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, Advisor

Dr. Michael Ogawa, Dean
College of Graduate Studies

Bowling Green State University
May 2016

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 trivial for individuals to store their personal files online. At the same time, there has been revelations about the existence of a billion dollar surveillance industry[1] that is building and selling tools to governments and dictatorships to snoop on its own citizens. In a world which is fast becoming Orwellian, storing personal files on storage provided by file storage providers is not even an option for some individuals. In the past, there have been separate efforts to come up 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 or "third parties" to access the user' files. combox is one such effort, it allows an individual to store their personal files on the storage provided by Google Drive and Dropbox in such a way that only part of each file (in encrypted form) is stored in Google Drive/Dropbox. combox is a python package compatible with GNU/Linux and OS X platforms. This report contains an overview of combox – what it is, how it was developed and tested; explains how combox is different from Vollmar's Combo-Box[2]; reviews projects similar to combox that help computer users to securely store personal files on storage provided by file storage providers; lastly, enlists things that can be done to improve combox.

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

List of Tables

List of Figures

List of Abbreviations

YAML	YAML Ain't Markup Language
CLI	Command Line Interface
GUI	Graphical User Interface
TUI	Text User Interface
JSON	JavaScript Object Notation

Preface

Faithfully follow the steps below with utmost diligence; after arriving at this page, always begin reading step 1.

1. Read the abstract? If yes, proceed to step 2; otherwise, go to page iii and read the abstract.

2. Is there enough time and motivation to read a long report? If so, set N equal to 1; if not, set N equal to 5.

3. Begin reading chapter N. Do *not* pay heed to the trite epigraph at the beginning of the chapter.

4. Set M equal to 1.

5. Start reading section $N.M$

6. Is section $N.M$ of any interest? If not, go to step 7; otherwise read this section and then go to step 7.

7. Increase M by one. If section $N.M$ exists, go to step 5; otherwise go to step 8.

8. Increase N by one. If $N = 6$, go to step 9; if not, go to step 3.

9. Close the report, do something else.

The above procedure is based on Knuth's procedure for reading his "The Art of Computer Programming" series[3].

Chapter 1

Introduction

> From a security perspective, if
> you're connected, you're screwed.
>
> *Daniel J. Bernstein*

Internet companies have made it trivial for computer users to store data/information on their servers and at the same time there is a lot of evidence of governments and other powerful organizations being able to access information/data stored on the Internet companies' computers [1]. Also, most companies add a standard clause in their privacy policy that allow them to disclose information about users or information stored/created by users to "third parties":

> *Law & Order.* We may disclose your information to third parties if we determine that such disclosure is reasonably necessary to (a) comply with the law; (b) protect any person from death or serious bodily injury; (c) prevent fraud or abuse of Dropbox or our users; or (d) protect Dropbox's property rights. – Dropbox Privacy Policy [4]

In this type of world, it would be good to have a program that would encrypt all the data/information before storing it on the storage provided by Internet companies. combox aims to be one such program which not only encrypts but stores only a part of the encrypted data/information on the storage provided by an Internet company,

thus making it non-trivial for "third parties" to access the user's data/information in its entirety. Section 1.1 gives a conceptual introduction to combox; Section 1.2 enumerates how combox is different from Vollmar's Combo-Box; lastly, Section 1.3 contains information on how one can start using combox.

1.1 What is combox?

combox allows the user to store all of their files in the "combox directory" and combox picks each file stored in the combox directory, splits them into N shards, encrypts each of the N shards and spreads the shards to N node directories. A "node directory" is the directory of the file storage provider (Dropbox directory is a node directory). Fig. 1-1, illustrates how a file called `strunk-white.pdf` is split, encrypted and spread across N node directories. Shards `strunk-white.pdf.shard0` to `strunk-white.pdf.shardN` are encrypted.

combox does not use the API provided by the file storage providers to sync encrypted shards stored in the node directories to the respective file storage providers' data store. Instead, it depends on the respective file storage provider's client program to sync the shards.

combox can be used on all of the user's computers. For instance, the user can install combox on their second computer and combox will reconstruct the file from the encrypted shards stored in the node directories into the combox directory on their second computer; Fig. 1-2 illustrates this. Here too, combox depends on the client program of the respective file storage provider to sync shards to/from the file storage provider's data store and to/from the respective node directory on the user's computer.

As of combox version `0.2.3`, combox is compatible on GNU/Linux and OS X, it supports just two file storage providers – Google Drive and Dropbox.

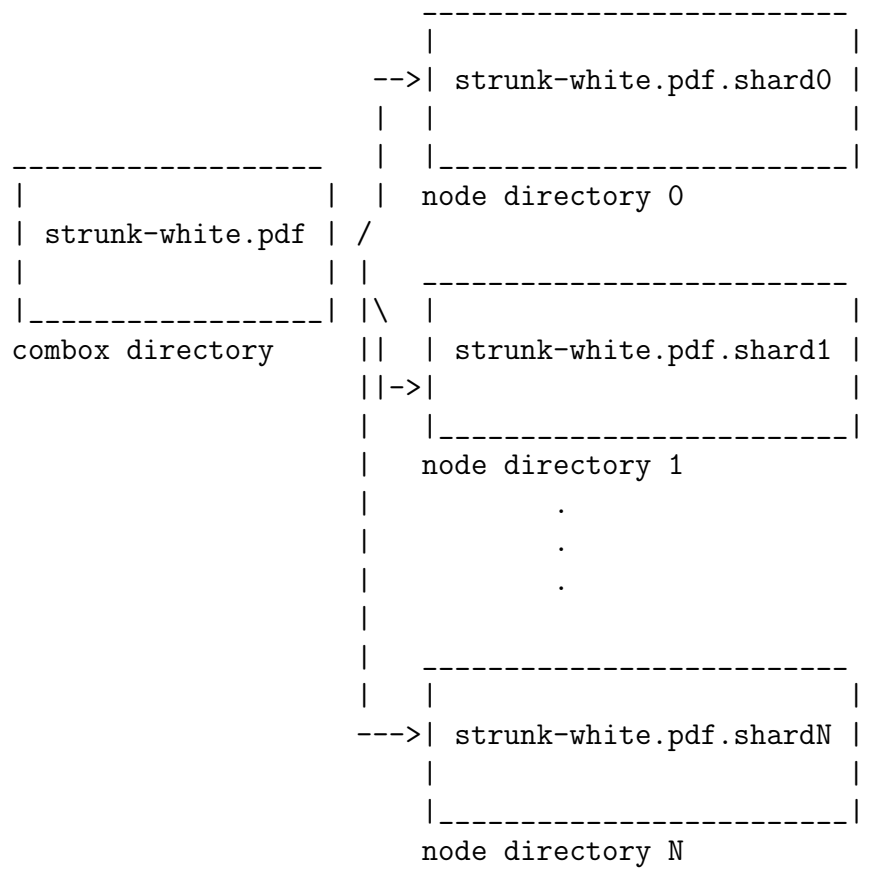

Figure 1-1: combox overview - Splitting a file and spreading it across N node directories.

1.2 How is combox different from Combo-Box?

Combo-Box by Wesley Vollmar [2] was the first implementation of the idea of storing encrypted shards of a file on storage provided different file storage providers and depending on the file storage provider's client to sync shards to their respective data store. Differences between Vollmar's Combo-Box and combox are enumerated below:

Platform Combo-Box runs on Microsoft Windows, whereas combox runs on GNU/Linux and OS X and is not compatible with Microsoft Windows as of version 0.2.3.

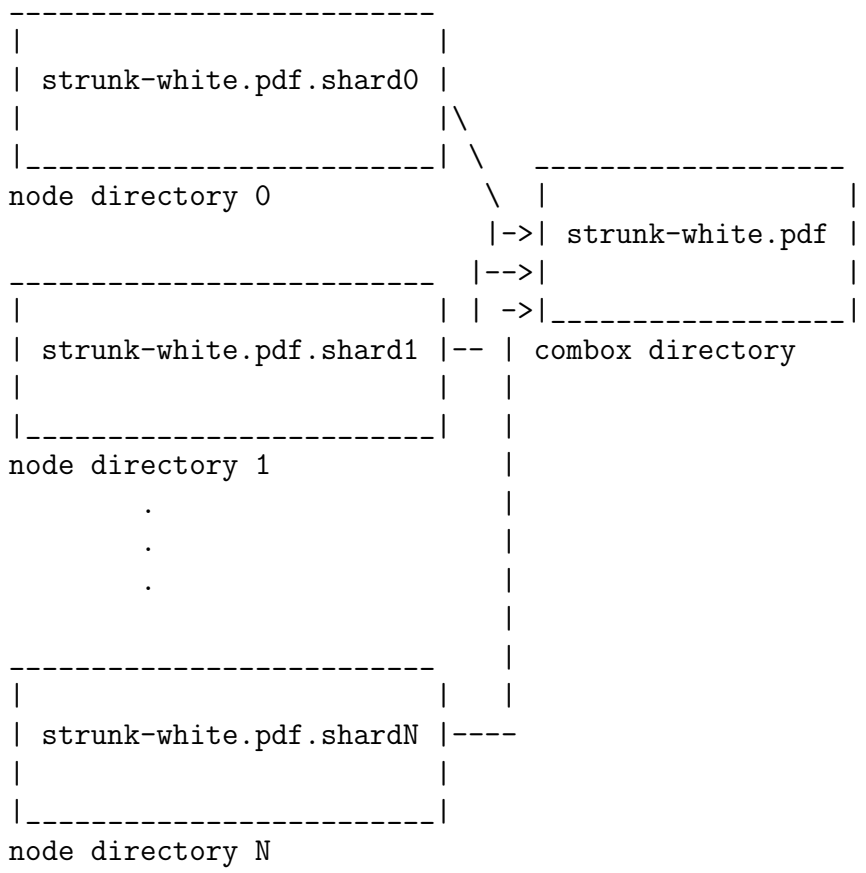

Figure 1-2: combox overview - Reconstructing a file from the encrypted shards.

