

A Project

entitled

combox

by

Siddharth Ravikumar

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the  
Masters of Science Degree in Computer Science

---

Dr. Robert C. Green II, Committee Chair

---

Dr. XX, Committee Member

---

Dr. XX, Committee Member

---

Dr. Michael Ogawa, Dean  
College of Graduate Studies

Bowling Green State University

May 2016

Public Domain, No Rights Reserved.

Siddharth Ravikumar has dedicated the work to the public domain by waiving all of his rights to the work worldwide under copyright law, including all related and neighboring rights, to the extent allowed by law. You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

See <https://creativecommons.org/publicdomain/zero/1.0/legalcode> for the full legal verbiage.

An Abstract of  
combox  
by  
Siddharth Ravikumar

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the  
Masters of Science Degree in Computer Science

Bowling Green State University  
May 2016

File storage providers on the Internet have made it non-trivial for individuals to store personal files on the file storage provider's computers. After Mr. Snowden disclosed information about the National Security Agency' (NSA) surveillance programs that allowed the NSA to access information stored on file storage provider' computers, online file storage became a non-solution for storing personal files for everyone who detested the possibility of somebody else being able to access their personal files. In the past, there have been separate efforts to come with a solution to allow individuals to use storage space provided by file storage providers in a way that it made it impossible for file storage providers and to access the files. combox is one such effort. It allows an individual to store personal files in the "combox directory" on all her computers (running GNU/Linux or OS X) and the combox program takes the files, splits and encrypts them and spreads them across file storage providers' directories. Therefore, when an individual uses storage space provided by file storage providers through combox, each file storage provider gets only a part of the file in an encrypted form.

Dedicated to the \$EDITOR I use to literally write everything.

# Acknowledgments

Dr. Robert C. Green II who gave me an opportunity to work on combox.

# Contents

Abstract	iii
Acknowledgments	v
Contents	vi
List of Tables	ix
List of Figures	x
List of Abbreviations	xi
Preface	xii
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>2</b>
<b>3 Literature Review</b>	<b>3</b>
<b>4 Structure and Design</b>	<b>4</b>
4.1 Structure of combox . . . . .	5
4.1.1 combox configuration . . . . .	5
4.1.2 combox directory monitor . . . . .	5
4.1.3 Node directory monitor . . . . .	5
4.1.4 Database structure . . . . .	5

4.2	combox modules overview . . . . .	5
4.3	Language choice . . . . .	8
4.4	DRY . . . . .	9
4.5	Operating system compatibility . . . . .	10
4.6	combox as a python package . . . . .	10
4.7	With the benefit of hindsight . . . . .	12
<b>5</b>	<b>Testing</b>	<b>14</b>
5.1	Unit testing . . . . .	14
5.1.1	Benefits . . . . .	15
5.1.2	Caveats . . . . .	15
5.2	Manual testing . . . . .	16
5.2.1	General setup and notes . . . . .	16
5.2.2	Testing on two GNU/Linux machines . . . . .	17
5.2.2.1	Issues found . . . . .	17
5.2.2.2	Demo . . . . .	18
5.2.3	Testing on a GNU/Linux and an OS X machine . . . . .	20
5.2.3.1	Issues found . . . . .	20
5.2.3.2	Demo . . . . .	21
5.2.4	Testing with a USB stick as a node . . . . .	22
5.2.4.1	Caveats . . . . .	23
5.2.4.2	Demo . . . . .	23
5.3	Stress testing . . . . .	25
5.3.1	flac dump (27 files - 424.798190MiB) . . . . .	26
5.3.1.1	Differences from previous stress test (2015-11-08) . . . . .	26
5.3.2	20MiB - 90MiB dump (27 files - 1620.000000MiB) . . . . .	26
5.3.2.1	Differences from previous stress test (2015-11-08) . . . . .	27

5.3.3	20MiB - 90MiB dump (99 files - 5940.000000MiB) . . . . .	27
5.3.3.1	Differences from previous stress test (2015-11-08) . .	27
5.3.4	20MiB - 90MiB dump (180 files - 10800.000000MiB) . . . . .	27
5.3.4.1	Differences from previous stress test (2015-11-08) . .	28
5.3.5	Tools used . . . . .	28
5.3.6	Observations . . . . .	28
5.3.7	Issues found . . . . .	30
<b>6</b>	<b>Conclusion and Future Work</b>	<b>33</b>
	<b>References</b>	<b>34</b>



# List of Tables

# List of Figures

4-1	High level view of combox on two computers. . . . .	5
5-1	time to process all files . . . . .	29
5-2	avg. time to split and encrypt . . . . .	30
5-3	time to process all files - difference between 2015 and 2016 . . . . .	31
5-4	avg. time to split and encrypt - difference between 2015 and 2016 . . . . .	32

# List of Abbreviations

YAML .....	YAML Ain't Markup Language
CLI .....	Command Line Interface
GUI .....	Graphical User Interface

# Preface

42.

# Chapter 1

## Introduction

# Chapter 2

## Background

# Chapter 3

## Literature Review

# Chapter 4

## Structure and Design

In general, when modeling phenomena in science and engineering, we begin with simplified, incomplete models. As we examine things in greater detail, these simple models become inadequate and must be replaced by more refined models.

