

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 . . . . .	15
4.4	Dealing with Sub-nodes . . . . .	17
4.5	How often to sync . . . . .	18
4.6	Wi-Fi vs 3G . . . . .	20
<b>5</b>	<b>Conclusion</b>	<b>20</b>
5.1	Future Work . . . . .	20
5.1.1	Mobile Nodes . . . . .	20
5.1.2	Feedback . . . . .	20
5.2	Results . . . . .	20
<b>A</b>	<b>WatchAndSync.py</b>	<b>20</b>
<b>B</b>	<b>ReadNet.py</b>	<b>33</b>
<b>C</b>	<b>onTheFly.sh</b>	<b>36</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.

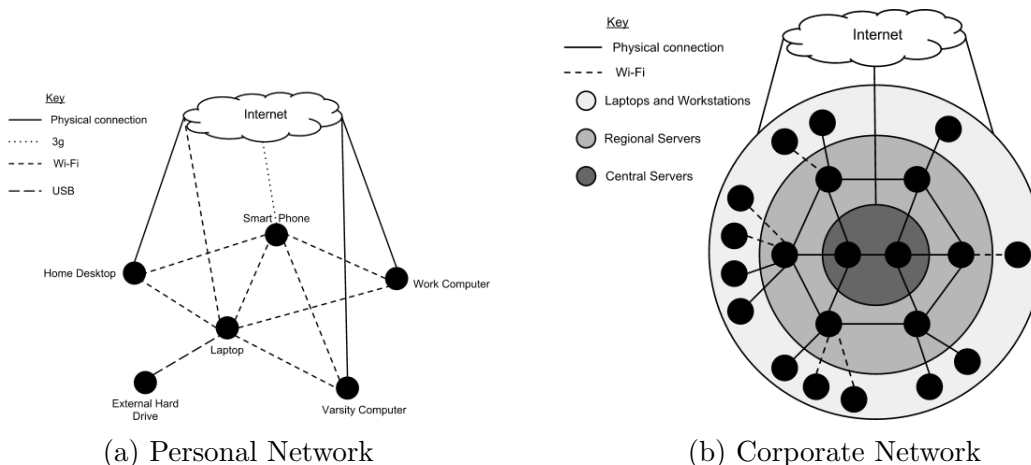


Figure 1: Example use cases

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<sup>2</sup>.

Graphviz is open source software for generating graphs. 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.

Graphviz takes input from scripts written in DOT language<sup>3</sup>. DOT language is a simple graph description language. I have written a script to read in these DOT files and interpret the graph to set up my virtual machines<sup>4</sup>. My script also calls a program called `neato` to generate graphs. This means I only have to write one DOT file to get a graph of my network topology and set up my virtual machines.

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 be-

---

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

<sup>3</sup><http://www.graphviz.org/doc/info/lang.html>

<sup>4</sup>Appendix A line 124 onwards

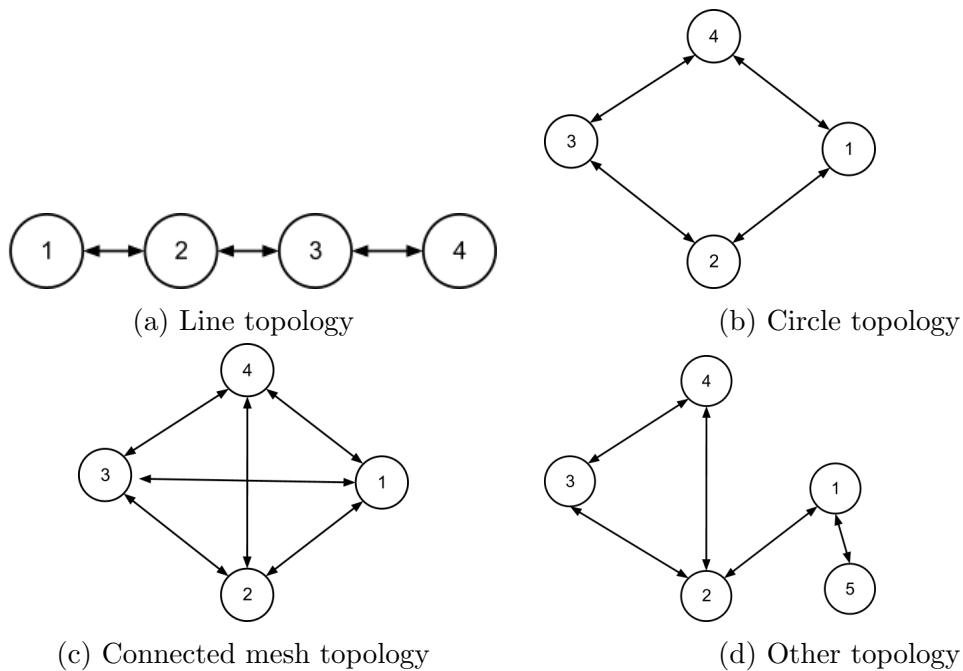


Figure 2: Simple network topologies

cause it means I will (hopefully) encounter fewer compatibility problems when running my program across different operating systems in the future.

## 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 syn-



chronising 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>5</sup> There is a Python module called `pyinotify`<sup>6</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>7</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>8</sup>.

I also looked at using the `kqueue`<sup>9</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

---

<sup>5</sup>[www.kernel.org/pub/linux/kernel/people/rml/inotify/README](http://www.kernel.org/pub/linux/kernel/people/rml/inotify/README)

<sup>6</sup><http://pyinotify.sourceforge.net/>

<sup>7</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>8</sup><http://pypi.python.org/pypi/MacFSEvents/0.2.1>

<sup>9</sup><http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man2/kqueue.2.html>

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.

