

Private Dropbox  
Final Report  
COSC480

Calum O'Hare  
Supervisor: David Eyers

# Abstract

I have written a program in Python which reads user settings from a file, and synchronises the appropriate files to the appropriate machines when they have been modified. It does this using an efficient two way file synchronising tool called Unison. I will discuss in this dissertation what I have done and how I have tested my program.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Project goals . . . . .	4
1.2	Background . . . . .	4
1.3	Example use case . . . . .	5
<b>2</b>	<b>Supporting architecture and program development</b>	<b>6</b>
2.1	Virtual Machines, Node networks . . . . .	6
2.2	Getting my IP, networking issues . . . . .	7
2.3	settings and stored data . . . . .	7
2.4	Python . . . . .	7
2.5	User control . . . . .	8
<b>3</b>	<b>Program mechanics</b>	<b>8</b>
3.1	Monitoring Directories . . . . .	8
3.2	Unison and temporary files . . . . .	10
<b>4</b>	<b>Program Evaluation</b>	<b>10</b>
4.1	Point-to-Point synchronisation . . . . .	10
4.2	Full graph replication . . . . .	12
4.3	When to stop copying . . . . .	14
4.4	Dealing with Sub-nodes . . . . .	16
4.5	How often to sync . . . . .	17
4.6	Wi-Fi vs 3G . . . . .	19
<b>5</b>	<b>Conclusion</b>	<b>19</b>
5.1	Future Work . . . . .	19
5.1.1	Mobile Nodes . . . . .	19
5.1.2	Feedback . . . . .	19
5.2	Results . . . . .	19
<b>A</b>	<b>WatchAndSync.py</b>	<b>19</b>
<b>B</b>	<b>ReadNet.py</b>	<b>28</b>
<b>C</b>	<b>onTheFly.sh</b>	<b>32</b>

# 1 Introduction

## 1.1 Project goals

The aim of this project was to develop a file synchronisation tool. Similar to Dropbox (and others) its main function should be to keep data synchronised between multiple devices. What makes it different however is it should:

- Support decentralised operation. It will not necessarily need to communicate with ‘the cloud’. The program should not require a centralised server. However it should be possible to configure the system to behave like a centralised system if the user wants to. The system should be flexible in this regard.
- Allow file synchronisation between multiple clients not just point-to-point between two clients. Synchronisation between two clients however is the basis for multiple client synchronisation. Clients may be running different operating systems. Clients may be connected to different networks, with different costs of access, including being disconnected from the Internet at times.
- The user should be able to choose what to replicate and how often to do it within different sets of files. Choosing what to replicate could be done based on file name, file types, file size *etc.* The system should allow for fine-grained user control for the majority of the program’s functionality.
- Show statistics about which files are being replicated, efficiency (time taken for the files to become fully up to date), cost (bandwidth, disk space used). These statistics could also possibly lead to a heuristic for when to synchronise a given file. For example if a file is updated and a node has many neighbours to potentially send the file too. Perhaps choosing the neighbour whose links has the lowest cost would be a good choice to send the data to first.
- Operate automatically, without the user having to initiate a file synchronisation themselves. The system’s autonomous operation should be influenced by the users choices on how often to sync and what files should be synced, this relates to the fine-grained controls mentioned above.

## 1.2 Background

There are already many services available that can synchronize your files between different devices. Dropbox, Google Drive, Microsoft SkyDrive, Apple iCloud are all examples of cloud based solutions for distributing your files across your devices. The problems with these services is privacy and availability. Storing your data with a third party gives them access to your documents. If you are a commercial organisation with sensitive information this might be concerning. You could of course choose to encrypt your files. Encrypting your files adds two slow extra steps, encrypting them before you upload

and decrypting files before you can use them, this is less than ideal. You also cannot guarantee that you will always be able to access your data, if the company that hosts your data goes bankrupt or decides to shutdown their service you could lose all of your data with little or no warning.

For example Megaupload, a file hosting service, has recently been shut down by the United States Department of Justice for alleged copyright infringement. According to its founder, 100 million users lost access to 12 billion unique files<sup>1</sup>.

There are other possible approaches to replicating files across multiple computers. For example you could use version control systems like Subversion, Mercurial, and CVS. One problem with these is that they are centralised (they rely on a central server), should that server fail the replication will break. Not only that, they create a bottleneck at the server which can slow replication down. Cloud based solutions are often centralised. Another problem is that even if they are decentralised like git, they will not automatically push updates to other working sets. This could be accomplished with some cron scripts or a post-commit hook to get git to propagate data onwards. Git might have made a promising base to build my application on top of, the only real problem was the version control overhead that comes with it. Old revisions would take up space on the hard disk and require more data to be transmitted across network links. I decided that as a file synchronisation program, revision history was out of scope and that my program would deal with just keeping files in sync. Using git would be an interesting extension to my program and could easily be integrated into my current system.

### 1.3 Example use case

Here is an example use case demonstrating why I find my program useful.

I like to keep all of the data on my laptop backed up to an external hard drive. The data on my computer that I wish to back up falls into three main categories: documents, music, and movies. Documents are mostly scripts and programs that I am writing for University or work projects. Documents also include reports for assessment. These documents change very frequently and are very important to me. Often these are small files (but not always). My music collection changes relatively infrequently, files are around  $\approx 5$ MB and I like to have a relatively current backup of this collection. My movie collection contains fairly large files but I do not need it to be backed up very often as it does not change very much and I do not care if I lose some of these movies. Files that I work on at University would be very useful to have on my laptop at home. Files that I work on at work mostly stay at work but occasionally I might want to bring something home to work on. The other device I always have with me and may be on one of any given (Wi-Fi or 3G) network at a certain time is my smart phone. I would like to have photos taken on this backed up to either (or both) my laptop and external hard drive.

---

<sup>1</sup><http://computerworld.co.nz/news.nsf/news/kim-dotcom-wants-his-money-back>

Some of the files that I move around are of a sensitive or personal nature and I would prefer not to store them with a third party vendor. I also have different synchronisation requirements for different types of data. For example my collection of large video files does not change that often and will chew up valuable network bandwidth whenever it has to transfer a new file. I like this to be replicated only occasionally as I do not use it that much. On the other hand my document collection which I use for work and coursework changes very often, is very important, and is fairly small. I would like this to be as up to date as possible.

Existing file synchronisation tools do not do enough for me. I do not have enough control over my data. I want to know which machines my files are going to and when. I want to feel confident that I will always be able to access my data even if the service closes down or my internet connection fails. My program is aimed at addressing these issues.

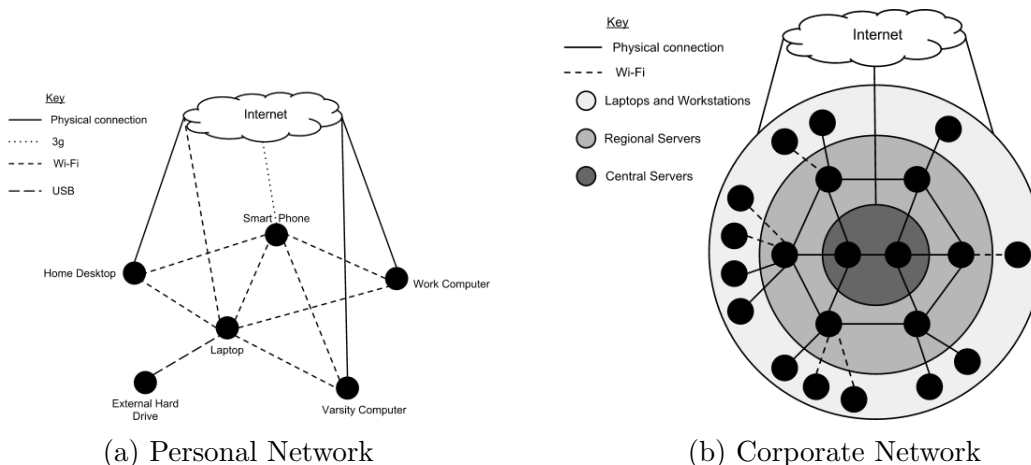


Figure 1: Example use cases

I have already described the personal network shown in figure 1a. Figure 1b shows a graph of a corporate network, this is another example use case. It will have many of the same basic needs as the personal graph. The coloured rings represent the need for different policies for different machines in a network. Something which Dropbox will not provide but my system does.

## 2 Supporting architecture and program development

### 2.1 Virtual Machines, Node networks

For testing my program I needed to have a network of computers that can be linked together in different arrangements easily. I decided to use virtual machines for this job since it means I do not need to have a large number of physical machines. I can create new machines very easily, and manipulate the links between them.

I have used Oracle's VirtualBox software. I chose VirtualBox because of its easy to use command line interface. My program should be able to run across any network of nodes. So I wanted to test as many different arrangements as possible. I decided that I needed to be able change network topologies easily without having to re-write my scripts. I built some bash scripts to run on top of a program called Graphviz.

"Graphviz is open source graph visualization software".<sup>2</sup> I used Graphviz to generate graphs of all the topologies I worked with. This made it easy to keep track of what a topology looked like which was useful for debugging. It was also useful to display results alongside an image of what the topology looks like. Building my program on top of Graphviz meant that I could couple the production of the topology graph and the configuration of the virtual machines together. I never wanted one without the other so this was very useful.

