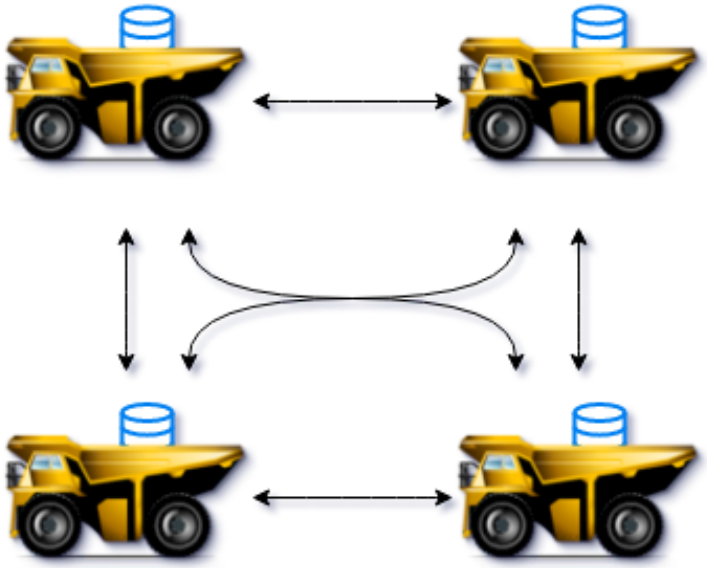


Decentralized Replication Using CRDTs In Systems with Limited Connection and Bandwidth

Master's thesis in Computer Systems and Networks

Fredrik Johansson & André Perzon

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Decentralized Replication Using CRDTs In Systems with Limited Connection and Bandwidth

FREDRIK JOHANSSON & ANDRÉ PERZON

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Decentralized Replication Using CRDTs In Systems with Limited Connection and Bandwidth
Fredrik Johansson
André Perzon

Supervisor: Philippas Tsigas, CSE, Chalmers University of Technology
Advisor: Erik Pihl & Fredrik Thune, CPAC Systems AB
Examiner: Marina Papatriantafilou, CSE, Chalmers University of Technology

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Vehicles synchronizing their databases

Typeset in LaTeX
Gothenburg, Sweden 2018

Decentralized Replication Using CRDTs In Systems with Limited Connection and Bandwidth
FREDRIK JOHANSSON & ANDRÉ PERZON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Distributed storage systems use replication to deal with problems such as fault tolerance, availability of data, lowering latencies and scalability. However, a prominent problem with distributed systems is how to keep replicas consistent during partitioning where ad-hoc solutions have proven to be error prone. For applications that do not require strict consistency, Conflict-Free Replicated Data Types (CRDTs) can be used. CRDTs provide a theoretically sound approach to replicating data under the eventual consistency model. Strict consistency is traded for improved availability, better response time and support for disconnected operations. Simple mathematical properties are used to solve inconsistencies between replicas that may occur due to the eventual consistency.

The State-Based CRDT provides full availability, is commutative, associative and idempotent making it suitable for applications on unreliable networks. The State-CRDT sends its whole state straining the network as the state increase. To deal with the weakness of the State-CRDT, Delta-CRDT was proposed. The Delta-CRDT has the same properties as the State-CRDT but splits states into deltas, limiting the load on the network. The Delta-CRDT is more complex to implement than State-CRDT and is more vulnerable to message loss and requires more communication overhead. This research aims to examine the viability of replicating data by applying CRDTs in the same environment as an existing centralized system. The questions answered in this project are: Can CRDTs be used in a real-life environment with large data sizes and a low number of updates on a low bandwidth network with connection losses? Which type of CRDTs are suitable for this environment? How does the performance of the CRDT systems compare to the existing system? What are the trade offs between the different types of systems? The metrics used to evaluate the systems are operation latency, message latency, time to reach consistency, bytes sent and tolerance to message loss by looking at implemented quality arrays. Testing was conducted on a virtual network with data collected from a real-life scenario in which vehicles operating in a mine synchronize their databases through a centralized system on an unreliable network with low bandwidth.

The results show that the State-Based CRDT system is not viable due to flooding the network with too many and too large messages. The Delta-State CRDT system perform better than the existing system in most metrics and is a viable replacement for the existing system if availability is prioritized.

Keywords: CRDT, Distributed Systems, Centralized, Decentralized, Replication, Strong Eventual Consistency

Acknowledgements

We would like to acknowledge CPAC Systems AB for their support, allowing us to use their facilities, providing us with the data that made it possible to accomplish this thesis and treating us as a part of the group from the very start. A special thanks to Fredrik Thune and Erik Pihl for their time and supervision. We would also like to thank our supervisor at Chalmers, Philippas Tsigas for his valuable guidance and feedback throughout the thesis.

Fredrik Johansson & André Perzon
Gothenburg, June 2019

Contents

List of Figures

List of Tables

List of Source Codes

1
Introduction

Distributed storage systems use replication to deal with problems such as fault tolerance, availability of data, lowering latencies, scalability and load balancing of large number of parallel connections. However, a prominent problem with distributed systems is how to keep replicas consistent as ad-hoc solutions have proven to be error prone [30]. Conflict-Free Replicated Data Types (CRDTs) provide a theoretically sound approach for replicating data under the eventual consistency model. This research aim to examine how different types of CRDTs perform in an environment where connection and bandwidth is limited and how the different trade offs between the CRDT types may correlate to the results.

