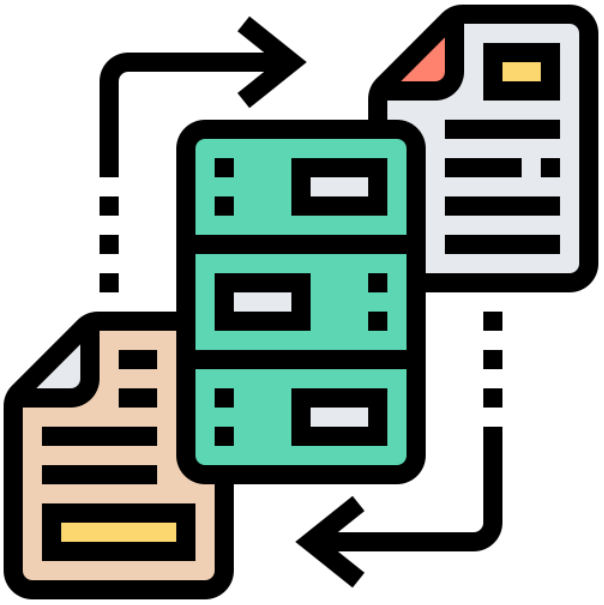
WIRELESS AND MOBILE COMPUTING

Synchronization in Ad-hoc Networks using CRDT



FINAL PROJECT REPORT

Submitted To

RK Yadav
Assistant Professor
CSE Department DTU

Submitted By

Ayush (2K17/CO/086)
Anukriti (2K17/IT/027)

INDEX

MOTIVATION

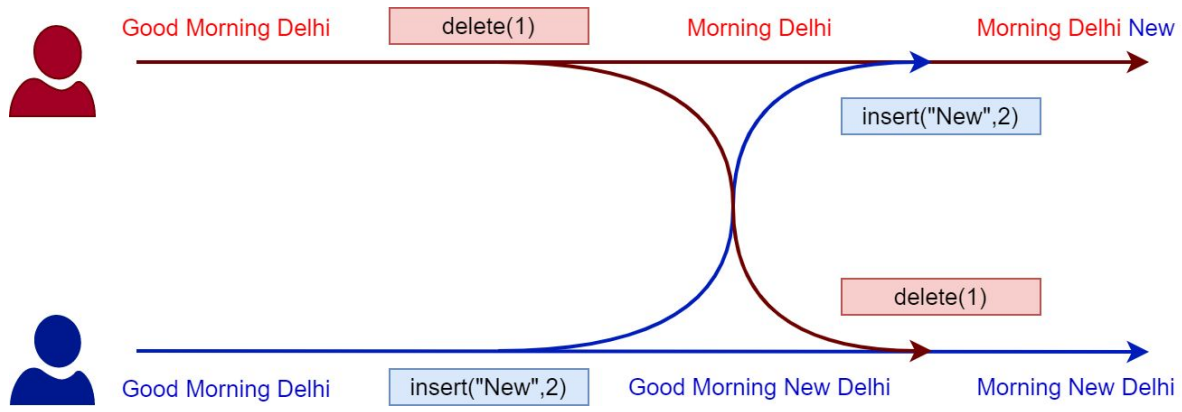
As more and more devices are connected to the internet, communication and coordination between large networks become areas of great interest. However, according to the CAP Theorem, it is impossible to achieve the following three simultaneously:

- **Consistency:** A read is guaranteed to always return the most recent write
- **Availability:** A non-failing node will return a response within a reasonable time limit
- **Partition Tolerance:** When a network partition occurs, the system will continue to function.

Only two of the three properties can be guaranteed at the same time. Such networks resorting to replication techniques often need to sacrifice the consistency of the system in order to attain high-availability.