## 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>10</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.

---

<sup>10</sup><http://www.cis.upenn.edu/~bcpierce/unison/>

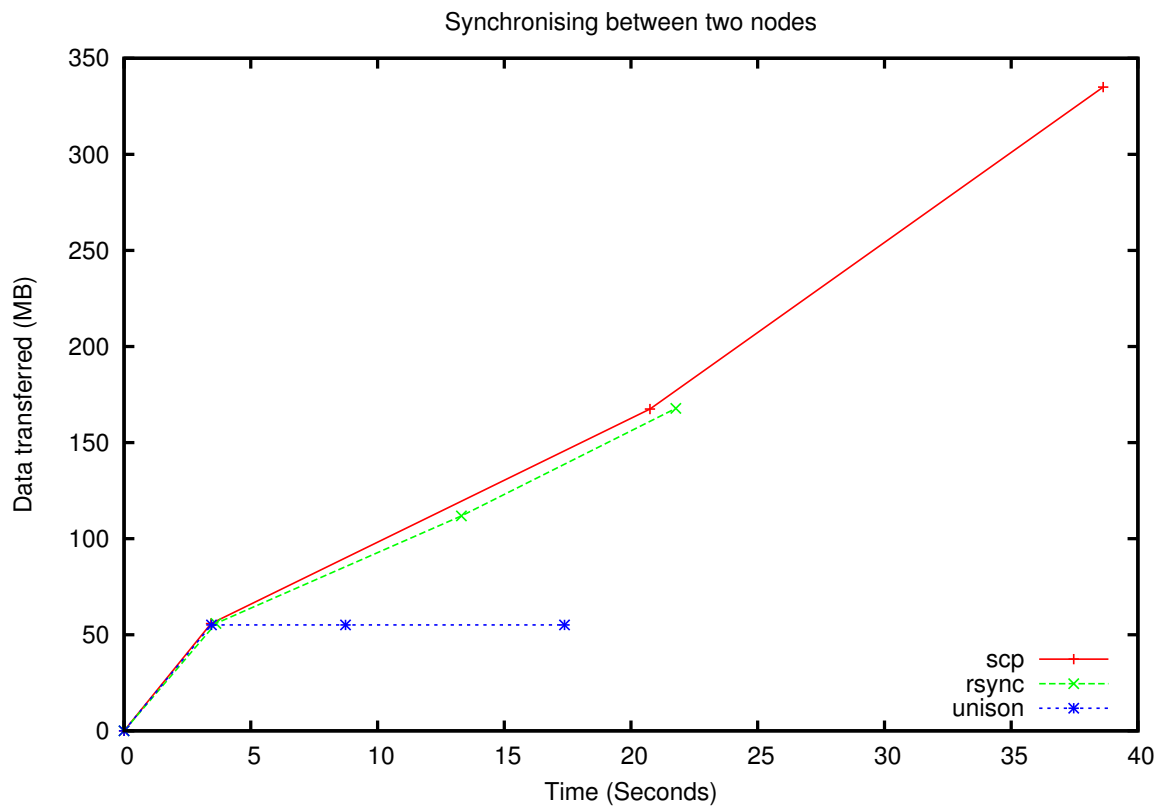


Figure 3: Comparison of SCP, Rsync, Unison. Three identical 50MB files with different names are being transferred between two nodes. Each point on the graph represents the start or end of a sync.

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

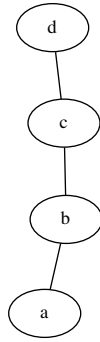
Rsync copies the expected 150MB for three 50MB files. Figure 3 illustrates another advantage of Unison over Rsync. The graph shows three zero filled binary files being copied from one node to another one after the other. Unison recognised that even though the files were named differently they were the 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 trialled 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.

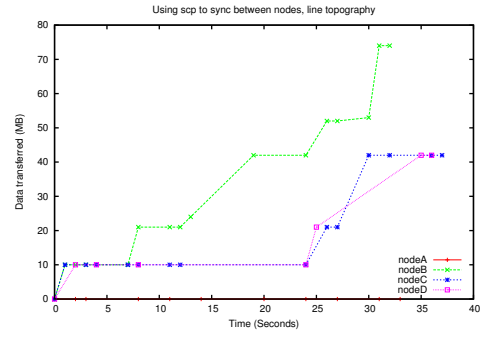
## 4.2 Full graph replication

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

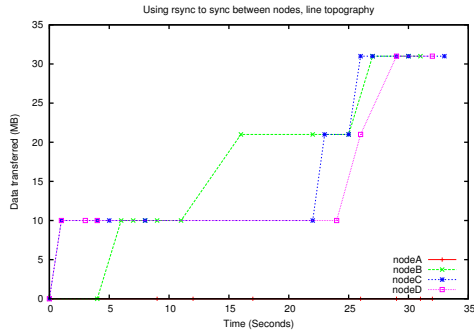
For next graphs I dropped SCP as not only is it inferior it is also completely wrong the programs behaviour is undefined for changes coming from two directions.



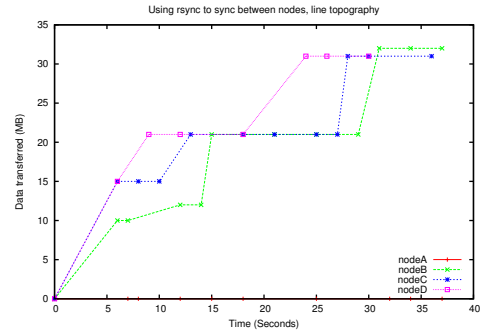
(a) Line - Generated Graph of Topology



(b) SCP

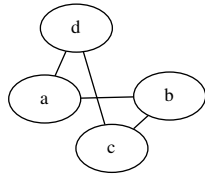


(c) Rsync

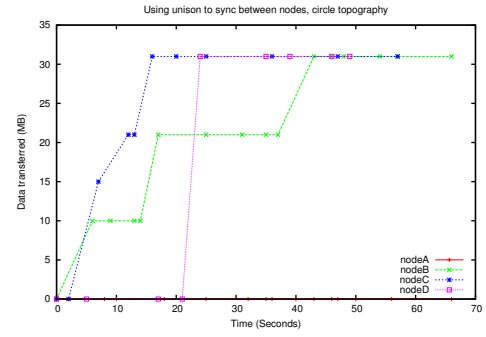


(d) Unison

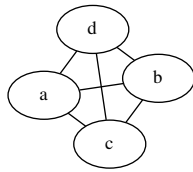
Figure 4: Comparison of methods over line topology



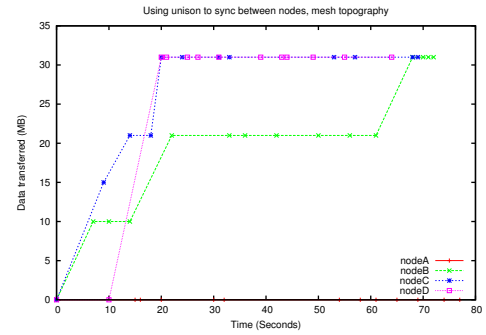
(a) Circle - generated graph of topology



(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

The way my program works it that Each node notices when changes have occurred to a folder it is watching and when a change occurs, copy these changes to other nodes that it is connected to. After testing my program on some simple topologies one problem became clear. 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 occurred between the nodes and would stop synchronising after one check (which had minimal overhead).