---

*Structure and Interpretation of  
Computer Programs, Section  
1.1.5[1]*



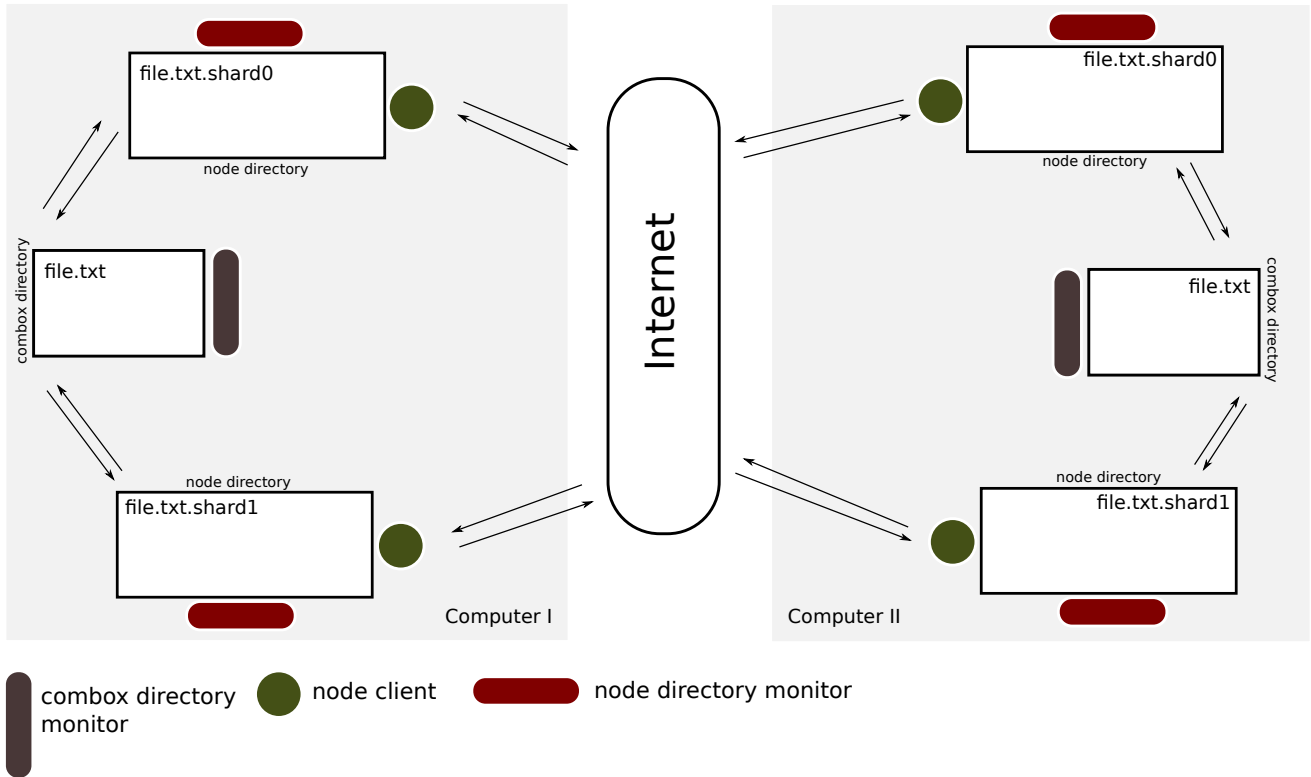


Figure 4-1: High level view of combox on two computers.

## 4.1 Structure of combox

### 4.1.1 combox configuration

### 4.1.2 combox directory monitor

### 4.1.3 Node directory monitor

### 4.1.4 Database structure

## 4.2 combox modules overview

combox is spread into modules that have functions and/or classes. As of 2016-02-04 combox is considerably a small program:

```

$ wc -l combox/*.py
144 combox/cbox.py
178 combox/config.py
241 combox/crypto.py
891 combox/events.py
541 combox/file.py
454 combox/gui.py
  0 combox/__init__.py
 71 combox/log.py
278 combox/silo.py
 29 combox/_version.py
2827 total

```

This section gives an overview of each of the combox modules with extreme brevity:

**combox.cbox** This module contains `run_cb` function runs combox; it creates an instance `threading.Lock` for database access and a shared `threading.Lock` for the `combox.events.ComboxDirMonitor` and `combox.events.NodeDirMonitor`; it initializes an instance `combox.events.ComboxDirMonitor` that monitors the combox directory and an instance of `combox.events.NodeDirMonitor` for each node directory for monitoring the node directories. This modules also houses the `main` function that parses commandline arguments, starts combox configuration if needed or loads the combox configuration file to start running combox.

**combox.config** Accomodates two import functions – `config_cb` and `get_nodedirs`. The `config_cb` is the combox configuration function that allows the user to configure combox; this function was designed in a such way that it was possible to use for both CLI and GUI methods of configuring combox. The `get_nodedirs`

function returns, as a list, the paths of the node directories; this function use used in numerous places in other combox modules.

**combox.crypto** This has functions for encrypting and decrypting data; encrypting and decrypting shards (`encrypt_shards` and `decrypt_shards`); a function for splitting a file into shards, encrypting those shards and spreading them across node directories (`split_and_encrypt`); a function for decrypting the shards from the node directories, reconstructing the file from the decrypted shards and put the file back to the combox directory (`decrypt_and_glue`). Functions `split_and_encrypt` and `decrypt_and_glue` are the two functions that that are extensively used by the `combox.events` module; all other functions in this module are pretty much helper functions are `split_and_encrypt` and `decrypt_and_glue` functions and are not used by other modules.

**combox.events** This module took the most time to write and test and it is the most complex module in combox at the time of writing this report. It contains just two classes – `ComboxDirMonitor` and `NodeDirMonitor`. The `ComboxDirMonitor` inherits the `watchdog.events.LoggingEventHandler` and is responsible for monitoring for changes in the combox directory and doing the right thing when change happens in the combox directory. The `NodeDirMonitor` also inherits `watchdog.events.LoggingEventHandler` and similarly responsible for monitoring a node directory and doing the right thing when a change happens in the node directory; subjectively, `NodeDirMonitor` is slightly more complex than the `ComboxDirMonitor`.

**combox.file** This is the second largest module in combox. It contains utility functions for reading, writing, moving files/directiores, hashing files, splitting a file into shards, glue shards into a file, manipulating directories inside combox and node directories.

**combox.gui** Contains the `ComboxConfigDialog` class; it is the graphical interface for configuring combox. The class uses the Tkinter library[2] for spawning graphical elements. Other graphical libraries include PyQt[3] were considered Tkinter was chosen over others because it works on all Unix systems and Microsoft's Windows and it is part of the core python (version 3).

**combox.log** All the messages to `stdout` and `stderr` are sent through the functions `log_i` and `log_e` functions defined in this module.

**combox.silo** Contains the `ComboxSilo` class which is the canonical interface for combox for managing information about the files in the combox directory. Internally, the `ComboxSilo` class uses the pickleDB library[4].

**combox.\_version** This is *private* module that contains variables that contain the value of the present version and release of combox. The `get_version` function in this module returns the full version number; this function used by `setup.py`.

### 4.3 Language choice

Back in October of 2014, I was learning to write in Python and when I had to start working on combox, I chose to write combox in Python. In my first commit to the combox repository, I had to say this about Python:

```
commit 2def977472b2e77ee88c9177f2d03f12b0263eb0
Author: rsiddharth <rsiddharth@ninthfloor.org>
Date:   Wed Oct 29 23:24:58 2014 -0400
```

```
Initial commit: File splitter & File gluer done.
```

```
...
```

I like to write python FWIW. But after reading a dialect of Lisp when I come back to python, it does not look very beautiful. I guess I'm pretty convinced that there is no language that can ape the beauty of Lisp.

If I were to write that commit message today (2016-02-04), I would've phrased my reflections about Python differently. While I've not found a language that is as intrinsically beautiful as Lisp, I think it is not quite right to compare Lisp and Python. Python is a very readable language and it tends to be very accessible to beginners. Also, it is hard to write unreadable Python code.

