**SecureCEdit: P2P Secure Document Editing Framework**

By

**Madhuri Shashidhar**

A Project Report submitted to

University at Albany

in partial fulfillment of the requirements  
for the degree of

MASTER OF SCIENCE

in

Computer Science and Engineering

---------------------------------------------------   
 Prof. Pradeep Atrey

---------------------------------------------------   
 Date

**AUTHORIZATION FOR REPRODUCTION  
OF PROJECT**

I grant permission for the reproduction of this project in its entirety, without further authorization from me, on the condition that the person or agency requesting reproduction, absorb the cost and provide proper acknowledgment of authorship.

Date: 15th May 2019 Madhuri Shashidhar

University at Albany, SUNY

1400 Washington Avenue,

Albany, NY 12222

**ABSTRACT**

There are two systems that are becoming popular, Peer-to-Peer systems and Cloud systems. Peer-to-Peer systems (P2P) have become very popular in recent times, because of their use to facilitate file sharing among systems. Another set of systems with wide scale utilization are Cloud Computing systems. These systems allow multiple systems to pool and share their resources for tasks such as data storage, complex calculations, and file sharing. Given the two independent systems of P2P and Cloud, it is interesting to develop a system that combines these two concepts. In this project, the theory and application of Cloud Based Peer-to-Peer Systems is applied to develop an application for collaborative document editing.

Peer-to-Peer(P2P) applications are composed of a collection of peer nodes that cooperate with each other in order to perform some task and share their resources. In the P2P framework, each peer can act as both a provider and consumer of services. A P2P collaborative editing system like Google Docs is a cloud based online system to enable users of real-time collaborative editing on a shared cloud-based document.

A cloud-based system enables services and storage facilities to be provided over the Internet and allows users to access the services and storage facilities through the Internet. It allows users to create, edit, and share their documents without installing a complex software locally. Hence user can expect to save thousands of dollars on both hardware and software. Besides, user also can put more focus on the document editing, regardless of the cost of upgrading for software and hardware.

SecureCEdit is one such system which uses Google Drive for managing the documents between the collaborators. It is a secure system where the information is stored on the cloud and this information is encrypted. The users download these documents to their local machines and the documents are decrypted at the user end. Hence there are no data loss or integrity issues. The information within the document is safe and secure for use by the peers. It allows users to edit the documents simultaneously and does not overwrite each other’s content. Thereby it achieves concurrency between the peers. To do so, it uses an algorithm called Differential Synchronization. This algorithm allows for guaranteed delivery of information between the peers even in case of a network failure.

**TABLE OF CONTENTS**

[1 Introduction 7](#_Toc8297858)

[2 Related Work 9](#_Toc8297859)

[3 Proposed Work 11](#_Toc8297860)

[3.1 Description of Components 11](#_Toc8297861)

[3.2 Differential Synchronization – Guaranteed Delivery Method 12](#_Toc8297862)

[3.3 Algorithm/Pseudocode 15](#_Toc8297863)

[3.4 Application Design 16](#_Toc8297864)

[4 Tool Implementation 20](#_Toc8297865)

[4.1 Assumptions 20](#_Toc8297866)

[4.2 Experimental Setup 20](#_Toc8297867)

[4.3 Analysis 21](#_Toc8297868)

[4.4 Observations 21](#_Toc8297869)

[4.5 Inference 25](#_Toc8297870)

[5 Conclusion 27](#_Toc8297871)

[6 References 28](#_Toc8297872)

**TABLE OF FIGURES**

[Figure 1: Synchronization between a server and multiple clients [4] 13](#_Toc8297873)

[Figure 2: The synchronization process between a peer and a server [4] 15](#_Toc8297874)

[Figure 3: Text Editor 17](#_Toc8297875)

[Figure 4: Workflow of creating, saving and sharing document on Google Drive 18](#_Toc8297876)

[Figure 5: UML Workflow between Peer, Controllers and other Peers 19](#_Toc8297877)

[Figure 6: Communication between peers 21](#_Toc8297878)

[Figure 7: Saving a document to Google Drive 22](#_Toc8297879)

[Figure 8: File saved to Google Drive 23](#_Toc8297880)

[Figure 9: Sharing a document with collaborators 24](#_Toc8297881)

[Figure 10: Opening a file from Google Drive 25](#_Toc8297882)

# Introduction

A P2P application is one where a peer can communicate with peers within the same or a different network and have access to multiple documents. It allows the users to easily and securely access and share documents real time with other users from their devices without a centralized support.

