented by a yellow round-cornered square. Yellow rectangles indicate categories.

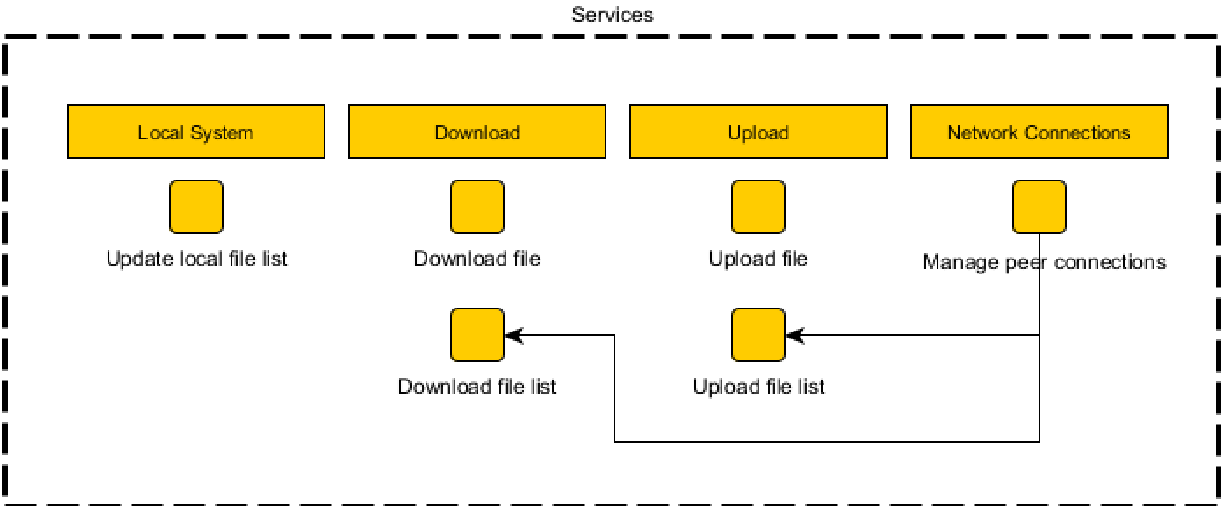

Figure 5: A low-level diagram of the services available in our application.

These services and the functions they perform are described in more detail below.

1. Manage peer connections
    a. Discovers new peers and initiates connections with them so they may join the local network.
    b. Maintains a list of peers active in the network.
    c. Closes connections with peers who leave, or peers who timeout.
2. Upload file
    a. Sends a file from the current device to a remote device.
3. Upload file list
    a. Sends a list of files available on the local device to a remote device.
4. Download file
    a. Receives a file from a remote device and stores it on the current

device.

5. Download file list
    a. Receives a list of files available on the local network, for browsing on the local device.
6. Update local file list
    a. Updates the list of files displayed on the My File interface to reflect the files stored on the device's file system.
    b. This service may be triggered using a refresh button, or when the user switches to the My File activity.

Additional services which may be added in a later phase of the project might include a battery life and consumption monitor, and a service for pushing data to other devices.

Content Providers / Local storage

A content provider is a local data storage mechanism. Content providers in Android are implemented as local SQLite databases, private application-specific files or files on the filesystem [23]. Content providers are used by applications to persist important information within an application's execution, and between application executions

Figure 6 contains a description of the content providers in our application. Each content provider is represented by a yellow round-cornered square. Yellow rectangles indicate contextual usage of content providers. We differentiate

between *My files* and the *File System* because we will create a list of the files on the file system, and store additional information about each file in this database.

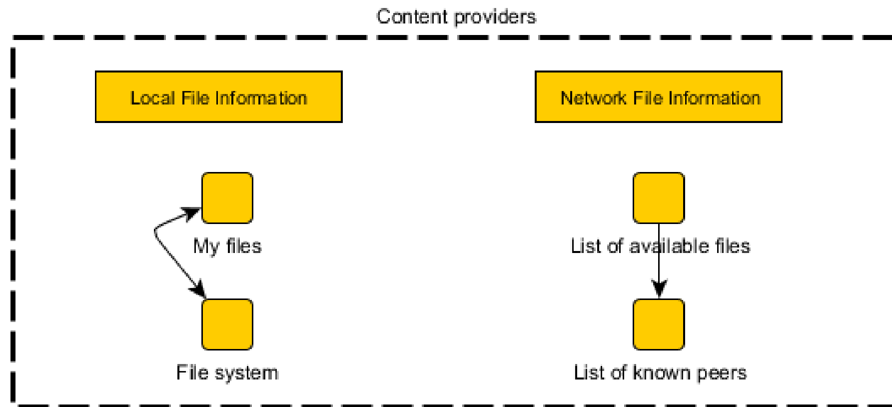

Figure 6: Content providers in our application. Yellow, round-cornered squares represent content providers, and yellow rectangles represent the context containing these content providers.