## 4.4 DRY

The core functionality of combox is to split, encrypt file shards, spread them across node directories (Google Drive and Dropbox) and decrypt, glue shards and put them back to the combox directory when a file is created/modified/deleted/moved in another computer. The plan was to use external libraries to accomplish things that fell outside the realm of what I consider the "core functionality of combox"; the main reason behind this decision was to duly be an indolent programmer and not indulge in trying to solve problems that others have already solved.

The `watchdog`[5] library was chosen for file monitoring; this library is compatible with Unix systems and Windows. The `pycrypto` library[6] was used for encrypting data; combox uses AES encryption scheme to encrypt file shards. The `pickleDB`[4] library was used to store information about files in the combox directory; this library is not very clean, but, it was what I exactly looking for, if there was no `pickleDB`, I would've most probably written something similar to it and made it as part of combox.

Looking back, the decision to use external libraries reduced the complexity of `combox`, reduced the time to complete the initial working version of `combox` and made it possible to spend more than 3 months just testing and fixing issues in `combox`.

## 4.5 Operating system compatibility

`combox` was developed on a GNU/Linux machine, a conscious effort was made to write in an operating system independent way. The top criteria for choosing a library to use in `combox` was that it had to be compatible on *all* of the three major computing platforms in 2014-2016<sup>1</sup>.

As we were nearing the 0.1.0 release, `combox` was tested on OS X (See chapter 5) and OS X specific issues that were found eventually were eventually fixed. The initial 0.1.0 release was compatible with GNU/Linux and OS X.

After the initial release of `combox`, we wanted to see if `combox` would be compatible with Windows. We found that:

- Setting up the paraphernalia to run `combox` was non-trivial[7].
- The unit tests for the `combox.file` module royally failed.

At the time of writing the report, `combox` is in version 0.2.2 and it still not compatible with Windows. Comprehensive documentation of setting up the development environment for `combox` on Windows was written[7] to make it less cumbersome for anyone would want to work on making `combox` compatible with Windows.

## 4.6 `combox` as a python package

Before version 0.2.0, the canonical way to install `combox` was to pull the source from the `git` repository with:

---

<sup>1</sup>GNU/Linux, OS X and, Windows

```
git clone git://ricketyospace.net/combox.git
```

Then, do:

```
cd combox
```

Finally install combox with:

```
python setup.py install
```

Yes, installing combox on a machine was indeed non-trivial.

Python has a package registry called CheeseShop<sup>2</sup>; all packages registered at the CheeseShop can be installed using `pip` – Python’s platform independent package management system[8] – with:

```
pip install packagename
```

To make it easier for (python) users to install combox on their machine, an effort was made to make it a python package[9]. From version 0.2.0, combox has been registered python package at the CheeseShop. (Python) users can now easily get a copy of combox on their machine with:

```
pip install combox
```

All versions of combox that is available through the CheeseShop are digitally signed using the following GPG key:

```
pub 4096R/00B252AF 2014-09-08 [expires: 2017-09-07]
    Key fingerprint = C174 1162 CEED 5FE8 9954 A4B9 9DF9 7838 00B2 52AF
uid                               Siddharth Ravikumar (sravik) <sravik@bgsu.edu>
sub 4096R/09CECEDB 2014-09-08 [expires: 2017-09-07]
```

All versions of combox’s source are also available as a compressed TAR ball and as a ZIP archive; they can be downloaded from <https://ricketyospace.net/combox/releases.html>.

---

<sup>2</sup>code name for Python Package Index, see <https://wiki.python.org/moin/CheeseShop>

## 4.7 With the benefit of hindsight

combox's node monitor (`combox.events.NodeDirMonitor`) was written with the assumption that the node monitor will be the only entity that will be making changes to the node directory that it is monitoring. When started testing combox with node clients (Dropbox client and Google Drive client), we observed that the node clients made changes to the node directory when a file was created/modified/renamed/deleted; for instance, when a shard, in the Dropbox node directory, was modified on a remote computer, the Dropbox client would first pull the newer version of the shard under the `.dropbox.cache` directory as a temporary file, move the older version of the shard under `.dropbox.cache` as a backup, and finally move the latest version of the shard, stored as a temporary file under the `.dropbox.cache` directory, to the respective location in the Dropbox node directory; when a shard, in the Google Drive node directory, was remotely modified on a remote computer, the Google Drive client would delete the older version of the shard from the Google Drive node directory and then create the newer version of the shard in the respective location under the Google Drive node directory. Since combox did not know about the node client's behaviour, it confused combox and broke it royally; we had to make major changes to the `combox.eventns.NodeDirMonitor` class to make combox aware of the node client's behavior, this eventually brutally obliterated the simplicity of the `combox.eventns.NodeDirMonitor` class which I was proud of.

I'm not sure how I would have written the `combox.events` module if I had known about the Dropbox and Google Drive client's behaviour before writing the `combox.events.NodeDirMonitor` or the `combox.events.ComboxDirMonitor` classes. Looking back, if there one thing I would want to re-think/redo, it is the `combox.events` module.

The most important lesson I'm taking away from the experience of writing combox



is the insight of how easy it is to ruthlessly crush the simplicity of a program due to unforeseen use cases.

<3

# Chapter 5

## Testing

Testing shows the presence, not the absence of bugs.

---

*Dijkstra*[10]

### 5.1 Unit testing

The `nose`[11] testing framework was used to write unit tests for the functions and classes part of the `combox.config`, `combox.crypto`, `combox.events`, `combox.file`, `combox.silo` `combox._version` modules. Unit tests were not written for `combox.cbox`, `combox.gui`, `combox.combox.log` modules.

Unit tests for `combox` become reality by pure serendipity. During the time, when I started working on `combox`, I was learning to use the `nose` library to unit test python code. Since, `combox` was being written in python, I started making it a norm to write unit tests for functions and classes in `combox` modules.

As mentioned before, unit tests were not written for some modules either because it would make no sense to write one (for the `combox.cbox` module, for instance, which basically uses functions and classes defined in other modules to run `combox`) or it was not clear how to write unit tests it (the `combox.gui` contains just the `ComboxConfigDialog` a graphical front-end which uses the configuration function

defined in the `combox.config` module to complete the combox configuration based on the user input).

It must be noted here that pure Test Driven Development (TDD) was not observed – most of the time the function/class was written before the its corresponding test was written.

### 5.1.1 Benefits

While writing unit tests definitely increased the time to write a particular feature, it enabled me to immediately check if a feature worked as it should for the given use case or given set of inputs.

