Real-Time Collaborative Text Editor

Project Report | Distributed Systems

Team 21 - Vasu S., Tanish L., Daksh R., Parv J.
Mentor: Mayank Musaddi

1 Introduction

A collaborative real-time editor is a type of collaborative software or web application which enables real-time collaborative editing, simultaneous editing, or live editing of the same digital document, computer file or cloud-stored data – such as an online spreadsheet, word processing document, database or presentation – at the same time by different users on different computers or mobile devices, with automatic and nearly instantaneous merging of their edits.

Unlike asynchronous (non-real-time, delayed or offline) collaborative editing, such as occurs with revision control systems like Git or Subversion, real-time editing performs automatic, periodic, often nearly instantaneous synchronization of edits of all online users as they edit the document on their own device. This is designed to avoid or minimize edit conflicts. In asynchronous collaborative editing, each user must typically manually submit (publish, push or commit), update (refresh, pull, download or sync) and (if any edit conflicts occur) merge their edits.

Real-time collaborative editing can occur online in web applications such as Microsoft Office on the web (formerly Office Online) – which supports online (web-based) simultaneous editing of Word documents, Excel spreadsheets, PowerPoint and other Microsoft Office documents stored on Office.com, OneDrive or SharePoint cloud storage – or Google Docs and other G Suite productivity (office suite) apps – for online collaborative editing of word processing and other documents stored in Google Drive.

2 Project Details

2.1 Problem Statement

Create a real-time collaborative text editor using WebRTC and CRDTs.

2.2 Background

A text editor is a space where you can insert or delete text characters and then save the resulting text to a file. Each character has a value and a numerical index that determines its position in the document.

In a non-collaborative text editor, where only one user will edit a document, the flow chart for insertion and deletion will be pretty straightforward as shown in Fig. 1. However, in case of a real-time collaborative text editor, multiple users share a document among themselves. Each user will have a different flow chart of insertion and deletion, but the document must reflect the edits made by all the users and that too with consistency.

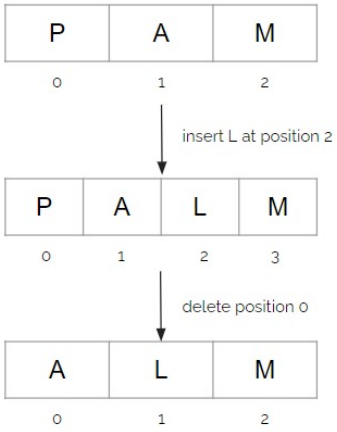

Figure 1: Insertion and Deletion in a Non-Collaborative Text Editor

The complexity of real-time collaborative editing solutions stems from communication latency. In theory, if communication were instantaneous, then creating a real-time collaborative editor would be no more difficult than creating a single-user editor, because a document could be edited using an algorithm similar to the following:

- Request an 'edit document' token from the server
- Wait until the server says it's our turn to edit the document
- Tell the server how to edit the document
- Release the 'edit document' token

However, the speed of communication is limited by network latency. This creates a fundamental dilemma: users need their own edits incorporated into the document instantly, but if they are incorporated instantly, then because of communication latency, their edits must necessarily be inserted into different versions of the document.

An example illustrates this problem. Suppose Bob and Alice start with a document containing the word Mary. Bob deletes 'M', then inserts 'H', to change the word into Hary. Alice, before she receives either edit from Bob, deletes 'r', then deletes 'a', to change it into My. Both Bob and Alice will then receive edits that were applied to versions of the document that never existed on their own machines.

Thus, the challenge of real-time collaborative editing is to figure out exactly how to apply edits from remote users, which were originally created in versions of the document that never existed locally, and which may conflict with the user's own local edits.

3 Conflict-free Replicated Data Type

3.1 Algorithms for Convergence

- Operational Transformation
  - Operational Transformation (OT) is an algorithm that compares concurrent operations and detects if they will cause the documents to not converge. If the answer is yes, the operations are modified (or transformed) before being applied.
  - OT was the first popular way to allow for collaborative editing. Unfortunately, it's extremely tough to actually implement.
- Conflict-free replicated data type
  - A conflict-free replicated data type (CRDT) is a data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies that might come up.