I use Graphviz configuration files to set up my virtual machines as well as generate graphs. This means I only have to write a configuration file once. Graphviz uses a simple notation which made it easy to have a bash script also read this file.

My bash script enables the appropriate network adaptors on each virtual machine. Then it sets these adaptors to use an internal network which it also creates. I chose to use an internal network as the link between any given machines because this way I could guarantee that no other programs were running over that interface using up network traffic.

I have decided to start testing my program with some simple topologies to see if I can gain any insight into how best to replicate data around a network with many nodes. The next step will be to use those principles and start running more complicated networks to see how the program performs.

## 2.2 Getting my IP, networking issues

route etc.

## 2.3 settings and stored data

stores files sync time, how up to date all graph data could be utilised

## 2.4 Python

I have chosen to use Python to implement my program. Python appealed to me because it supports many different platforms (Windows, Linux, Mac OS X). This is useful because it means I will (hopefully) encounter fewer compatibility problems when running my program across different operating systems in the future.

---

<sup>2</sup>[www.graphviz.org](http://www.graphviz.org)

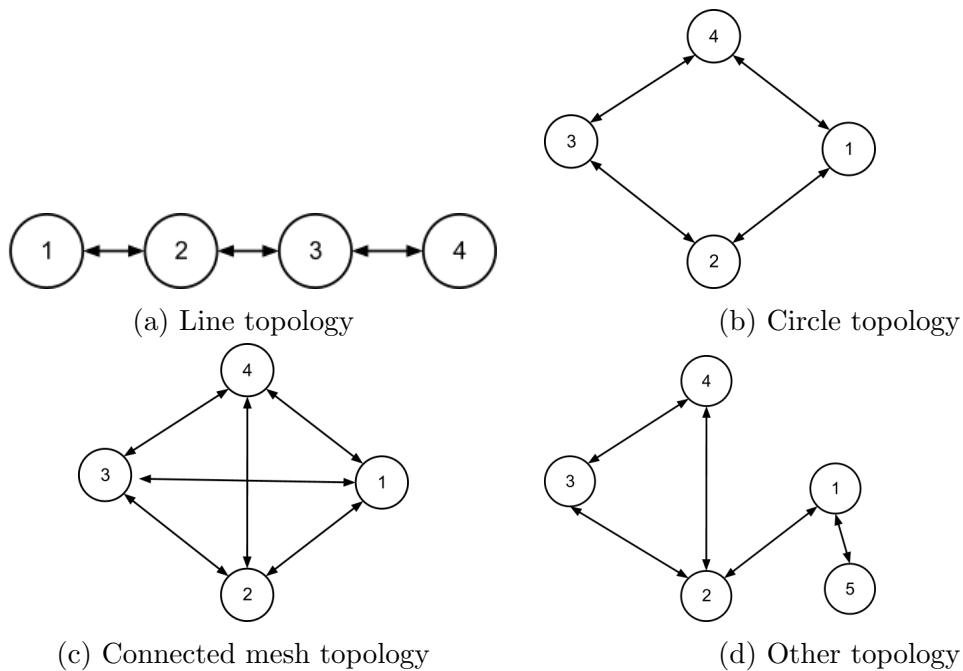


Figure 2: Simple network topologies

## 2.5 User control

One of the main goals of my project is to allow the user to have a large amount of control over how the program behaves. I currently have the program reading from configuration files that allow the user to specify which directories they want to watch and where those directories should be synchronised to.

I chose to use directories as my granularity for replication as opposed to files because keeping track of a large list of files may become unwieldy, and because I replicate directories recursively, I can replicate large amounts of data without a cluttered configuration file.

Another reason I chose directories as my granularity was because it may be handy to have a directory full of symlinks pointing to other directories.

## 3 Program mechanics

### 3.1 Monitoring Directories

The application needs to monitor directories for changes (see section 4.4) so that it knows when to perform a sync. The reason I have chosen to do this is because synchronising a directory that has not been changed is a waste of time. I do not however want to be continually polling the watched directories to see if there have been any changes made. This would be a significant waste of CPU time and the input/output



time associated with checking the disk. Instead I have looked into ways of being notified of a change in the file system below the watched directory.

- Inotify

- Inotify is a kernel feature that has been included in the Linux kernel since version 2.6. It is used to watch directories for changes and notify listeners when a change occurs. Inotify is inode based and replaced dnotify, an older system that provided the same functionality. Dnotify however was inefficient, it opened up the file descriptors for each directory it was watching which meant the backing device could not be unmounted. It also had a poor user-space interface which used SIGIO. Inotify only uses one file descriptor and returns events to the listener as they occur<sup>3</sup> There is a Python module called `pyinotify`<sup>4</sup> that provides a Python interface to inotify, which I have used in my program. Another reason I chose inotify was because different kinds of changes triggered different inotify events. So I can differentiate between a file being deleted, created or modified, *etc.*

- FSEvents

- FSEvents is an API in MacOS X<sup>5</sup> It is similar to inotify in that it provides a notification to other applications when a directory is changed however it does not inform you which file in the directory was changed. This does not matter for my application since Unison is smart enough not to copy unchanged files in a directory. There is a Python module for FSEvents called `MacFSEvents`<sup>6</sup>.

I also looked at using the `kqueue`<sup>7</sup> system call that is supported by OS X and FreeBSD. It notifies the user when a kernel event occurs. I decided against using `kqueue` as the high level approach of FSEvents suits my application's needs.

- ReadDirectoryChangesW

- Windows, like the other operating systems I have examined, provides a way of doing this too. There is a function called `ReadDirectoryChangesW`. There is a `FileSystemWatcher` Class in .NET version 4 and above. IronPython might prove to be a good choice for a Windows implementation as it is a version of Python integrated with the .NET framework. I have chosen only to implement my program on Linux because portability was not in the main scope of the project. I would have liked to look at it further but became too time consuming and not interesting from a research perspective.

---

<sup>3</sup>[www.kernel.org/pub/linux/kernel/people/rml/inotify/README](http://www.kernel.org/pub/linux/kernel/people/rml/inotify/README)

<sup>4</sup><http://pyinotify.sourceforge.net/>

<sup>5</sup>[https://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/FSEvents\\_ProgGuide/Introduction/](https://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/FSEvents_ProgGuide/Introduction/)

<sup>6</sup><http://pypi.python.org/pypi/MacFSEvents/0.2.1>

<sup>7</sup><http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man2/kqueue.2.html>

## 3.2 Unison and temporary files

I noticed that when Unison ran it created temporary files in the directory and once these files had been fully copied it renamed them to their intended name. The problem with this was that my program was picking up these temporary files as they were created and trying to copy them to the next node, only to find that these files no longer existed. To get around this problem I decided to implement a filter on the files to be copied. The program filters out files that contain ".tmp" in the filename. Unison is not the only program that uses temporary files. I decided that this should be a user set preference given that users may want to filter out different files.

My program simply reads from a file with each file pattern to exclude listed on a new line. It is easy to add to/remove from. As I said above I added .tmp to the file as a default. This could easily be extended to allow a user to omit certain files from the replication by adding all files in my programs ignore file to Unisons ignore list. Or conversely by maintaining a white list of files to sync. This would allow for greater granularity when syncing nodes.

## 4 Program Evaluation

### 4.1 Point-to-Point synchronisation

After looking for cross-platform, open source, file synchronisation tools, I have found a tool called Unison <sup>8</sup> to be a promising starting base for this project. Unison is an open source file synchronisation tool. It supports efficient (*i.e.*, it attempts to only send changes between file versions) file synchronisation between two directories (including sub-folders) between two "roots" that may or may not be on the same machine. Unison calls the directories it is synchronising, roots.

I decided to run some tests using Unison and two machines running on the same network to determine whether this would make a good base for my program or not.

I looked at three methods of file synchronisation across these two machines. Naïve copying; using Rsync, an application designed for efficiently copying files in one direction by looking at the differences in the files; and Unison described above.

Rsync and Unison performed significantly better than the naïve copy method (as expected). After the initial file transfer new files added to the directory resulted in much less data being transmitted over the network, which meant the node graph became up to date much more quickly.

