# Private Dropbox
# Final Report
# COSC480

Calum O'Hare
Supervisor: David Eyers

# 1 Abstract

I have written a program in python which reads user settings from a file. Synchronises the appropriate files to the appropriate machines when they have been modified. Using an efficient two way file synchronising tool called unison. I will discuss in this paper what I have done and how I have tested it.

# Contents

# 2 Introduction

## 2.1 Project goal

The aim of this project is 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:

- Be decentralised. It will not necessarily need to be run in "the cloud" there should be no centralised server, just many cooperating client nodes. However it should be possible to configure the system to be centralised 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. Although still synchronise between two Clients as this is the basis for multiple client synchronisation. Clients may be running different operating systems. Clients may run on different networks, with different costs of access, including being disconnected from the Internet at times.
- Allow for fine-grained user control for the majority of the program's functions, *e.g.*, how often, and what, to replicate within different sets of files. 'What' could be file name, file type, file size, *etc.*
- 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.
- Operate automatically, without the user having to initiate a file synchronisation themselves. The user should be able to set when and where they would like synchronisation to occur.

## 2.2 Background

There are already many services available that synchronize your files. Dropbox, Google Drive, Microsoft SkyDrive, Apple iCloud all offer cloud based solutions for automatically synchronizing your files. 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 also cannot guarantee that you will always be able to access your data, if the company who owns 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.com a file hosting service has recently been shut down by the United States Department of Justice for alleged copyright infringement. According to the founder, 100 million users lost access to 12 billion unique files[1].

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 but they create a bottleneck at the server. Cloud based solutions are also often centralised. Another problem is that even if they are decentralised like git, they won't automatically push updates to other working sets.

## Example use case

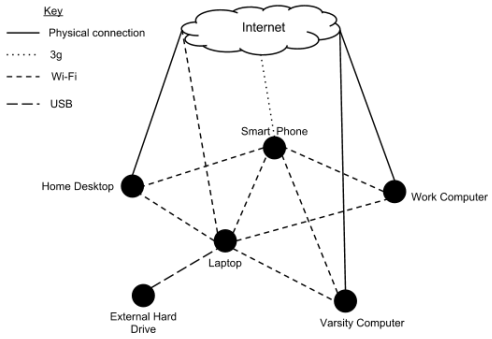
Here is how I would use such a tool as an example use case.

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 ≈5MB 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 loose a couple of DVDs. 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.

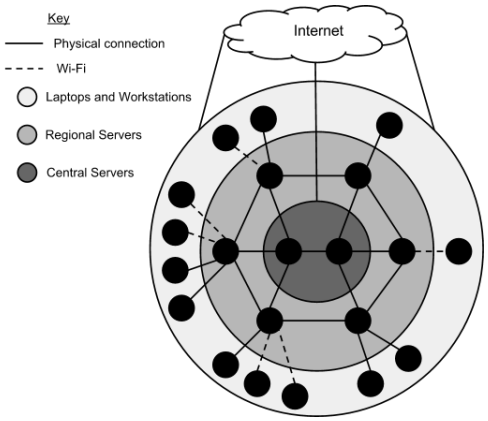
Some of the files that I move around are of a sensitive or personal nature and I would prefer not to store them with a third party vendor. I also have different synchronisation requirements for different types of data. For example my collection of large video files does not change that often and will 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.

An effective file synchronisation tool would be of great use to me personally. Dropbox does not do enough for me. It does not give me 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 Dropbox closes down or my internet connection dies.

The graph of a personal network has been described above, the graph of a corporate network 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 private dropbox




(a) Personal Network




(b) Corporate Network

will.

# 3 Virtual Machines, Node networks

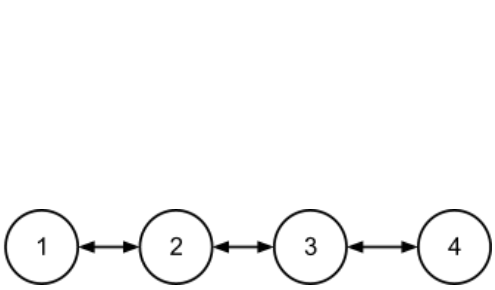
For testing my program I needed to have a network of computers that can be linked together in different arrangements easily. I decided to use virtual machines for this job since it means I do not need to have a large number of physical machines. I can create new machines very easily, and manipulate the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

I have used Oracle's VirtualBox software. I chose VirtualBox because of its easy to use command line interface. I have several scripts that call the `vbmoxmange` command to set up the internal network connections between machines and then start up the machine itself. This makes switching between network configurations very easy as I can just run a different script depending on which network topology I would like to test.

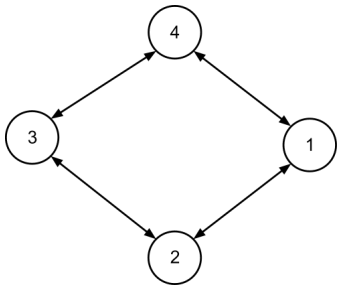
I have decided to start testing my program with some simple topologies to see if I can gain any insight into how best to replicate data around a network with many nodes. The next step will be to use those principles and start running more complicated networks to see how the program performs.