The increased availability of online Internet connections has increased the demand for applications which allow multiple users to collaborate with each other in real-time. An example of such an application is Google Docs. These applications allow the peers to broadcast the changes made to their documents through a subnet. This requires a need for a synchronization algorithm between users.

The application developed here uses Differential Synchronization (DS), which is the simplest synchronization mechanism and it also has a minimal impact on application design. This algorithm can also be incorporated in an already built, existing application.

DS is a state-based optimistic synchronization algorithm where concurrent changes are reconciled by patching the changes from one peer into the copy on another peer. Changes are detected by differencing the current state against the previously established state, yielding a diff. Updates are propagated between peers as diff.

Below are the properties of this algorithm:

• It has the same code on all the peers.

• It is state-based and may not require all the peers to keep track of what change was made when.

• Asynchronous and does not block the peer from editing while waiting for a response over the network.

• Convergent and errors do not cause different copies to diverge.

• Highly scalable across any number of peers.

DS keeps all open connections and versions constantly in sync. The user's keystroke events are mirrored in close to real-time across all the peers. Thus, there is no restriction of any order to be followed by the peers. The peers can edit the documents in any order.

# Related Work

The concept behind how a real-time collaborative editing works is to handle concurrent editing in multi user environment. The main challenge with collaborative editing is to achieve concurrency control between the peers.

There are a few solutions that can be used to simplify this problem. One such solution is Operational Transformation (OT). OT relies on capturing all user actions(keystrokes) and mirroring them across the network to other users. A practical challenge with OT is that all user actions must be captured. User actions such as typing can be handled. But edits such as cut, paste, drag, drop, replacements and autocorrect must also be caught and handled. Since each of these edits changes the location of subsequent edits, if there is an edit that is lost, it may cause the subsequent edits to be applied incorrectly across all the peers and thus increasing the gap between the two versions. This would be an overhead if we had to fix this problem.

Another solution is Three-Way merge. Here all the peers send their version of edits to the server which is the main part of communication between the peers. The server then merges these versions into one single version and sends them back to the clients. The main issue with this method is that, if a peer changes anything while the server is in the process of merging the versions then those changes that the peer made are gone when the server forces its merged version to be used by the peers.

There is a solution that overcomes all these problems. It is the Differential Synchronization (DS) method proposed by Neil Fraser [4]. He provides two versions of the algorithm. The simple one being the Dual Shadow method. But this method does not account to check if the edits have been delivered to the peers and there is no response in turn received from the peers about the delivery of the edits. The other method proposed by him is the Guaranteed Delivery method that ensures there is no loss of edits in between the peers in case of any network failure or package drop. The edits are queued and are sent back to the peers ensuring that all the peers have the same content with them. This method is the only possible solution available now that takes into account all the factors like network failure, duplicate packet, Loss of a packet and many other factors that are common in a communication network.

# Proposed Work

## Description of Components

1. **Peer:** Peers are computers that are used to send messages between each other directly without passing through a centralized server.
2. **Controller:** Controller acts as a server. All the peers are connected to the controller through its IP address and port. It is used to keep the communication active between peers.
3. **Sockets:** Socket is an endpoint in a communication between two computers over a [network](https://searchnetworking.techtarget.com/definition/network). A client socket is used to establish connection with a server socket and is connected to the server IP address and the server port. A server socket accepts all the client connections connected to it through IP address on its port.
4. **Port:** A network port is communication endpoint, identified by port number.
5. **Subnetworks:** Subnet is a logical partition of a network into multiple, smaller networks.
6. **Input/output Streams:** Input Stream is used to read data from a data source. Output Stream is used for writing data to a destination. Here the peers write to the Controllers output stream. The Controller reads from its input stream and writes to the output stream of the other peers.
7. **Google Drive API:** It is used to connect to the Google Drive from the SecureCEdit application to create, open and save documents to Google Drive.
8. **Google OAuth 2.0:** It is used to authorize the SecureCEdit application and the user connected to the application with Google.
9. **Client Text:** The active text editor copy of the peer. As the peer types something to its text editor the Client Text is also updated simultaneously. Client Text may be edited at any time.
10. **Client Shadow:** The previous version of the text editor that is present before a peer changes the text in its text editor.
11. **Diff:** A difference of the Client Text and the Client Shadow. A diff operation makes the Client Text, Client Shadow, Server Shadow and Server Text identical.