The reason naïve copy sent over 300MB of data to copy three 50MB files was because my implementation is deliberately naïve; it will copy the entire directory each time it is changed. Rsync and Unison were able to send less data because they work based on the differences between the files. However copy doesn't look at the files it just copies everything in the directory tree. Hence it will copy 50MB after file one is created,

---

<sup>8</sup><http://www.cis.upenn.edu/~bcpierce/unison/>

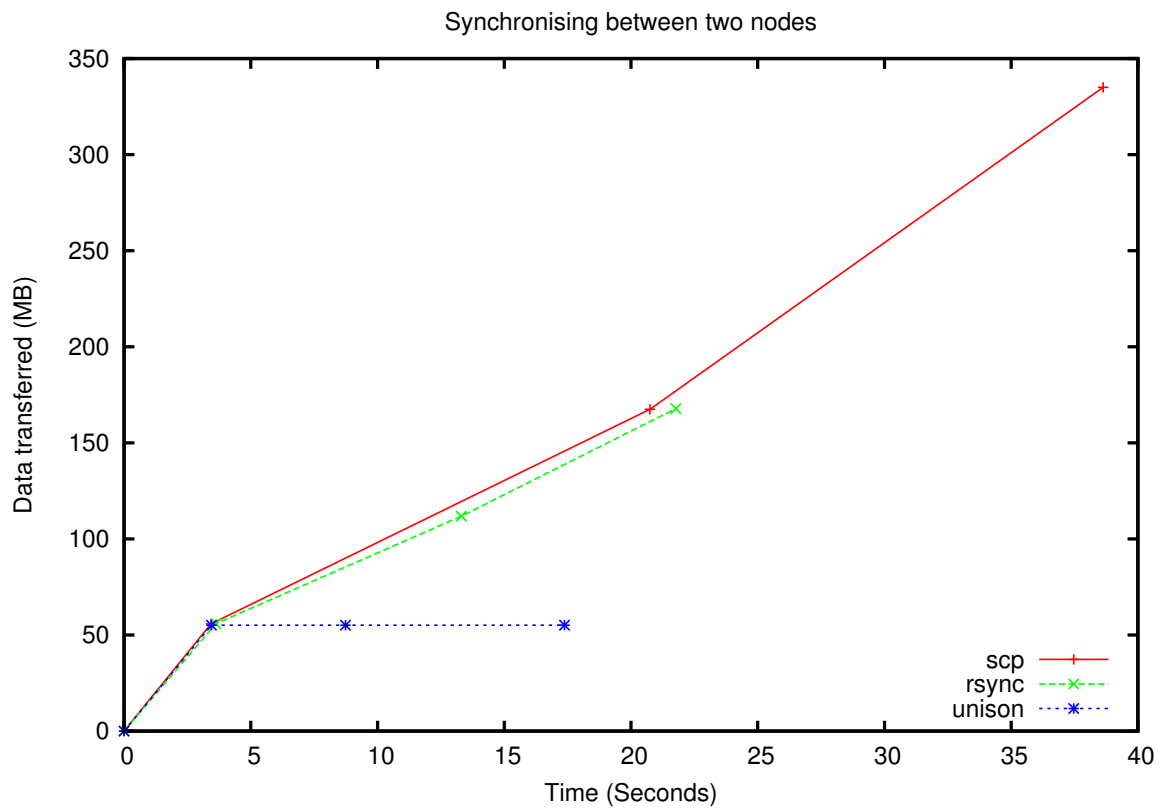


Figure 3: Comparison of SCP, Rsync, Unison. Three identical 50MB files with different names are being transfered between two nodes.

100MB after the second file is added and finally 150MB when all three files are present for a total of 300MB.

Rsync copies the expected 150MB for three 50MB files. Figure 3 illustrates another advantage of Unison over Rsync. The graph shows three zero filled binary files being copied from one node to another one after the other. Unison recognised that even though the files were named differently they were the same file. Another advantage of Unison is that it handles replication in two directions without overwriting the files on the other side.

Each of the three methods I trialed had some overhead associated with them. This overhead was due to the secure shell (SSH) tunnel between the machines that all three methods used. Unison and Rsync also incur some overhead when comparing the differences between the files in the directories. This is why the graph shows the three lines slightly above where you might expect them to be for the amount of data that was copied.

## 4.2 Full graph replication

As you can see Unison and Rsync outperformed SCP advantage of Unison is two-way sync

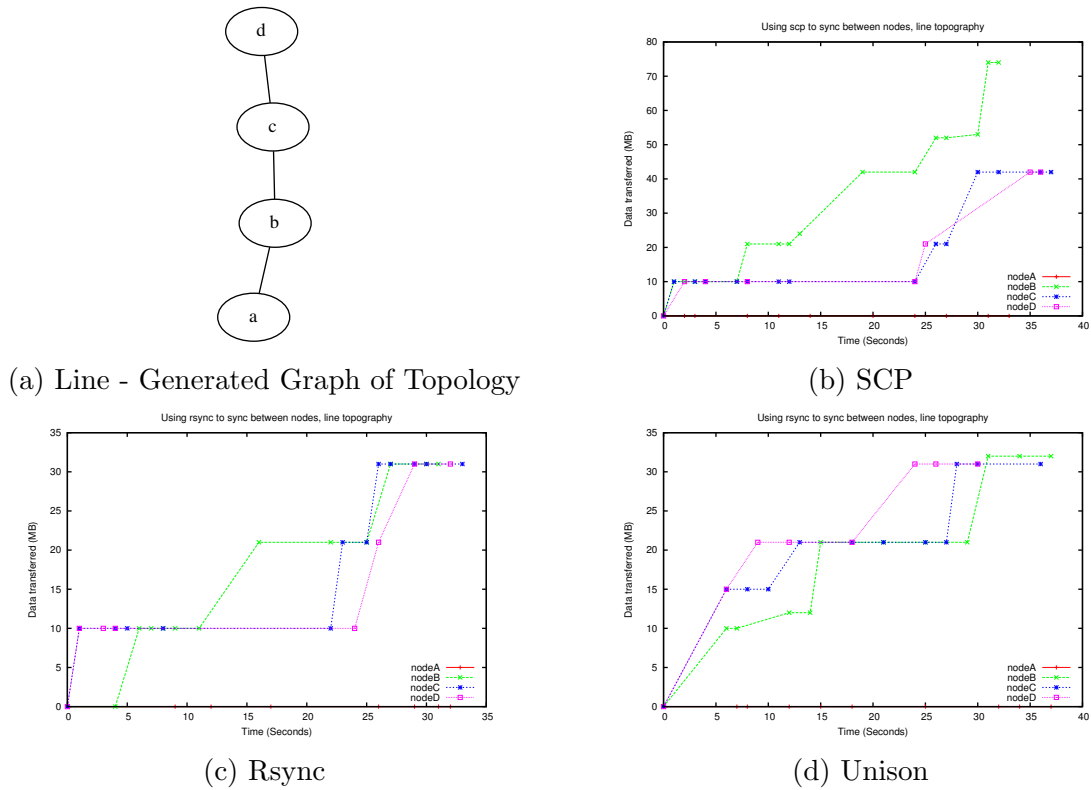
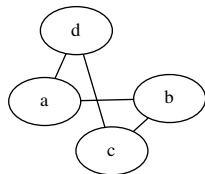
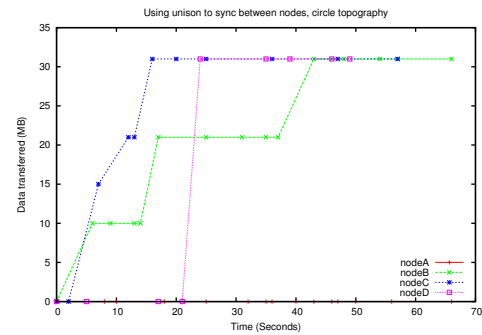


Figure 4: Comparison of methods over line topology

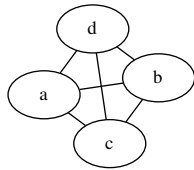
For next graphs I dropped SCP as not only is it inferior it is also completely wrong the programs behaviour is undefined for changes coming from two directions.



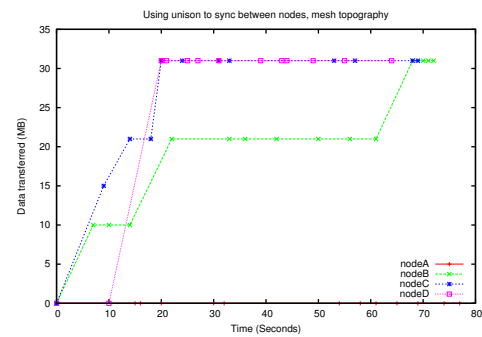
(a) Circle - generated graph of topology



(b) Unison running over circle topology



(c) Mesh - generated graph of topology



(d) Unison running over mesh topology

Figure 5: Comparison of different topologies

### 4.3 When to stop copying

After testing my program on some simple topologies one problem became clear. Each node would notice changes had occurred to a folder it was watching and would then try to copy these changes to other nodes that it was connected to. The problem was that if the changes came from one of its neighbour nodes this would cause an infinite loop of two nodes trying to copy changes to each other. This was particularly a problem when using SCP to copy. When using Unison this was not as much of a problem because it could detect that no changes had occur between the nodes and would stop syncing after one check (which had minimal overhead).