3.2 CRDT Overview

A conflict-free replicated data type (CRDT) is a data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies that might come up.

3.3 Naive Approach for Real-Time Collaborative Editing using CRDT

Each character in the document is given a unique identifier. Usually, this identifier is a float between two fixed numbers, say 0 and 1, coupled with the node id. For eg., (0.1, A), where 0.1 is the identifier and A is the node id. Each character that is added to the document will have a unique float value, the value should be between the value of the previous and next character in the document. With the help of these unique identifiers, the final document is convergence is achieved by ordering the characters using their identifiers. A simple example is shown in Fig. 2

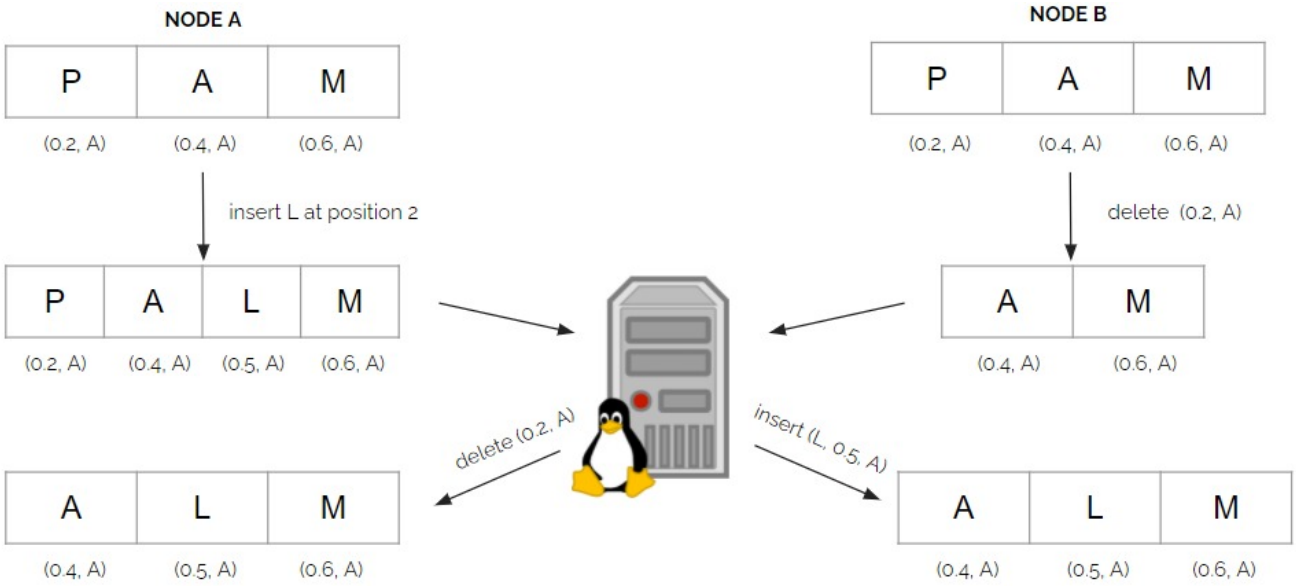

Figure 2: Using unique identifiers for collaborative text editing

However, though the final output is consistent across all the nodes, it may not be what the users desire. The insertions made at different nodes can get **interleaved** with each other. Fig. 3