For example, if the word "cat" was deleted and replaced with "hag", then technically one could think of it as the replacement of the first and third letters, with the second letter being preserved. This would be the minimal diff.

1. **m:** m is the Client version number.
2. **n:** n is the Server version number.
3. **Patch:** The diff that is sent from the peer to the controller and is applied on the Server Text and Server Shadow. The patches must be applied 'delicately', taking care not to overwrite changes which do not need to be overwritten.
4. **Server Text:** The active copy at the Server. The Server Text is common for all the connected peers and should be equal. Server Text may be edited at any time.
5. **Server Shadow:** The previous version maintained by the server before any patch update made by any client. It is the most up to date copy.
6. **Backup Shadow:** Backup Shadow is updated with the content present in Server Shadow at every patch update made by the peer. It is the previous version for use if the previous transmission was not received.

## Differential Synchronization – Guaranteed Delivery Method

Differential Synchronization (DS) is a symmetrical algorithm with a continuous cycle of difference (diff) and patch operations. It assumes two documents (called Client Text and Server Text). It starts with Client Text and a Client Shadow. The server maintains a Server Shadow and a Backup Shadow for every peer that is connected as shown in Figure 1[4]. The goal is always to keep these two texts as close as possible with each other.

The Guaranteed Delivery method ensures that there is no loss of data in the event of a transitory network failure. Whenever there is network failure, the client might stop synchronizing for a while until the connection times out and the communication ends. If the connection is restored, the shadows will be out of sync with each other. This requires a transmission of the full text to get back in sync. This will destroy all changes since the previous successful synchronization. This method ensures that there is no data loss and the shadows are at sync even in the time of a network failure.

The below diagram shows the connected between a server and multiple clients. The Server Text is common for all connected peers. When Client 1 changes its document, Server Text is updated upon the next synchronization cycle, and those changes are passed on to all other clients on the following cycle.

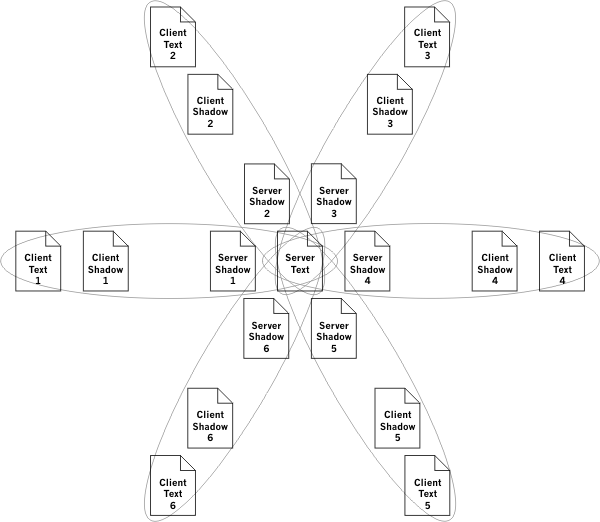


Figure 1: Synchronization between a server and multiple clients [4]

In case of a duplicate packet, the client sends the Edits 'n' to the server twice. The first communication takes place as usual and the response is sent. The server side of client version, n is incremented. The second communication has an 'n' that is smaller than the 'n' recorded on the server side. The server does not process any other edits that it has already processed, so the server does nothing and sends back a normal response.

In case of a lost outbound packet, the client sends Edits 'n' to the server. The server will never receive this. The server will never acknowledge the receipt of the edit. The client leaves the edits in the outbound stack. After the connection times out, the client takes another diff, updates the 'n' again, and sends both sets of edits to the server. The stack of edits transmitted keeps increasing until the server eventually responds with acknowledgment that it got a certain version. [4]

In case of a lost return packet, the client sends Edits 'n' to the server. The server receives it, but the response is lost. The client leaves the edits in the outbound stack. After the connection times out, the client takes another diff, updates the 'n' again, and sends both sets of edits to the server. The server observes that the server version number 'm' which the client is sending does not match the server version number on the server side. But both server and client version numbers do match the Backup Shadow. This indicates that the previous response must have been lost. Therefore, the server deletes its edit stack and copies the Backup Shadow into Shadow Text. The server then throws away the first edit because it already processed. The server applies the second edit, then computes and transmits a fresh diff to the client. [4]

In case of an out of order packet, the server appears to lose a packet, one of the lost packet scenarios is carried out. Then the lost packet arrives, and the duplicate packet scenario is carried out. [4]

In case of a data corruption in memory or network, there are too many potential failures, however if the shadow checksums become out of sync, or one side's version number skips into the future, the system will reinitialize itself. This will result in data loss for one side, but it will never result in an infinite loop of polling.