1.1 Background

Keeping replicas in a distributed storage system both consistent and available to accept new updates is a problem that the CAP-theorem [5] states is impossible when partitioning is present. As partitioning is not a design choice in a system and is not possible to be predicted or avoided, networks can disconnect or devices fail for any reason, it must be taken into account. However, it is possible to mitigate the CAP-theorem through the use of consistency models. The general concept is that stronger consistency lowers availability and higher availability lowers consistency [20]. Depending on the application different consistency models are suitable to find the proper balance between consistency and availability.

CRDTs use eventual consistency which is a consistency model that allows replicas to diverge, i.e., to accept new updates without prior synchronization with other replicas, thus providing high availability [15]. They use simple mathematical properties to solve any inconsistencies that might occur when the replicas perform updates independently of each other. CRDTs were first implemented in practice for example collaborative text editing and peer-to-peer computing and then later formalized by Shapiro et. al. [30], [31]. Both collaborative text editing and peer-to-peer computing value high availability in order to maximize the effective time they can do operations, i.e, minimize the time they need to wait for synchronization.

1.2 Problem Statement

There is a need to replicate data across multiple databases which are located on board vehicles over a network where partitioning is expected to be high due to the environment. When the vehicles are connected to the network the bandwidth and latency will be limited because of them having 3G connection limiting the amount of data the system can send to reach consistency. As the vehicles still are operational during the presence of partitioning there is a demand for high availability in the system even when there is partitioning on the network.

The vehicles are currently using a centralized replication system which does not provide full availability but provides consistency in the form of linearizability. By removing the single point of failure in the centralized system, there is potential to gain better performance in the environment in which the vehicles are deployed in addition to higher availability.

CRDTs were designed to provide consistency in systems where small updates are performed frequently, for example the number of times a song has been played. In the case of keeping track of how many times a song has been played the amount of data sent is small since the state consist of integers. When it comes to replication of databases where each entry has several parameters the amount of data is much larger and updates are subsequently often less frequent. In such a scenario CRDTs should not be preferable, especially if the system is using a network with low bandwidth.

Throughout this thesis real-life system, real-life scenario and real-life environment refers to the scenario described in this problem statement. Real-life is used to differentiate from the virtual systems implemented in this thesis. The questions asked at the beginning of this project were: Can CRDTs be used in a real-life environment with large data sizes and a low number of updates on a low bandwidth network with connection losses? Which type of CRDTs are suitable for this environment? How does the performance of the CRDT systems compare to the existing system? What are the trade offs between the different types of systems?

In order to answer the questions a number of metrics will be compared to measure the performance of each system, these evaluation metrics are explained in Section 1.5. All testing was conducted on a virtual network configured with data collected from the system in the scenario described in this problem statement.

1.3 Aim

The aim of this research is to examine the viability of replicating data by applying CRDTs in the same environment as an already existing replication protocol. For this reason research about the strengths and weaknesses of different CRDTs is done. Based on this research two different CRDT solutions are created, one basic and one more advanced. In addition the performance is recorded in different metrics during

run-time to be able to compare the CRDT solutions with the existing solution.

Drawing from the problem statement the following questions are researched:

- Can CRDTs be used in a real-life environment with large data sizes and low number of updates on a low bandwidth network?
- Which type of CRDTs are suitable for this environment?
- How does the performance of the CRDT systems compare to the existing system?
- What are the trade-offs between the different types of systems?

The results from this thesis are to be used as guidelines when investigating the areas where the CRDTs can be used for replication of data in addition to where they can outperform more common replication protocols as the centralized system with strong consistency.

1.4 Scope

There is no possibility to test the replication protocols on the vehicles directly as they are in production, instead all testing will be made in an emulated environment. To be able to reach the goals of the thesis, data is gathered about the behaviour of the vehicles such as updates performed as well as the quality of the network; bandwidth, latency and connectivity. This is done to be able to simulate an environment that is as close to the real-life scenario as possible to gain an accurate performance validation of the CRDTs.

The performance of the different systems are compared in the following metrics; total bytes sent, message latency, message size, update latency, converge time and quality arrays which displays the messages lost where both the total number of lost messages in addition to the number of consecutive messages lost.

1.5 Performance Evaluation Metrics

Different metrics are used to evaluate the communication times the first is *message latency*, the time in milliseconds it takes from that a connection is established until the entire message has been received. The next metric is the *message size* which is the size in bytes of each message a node send to another node. Finally, the *bytes sent* is the total amount of bytes each node in the system have sent during a test.

To be able to draw conclusions regarding the processing times two different metrics are used. Firstly, the *merge latency* metric is used, the time in milliseconds it takes from a message, consisting of a state, is received until the receiving node has merged

the state with its own state. Secondly, the *message size* is also used to evaluate the processing times as there is a relation between the size of a message (state) and the *message latency*.

Finally, the accuracy of a system is measured by looking at the *converge time*, the time in seconds it takes for the system to reach convergence, from the start of the test. The test start at time 0 but systems can not converge before 300 seconds because the system is still being updated for the first 300 seconds. In addition to *converge time* a metric called *quality arrays* was created to record which messages were dropped by which node and from that information derive how long time nodes are out of sync with the rest of the nodes. The quality arrays can be used to compare how the systems are able to handle different disconnect rate on the network.