Snippet from one of my network scripts:

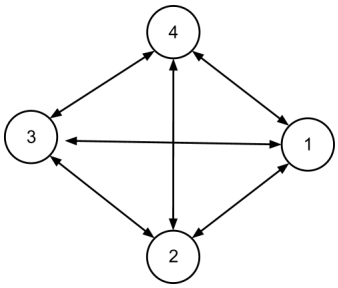
```
VBoxManage modifyvm "Ubuntu-Test" --nic2 intnet
VBoxManage modifyvm "Ubuntu-Test" --intnet2 "intnet"
VBoxManage startvm "Ubuntu-Test"
```



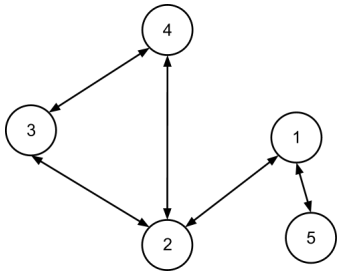

(c) Line topology




(d) Circle topology




(e) Connected mesh topology




(f) Other topology

Figure 1: Simple network topologies

# 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.

# 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.

# 6 Monitoring Directories

The application needs to monitor directories for changes 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 and my application is designed to be as efficient as possible. 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. Instead I have looked into ways of being notified of a change in the file system below the watched directory.

- Inotify
  - Inotify is a linux 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 which provided the same function. 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 uses SIGIO. Inotify only uses one file descriptor and returns events to the listener as they occur[2]. It works well and does what I need it to do. There is a Python module called pyinotify that provides a Python interface to inotify, which I have used and tested 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[3]. 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, as well.

    I also looked at using the `kqueue`[4] 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 the application's needs.
- ReadDirectoryChangesW
  - Windows, like the other operating systems I have looked up, provides a nice 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.

I have chosen only to implement my program on linux because portability wasn't in the main scope of the project. It would have been nice to look at it further but became too time consuming and not interesting from a research perspective.

# 7 Point-to-Point synchronisation

After some preliminary analysis of the available file synchronisation tools I have found a tool called Unison 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 machines (or the same machine).

I decided to run some tests using unison and the network I had set up to determine whether this would make a good base for my program or not.

I looked at three methods of file synchronisation across different networks. Naive 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.

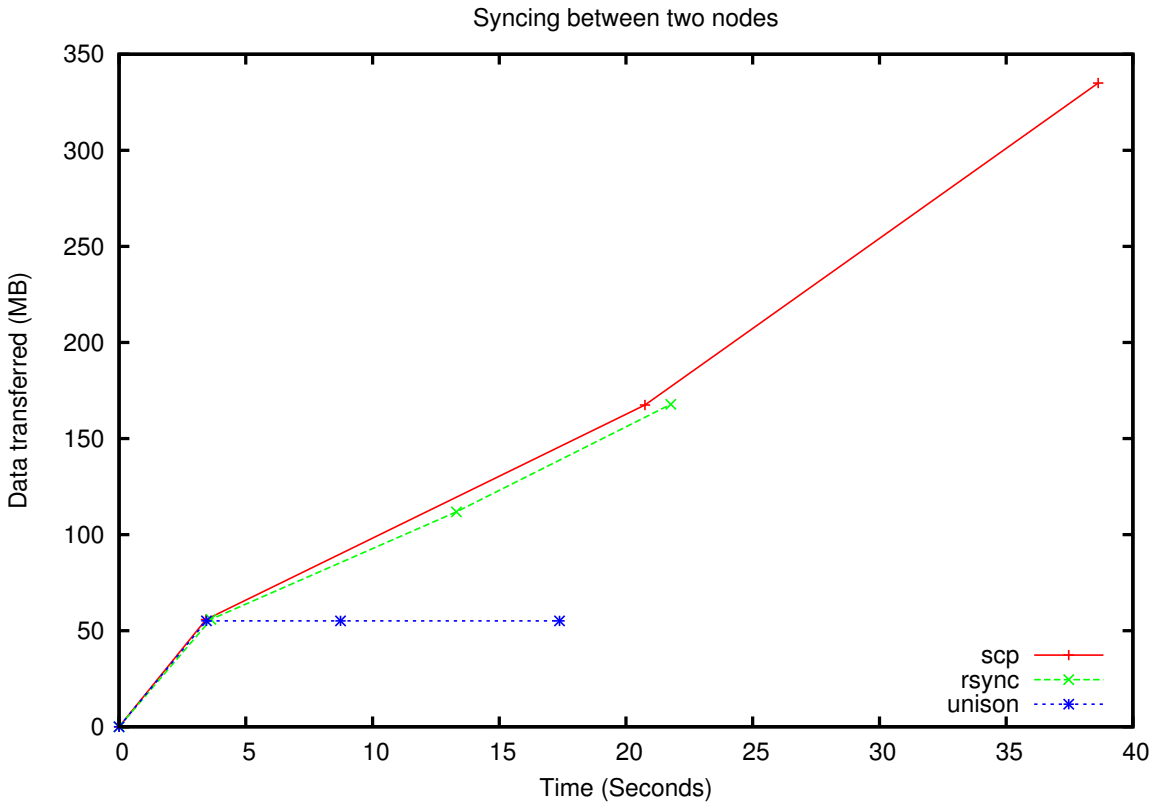



Figure 2: Comparison of SCP, Rsync, Unison, between two nodes

Rsync and unison performed significantly better than the naive copy method (as

expected). After the initial file transfer subsequent edits to the file meant much less data had to be transmitted over the network, which meant the node graph became up to date much more quickly.