## Algorithm/Pseudocode

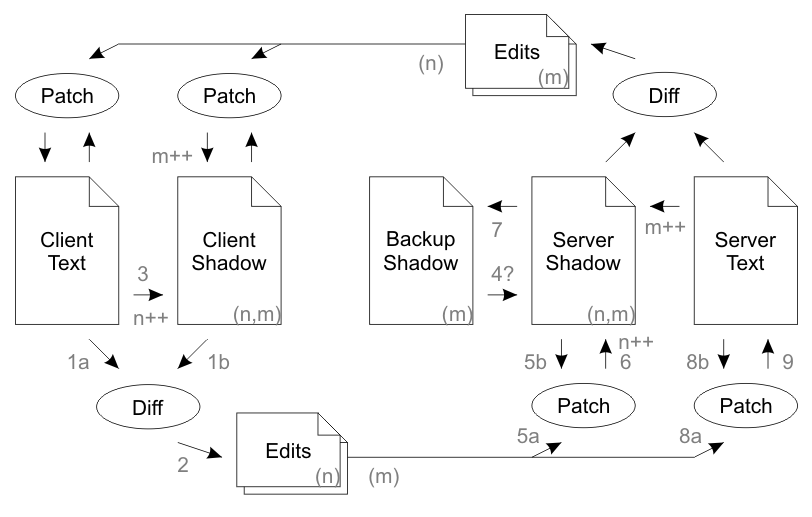


Figure 2: The synchronization process between a peer and a server [4]

Communication from the Client to the Server:

1. Initially the content across Client Text, Client Shadow, Server Shadow, Server Text and Backup Shadow are all same. For instance, “ABC”.
2. The text in text editor is changed by a peer to “ABCD”. The Client Text is updated to reflect this change, “ABCD”.
3. A Diff is computed between Client Text and Client Shadow to obtain a set of edits which were made by the user.

@@ -1,3 +1,4 @@

abc

+d

These edits are tagged with a client version number(n) relating to the version of Client Shadow they were created from.

1. Client Shadow is updated to reflect the current value of Client Text, and the client version number, n is incremented.
2. The edits are sent to the server along with the client's acknowledgment of the current server version number(m) from the previous connection.
3. The version numbers at the server should match both the provided client version number and the provided server version number.
4. The server then patches the edits onto Server Shadow, increments the client version number, n of Server Shadow and takes a backup of Server Shadow into Backup Shadow.
5. Finally, the server then patches the edits onto Server Text.
6. The process then repeats symmetrically from the server to the client, with the exception that the client doesn't take a backup shadow.
7. Any changes which have been made to Client Text in the meantime will be patched around and incorporated into the next synchronization cycle.

Return communication from the Server to the Client:

1. The server will inform the client that it received the edits for version 'n', whereupon the client will delete edits 'n' from the stack of edits to send.

## Application Design

The application is developed as a GUI client with a simple text editor. The text editor has options to create, open documents from Google Drive, save documents to Google Drive, Share the documents with other users on Google Drive. It also has options for cut, copy and paste text. The Figure 3. shows the Text Editor.

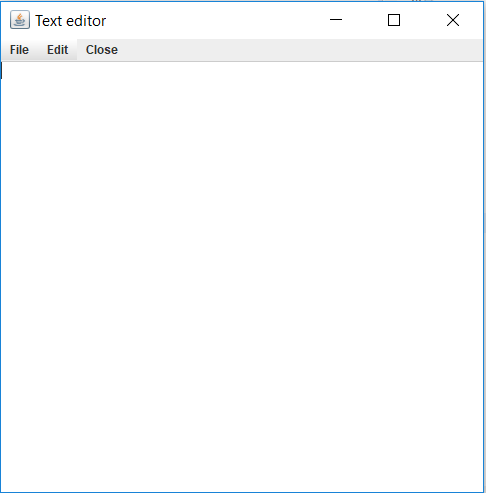


Figure 3: Text Editor

The basic workflow of creating, opening, saving and sharing the document is given in Figure 4. Each document has an owner who creates the document and collaborators who add content to the documents. The owner opens the application and creates the document and saves it on Google Drive through the application by providing a name to the document.