With the benefit of hindsight, unit tests greatly helped in testing the compatibility of combox on OSX. Before the `v0.1.0` release, combox's node directory monitor always assumed that a file's first shard (`shard0`) is always available; while this assumption did not create any problems on GNU/Linux, on OS X, this assumption made the node directory monitor to behave erratically – this issue (bug #4[12] was immediately found when the unit tests were run for the first time on OS X. Another instance where unit tests helped was just before the `v0.2.0` release; major changes, including the introduction of file locks in the `ComboxDirMonitor`, were made to the `combox.events`. When the unit tests were run OS X, two tests failed, revealing a difference in behavior of `watchdog`[5] on GNU/Linux and OS X on file creation[13]; without unit tests, there is a high probability that this bug would never have been found by now.

### 5.1.2 Caveats

Unit tests are helpful in testing the correctness of a feature for N number of use cases but it does not necessarily mean the written feature correctly behaves for use

cases that the author of the feature did not consider or did not think about while writing the respective feature. As Dijkstra correctly observed:

Unit tests failed to reveal bugs #4, #5 #6 #7 #5 #10 #11[12]; these bugs were found when manually testing `combox`.

## 5.2 Manual testing

The unit tests for the `combox.events` module test the correctness of the `ComboxDirMonitor` and `NodeDirMonitor` independently; in order to comprehensively test the correctness of both `ComboxDirMonitor` and `NodeDirMonitor`, it was required to manually test `combox` running on more than one computer. As you'll see in the following subsections, several bugs were found and fixed while doing manual testing.

Three different types of setups were used to test `combox`. The first kind of setup has two GNU/Linux machines each using `combox` to sync files between each other with Dropbox and Google Drive being the nodes; the second kind of setup has a GNU/Linux machine and a OS X machine each using `combox` to sync files between each other with Dropbox and Google Drive being the nodes; the third kind of setup has a GNU/Linux machine and OS X machine each using `combox` to sync files between each other with Dropbox, Google Drive and a USB stick as nodes.

### 5.2.1 General setup and notes

- On the GNU/Linux machines, the official Dropbox client was used to sync the Dropbox node directory to Dropbox' servers. `rc1one`[14] was used to sync the Google Drive node directory to Google Drive' servers; At the time of testing, Google Drive did not have client for GNU/Linux.
- On OS X, the official Dropbox client was used to sync the Dropbox node directory to Dropbox's servers; the official Google Drive client was used to sync the

Google Drive node directory to Google Drive' servers.

- Since combox is extremely event-driven, combox must be started before the Dropbox and Google Drive clients start syncing their respective directories (nodes).

## 5.2.2 Testing on two GNU/Linux machines

combox was run to two GNU/Linux machines and a file was alternatively created/modified/renamed/deleted on an of the GNU/Linux machine and it was verified if the respective file was also created/modified/renamed/deleted on the other GNU/Linux machine. One of the GNU/Linux machine (`lyra`) was a virtual machine running Debian GNU/Linux stable (version 8.x); the other GNU/Linux machine (`grus`) was a physical machine running Debian GNU/Linux testing. The node directories to scatter the files' shards were the Dropbox directory and Google Drive directory. The official Dropbox client was used to automatically sync files from the Dropbox directory to the Dropbox' server; `rc1one`[14] was used to sync files from Google Drive directory to Google Drive' server.

### 5.2.2.1 Issues found

- Some editors, especially on POSIX complaint systems, create backup version of the file being edited. combox was detecting this backup file as a “new file” and it split it into shards, encrypted the shards and scattered the shards across the node directories. The right thing for combox to do was to ignore these backup files and do nothing about them. This issue was fixed on 2015-09-29[12]. Now the `ComboxDirMonitor`, on a “file created” or “file modified” event, returns from the `on_created` or `on_modified` callback when it finds that the file is a backup/temporary file.

- Dropbox client maintains the `.dropbox.cache` directory under the root of the Dropbox directory.
  - When a file (shard) was created on another computer, the Dropbox client pulls the new file (shard) to this computer into `.dropbox.cache` as a temporary file and then moves the new file (shard) to its respective location with the appropriate name.
  - When a file (shard) was modified on another computer, the Dropbox client pulls the modified file (shard) to this computer into the `.dropbox.cache` as a temporary file; moves the old version of the file (shard) under the Dropbox directory into the `.dropbox.cache`; finally moves the updated copy of the file, stored as a temporary file, into the Dropbox directory to its respective location with the appropriate name.
  - When a file (shard) was deleted on another computer, the Dropbox client moves the delete file into the `.dropbox.cache` directory on this computer.

All of the above behavior of the Dropbox client epically broke combox. Commits `3d714c5` to `6e1133f`<sup>[15]</sup> fixed combox by making it aware of Dropbox's client behavior.

#### 5.2.2.2 Demo

Demo of combox being used on two GNU/Linux machines can be viewed at <https://rickety.space.net/combox/combox-2-gnus.webm>.

`lyra` (virtual machine) and `grus` (bare-metal) are the two GNU/Linux machines being used for the demo.

Description of what happens in the demo follows:

- (lyra) install combox.
- (lyra) run combox (test mode).

- (lyra) create file `walden.pond` with content “It must be beautiful there”.
  - (lyra) sync Google Drive using `rclone`.
  - (grus) sync Google Drive using `rclone`.
  - (grus) git pull latest copy of `combox`.
  - (grus) install `combox`
  - (grus) run `combox` (testing mode).
  - (grus) verify that `walden.pond` was create on this machine.
  - (grus) append 'Peaceful too.' to `walden.pond`.
  - (grus) sync Google Drive using `rclone`.
  - (lyra) sync Google Drive using `rclone`.
  - (lyra) verify that the latest copy of `walden.pond` is there in the `combox` directory;
- it should contain 'Peaceful too.' in the last line.
- (lyra) append “I’ve a dream” to `walden.pond`.
  - (lyra) sync Google Drive using `rclone`.
  - (grus) sync Google Drive using `rclone`.
  - (grus) verify that the latest copy of `walden.pond` is there in the `combox` directory;
- it should contain “I’ve a dream” in the last line.
- (grus) remove `walden.pond` from `combox` directory.
  - (grus) sync Google Drive using `rclone`.
  - (lyra) sync Google Drive using `rclone`.
  - (lyra) verify that `walden.pond` is removed from the `combox` directory.
  - (grus) open dropbox and Google drive accounts from the web browser.
  - (lyra) create file `manufacturing.consent`. with content “Chomsky stuff?”.
  - (lyra) sync Google Drive using `rclone`.
  - (grus) sync Google Drive using `rclone`.
  - (grus) verify that `manufacturing.consent` was created in the `combox` directory.