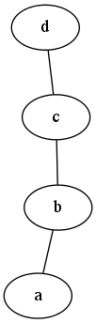
The reason naive copy sent over 300Mb of data to copy three 50Mb files was because my implementation is deliberately naive, it will copy the entire directory each time it is changed. For rsync and unison this is not a problem because they work based on the differences between the files. However copy 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.

50Mb + 100Mb + 150Mb = 300Mb

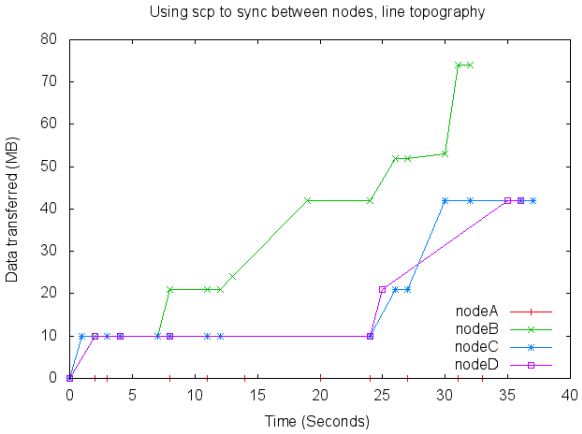
Rsync copies the expected 150Mb for three 50Mb files. While Figure 2 illustrates another advantage of unison over rysnc. The graph shows three zero filled binary files being copied from one node to another one after the other. Unison recognised that even though the files were named differently they were the same file. Another advantage of unison is that it handles replication in two directions without clobbering 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 ssh tunnel between the machines which all three methods used. Unison and rsync also require some overhead when checking 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.