File splitting Combo-Box splits a file into shards based on the space available on each node directory [2], while combox is not yet cognizant about space left on each node directory and splits the file into N equal shards, where N is equal to the number of node directories.

User Interface Combo-Box is a graphical application while combox is mostly a command-line program. combox's configuration wizard has a graphical interface. The configuration wizard has a command-line interface too for users who like TUI.

Database Combo-Box uses a traditional SQL database with two tables to keep track of files' shards, files' hash, files' last "sync time" and for "security and stability" uses stored procedures that retrieve/store information in the database [2].

combox on the other hand uses a key-value data store to track the files stored in the combox directory using the pickleDB library [5]. The key-value data store is a JSON file and all access to this data store is done through an instance of `combox.silo.ComboxSilo` class [1] which ensures that only one thread can read from or write to the data store at any time through a lock (`threading.Lock`). In the data store, combox keeps track of the hashes of all the files stored in the combox directory; the data store also contains dictionaries that track number of shards which have been create/moved/modified/deleted on another computer.

Installation Combo-Box uses the proprietary InstallShield [6] to install the program, setup shortcuts and registry settings [2].

combox is a python package, it can either be installed through python's package manager (`pip` [7]) with `pip install combox` or it can be installed from the source with the standard `python setup.py install`.

Configuration Combo-Box saves its configuration inside the Combo-Box directory and this configuration is shared by all computers on which the user chooses to run Combo-Box, by virtue of this, the file providers' directories and the Combo-Box directory must be in the same locations on all the computers.

combox stores its configuration at `$HOME/.combox/config.yaml`. The configuration file is not shared on computers on which the user runs combox. This makes it possible to keep the combox directory and the directories of the file storage providers' (node directories) in different locations on each computer.

[1] https://git.ricketyspace.net/combox/tree/combox/silo.py?id=fb7fdd218#n29

The configuration file is a YAML file and can be directly edited by the user if they wish to.

1.3 Using combox

Installing and running combox is relatively easy for Unix users:

```
$ pip install combox
$ combox
```

For detailed information on installing combox, see https://ricketyspace.net/combox/setup/.

1.3.1 Caveats

combox is extremely event-driven and depends on filesystem events to do the correct action when a file is created/modified/moved/deleted, so the user must make sure to start combox before starting the file storage providers' client programs that sync encrypted shards to the respective node directories. On GNU/Linux distributions this can be automated through the distribution's start-up system (most GNU/Linux distributions seem to use `systemd` [8]).

Chapter 2

Background and Literature Review

> Books serve to show a man that those original thoughts of his aren't very new after all
>
> *Abraham Lincoln*

The idea of unifying the storage provided by multiple Internet file storage providers and storing all the content in an encrypted form is not new. In the past, computer researchers and programmers have devised different methods to use multiple file storage providers' storage space. This chapter gives an overview of the work done by Yeo et al. in unifying the storage provided by Dropbox, Box, Google Drive and Skydrive on Android devices [9](Section 2.1); SkyCDS, a content delivery service, by Gonzalez et al., which uses publish/subscribe overlay paradigm and stores the content across multiple cloud storage providers such that only part of the content (in encrypted form) is stored on each file storage provider [10](Section 2.2); lastly, `git-annex`, by Joey Hess [11], that allows one to version control and keep track of large files with a possibility of encrypting files that are stored in "special remotes" – storage provided by Internet file storage providers (Section 2.3).

2.1 Multi Cloud Storage Prototype

In the paper "Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices", Yeo et al. show their Android mobile application, a prototype, which unifies storage provided by Dropbox, Box, Google Drive and SkyDrive. The application allows the user to store all their information in a single location on their phone and it uses erasure coding [12] to split each file into `n + k` fragments and spreads the encrypted fragments across storage provided by the file storage providers. All basic file operations – Create, Rename, Update, Delete (CRUD) – are possible. Information about the files stored in the unified location is stored in a SQLite database. Unlike combox, which depends the file storage provider' client to sync file fragments/shards to the file storage provider's data store, the Android application developed by Yeo et al. takes the responsibility to sync file fragments/shards to each file storage provider and uses the OAuth 2.0 [13] protocol for authorization.

For encrypting file fragments, they use AES-256. The key for encrypting file fragments is derived from the user's password by using Password-Based Key Derivation Function (PBKDF2) [14]. For erasure coding they use the JigDFS library [15]. The Android application is able do "progressive streaming" of media files, this means large media files can be streamed in real-time from the file storage providers' data store. "Progressive streaming" is an attractive feature in a "resource constrained" device where storage is expensive.

Yeo et al. propose methods for achieving data de-duplication, file compression based on file type, intelligent pre-fetching and caching of file fragments, and "automatic restoration in exploiting file-versioning". These features were not implemented in the prototype Android application.

It becomes apparent that Yeo et al. work is of immense importance when the

research done by Yang et al. is taken into consideration, which found that 59% of the users who use "cloud storage service" access the service through a smart phone and 42.2% users access it for audio/video [16]. The research by Yang et al. suggests a trend of users' preference for small hand-held computers over laptops and desktops.

2.2 SkyCDS

SkyCDS, by Gonzalez et al., is a content delivery system that splits and spreads the content across multiple file storage providers [10]. According to Gonzalez et al., the main reason for designing and developing SkyCDS was to prevent content providers from getting locked into just one file storage provider and to minimize loss when a file storage provider goes out of business or if there is temporary outage in the storage service provided by the file storage provider.

In SkyCDS, the content delivery to subscribers of the content is segregated into two distinct layers – Metadata Flow Layer and the Content Flow Layer. The publisher of the content largely interacts with the Metadata Flow Layer that controls and keeps track of what content is published and the subscriber also largely interacts with the Metadata Flow layer to subscribe to content published in the content delivery system. The Content Flow Layer is where the content is stored across multiple file storage providers. The publisher is responsible for publishing the content using the "delivery workflow" (part of the Content Flow Layer) and the subscriber uses the "retrieve workflow" to get access to the subscribed content.

When content has to be dispersed to k file storage providers, the content is split into n chunks, $n > k$. This file splitting seems to produce 66.7% of redundancy overhead [10]. This file splitting scheme also looks very similar to erasure coding, but Gonzalez et al. don't explicitly state that the content splitting scheme is indeed "erasure coding". The splitting of content is done by the "delivery workflow" engine

which is invoked when the publisher triggers the action to publish the respective content to subscribers.

To evaluate the effectiveness of SkyCDS, Gonzalez et al. state that they've done a case study using the data obtained from the European Space Astronomy Center (ESAC) for the Soil Moisture Ocean Salinity. In this study, a group of organizations, in two different continents, used SkyCDS to share satellite images with each other. According to Gonzalez et al. this study attested SkyCDS as a viable option for content delivery with respective to performance, cost of file storage space and reliability.

2.3 git-annex

`git-annex` allows one to version controlled large files that are not usually feasible to version control under `git` [17]. `git-annex` checks in the name and other meta-data about the files in git and stores the actual content under `.git/annex` directory. When a file is added to `git-annex`, a symlink of the file is created in place of the file and the content of the file itself is stored under the `.git/annex` directory.