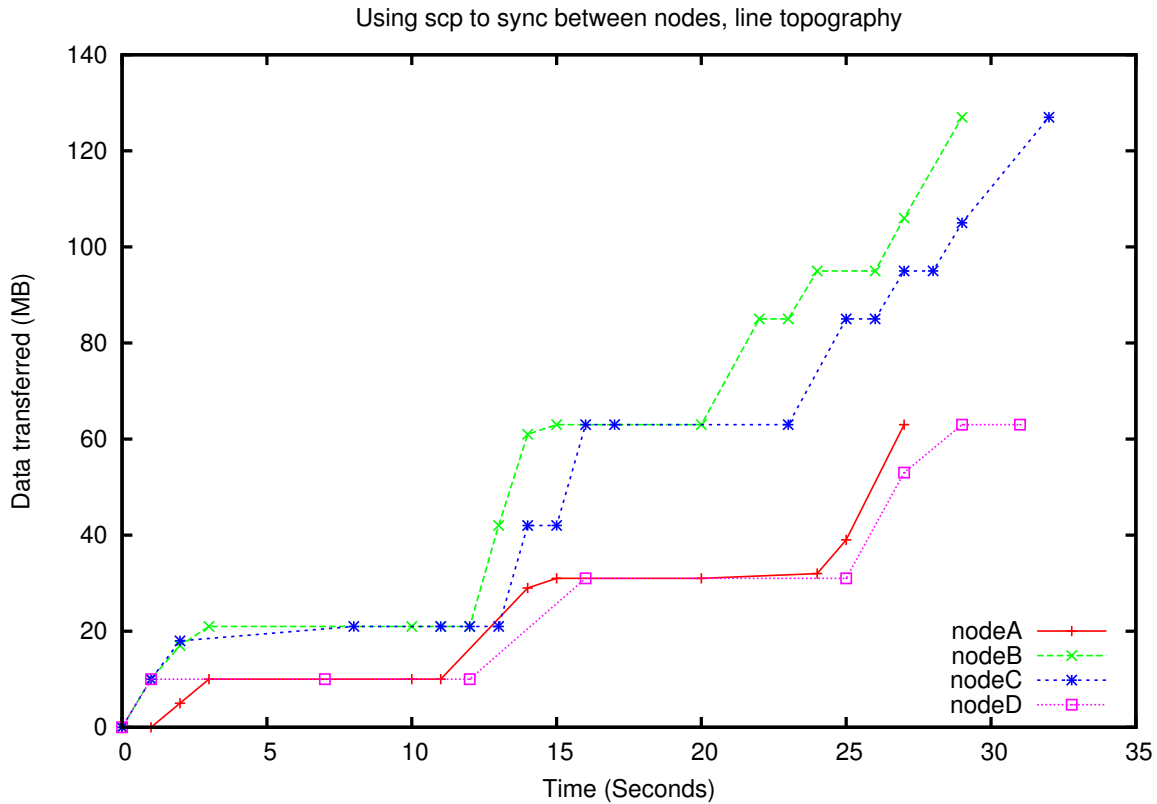


Figure 6: Line topology, using SCP, nodes copying data back and forth

Figure 6 shows three 10MB files being copied to a node in a line topology. The problem is that nodeB and nodeC continue to send data to each other even after every node has all of the files. NodeA receives a lot of data even though it was the source of the file changes.

The data points in figure 7 show that when using Unison although no extra data was sent Unison had to make checks to see whether there were any changes or not.

I used a configuration file to get around this problem. Each time a node synchronised with another node it would write out a configuration file telling the other node what files had been copied, who sent them and what the modification time of the files were.

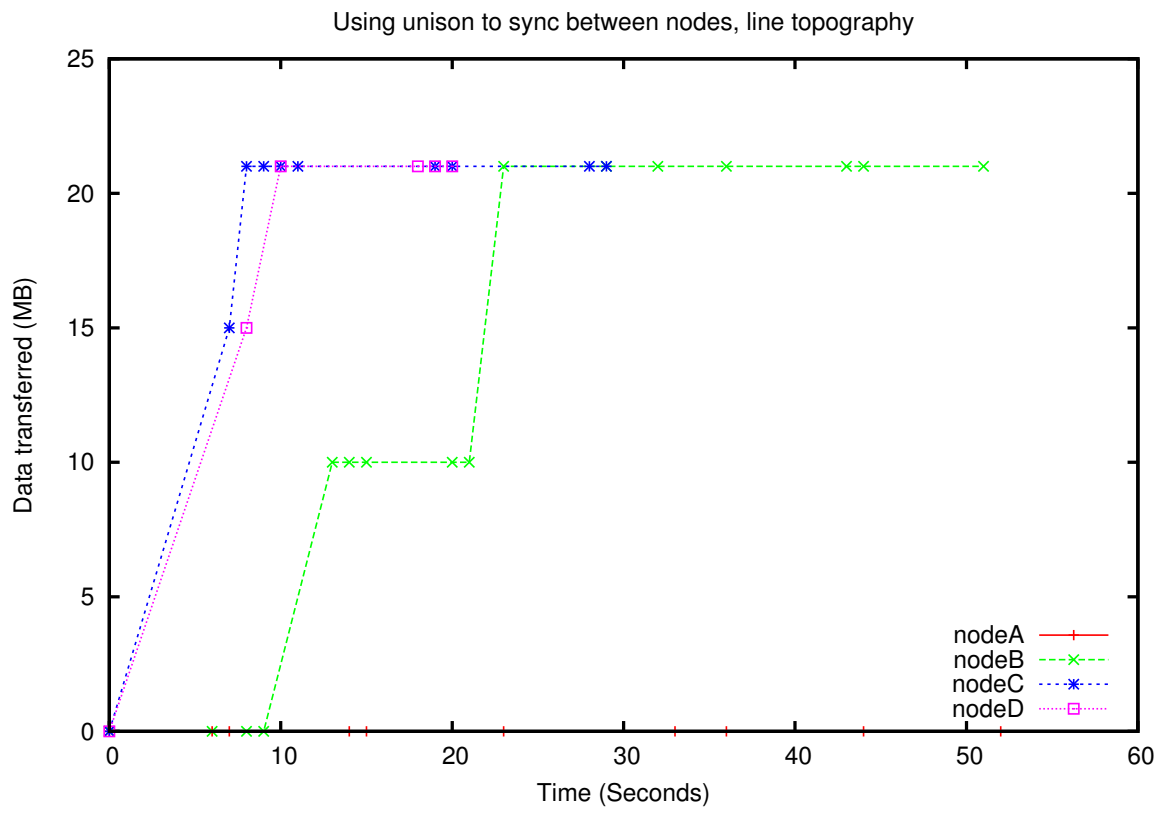


Figure 7: Line topology, using Unison, unnecessary checking

In this way a node could check if it was about to synchronise a file back to the node it received the file from or if it had local changes that were newer than a received file it could continue with its sync.

## 4.4 Dealing with Sub-nodes

I chose to classify directories as ‘sub nodes’ of a graph. The reason I choose directories is because they are easy to manage a configuration file of directories to keep in sync (from the users point of view). If we wanted to only synchronize certain files in a directory we could write a Unison configuration file with exclusions/inclusions in it. The other reason directories are a good choice is because I can have different directories in different places on different file systems by using symbolic links. I wanted to see how the freshness of different sub-nodes varied between nodes when the program was running.

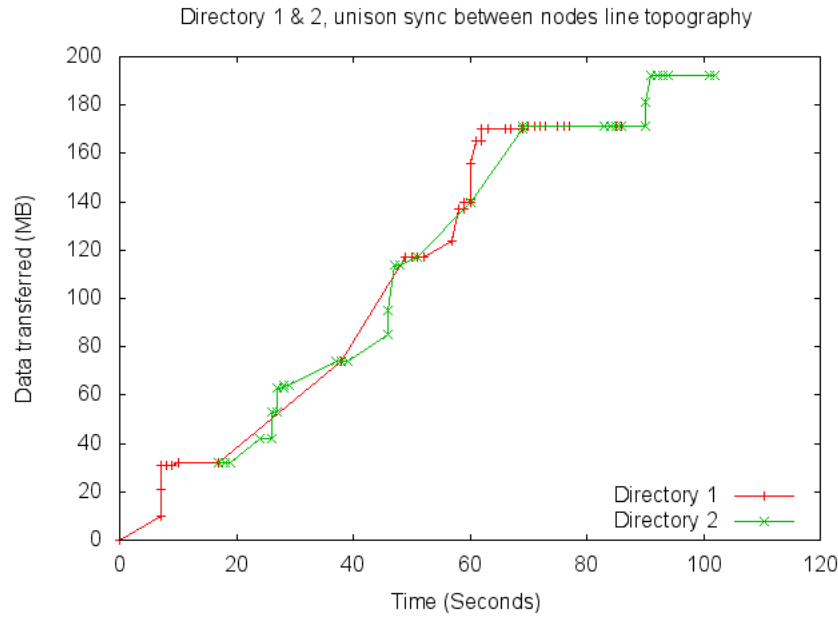


Figure 8: Line topology, using Unison, two different directories being synced



## 4.5 How often to sync

So how often should I sync once I noticed a change. If lots of small changes are occurring frequently it might be more efficient to perform a synchronisation after several changes have occurred. Given that there is overhead with each synchronisation, fewer copies means less data sent over the network.