The specific functions and responsibilities of these content providers are laid out below.

1. My Files
    a. Files may be public, or private, or have additional privacy settings in later phases of the project. The privacy status of each file may be changed in the My Files activity.
    b. Additional file information:
        i. File name
        ii. Last modified date
        iii. File size
        iv. File type (image, video, sound, text)
        v. Original owner

vi. Internet URL if applicable

2. List of available files
    a. File information:
        i. File name
        ii. Last modified date
        iii. File size
        iv. File type (image, video, sound, text)
        v. Original owner
        vi. Remote owner
        vii. Internet URL if applicable
3. List of active peers
    a. A persistent list of peers the local device has encountered.
        i. Also maintains statistics about communication with the remote device, such as pairings with the user's contact list, and information about files exchanged with this user.
4. The file system
    a. After downloading a file, it is added to the local file system.
    b. When the local file system changes, the *My Files* list is updated, either manually or automatically.

We may add additional content providers based on additional activities implemented. Additionally, we may expand the data stored in each content provider, as necessary.

A sample data storage schema for the local and remote file content

providers is outlined below, in Figure 7.

Field	Type	Restrictions
id	Integer	primary key, not null, automatically incremented
filename	String	not null, indexed, maximum of 256 bytes
directory	String	not null
last modified	Date	
public	Boolean	
size	Integer	[number of bytes in the file]
type	String	
checksum	String	
original owner	Peer	
remote owner	Peer	
url	String	

Figure 7: A storage schema for the files content providers.

Broadcast Receivers / Emitters

A broadcast receiver is a component that listens for and responds to system-wide broadcasts. A broadcast emitter generates these announcements [24]. These broadcasts can be used to control the execution of the application, prompt a user for input, or general inter or intra-application message passing.

Figure 8 is a diagram of the broadcast receivers and emitters in our system.

The yellow triangles represent events generated by an emitter and dealt with by a receiver.

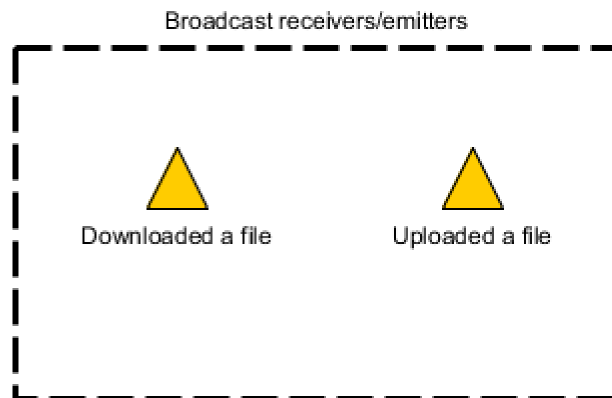

Figure 8: A diagram of the broadcast emitters in our application.

The behaviour and functionality of these emitters is outlined below.

1. Downloaded a file
    a. Upon completion of a file download, an emitter broadcasts a message indicating there are changes to locally stored files. Receipt of this emission results in updating the list of locally stored files on the My Files screen.
2. Uploaded a file to a user
    a. Upon completion of a file upload, an emitter broadcasts a message to indicate that information about that upload may be logged.

Overview

Combining the components outlined above, a full low-level architecture diagram can be obtained. The complete low-level architecture is given in Figure 9. This diagram models each of the activities, services, content providers, and broadcast receivers and emitters, and the interactions between these components.