For instance, say there is a file called `deb-nicholson-80s.medium.webm` that was downloaded from the Internet to the `git-annex` directory:

```
 git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)


   deb-nicholson-80s.medium.webm


 ls -l
total 105708
```

```
...
-rw-r--r-- 1 rsd rsd 108196923 May  5  2015 deb-nicholson-80s.medium.webm
...
```

When this file is added to `git-annex` with `git annex add`, the file turns into a symlink to a file under the `.git/annex` directory:

```
 git annex add deb-nicholson-80s.medium.webm
add deb-nicholson-80s.medium.webm ok
(recording state in git...)


 ls -l
...
lrwxrwxrwx 1 rsd rsd   207 May  5  2015 deb-nicholson-80s.medium.webm
-> ../.git/annex/objects/3j/vG/SHA256E-s108196923--7de9484ee96908268e
21b451eb9805552c32b44da08e70ee861332c87352944f.webm/SHA256E-s10819692
3--7de9484ee96908268e21b451eb9805552c32b44da08e70ee861332c87352944f.w
ebm


 git commit -m "Added video/deb-nicholson-80s.medium.webm"
[master efa1775] Added video/deb-nicholson-80s.medium.webm
 1 file changed, 1 insertion(+)
 create mode 120000 video/deb-nicholson-80s.medium.webm
```

Now, the file `deb-nicholson-80s.medium.webm` is checked into `git-annex` and the command `git annex sync` can be issued to sync the repository to other `git-annex` repositories. It must be noted here that when the repository is synced, the file content itself is not transferred to the other `git-annex` repositories, only the file's name and its meta-data that is stored in a separate git branch called `git-annex` are transferred [18]. In order to create a copy of a given file in another git annex repository, `git annex get /path/to/filename.ext` has to done.

`git-annex` has this feature called "special remotes" [19], that allows one to push files checked into `git-annex` to storage provided by file storage providers. At the time of writing this report, `git-annex` supports pushing data to the following file storage services:

- Amazon S3
- Amazon Glacier
- Internet Archive via S3
- Box.com
- Google drive
- Google Cloud Storage
- Mega.co.nz
- SkyDrive
- OwnCloud
- Flickr
- IMAP
- Usenet
- chef-vault
- hubiC
- pCloud
- ipfs
- Ceph
- Blackblaze's B2

All data pushed to file storage provider's servers can optionally be encrypted using one's GPG key. For instance, to encrypt data that is pushed to the Amazon S3 special remote, the following command is used [20]:

```
$ git annex initremote cloud type=S3 keyid=2512E3C7
initremote cloud (encryption setup with gpg key C910D9222512E3C7)
                 (checking bucket) (creating bucket in US) (gpg) ok
$ git annex describe cloud "at Amazon's US datacenter"
describe cloud ok
```

where `2512E3C7` is the id of the GPG key to use for encrypting data pushed to the Amazon S3 special remote. It is also possible to store each file that is pushed to the remotes as a set of chunks of size `N`, to do that we do:

```
$ git annex initremote cloud type=S3 chunk=1MiB keyid=2512E3C7
initremote cloud (encryption setup with gpg key C910D9222512E3C7)
                 (checking bucket) (creating bucket in US) (gpg) ok
$ git annex describe cloud "at Amazon's US datacenter"
describe cloud ok
```

Upon completion, each file that has to be pushed to the Amazon S3 special remote is divided into 1MiB chunks, each chunk is encrypted using the GPG key `2512E3C7` and the encrypted chunks are finally pushed to the Amazon S3 remote. It must be noted here that unlike the Multi Cloud Storage Prototype or SkyCDS or combox, in `git-annex` when we are using file chunking all the chunks go to the same location – in this case, the Amazon S3 remote.

Chapter 3

Architecture 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* [21]

3.1 Structure of combox

combox consists of two main components – the combox directory and the node directories. The combox directory is the place where the user stores all of their files, the node directories are the directories under which encrypted shards of the files (in the combox directory) are scattered to. A node directory is the file storage provider's directory. For instance, the Dropbox directory and the Google Drive directory are

node directories.

When a file, `humans.txt`, is created in the combox directory, combox splits `humans.txt` into `N` shards, where `N` is the number of node directories. If there are two node directories (Dropbox directory and Google Drive directory), then 2 shards are created. Each shard of the file is then encrypted and the encrypted shards are spread evenly across the node directories. Now, the Dropbox client and the Google client will sync the respective shards that was place under their directories to their respective data store.

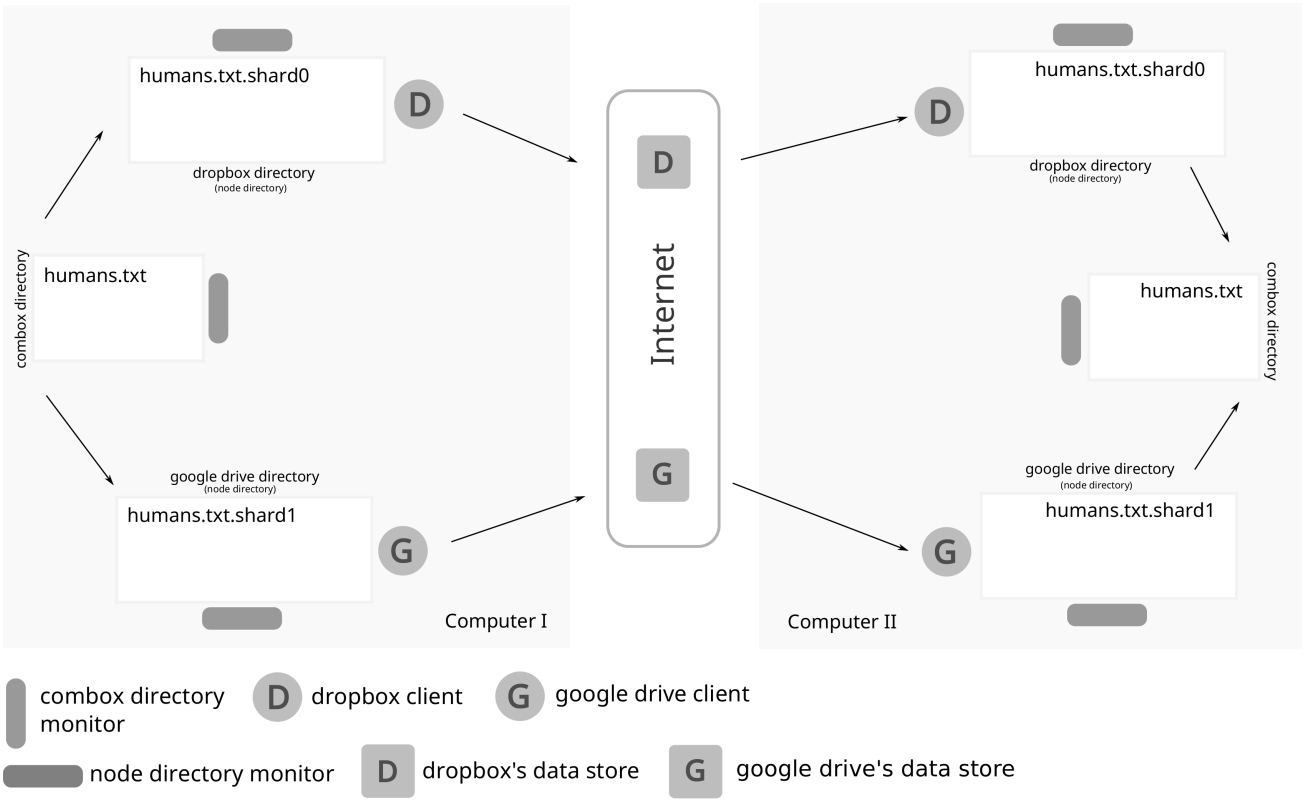

Figure 3-1: Overview of how file creation works when combox is setup on two computers.

Now, when the user moves to their second computer, the node clients (Dropbox client and the Google Drive client) will sync the new encrypted shards to their respective directories. Once the encrypted shards are synced to the node directories,

combox will pick the encrypted shards – `humans.txt.shard0`, `humans.txt.shard1` – decrypt them and reconstruct into `humans.txt` and place it in the respective location under the combox directory. Fig. 3-1 illustrates this. The process is similar for file modification, deletion and rename/move.

3.1.1 combox configuration

The combox configuration wizard triggers automatically when combox finds that it is not configured. The combox configuration wizard configures the combox directory, asks the user to point to the location of the node directories, and reads the key (passphrase) to be used to encrypt file shards that are spread across the node directories. The combox configuration is written to `$HOME/.combox/config.yaml`. This YAML configuration file can be manually edited by the user.

The `config_cb` [1] function in the `combox.config` module is responsible for carrying out the combox configuration. Prior to version `0.2.0`, the combox configuration was purely done through the Command Line Interface (CLI). From `0.2.0` on wards, by default, the combox configuration is done through a graphical interface. It is still possible to configure combox through the CLI with the `--cli` switch.

A demo of combox configuration using the graphical interface on GNU/Linux can be viewed `https://ricketyspace.net/combox/combox-config-gui-glued-gnu.webm`. T he same demo of combox configuration using the graphical interface on OS X can be viewed `https://ricketyspace.net/combox/combox-config-gui-glued-osx.webm`.

3.1.2 combox directory monitor

combox directory monitor is an instance of `combox.events.ComboxDirMonitor` [2] monitoring the combox directory for changes. When changes are made to the combox

[1] https://git.ricketyspace.net/combox/tree/combox/config.py?id=fb7fdd21#n90
[2] https://git.ricketyspace.net/combox/tree/combox/events.py?id=fb7fdd21#n42

directory, the combox directory monitor is responsible for correctly detecting the type of change and doing the right thing at that instance of time.