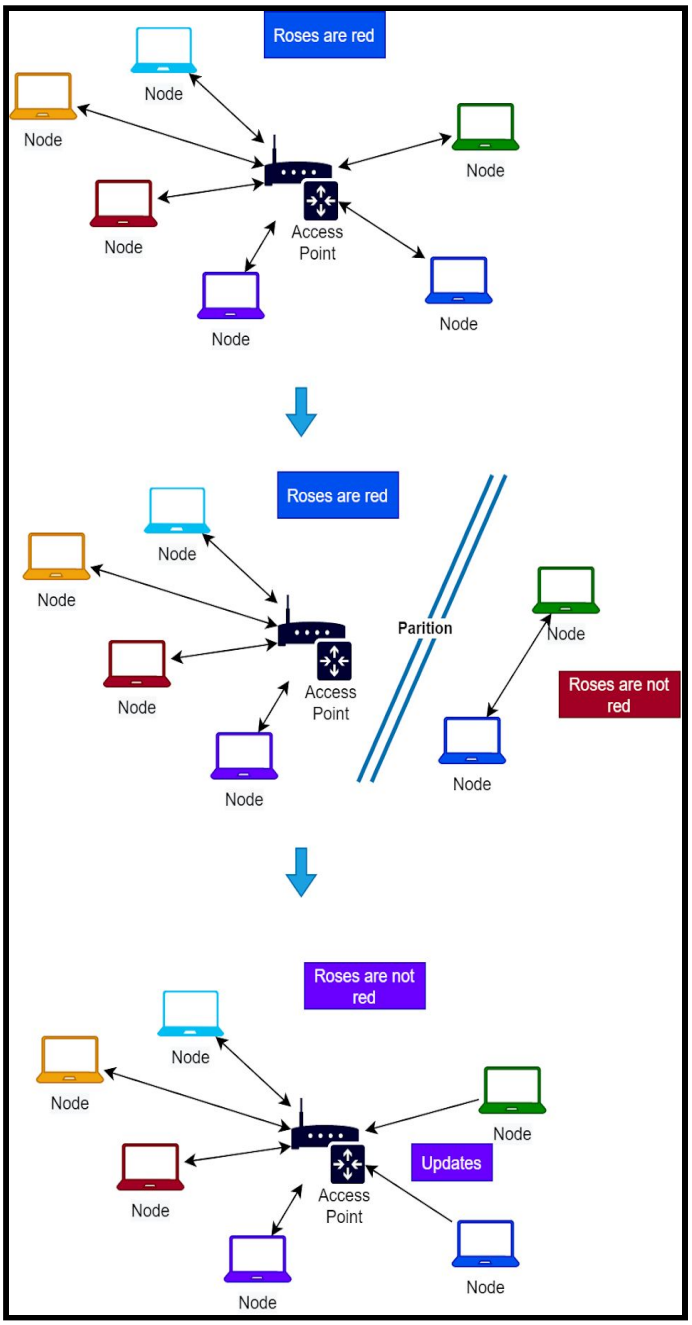
Hence, in this field, Data Replication Strategy has become a fundamental research topic especially in order to achieve fault-tolerance and load distribution. Here, one common approach is to allow replicas of some data type to temporarily diverge, making sure these replicas will eventually converge to the same state in a deterministic way. As we know already that no collaborative editing system can avoid network failures, due to users being connected over the internet. So, availability and partition tolerance must be used. Infact, availability is the key for user experience in a collaborative application, in order for users to work offline and when a user disconnects, the system must continue to function according to the partition tolerance criteria. Keeping this in mind, we decided to propose a collaborative editing solution that will be solving these issues.