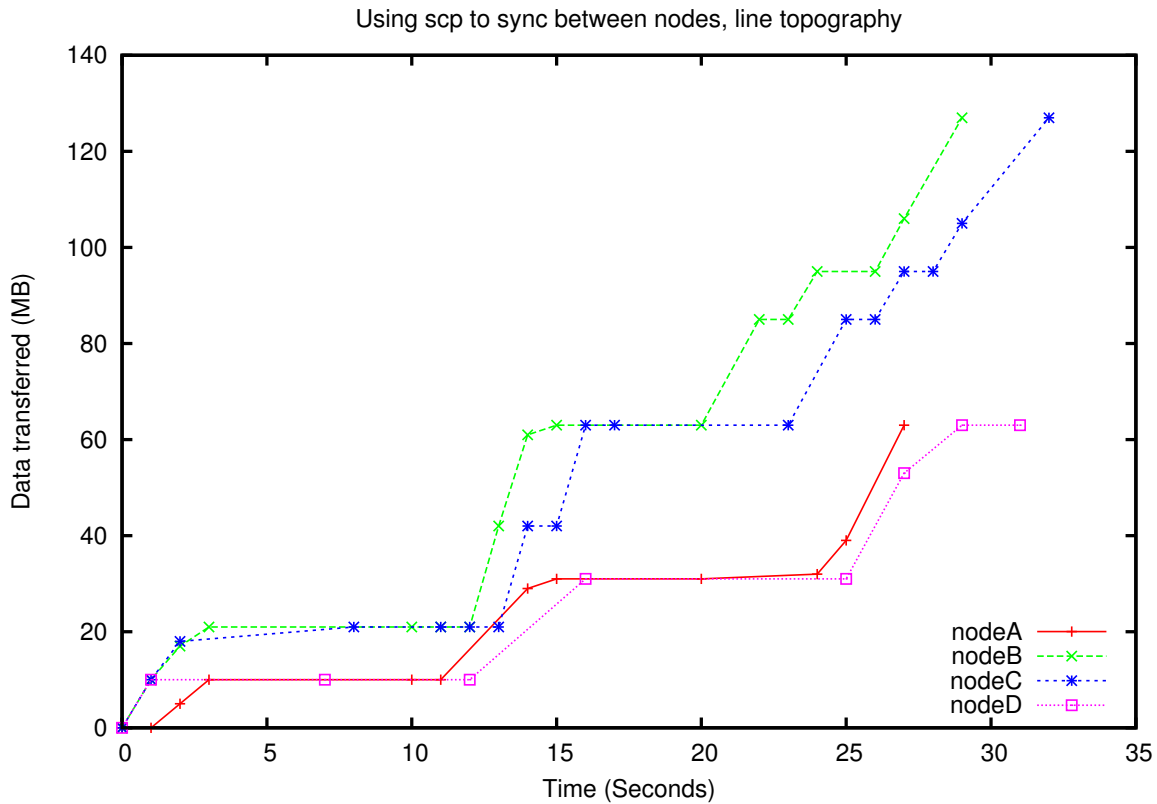


Figure 6: Line topology, using SCP, nodes copying data back and forth

Figure 6 shows three 10MB files being copied to nodeA 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 still had to make checks to see whether there were any changes or not.

I used a control file to get around this problem. Each time a node synchronised with another node it would write out a control file telling the other node what files had

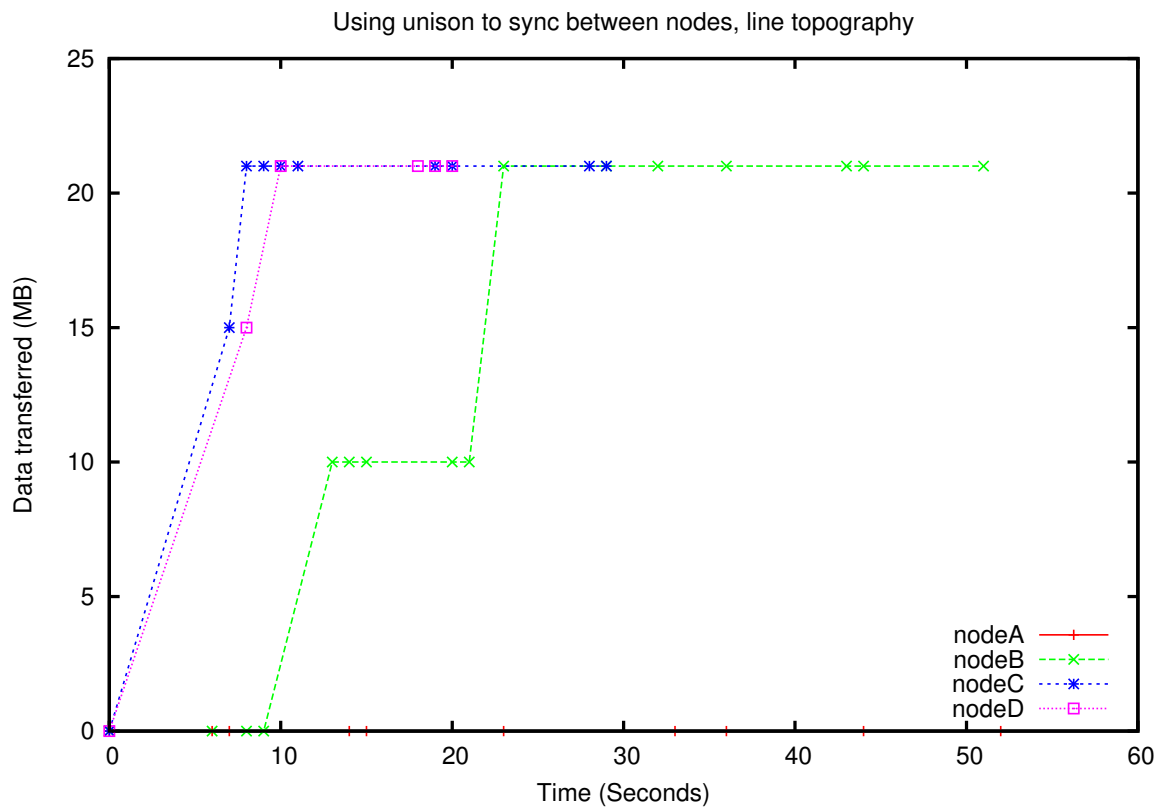


Figure 7: Line topology, using Unison, program continues to check for differences in files even after all nodes are the same. Each point represents an attempted sync.



been copied, who sent them and what the modification time of the files were. In this way a node could check if it was about to synchronise a file back to the node it had just received the file from or if local changes really had occurred to that file that were newer than a received file it should 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.