1.6 Limitations

Simulations are used instead of real world testing. The cost is simply too high to do real world testing and would most likely produce new challenges that need to be overcome for the specific real world setting thus making it unreasonable for the time frame of this project. The simulated problems the system encounter are loss of connection (partitioning) and low bandwidth (hardware limitation). Detecting byzantine or faulty nodes are not dealt with. All nodes will be considered to work correctly with no other faults than those mentioned above.

The same database structure, that is already present in the centralized solution operating on the vehicles, will be used for all models. The database on the master node is a PostgreSQL database and vehicles' local databases are SQLite databases. Throughout this thesis only SQLite databases are used as using different databases might provide skewed results if their performance is not identical. This also enhances the development time as time is limited. Time will not be spent to try to create optimized queries, efficient sorting etc.

The limitations for this project are:

- Real world testing
  - Will not have access to any vehicles
  - Simulations will be used instead
- Byzantine behaviour nodes
  - No nodes are malicious
  - All nodes send correct messages, e.g., not broken
  - Nodes have robust memory in case of a crash
- Database performance
  - Efficient sorting, queries etc. will not be considered

2
State of the Art

State of the Art covers theory of fundamentals that all designers of distributed systems must consider in order to create a functional system. Section 2.1 presents the definition of a distributed system as well as fallacies regarding distributed system design. The two fundamental types of distributed systems, centralized and decentralized, are explained in Section 2.2 and Section 2.3 respectively.
Section 2.4 explain the CAP-theorem which is an aspect all distributed system designs must consider in order to perform as intended. Linked to this is the different consistency models used in distributed systems, explained in Section 2.5.
Section 2.6 provide the theory behind the Conflict-free Replicated Data Types (CRDTs). They provide strong eventual consistency and are therefore attractive to distributed system designs where availability is important despite the presence of partitioning on the network. CRDTs are the main focus of this thesis and are implemented and tested in two out of three systems.

2.1 Replication in Distributed Systems

The introduction of network communication between different machines in the 60's and the continued technological advances during the next decades set the foundation for the machine-to-machine communication as we know it today [28]. Machine-to-machine communication provided the ability for machines to coordinate their work towards a common task by communicating. This gave birth to a new kind of system, distributed replication systems, or distributed systems for short, are defined as multiple devices connected via a network cooperating to perform some task [26]. Sabu M. Thampi explains one of the many reason why distributed systems have grown with the entrance of the information era, Thampi defines in his paper [25]:

> "When a handful of powerful computers are linked together and communicate with each other, the overall computing power available can be amazingly vast. Such a system can have a higher performance share than a single supercomputer."

The structure of such a system can be constructed in two different ways: centralized, where one node is the master over a number of slave nodes in the network or decentralized where all nodes have the same priority. In a centralized system all communication from the slave nodes are directed towards the master node while in the decentralized system communication is made between arbitrary nodes. The pros

and cons of the two approaches will be discussed in Section 2.2 and Section 2.3.

In distributed systems there exist eight fallacies that distributed system designers often forget to take under consideration and that leads to design flaws that eventually will appear when the system is in production. The fallacies were first mentioned back in 1994 by Peter Deutsch, but they are still relevant for designers today. Arnon Rotem-Gal-Oz summarizes the eight fallacies in his paper [22] and the eight fallacies can be sen in Table 2.1.

Table 2.1: The eight fallacies of distributed systems.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology does not change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Depending on what type of system the designer is looking to create the importance of each fallacy may differ. A closed system on a local network may always have the same topology whilst an application on a network with nodes continuously connecting and disconnecting does not, the topology is changing with each connect/disconnect (breaking fallacy 5). Another thing to point out is that some fallacies are losing their importance for designers. As performance and reliability of hardware increases, failures on a network become less likely [22]. Throughout this project the designs will not take into consideration point 4, 6 and 8 in Table 2.1 because of the limitations mentioned in Section 1.6.

2.2 Replication with Centralized Management

A system with a set of replicas $P_1, P_2, ..., P_n$ where $n-1$ replicas are symmetric but one node is different in terms of possible states and actions is called a centralized system. Figure 2.1 displays a possible network of five replicas where four out of the five replicas are symmetric and the middle replica differs from the other replicas. This is the primary replica and the others are secondary replicas.
Secondary replicas that receive updates do not apply the updates immediately. Instead they notify the primary replica that updates have been received and wait for the main replica to process consequent updates. An example would be when two secondary replicas receive an update at the same time, notify the primary replica at the same time and applies the updates in the same order due to the decision made by the primary replica.