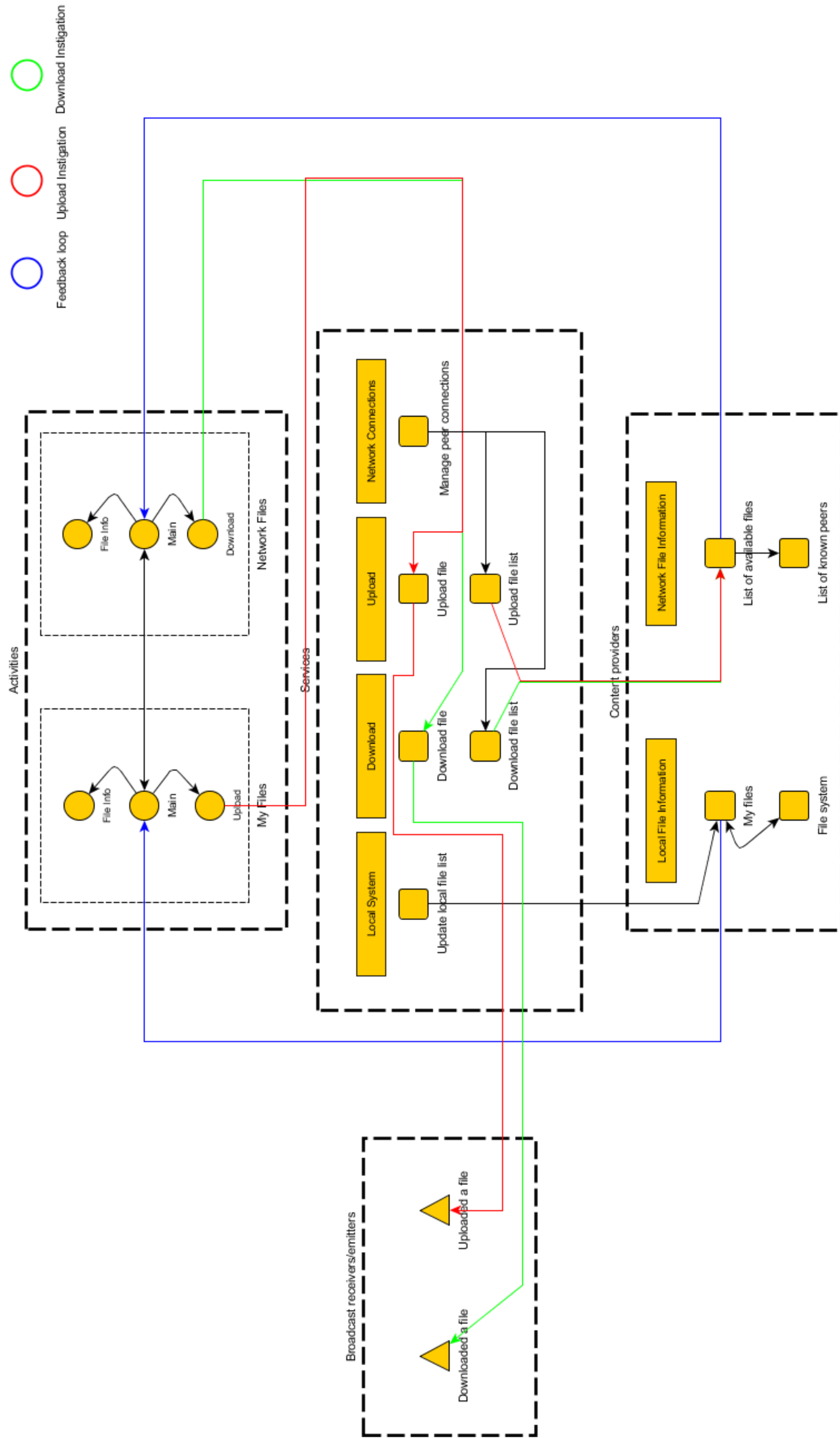

Figure 9: The full low-level architecture of our application, with modules linked together.

Network Protocol

The network protocol is the major behind-the-scenes component of the application. We want to provide efficient interactions between spatially close peers. Although the application will be written for the Bluetooth technology, the application is designed to have a modular network protocol that hides the wireless technology from its supported features. The set of core network features that will be implemented during the first phase of our implementation is presented in this section, as well as extensions that will be subject to consideration in later phases of the project.

NetworkAdapter Interface

The following set of public methods has been defined to make it easy to change the underlying wireless technology used by the application.

discoverPeers(): This is the first network operation executed when the application launches. The smartphone must be able to look for nearby smartphones. The connectTo() method is triggered when a new peer is reachable. This method is called repetitively during the application execution to keep the list of connected users fresh.

listenForConnection(): This is the second network operation to start its continuous execution until the application terminates. The application must

accept a minimum of connections with other users. If a connection request is received, this method will relay it to the receiveConnection() method and continue to listen.

connectTo(Peer p): This method allows the device to initiate connections with discovered peer devices. Once a connection is successfully established, this method relays the first exchange between peers to the getFileList() method. If the connection fails, the device may try a second time but must be cautious to avoid using too much energy.

receiveConnectionFrom(Peer p): This method allows the device to accept connections with other devices. The device must keep this connection alive as long as the slave wants, or until either peer leaves the network. Once a connection is successfully established, this method relays the first exchange between peers to the getFileList() method.

getFileList(Peer p): The first information exchanged by peers is the list of available files that each can upload. This is before they will be able to download any files. Initially, a list of all files on the phone will be sent between devices. The information that can be communicated by the getFileList and sendFileList methods include the filenames, last modified dates, file size in bytes, the file type, the original owner and the checksum.

sendFileList(Peer p): This method is responsible for sending a list of files to the peer which is passed as an argument. This will create the file list for the current phone, and transmit it to the provided peer.

download(File f, Peer p): This method initiates the download of one file from one peer. Only one file can be downloaded at any time, and this download may be interrupted by the user.

upload(File f, Peer p): This method is the dual of the download method. If upload is called for one of the phones, it will initiate an upload of the file over the Bluetooth link which is connecting the two phones.

BluetoothAdapter Class

Android and its Bluetooth package use constructs similar to TCP sockets to establish connections between pairs of devices. There is an important distinction between having two devices *paired* and two devices *connected* together. Two paired devices only indicate that they are aware of each other's existence, but in order to exchange data information, they must establish a connection. This will have an important impact on connecting and receiving connections. The implications involve additional dialogs displayed to the user for pairing with new devices and stricter management of connections. Bluetooth's limited number of connections and transfer rate will also need to be taken into account.