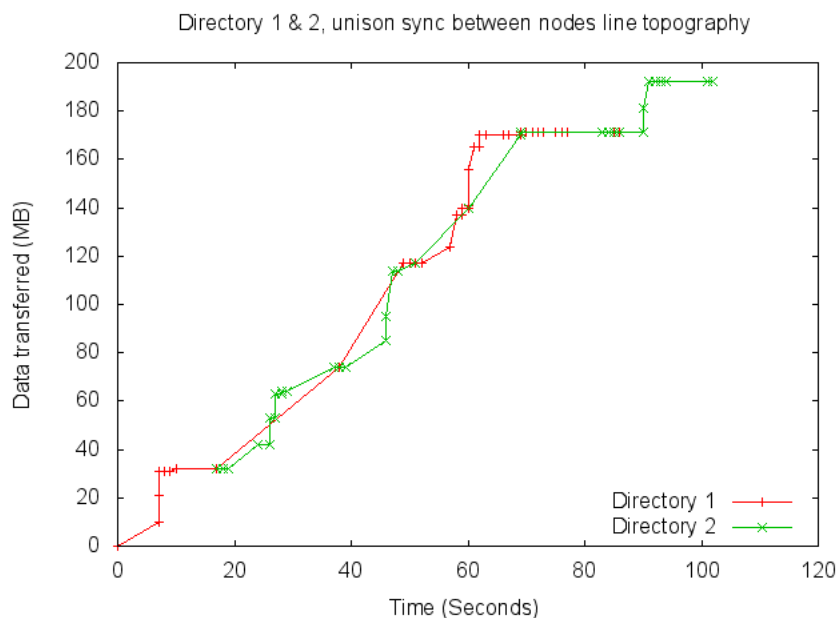


Figure 8: Line topology, using Unison, two different directories being synced

## 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.

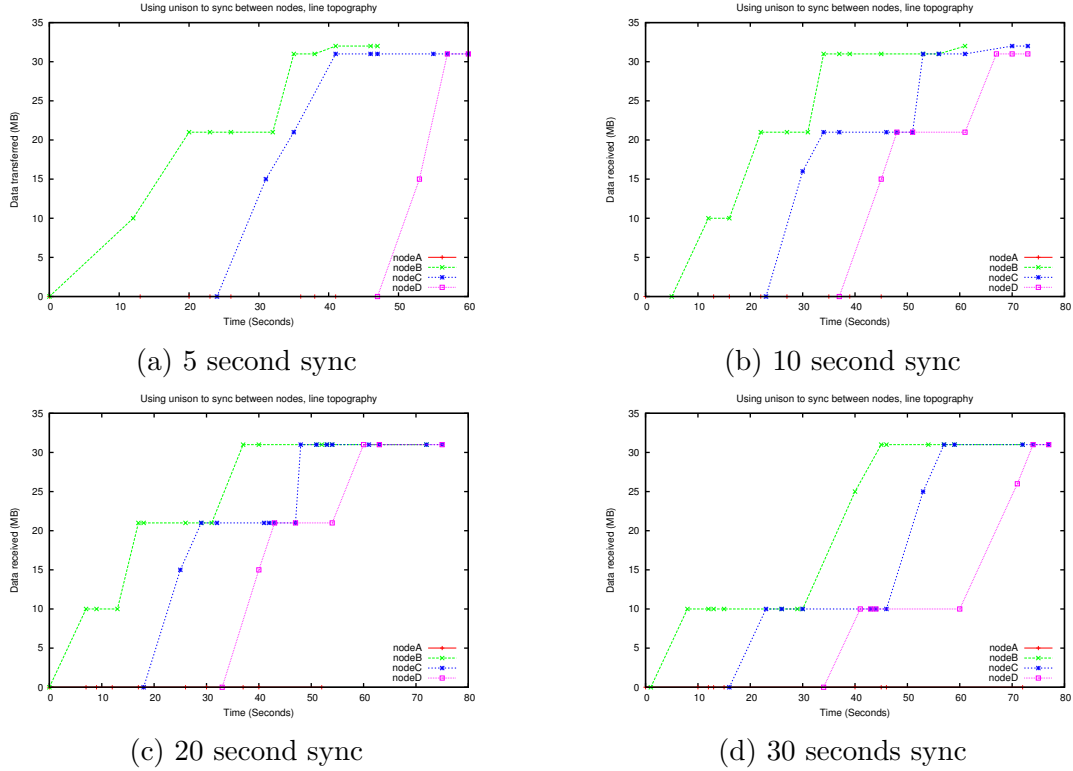


Figure 9: Comparison how frequently to sync

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...?