When a file is created in the combox directory, the combox directory monitor will take that file, split it into `N` (equal to the number of node directories) shards, encrypt the shards, spread the encrypted shards to the node directories, and finally store the hash of the file in the local combox data store.

When a file is modified in the combox directory, the combox directory monitor will take that modified file, split it into `N` (equal to the number of node directories) shards, encrypt the shards, spread the encrypted shards to the node directories, and finally update the hash of the file in the local combox data store.

When a file is deleted in the combox directory, the combox directory monitor will remove the encrypted shards of the file in the node directories and get rid of the file's hash from the local combox data store.

When a file is moved/renamed in the combox directory, the combox directory monitor will move/rename encrypted shards in all the node directories, remove the file's hash from the local combox data store and store the hash of file under its new name.

3.1.3 Node directory monitor

Node directory monitor is an instance of `combox.events.NodeDirMonitor` [3] that monitors a node directory. When changes are made to the node directory, the node directory monitor is responsible for correctly detecting the type of change and doing the right thing at that instance of time. Each node directory has a dedicated node directory monitor. If there are 2 node directories, then combox will instantiate 2 node directory monitors.

[3] https://git.ricketyspace.net/combox/tree/combox/events.py?id=fb7fdd21#n352

When an encrypted shard is created in the node directory due to a file created on another computer, the node directory first checks if the respective file' encrypted shard(s) has/have arrived in other node directory/directories. If all encrypted shards have arrived, then the node directory takes all the encrypted shards, decrypts them, reconstructs the file and puts the file in the combox directory on this computer and stores the hash of the newly created file in the local combox data store. If all the encrypted shards have not arrived, then the node directory does not do anything. It must be observed here that the node directory monitor of the last node directory which gets the encrypted shard will be the one to perform the file reconstruction and creation.

When an encrypted shard is modified in the node directory due to a file modified on another computer, the node directory first checks if the respective file' modified encrypted shard(s) has/have arrived in other node directory/directories. If all modified encrypted shards have arrived, then the node directory takes all the modified encrypted shards, decrypts them, reconstructs the file and puts the modified version of the file in the combox directory on this computer and updates the file's hash in the local combox data store. If all the modified encrypted shards have not arrived, then the node directory does not do anything. It must be observed here that the node directory monitor of the last node directory which gets the modified encrypted shard will be the one to perform the file reconstruction and will place the modified file in the combox directory.

When an encrypted shard is deleted in the node directory due to a file deleted on another computer, the node directory first checks if the respective file' encrypted shard(s) has/have been deleted in other node directory/directories. If all encrypted shards have been deleted from the node directories, then the node directory deletes the file in the combox directory on this computer and removes its information from the local combox data store. If all encrypted shards have not been deleted, then

the node directory does not do anything. It must be observed here that the node directory monitor of the last node directory in which the encrypted shard is deleted will be the one to delete the file from the combox directory.

When an encrypted shard is moved/renamed in the node directory due to a file moved/renamed on another computer, the node directory first checks if the respective file' moved/renamed encrypted shard(s) has/have arrived in other node directory/directories. If all moved/renamed encrypted shards have arrived, then the node directory takes all the moved/renamed encrypted shards, decrypts them, reconstructs the moved/renamed file and puts the moved/renamed file in the combox directory on this computer and stores the hash under the file' new name in the local combox data store. If the all the moved/renamed encrypted shards have not arrived, then the node directory does not do anything. It must be observed here that the node directory monitor of the last node directory which gets the moved/renamed encrypted shard will be the one to perform the file reconstruction and will place the moved/renamed file in the combox directory.

3.1.4 combox data store

To "keep it simple, stupid", combox tracks bare minimum information about the files that are stored in the combox directory, depending on file system events to do the right thing when changes takes place in the combox directory.

The only information that is stored in the combox data store with regards to a file in the combox directory is its SHA-512 hash. The SHA-512 hash of a file is enough information to detect changes in the file. In the data store, there are also four dictionaries – `file_moved`, `file_deleted`, `file_created`, `file_modified` – which track the number of shards of a file that were moved/deleted/created/modified due the respective file being moved/deleted/created/modified on another computer. These four dictionaries are primarily used by the `NodeDirMonitor` to detect remote

file movement/deletion/creation/modification and triggering file reconstruction from the encrypted shards at the right time.

The data store is a JSON file on the disk, stored by default at `$HOME/.combox/silo.db`. The `combox.silo.ComboxSilo` [4] is the sole interface to read from and write to the data store. The data store is primarily accessed and modified by the combox directory monitor (`ComboxDirMonitor`) and the node directory monitor (`NodeDirMonitor`) through a shared `threading. Lock` that ensures that only one entity [5] can access/modify the database at a time.

Below is an illustration of the structure of the combox data store:

```
{
  "/home/rsd/combox/ipsum.txt": "e3206df2bb2b3091103ab9d...",
  "/home/rsd/combox/tk-shot-osx.png": "7fcf1b44c15dd95e0...",
  "/home/rsd/combox/thgttg-21st.png": "0040eedfc3eeab546...",
  "/home/rsd/combox/lorem.txt": "5851dd7a4870ff165facb71...",
  "/home/rsd/combox/the-red-star.jpg": "4b818126d882e552...",
  "file_moved": {},
  "file_deleted": {},
  "file_created": {},
  "file_modified": {},
}
```

The `combox.silo.ComboxSilo`, which is the sole interface to read from and write to the database, uses the pickleDB library [5]. The pickleDB is a very basic key-value store which allows one to store information in the JSON format.

It must be noted that the combox data store on each computer is independent

[4] https://git.ricketyspace.net/combox/tree/combox/silo.py?id=v0.2.2#n29

[5] An entity can be the combox directory monitor or one of the node directory monitors

and does not communicate or make transactions with the combox data store located in other computers.

3.2 combox modules overview

combox is spread into modules that have functions and/or classes. Currently, combox is considerably a small program consisting of the following files:

```
$ 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 [6] This module contains `run_cb` function that starts/initiates combox. The `run_cb` function creates an instance `threading.Lock` for combox data store access and another instance of `threading.Lock` which is shared by instances of `combox.events.ComboxDirMonitor` and `combox.events.NodeDirMonitor`, initializes an instance `combox.events.ComboxDirMonitor` that monitors the

[6] https://git.ricketyspace.net/combox/tree/combox/cbox.py?id=fb7fdd21

combox directory and an instance of `combox.events.NodeDirMonitor` for each node directory. 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 [7] Accommodates two import functions – `config_cb` and `get_nodedirs`. The `config_cb` is the combox configuration function that allows the user to configure combox, it was designed in a such way that it could be used by both the commandline and graphical interfaces for configuring combox. The `get_nodedirs` function returns, as a list, the paths of the node directories, it is used in numerous places in other combox modules.

combox.crypto [8] 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 putting the file 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 for `split_and_encrypt` and `decrypt_and_glue` functions and are not used by other modules.

combox.events [9] 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

[7] https://git.ricketyspace.net/combox/tree/combox/config.py?id=fb7fdd21
[8] https://git.ricketyspace.net/combox/tree/combox/crypto.py?id=fb7fdd21
[9] https://git.ricketyspace.net/combox/tree/combox/events.py?id=fb7fdd21

monitoring for changes in the combox directory and doing the right thing when a 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 [10] This is the second largest module in combox. It contains utility functions for reading, writing, moving files/directories, hashing files, splitting a file into shards, gluing shards into a file, manipulating directories inside combox and node directories.

combox.gui [11] Contains the `ComboxConfigDialog` class; it is the graphical interface for configuring combox. The class uses the Tkinter library [22] for spawning graphical elements. Other graphical libraries including PyQt [23] were considered, Tkinter was chosen over others due to compatibility with all Unix, Unix-like systems and Microsoft Windows and it is part of the standard python library from python version 3 on wards.

combox.log [12] All the messages to `stdout` and `stderr` are sent through the `log_i` and `log_e` functions defined in this module.

combox.silo [13] 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 [5].

combox._version [14] This is *private* module that contains variables that contain the

[10] https://git.ricketyspace.net/combox/tree/combox/file.py?id=fb7fdd21
[11] https://git.ricketyspace.net/combox/tree/combox/gui.py?id=fb7fdd21
[12] https://git.ricketyspace.net/combox/tree/combox/log.py?id=fb7fdd21
[13] https://git.ricketyspace.net/combox/tree/combox/silo.py?id=fb7fdd21
[14] https://git.ricketyspace.net/combox/tree/combox/_version.py?id=fb7fdd21

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`.

3.3 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 the "core functionality of combox". The main reason behind this decision was to not indulge in trying to solve problems that others have already solved.

Accordingly, the `watchdog` [24] library was chosen for file monitoring. This library is compatible with Unix, Unix-like systems and Microsoft Windows. The `pycrypto` library [25] was used for encrypting data. Combox uses AES encryption scheme to encrypt file shards. The `pickleDB` [5] library was used to store information about files in the combox directory.

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.

3.4 Operating system compatibility

combox was developed on a GNU/Linux machine. A conscious effort was made to write the software 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 [15].

Prior to the `0.1.0` release, combox was tested on OS X (See chapter 4) and OS X specific issues that were found were eventually fixed. The initial `0.1.0` release of combox was compatible with GNU/Linux and OS X.

After the initial release of combox, it was seen if combox would be compatible with Microsoft Windows out of the box. it was found that:

- Setting up the paraphernalia to run combox was non-trivial [26].
- The unit tests for the `combox.file` module failed on the Windows Operating System.

At the time of writing the report, combox is at version `0.2.3` and it is not compatible with Microsoft Windows. Comprehensive documentation for setting up the development environment for combox on Microsoft Windows was written [26] to make it less cumbersome for anyone who would want to work on making combox compatible with Microsoft Windows.

3.5 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:

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

Then, do:

```
cd combox
```

Finally install combox with:

[15] GNU/Linux, OS X and, Microsoft Windows

```
python setup.py install
```

Python has a package registry called CheeseShop [16]. All packages registered at the CheeseShop can be installed using `pip` – Python's platform independent package management system [7] – 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 [27]. From version `0.2.0`, combox has been registered as a 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 are 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://ricketyspace.net/combox/releases.html`.

[16] code name for Python Package Index, see https://wiki.python.org/moin/CheeseShop

Chapter 4

Testing

> Testing shows the presence, not the absence of bugs.
>
> *Dijkstra*[28]

4.1 Unit testing

The `nose` [29] testing framework was used to write unit tests for the functions and classes that are part of the `combox.config`, `combox.crypto`, `combox.events`, `combox.file`, `combox.silo` and `combox._version` modules. Unit tests were not written for `combox.cbox`, `combox.gui` and `combox.combox.log` modules either because it did not sense to write one – for instance, the `combox.cbox` module, which uses functions and classes defined in other modules which are unit tested – or it was not clear how to write unit tests for it (the `combox.gui` module).

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.

4.1.1 Benefits

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

Unit tests greatly helped in testing the compatibility of combox on OS X. 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) 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 on OS X, two tests failed, revealing a difference in behavior of watchdog [24] on GNU/Linux and OS X on file creation [1]. Without unit tests, there is a high probability that this bug would never have been found by now.

4.1.2 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.

Unit tests failed to reveal bugs #5, #6, #7, #10 and #11 [2]. These bugs were found when manually testing combox.

[1] https://git.ricketyspace.net/combox/commit/?id=8c86e7c28738c66c0e04ae7886b44dbcdfc6369exo
[2] https://git.ricketyspace.net/combox/plain/TODO.org

4.2 Manual testing

The unit tests for the `combox.events` module tested 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. Several bugs were found and fixed while doing manual testing.

Three different types of setups were used to manually 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.

4.2.1 General setup and notes

- On the GNU/Linux machines, the official Dropbox client was used to sync the Dropbox node directory to Dropbox' data store. `rclone` [30] was used to sync the Google Drive node directory to Google Drive' data store. At the time of testing, Google Drive did not have a client program for GNU/Linux which can sync to Google Drive's data store.

- On OS X, the official Dropbox client was used to sync the Dropbox node directory to Dropbox's data store. The official Google Drive client was used to sync the Google Drive node directory to Google Driver' data store.

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

4.2.2 Testing on two GNU/Linux machines

combox was run on two GNU/Linux machines and a file was alternatively created/modified/renamed/deleted on one 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 machines, (`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' data store. `rclone` [30] was used to sync files from Google Drive directory to Google Drive' data store.

4.2.2.1 Issues found

- Some editors, especially on POSIX complaint systems, create a 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` [3]. 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 tem-

[3] https://git.ricketyspace.net/combox/plain/TODO.org

porary 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`, and 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 deleted file into the `.dropbox.cache` directory on this computer.

All of the above behavior of the Dropbox client broke combox. Commits between `3d714c5` to `6e1133f` [4] fixed combox by making it aware of Dropbox's client behavior.

4.2.2.2 Demo

A demo of combox being used on two GNU/Linux machines can be viewed at `https://ricketyspace.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.

[4] https://git.ricketyspace.net/combox/log/?qt=range&q=3d714c5..6e1133f

- (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.

4.2.3 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' data store 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' data store on OS X and `rclone` [30] was used to sync files from the Google Drive directory to Google Drive's data store on GNU/Linux.

4.2.3.1 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 [5]. 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 4.2.2.1.

[5] https://git.ricketyspace.net/combox/commit/?id=d5b52030348d40600b4c9256f76e5183a85fbb17

- When the official Google Drive client pulls an updated version of the file from Google Drive' data store, 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 client confused and broke combox. This issue was fixed 2015-09-06 by making combox aware of the official Google Client's behavior [6].

- 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, sometimes, the files under the directory being renamed were not deleted when it was time for `NodeDirMonitor` to `rmdir` the old directory. This issue was fixed on 2015-09-12 [7].

- 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 [8].

4.2.3.2 Demo

A demo of combox being used on a GNU/Linux machine and OS X machine can be viewed at `https://ricketyspace.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[31].

Description of what happens in the demo follows:

- (`lyra`) create file `cat.stevens` with content "peace train".

- (`lyra`) sync Google Drive using `rclone`.

[6] https://git.ricketyspace.net/combox/commit/?id=37385a90f90cb9d4dfd13d9d2e3cbcace8011e9e
[7] https://git.ricketyspace.net/combox/commit/?id=9d14db03da5d10d5ab0d7cc76b20e7b1ed5523bf
[8] https://git.ricketyspace.net/combox/commit/?id=422238eb4904de14842221fa09a2b4028801afb1

- (`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`.

4.2.4 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 testing, 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' data store 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' data store on OS X and `rclone` [30] was used to sync files from the Google Drive directory to Google Drive's data store on GNU/Linux, the same USB stick (`ZAPHOD`) was used on both GNU/Linux

and Dropbox to store the third shard (shard2) of the files stored in combox directory.

4.2.4.1 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 an 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.

4.2.4.2 Demo

A demo of combox being used with a USB stick as the third node can be viewed at `https://ricketyspace.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.

4.3 Stress testing