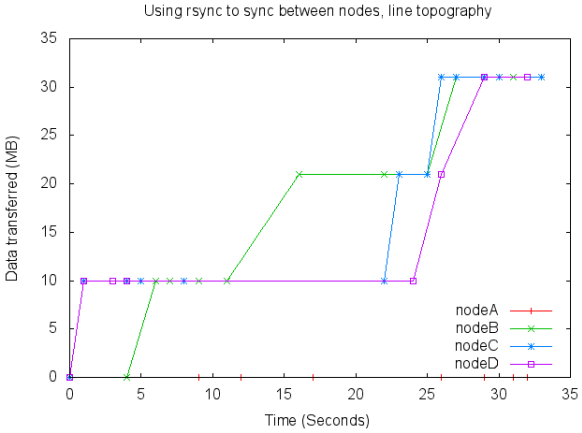
# 8 Full graph replication



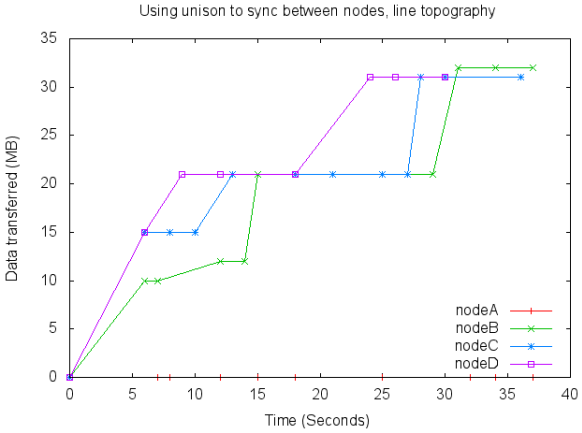

(a) Line - Generated Graph of Topology




(b) SCP




(c) Rsync




(d) Unison

Figure 3: Comparison of methods over line topology

# 9 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).

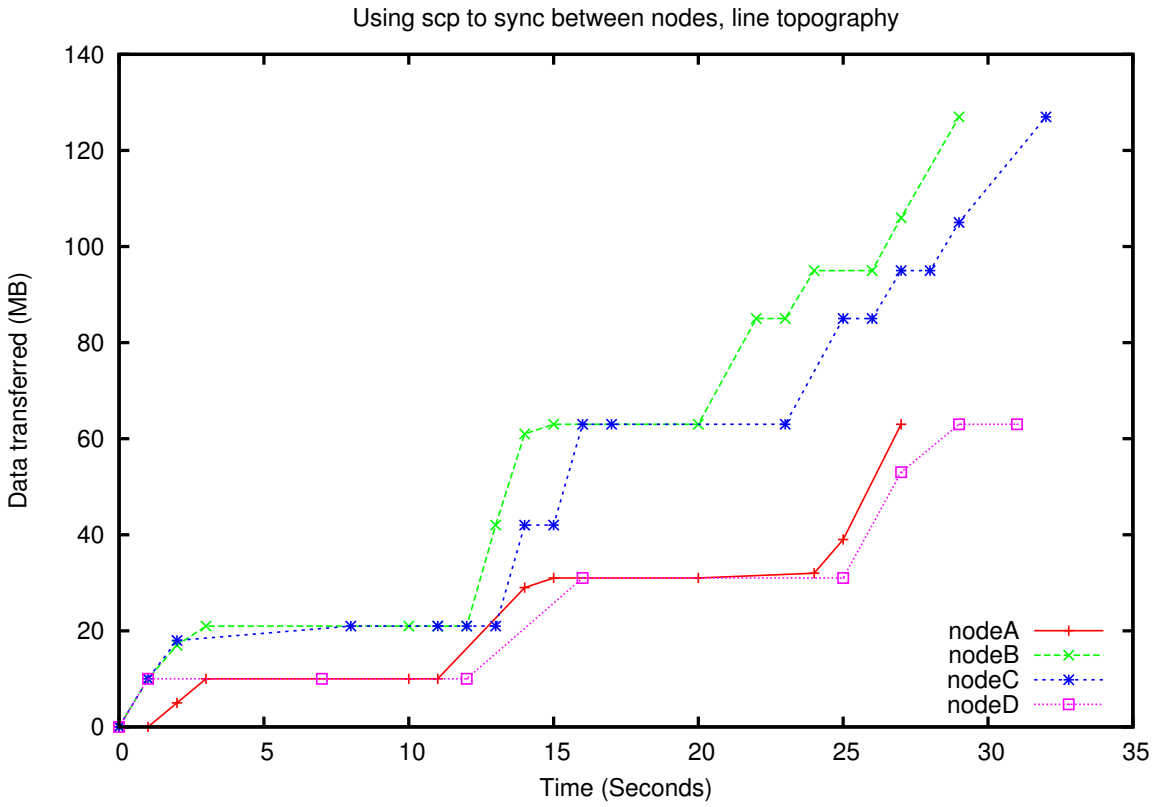



Figure 4: Line topology, using scp, nodes copying data back and forth

Figure 4 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 5 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

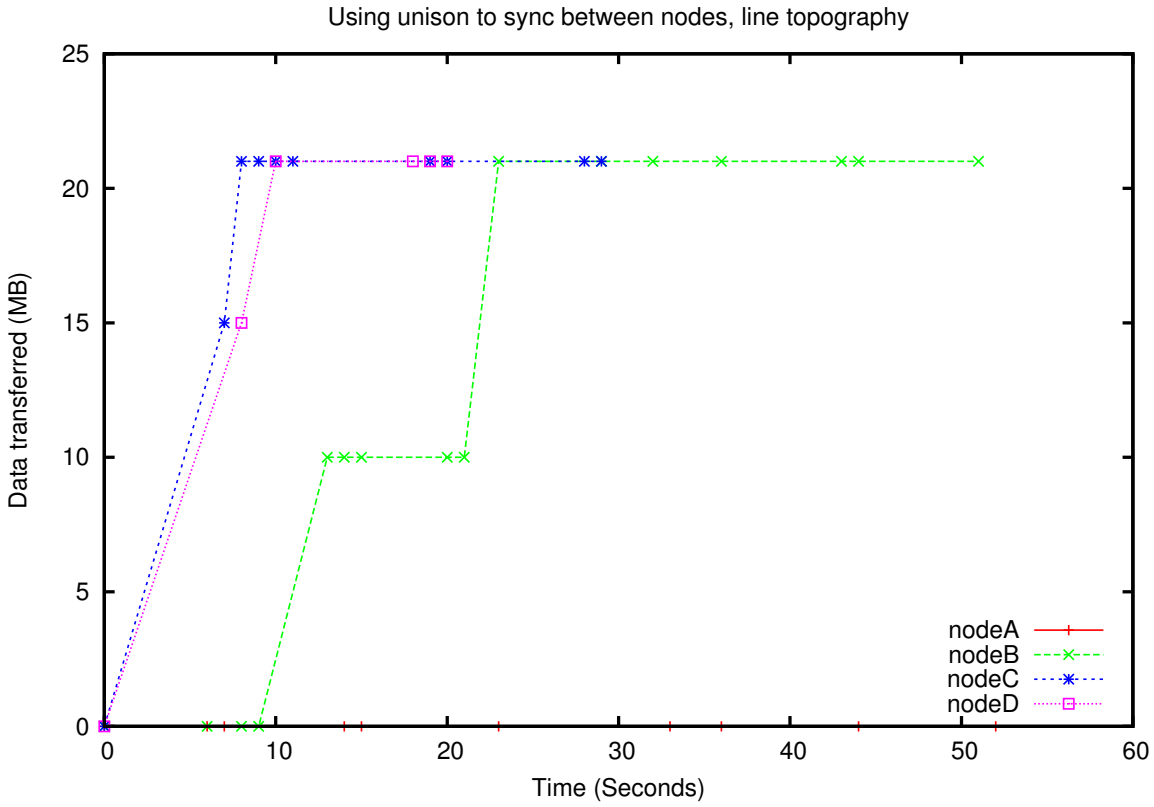



Figure 5: Line topology, using unison, unnecessary checking

files had 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 received the file from or if it had local changes that were newer than a received file it could continue with its sync.