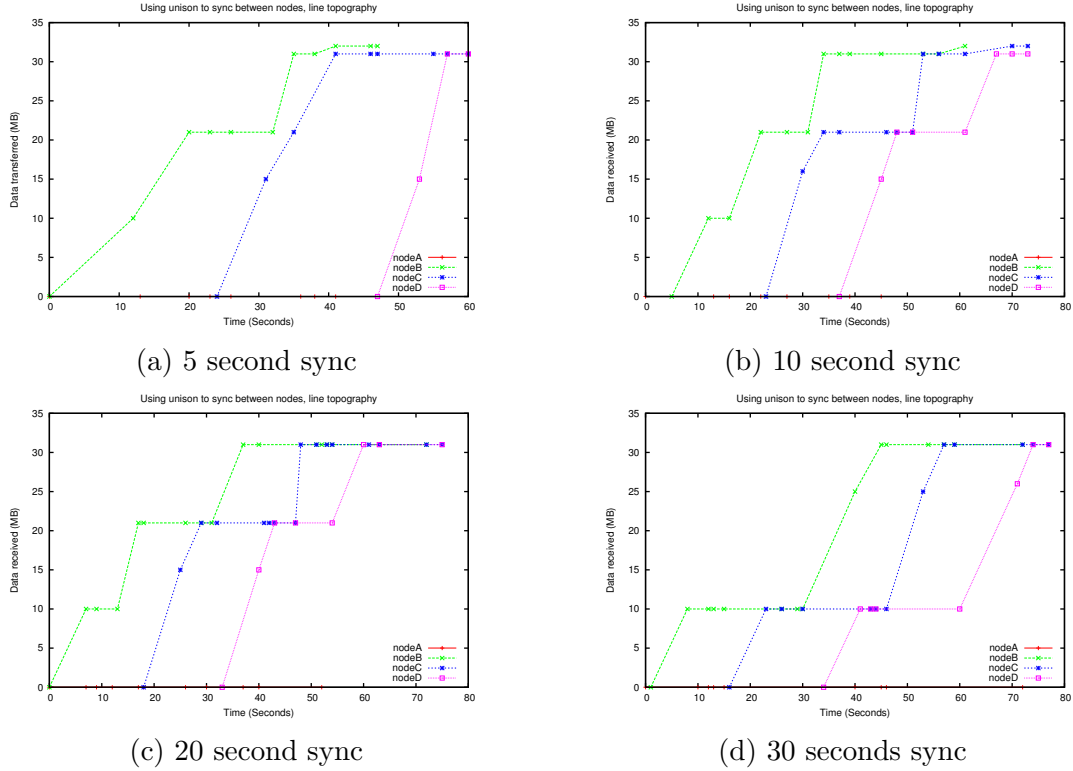


Figure 9: Comparison how frequently to sync

As you can see in figure when using Unison it is best to just sync as often as possible. This is because Unison only sends the difference between files and the network overhead associated with synchronising is negligible when dealing with large files.

SCP showed a noticeable difference however...?

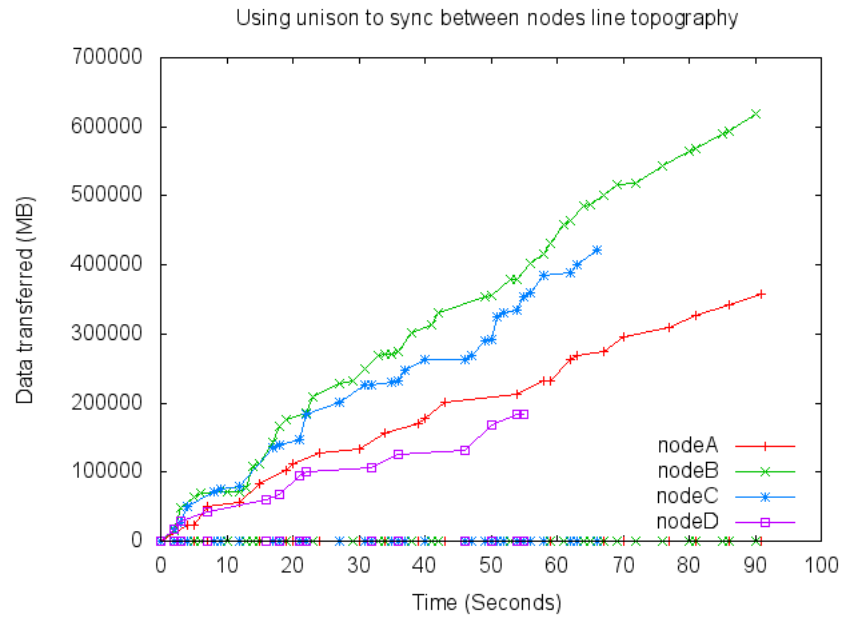


Figure 10: 2 seconds sleep text file

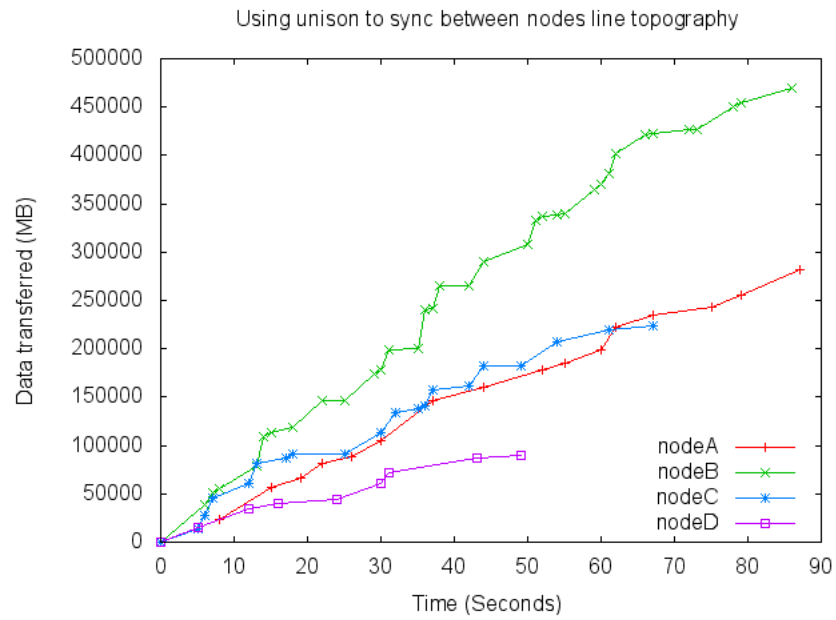


Figure 11: 5 seconds sleep text file

## 4.6 Wi-Fi vs 3G

Circle topology start at node A, node B is on Wi-Fi and node C is on 3G. Results and discussion to come.

# 5 Conclusion

## 5.1 Future Work

### 5.1.1 Mobile Nodes

### 5.1.2 Feedback

## 5.2 Results

My program does better than naïve copying, works in a variety of situations allows for fine grained user control.

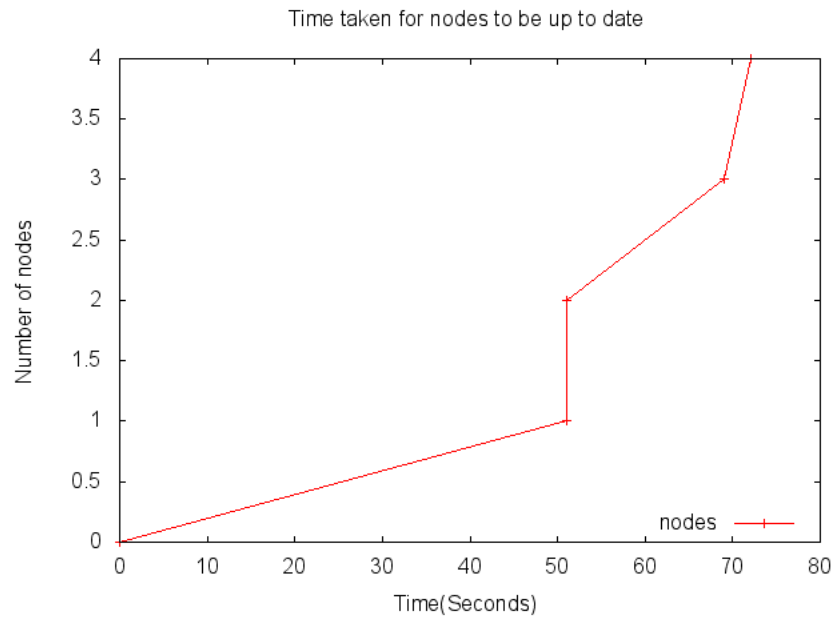


Figure 12: Unison, line, finishing times

## A WatchAndSync.py

```
1 import pyinotify, os, subprocess, argparse, socket, time, glob
   , datetime
2 import readnet
```

```

3
4 wm = pyinotify.WatchManager()
5 watchedfolders = {}
6 homepath = "/home/cal/Documents/Private-Sync/"
7 #homepath = "/Users/calum/Documents/Private-Sync/"
8
9 parser = argparse.ArgumentParser()
10 parser.add_argument("-c", "--scp", action="store_true", help="
    Copy_using_scp")
11 parser.add_argument("-r", "--rsync", action="store_true", help="
    Copy_using_rsync")
12 args = parser.parse_args()
13
14 class Tools():
15     def updateFolderInfo(self, wfolds):
16         f = open( './folders.dat', 'w')
17         for fold in wfolds:
18             f.write(fold + " ")
19             for i in range(0, len(wfolds[fold]) - 1):
20                 f.write(wfolds[fold][i] + " ")
21             f.write(wfolds[fold][len(wfolds[fold]) - 1] + "\n")
22         f.close()
23
24     def timeElapsed(self, dtstamp, diff):
25         if diff == "*":
26             print "Sync ASAP"
27             return
28         diff = int(diff)
29         FMT = '%Y-%m-%d %H:%M:%S.%f'
30         #FMT = '%Y-%m-%d %H:%M:%S'
31         tdelta = datetime.datetime.now() - datetime.datetime.
            strptime(dtstamp, FMT)
32         print tdelta.total_seconds()
33         timeDiff = tdelta.total_seconds()
34         if (timeDiff >= diff):
35             print "Time_period_reached"
36         else:
37             print "Time_not_elapsed, sleeping_for " + str(diff
                - timeDiff + 1)
38             time.sleep(int(diff - timeDiff + 1))
39
40 class MyEventHandler(pyinotify.ProcessEvent):
41     def flipIP(self, ip):