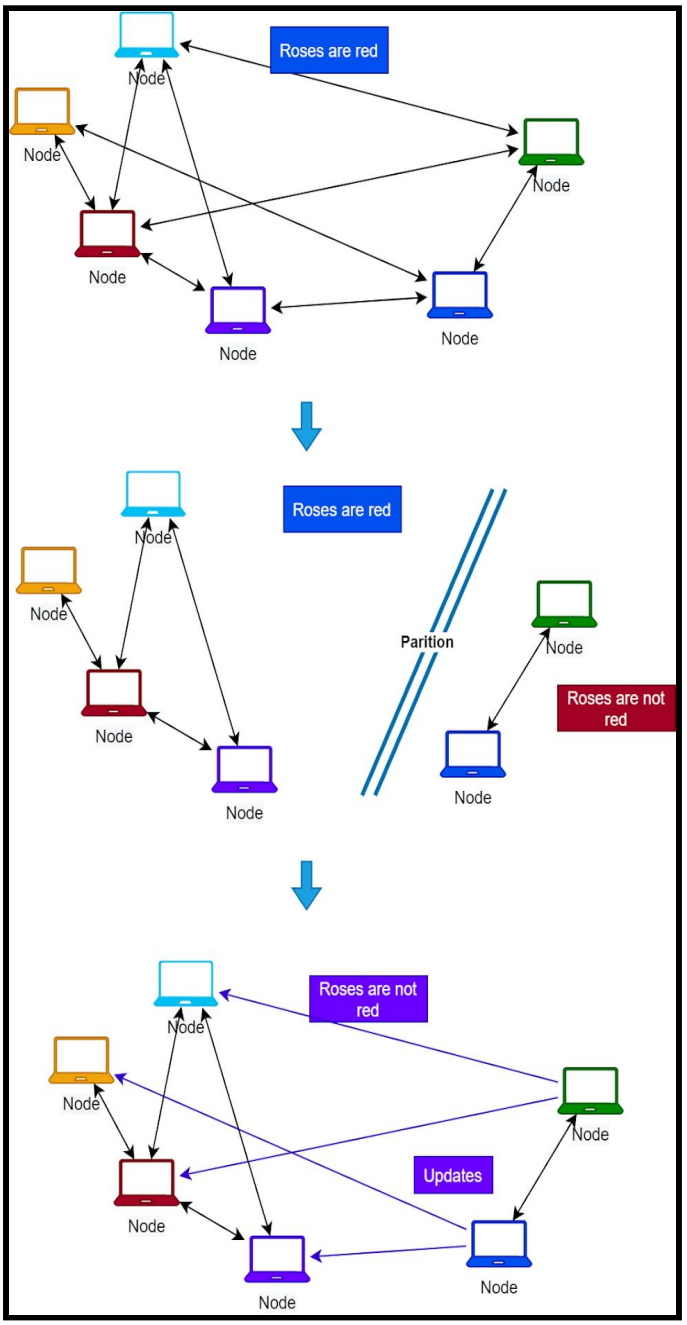
Problem of Synchronization



Infrastructure Network

Ad-hoc Network





Need for Synchronization in Ad-hoc Networks

As shown above, these problems of inconsistency and synchronization do exist and need to be handled carefully, especially in ad-hoc networks which do not have any Access Point to make significant decisions for the nodes in the network. The movement of nodes in such networks results in intermittent connectivity and network partition, which introduces challenges for applications in demand of reliable data synchronization among

multiple parties. For example, in the disaster relief scenario, for instance, a situation after an earthquake, where network infrastructure has been damaged, rescuers working on the field need to synchronize information regarding survivors, supplies, and damage status in order to collaborate in rescue efforts. Hence, they need ways to solve this situation effectively.

Earlier Approaches

Previous work has focused on maintaining a global state of order of operations but we must keep in mind that using a global state of order of operations negatively impacts the performance of an application as internet users can have a high latency, slow bandwidth and unstable access.

Traditional techniques adopting strong consistency give the illusion of a single copy of data by synchronizing replicas on each update. These are clearly not suited for wide-area networks, where the synchronization required between replicas increases the latency of requests, and decreases the system's throughput.

However, recent technological advancements have now made it possible for users to use the internet to effectively communicate and collaborate in a new way. There have been various algorithms circulating that can solve the problem of collaborative editing and these revolve around handling conflicting updates that can occur when users make updates concurrently. One such algorithm used is **Conflict-Free Replicated Data Types (CRDT)**.

Hence, in this project, we will only be focussing only on CRDT algorithms and their implementation.

RELATED WORK

In the context of remote file synchronization, Efficient Algorithms for Sorting and Synchronization, rsync synchronizes two files placed on different machines, by generating file block signatures, and using these signatures to identify the missing blocks on the backup file. In this strategy, there's a trade-off between the size of the blocks to be signed, the number of signatures to be sent, and the size of the blocks to be received: bigger blocks to be signed implies fewer signatures to be sent, but the blocks received (deltas) can be bigger than necessary. Inspired by rsync, XDelta computes a difference between two files, taking advantage of the fact that both files are present. Consequently the cost of sending signatures can be ignored and the produced deltas are optimized.