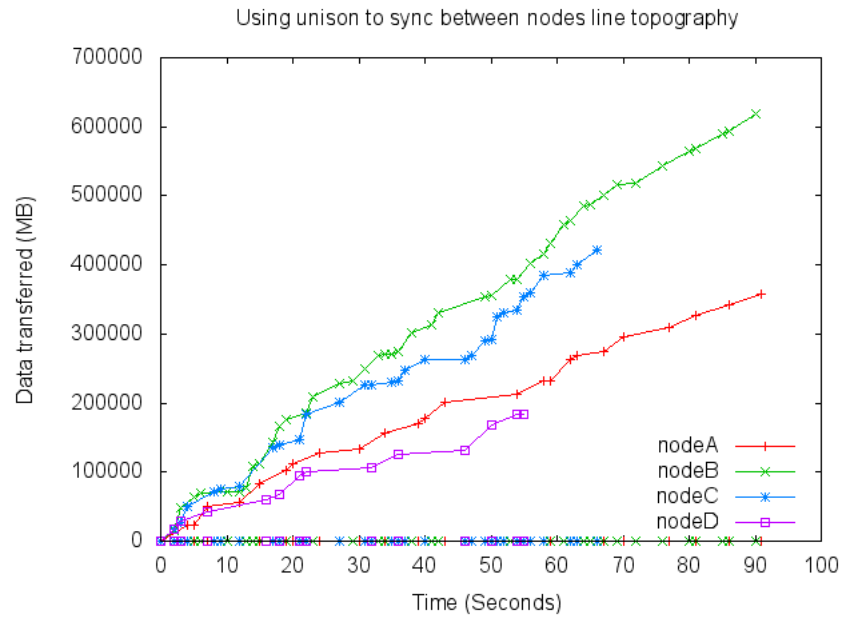


Figure 10: 2 seconds sleep text file

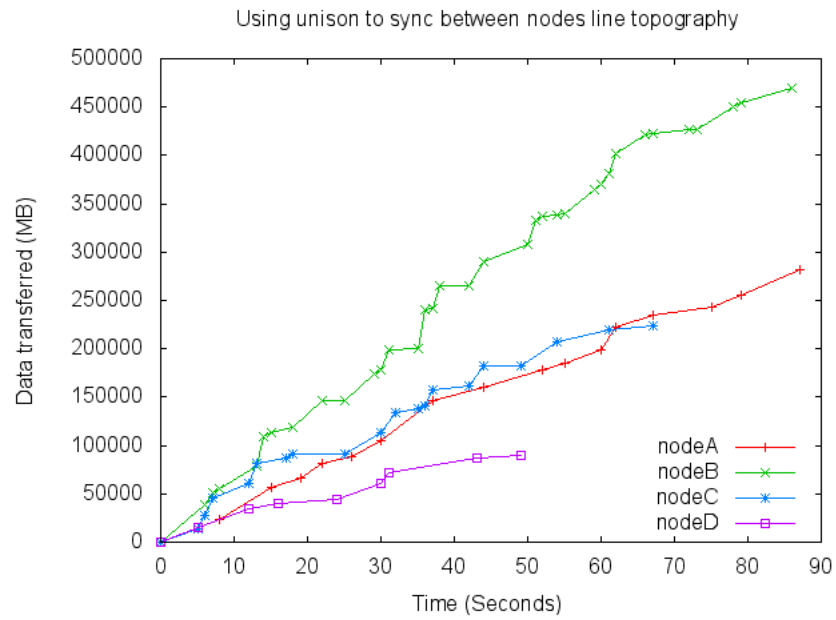


Figure 11: 5 seconds sleep text file

## 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.

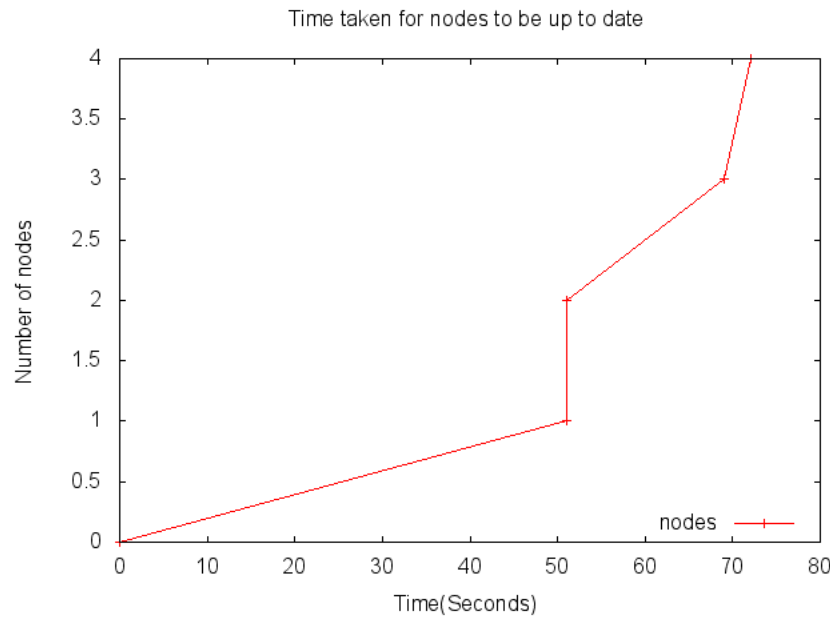


Figure 12: Unison, line, finishing times

## 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 time.ctime(0)
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\n"
87         else:
88             print "local_" + str(path) + "_modtime\n" + 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] + " "
106                      + stopIPs[k][1] + "\n")
107             #f.close()
108             #stopIPs.clear()
109             return True
110
111         f.close()
112         #stopIPs.clear()
113         if tmpcount == 0:
114             break
115
116     return False

```

```

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     def setLastSync(self):
156         #print "echo \" " + str(datetime.datetime.now()) + "\" >
157         " + homedir + "lastSync"
158         #subprocess.call(["echo", str(datetime.datetime.now())
159         + " > " + homedir + "lastSync"])

```



```

141         f = open(homepath + "lastSync", "w")
142         f.write(str(datetime.datetime.now()))
143         f.close()
144
145     def getLastSync(self):
146         f = open(homepath + "lastSync", "r")
147         time = f.read()
148         f.close()
149         return time.rstrip()
150
151     def newerThanLast(self, fileName):
152         stop = False
153         print "Last_sync_time:_" + str(self.getLastSync())
154         FMT = '%Y-%m-%d_%H:%M:%S.%f'
155         #datetime.datetime.strptime(dtstamp, FMT)
156         ts2 = time.strptime(self.getLastSync(), FMT)
157         modTime = self.getModTime(fileName)
158         print "local_" + str(fileName) + "_modtime:_" +
            modTime
159         ts1 = time.strptime(modTime, "%a_%b_%d_%H:%M:%S_%Y")
160         print "local_<=_stop:_" + str(ts1 <= ts2)
161         if ts1 > ts2:
162             print "Newer_file_than_last_sync!"
163             stop = True
164         return stop
165
166
167     #Get node name from whoami file
168     def getNodeName(self):
169         w = open(homepath + "whoami", "r")
170         nodename = w.read()
171         nodename = nodename[0].upper()
172         w.close()
173         return nodename
174
175     #Deprecated stop file
176     def setStopFile(self, ip, myIP, path):
177         subprocess.call(["ssh", ip, "echo_" + myIP + "_" + path
            + ">" + homepath + "stop"])
178         print "ssh", ip, "echo_" + myIP + ">" + homepath + "
            stop"
179
180     def rmTree(self, path):

```

```

181         subprocess.call(["ssh",ip,"rm-r_" + path + ""'])
182         print "ssh",ip,"rm-r_" + path + ""'
183
184     #Exclude files matching patterns in the ignore file
185     def exclusions(self, path):
186         try:
187             f = open("./ignore",'r')
188             for line in f:
189                 if line.rstrip() in path:
190                     #print "Ignoring: " + path
191                     return True
192             f.close()
193         except error, e:
194             print e
195         return False
196
197     def initFileSync(self, event):
198         if self.exclusions(event.pathname):
199             #print "Excluded returning"
200             return
201         pathparts = event.pathname.split("/")
202         foldName = "/" .join(pathparts[0:len(pathparts)-1])
203
204         print "Removing_watch_on:_" + foldName
205         wm.rm_watch(wm.get_wd(foldName), rec=True)
206
207         self.setLastSync()
208         self.fileSync(event.pathname)
209
210         print "Putting_watch_back_on:_" + foldName
211         wm.add_watch(foldName.rstrip(),pyinotify.ALL_EVENTS,
212                     rec=True, auto_add=True)
213
214         for i in range(0, len(watchedfolders[foldName]),4):
215             # ip = watchedfolders[foldName][i]
216             # myIP = readnet.getMyIP(ip)
217             for f in glob.glob(foldName + "/*"):
218                 if self.newerThanLast(f):
219                     print "init:_CONTINUE"
220                     self.setLastSync()
221                     self.fileSync(f)
222                 else:
223                     print "init:_STOP"

```