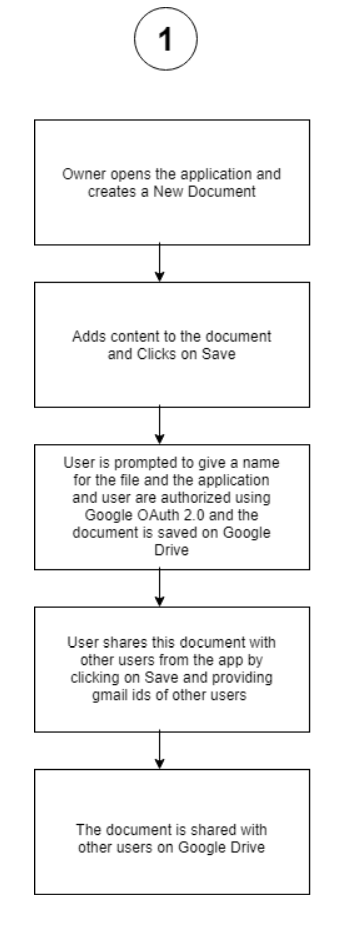


Figure 4: Workflow of creating, saving and sharing document on Google Drive

The owner then shares the document from the application by providing the Gmail IDs of the collaborators suing the Share option from the text editor menu. The document is then shared on Google Drive with other collaborators. The owner and other collaborators then open the document from the Google Drive and start editing.

The below Figure 5. shows the communication flow between the peers and the controller.

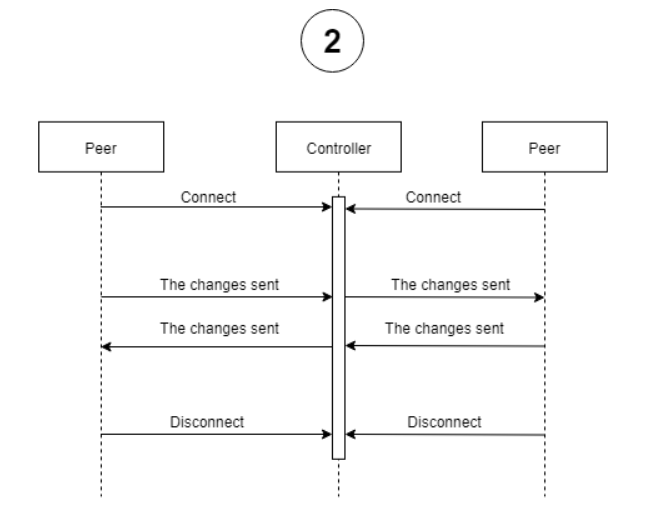


Figure 5: UML Workflow between Peer, Controllers and other Peers

When a peer changes its content on the text editor, these changes are sent to the Controller by writing to the output stream of the controller. The Controller reads from its input streams and writes it to the output streams of the other peers. This is read by the other peers and the content on their respective text editors are updated. When all the peers disconnect, the Controller disconnects.

# Tool Implementation

## Assumptions

This application was built on Windows 10 operating system with the following specifications:

Intel ® Core™ i7-7500 CPU @ 2.70GHz 2.90 GHz with 12GB RAM and 64-bit operating system, x64-based processor.

The application was tested on virtual machines built on the above Windows machine. The specifications of the VMs are as below:

Ubuntu with 2GB RAM, Dual core Processor, 30GB Hard disk.

Eclipse was used to develop the project with Google Drive as a cloud storage. VMs were used to test the P2P functionality of this application. The project was developed in Java with the GUI developed in Java Swing. Google OAuth 2.0 to authenticate the users using SecureCEdit with Google account. Google Drive API is used to connect SecureCEdit with Google Drive and to store, share and download the documents.

We assume that there is only one Controller(server) used in this application and that all the peers are connected through this Controller through the Controllers socket. The application is launched on every peer and is authorized using the peers Gmail ID. Each peer has access to the Google Drive with a valid Gmail ID.