A 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 `424.80MiB` (27 files) to `10,800.00MiB` (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 2015, the `ComboxDirMonitor` was drastically modified to make it use the file Lock shared by the instances of `NodeDirMonitor` [9]. The hypothesis 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 4.3.1 to 4.3.4. Section 4.3.5 gives information about the tools used for stress testing, section 4.3.6 contains the observations and comparisons between this stress test and the one done on `2015-11-08`, and, lastly section 4.3.7 reveals the issues that were found with combox by virtue of doing the stress tests.

4.3.1 flac dump (27 files - 424.80MiB)

4.3.1.1 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 28.33ms.

[9] https://git.ricketyspace.net/combox/commit/?id=5aa1ba0c1dcad62931ba27bb66bf115233086d6c

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	445433187.00 bytes (424.79MiB)
avg. file size	16497525.00 bytes (15.73MiB)
avg. time to split and encrypt a file	352.58 ms

Table 4.1: Stress Testing combox - flac dump (424.79MiB)

4.3.2 20MiB - 90MiB dump (27 files - 1620.00MiB)

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	1698693120.00 bytes (1620.00MiB)
avg. file size	62914560.00 bytes (60.00iB)
avg. time to split and encrypt a file	2670.59ms

Table 4.2: Stress Testing combox - 20MiB - 90MiB dump (1620.00MiB)

4.3.2.1 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 25.52ms.

4.3.3 20MiB - 90MiB dump (99 files - 5940.00MiB)

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	6228541440.00 bytes (5940.00MiB)
avg. file size	62914560.00 bytes (60.00MiB)
avg. time to split and encrypt a file	2979.64ms

Table 4.3: Stress Testing combox - 20MiB - 90MiB dump (5940.00MiB)

4.3.3.1 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 206.20ms.

4.3.4 20MiB - 90MiB dump (180 files - 10800.00MiB)

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	11324620800.00 bytes (10800.00MiB)
avg. file size	62914560.00 bytes (60.00MiB)
avg. time to split and encrypt a file	3423.08ms

Table 4.4: Stress Testing combox - 20MiB - 90MiB dump (10800.00MiB)

4.3.4.1 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 399.87ms.

4.3.5 Tools used

The `dump` script [10] 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 [11]. Lastly `org-mode` was used to document all data gathered during stress testing [12].

4.3.6 Observations

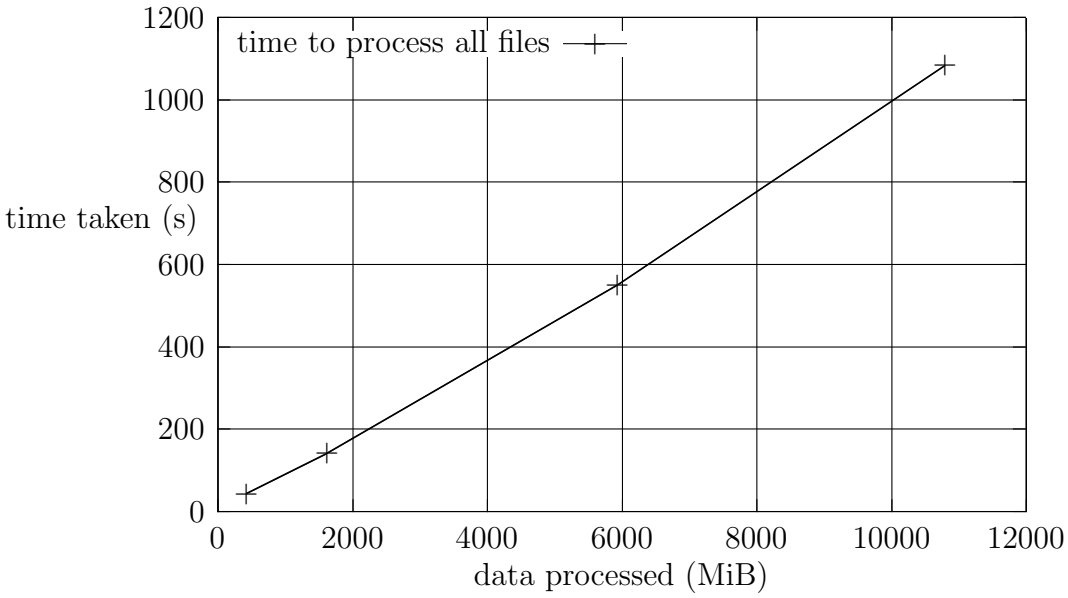

Figure 4-1: Stress testing combox - Observations - Time taken to process all files.

- Fig. 4-1 shows the time it takes combox to process files for a given file dump [13]. As can be observed from the graph, the total time taken to process all the files tends almost linearly increase with the increase in the size of the file dump [14].

[10] https://git.ricketyspace.net/combox-paper/plain/dumper/dump
[11] https://git.ricketyspace.net/combox-paper/plain/scripts/dumps.el
[12] https://git.ricketyspace.net/combox-paper/plain/notes/benchmarks.org
[13] A "file dump" here means a bunch of files copied to the combox directory between 1 sec intervals.
[14] The "size of the file dump" is the total size of all files in a given file dump.

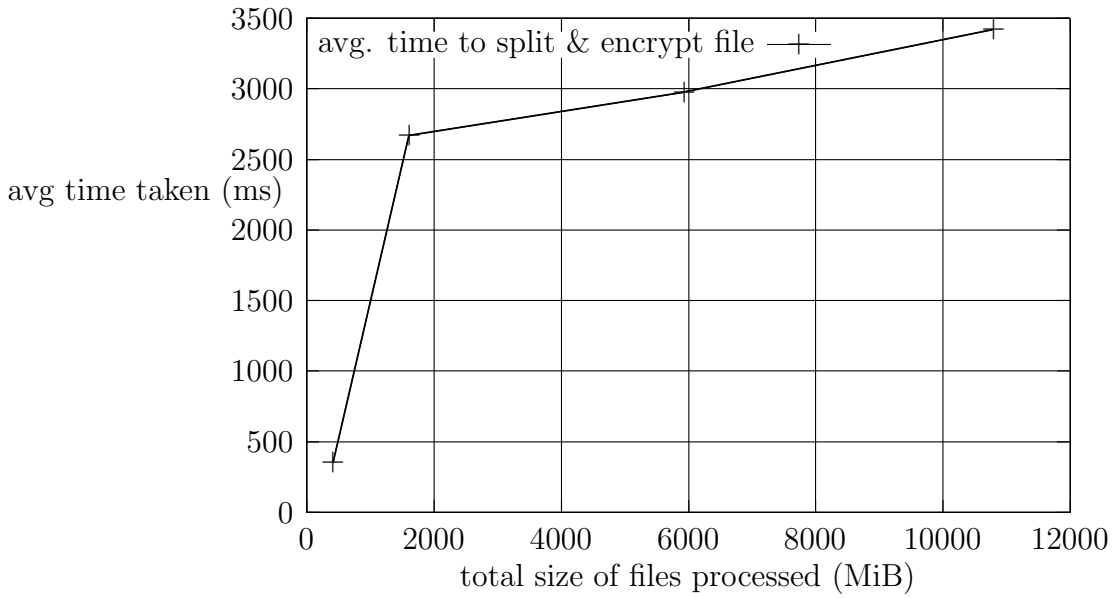

Figure 4-2: Stress testing combox - Observations - Avg. time to split and encrypt a file.

- Fig. 4-2 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 `424.79MiB` dump and the `1620.00MiB` 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 `1620.00MiB` to `10800.00MiB` are the same.

- Fig. 4-3 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 `5940.00MiB` file dump when compared to the `2015` stress test results and it seems to be slightly higher for the `10800.00MiB` file dump when compared to the `2015` stress test.

- Similarly, Fig. 4-4 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 `424.79MiB` and the `1620.00`

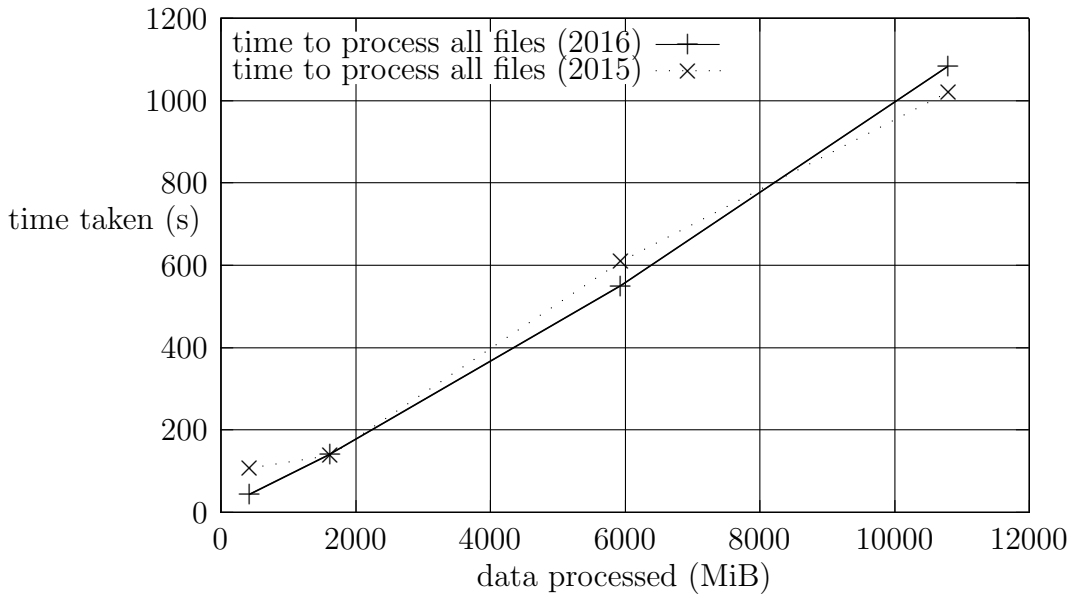

Figure 4-3: Difference between 2015 and 2016 tests - time taken to process all files.

dump, but for the `5940.00MiB` and the `10800.00MiB` dump the average time taken seems to higher for the `2016` stress test when compared to the `2015` stress test.

4.3.7 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` [24] 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 the size of the file was created in the `on_modified` callback methods in both

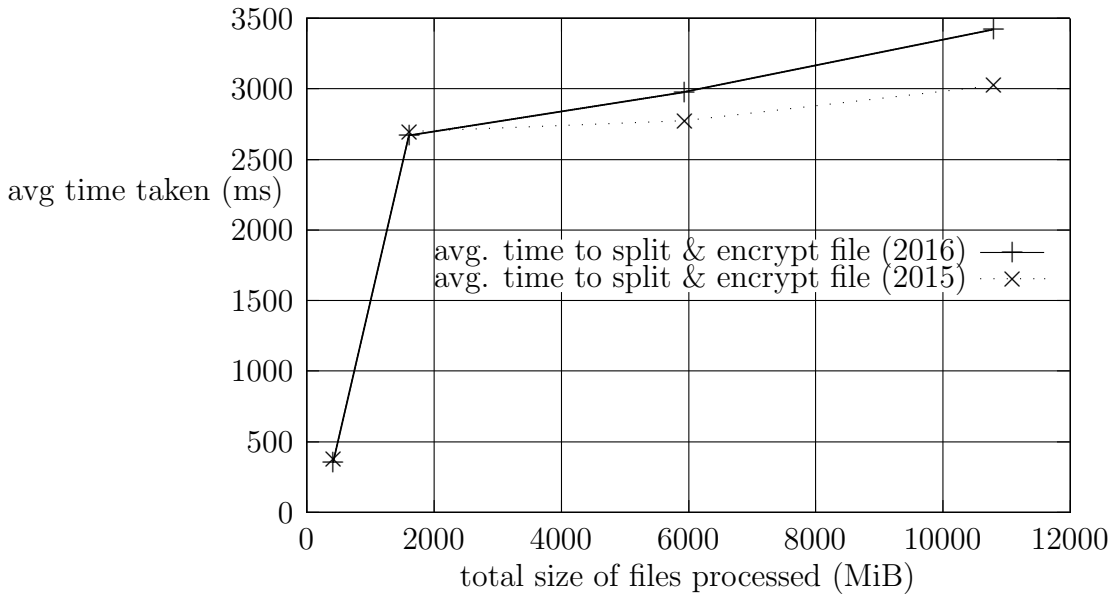

Figure 4-4: Difference between 2015 and 2016 tests - Avg. time to split and encrypt a file.

`ComboxDirMonitor` and `NodeDirMonitor` [15], this fixed the issue.

[15] https://git.ricketyspace.net/combox/commit?id=7ed3c9cbe6e56223b043a23408474f9df08f119e

Chapter 5

Conclusion and Future Work

> In general, I hope to contribute to a world where we value skills and relationships over careers and money, where we know better than to trust cops or politicians, and where we're passionate about building and creating things in a self-motivated and self-directed way.
>
> *Moxie Marlinspike*

combox is at a stage where it can be used as a tool to use the storage provided by two file storage providers – Google Drive and Dropbox – such that only part of each file in the encrypted form is stored on the data store of the file storage providers. This method of storing files on file storage providers makes it difficult, but not impossible, for file storage providers or "third parties" to gain access to the user's personal files.

combox is at version 0.2.3, it is a python package licensed under the GNU General Public License version 3 or later. It is compatible with GNU/Linux and OS X. The program is considered to be in "alpha" stage and must be used for experimental use only. It is not recommended to store critical files on storage provided by file storage providers using combox. Individuals who wish to try combox

would want to look at `https://ricketyspace.net/combox/setup/` to get the program installed on their machines, individuals who want to hack/learn about combox would want to look at `https://ricketyspace.net/combox/api/`. combox's canonical source repository is at `https://git.ricketyspace.net/combox`, the repository is also mirrored at `https://bitbucket.org/bgsucodeloverslab/combox/src` and `http://rsiddharth.ninth.su/git/cb.git/`.

There are a lot of things that can be done to improve combox, and what follows is a non-exhaustive list of things to do in the future:

- Make combox cognizant about space available on each node directory. At the moment, combox reads the amount of free space available on each node directory (file storage provider's directory) when configuring combox on a computer but does not use this information to reckon the space left in each node directory. The major issue here is how to determine what space is available without interacting with a service provider's API or asking the end user.