```

```

42         octets = ip.split(".")
43         if(octets[3] == "1"):
44             octets[3] = "2"
45         elif(octets[3] == "2"):
46             octets[3] = "1"
47         else:
48             octets[3] = "1"
49         return ".".join(octets)
50
51     #Get the last modified time of a file
52     def getModTime(self, path):
53         try:
54             return time.ctime(os.path.getmtime(path))
55         except Exception, e:
56             return ""
57
58     #Deprecated - Check for IP not to copy too
59     def getStopInfo(self):
60         stopIP = ["", ""]
61         try:
62             o = open("./stop", 'r')
63             stopIP = o.read().split()
64             o.close()
65         except IOError, e:
66             pass
67         return stopIP
68
69     def inStopFile(self, ip, path):
70         stopIPs = {}
71         stop = False
72         modTime = self.getModTime(path)
73         while True:
74             tmpcount = 0
75             print "Files _found:_ " + str(glob.glob("Stop-*"))
76             for files in glob.glob("Stop-*"):
77                 #print "File: " + str(files)
78                 if ".tmp" in files:
79                     tmpcount += 1
80                     time.sleep(5)
81                     break
82                 f = open(files, "r");
83                 for line in f:
84                     l = line.split()

```

```

85         if self.exclusions(l[1]):
86             print str(l[1]) + " was in ignore file
               skipping"
87         else:
88             print "local" + str(path) + " modtime
               : " + modTime
89             print "Stop" + l[1] + " modtime: " +
               str(l[2:])
90             ts1 = time.strptime(modTime, "%a_%b_%d_
               %H:%M:%S_%Y")
91             ts2 = time.strptime(" ".join(l[2:]), "%
               a_%b_%d_%H:%M:%S_%Y")
92             print "local <= stop: " + str(ts1 <=
               ts2)
93             #if l[0] == ip and l[1] == path and
               ts1 <= ts2:
94             #If IP sending to has sent data more
               recently don't send back
95             if l[0] == ip and ts1 <= ts2:
96                 print "Stop = True, file: " + l[0]
97                 stop = True
98             else:
99                 stopIPs[l[0]] = [l[1], " ".join(l
               [2:])]
100
101         if stop:
102             f.close()
103             #f = open(files, "w")
104             #for k in stopIPs.keys():
105             #     f.write(k + " " + stopIPs[k][0] + " "
               + stopIPs[k][1] + "\n")
106             #f.close()
107             #stopIPs.clear()
108             return True
109
110         f.close()
111         #stopIPs.clear()
112     if tmpcount == 0:
113         break
114
115     return False
116

```

```

117     #Set flag on other server telling it not to immediately
118     try and copy data here
119     def setStopFileUniq(self, ip, myIP, path, folder):
120         nodename = self.getNodeName()
121         #print "ssh", ip, "echo " + myIP + " " + path + " " +
122         self.getModTime(path) + " >> " + homedir + "Stop-"
123         + nodename + ".tmp;"
124         #subprocess.call(["ssh", ip, "echo " + myIP + " " + path
125         + " " + self.getModTime(path) + " >> " + homedir +
126         "Stop-" + nodename + ".tmp;"])
127         subprocess.call(["ssh", ip, "rm_" + homedir + "Stop-" +
128         nodename + ".tmp;"])
129         for cpFile in glob.glob(folder + "/*"):
130             subprocess.call(["ssh", ip, "echo_" + myIP + "_" +
131             cpFile + "_" + self.getModTime(cpFile) + " _>>_"
132             + homedir + "Stop-" + nodename + ".tmp;"])
133
134     #Sets the config files on the remote node
135     def beginCopy(self, ip):
136         nodename = self.getNodeName()
137         print "ssh", ip, "touch_" + homedir + "Stop-" +
138         nodename + ".tmp; mv_" + homedir + "Stop-" +
139         nodename + "_" + homedir + "Stop-" + nodename + ".
140         tmp;"
141         subprocess.call(["ssh", ip, "touch_" + homedir + "Stop-
142         " + nodename + ".tmp; mv_" + homedir + "Stop-" +
143         nodename + "_" + homedir + "Stop-" + nodename + ".
144         tmp;"])
145
146     #Moves the Stop files back into place
147     def endCopy(self, ip):
148         nodename = self.getNodeName()
149         print "ssh", ip, "mv_" + homedir + "Stop-" + nodename +
150         ".tmp_" + homedir + "Stop-" + nodename
151         subprocess.call(["ssh", ip, "mv_" + homedir + "Stop-" +
152         nodename + ".tmp_" + homedir + "Stop-" + nodename
153         ])
154
155     #Get node name from whoami file
156     def getNodeName(self):
157         w = open(homedir + "whoami", "r")
158         nodename = w.read()
159         nodename = nodename[0].upper()

```

```

143         w.close()
144         return nodename
145
146     #Deprecated stop file
147     def setStopFile(self, ip, myIP, path):
148         subprocess.call(["ssh", ip, "echo_" + myIP + "_" + path
149             + ">" + homedir + "stop"])
150         print "ssh", ip, "echo_" + myIP + ">" + homedir + "
151             stop"
152
153     def rmTree(self, path):
154         subprocess.call(["ssh", ip, "rm_r_" + path + ""])
155         print "ssh", ip, "rm_r_" + path + ""
156
157     #Exclude files matching patterns in the ignore file
158     def exclusions(self, path):
159         try:
160             f = open("./ignore", 'r')
161             for line in f:
162                 if line.rstrip() in path:
163                     #print "Ignoring: " + path
164                     return True
165             f.close()
166         except error, e:
167             print e
168         return False
169
170     def initFileSync(self, event):
171         if self.exclusions(event.pathname):
172             #print "Excluded returning"
173             return
174         pathparts = event.pathname.split("/")
175         foldName = "/" .join(pathparts[0:len(pathparts)-1])
176         print "Removing_watch_on:" + foldName
177         wm.rm_watch(wm.get_wd(foldName), rec=True)
178         self.fileSync(event)
179         print "Putting_watch_back_on:" + foldName
180         wm.add_watch(foldName.rstrip(), pyinotify.ALL_EVENTS,
181             rec=True, auto_add=True)
182
183     #Sync files
184     def fileSync(self, event):
185         t = Tools()

```



```

183     if os.path.isdir(event.pathname):
184         print "Watching:_" , event.pathname
185     for folder in watchedfolders.keys():
186         print "For_each_folder:_" + str(folder) + "_in_"
187         watchedfolder_keys"
188         if folder in event.pathname:
189             for i in range(0, len(watchedfolders[folder])
190             ,4):
191                 ip = watchedfolders[folder][i]
192                 path = watchedfolders[folder][i+1]
193                 waitTime = watchedfolders[folder][i+2]
194                 lastTime = watchedfolders[folder][i+3]
195                 print "Wait:_" + str(waitTime) + "_Last:_"
196                 + str(lastTime)
197                 print "Current_ip_and_path:_" + ip + "_" +
198                 path
199                 readnet.logIPtraffic(ip, event.pathname)
200                 myIP = readnet.getMyIP(ip)
201                 subprocess.call(["ssh",ip,"/usr/bin/python
202                 _" + hompath + "readnet.py_i_" + myIP
203                 + "_f_" + event.pathname])
204                 print "ssh",ip,"'/usr/bin/python_" +
205                 hompath + "readnet.py_i_" + myIP + "_-
206                 f_" + event.pathname + ""
207                 fparts = folder.split("/")
208                 fname = fparts[len(fparts)-1]
209                 #stopIP = self.getStopInfo()
210                 #print "STOP: " + stopIP[0] + " " + stopIP
211                 [1]
212                 #if stopIP[0] == ip and stopIP[1] == event
213                 .pathname:
214                 if self.inStopFile(ip, event.pathname):
215                     print "STOPPED_to_" + ip + "_" + path
216                     #os.remove("./stop");
217                 else:
218                     print "CONTINUE"
219                     t.timeElapsed(lastTime, waitTime)
220                     watchedfolders[folder][i+3] = str(
221                     datetime.datetime.now())
222                     t.updateFolderInfo(watchedfolders)
223                     self.beginCopy(ip)
224                     if args.scp:

```

```

214         #print "SCP: For cpFile in " +
215         folder
216         for cpFile in glob.glob(folder + "
217         /*"):
218             #print "SCP GLOB:" + cpFile
219             print "scp","-rp",cpFile,ip +
220             ":" + cpFile + ".tmp"
221             subprocess.call(["scp","-rp",
222             cpFile,ip + ":" + cpFile + "
223             .tmp"])
224             #subprocess.call(["ssh",ip,"
225             yes y | find /tmp/" + fname
226             + " -type f -exec cp -p {} "
227             + path + fname + "/" \; rm /
228             tmp/" + fname])
229             print "ssh",ip,"mv_" + cpFile
230             + ".tmp_" + cpFile
231             subprocess.call(["ssh",ip,"mv_"
232             + cpFile + ".tmp_" +
233             cpFile])
234             print "END_SCP_GLOB"
235     elif args.rsync:
236         print "rsync","-rt",folder,ip + ":"
237         + path
238         subprocess.call(["rsync","-rt",
239         folder,ip + ":" + path])
240     else:
241         time.sleep(5)
242         print "unison","-batch","-
243         confirmbigdel=false","-times",
244         folder,"ssh://" + ip + "/" +
245         path + fname
246         subprocess.call(["unison","-batch"
247         ,"-confirmbigdel=false","-times"
248         ,folder,"ssh://" + ip + "/" +
249         path + fname])
250     print "Set_stop_files_uniq_" + event.
251     pathname
252     self.setStopFileUniq(ip,myIP,event.
253     pathname,folder)
254     self.endCopy(ip)
255     subprocess.call(["ssh",ip,"/usr/bin/python
256     _" + homedir + "readnet.py_-i_" + myIP