The following figure gives an overview of the Bluetooth network topology. This figure illustrates the roles of peers as slaves (S), masters (M) or inactive but reachable (P) devices.

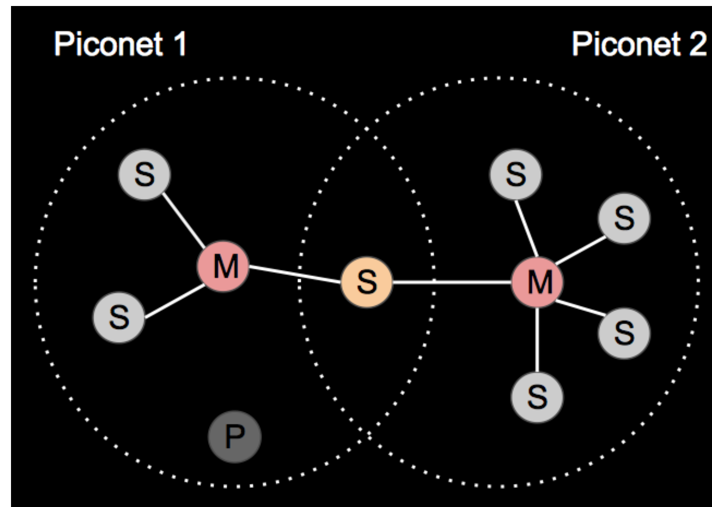

Figure 10: Bluetooth network topology, where M are masters, and S, slaves

Following the previous interface, the BluetoothAdapter class describes how each method will be implemented based on features and limitations of the Bluetooth technology. Note that the following implementation details are based on the API found on Google's website [15].

discoverPeers(): Android's Bluetooth module provides the ability to scan for other Bluetooth devices, including smartphones, using the method called startDiscovery(). Bluetooth is automatically enabled upon calling this function. This method is particularly power consuming, so we must minimize the frequency at which it is called. Discovery attempts will execute every 2 minutes as long as other phones are part of the network. For cases where no devices are

found, the scan frequency will be set to one minute until other phones are found. When at least one device is found, then the scan frequency will be reset to 2 minutes. These policies will be tested and adjusted. Note that an inquiry scan on Android takes around 12 seconds so it will seem to take more than 1 or 2 minutes to discover peers.

listenForConnection(): Each device creates and runs a BluetoothServerSocket to listen for incoming requests. Once a request comes in, the server socket will break its loop, relay the request to the receiveConnection() method, and then resume its continuous listening.

connectTo(Peer master), download(File f, Peer p): Discovered devices, or masters, are connected to with the connecting device set as a slave. Having one download at a time may be restricted by allowing only connections if the device does not already share a connection with the peer as a slave. When connecting as a slave, the device must select a service id to connect to; hence, the master can refuse a connection if all its service ids are currently occupied, which in this case, provokes a connection failure. The feature to connect peers as slave and master lets users exchange information in a duplex fashion, since the Bluetooth API only provides blocking reads and possibly blocking writes. Since there can only be eight Bluetooth connections per device, we chose the maximum number of connections as slaves to be four. This is justified because we want to allow efficient exchange in both ways between peers, so that peers do not have to wait

until the user has finished using the link to start their own download with the same device. This limit leaves enough room for four other devices in the state of master. In order to tell the master which file the user wants, an initial message is sent containing the name or id in the file list. Immediately after sending this information, the device can read its input stream with the actual file coming in. If the master disconnects or an error occurs in the middle of the download, it is the slave responsibility to re-initiate connection to the master and renew the connection using the connectTo() method.

receiveConnection(Peer slave), upload(File f, Peer slave): A list of service ids must be handled in agreement with the connectTo() method to allow connection between devices. At most 4 connections and uploads can occur in the role of master. This makes it symmetric to connectTo; hence, it will be easier to distribute the service ids for these two types of connections (master or slave). Since there is no restriction as one download at a time for uploading, but this is the case for the slaves, the device can upload as many as one file per slave.

getFileList(Peer master) and sendFileList(Peer slave): The device sends its file list in the master state and receives one file list in the slave state; hence, the download or upload can immediately occur if the connection is still alive between peers. The device blocks by opening its input stream for any request from the user. The device will receive the name or id of the file to upload, and then switch to its output stream to send the corresponding file. If an error occurs

or the slave leaves, the upload is stopped and the connection is dropped, leaving the responsibility to the slave to re-initiate the upload.

Extensions

The following extensions aim at improving the efficiency and user-friendliness of communication between peers.

Discovering new peers and listening for connections indefinitely can consume battery inefficiently in certain cases. Discovery of new peers can be halted when the application is run in the background, since the user does not look at available files during this time. When the application returns from its background state, the discovery can resume its continuous execution. Bluetooth does not allow an infinite number of connections to other devices; hence, there is no need to listen for connections if all server sockets are occupied for receiving connections. Further optimization also affects the behaviour of those methods, such as in the possibility to close connections with peers after the file lists are shared to promote connectivity with a higher number of users.