- Re-think `combox.events` module. This module was written with the assumption that combox will be the only one to make changes to the node directories. This assumption was found to be not true when manually testing combox with node clients (Google Drive and Dropbox client that sync files to/from the respective node directories to/from their respective data stores). Both the Google Drive and the Dropbox client make modifications to the Google Drive and Dropbox directory respectively whenever pulling a modified shard from their data store to the user's computer, this behavior broke combox and major changes were made to the `combox.events` module to make it understand the node client's behavior in the node directory. These changes increased the complexity of the classes defined in the `combox.events`. Tt would be great to re-think this module in such a way that it reduces its complexity.

- Evaluate if more information needs to tracked about each file in the combox directory. At the moment, combox only keeps track of the SHA-256 hash of each file stored in the combox directory.

- Support more file storage providers. For this, ideally no code needs to be written for supporting a new file storage provider, combox must be tested with the new file storage provider's directory as a node directory. If the new file storage provider's client (that sync's the shards their data store) makes non-standard changes to its directory (like the official Dropbox and Google Drive clients do), then the `combox.events.NodeDirMonitor` must be accordingly updated to make combox cognizant about the file storage provider client's non-standard behavior.

- Make unit tests more modular. At the moment, there are some unit test functions that test more than one usecase/facet of a function or class. For instance, the `test_CDM` test method, part of the the `tests.events_test.TestEvents` test class tests the correctness of the `combox.events.ComboxDirMonitor` for file creation, deletion, rename and modification, this method would ideally be broken down into four tests methods.

- Make combox Python 3 compatible. The `2to3` program (which is part of the standard Python library since Python version 2.6) and the `six` library can be used to achieve this. See Appendix A for more information on this.

- Support Microsoft Windows. The way to make combox compatible with Windows will be to run unit tests on Windows. The failing tests might give pointers to what parts of combox needs to be changed/updated in order for it to be compatible with Windows. Individuals interested in making combox compatible with Windows might find `https://ricketyspace.net/combox/setup/#windows`

useful. It contains information about setting up the development environment for combox on Windows.

References

[1] "Wikileaks - spyfiles." [Online]. Available: https://wikileaks.org/spyfiles/

[2] W. Vollmar, "Combox-box," Master's Project, Bowling Green State University, April 2014.

[3] D. E. Knuth, *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[4] "Dropbox privacy policy." [Online]. Available: https://www.dropbox.com/privacy

[5] "pickledb - lightweight and simple key-value store." [Online]. Available: https://pythonhosted.org/pickleDB

[6] "Installshield - proprietary tool for creating package installers for windows." [Online]. Available: http://www.installshield.com/

[7] "pip - pypa recommended tool for installing python packages." [Online]. Available: https://pip.pypa.io/en/stable/

[8] "systemd - system and service manager." [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/

[9] H.-S. Yeo, X.-S. Phang, H.-J. Lee, and H. Lim, "Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices," *Journal of Network and*

Computer Applications, vol. 43, pp. 142 – 156, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804514000897

[10] J. Gonzalez, J. C. Perez, V. J. Sosa-Sosa, L. M. Sanchez, and B. Bergua, "Skycds: A resilient content delivery service based on diversified cloud storage," *Simulation Modelling Practice and Theory*, vol. 54, pp. 64 – 85, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X15000477

[11] "Joey hess." [Online]. Available: https://joeyh.name

[12] H. Weatherspoon and J. D. Kubiatowicz, *Peer-to-Peer Systems: First International Workshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, ch. Erasure Coding Vs. Replication: A Quantitative Comparison, pp. 328–337. [Online]. Available: http://dx.doi.org/10.1007/3-540-45748-8_31

[13] D. Hardt, "The OAuth 2.0 Authorization Framework," RFC 6749 (Proposed Standard), Internet Engineering Task Force, Oct. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6749.txt

[14] B. Kaliski, "PKCS #5: Password-Based Cryptography Specification Version 2.0," RFC 2898 (Informational), Internet Engineering Task Force, Sep. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2898.txt

[15] J. Bian and R. Seker, "Jigdfs: A secure distributed file system," in *Computational Intelligence in Cyber Security, 2009. CICS'09. IEEE Symposium on*. IEEE, 2009, pp. 76–82.

[16] H.-L. Yang and S.-L. Lin, "User continuance intention to use cloud storage service," *Computers in Human Behavior*, vol. 52, pp. 219 – 232, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074756321500446X

[17] "git - the stupid content tracker." [Online]. Available: https://git-scm.com/

[18] "git-annex - how it works." [Online]. Available: https://git-annex.branchable.com/how_it_works/

[19] "git-annex - special remotes." [Online]. Available: https://git-annex.branchable.com/special_remotes/

[20] "git-annex - special remote - amazon s3." [Online]. Available: https://git-annex.branchable.com/tips/using_Amazon_S3/

[21] H. Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*, 2nd ed. MIT Press, 1996.

[22] "Tkinter - python interface to tcl/tk." [Online]. Available: https://docs.python.org/2/library/tkinter.html

[23] "Pyqt - python binding of the cross-platform gui toolkit qt." [Online]. Available: https://riverbankcomputing.com/software/pyqt/intro

[24] "Watchdog - python api library and shell utilities to monitor file system events." [Online]. Available: https://pythonhosted.org/watchdog/

[25] "Pycrypto - the python cryptography toolkit." [Online]. Available: https://www.dlitz.net/software/pycrypto/

[26] "setup combox on windows." [Online]. Available: https://ricketyspace.net/combox/setup#windows

[27] "Python packaging user guide." [Online]. Available: https://packaging.python.org/en/latest/

[28] 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

[29] "Nose - a nicer testing for python." [Online]. Available: https://nose.readthedocs.org/en/latest/

[30] "rclone - command line program to sync files and directories to and from google drive." [Online]. Available: http://rclone.org/

[31] 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.

Appendix A

Making combox Python 3 compatible

> Indeed, when you see new 3.x versions rolling off the line and no one using them, its hard to shake the feeling that Python might die in this transition. How will we ever make it across the chasm?
>
> *Aaron Swartz, March 2012*

What follows is the changes that will have to be made to make combox compatible with Python version 3.x, it was generated by the `2to3` program.