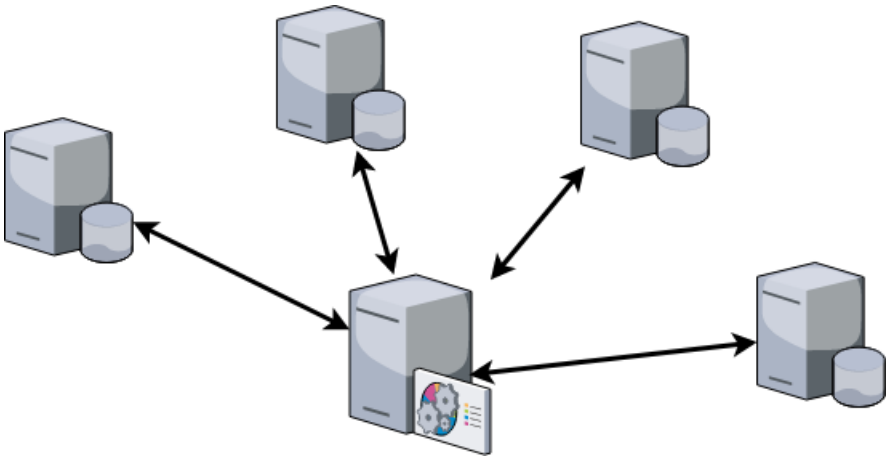


Figure 2.1: An illustration of a centralized replication.

By performing all updates from the secondary nodes on the primary node, the level of consistency is linearizable and sequentially consistent. It is a simple approach for solving the problem of concurrent updates on multiple nodes. There are pros and cons for choosing to build a centralized system, where simplicity is a strong argument for this kind of system [18]. A weakness in a centralized system is the single point of failure, all secondary nodes rely on the primary node for updates. If the primary node never responds with updates to the secondary nodes they can never update and thus lose their availability and consistency. The system is also faced with the possible bottleneck of the primary node not having the performance to handle all requests from the secondary nodes making the whole system not function properly. If this is the case then it does not matter if the secondary nodes might have enough performance to handle their tasks if the primary node is the point of congestion.

2.3 Replication with Distributed Management

A system that has decentralized replication consists of a set U containing N nodes. The nodes can be identical but does not have to be. Each node $k \in U$ has a connection to $m \subseteq U$ nodes as is illustrated in Figure 2.2. The size of subset m for each node is not a fixed size and may change over time.

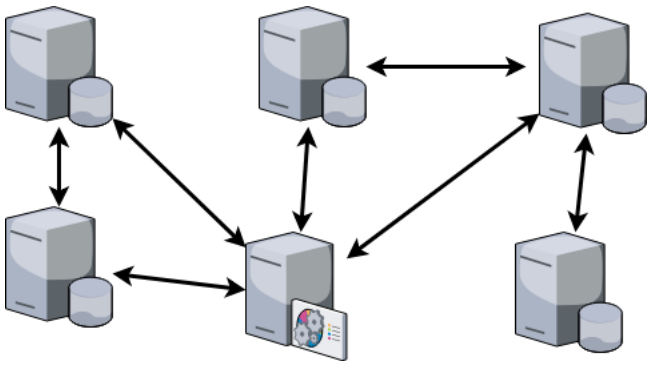


Figure 2.2: Illustration of decentralized replication.

All nodes in the system perform actions to reach a consistent state by communicating with other nodes. Actions are performed locally and the changes of the state are communicated to nodes currently in the subset m. The updates made propagate across the network and reaches all nodes $N \in U$ either directly or indirectly if no node $k \in U$ crash or disconnect indefinitely. In this case one could argue that $U = U - k$.

In comparison to centralized replication decentralized replication has no single point of failure, all nodes can continue working if any node $k \in U$ disconnects. The advantage of no single point of failure leads to higher complexity for keeping consistency when replicating data as all nodes must reach consensus. The complexity of the replication depends on the level of consistency required in the system. Stronger consistency requires more complex solutions than weak consistency.

2.4 Consistency, Availability, Partitioning; The CAP Theorem

A problem designers of distributed systems are faced with is the CAP Theorem [5, 6]. CAP stands for Consistency, Availability and Partitioning. When designing a distributed system you want the system to have strong consistency, always be available and still be operational even when the network is partitioned due to any kind of failure. In the year of 2000, Brewer [5] made a conjecture stating that it is impossible to provide both consistency and availability when the network is experiencing partitioning.

Figure 2.3: Illustration of the CAP theorem, the center is the desired state but unreachable as stated by Brewer.

When a distributed system has geographically long distanced replicas of data and information is sent between the replicas, network partitioning will be present [23]. What the CAP theorem states is that there are two main options when handling a system that is in a partitioned state. One of the options is to block all operations to replicas thus keeping consistency but providing no availability. The other

is to allow all operations to replicas even though it breaks consistency [8]. However, choosing between **C**onsistency and **A**vailability is not exclusive, trade offs between consistency and availability can be made [13], for example reading but not writing may be allowed or certain data might be changed thus keeping some level of relation between consistency and availability. An illustration of Brewers conjecture is illustrated in Figure 2.3 where it is impossible to reach the area where all three circles overlap.

Strong consistency is desirable but costly for the system's performance and it also block operations thus lowering the availability of the system [20]. A bank transfer requires stronger consistency, because of the nature of the transaction, than for example Amazon Dynamo [11] which has sacrificed strong consistency for better availability.

2.5 Consistency Models

There are different consistency models, linearizability (strong), sequential, casual, eventual and weak consistency. The order in which they are mentioned corresponds to the strength of consistency [2]. As mentioned in Section 2.4 which consistency model a system needs depends on the area the system is to be used in. In this thesis eventual consistency is the main focus which is a weak consistency model that can be defined as: *if no more updates are performed, all nodes will eventually converge to the same state within a finite time.*

Systems with weaker consistency models provide the ability to perform updates without the need for consensus and/or synchronization between nodes. This makes weaker consistency models better at operating in low bandwidth networks prone to disconnects because of the message complexity needed for synchronization between nodes [27]. This makes weaker consistency models popular with the growing need for available and scalable services like the Amazon Dynamo [11] and Facebook's Cassandra [21] which both implement eventual consistency.