- (grus) verify that the shards of `manufacturing.consent` were created on Dropbox and Google Drive through the web browser.

### 5.2.3 Testing on a GNU/Linux and an OS X machine

combox was run on a GNU/Linux machine and an OS X machine and a file was alternatively created/modified/renamed/deleted on one of the machine and it was verified if the respective file was also created/modified/renamed/deleted on the other machine. The GNU/Linux machine was a virtual machine (`lyra`) running Debian GNU/Linux stable; the OS X machine was on Mavericks (10.9) during the initial stage of testing, later it was upgraded to Yosemite (10.10). The node directories to scatter files' shards were the Dropbox directory and the Google Drive directory. The official Dropbox client was used to automatically sync files from the Dropbox directory to the Dropbox' server on both the GNU/Linux machine and the OS X machine; the official Google Drive client was used to automatically sync files from the Google Drive directory to Google Drive' server on OS X and `rc1one`[14] was used to sync files from the Google Drive directory to Google Drive's server on GNU/Linux.

#### 5.2.3.1 Issues found

- When a file was modified on another computer, on this computer combox assumed that first shard (`shard0`) will be updated first and also counted on the existence of the first shard (`shard0`). It was observed that the order in which the shards were updated were unpredictable on this computer and if the first shard (`shard0`) was stored in the Dropbox directory, it will momentarily disappear before the most updated shard becomes available in the Dropbox directory; this broke combox. This issue was fixed on 2015-08-25[16]. This issue is not got to do with the nature of the setup but it is related to the Dropbox's behavior elaborated in section 5.2.2.1.



- The official Google Drive client when it pulls an updated version of the file from Google Drive' server, instead directly updating the respective file on the computer, it deletes the older version of the file and creates the latest version of the file at the respective location in the Google Drive directory; this behavior of the Google Drive confused and broke combox. This issue was fixed 2015-09-06 by making combox under the official Google Client's behavior[17].
- When a non-empty directory was move/renamed on another computer, the old directory was not getting properly deleted on this computer; this was happening because the files under the directory being renamed were not deleted when it was time for `NodeDirMonitor` to `rmdir` the old directory. This issue again is not specific to the nature of the setup but was found while testing combox on this setup. This issue was fixed on 2015-09-12[18].
- It was found that `combox.file.rm_path` function failed when it was given a non-existent path to remove; this issue was fixed on 2015-09-12[19].

### 5.2.3.2 Demo

Demo of combox being used on a GNU/Linux machine and OS X machine can be viewed at <https://ricketyospace.net/combox/combox-gnu-osx.webm>

`lyra` is the GNU/Linux (virtual) machine and `dhcp-129-1-66-1` is the OS X machine that is being used for the demo. The OS X machine is accessed through VNC[20].

Description of what happens in the demo follows:

- (`lyra`) create file `cat.stevens` with content "peace train".
- (`lyra`) sync Google Drive using `rclone`.
- (`dhcp-129-1-66-1`) verify that file `cat.stevens` is created with content "peace train".

- (dhcp-129-1-66-1) append string “moonshadow” to file `cat.stevens`.
- (lyra) sync Google Drive using `rclone`.
- (lyra) verify that the file `cat.stevens` was updated (modified); last line must have the string “moonshadow”.
- (lyra) append string “father and son” to the file `cat.stevens`.
- (lyra) sync Google Drive using `rclone`.
- (dhcp-129-1-66-1) verify that the file `cat.stevens` was updated (modified); last line must have the string “father and son”.
- (dhcp-129-1-66-1) rename file `cat.stevens` to `yusuf.islam`
- (lyra) sync Google Drive using `rclone`.
- (lyra) verify that the file `cat.stevens` was renamed to `yusuf.islam`.

#### 5.2.4 Testing with a USB stick as a node

combox was run on a GNU/Linux machine and an OS X machine and a file was alternatively created/modified/deleted on one of the machine and it was verified if the respective file was also create/modified/deleted on the other machine. The GNU/Linux machine was a physical machine (`grus`) running Debian GNU/Linux stable; The OS X machine was on Mavericks (10.9). The node directories to scatter files’ shards were the Dropbox directory, Google Drive directory and the USB stick (`ZAPHOD`, FAT filesystem). The official Dropbox client was used to automatically sync files from Dropbox directory to Dropbox’ server on both the GNU/Linux machine and OS X machine; the official Google Drive client was used to automatically sync files from the Google Drive directory to Google Drive’ server on OS X and `rclone`[14] was used to sync files from the Google Drive directory to Google Drive’s server on GNU/Linux; the same USB stick (`ZAPHOD`) was used on bothe GNU/Linux and Dropbox to store the third shard (`shard2`) of a file.

#### 5.2.4.1 Caveats

- When a removable USB disk is used as a node, combox must be turned off before ejecting/unmounting the USB disk; combox does not expect a node directory to disappear when it is running, if the USB disk is removed when combox is running, then combox goes to a undefined state.
- When a file modified on machine A is synced to machine B, combox must be turned on first before turning on Dropbox and Google Drive clients and the shard in the USB disk needs to be “touched” for combox to detect that the file was modified on the remote computer and update the file locally on this machine.
- File rename/move does not work. To make it work, core functionality of combox must be re-written.

#### 5.2.4.2 Demo

Demo of combox being used with a USB stick as the third node can be view at <https://rickety.space.net/combox/combox-usb-node-demo.webm>

`grus` is the GNU/Linux machine and `dhcp-129-1-66-1` is the OS X machine that is being used for the demo. `ZAPHOD` is the FAT32 USB stick used as the third node.

Description of what happens in the demo follows:

- (`grus`) start combox.
- (`grus`) create a file called `simon.and.garfunkel` with content “the boxer”.
- (`grus`) sync Google Drive using `rclone`.
- (`grus`) stop combox.
- (`grus`) unmount USB stick (`ZAPHOD`) from `grus`.
- (`dhcp-129-1-66-1`) mount USB stick (`ZAPHOD`) to (`dhcp-129-1-66-1`).
- (`dhcp-129-1-66-1`) start Dropbox client.