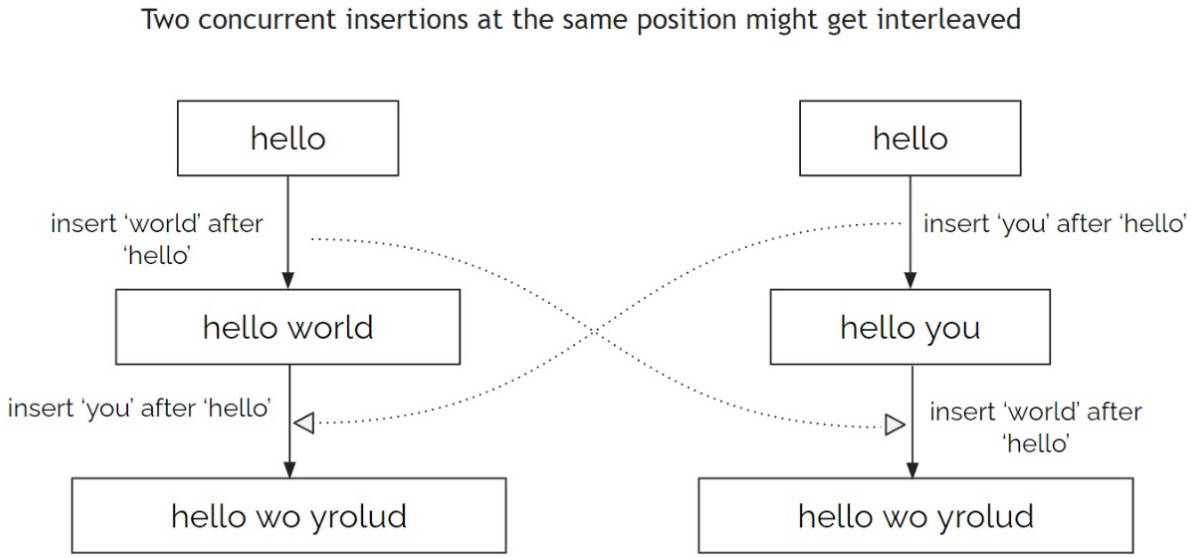

Figure 3: Example of interleaving

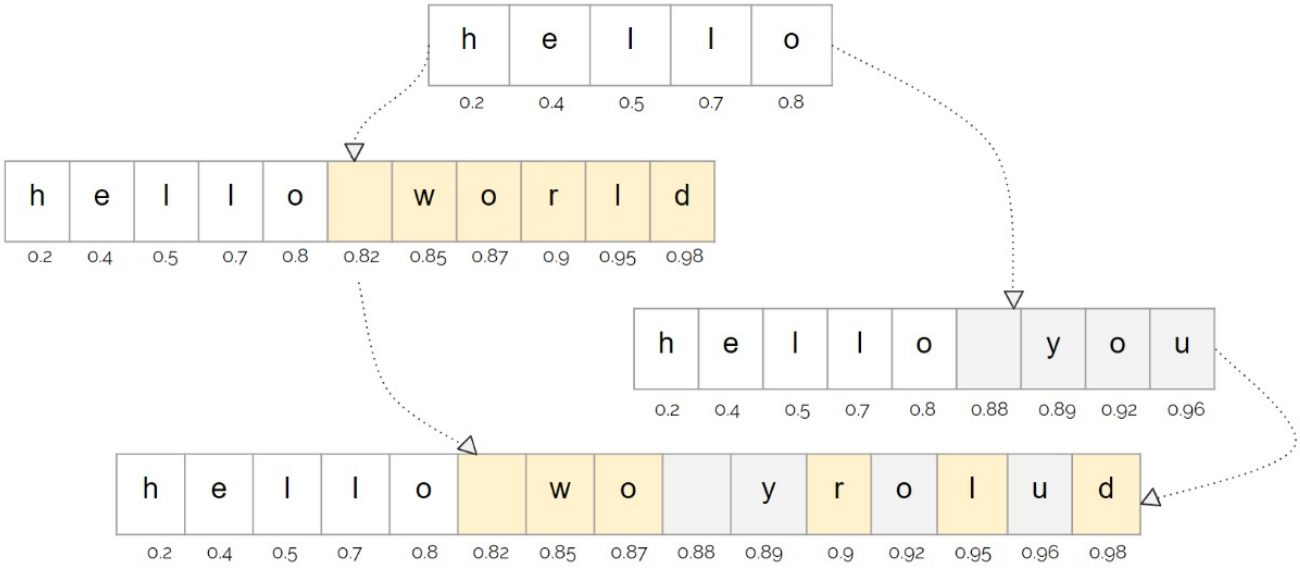

Figure 4: Reason for interleaving

The reason such interleaving happens is because of the way the final result is obtained. The final result is obtained by ordering the unique identifiers. Characters inserted in the document at two different nodes between same two initial characters will have their float values in the same range. And it is very much possible that the range for a new word that was added at node A overlap with the range for a new word that was added at node B. In such a case, the final ordering of the characters of the two newly added words may be a jumbling of their characters. Fig. 4

3.4 RGA

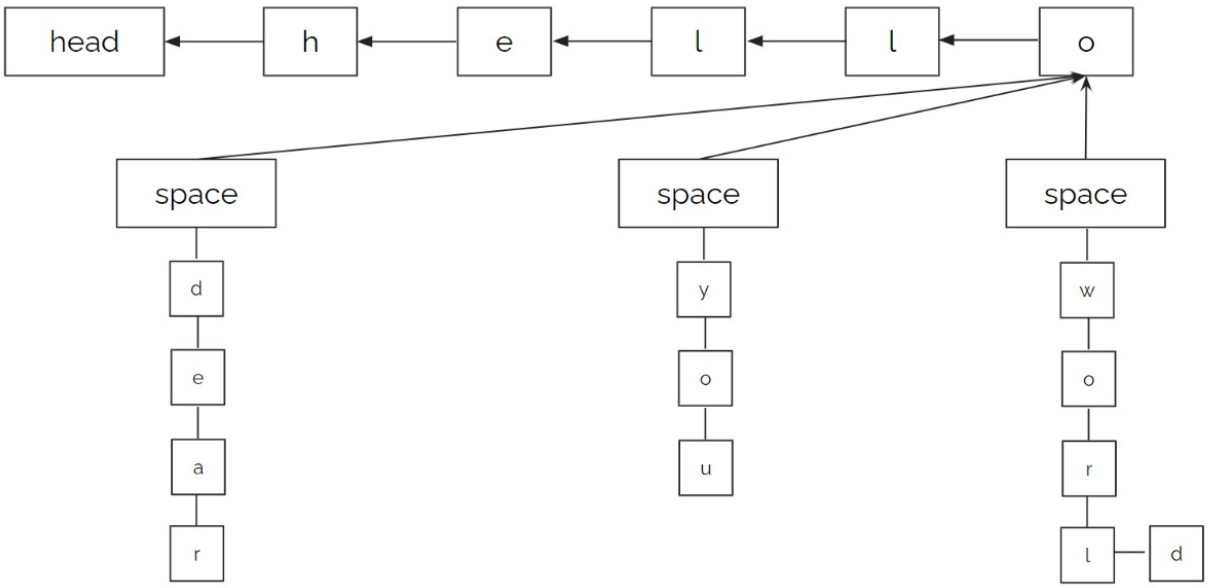

Figure 5: RGA Tree

RGA method overcomes the character interleaving problem we mentioned above. In this method, each new character that is added to the document keeps a pointer to its previous character, sort of like a linked list. And on deletion, the character and its links are deleted and appropriately adjusted in the surrounding characters. What this means is that suppose there is a word "Hello" in the initial document. When node A adds a new character after the word hello, it will point to "o". Similarly, when node B adds a new character after the word hello, it will point to "o" as well. So, we will have a graph as shown in Fig. 5. The final text is formed by doing a depth first traversal of the RGA tree.

So in this case, the possible outcomes of the final text are:

- hello dear you world (Fig. 6)
- hello dear world you
- hello world dear you
- hello world you dear
- hello you dear world
- hello you world dear

So, as we see, the problem of character interleaving is solved by RGA, but there is still the problem of word-level interleaving.