In Efficient Synchronization of State-Based CRDTs, they have proposed two techniques that can be used to synchronize two state-based CRDTs after a network partition, avoiding bidirectional full state transmission. Similarly to digest-driven synchronization, Δ-CRDTs | Proceedings on the Principles and Practice of Consistency for Distributed Data have provided insights over Δ-CRDTs and how they exchange metadata used to compute a delta that reflects missing updates. In this approach, CRDTs need to be extended to maintain additional metadata for delta derivation, and if this metadata

needs to be garbage collected, the mechanism falls back to standard bidirectional full state transmission.
Related work for the real-time approach adopted in our systems can be found in CoDoc: Multi-mode Collaboration over Documents. Operation-based merging approach used for asynchronous communication is closely related to the works described in Flexible Merging for Asynchronous Collaborative Systems and Using the transformational approach to build a safe and generic data synchronizer. As in FORCE, they use semantic rules on top of syntactic merging. However, their approach is described only for the linear representation of text documents, whereas the hierarchical representation used in our approach yields a set of advantages such as increased efficiency and improvements in the semantics. So far, we have considered text and graphical documents as representative for our research. In Using the transformational approach to build a safe and generic data synchronizer, the authors proposed using the operational transformation approach to define a general algorithm for synchronizing a file system and file contents. The synchronizer proposed in their research automatically finds a solution in the case of conflict.

Still, these solutions require a significant number of message exchanges to identify the source of divergence between the state of two processes. Additionally, these solutions might incur significant processing overhead due to the need of computing hash functions and manipulating complex data structures, such as Merkle trees. With the exception of Xdelta, all these techniques do not assume knowledge prior to synchronization, and thus delay reconciliation, by always exchanging state digests in order to detect state divergence.

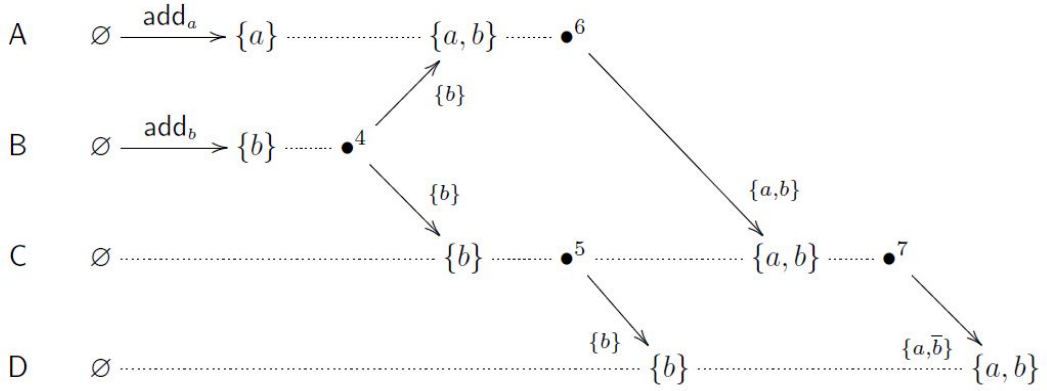
PROPOSED SOLUTION

As we have already noted from the above that in order to ensure high availability, Conflict-free Replicated Data Types (CRDTs) relax consistency by allowing immediate query and update operations at the local replica, with no need for remote synchronization. Earlier, they were of only two types: **State Based** and **Operation Based** CRDT.
State-based CRDTs synchronize replicas by periodically sending their full state to other replicas, which can become extremely costly as the CRDT state grows. Operation-based CRDTs disseminate operations (i.e., small states) assuming an exactly-once reliable dissemination layer. However, when **Delta-State CRDTs** (δ-CRDT) were introduced, they could achieve the best of both worlds: small messages with an incremental nature, as in operation-based CRDTs, disseminated over unreliable communication channels, as in traditional state-based CRDTs.

Algorithm

Although state-based CRDTs can be disseminated over unreliable communication channels, as the state grows, sending the full state becomes unacceptably costly. Delta-based CRDTs address this issue by producing small incremental states (deltas) to be

used in synchronization instead of the full state. They first define delta-mutators that return a delta (δ), typically much smaller than the full state of the replica, to be merged with the local state. The same δ is also added to an outbound δ-buffer, to be periodically propagated to remote replicas. Hence, now let us have a look towards the algorithm associated with them.