- (dhcp-129-1-66-1) start Google Drive client.
- (dhcp-129-1-66-1) start combox.
- (dhcp-129-1-66-1) verify that the file `simon.and.garfunkel` with content “the boxer” was created.
- (dhcp-129-1-66-1) append string “mrs. robinson” to file `simon.and.garfunkel`.
- (dhcp-129-1-66-1) stop combox.
- (dhcp-129-1-66-1) stop Google Drive client.
- (dhcp-129-1-66-1) stop Dropbox client.
- (dhcp-129-1-66-1) unmount the USB stick (ZAPHOD) from (dhcp-129-1-66-1).
- (grus) mount the USB stick (ZAPHOD) to (grus).
- (grus) start combox.
- (grus) start Dropbox client.
- (grus) sync Google Drive using `rclone`.
- (grus) touch `simon.and.garfunkel.shard2` in the USB stick (ZAPHOD).
- (grus) verify that the file `simon.and.garfunkel` is updated; the last line must contain the string “mrs. robinson”.
- (grus) remove the file `simon.and.garfunkel`.
- (grus) sync Google Drive using `rclone`.
- (grus) unmount the USB stick (ZAPHOD) from (grus).
- (grus) stop Dropbox client.
- (dhcp-129-1-66-1) mount the USB stick (ZAPHOD) to (dhcp-129-1-66-1).
- (dhcp-129-1-66-1) start Google Drive client.
- (dhcp-129-1-66-1) start Dropbox client.
- (dhcp-129-1-66-1) start combox.
- (dhcp-129-1-66-1) verify that the file `simon.and.garfunkel` was deleted.

## 5.3 Stress testing

Large number of files of different sizes were dumped to the combox directory between an one second interval to see how combox responds to high load. The file dump size was varied from 424.798190MiB (27 files) to 10800.000000MiB (180 files); the average time taken to split a file and the total time to process all files were calculated for each dump.

Stress testing was first done on 2015-11-08. In mid November the `ComboxDirMonitor` was drastically modified to make it use the file Lock shared the instances of `NodeDirMonitor`[21]; my hunch was that this change in `ComboxDirMonitor` directly affected the performance of combox and therefore the results that were got from stress testing on 2015-11-08 would no longer be valid. Stress testing was again done on 2016-01-16; the results of this stress test are in sections 5.3.1 to 5.3.4, section 5.3.5 gives information about the tools used for stress testing, section 5.3.6 contains the observations and comparisons between this stress test and the one done on 2015-11-08, lastly section 5.3.7 reveals the issues that were found with combox by virtue of doing the stress tests.

### 5.3.1 flac dump (27 files - 424.798190MiB)

field	value
delay between a file dump	1s
start time of processing	11:00:54
end time of processing	11:01:38
total time taken to process all files	00:00:44
no. of files	27
total size of all files	445433187.000000 bytes (424.798190MiB)
avg. file size	16497525.000000 bytes (15.733266MiB)
avg. time to split and encrypt a file	352.583370 ms

#### 5.3.1.1 Differences from previous stress test (2015-11-08)

- Total time to process all files was faster by 1min3secs.
- Average time to split and encrypt a file reduced by 28.3379630000000002ms.

### 5.3.2 20MiB - 90MiB dump (27 files - 1620.000000MiB)

field	value
delay between a file dump	1s
start time of processing	12:26:45
end time of processing	12:29:07
total time taken to process all files	00:02:22
no. of files	27
total size of all files	1698693120.000000 bytes (1620.000000MiB)
avg. file size	62914560.000000 bytes (60.000000MiB)
avg. time to split and encrypt a file	2670.596556ms

### 5.3.2.1 Differences from previous stress test (2015-11-08)

- Total time to process all files was slower by 4secs.
- Average time to split and encrypt a file reduced by 25.52536999999984ms.

### 5.3.3 20MiB - 90MiB dump (99 files - 5940.000000MiB)

field	value
delay between a file dump	1s
start time of processing	13:10:16
end time of processing	13:19:26
total time taken to process all files	00:09:10
no. of files	99
total size of all files	6228541440.000000 bytes (5940.000000MiB)
avg. file size	62914560.000000 bytes (60.000000MiB)
avg. time to split and encrypt a file	2979.647586ms

### 5.3.3.1 Differences from previous stress test (2015-11-08)

- Total time to process all files was faster by 59secs.
- Average time to split and encrypt a file increased by 206.20906100000002ms.

### 5.3.4 20MiB - 90MiB dump (180 files - 10800.000000MiB)

field	value
delay between a file dump	1s
start time of processing	13:42:06
end time of processing	14:00:10
total time taken to process all files	00:18:04
no. of files	180
total size of all files	11324620800.000000 bytes (10800.000000MiB)
avg. file size	62914560.000000 bytes (60.000000MiB)
avg. time to split and encrypt a file	3423.087539ms

#### 5.3.4.1 Differences from previous stress test (2015-11-08)

- Total time to process all files was slower by 1min2secs
- Average time to split and encrypt a file increased by 399.87623299999996ms.

#### 5.3.5 Tools used

The `dump` script[22] was used to dump files to the `combox` directory between one second intervals; a night of Emacs Lisp indulgence made it possible to quickly slurp the required data from the `combox` output and calculate the average time to split and encrypt a file and the total amount of time taken to process the files for a given dump[23]; lastly `org-mode` was used to document all data gathered during stress testing[24].

#### 5.3.6 Observations

- Figure 5-1 shows the time it takes `combox` to process files for a given file dump<sup>1</sup>. As can be observed from the graph, the total time taken to process all the files

---

<sup>1</sup>A “file dump” here means a bunch of files copied to the `combox` directory between 1 sec intervals.



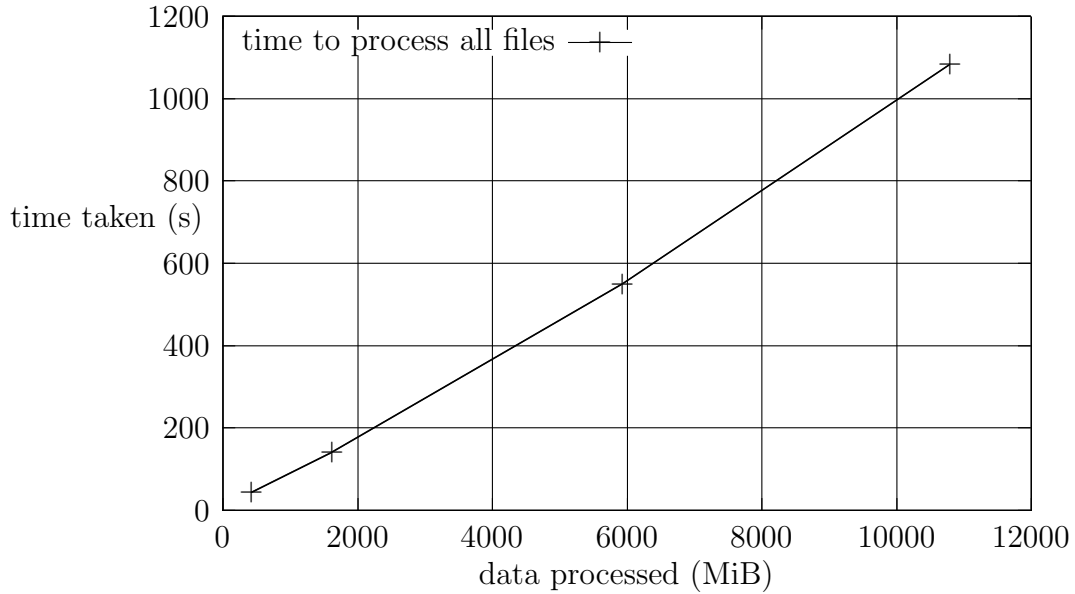


Figure 5-1: time to process all files

tends almost linearly increase with the increase in the size of the file dump<sup>2</sup>.