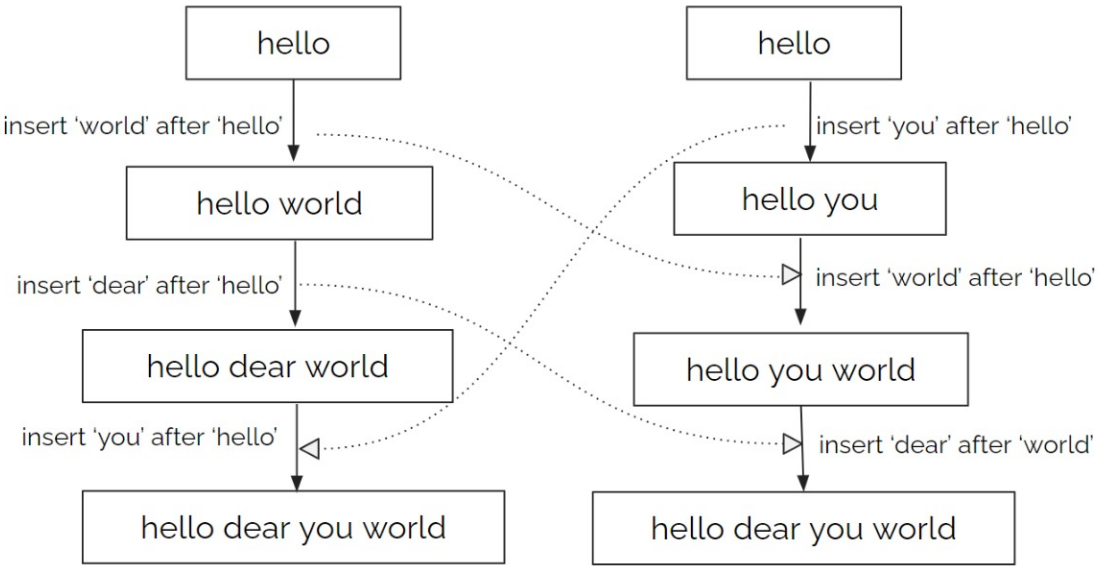

Figure 6: RGA Output

3.5 YATA

YATA is another CRDT algorithm that is used to implement real time collaborative text editors. YATA represents text as a doubly linked list. We define only two types of changes on this representation: insert and delete. When an insertion is deleted, it is just marked as such and not removed from the list (i.e. tombstone approach). Therefore, delete operations do not have an effect on our insert algorithm.

Each insert operation is denoted as $(id, origin, left, right, isDeleted, content)$, where id is a unique identifier given to each operation, $content$ is the content, $isDeleted$ is a flag that marks the content as deleted. $left$ refers to the previous node in the list and $right$ refers to the next node in the list. $origin$ denotes the direct predecessor at creation time. Whenever we want to insert a new character at any place, we space the characters between which it needs to be inserted, which is defined through $left$ and $right$. The intention of an insertion is thus preserved if and only if the insertion is integrated somewhere between $left$ and $right$. YATA assures that it will be placed somewhere between these letters. Convergence is therefore ensured, unless one or more remote operations were already inserted between $left$ and $right$, which then leads to a conflict that needs to be solved.

Take for example the case shown in Figure 7 where the user has already type "HELO" and wants to insert a single character 'L' between the characters 'H' and 'O'. The user intention is thus preserved if 'L' is inserted at any of the three positions present between 'H' and 'L' as shown by dotted lines in the figure. However, to allow different nodes to place the character 'L' at the same position, we want to deterministically decide at what position should the letter 'L' be placed. This is done by giving a strict ordering to each letter present in the text. This ordering is done based on the node structure and this ordering guarantees that every process will place the letter 'L' at exactly the same location between 'H' and 'O'.