```

```

234         + "_f_" + event.pathname])
235         readnet.logIPtraffic(ip, event.pathname)
236
237     #def process_IN_CREATE(self, event):
238     #     print "Create:", event.pathname
239     def process_IN_DELETE(self, event):
240         print "Delete: ", event.pathname
241         #self.initFileSync(event)
242     def process_IN_CREATE(self, event):
243         print "CREATE: ", event.pathname
244         self.initFileSync(event)
245     def process_IN_MOVED_FROM(self, event):
246         print "Move from: ", event.pathname
247         #self.initFileSync(event)
248     def process_IN_MODIFY(self, event):
249         #print "Modify: ", event.pathname
250         self.initFileSync(event)
251     def process_IN_MOVED_TO(self, event):
252         print "Move to: ", event.pathname
253         self.initFileSync(event)
254
255 def main():
256     t = Tools()
257     f = open('./folderstowatch', 'r')
258
259     for folder in f:
260         if(folder[0] == '#'):
261             pass
262         else:
263             info = folder.split()
264             wm.add_watch(info[0].rstrip(), pyinotify.ALL_EVENTS
265                           , rec=True, auto_add=True)
266             print "Watching: ", info[0].rstrip()
267             if info[0] not in watchedfolders.keys():
268                 watchedfolders[info[0].rstrip()] = []
269                 watchedfolders[info[0].rstrip()].append(info[1])
270                 watchedfolders[info[0].rstrip()].append(info[2])
271                 watchedfolders[info[0].rstrip()].append(info[3])
272                 watchedfolders[info[0].rstrip()].append(str(
273                     datetime.datetime.now()))
274     f.close()

```

```

274     try:
275         f = open( './folders.dat', 'r' )
276         for folder in f:
277             if( folder[0] == '#' ):
278                 pass
279             else:
280                 info = folder.split()
281                 if info[0] in watchedfolders.keys():
282                     del watchedfolders[info[0].rstrip()]
283                     #wm.add_watch( info[0].rstrip(), pyinotify.
284                       ALL_EVENTS, rec=True, auto_add=True)
285                     #print "Watching: ", info[0].rstrip()
286                     if info[0] not in watchedfolders.keys():
287                         watchedfolders[info[0].rstrip()] = []
288                         watchedfolders[info[0].rstrip()].append(
289                             info[1])
290                         watchedfolders[info[0].rstrip()].append(
291                             info[2])
292                         watchedfolders[info[0].rstrip()].append(
293                             info[3])
294                         watchedfolders[info[0].rstrip()].append(
295                             str(datetime.datetime.now()))
296                 else:
297                     print "Removing:_" + info[0]
298             f.close()
299     except IOError, e:
300         print "Folders.dat_does_not_exist, _skipping"
301
302     t.updateFolderInfo(watchedfolders)
303
304     #print watchedfolders
305     eh = MyEventHandler()
306
307     notifier = pyinotify.Notifier(wm, eh)
308     notifier.loop()
309
310 if __name__ == '__main__':
311     main()

```

## B ReadNet.py

```

1 import subprocess, datetime, socket, argparse
2

```

```

3 homedir = "/home/cal/Documents/Private-Sync/"
4 #homedir = "/Users/calum/Documents/Private-Sync/"
5
6 parser = argparse.ArgumentParser()
7 parser.add_argument( '-i', action="store", dest='ip', help='IP_
    address_to_record_for' )
8 parser.add_argument( '-f', action="store", dest='fold', help='
    Folder_to_record_for' )
9
10 interfacenames = []
11
12 w = open(homedir + "whoami", "r")
13 nodename = w.read()
14 nodename = nodename[0]
15 w.close()
16
17 #Get my ip corresponding to the interface with ipaddr
18 def getMyIP(ipaddr):
19     route = subprocess.check_output("ip_route_get_" + ipaddr,
        shell=True)
20     words = route.split()
21     interface = ""
22     for word in words:
23         if word.startswith("eth"):
24             interface = word
25             #print interface
26             break
27     ifconf = subprocess.check_output("ifconfig_" + interface,
        shell=True)
28     words = ifconf.split()
29     now = False
30     for word in words:
31         if word == "inet":
32             now = True
33         elif now:
34             word = word.split(":")
35             #print word[1]
36             return word[1]
37
38 #Log interface corresponding to ipaddr
39 def logIPtraffic(ipaddr, folder):
40     route = subprocess.check_output("ip_route_get_" + ipaddr,
        shell=True)

```

```

41     words = route.split()
42     interface = ""
43     for word in words:
44         if word.startswith("eth"):
45             interface = word
46             #print interface
47             break
48     writeIface(interface, folder)
49
50 def writeIface(iface, folder):
51     ifs = subprocess.check_output("ifconfig -s", shell=True)
52     ilines = ifs.split("\n")
53     for i in range(1, len(ilines)-1):
54         interfacenames.append(ilines[i].split()[0])
55     output = subprocess.check_output("ifconfig", shell=True)
56     splitput = output.split()
57     interface = False
58     interfacename = ""
59     nex = ""
60     count = 0
61     upload = 0
62     download = 0
63     for split in splitput:
64         if split in interfacenames:
65             interface = True
66             interfacename = split
67             #print interfacename
68         if(nex != ""):
69             sp = split.split(":")
70             if(sp[0] == "bytes"):
71                 if(nex == "RX"):
72                     download = int(sp[1])
73                 else:
74                     upload = int(sp[1])
75             nex = ""
76             count += 1
77             if(count == 2):
78                 interface = False
79                 if interfacename == iface:
80                     f = open(homepath + "log/" \
81 + "node" + nodename.upper() + "-" \
82 + iface + ".log", 'a')
83                     f.write("#D_" + folder + "\n")

```

```

84             f.write(str(datetime.datetime.now()) +
85                     " " + interfacename + "download:" +
86                     + str(download) + "upload:" + str
87                     (upload) + "\n")
88             f.close()
89             count = 0
90         elif(interface):
91             if(split == "RX" or split == "TX"):
92                 nex = split
93
94     #Log all interfaces
95     def main():
96         ifs = subprocess.check_output("ifconfig -s", shell=True)
97         ilines = ifs.split("\n")
98         for i in range(1, len(ilines)-1):
99             interfacenames.append(ilines[i].split()[0])
100         output = subprocess.check_output("ifconfig", shell=True)
101         splitput = output.split()
102         interface = False
103         interfacename = ""
104         nex = ""
105         count = 0
106         upload = 0
107         download = 0
108         for split in splitput:
109             if split in interfacenames:
110                 interface = True
111                 interfacename = split
112                 #print interfacename
113                 if(nex != ""):
114                     sp = split.split(":")
115                     if(sp[0] == "bytes"):
116                         if(nex == "RX"):
117                             download = int(sp[1])
118                         else:
119                             upload = int(sp[1])
120                     nex = ""
121                     count += 1
122                 if(count == 2):
123                     interface = False
124                     f = open(homepath + "log/" +
125                             + str(socket.gethostname()) + "-" +
126                             + interfacename + ".log", 'a')

```

```

124             f.write(str(datetime.datetime.now()) + "\n"
                      + interfacename + "\ndownload:\n" + str(
                          download) + "\nupload:\n" + str(upload) +
                          "\n")
125             f.close()
126             count = 0
127         elif(interface):
128             if(split == "RX" or split == "TX"):
129                 nex = split
130
131 if __name__ == "__main__":
132     args = parser.parse_args()
133     if args.ip != None:
134         logIPtraffic(args.ip, args.fold)
135         #getMyIP(args.ip)
136     else:
137         pass
138     main()

```

## C onTheFly.sh

```

1 vm_name_arr=("Ubuntu-Pool" "Ubuntu-Silence" "Ubuntu-Wild" "
  Ubuntu-Spheros")
2 vm_addr_arr=("192.168.0.28" "192.168.0.27" "192.168.0.19" "
  192.168.0.14")
3 intnetarr=("lion" "tiger" "cat" "dog" "fish" "kiwi")
4 #These should all be in one big dictionary apart from inet
  names
5 letterarr=("a" "b" "c" "d" "e" "f" "g")
6 ifcountarr=(2 2 2 2 2 2 2 2)
7 ethcountarr=(1 1 1 1 1 1 1 1)
8 incount=1
9 bigncount=2
10 littlencount=1
11 folderpath="/home/cal/Documents/t03"
12 folderpath2="/home/cal/Documents/t02"
13 homepath="/home/cal/Documents/Private-Sync/"
14 waitTime=10
15
16 function clear_ifaces() {
17     i=0
18     while [ "$i" -lt "${#vm_name_arr[@]}" ]; do
19         VBoxManage modifyvm ${vm_name_arr[$i]} --nic2 none