The loss of connectivity in the middle of a file transfer triggers a new connection to occur between peers. This operation is not optimized if users must exchange their file list before resuming the exchange. The download() method can keep track of peers with which the connection failed in the middle of the download operation and the getFileList() method can save the exchange of file list for later.

In later phases, the getFileList and sendFileList methods should be re-

worked to only share a list of at most 50 files at a time. This will prevent very large transmissions between phones which store a lot of files. The getFileList() method should be called repetitively during the connection with peers or receive notifications if the list of file changes. The user could even manually trigger refresh of this file list using a user interface element on the Network File screen. Whenever the peer requires more than the original 50 files, it will need to issue a request for more files. This list of files can be compressed using the Gzip compression algorithm before being transmitted to reduce the amount of data being sent. The getFileList() method would listen to the Bluetooth socket for peer p for the file list from that peer, in this particular implementation.

During download and upload, files can be opportunistically gzipped if such compression will significantly decrease the size of the file. For files which consists only of text, Gzip may be employed to compress the file before transmission. In the case of files which have already been compressed, such as epub, jpeg, png, mp3, and mp4 among others, additional compression will be fruitless, so the file will be sent as is. If the file is gzipped for transmission, the downloading peer will have to extract the file upon receipt. Additionally, upon receipt of the file, the downloader will re-compute the checksum used to uniquely identify the file, and verify that no errors have occurred during transmission. This can become tedious for large files, where a single bit error in transmission requires retransmission of the whole file. To address this issue, in a later phase, we will implement a scheme to divide large files into chunks, and compute the checksum for each chunk. Then, these chunks will be transferred sequentially

from host to host, and reunited at the destination. This means that a single bit error only requires retransmission of the chunk, rather than the whole file.

For the case, where we would like to have more than one download at the same time, modifications may need to be applied in order to have a fair number of connections as master and slaves (specifically for Bluetooth) and also take into account the case where more than one file are transmitted simultaneously from the same user. These additional features may involve changes to the user interface.

One of the major trade-offs outlined in [9] of peer-to-peer mobile networks is energy consumption over speed or frequency. While there could be many peers in a local network, in order to maintain communication with a limited number of them, for energy consumption reasons, we may want to use what was outlined in the previous reference to choose peers that may be most interesting for the user's community. By choosing weak ties, or in other words users that are not met very often, we can allow peers to view each other's content that may differ from strong ties. This would involve keeping track of a list of known devices and the frequencies at which the device communicates with those. More precisely, we will emphasize connections with peers that are not met regularly to keep data as diverse as possible among communities of users.

In addition to limiting the number of connected peers, we can limit power consumption by reducing the application's rate of activity when the battery is low. In those cases, the number of uploads would be reduced, as well as the number of connections, so that the user must explicitly permit any uploads

that occur from the device. This would protect the user from having his or her phone's battery be drained by the application.

Development Stages

SCRUM is the software development model chosen for next semester. It will provide flexibility for the concurrent work to do in other classes and focus on the core features to implement next semester.

The development plan over the next semester has been divided into 2 major phases. The first 10 weeks encompass the development of the core features, and the last 5 weeks are reserved for extensions and optimizations of the software. The first 10 weeks will be divided into small sprints for better time management and frequent iterations.