## Experimental Setup

There are 3 peers Peer 1, Peer 2 and Peer 3 that are launched on 3 virtual machines. The Controller is launched first on Peer 1. Peer 1 acts as the document owner and creates a document ‘File’ with the content ‘Text’ in it. Peer 1([madhuri29892@gmail.com](mailto:madhuri29892@gmail.com)) first saves the document on its Google Drive. The document is then shared by Peer 1 through the application to Peer 2 ([poornach7@gmail.com](mailto:poornach7@gmail.com)) and Peer 3([maad.madhuri@gmail.com](mailto:maad.madhuri@gmail.com)). Peer 2 and Peer 3 who act as collaborators open the document once they are launched on their respective virtual machines. All the peers start editing the document.

## Analysis

The peers start editing the document. The text added by Peer 1 is seen on Peer 2 and Peer 3. It happens the same when the other peers edit the document as well.

## Observations

It is seen that there is no overwriting or repetition of text in the text editor of the peers. The text is the same across all the peers. When the text is added at the end or middle or the text is deleted from the end or the middle, the text remains the same across all the peers. The screenshots of the working of the application are given below:

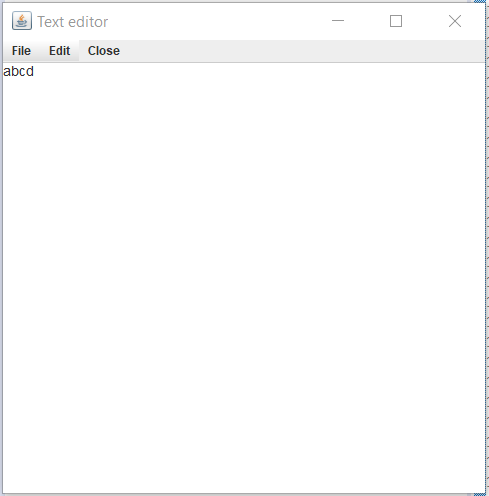
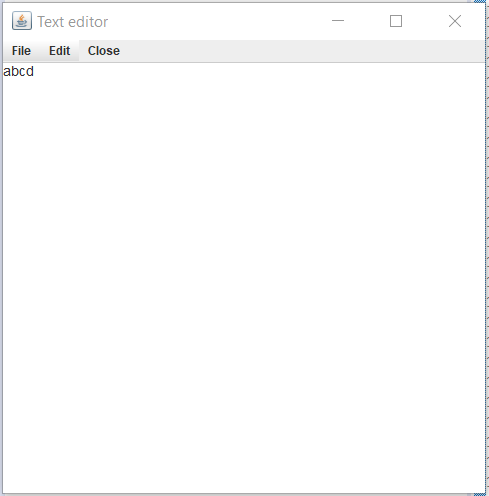
 