```



```

20      echo "VBoxManage modifyvm _${vm_name_arr[$i]} _--nic2 _
        none"
21      VBoxManage modifyvm ${vm_name_arr[$i]} --nic3 none
22      echo "VBoxManage modifyvm _${vm_name_arr[$i]} _--nic3 _
        none"
23      VBoxManage modifyvm ${vm_name_arr[$i]} --nic4 none
24      echo "VBoxManage modifyvm _${vm_name_arr[$i]} _--nic4 _
        none"
25      let "i++"
26  done
27 }
28
29 function clear_watched_folders() {
30     i=0
31     while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
32         ssh cal@${vm_addr_arr[$i]} "echo _\"#Local folder path
            to watch, host to copy to, remote dir to copy to,
            min time between syncs\" > /home/cal/Documents/
            Private-Sync/folderstowatch; echo ${letterarr[$i]} >
            /home/cal/Documents/Private-Sync/whoami"
33         let "i++"
34     done
35 }
36
37 function git_pull() {
38     i=0
39     while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
40         echo "ssh _cal@${vm_addr_arr[$i]} _\"cd /home/cal/
            Documents/Private-Sync; git pull origin master\""
41         ssh cal@${vm_addr_arr[$i]} "cd _/home/cal/Documents/
            Private-Sync; _git_pull _origin _master"
42         let "i++"
43     done
44 }
45
46 function search_letters() {
47     index=0
48     while [ "$index" -lt "${#letterarr[@]}" ]; do
49         if [ "${letterarr[$index]}" = "$1" ]; then
50             echo $index
51             return
52         fi
53         let "index++"

```

```

54     done
55     echo "None"
56 }
57
58 function vbmMOD {
59     echo "VBoxManage modifyvm $1 --nic$3 intnet"
60     VBoxManage modifyvm $1 --nic$3 intnet
61     echo "VBoxManage modifyvm $1 --intnet$3 $2"
62     VBoxManage modifyvm $1 --intnet$3 $2
63 }
64
65 function gatherLogs {
66     index=0
67     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
68         echo "scp cal@${vm_addr_arr[$index]}:/home/cal/
69             Documents/Private-Sync/log/*../logs/"
70         scp cal@${vm_addr_arr[$index]}:/home/cal/Documents/
71             Private-Sync/log/*../logs/
72         let "index++"
73     done
74 }
75
76 function clean {
77     index=0
78     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
79         echo "ssh cal@${vm_addr_arr[$index]} \"rm ${homepath}
80             log/*; rm ${homepath}Stop-*; rm ${homepath}folders.
81             dat\""
82         ssh cal@${vm_addr_arr[$index]} "rm ${homepath}log/*;
83             rm ${homepath}Stop-*; rm ${homepath}folders.dat"
84         let "index++"
85     done
86 }
87
88 function cleanFold {
89     index=0
90     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
91         echo "ssh cal@${vm_addr_arr[$index]} \"rm -rf ${
92             folderpath}/*;\n\""
93         ssh cal@${vm_addr_arr[$index]} "rm -rf ${folderpath
94             }/*;"
95         let "index++"
96     done

```

```

90 }
91
92 function sendKeys {
93     index=0
94     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
95         ssh cal@${vm_addr_arr[$index]} "rm /home/cal/.ssh/
           authorized_keys"
96         for file in /Users/calum/.ssh/*.pub; do
97             echo "$file"
98             echo "cat $file | ssh cal@${vm_addr_arr[$index]} \
           "cat >> /home/cal/.ssh/authorized_keys\"
99             cat $file | ssh cal@${vm_addr_arr[$index]} "cat >>
           /home/cal/.ssh/authorized_keys"
100         done
101         let "index++"
102     done
103     #for file in /Users/calum/.ssh/*.pub; do
104     #    echo "$file"
105     #    cat $file | ssh cal@192.168.0.17 "cat >> /home/cal/.
           ssh/testfile"
106     #    echo "cat $file | ssh cal@192.168.0.17 \"cat >> /home
           /cal/.ssh/testfile\"
107     #done
108 }
109
110 function ifconf {
111     echo "ssh cal@$1 'sudo /sbin/ifconfig eth$2 192.168.$3.$4
           netmask 255.255.255.0 up; echo \" $folderpath 192.168.$3.
           $5 /home/cal/Documents/ $waitTime\" >> /home/cal/
           Documents/Private-Sync/folderstowatch'"
112     ssh cal@$1 "sudo /sbin/ifconfig eth$2 192.168.$3.$4
           netmask 255.255.255.0 up; echo \" $folderpath 192.168.$3.
           $5 /home/cal/Documents/ $waitTime\" >> /home/cal/
           Documents/Private-Sync/folderstowatch" < /dev/null
113 }
114
115 function ifconf2 {
116     echo "ssh cal@$1 \"sudo /sbin/ifconfig eth$2 192.168.$3.$4
           netmask 255.255.255.0 up; echo \" $folderpath 192.168.$3
           . $5 /home/cal/Documents/_*\" >> /home/cal/Documents/
           Private-Sync/folderstowatch; echo \" $folderpath2
           192.168.$3.$5 /home/cal/Documents/_*\" >> /home/cal/
           Documents/Private-Sync/folderstowatch\" < /dev/null"

```

```

117     ssh cal@$1 "sudo _/sbin/ifconfig _eth$2 _192.168.$3.$4 _
        netmask _255.255.255.0 _up; _echo _\" $folderpath 192.168.$3.
        $5 /home/cal/Documents/ *\" _>> _/home/cal/Documents/
        Private-Sync/folderstowatch; _echo _\" $folderpath2
        192.168.$3.$5 /home/cal/Documents/ *\" _>> _/home/cal/
        Documents/Private-Sync/folderstowatch" < /dev/null
118 }
119
120 if [ $2 == "vm" ]; then
121     clear_ifaces
122
123     while read line
124     do
125         first=$(echo "$line" | awk '{print $1}')
126         last=$(echo "$line" | awk '{print $(NF)}' | sed 's
            /[:]//g')
127         #echo "$first and $last"
128         index=$(search_letters $first)
129         if [ "$index" = "None" ]; then
130             #echo "None"
131             :
132         else
133             vmMOD ${vm_name_arr[$index]} ${intnetarr[$incount
                ]} ${ifcountarr[$index]}
134             #echo "in: $index"
135             (( ifcountarr[$index]++ ))
136             index=$(search_letters $last)
137             vmMOD ${vm_name_arr[$index]} ${intnetarr[$incount
                ]} ${ifcountarr[$index]}
138             #echo "in: $index"
139             (( ifcountarr[$index]++ ))
140             incount=$((incount+1))
141         fi
142     done <graphs/$1
143 elif [ $2 == "if" ]; then
144     clear_watched_folders
145
146     while read line
147     do
148         first=$(echo "$line" | awk '{print $1}')
149         last=$(echo "$line" | awk '{print $(NF)}' | sed 's
            /[:]//g')
150         echo "$first_and_$last"

```

```

151     index=$(search_letters $first)
152     if [ "$index" = "None" ]; then
153         #echo "None"
154         :
155     else
156         ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index]}
157             $bigncount $littlencount $(( $littlencount+1 ))
158         #echo "in: $index"
159         (( ethcountarr[$index]++ ))
160         (( littlencount++ ))
161         index=$(search_letters $last)
162         ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index]}
163             $bigncount $littlencount $(( $littlencount-1 ))
164         #echo "in: $index"
165         (( ethcountarr[$index]++ ))
166         incount=$((incount+1))
167         (( bigncount++ ))
168         (( littlencount-- ))
169     fi
170 done <graphs/$1
171 elif [ $2 == "if2" ]; then
172     clear_watched_folders
173 while read line
174 do
175     first=$(echo "$line" | awk '{print $1}')
176     last=$(echo "$line" | awk '{print $(NF)}' | sed 's/
177         /[:]//g')
178     echo "$first_and_$last"
179     index=$(search_letters $first)
180     if [ "$index" = "None" ]; then
181         #echo "None"
182         :
183     else
184         ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[
185             $index]} $bigncount $littlencount $((
186                 $littlencount+1 ))
187         #echo "in: $index"
188         (( ethcountarr[$index]++ ))
189         (( littlencount++ ))
190         index=$(search_letters $last)

```

```

187         ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[
            $index]} $bigncount $littlencount $((
                $littlencount-1 ))
188         #echo "in: $index"
189         (( ethcountarr[$index]++ ))
190         incount=$((incount+1))
191         (( bigncount++ ))
192         (( littlencount-- ))
193     fi
194 done <graphs/$1
195 elif [ $2 == "key" ]; then
196     sendKeys
197 elif [ $2 == "gather" ]; then
198     gatherLogs
199 elif [ $2 == "clean" ]; then
200     clean
201 elif [ $2 == "pull" ]; then
202     git_pull
203 elif [ $2 == "clean-fold" ]; then
204     cleanFold
205 elif [ $2 == "help" ]; then
206     echo "vm-----setup vm networking"
207     echo "if-----setup network addresses etc for each
        vm"
208     echo "if2-----setup network addresses etc for each
        vm for two folders"
209     echo "gather-----gather the logs in"
210     echo "clean-----clean out the logs/config files"
211     echo "clean-fold-----clean out the files folder"
212     echo "pull-----pull the latest code from the
        repository to each vm"
213     echo "help-----display this help message"
214 else
215     echo "Oops try again"
216 fi
217
218 neato -Tpng graphs/$1 > graphs/$1-graph.png

```