# 10 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.

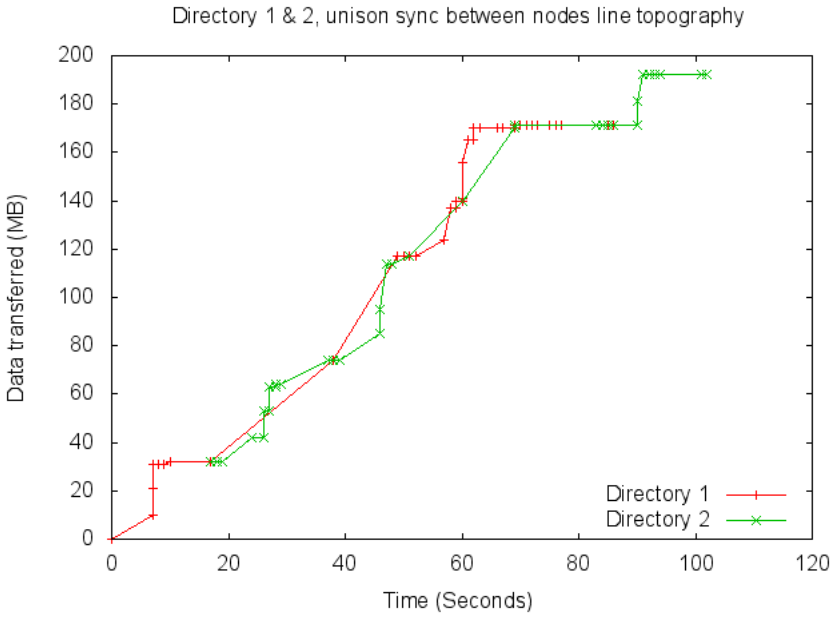



Figure 6: Line topology, using unison, two different directories being synced

# 11 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.

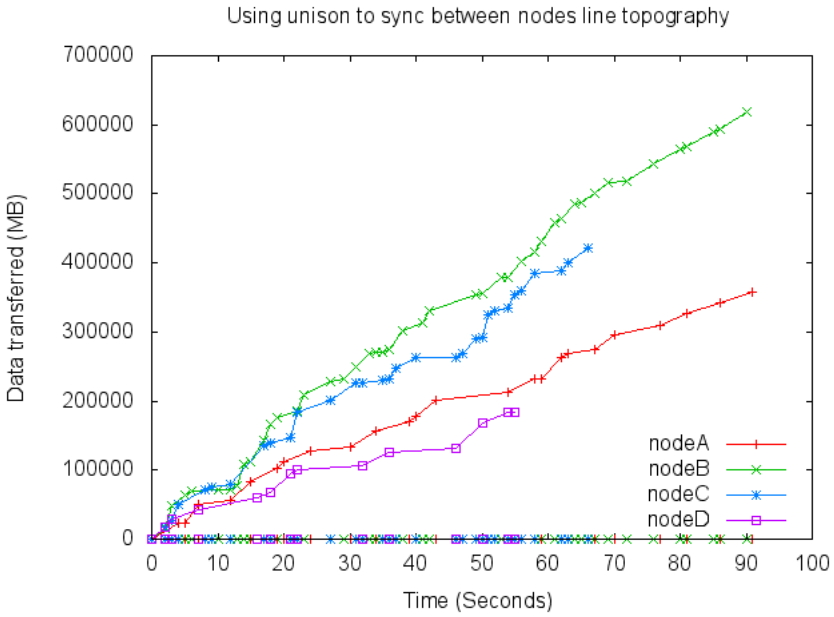



Figure 7: 2 seconds sleep text file

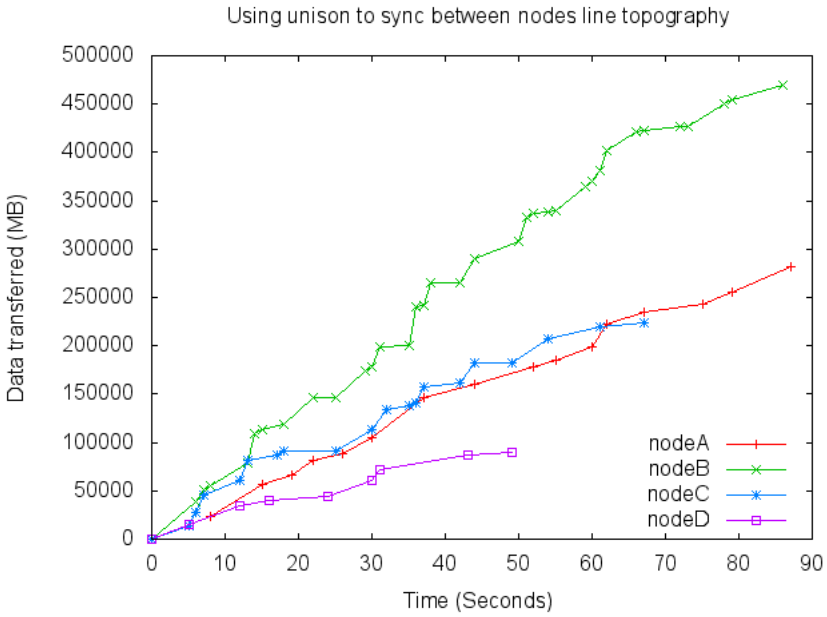



Figure 8: 5 seconds sleep text file

# 12 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.

# 13 Results

# 14 Conclusion

# References


[1] Foreman, Michael "Kim Dotcom v United States of America". Computerworld. 3 February 2012.