```

223
224 #Sync the files
225 def fileSync(self,pathname):
226     t = Tools()
227     if os.path.isdir(pathname):
228         print "Watching:_" ,pathname
229     for folder in watchedfolders.keys():
230         print "For_each_folder:_" + str(folder) + "_in_"
            watchedfolder_keys"
231     if folder in pathname:
232         for i in range(0, len(watchedfolders[folder])
            ,4):
233             ip = watchedfolders[folder][i]
234             path = watchedfolders[folder][i+1]
235             waitTime = watchedfolders[folder][i+2]
236             lastTime = watchedfolders[folder][i+3]
237             print "Wait:_" + str(waitTime) + "_Last:_"
                + str(lastTime)
238             print "Current_ip_and_path:_" + ip + "_" +
                path
239             readnet.logIPtraffic(ip, pathname)
240             myIP = readnet.getMyIP(ip)
241             subprocess.call(["ssh",ip,"/usr/bin/python
                _" + homopath + "readnet.py_i_" + myIP
                + "_f_" + pathname])
242             print "ssh",ip,"'/usr/bin/python_" +
                homopath + "readnet.py_i_" + myIP + "_-
                f_" + pathname + ""
243             fparts = folder.split("/")
244             fname = fparts[len(fparts)-1]
245             #stopIP = self.getStopInfo()
246             #print "STOP: " + stopIP[0] + " " + stopIP
                [1]
247             #if stopIP[0] == ip and stopIP[1] ==
                pathname:
248             if self.inStopFile(ip,pathname):
249                 print "STOPPED_to_" + ip + "_" + path
250                 #os.remove("./stop");
251             else:
252                 print "CONTINUE"
253                 t.timeElapsed(lastTime, waitTime)
254                 watchedfolders[folder][i+3] = str(
                    datetime.datetime.now())

```

```

255 t.updateFolderInfo(watchedfolders)
256 self.beginCopy(ip)
257 self.beginCopy(myIP)
258 if args.scp:
259     #print "SCP: For cpFile in " +
        folder
260     for cpFile in glob.glob(folder + "
        /*"):
261         #print "SCP GLOB:" + cpFile
262         print "scp","-rp",cpFile,ip +
            ":" + cpFile + ".tmp"
263         subprocess.call(["scp","-rp",
            cpFile,ip + ":" + cpFile + "
            .tmp"])
264         #subprocess.call(["ssh",ip,"
            yes y | find /tmp/" + fname
            + "-type f -exec cp -p {} "
            + path + fname + "/ \; rm /
            tmp/" + fname])
265         print "ssh",ip,"mv_" + cpFile
            + ".tmp_" + cpFile
266         subprocess.call(["ssh",ip,"mv_"
            + cpFile + ".tmp_" +
            cpFile])
267         print "END_SCP_GLOB"
268 elif args.rsync:
269     print "rsync","-rt",folder,ip + ":"
        + path
270     subprocess.call(["rsync","-rt",
        folder,ip + ":" + path])
271 else:
272     time.sleep(5)
273     print "unison","-batch","-
        confirmbigdel=false","-times",
        folder,"ssh://" + ip + "/" +
        path + fname
274     subprocess.call(["unison","-batch"
        ,"-confirmbigdel=false","-times"
        ,folder,"ssh://" + ip + "/" +
        path + fname])
275     print "Set_stop_files_uniq:" +
        pathname
276     #Set stop file on foreign host

```

```

277         self.setStopFileUniq(ip, myIP, pathname,
278                               folder)
279         #Set stop file for myself to look at
280         self.setStopFileUniq(myIP, myIP,
281                               pathname, folder)
282         self.endCopy(ip)
283         self.endCopy(myIP)
284         subprocess.call(["ssh", ip, "/usr/bin/python
285                          _" + homedir + "readnet.py _i_" + myIP
286                          + " _f_" + pathname])
287         readnet.logIPtraffic(ip, pathname)
288
289 #Sync files - DEPRECATED
290 def oldfileSync(self, event):
291     t = Tools()
292     if os.path.isdir(event.pathname):
293         print "Watching: _", event.pathname
294     for folder in watchedfolders.keys():
295         print "For_each_folder: _" + str(folder) + " _in _
296               watchedfolder_keys"
297         if folder in event.pathname:
298             for i in range(0, len(watchedfolders[folder])
299                             ,4):
300                 ip = watchedfolders[folder][i]
301                 path = watchedfolders[folder][i+1]
302                 waitTime = watchedfolders[folder][i+2]
303                 lastTime = watchedfolders[folder][i+3]
304                 print "Wait: _" + str(waitTime) + " _Last: _"
305                       + str(lastTime)
306                 print "Current_ip_and_path: _" + ip + " _" +
307                       path
308                 readnet.logIPtraffic(ip, event.pathname)
309                 myIP = readnet.getMyIP(ip)
310                 subprocess.call(["ssh", ip, "/usr/bin/python
311                                 _" + homedir + "readnet.py _i_" + myIP
312                                 + " _f_" + event.pathname])
313                 print "ssh", ip, "'/usr/bin/python_" +
314                       homedir + "readnet.py _i_" + myIP + " _" +
315                       f_" + event.pathname + " '"
316                 fparts = folder.split("/")
317                 fname = fparts[len(fparts)-1]
318                 #stopIP = self.getStopInfo()

```

```

307         #print "STOP: " + stopIP[0] + " " + stopIP
308         #if stopIP[0] == ip and stopIP[1] == event
309         #.pathname:
310         if self.inStopFile(ip,event.pathname):
311             print "STOPPED_to_" + ip + "_" + path
312             #os.remove("./stop");
313         else:
314             print "CONTINUE"
315             t.timeElapsed(lastTime, waitTime)
316             watchedfolders[folder][i+3] = str(
317                 datetime.datetime.now())
318             t.updateFolderInfo(watchedfolders)
319             self.beginCopy(ip)
320             if args.scp:
321                 #print "SCP: For cpFile in " +
322                 folder
323                 for cpFile in glob.glob(folder + "
324                     /*"):
325                     #print "SCP GLOB:" + cpFile
326                     print "scp","-rp",cpFile,ip +
327                         ":" + cpFile + ".tmp"
328                     subprocess.call(["scp","-rp",
329                         cpFile,ip + ":" + cpFile + "
330                         .tmp"])
331                     #subprocess.call(["ssh",ip,"
332                         yes y | find /tmp/" + fname
333                         + " -type f -exec cp -p {} "
334                         + path + fname + "/ \; rm /
335                         tmp/" + fname])
336                     print "ssh",ip,"mv_" + cpFile
337                         + ".tmp_" + cpFile
338                     subprocess.call(["ssh",ip,"mv_"
339                         + cpFile + ".tmp_" +
340                         cpFile])
341                     print "END_SCP_GLOB"
342             elif args.rsync:
343                 print "rsync","-rt",folder,ip + ":"
344                     + path
345                 subprocess.call(["rsync","-rt",
346                     folder,ip + ":" + path])
347             else:
348                 time.sleep(5)

```