Applications that use eventual consistency models are able to tolerate an inconsistent state for periods of time. Updates between different replicas occur rarely and instead updates are performed on the local replica to maintain availability for the client. Clients may read old data due to the slow replication of updates as shown in Figure 2.4 where a client deletes (1) the value "cat" from database B, "cat" is then removed (2), the client then performs a read (3) operation on B and finds it is empty. The client reads (4) once again but this time from another replica A, this replica returns the value "cat" as A and B have not yet synchronized (5) their data. The Amazon shopping cart is a famous real-life example of how this issue can become a problem [15]. Where items would reappear in clients shopping carts that had been removed earlier due to receiving "old" data from another replica that had not received the remove update performed by the client on the original replica where the remove was performed.

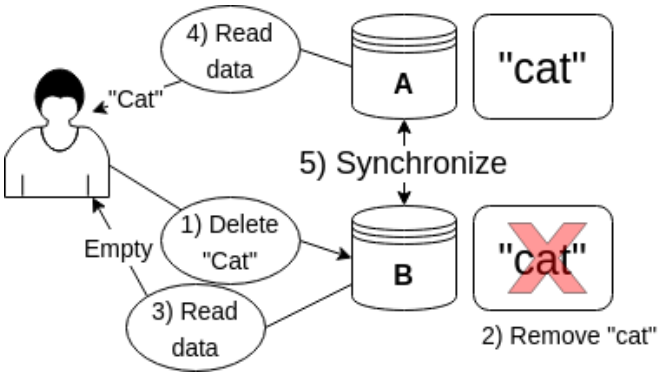


Figure 2.4: Example of an eventual consistent system providing old data to client due to slow propagation of updates.

2.6 Conflict-Free Replicated Data Types

Conflict-Free Replicated Data Types (CRDTs) are distributed data types that provide a theoretically sound approach to enable distributed objects to be eventually consistent and non ad-hoc [31]. Ad-hoc approaches have previously proven to be error prone with the anomalies of the Amazon Shopping Cart [11] as a well known example where removed items in the shopping carts may reappear. Eventual consistency is ensured by CRDTs through the use of simple mathematical properties. As explained in Section 2.5 eventual consistency allow for a process to execute an operation without synchronizing with other replicas beforehand thus providing availability of a replica to clients even in the presence of partitioning.

This thesis will use the same terminology of *atoms* and *objects* as defined by Shapiro et. al. in [31]. They define that an atom is an immutable data type, for example integers and strings, that can be copied by processes. An object is a mutable, replicated data type that has an identity, a content which can be any number of atoms or objects, an initial state and an interface consisting of *operations*. Operations are called by clients to either acquire data from or modify an object. When a client update an object the operation is first called from the interface of the object and then the update is disseminated asynchronously. Two objects with the same identity located at different nodes are *replicas* of one another.

There are two types of CRDTs: State-Based CRDTs (referred to as State-CRDTs in this thesis) and Operation-Based CRDTs [24] with the former being focused on in this thesis. The two types are explained in Section 2.6.1 and 2.6.2. CRDTs is a concept that was first implemented in practice in a cooperative text editor software [14] and then later formalized by Shapiro et. al. [31]. Another practical use of CRDTs is found in latency tolerant computing. Processes in a distributed computing system that use CRDTs can maximize the time they spend on actual computation and lower the amount spent on waiting for synchronization since CRDTs do not use consensus.

The strength of CRDTs, providing full availability by not using consensus, is also a strong limitation of where CRDTs can be used. A trivial case where CRDTs cannot be used is when consensus is imperative for example with transfers of money in a bank. Something as simple as division creates trouble when designing a CRDT as division neither is commutative, associative nor idempotent, properties that a CRDT must fulfill. CRDTs do, despite full availability, provide strong eventual consistency. Strong eventual consistency does not guarantee that atoms in an object constitute coherent information from a human point of view. In the case of collaborative text editing software the CRDT only eventually guarantee that all writers see the same text. It does not guarantee that the text is grammatically correct.

2.6.1 State-CRDT

An operation on a State-Based CRDT object occurs directly on the source and is then disseminated by sending the entire modified state to other replicas. When another replica receives a state U, it merges its own state A with the received state, meaning that the set difference $(U \setminus A)$ is added to A. The merge operation follows the rules of a join-semilattice and is therefore commutative, associative and idempotent which are essential properties since State-CRDTs propagate their states asynchronously.

State-CRDTs are easy to reason about and due to the states that are sent require few guarantees from the network, if a state is not able to be transferred properly it will be sent again without interrupting any other functionality. This is why State-CRDTs can function under extreme partitioning. The simplicity of State-CRDTs comes with the drawback of unbounded growth. Large states have a negative impact on the performance of the network especially when scaling the system. A workaround to this is presented in Section 3.2.4.