[2] www.kernel.org/pub/linux/kernel/people/rml/inotify/README, 22 September 2004.

[3] Apple Inc. `https://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/FSEvents_ProgGuide/Introduction/Introduction.html`, 11 October 2011.

[4] Apple Inc. `http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man2/kqueue.2.html`


# 15 Bibliography

# 16 Glossary

# 17 Index

# 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")
```

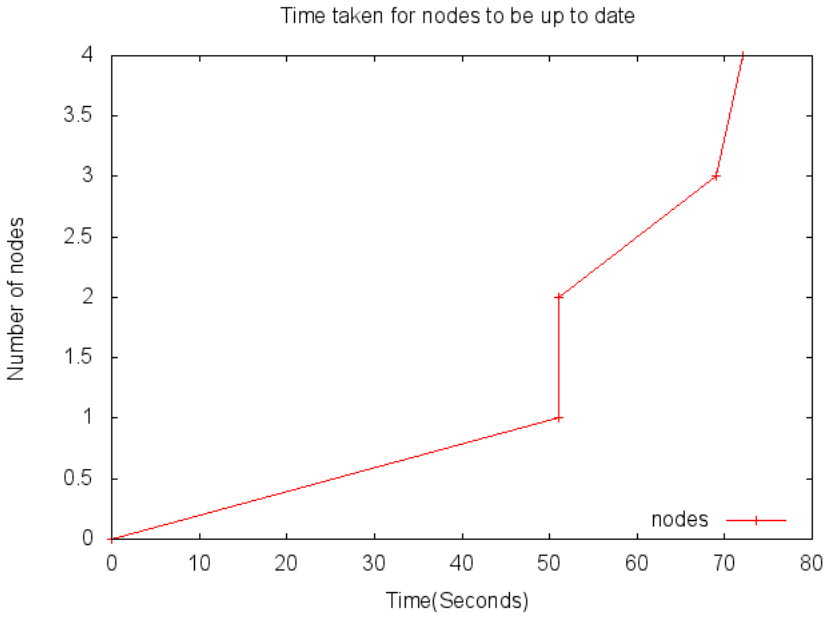



Figure 9: Unison, line, finishing times

```
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
        try and copy data here
118     def setStopFileUniq(self,ip,myIP,path,folder):
119         nodename = self.getNodeName()
120         #print "ssh",ip,"echo " + myIP + " " + path + " " +
            self.getModTime(path) + " >> " + homepath + "Stop-"
            + nodename + ".tmp;"
121         #subprocess.call(["ssh",ip,"echo " + myIP + " " + path
            + " " + self.getModTime(path) + " >> " + homepath +
            "Stop-" + nodename + ".tmp;"])
122         subprocess.call(["ssh",ip,"rm " + homepath + "Stop-" +
            nodename + ".tmp;"])
123         for cpFile in glob.glob(folder + "/*"):
124             subprocess.call(["ssh",ip,"echo " + myIP + " " +
                cpFile + " " + self.getModTime(cpFile) + " >> "
                + homepath + "Stop-" + nodename + ".tmp;"])
125
126     #Sets the config files on the remote node
127     def beginCopy(self, ip):
128         nodename = self.getNodeName()
129         print "ssh",ip,"touch " + homepath + "Stop-" +
            nodename + ".tmp; mv " + homepath + "Stop-" +
            nodename + " " + homepath + "Stop-" + nodename + ".
            tmp;"
130         subprocess.call(["ssh",ip,"touch " + homepath + "Stop-
            " + nodename + ".tmp; mv " + homepath + "Stop-" +
            nodename + " " + homepath + "Stop-" + nodename + ".
            tmp;"])
131
132     #Moves the Stop files back into place
133     def endCopy(self, ip):
134         nodename = self.getNodeName()
135         print "ssh",ip,"mv " + homepath + "Stop-" + nodename +
            ".tmp " + homepath + "Stop-" + nodename
136         subprocess.call(["ssh",ip,"mv " + homepath + "Stop-" +
```

```
                nodename + ".tmp " + homepath + "Stop-" + nodename
            ])
137
138     #Get node name from whoami file
139     def getNodeName(self):
140         w = open(homepath + "whoami","r")
141         nodename = w.read()
142         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
                + ">" + homepath + "stop"])
149         print "ssh",ip,"echo " + myIP + ">" + homepath + "
                stop"
150
151     def rmTree(self,path):
152         subprocess.call(["ssh",ip,"rm -r '" + path + "'"])
153         print "ssh",ip,"rm -r '" + path + "'"
154
155     #Exclude files matching patterns in the ignore file
156     def exclusions(self, path):
157         try:
158             f = open("./ignore",'r')
159             for line in f:
160                 if line.rstrip() in path:
161                     #print "Ignoring: " + path
162                     return True
163             f.close()
164         except error, e:
165             print e
166         return False
167
168     def initFileSync(self,event):
169         if self.exclusions(event.pathname):
170             #print "Excluded returning"
171             return
172         pathparts = event.pathname.split("/")
173         foldName = "/".join(pathparts[0:len(pathparts)-1])
174         print "Removing watch on: " + foldName
175         wm.rm_watch(wm.get_wd(foldName), rec=True)
```