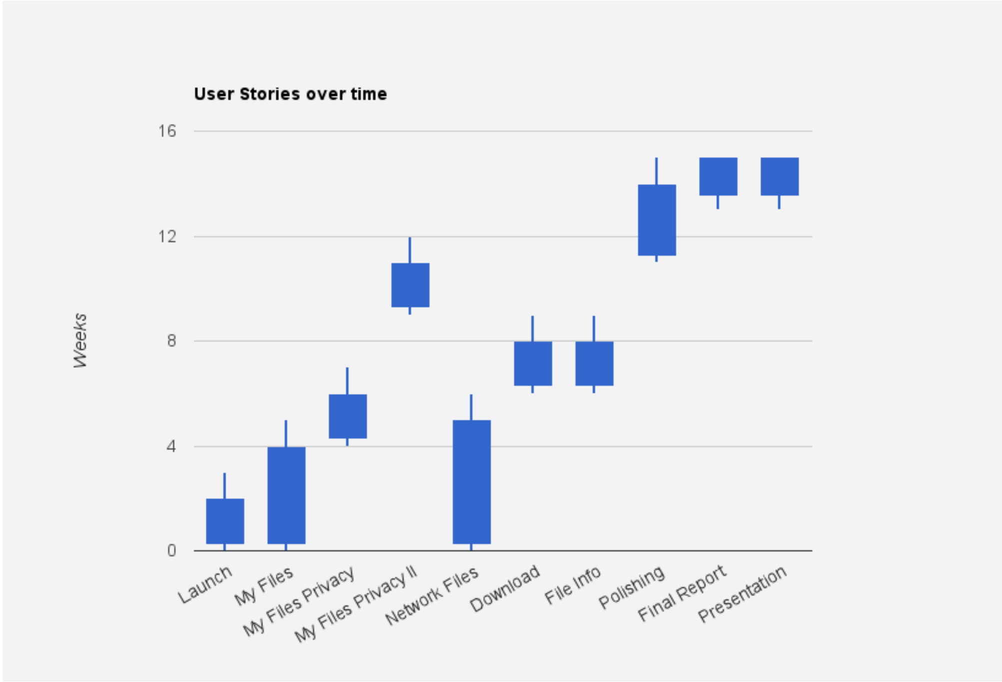

Figure 11: developing stages

The next figure is a screenshot of user stories and their associated tasks.

Backlog

User = Android device owner of the application

		User Story						
#	As a...	I want to...	[Benefit]	Tasks			User Tests	
0	Overall Application							
0.1	User	launch the application on my Android device	Start using the application and download it from the App store.	Create application icon	Create basic UI as loading screen		Launch the app.	Close the app.
1	My Files							
1.1	User	browse the files on my device.	See the content on my device that can be shared with others.	Create 'My Files' database table	Link database to device global file system	Add a corr. Activity bound to this database	Transfer file to device manually and visit this screen.	
1.2	User	set the privacy of certain files.	Control which file is public or private.	Add new fields to 'My Files' table	Modify UI to allow this feature		Change privacy setting from public to private and vice versa.	
1.2.1	User	set the privacy of certain files in terms of my contact list.	Let my friends have access to different content than my coworkers.	Create an interface with the contact list	Change fields to 'My Files' table		Change privacy setting in terms of its contact list categories.	
2	Network Files							
2.1	User	consult list of available files for download in the network.	See what is available to download among my peers.	Create NetworkFiles class and KnownPeers database table	Integrate Bluetooth multipoint architecture	Add a corr. Activity	Open application in presence of another peer device	
2.2	User	download a file from a peer.	Obtain what is available to download among my peers.	Add new fields to 'My Files' database	Modify UI to allow this feature	Update Activity	Download a file from a peer	
2.3	User	see more information about a certain file	Consult the file information before initiating download.	Modify the initial exchange of file lists to include more file information	Update NetworkFiles class	Update Actibity	Select file from the list and press download.	

Figure 12: User stories and associated tasks

Validation

Validation is an important part of any software development project. In our project, we must worry about testing the functionality of the application on a single phone, as well as testing the network protocol that we implement.

We will test the behaviour of the application on a single phone using the Android emulator provided by Google. This will allow us to test execution paths through the software in a non-connected environment. In this phase, the only major feature to test is the user interface. We can implement a stub to emulate the network layer and allow testing of the user interface against a set of requirements, to ensure that the interface is displaying all of the correct information it receives from the network layer.

The bulk of testing our application lies in verifying that the network functions work as expected. We will need to verify that phones can be paired with one another, that they can exchange file lists with one another, and that they can each upload and download files. A more specific plan for validation is outlined in figure 10, below.

Function	Required Validation
User Interface	Ensuring that control flow paths through the application are logical and do not cause errors.
Connections	Ensure that devices are connected, and can share data.
Sending File List	Make sure that devices can send file lists to and from one another, and that these file lists are properly displayed by the Network Files activity.
Sending Files	Make sure that files transferred between devices are transferred successfully, and that the application is alerted of this. Ensure if a file is received with a checksum which differs from the expected checksum, that the file is rejected.

Figure 13: Validation requirements.

We will make extensive use of the Android logging system for validating the software, and to make debugging the application simpler.

Environmental/Social Impact

The merits and drawbacks of an application for sharing content between mobile phones in the absence of network connectivity are not easily discernible. This application is similar in many ways to the BitTorrent protocol, but extended to a mobile environment. BitTorrent provides an efficient, reliable distribution mechanism for files of all sizes to a network of peers [16]. BitTorrent is the most popular peer-to-peer protocol, a family of protocols which are responsible for the highest proportion of Internet traffic [25].

While BitTorrent is commonly used to illegally share copyrighted material between users, it is also frequently used for legitimate means. Many Linux distributions employ BitTorrent to distribute their disk images [26]. Facebook and Twitter use BitTorrent to distribute code updates across vast arrays of servers [27] [28].

There is clearly a high demand for peer-to-peer sharing applications, and if commercialized, our application could address this demand. While it is possible that this application could be used to share copyrighted material, mechanisms may be built in to discourage this sort of sharing. Additionally, this application serves only as a peer-to-peer file distribution protocol, and certainly does not lend itself to a particular file type, or usage pattern other than those outlined above.