```
--- combox/_version.py  (original)
+++ combox/_version.py  (refactored)
@@ -18,8 +18,8 @@
 #   along with Combox (see COPYING).  If not, see
 #   <http://www.gnu.org/licenses/>.

-__version__ = u"0.2"
-__release__ = u"2"
+__version__ = "0.2"
+__release__ = "2"
```

```
 def get_version():
--- combox/config.py    (original)
+++ combox/config.py    (refactored)
@@ -84,7 +84,7 @@

     prompt = "%s: "  % (prompt)

-    return raw_input(prompt)
+    return input(prompt)


 def config_cb(config_dir = path.join(expanduser("~"), '.combox'),
@@ -143,7 +143,7 @@

     if not path.exists(config_dir):
         # Create combox config directory.
-        os.mkdir(config_dir, 0700)
+        os.mkdir(config_dir, 0o700)

     if not path.exists(config_info['combox_dir']):
         # Create combox directory.
@@ -171,7 +171,7 @@
     """
     nodes = []

-    for node in config['nodes_info'].itervalues():
+    for node in config['nodes_info'].values():
         node_path = path.abspath(node['path'])
         nodes.append(node_path)

--- combox/crypto.py    (original)
+++ combox/crypto.py    (refactored)
@@ -178,7 +178,7 @@
     rel_path = relative_path(fpath, config)

     # no. of shards = no. of nodes.
-    SHARDS = len(config['nodes_info'].keys())
+    SHARDS = len(list(config['nodes_info'].keys()))

     f = path.join(config['combox_dir'], rel_path)
```

```
--- combox/events.py    (original)
+++ combox/events.py    (refactored)
@@ -135,7 +135,7 @@

         # Remove information about files that were deleted.
         fpath_filter = lambda x: x not in self.silo.nodedicts()
-        fpaths = filter(fpath_filter, self.silo.keys())
+        fpaths = list(filter(fpath_filter, list(self.silo.keys())))

         for fpath in fpaths:
             if not path.exists(fpath):
@@ -336,7 +336,7 @@
                 # event; we're tracking this behaviour and ignoring
                 # the 'file modified' event.
                 #
-                if (self.just_created.has_key(event.src_path) and
+                if (event.src_path in self.just_created and
                     self.just_created[event.src_path] and
                     platform.system() == 'Linux'):
                     self.just_created[event.src_path] = False
@@ -476,7 +476,7 @@
         # deleted.
         # Remove information about files that were deleted.
         fpath_filter = lambda x: x not in self.silo.nodedicts()
-        fpaths = filter(fpath_filter, self.silo.keys())
+        fpaths = list(filter(fpath_filter, list(self.silo.keys())))

         for fpath in fpaths:
             del_num = 0
@@ -522,7 +522,7 @@
                         else:
                             files_created[file_cb_path] += 1

-        for f_cb_path, crt_num in files_created.items():
+        for f_cb_path, crt_num in list(files_created.items()):
             if crt_num == self.num_nodes:
                 log_i("%s was created remotely. Creating it locally now..." %
                       f_cb_path)
@@ -663,7 +663,7 @@
                 else:
```

```
                    try:
                        os.renames(src_cb_path, dest_cb_path)
-                   except OSError, e:
+                   except OSError as e:
                        log_e("Jeez, failed to rename path. %r" % e)
                    self.silo.node_rem(silo_node_dict, src_cb_path)

@@ -859,7 +859,7 @@
            # tracking this behaviour and ignoring the 'file modified'
            # event.
            #
-           if (self.just_created.has_key(event.src_path) and
+           if (event.src_path in self.just_created and
                self.just_created[event.src_path] and
                platform.system() == 'Linux'):
                self.just_created[event.src_path] = False
--- combox/file.py      (original)
+++ combox/file.py      (refactored)
@@ -58,7 +58,7 @@

    if directory is None:
        err_msg = "invalid path %s" % p
-       raise ValueError, err_msg
+       raise ValueError(err_msg)

    return p.partition(directory)[2]

@@ -192,7 +192,7 @@
    """
    try:
        os.mkdir(directory)
-   except OSError, e:
+   except OSError as e:
        log_e("Error when trying to make directory %s" % directory)


@@ -227,7 +227,7 @@
        elif path.isdir(fpath):
            purge_dir(fpath)
            os.rmdir(fpath)
-   except OSError, e:
```

```
+    except OSError as e:
         log_e("Error when trying to remove path %s" % fpath)


@@ -253,7 +253,7 @@
         dest_dir_path = path.join(node, dest_rel_path)
         try:
             os.renames(src_dir_path, dest_dir_path)
-        except OSError, e:
+        except OSError as e:
             log_e("Error when trying to rename %s -> %s" % (src_dir_path, dest_dir_path))


@@ -284,7 +284,7 @@
         shard = shard_glob[0]
         try:
             os.remove(shard)
-        except OSError, e:
+        except OSError as e:
             log_e("Error when trying to remove shard %s" % shard)


@@ -325,7 +325,7 @@
                                     shard_no)
         try:
             os.renames(src_shard, dest_shard)
-        except OSError, e:
+        except OSError as e:
             log_e("Error when trying to rename shard %s -> %s" % (src_shard, dest_shard))


--- combox/gui.py       (original)
+++ combox/gui.py       (refactored)
@@ -20,12 +20,12 @@

 import os

-import tkFileDialog
+import tkinter.filedialog

 from os import path
```

```
 from os.path import expanduser

-from Tkinter import *
+from tkinter import *

 from combox.config import config_cb

@@ -165,7 +165,7 @@

         .. _Formatted string: https://docs.python.org/2/library/stdtypes.html#string-formatting
         """
-        print type(args), args
+        print(type(args), args)
         self.status_bar.config(text=format % args)
         self.status_bar.update_idletasks()

@@ -284,14 +284,14 @@
             return False

         # validate node paths
-        for i in xrange(len(self.node_path_entries)):
+        for i in range(len(self.node_path_entries)):
             if not self.node_path_entries[i].get():
                 self.status_bar_set("%s %d", "give the path for node", i)
                 self.node_path_entries[i].focus_set()
                 return False

         # validate node sizes
-        for i in xrange(len(self.node_size_entries)):
+        for i in range(len(self.node_size_entries)):
             if not self.node_size_entries[i].get():
                 self.status_bar_set("%s %d", "give the size of node", i)
                 self.node_size_entries[i].focus_set()
@@ -323,13 +323,13 @@
         config_info = [combox_name, combox_dir, '',  no_nodes]

         # get info about nodes.
-        for i in xrange(len(self.node_path_entries)):
+        for i in range(len(self.node_path_entries)):
             config_info.append("node_%d" % i)
             config_info.append(self.node_path_entries[i].get())
```

```
                config_info.append(self.node_size_entries[i].get())

            config_info_iter = iter(config_info)
-           def_input = lambda(x): next(config_info_iter)
+           def_input = lambda x: next(config_info_iter)
            def_pass = lambda: passp

            config_cb(config_dir=self.config_dir,
@@ -367,7 +367,7 @@
            entry.delete(0, 'end')

            # spawn dialog to choose directory.
-           dir_path = tkFileDialog.askdirectory()
+           dir_path = tkinter.filedialog.askdirectory()
            entry.insert(0, dir_path)


@@ -376,7 +376,7 @@

            """

-           for i in xrange(len(self.node_path_labels)):
+           for i in range(len(self.node_path_labels)):
                self.node_path_labels[i].destroy()
                self.node_path_entries[i].destroy()
                self.node_size_labels[i].destroy()
@@ -416,7 +416,7 @@
                # information" before; get rid of 'em.
                self.clear_node_info_fields()

-           for i in xrange(no_nodes):
+           for i in range(no_nodes):
                node_path_str = 'node %d path' % i
                node_size_str = 'node %d size (in mega bytes)' % i

--- combox/silo.py      (original)
+++ combox/silo.py      (refactored)
@@ -109,7 +109,7 @@
            # instead of PickleDB
            self.reload()
            with self.lock:
```

```
-            return self.db.db.keys()
+            return list(self.db.db.keys())


    def remove(self, filep):
@@ -128,7 +128,7 @@
            self.reload()
            with self.lock:
                return self.db.rem(filep)
-        except KeyError, e:
+        except KeyError as e:
            # means 'filep' not present in db.
            return False

@@ -209,7 +209,7 @@
            try:
                num = self.db.dget(type_, file_)
                num += 1
-            except KeyError, e:
+            except KeyError as e:
                # I don't think this is the right way to do this. :|
                #
                # If we are here it means file_ is not already there,
@@ -252,7 +252,7 @@
        with self.lock:
            try:
                return self.db.dget(type_, file_)
-            except KeyError, e:
+            except KeyError as e:
                # file_ info not there under type_ dict.
                return None

@@ -272,7 +272,7 @@
        with self.lock:
            try:
                return self.db.dpop(type_, file_)
-            except KeyError, e:
+            except KeyError as e:
                # means file_'s info was already removed.
                # do nothing
                pass
```