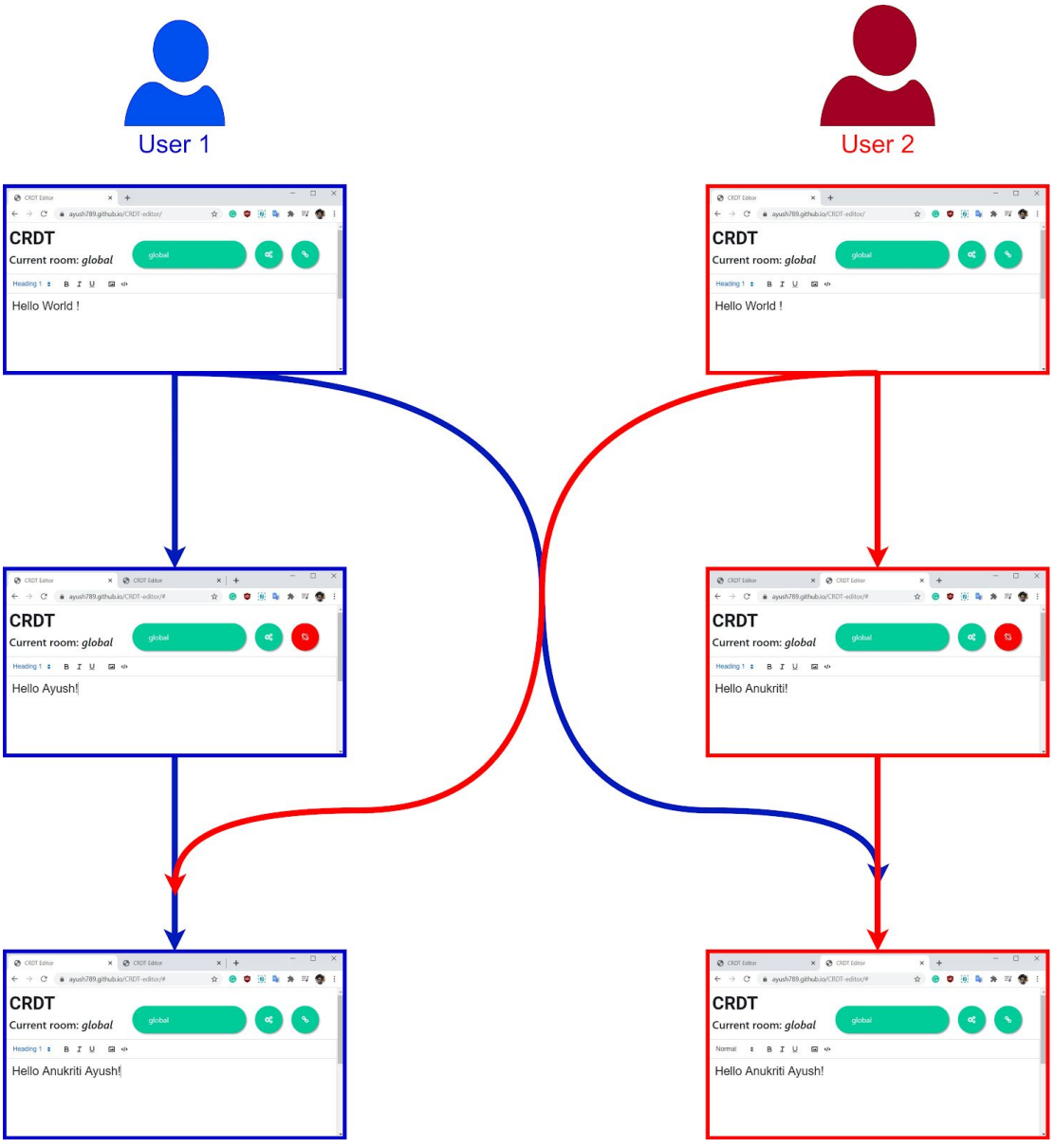

```
1  inputs:
2    n_i ∈ P(I), set of neighbors
3  state:
4    x_i ∈ L, x_i^0 = ⊥
5    B_i ∈ P(L), B_i^0 = ∅                 B_i ∈ P(L × I), B_i^0 = ∅
6  on operation_i(m^δ)
7    δ = m^δ(x_i)
8    store(δ, i)
9  periodically // synchronize
10   for j ∈ n_i
11     d = ⨆ B_i                           d = ⨆{s | ⟨s, o⟩ ∈ B_i ∧ o ≠ j}
12     send_{i,j}(delta, d)
13   B'_i = ∅
14 on receive_{j,i}(delta, d)
15                                         d = Δ(d, x_i)
16   if d ⋤ x_i                            if d ≠ ⊥
17     store(d, j)
18 fun store(s, o)
19   x'_i = x_i ⊔ s
20   B'_i = B_i ∪ {s}                      B'_i = B_i ∪ {⟨s, o⟩}
```