A simple example of a State-CRDT, illustrated in Figure 2.5, is the CRDT that solve inconsistencies by using Max integer operation. The base state all replicas start with is an integer with value zero. Each replica may increment the integer an arbitrary number of times. The difference in amount of increments leads to diverging values on the replicas. Every replica sends its state which in this case is a single integer to its neighbours. When a replica receives an integer it persists the larger of the received integer and its local integer. If no further updates are made the replicas will all converge to the same state, i.e. having the same integer value, after a finite number of steps.

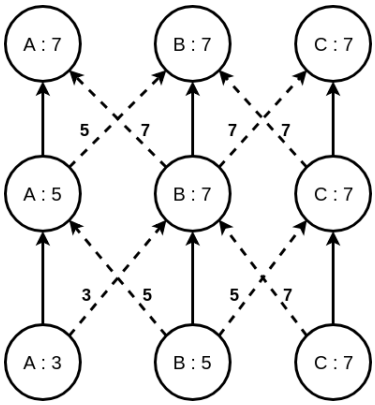


Figure 2.5: Example of integer state with Max merge function converging

2.6.2 Operation-Based CRDT

Operation-Based CRDTs (Op-CRDTs), unlike State-CRDTs, requires *updates* to be delivered on a reliable channel; updates are delivered in the correct order. Op-CRDTs are also able to reason about history, giving them more expressive power while at the same time making them more complex to implement.
The Op-CRDT has three fundamental parts (notation taken from [30]): *payload*, *query* and *update* which is split into two parts *atSource* and *downstream*. The *payload* is an operation that is to be executed asynchronously at other replicas while a *query* is an operation that does not alter the state of the local source replica. The *update* is a function with the first part *atSource* which is executed if its precondition is true after evaluating the arguments it takes. It executes entirely at the local replica atomically and may return calculated results to the caller and/or prepare arguments for the second part, *downstream*. *Downstream* is first triggered if its precondition is true either by arguments from *atSource* or by the arguments contained in a *payload*. When the *downstream* has executed all the operations the state of the replica has been altered and as long as the network delivers all messages in causal order all replicas will eventually converge into the same state.

Since most operations are not associative, commutative and idempotent by themselves reasoning about history must be implemented together with the Op-CRDTs. This is done either by relying on the network delivering messages containing operations at most once and in a causal order or by the replica maintaining a history about which messages have been delivered which would be part of the *downstream's* precondition. Being able to reason about history gives Op-CRDTs more expressive power while at the same time making them more complex to implement.

2.6.3 State-CRDTs with Delta-mutators

Delta-CRDTs, proposed by Almeida, Shoker and Baquerro [4], is a solution to the State-CRDTs' problem of ineffective dissemination as the states get bigger. Delta-CRDTs create delta-states that have smaller message sizes by using *delta-mutators*. Every local update is first run through the delta-mutator which is a function that creates the delta-state, only containing new information, which is then applied to the local replica before being broadcasted to all other nodes over the network.

Delta-states are not idempotent in comparison to the states of State-CRDTs meaning that reasoning about history similar to Op-CRDTs must be taken into consideration. When nodes do updates often they create entropy by sending many small delta-states that are also propagated directly between nodes which may congest the network. A way to limit this entropy is to accumulate messages up to a threshold which once reached trigger the sending of the delta-states. Almeida et al. [4] propose two algorithms for anti-entropy of the delta-states, one basic that sends the whole state periodically and another that uses unique IDs for each delta-state that are garbage collected once all nodes have acknowledged a delta-state.

3
Design and Implementation

In this chapter the designs and implementations of the three systems are presented. This chapter starts by explaining the designs of the different systems. The first Section 3.1 being the Reference system which is a Centrally managed system, as explained in 2.2. Following are the other system designs, Section 3.2, that are decentralized management systems and use CRDTs to reach consistency.

After the CRDT design Section 3.2 continues explaining the design behind the **merge operation** that is a vital part in both of the CRDT systems in Section 3.2.2. The different design variations of the CRDT systems are then further explained in Section 3.2.3 together with 3.2.4.

In Section 3.3, Virtual Test Environment, the network emulator with its functionality and usage is presented. This is the environment which the systems will be deployed to and that in turn has implementation of backend network communication. Section 3.4, Backend Communication, explain different choices of network layer protocols and why sockets running TCP was deemed the best option.

Real-life scenario and vehicles are in this section referring to those mentioned in the Problem Statement in Section 1.2. The real-life scenario is emulated by the virtual network and the vehicles are represented by replicas in the virtual network.

The programming language used was Python. SQLite was used to create and handle databases. Python was chosen because of its excellent support for both Mininet and SQLite.

3.1 Centrally Managed Replication Without CRDTs

The implementation of the Centrally managed system was based on the information received about the system in use in real life when the thesis started. As mentioned in section 3.3 the Centrally managed system have $N-1$ slave nodes and one master node. The master node in the real-life system is using PostgreSQL though in this thesis the master node runs SQLite the same way as the slave nodes. By using the same type of database there was the possibility to reuse the same solutions for all nodes making implementation faster and easier in addition to giving a fairer comparison for the CRDT systems as the speed of the database structures should

not affect the results. A restriction to not focus on database performance was made before the thesis started and the usage of the same database type aligns with this restriction.