- Figure 5-2 show the average time it takes combox to split and encrypt a file for a given file dump. There is a steep increase in the average time from the 424.798190MiB dump and the 1620.000000MiB dump, after which the average time to split and encrypt a file seems to almost linearly increase; The main reason for this is that the average file size for dumps from 1620.000000MiB to 10800.000000MiB are the same.
- Figure 5-3 shows the graphs for the total amount of time taken to process all files for a given file dump in the 2016-01-16 and 2015-11-8 stress test. The amount of time needed to process all fills seems to be reduced for the 5940.000000MiB file dump when compared to the 2015 stress test results and it seems to be slightly higher for the 10800.000000MiB file dump when compared to the 2015 stress test.

---

<sup>2</sup>The “size of the file dump” is the total size of all files in a given file dump.

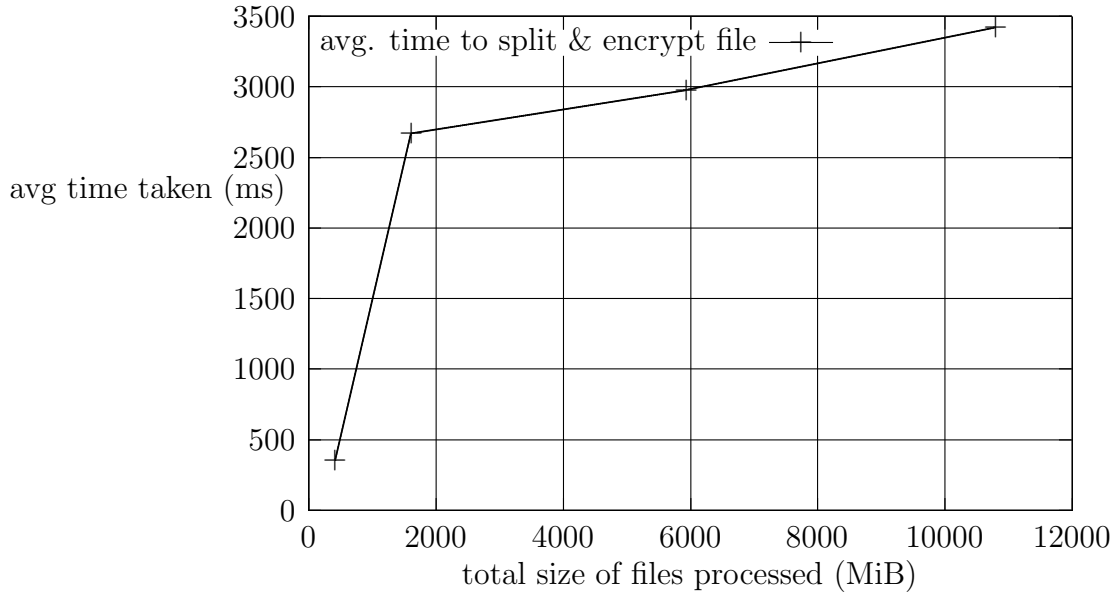


Figure 5-2: avg. time to split and encrypt

- Similarly, figure 5-4 shows the graphs for the average time to split and encrypt for a given file dump in the 2016-01-16 and the 2015-11-8 stress test. The average time taken seems to be almost the same for the 424.798190MiB and the 1620.000000 dump, but for the 5940.000000MiB and the 10800.000000MiB dump the average time taken seems to be higher for the 2016 stress test when compared to the 2015 stress test.

### 5.3.7 Issues found

- Initially when combox was stress tested with huge files, combox would get overwhelmed leading to the computer running out of memory and the load average sometimes peaking at 8. At first, it was assumed that there was a bug in combox which caused this to happen, but later it was found that `watchdog[5]` was generating a large number “file modified” events when a huge file (~500MiB was modified). To prevent `watchdog` from generating a large number “file modified” events for a single modification of a huge file, a delay proportional to

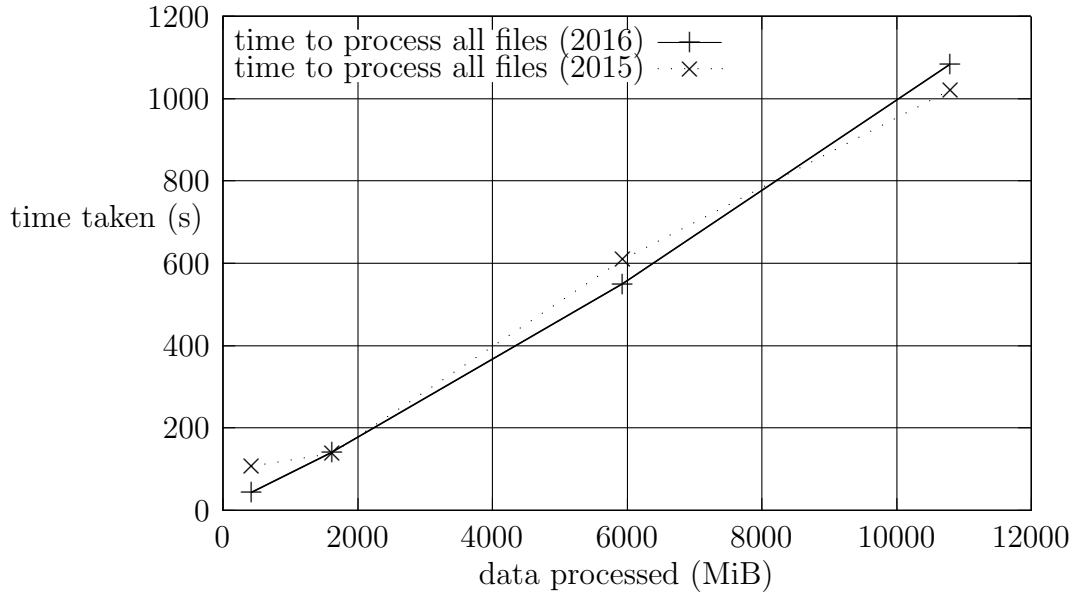


Figure 5-3: time to process all files - difference between 2015 and 2016

the size of the file was created in the `on_modified` callback methods in both `ComboxDirMonitor` and `NodeDirMonitor`[25], this fixed the issue. Also, this it might be useful to note here that this was “the” hardest issue I dealt with in working on `combox`.