```
176             self.fileSync(event)
177             print "Putting watch back on: " + foldName
178             wm.add_watch(foldName.rstrip(),pyinotify.ALL_EVENTS,
                    rec=True, auto_add=True)
179
180         #Sync files
181         def fileSync(self,event):
182             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
                        watchedfolder keys"
187                 if folder in event.pathname:
188                     for i in range(0, len(watchedfolders[folder])
                            ,4):
189                         ip = watchedfolders[folder][i]
190                         path = watchedfolders[folder][i+1]
191                         waitTime = watchedfolders[folder][i+2]
192                         lastTime = watchedfolders[folder][i+3]
193                         print "Wait: " + str(waitTime) + " Last: "
                                + str(lastTime)
194                         print "Current ip and path: " + ip + " " +
                                path
195                         readnet.logIPtraffic(ip, event.pathname)
196                         myIP = readnet.getMyIP(ip)
197                         subprocess.call(["ssh",ip,"/usr/bin/python
                            " + homepath + "readnet.py -i " + myIP
                            + " -f " + event.pathname])
198                         print "ssh",ip,"'/usr/bin/python " +
                            homepath + "readnet.py -i " + myIP + " -
                            f " + event.pathname + "'"
199                         fparts = folder.split("/")
200                         fname = fparts[len(fparts)-1]
201                         #stopIP = self.getStopInfo()
202                         #print "STOP: " + stopIP[0] + " " + stopIP
                                [1]
203                         #if stopIP[0] == ip and stopIP[1] == event
                                .pathname:
204                         if self.inStopFile(ip,event.pathname):
205                             print "STOPPED to " + ip + " " + path
206                             #os.remove("./stop");
207                         else:
```

```
208                     print "CONTINUE"
209                     t.timeElapsed(lastTime, waitTime)
210                     watchedfolders[folder][i+3] = str(
                            datetime.datetime.now())
211                     t.updateFolderInfo(watchedfolders)
212                     self.beginCopy(ip)
213                     if args.scp:
214                         #print "SCP: For cpFile in " +
                                folder
215                         for cpFile in glob.glob(folder + "
                                /*"):
216                             #print "SCP GLOB:" + cpFile
217                             print "scp","-rp",cpFile,ip +
                                    ":" + cpFile + ".tmp"
218                             subprocess.call(["scp","-rp",
                                    cpFile,ip + ":" + cpFile + "
                                    .tmp"])
219                             #subprocess.call(["ssh",ip,"
                                    yes y | find /tmp/" + fname
                                    + " -type f -exec cp -p {} "
                                     + path + fname + "/ \; rm /
                                    tmp/" + fname])
220                             print "ssh",ip,"mv " + cpFile
                                    + ".tmp " + cpFile
221                             subprocess.call(["ssh",ip,"mv "
                                    + cpFile + ".tmp " +
                                    cpFile])
222                             print "END SCP GLOB"
223                     elif args.rsync:
224                         print "rsync","-rt",folder,ip + ":
                                " + path
225                         subprocess.call(["rsync","-rt",
                                folder,ip + ":" + path])
226                     else:
227                         time.sleep(5)
228                         print "unison","-batch","-
                                confirmbigdel=false","-times",
                                folder,"ssh://" + ip + "/" +
                                path + fname
229                         subprocess.call(["unison","-batch"
                                ,"-confirmbigdel=false","-times"
                                ,folder,"ssh://" + ip + "/" +
                                path + fname])
```

```
230                             print "Set stop files uniq: " + event.
                                    pathname
231                             self.setStopFileUniq(ip,myIP,event.
                                    pathname,folder)
232                             self.endCopy(ip)
233                         subprocess.call(["ssh",ip,"/usr/bin/python
                                " + homepath + "readnet.py -i " + myIP
                                + " -f " + event.pathname])
234                         readnet.logIPtraffic(ip, event.pathname)
235
236     #def process_IN_CREATE(self, event):
237     #    print "Create:",event.pathname
238     def process_IN_DELETE(self, event):
239         print "Delete: ",event.pathname
240         #self.initFileSync(event)
241     def process_IN_CREATE(self, event):
242         print "CREATE: ",event.pathname
243         self.initFileSync(event)
244     def process_IN_MOVED_FROM(self, event):
245         print "Move from: ",event.pathname
246     #    self.initFileSync(event)
247     def process_IN_MODIFY(self, event):
248         #print "Modify: ",event.pathname
249         self.initFileSync(event)
250     def process_IN_MOVED_TO(self, event):
251         print "Move to: ",event.pathname
252         self.initFileSync(event)
253
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
                    , rec=True, auto_add=True)
265             print "Watching: ", info[0].rstrip()
266             if info[0] not in watchedfolders.keys():
267                 watchedfolders[info[0].rstrip()] = []
```