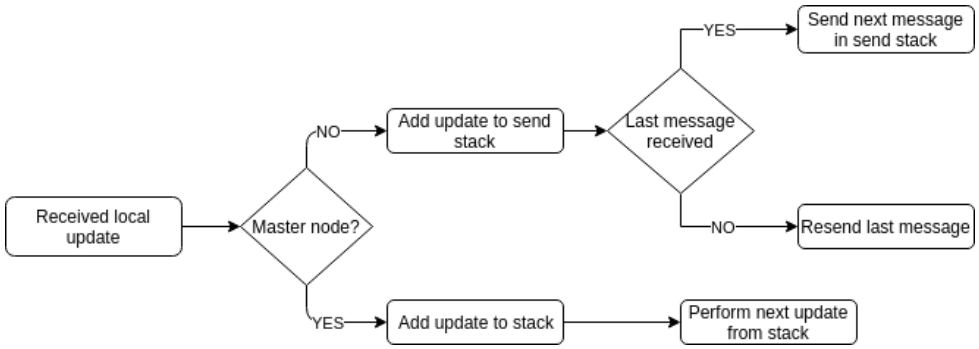


Figure 3.1: Illustration of the system design for the thread that performs all actions.

The simulated nodes can perform three different actions to alter the state of the database, insert, update and delete. All nodes can perform actions but in the Centrally managed system the master node perform actions immediately. The slave nodes on the other hand send the actions they receive to the master node which performs the actions i.e. updates the state, as illustrated in Figure 3.1, the state is then broadcasted to all slave nodes. When slave nodes receive the state they update their own state.

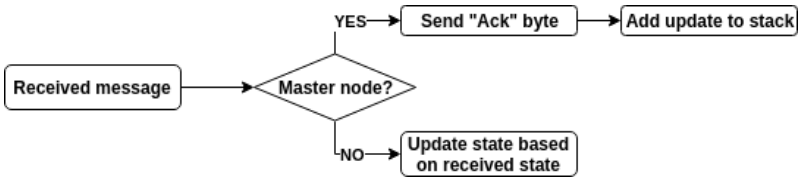


Figure 3.2: Illustration of the system design for the thread that handles connections from other nodes.

The master node repeatedly broadcasts its state to all other nodes. When the slave nodes continuously merge the received states from the master node they stay synchronized. If a node loses connection to the network it can still perform updates even though they will not be seen until the connection to the master node is restored. Figure 3.2 Illustrates the behaviour of the primary and a secondary node when a message is received from another node.

3.2 CRDT System Variations

State-Based CRDT systems were determined to be better suited than Operation-Based CRDT systems in a scenario were connections are unreliable and thus the order in which messages are received cannot be guaranteed. Shapiro et. al. show

that it is possible to emulate Operation-Based CRDTs with State-Based CRDTs and vice versa [30]. This is exactly what would happen if an Operation-Based CRDT was used since the same information the State-Based CRDT sends would have to be sent by the Operation-Based CRDT.

For this reason a primitive State-Based CRDT as well as a more advanced and complex version of State-Based CRDT, Delta-CRDT, have been used with corresponding explanations in 3.2.3 and 3.2.4, respectively. Common aspects of both types are that access and storage of data which is made via the SQLite databases which support the operations described in section 3.5. When a node detects a previously unknown node it creates a copy with the structure (without the data) of its own database and assigns the newly discovered node's ID to the new copy. This means that each node will have a copy of the other nodes' databases where a database depicts a node's state. When a node is merging an atom, a container for the entire state or a delta-state of another node, it only access the database that the atom is to be entered in.

We make the following assumptions which are fundamental for the systems to function:

Assumption 1. *Every atom a_{n_m} can only originate from node N_n*

Assumption 2. *Entries in a state are discarded if they do not follow sequential numbers e.g. update u_{i+1} can only be applied if update u_i exist in the database.*

Assumption 3. *Broken messages are discarded and nodes have robust memory which survive crashes without being corrupted.*

3.2.1 Software Design Layout

The software layout for the State/Delta-CRDT systems can be seen in Figure 3.3. **State/Delta-Topology (Mininet)**, first box, sets up a virtual network with the desired number of nodes and the network configuration supplied. On each node a **State/Delta-Backend**(server), second box, is started. Each **Backend** listens for incoming connections on a certain port, the ability to setup connections to other **Backends** for sending messages as well as **State/Delta-CRDT**, third box, object which provides utility to converge with other nodes. The **State/Delta-CRDT** object has the ability to create new databases, one for each new node, in addition to all other database actions as querying for the current state, merging state and delete. All database actions are made using the DbConnect object, fourth box.
The **State/Delta-CRDT** object has the ability to create new databases, one for each new node, in addition to all other database actions as quering for the current state, mergeing state and delete. All database actions are made using the DbConnect object, fourth box.