Figure 6: Communication between peers

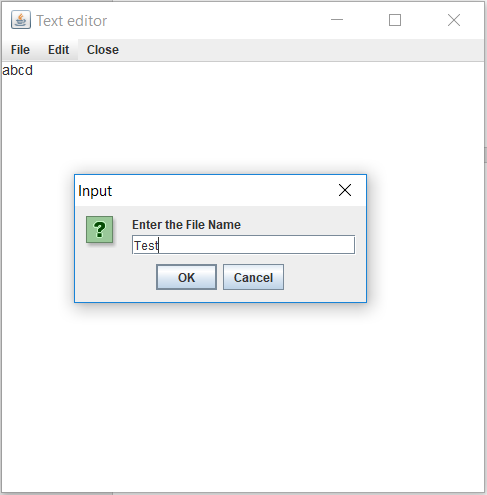


Figure 7: Saving a document to Google Drive

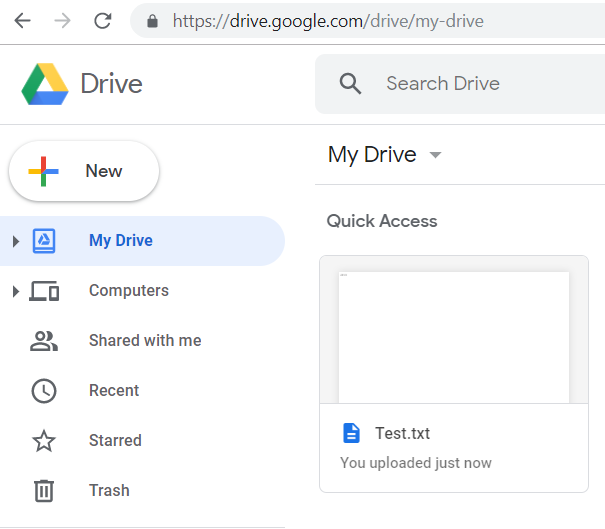


Figure 8: File saved to Google Drive

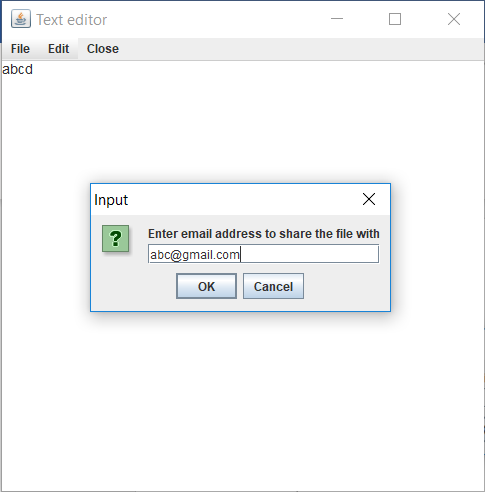


Figure 9: Sharing a document with collaborators

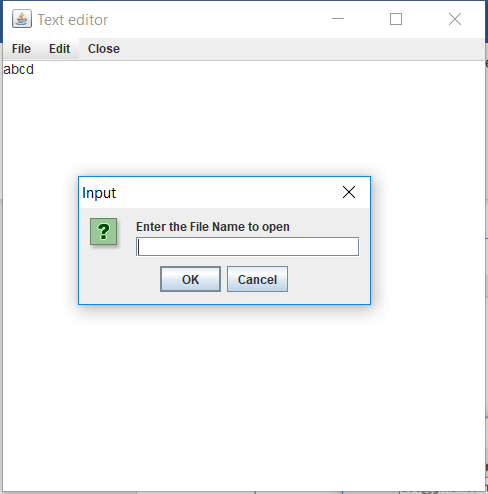


Figure 10: Opening a file from Google Drive

The links to the video demo of the application are below:

Using VMs

<https://www.screencast.com/t/4i3bLQHbOQ>

<https://www.screencast.com/t/88Tl3iCE>

<https://www.screencast.com/t/aClvbTmaomAu>

Using TeamViewer

https://www.screencast.com/t/ESARUifcMAr

https://www.screencast.com/t/5ijo25AF6Ao2

<https://www.screencast.com/t/WW35cMwD>

Video

https://drive.google.com/file/d/1MfbrnZt7u7uNP73RdTEYys1BQ2Xmpi7p/view?usp=sharing

## Inference

This application assumes that the Controller is already running, and all the peers are connected to it. Once all the peers disconnect, the Controller disconnects. This application uses DS algorithm to show that the concurrency can be achieved in a collaborative environment with peers editing and working on the same document. It also shows how Google Drive or cloud storage can be used for storing the documents and sharing the documents between peers.

# Conclusion

Differential Synchronization can be built upon existing application by applying difference and patch algorithms to produce a robust, concurrent collaborative system. The need to detect user keystrokes can be eliminated by using diff and patch operations. This makes the system convergent. The guaranteed delivery method can be used in case of a network failure. It can be used for any number of peers working on a collaborative editing system. The limits of social scalability depend on the nature of the collaborators, the size of the document and the nature of the tasks being performed.

Scalability can become an issue as the number of peers increase. The server may become overloaded when there are so many diff and patch operations. This can be expensive. There are methods for distributing the load on the server. One method is to use a database. A database would handle any number of loads on the server. If the database is identical, the system remains consistent.

Another method is to have a server-to-server topology. The peers can be divided equally between the servers and the servers can be linked to each other in the same way as the peers are connected to the server. More servers can be added whenever the number of peers increase. Servers can also be removed when their respective peers disconnect, and they only have a single connection to another server.

As the number of peers increase, latency can be a problem. If change is made by peer 1 then for the change to get propagated through peer 50, there may be a latency. As latency increases, collisions also increase. A balanced tree can be used as it offers the shortest path between clients, and thus minimizes the latency. Latency may also be reduced by significantly increasing the synchronization frequency between servers.

# References

[1] <https://neil.fraser.name/writing/sync/>

[2] <https://en.wikipedia.org/wiki/Network_socket>

[3] <https://docs.oracle.com/javase/7/docs/api/index.html>

[4] <https://neil.fraser.name/writing/sync/eng047-fraser.pdf>

[5] <https://www.youtube.com/watch?v=S2Hp_1jqpY8>