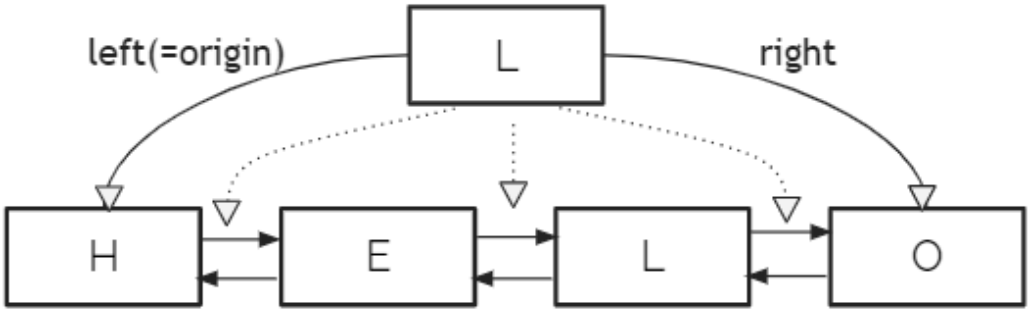

Figure 7: YATA doubly linked structure

CRDT	Local INS	Local DEL	Remote INS	Remote Del
RGA	O(H)	O(H)	O(H)	O(log(H))
YATA	O(log(H))	O(log(H))	O(H^2)	O(log(H))

Table 1: Performance Comparison between RGA and YATA

3.6 Performance comparison

The above table shows a comparison between RGA and YATA algorithm for local and remote insertion and delete operations. Here H denotes the total number of operations that affect a shared document. Since the performance of YATA algorithm is better than RGA algorithm in two out of four fields and worse in only one, we have decided to implement our real-time collaborative text editor using YATA algorithm.

4 WebRTC

4.1 Limitations of a Central Relay Server

The current system architecture relies on the client-server model of communication. It supports multiple users editing a shared document, and between all of our users lies a central server that acts as a relay by forwarding operations to every user in the network of that shared document.

- High Latency between users
  All operations are currently routed through the server, so even if users are sitting right next to each other, they still must communicate with each other through the server.
- Scaling requires additional resources and money
  As the number of users increases, the amount of work the server must do increases accordingly. To support this, the server would require additional resources, which costs money.
- Must trust the central server
  The client-server model requires that users trust the server and anyone that has access to