While it is possible that an application such as this could be misused, the

application also presents a number of social benefits. This application could easily be used by a higher-level application, in effect adding another layer to the networking stack of the phone, to enable more complex functionality. Uses of this might include sharing body scan imagery, such as MRIs, X-Rays, and CT scans between doctor's phones when they must consult each other; creating networks of phones in less developed countries, where no Internet connections exist; and controlling software updates on the device.

The application could also be used for more mundane purposes. Applications in sharing image files between family and friends come to mind. Another possible use for the application might involve sharing music files as "previews." Supposing one user downloaded a new song from the Google Music Store, they might get five "shares," and could share the song with five of their friends, who may each listen to it exactly once. A point-to-point messaging application for users near one another could be another use of the protocol. These are only a few possible higher-level uses of this application, but it is apparent that such an application could be used to serve society well.

There are clearly many uses for this application. This is expected, as implementing a general protocol like file-transfer through ad-hoc local area connections is quite a broad topic. While the social and economic impacts of this application are not entirely clear yet, we have looked at a few potential uses for the application.

Conclusion

In this report we have outlined the purpose of our design project, the research that we have made on it and its projected implementation for the next semester.

Our design project is an application for peer-to-peer file transfer on Android mobile phones. It will provide an interface to browse files on the device and on the network. As outlined, the components making up this application are tied to the Android development model -Activities, Services, Content Providers and Broadcast Receivers/Emitters- and the network technology used to transfer files. We decided to go with the Bluetooth technology because of the API available to us, but we will organize the software to allow for other potential technologies like Wifi-direct.

Because of the importance of our network protocol, we have also explained how our application protocol will work by providing a tentative API for it. We duly noted the challenges of our application and their potential solutions in the face of our choice of technologies.

We then described broadly the work to be done next semester. The SCRUM development model (adjusted for our needs) was chosen for its flexibility and its methodology based on frequent results.

Finally the potential social impacts of our endeavour were discussed. A peer-to-peer application has mitigated and unpredictable consequences.

References

[1] D. Murph. (2011, November). *Gartner's Q3 2011 smartphone figures:*

Samsung on top globally, Android tops 50 percent share [Online]. Available:
Engadget Blog & Gartner website (originally but link is broken):
http://www.engadget.com/2011/11/15/gartners-q3-2011-smartphone-figures-samsung-on-top-globally-a/

[2] P. Hui, J. Crowfort and E. Yoneki (2011, November). BUBBLE Rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing, v10 n11 (2011 11 01): 1576-1589* [Online]. Available: www.cl.cam.ac.uk/~ph315/publications/hoc86309-hui.pdf

[3] R. Jones. "A technological revolution: mobile internet will be a major growth area in the decade ahead. Despite fragmented access at present, all Middle East countries are expected to have full mobile coverage by 2015". *MEED Middle East Economic Digest,* vol. 54 (53) pp. 31, Dec. 31, 2010.

[4] J. Wortham. (2011, October 12). BlackBerry's Service Hiccups Spread; Five Continents Affected. *The New York Times* [Online]. Available: The New York Times website: http://www.nytimes.com/2011/10/13/technology/hiccups-in-blackberry-service-continue.html

[5] bump Technologies, Inc. (n.d.). *The Bump App for iPhone and Android* [Online]. Available: http://bu.mp/

[6] touchbyte GmbH. (n.d.). *Photosync* [Online]. Available: http://www.photosync-app.com/

[7] SoftFrame, Inc. (n.d.). QwikCards for iPhone [Online]. Available: http://www.qwikcards.com/

[8] S. Ioannidis, A. Chaintreau and L. Massoulié. (2009). Optimal and Scalable Distribution of Content Updates over a Mobile Social Network. *Proceedings - IEEE INFOCOM* [Online]. Available: www.thlab.net/~lmassoul/ICM.Infocom09.pdf

[9] S. Ioannidis and A. Chaintreau. (2009, March). On the strength of weak ties in mobile social networks. *Proceedings of the Second ACM Workshop on Social Network Systems (SNS)* [Online]. Available: http://www.cs.columbia.edu/~augustin/pub/ioannidis09strength.pdf

[10] Apple, Inc. (n.d.). *iOS Dev Center - Apple Developers* [Online]. Available: https://developer.apple.com/devcenter/ios/index.action

[11] Google, Inc. (n.d.). *Android Developers* [Online]. Available: https://developer.android.com/index.html

[12] Google, Inc. (n.d.). *Package Index* [Online]. Available: https://developer.android.com/reference/packages.html

[13] Google, Inc. (n.d.). *The Developer's Guide* [Online]. Available: https://developer.android.com/guide/index.html

[14] Wi-Fi Alliance. (n.d.). *Wi-Fi Alliance: Wi-Fi Direct™* [Online]. Available: http://www.wi-fi.org/Wi-Fi_Direct.php

[15] Google, Inc. (n.d.). *Bluetooth* [Online]. Available: https://developer.android.com/guide/topics/wireless/bluetooth.html

[16] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. Boston, MA: Addison-Wesley, 2010.

