AirSync

Robert Durham

April 15, 2012

1 Introduction

Many portable devices have become a part of our everyday lives impacting work as well as our home lives and the way we travel. Most of these devices have their own unique means of transferring media and documents. All of these devices can access the internet via some means, e.g. WiFi or cellular network. I will use my situation as an example, between myself and my wife we have at least eight different devices that we like to access music libraries, documents, photos and videos from:

Device	**Description**	**Connectivity**
Desktop PC	Linux	Wired(Home)
Home Theater PC	Linux	Wired(Home)
Laptop(Mine)	Windows 7/Linux	WiFi
Laptop(Spouse)	Windows XP	WiFi
Motorola Droid 4(Mine)	Android 4 Phone	4G & WiFi
Motorola Droid 3(Spouse)	Android 2.3 Phone	3G & WiFi
Nexus 7	Android 4 Tablet	WiFi
Kindle Fire HD	E-Reader/Tablet (Android 4 based)	WiFi

Several 'Cloud' services have offered means to maintain and provide access to media and documents via the internet, e.g. Apple iCloud, DropBox, Asus WebStorage, Ubuntu One. All of these have limited storage, cost money to expand that storage, and don't support all necessary devices. These issues are outside of the fact that these require trusting your personal data to a 3^{rd} party. DropBox and Ubuntu One support almost all of these devices, but transferring large media files through internet is slow. What if there were a solution to easily synchronize media/files through your home network, yet

still allow access to these files through the internet without relying on the privacy policy of a 3^{rd} party provider? Some open source solutions can help in desktop/laptop situations, but not with mobile devices.

2 Goal

The goal of this project is to provide a means to easily synchronize files between multiple types without relying on 3^{rd} party services to protect sensitive data. In order provide a replacement option for services like dropbox, AirSync must provide a means to access files remotely through the internet.

3 Use Cases

Their are three types of devices and two types of connections used to depict each of the use cases. The three types of devices are static PC, mobile PC and mobile device. The two types of connections are local and internet connected. A localized connection refers to the situation where the devices do not need to utilize internet bandwidth to transfer data. Local connections are typically at least 54 mb/s where as most internet connections in 2010 averaged 3.7 mb/s[1]. 54 mb/s is the theoretical limit of 802.11g wireless networking protocol. Hotels have notoriously slow internet connections.

1. Synchronize files between server and devices on local network
2. Manage Android file system from static or mobile PC on local network
3. Download/Upload files between server and Android Device or Mobile PC via internet connection

4 Design

AirSync will provide the means to synchronize files between any Windows, Linux and Android devices. It will include the following three software components:

[1]http://arstechnica.com/telecom/news/2010/01/us-broadband-still-lagging-in-speed-and-penetration.ars

Component	Description
Service	Runs as system service or daemon on server and client PCs. Acts as Server and/or Client depending configuration. *A client needs to be capable of becoming a server...*
Service Configuration UI	Graphical User Interface that will monitor status and change settings with AirSync Service on PCs. Will also provide remote management of files on Android devices.
Android Client Application	Activity App that will provide the user with a choice of two operating modes. One mode will allow remote management of the android devices memory space from the Service Configuration UI. The second mode will allow the Android user to browse the files available from the server and select files to download.

All application components will be developed in Java to maximize code reusability and portability between platforms. *Android*[2] is essentially any embedded Linux environment with a specialized Java Virtual Machine. If Android specific code sections are properly abstracted into separate Java Classes, the large majority of the android client code will be re-usable with the PC application. The Integrated Development Environment (IDE) will be *Eclipse*[3] with the Android Developer Tools(ADT) addons. *Git*[4] will be used for source versioning control with the git server being hosted on *assembla.com*[5]. This project will not include the development of an AirSync application for *Apple IOS*[6]. The development environment for IOS only supports Mac OS X. The IOS development language is a managed C++, much of the code would need to be ported and I do not have access to a Mac, so I have exclude support, for now.

[2]http://www.android.com/developers/
[3]http://eclipse.org/
[4]http://git-scm.com/
[5]http://www.assembla.com/
[6]https://developer.apple.com/devcenter/ios/index.action

4.1 Service Design

- Server side component, provides remote access to files based on system configuration
- Tracks initialized devices and associated unique ids
- Use XML configuration files
- Communicate transfer logs and status

4.2 Service Configuration UI Design

- Graphical interface to service for configuration and status monitoring
- Enable access to certain files and folders. Potentially on a per device basis(Access Restrictions).
- Log remote transfers

4.3 Android Client Application Design

- Provide client file synchronization with service
- Allow remote file selection for download
  - when not on localized network
  - user selectable
- Provide remote file system management to service

4.4 Source Code Organization

This section is being updated as the project progresses.

The source code and build files for this project can be found at *http://github.com/durhamrj/AirSync*
The Service and Service Configuration UI share a common Eclipse/Java project. The components are separated by the package structure within this project. The folder structure of the project is as follows:

http://github.com/durhamrj/AirSync	
AirSyncServer/	Cross Platform Java application for Desktop and Laptop computers
RemoteSync/	Android application source code and project files
docs/	Contains documentation for project include latex source for this PDF
README.md	An autogenerated file created by github

4.4.1 AirSyncServer Project Structure

AirSyncServer Source	
flyinpig.sync.service	
CommandHandler.java	Handles socket IO with client connections as a separate thread
Listener.java	Opens listening port in a thread and spawns new CommandHandler threads for each new incoming connection. New connections are tracked for display in the UI.
Main.java	Singleton class. Includes the application main and fields/structures to track connection application state information.
flyinpig.sync.service.structures	
CommandResponse.java	Handles conversion of commands & responses between Java objects and binary data to be transmitted over Socket IO. This was initially intended to use XML serialization, but due to a lack of android support the conversion code is all in this file and uses not additional system libraries.
DeviceInfo.java	Contains specific information about a device sent on initial connection. Used to hold information displayed within the UI to uniquely identify each device.
ParsingException.java	Exception thrown by CommandResponse when errors occur parsing command & responses.

4.4.2 RemoteSync Project Structure

The file structure in the Android project is largely made of generated files. I will only identify the files that I have manually modified as well as source code I have written. Many of the source files are identical between the AirSyncServer project and the RemoteSync Android project.

RemoteSync Source	
AndroidManifest.xml	This file is most importantly used to set application permissions.
flyinpig.sync	
RemoteSyncActivity.java	Main class for application. Monitors/Controls application view and state. Views are generated with a graphical interface in the android developement kit.
UIListener.java	OnClickListener for Views.
flyinpig.sync.io	
ClientThread.java	Handles Client Socket IO. Similar to CommandHandler but more specific to android threading and IO paradigm.

5 Using AirSync

6 Lessons Learned

7 Conclusion

8 TODO

- SharedPreferences
- MediaStore