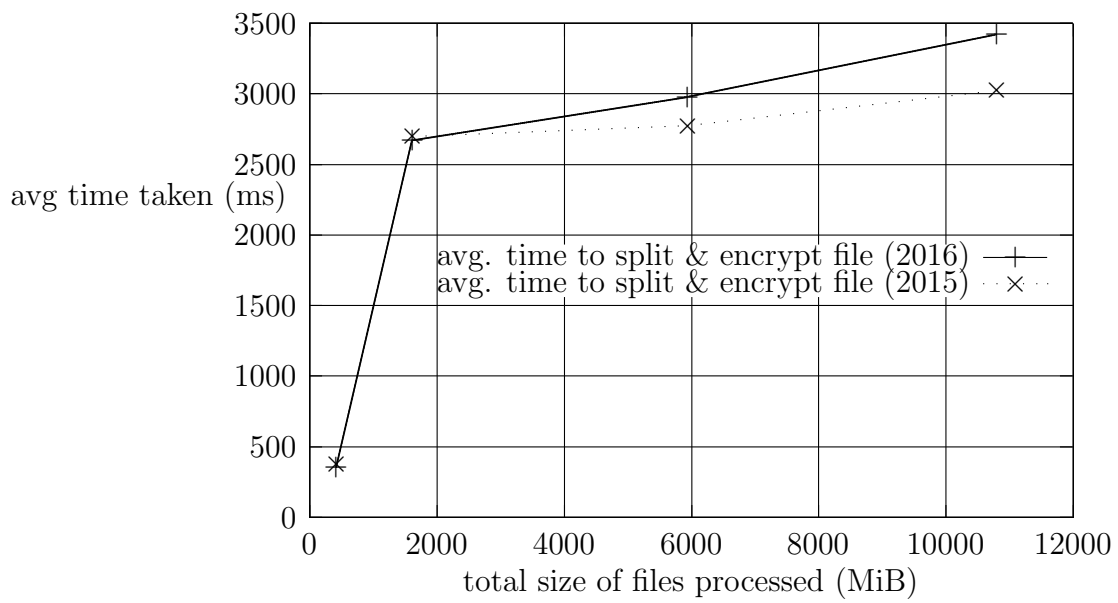


Figure 5-4: avg. time to split and encrypt - difference between 2015 and 2016

# Chapter 6

## Conclusion and Future Work

# References

- [1] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. MIT Press, 1996.
- [2] “Tkinter - python interface to tcl/tk.” [Online]. Available: <https://docs.python.org/2/library/tkinter.html>
- [3] “Pyqt - python binding of the cross-platform gui toolkit qt.” [Online]. Available: <https://riverbankcomputing.com/software/pyqt/intro>
- [4] “pickledb - lightweight and simple key-value store.” [Online]. Available: <https://pythonhosted.org/pickleDB>
- [5] “Watchdog - python api library and shell utilities to monitor file system events.” [Online]. Available: <https://pythonhosted.org/watchdog/>
- [6] “Pycrypto - the python cryptography toolkit.” [Online]. Available: <https://www.dlitz.net/software/pycrypto/>
- [7] “setup combox on windows.” [Online]. Available: <https://ricketyospace.net/combox/setup#windows>
- [8] “pip - pypa recommended tool for installing python packages.” [Online]. Available: <https://pip.pypa.io/en/stable/>
- [9] “Python packaging user guide.” [Online]. Available: <https://packaging.python.org/en/latest/>

- [10] J. Buxton and B. Randell, “Software engineering techniques,” NATO Science Committee, Tech. Rep. p. 16, 1969. [Online]. Available: <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF>
- [11] “Nose - a nicer testing for python.” [Online]. Available: <https://nose.readthedocs.org/en/latest/>
- [12] “combox issue tracker (org-mode).” [Online]. Available: <https://git.ricketyspace.net/combox/plain/TODO.org>
- [13] “combox - watchdog 'file create event' bug fix.” [Online]. Available: <https://git.ricketyspace.net/combox/commit/?id=8c86e7c28738c66c0e04ae7886b44dbcdcf6369>
- [14] “rclone - command line program to sync files and directories to and from google drive.” [Online]. Available: <http://rclone.org/>
- [15] “combox - git commits - dropbox client behavior fix.” [Online]. Available: <https://git.ricketyspace.net/combox/log/?qt=range&q=3d714c5..6e1133f>
- [16] “combox - git commit - shard modification fix.” [Online]. Available: <https://git.ricketyspace.net/combox/commit/?id=d5b52030348d40600b4c9256f76e5183a85fbb>
- [17] “combox - git commit - google client behavior fix.” [Online]. Available: <https://git.ricketyspace.net/combox/commit/?id=37385a90f90cb9d4dfd13d9d2e3cbcace8011e9>
- [18] “combox - git commit - bug six fix.” [Online]. Available: <https://git.ricketyspace.net/combox/commit/?id=9d14db03da5d10d5ab0d7cc76b20e7b1ed552>
- [19] “combox - git commit - bug seven fix.” [Online]. Available: <https://git.ricketyspace.net/combox/commit/?id=422238eb4904de14842221fa09a2b4028801af1>
- [20] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, “Virtual network computing,” *Internet Computing, IEEE*, vol. 2, no. 1, pp. 33–38, Jan 1998.

- [21] “combox - git commit - bug eleven fix.” [Online]. Available:  
<https://git.ricketyspace.net/combox/commit/?id=5aa1ba0c1dcad62931ba27bb66bf115233086c>
- [22] “dump script (python) for stressing testing combox.” [Online]. Available:  
<https://git.ricketyspace.net/combox-paper/plain/dumper/dump>
- [23] “dumps.el - emacs lisp magic to slurp and process output from combox.” [Online].  
Available: <https://git.ricketyspace.net/combox-paper/plain/scripts/dumps.el>
- [24] “benchmarks.org - document containing all information  
about the stress testing combox.” [Online]. Available:  
<https://git.ricketyspace.net/combox-paper/plain/notes/benchmarks.org>
- [25] “combox - git commit - bug ten fix.” [Online]. Available:  
<https://git.ricketyspace.net/combox/commit?id=7ed3c9cbe6e56223b043a23408474f9df08f119e>