[17] G. Held, *Data over wireless networks: Bluetooth, WAP, and wireless LANS*, 1st edition. New York, NY: McGraw -Hill, 2000.

[18] M. Barbeau and E. Kranakis, *Principles of ad hoc networking*, 1st edition. New York, NY: McGraw-Hill, 2007.

[19] C. Bisdikian, "An Overview of the Bluetooth Wireless Technology," *IEEE Commun. Mag.*, vol. 39, no. 12, pp.86-94, Dec. 2001.

[20] Google, Inc. (n.d.). *Activities* [Online]. Available: https://developer.android.com/guide/topics/fundamentals/activities.html

[21] Google, Inc. (n.d.). *MessageDigest* [Online]. Available: http://developer.android.com/reference/java/security/MessageDigest.html

[22] Google, Inc. (n.d.). *Services* [Online]. Available: https://developer.android.com/guide/topics/fundamentals/services.html

[23] Google, Inc. (n.d.). *Content Providers* [Online]. Available: https://developer.android.com/guide/topics/providers/content-providers.html

[24] Google, Inc. (n.d.). *Application Fundamentals* [Online]. Available: https://developer.android.com/guide/topics/fundamentals.html

[25] H. Schulze and K. Mochalski, "Internet Study 2008/2009," ipoque, Leipzig, Germany, 2009.

[26] Linuxtracker. *Extra Stats* [online]. Available: http://linuxtracker.org/index.php?page=extra-stats

[27] T. Cook, "A Day in the Life of Facebook Operations," presented at the Open Source Bridge Conference, Portland, OR, 2012. Viewable at: http://www.youtube.com/watch?v=T-Xr_PJdNmQ&feature=player_embedded#t=14m

[28] L. Gadea, "Using BitTorrent for Fast Website Deploys," presented at the Canadian University Software Engineering Conference, Montreal, QC, 2010. Viewable at: http://vimeo.com/11280885

[29] *Specification of the Bluetooth System*, Version 4.0, June 30, 2010.

[30] F. Mazzenga, D. Cassioli, P. Loreti, and F. Vatalaro, "Evaluation of Packet Loss Probability in Bluetooth Networks," in *IEEE International Conference on Communications*, New York, NY., 2002, pp. 313-317.

Appendix

Prototype Code

See ECSE476GR6_SP1Prototype.zip for prototype code. Also included in this zip are screenshots of the prototype, for those who would not like to go through the installation procedure.

Speed Calculation

A calculation of the time required to send a selection of different files is outlined below. This calculation assumes the use of Bluetooth protocol specification 4.0 or higher, and will consider using the high speed operation of the protocol, which offers transmission speeds up to 24 megabits per second, the enhanced data rate operation of the protocol, which supports 2.1 megabits per second, and the regular operation of the protocol, which is capable of 721.2 kilobits per second [29]. We will make the conservative assumption that 10% of all data packets are dropped, which would be expected if there were 10 piconets in a 20 meter squared area [30]. The files used for the analysis are:

1. Text File - Thomas Friedman's New York Times column, "The Arab Awakening and Israel," which can be stored with relevant information in 8 Kilobytes in the RTF format. This can be compressed to 4 Kilobyes using Gzip.
2. Text File - War and Peace electronic book in ePUB format, from Project Gutenberg, which can be stored in 1.3 Megabytes. No reasonable compression ratio can be achieved using a standard compression algorithm.
3. Image - JPEG compression ratio 8 in Adobe Photoshop, image dimensions 1000 x 650, a photograph of a person holding a sign. The size of the image is 130 Kilobytes. No reasonable compression ratio can be achieved using a

standard compression algorithm.

4. Music File - Led Zeppelin's Stairway to Heaven, encoded in MP3 format at 128 Kilobits per second. The total file size is 7.31 Megabytes. No reasonable compression ratio can be achieved using a standard compression algorithm.
5. Video File - The first episode of the first season of *Scrubs*, encoded in AVI format at 512x384 and 24 frames per second with 192 kbps audio is 176 Megabytes. No reasonable compression ratio can be achieved using a standard compression algorithm.

We can calculate the time required for transmission using the equation:

$$t = \frac{1.1 \times (size\,in\,bits)}{(transmission\,rate\,in\,bits\,per\,second)}$$

	Regular (721.2 Kbps)	Enhanced (2.1 Mbps)	High Speed (24 Mbps)
1 (RTF) [4 kB]	0.0488 s	0.0168 s	0.0014 s
2 (EPUB) [1.3 MB]	15.8625 s	5.4476 s	0.4767 s
3 (JPEG) [130 kB]	1.5862 s	0.5448 s	0.0477 s
4 (MP3) [7.31 MB]	89.1958 s	30.6324 s	2.6803 s
5 (AVI) [176 MB]	2147.5319 s	737.5238 s	64.5333 s

Figure 14: A calculation of the file transfer speeds achievable over Bluetooth. These calculations assume a 10% rate of packet loss.