```

333         print "unison","-batch","-
            confirmbigdel=false","-times",
            folder,"ssh://" + ip + "/" +
            path + fname
334         subprocess.call(["unison","-batch"
            ,"-confirmbigdel=false","-times"
            ,folder,"ssh://" + ip + "/" +
            path + fname])
335     print "Set_stop_files_uniq:" + event.
        pathname
336     #Set stop file on foreign host
337     self.setStopFileUniq(ip,myIP,event.
        pathname,folder)
338     #Set stop file for myself to look at
339     self.setStopFileUniq(myIP,myIP,event.
        pathname,folder)
340     self.endCopy(ip)
341     subprocess.call(["ssh",ip,"/usr/bin/python
        _" + homopath + "readnet.py_i_" + myIP
        + "_f_" + event.pathname])
342     readnet.logIPtraffic(ip, event.pathname)
343
344     #def process_IN_CREATE(self, event):
345     #     print "Create:",event.pathname
346     def process_IN_DELETE(self, event):
347         print "Delete:" + event.pathname
348         #self.initFileSync(event)
349     def process_IN_CREATE(self, event):
350         print "CREATE:" + event.pathname
351         self.initFileSync(event)
352     def process_IN_MOVED_FROM(self, event):
353         print "Move_from:" + event.pathname
354         #     self.initFileSync(event)
355     def process_IN_MODIFY(self, event):
356         #print "Modify: ",event.pathname
357         self.initFileSync(event)
358     def process_IN_MOVED_TO(self, event):
359         print "Move_to:" + event.pathname
360         self.initFileSync(event)
361
362
363     def main():
364         t = Tools()

```

```

365     f = open( './folderstowatch', 'r' )
366
367     for folder in f:
368         if( folder[0] == '#' ):
369             pass
370         else:
371             info = folder.split()
372             wm.add_watch( info[0].rstrip(), pyinotify.ALL_EVENTS
373                 , rec=True, auto_add=True)
374             print "Watching:_" , info[0].rstrip()
375             if info[0] not in watchedfolders.keys():
376                 watchedfolders[info[0].rstrip()] = []
377                 watchedfolders[info[0].rstrip()].append( info[1] )
378                 watchedfolders[info[0].rstrip()].append( info[2] )
379                 watchedfolders[info[0].rstrip()].append( info[3] )
380                 watchedfolders[info[0].rstrip()].append( str(
381                     datetime.datetime.now() ) )
382
383     f.close()
384
385     try:
386         f = open( './folders.dat', 'r' )
387         for folder in f:
388             if( folder[0] == '#' ):
389                 pass
390             else:
391                 info = folder.split()
392                 if info[0] in watchedfolders.keys():
393                     del watchedfolders[info[0].rstrip()]
394                     #wm.add_watch( info[0].rstrip(), pyinotify.
395                         ALL_EVENTS, rec=True, auto_add=True)
396                     #print "Watching: ", info[0].rstrip()
397                     if info[0] not in watchedfolders.keys():
398                         watchedfolders[info[0].rstrip()] = []
399                         watchedfolders[info[0].rstrip()].append(
400                             info[1] )
401                         watchedfolders[info[0].rstrip()].append(
402                             info[2] )
403                         watchedfolders[info[0].rstrip()].append(
404                             info[3] )
405                         watchedfolders[info[0].rstrip()].append(
406                             str( datetime.datetime.now() ) )
407                     else:
408                         print "Removing:_" + info[0]

```



```

401         f.close()
402     except IOError, e:
403         print "Folders.dat does not exist , skipping"
404
405     t.updateFolderInfo(watchedfolders)
406
407     #print watchedfolders
408     eh = MyEventHandler()
409
410     notifier = pyinotify.Notifier(wm, eh)
411     notifier.loop()
412
413 if __name__ == '__main__':
414     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 ,
28         shell=True)
29     words = ifconf.split()
30     now = False
31     for word in words:
32         if word == "inet":
33             now = True
34         elif now:
35             word = word.split(":")
36             #print word[1]
37             return word[1]
38 #Log interface coresponding to ipaddr
39 def logIPtraffic(ipaddr , folder):
40     route = subprocess.check_output("ip_route_get_" + ipaddr ,
41         shell=True)
42     words = route.split()
43     interface = ""
44     for word in words:
45         if word.startswith("eth"):
46             interface = word
47             #print interface
48             break
49     writeIface(interface , folder)
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

```

```

105     for split in splitput:
106         if split in interfacenames:
107             interface = True
108             interfacename = split
109             #print interfacename
110         if(nex != ""):
111             sp = split.split(":")
112             if(sp[0] == "bytes"):
113                 if(nex == "RX"):
114                     download = int(sp[1])
115                 else:
116                     upload = int(sp[1])
117             nex = ""
118             count += 1
119             if(count == 2):
120                 interface = False
121                 f = open(homepath + "log/" \
122                     + str(socket.gethostname()) + "-" \
123                     + interfacename + ".log", 'a')
124                 f.write(str(datetime.datetime.now()) + "_"
125                     + interfacename + "_download:" + str(
126                         download) + "_upload:" + str(upload) +
127                         "\n")
128                 f.close()
129                 count = 0
130             elif(interface):
131                 if(split == "RX" or split == "TX"):
132                     nex = split
133
134 if __name__ == "__main__":
135     args = parser.parse_args()
136     if args.ip != None:
137         logIPtraffic(args.ip, args.fold)
138         #getMyIP(args.ip)
139     else:
140         pass
141     main()

```

## C onTheFly.sh

```

1 vm_name_arr=("Ubuntu-Pool" "Ubuntu-Silence" "Ubuntu-Black" "
    Ubuntu-Spheros" "Ubuntu-Wild")

```

```

2  vm_addr_arr=("192.168.0.28" "192.168.0.27" "192.168.0.30" "
    192.168.0.14")
3  #Wild = 19
4  intnetarr=("lion" "tiger" "cat" "dog" "fish" "kiwi" "swish" "
    boom" "roar")
5  #These should all be in one big dictionary apart from inet
    names
6  letterarr=("a" "b" "c" "d" "e" "f" "g" "h" "i" "j")
7  ifcountarr=(2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2)
8  ethcountarr=(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
9  incount=1
10 bigncount=2
11 littlencount=1
12 folderpath="/home/cal/Documents/t18"
13 folderpath2="/home/cal/Documents/t02"
14 hompath="/home/cal/Documents/Private-Sync/"
15 waitTime=5
16
17 function clear_ifaces() {
18     i=0
19     while [ "$i" -lt "${#vm_name_arr[@]}" ]; do
20         VBoxManage modifyvm ${vm_name_arr[$i]} --nic2 none
21         echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic2 _
            none"
22         VBoxManage modifyvm ${vm_name_arr[$i]} --nic3 none
23         echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic3 _
            none"
24         VBoxManage modifyvm ${vm_name_arr[$i]} --nic4 none
25         echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic4 _
            none"
26         let "i++"
27     done
28 }
29
30 function clear_watched_folders() {
31     i=0
32     while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
33         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"
34         let "i++"

```