```
268                 watchedfolders[info[0].rstrip()].append(info[1])
269                 watchedfolders[info[0].rstrip()].append(info[2])
270                 watchedfolders[info[0].rstrip()].append(info[3])
271                 watchedfolders[info[0].rstrip()].append(str(
                        datetime.datetime.now()))
272         f.close()
273
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.
                                ALL_EVENTS, rec=True, auto_add=True)
284                         #print "Watching: ", info[0].rstrip()
285                         if info[0] not in watchedfolders.keys():
286                             watchedfolders[info[0].rstrip()] = []
287                         watchedfolders[info[0].rstrip()].append(
                                info[1])
288                         watchedfolders[info[0].rstrip()].append(
                                info[2])
289                         watchedfolders[info[0].rstrip()].append(
                                info[3])
290                         watchedfolders[info[0].rstrip()].append(
                                str(datetime.datetime.now()))
291                     else:
292                         print "Removing: " + info[0]
293             f.close()
294         except IOError, e:
295             print "Folders.dat does not exist, skipping"
296
297         t.updateFolderInfo(watchedfolders)
298
299         #print watchedfolders
300         eh = MyEventHandler()
301
302         notifier = pyinotify.Notifier(wm,eh)
303         notifier.loop()
304
```

```
305 if __name__ == '__main__':
306     main()
```

# B ReadNet.py

```
 1 import subprocess, datetime, socket, argparse
 2
 3 homepath = "/home/cal/Documents/Private-Sync/"
 4 #homepath = "/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(homepath + "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 coresponding 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()) +
                                " " + interfacename + " download: "
                                + str(download) + " upload: " + str
                               (upload) + "\n")
85                          f.close()
86                      count = 0
87          elif(interface):
88              if(split == "RX" or split == "TX"):
89                  nex = split
90
91  #Log all interfaces
92  def main():
93      ifs = subprocess.check_output("ifconfig -s",shell=True)
94      ilines = ifs.split("\n")
95      for i in range(1,len(ilines)-1):
96          interfacenames.append(ilines[i].split()[0])
97      output = subprocess.check_output("ifconfig",shell=True)
98      splitput = output.split()
99      interface = False
100     interfacename = ""
101     nex = ""
102     count = 0
103     upload = 0
104     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()) + "␣"
                                + interfacename + "␣download:␣" + str(
                               download) + "␣upload:␣" + 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 2)
7  ethcountarr=(1 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[$index]}:/home/cal/
               Documents/Private-Sync/log/* ../logs/"
69          scp cal@${vm_addr_arr[$index]}:/home/cal/Documents/
               Private-Sync/log/* ../logs/
70          let "index++"
71      done
72  }
73
74  function clean {
75      index=0
76      while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
77          echo "ssh cal@${vm_addr_arr[$index]} \"rm ${homepath}
               log/*; rm ${homepath}Stop-*; rm ${homepath}folders.
               dat\""
78          ssh cal@${vm_addr_arr[$index]} "rm ${homepath}log/*;
               rm ${homepath}Stop-*; rm ${homepath}folders.dat"
79          let "index++"
80      done
81  }
82
83  function cleanFold {
84      index=0
```

```
85      while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
86          echo "ssh cal@${vm_addr_arr[$index]} \"rm -rf ${
               folderpath}/*;\""
87          ssh cal@${vm_addr_arr[$index]} "rm -rf ${folderpath
               }/*;"
88          let "index++"
89      done
90  }
91
92  function sendKeys {
93      index=0
94      while [ "$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             vbmMOD ${vm_name_arr[$index]} ${intnetarr[$incount
                ]} ${ifcountarr[$index]}
134             #echo "in: $index"
135             (( ifcountarr[$index]++ ))
136             index=$(search_letters $last)
137             vbmMOD ${vm_name_arr[$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
                    ]} $bigncount $littlencount $(( $littlencount+1
                    ))
157             #echo "in: $index"
158             (( ethcountarr[$index]++ ))
159             (( littlencount++ ))
160             index=$(search_letters $last)
161             ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index
                    ]} $bigncount $littlencount $(( $littlencount-1
                    ))
162             #echo "in: $index"
163             (( ethcountarr[$index]++ ))
164             incount=$incount+1
165             (( bigncount++ ))
166             (( littlencount-- ))
167         fi
168     done <graphs/$1
169 elif [ $2 == "if2" ]; then
170     clear_watched_folders
171
172     while read line
173     do
174         first=$(echo "$line" | awk '{print $1}')
175         last=$(echo "$line" | awk '{print $(NF)}' | sed 's
                /[;]//g')
176         echo "$first and $last"
177         index=$(search_letters $first)
178         if [ "$index" = "None" ]; then
179             #echo "None"
180             :
181         else
```

```
182             ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[
                    $index]} $bigncount $littlencount $((
                    $littlencount+1 ))
183             #echo "in: $index"
184             (( ethcountarr[$index]++ ))
185             (( littlencount++ ))
186             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
```