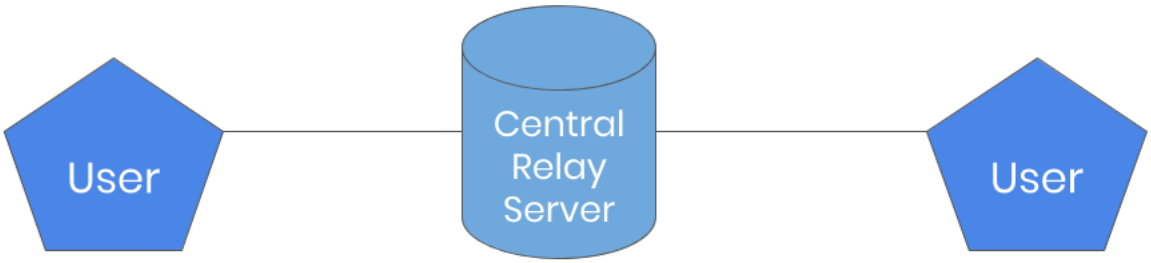

Figure 8: Two users connected through a central relay server

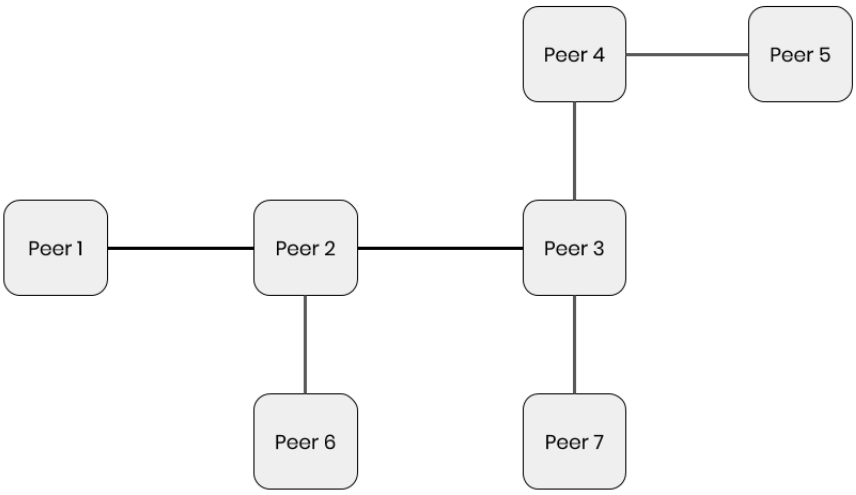

Figure 9: Peer-to-peer message relay network.

it with their data. That includes the application developers (us in this case), our hosting service, and potentially the government.

- Single point-of-failure
  If the server were to go down, all users will immediately lose their ability to collaborate with each other.

4.2 Peer to Peer Architecture

We can remove these limitations by switching to a peer-to-peer architecture where users send operations directly to each other. In a peer-to-peer system, rather than having one server and many clients, each user (or peer) can act as both a client and a server. This means that instead of relying on a server to relay operations, we can have our users perform that work for free (at least in terms of money). In other words, our users will be responsible for relaying operations to other users they're connected to.

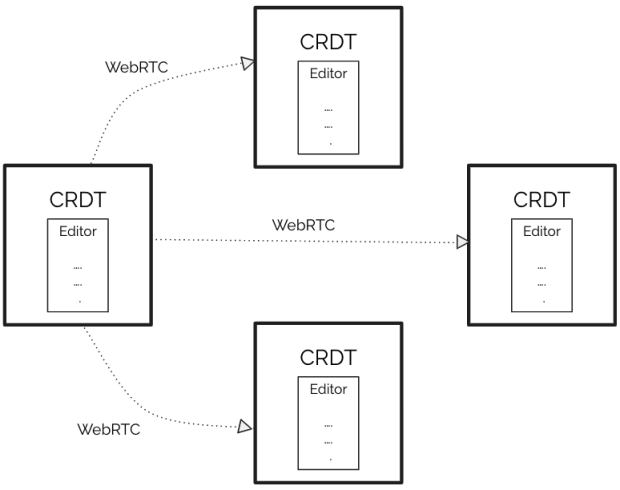

Figure 10: Building them all together

4.3 How will users send messages directly to each other?

To allow nodes to send and receive messages, we used a technology called WebRTC. WebRTC is a protocol that was designed for real-time communication over peer-to-peer connections. It's primarily intended to support plugin-free audio or video calling but its simplicity makes it perfect for us even though we're really just sending text messages.

While WebRTC enables our users to talk directly to one another, a small server is required to initiate those peer-to-peer connections in a process called "signaling".

It's important to mention that while WebRTC relies on this signaling server, no document content will ever travel through it. It's simply used to initiate the connection. Once a connection is established, the server is actually no longer necessary.

4.4 Is WebRTC Secure?

Yes, it is! Encryption is a mandatory feature of WebRTC, and is enforced on all components, including signaling mechanisms. Resultantly, all media streams sent over WebRTC are securely encrypted, enacted through standardised and well-known encryption protocols. The encryption protocol used depends on the channel type; data streams are encrypted using Datagram Transport Layer Security (DTLS) and media streams are encrypted using Secure Real-time Transport Protocol (SRTP).

5 Building them all together

The final system architecture can be represented by the figure 10. Here, CRDT controls every text editor and it makes sure that each editor converges to the same state in the end once no further edits are happening. This is it ensures by using the YATA algorithm as discussed above. Every user must notify every other user about the local operations that particular user is doing and for this purpose we need a peer-to-peer networking system which allows users to pass message to each other. This functionaly is provided by WebRTC.

6 YouTube Link

Click Here