```

35     done
36 }
37
38 function git_pull() {
39     i=0
40     while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
41         echo "ssh_cal@${vm_addr_arr[$i]} \"cd /home/cal/
42             Documents/Private-Sync; git pull origin master\""
43         ssh_cal@${vm_addr_arr[$i]} "cd /home/cal/Documents/
44             Private-Sync; git pull origin master"
45         let "i++"
46     done
47 }
48
49 function search_letters() {
50     index=0
51     while [ "$index" -lt "${#letterarr[@]}" ]; do
52         if [ "${letterarr[$index]}" = "$1" ]; then
53             echo $index
54             return
55         fi
56         let "index++"
57     done
58     echo "None"
59 }
60
61 function vbmMOD {
62     echo "VBoxManage modifyvm $1 --nic$3 intnet"
63     VBoxManage modifyvm $1 --nic$3 intnet
64     echo "VBoxManage modifyvm $1 --intnet$3 $2"
65     VBoxManage modifyvm $1 --intnet$3 $2
66 }
67
68 function gatherLogs {
69     index=0
70     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
71         echo "scp_cal@${vm_addr_arr[$index]}:/home/cal/
72             Documents/Private-Sync/log/*../logs/"
73         scp_cal@${vm_addr_arr[$index]}:/home/cal/Documents/
74             Private-Sync/log/*../logs/
75         let "index++"
76     done
77 }

```

```

74
75 function clean {
76     index=0
77     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
78         echo "ssh_cal@${vm_addr_arr[$index]} \\"rm ${homepath}
            log/*; rm ${homepath}Stop-*; rm ${homepath}folders.
            dat\\"
79         ssh_cal@${vm_addr_arr[$index]} "rm_${homepath}log/*; \
            rm_${homepath}Stop-*; \rm_${homepath}folders.dat"
80         let "index++"
81     done
82 }
83
84 function cleanFold {
85     index=0
86     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
87         echo "ssh_cal@${vm_addr_arr[$index]} \\"rm -rf ${
            folderpath}/*;\\"
88         ssh_cal@${vm_addr_arr[$index]} "rm -rf_${folderpath}
            }/*;"
89         let "index++"
90     done
91 }
92
93 function sendKeys {
94     index=0
95     while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
96         #ssh_cal@${vm_addr_arr[$index]} "rm /home/cal/.ssh/
            authorized_keys"
97         for file in /Users/calum/.ssh/*.pub; do
98             #echo "$file"
99             echo "cat_${file} | \ssh_cal@${vm_addr_arr[$index]} \
                "cat >> /home/cal/.ssh/authorized_keys\\"
100            cat $file | ssh_cal@${vm_addr_arr[$index]} "cat \>>
                \home/cal/.ssh/authorized_keys"
101        done
102        let "index++"
103    done
104    #for file in /Users/calum/.ssh/*.pub; do
105    #     echo "$file"
106    #     cat $file | ssh_cal@192.168.0.17 "cat >> /home/cal/.
        ssh/testfile"

```

```

107      #      echo "cat $file | ssh cal@192.168.0.17 \"cat >> /home
           /cal/.ssh/testfile\""
108      #done
109  }
110
111  function ifconf {
112      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`"
113      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
114  }
115
116  function ifconf2 {
117      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"
118      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
119  }
120
121  if [ $2 == "vm" ]; then
122      clear_ifaces
123
124      #
125      while read line
126      do
127          first=$(echo "$line" | awk '{print $1}')
128          last=$(echo "$line" | awk '{print $(NF)}' | sed 's
              /[:]//g')
129          #echo "$first and $last"
130          index=$(search_letters $first)
131          if [ "$index" = "None" ]; then

```



```

132         #echo "None"
133         :
134     else
135         vmMOD ${vm_name_arr[$index]} ${intnetarr[$incount
136             ]} ${ifcountarr[$index]}
137         #echo "in: $index"
138         (( ifcountarr[$index]++ ))
139         index=$(search_letters $last)
140         vmMOD ${vm_name_arr[$index]} ${intnetarr[$incount
141             ]} ${ifcountarr[$index]}
142         #echo "in: $index"
143         (( ifcountarr[$index]++ ))
144         incount=$((incount+1))
145     fi
146 done <graphs/$1
147 elif [ $2 == "if" ]; then
148     clear_watched_folders
149     while read line
150     do
151         first=$(echo "$line" | awk '{print $1}')
152         last=$(echo "$line" | awk '{print $(NF)}' | sed 's/
153             /[:]/g')
154         echo "$first_and_$last"
155         index=$(search_letters $first)
156         if [ "$index" = "None" ]; then
157             #echo "None"
158             :
159         else
160             ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index
161                 ]} $bigncount $littlencount $(( $littlencount+1
162                 ))
163             #echo "in: $index"
164             (( ethcountarr[$index]++ ))
165             (( littlencount++ ))
166             index=$(search_letters $last)
167             ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index
168                 ]} $bigncount $littlencount $(( $littlencount-1
169                 ))
170             #echo "in: $index"
171             (( ethcountarr[$index]++ ))
172             incount=$((incount+1))
173             (( bigncount++ ))

```

```

168         (( littlencount — ))
169     fi
170 done <graphs/$1
171 elif [ $2 == "if2" ]; then
172     clear_watched_folders
173
174     while read line
175     do
176         first=$(echo "$line" | awk '{print $1}')
177         last=$(echo "$line" | awk '{print $(NF)}' | sed 's
            /[;]//g')
178         echo "$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[
                $index]} $bigcount $littlencount $((
                $littlencount+1 ))
185             #echo "in: $index"
186             (( ethcountarr[$index]++ ))
187             (( littlencount++ ))
188             index=$(search_letters $last)
189             ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[
                $index]} $bigcount $littlencount $((
                $littlencount-1 ))
190             #echo "in: $index"
191             (( ethcountarr[$index]++ ))
192             incount=$incount+1
193             (( bigcount++ ))
194             (( littlencount — ))
195         fi
196     done <graphs/$1
197 elif [ $2 == "key" ]; then
198     sendKeys
199 elif [ $2 == "gather" ]; then
200     gatherLogs
201 elif [ $2 == "clean" ]; then
202     clean
203 elif [ $2 == "pull" ]; then
204     git_pull
205 elif [ $2 == "clean-fold" ]; then

```

```

206     cleanFold
207 elif [ $2 == "help" ]; then
208     echo "vm-----setup vm networking"
209     echo "if-----setup network addresses etc for each
        vm"
210     echo "if2-----setup network addresses etc for each
        vm for two folders"
211     echo "gather-----gather the logs in"
212     echo "clean-----clean out the logs/config files"
213     echo "clean-fold---clean out the files folder"
214     echo "pull-----pull the latest code from the
        repository to each vm"
215     echo "help-----display this help message"
216 else
217     echo "Oops try again"
218 fi
219
220 neato -Teps graphs/$1 > graphs/$1-graph.eps

```