In classic delta-based synchronization, each replica i maintains a lattice state xi ∈ L (line 4), and δ-buffer Bi ∈ P(L) as a set of lattice states (line 5). When an update operation occurs (line 6), the resulting δ is merged with the local state xi (line 19) and added to the buffer (line 20), resorting to function storage. Periodically, the whole content of the δ-buffer (line 11) is propagated to neighbors (line 12). For simplicity of presentation, we assume that communication channels between replicas cannot drop messages (reordering and duplication is considered), and that is why the buffer is cleared after each synchronization step (line 13). This assumption can be removed by simply tagging each entry in the δ-buffer with a unique sequence number, and by exchanging acknowledgements between replicas: once an entry has been acknowledged by every neighbour, it is removed from the δ-buffer. When a δ-group is received (line 14), then it is checked whether it will induce an inflation in the local state (line 16). If this is the case, the δ-group is merged with the local state and added to the buffer (for further propagation), resorting to the same function store. The precondition in line 16 appears to be harmless, but it is in fact, the source of most redundant state propagated in this synchronization algorithm. Detecting inflation is not enough, since almost always there's something new to incorporate. Instead, synchronization algorithms must extract from the received δ-group the lattice state responsible for the inflation, as done by the RR optimization. Few changes are required in order to incorporate this and the BP optimization in the classic algorithm, as we have shown in the highlighted regions of the algorithm.

Implementation

We have built a real-time, collaborative text editor for the browser in Javascript based on the Delta-State algorithm to allow multiple users the ability to edit the same application at the same time from different locations, where the edits are merged into the same state for all users. We have used the YATA algorithm to make sure all users stay in-sync and WebSockets to allow users to send messages directly to one another. The implementation allows edits to be merged, using WebSockets in order to handle synchronization between clients and server. The result is a private and decentralized way to collaborate on documents. The implementation allows users to work offline, and when they are connected to the server the updates will merge into the same state for all connected users. For this, we have used YJS library which operates on Delta-Based CRDT algorithm.

DEMO



Source Code: [Github-CollaborativeTextEditor_CRDT]

Project Demo: [ProjectDemo-CRDT]

CONCLUSION AND FUTURE SCOPE

We have successfully analyzed the algorithm behind delta based CRDT and studied the problem of synchronization used in ad-hoc networks. We also analyzed the previous approaches and work done by researchers in this field earlier. We further implemented a real-time collaborative editing solution using YATA Algorithm and WebSockets.

It should be further investigated how garbage collection can be implemented in CRDTs, since unbounded growth is one of drawbacks of using this data type. CRDTs could become more efficient if the removal of objects could be done immediately instead of keeping removed objects in memory. It should also be investigated how CRDTs can be made more efficient, such that the runtime is lowered once operations are accumulated. Further work should be done regarding the expectations from users who are using a collaborative tool such as the one we implemented in this project.

REFERENCES

A simple approach to building a real-time collaborative text editor

Building a collaborative text editor with WebTC and CRDTs

Efficient Synchronization of State-based CRDTs

Are CRDTs suitable for shared editing?

CRDT Resources • Conflict-free Replicated Data Types

CRDTs: Part 1

yjs/yjs: Peer-to-peer shared types

A novel CRDT-based synchronization method for real-time collaborative CAD systems

Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types