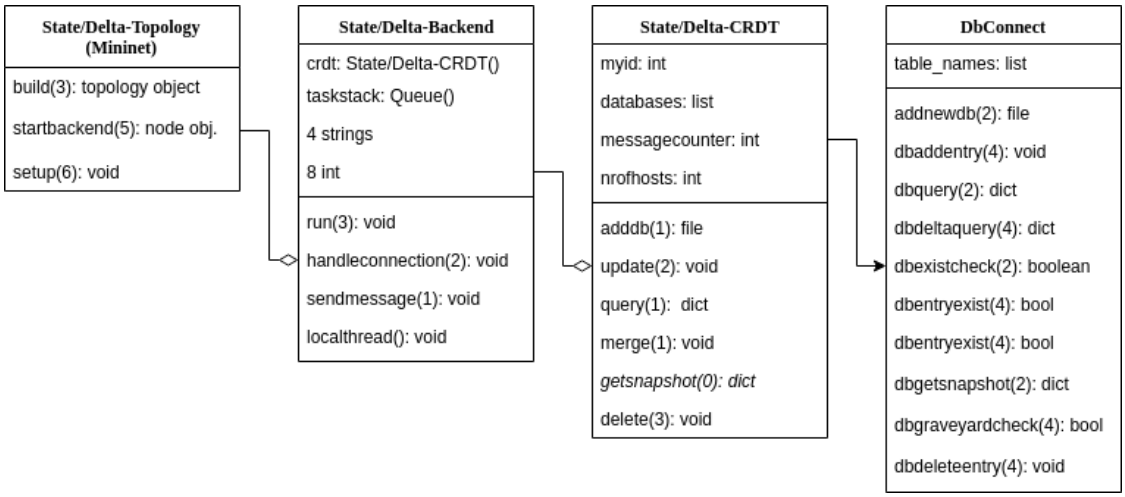


Figure 3.3: System software layout of State/Delta-CRDT systems

3.2.2 Merge Operation

Both State-CRDTs' state and Delta-CRDTs' state S are implemented as two grow-only sets $(Y + X)$, called 2P-sets by Shapiro et. al. [31]. An atom may exist in either Y or X or both. Grow-only set Y contains all atoms and a grow-only set X, also called graveyard, contains deleted atoms. Thus, doing $Y - X$ will show all not deleted atoms. Both sets are represented as tables in the same database.

The CRDT systems use a merge operation to reach convergence across all nodes. Actions preceding the merge operation differ between State-CRDT and Delta-CRDT, explained in Section 3.2.3 and 3.2.4, but they both use the merge operation. The merge operation is using the same principle as the *Union* in set theory.

The set of nodes $A = \{N_1, ..., N_n\}$, where n is the total number of nodes in the network, each node $N_k \forall k \in \{1, .., n\}$ has a state S_k. Each S_k has a set of sub-states $G_m \forall m \in \{1, .., n\}$. The sub-state $G_{m=k}$ is the *local* sub-state of node N_k with the other sub-states $G_{m \neq k}$ being representations of the *local* sub-states from other nodes N_m. The system have reached convergence when $\forall N_k, S_k = S_m \forall m \neq k$.

A node N_{k+1} receiving a state S_k from another node applies the merge operation. The merge operation done by N_{k+1} does the following with the received state S_k; $\forall G \in S_{k+1} = \forall G \in S_k \cup \forall G \in S_{k+1}$. The sub-states are updated with all the updates that differ between the received sub-states from S_k and the current sub-states in S_{k+1}. When N_{k+1} has applied the merge operation it has *at least* the newest information in G_k unless local updates has occurred at N_k during message transit and the processing overhead of the merge operation.

If no more updates occur in the network, no nodes disconnect and the network delivers all messages then all nodes will eventually convergence to the same state.

3.2.3 State-based CRDT - State Machine

The State-Based CRDT that has been used, is primitive in every sense which is visualized through the flowchart in Figure 3.4. Nodes make local updates which are broadcasted periodically, by sending *all* data the nodes possess to all other nodes, whenever their timer reaches 0. When broadcasting no consideration is made for if there's been new local updates, if a state is yet received by another node or if other nodes need to be updated.

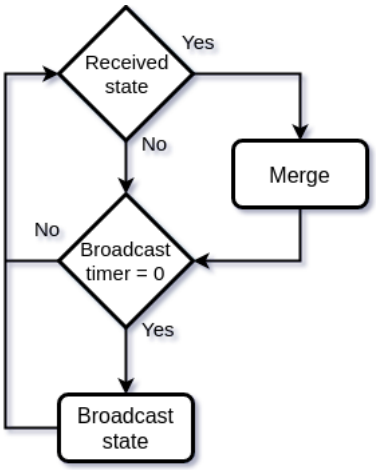


Figure 3.4: Flowchart of State-based CRDT

The State-Based CRDT utilize the merge operation explained in Section 3.2.2 to reach convergence on all nodes. The merge operation become more inefficient the larger the states sent are.

3.2.4 Delta-CRDT with Snapshots - State Machine

The Delta-CRDT designed and implemented in this thesis builds upon the State-based CRDT explained in 3.2.3. The merge operation is exactly the same while the differences that makes it more efficient is found with the state snapshots and calculations of delta-states.

A state snapshot describes the current state of a replica. Each update performed by a node locally is stored in its own database and each update receives an auto-incremented key, an integer. The state snapshot consist of the highest incremented key of each table in each database the node has. Values are mapped to table names which in turn are mapped to their database in a nested dictionary, i.e., nested hash table.

The state snapshot, which is considerably smaller than the full state, is broadcasted to all nodes. Node A that has received a state snapshot from node B creates a state snapshot of its local databases and calculates a delta snapshot $A_{snapshot_i} - B_{snapshot_i} = \delta_{snapshot_i}$ where i is $[1, 2, ..., n]$ in the snapshot dict. $\delta_{snapshot}$ is then used by B to create the delta-state, containing only entries that A does not have, and send it

directly to A. A then applies the merge operation, explained in Section 3.2.2, on the received state.

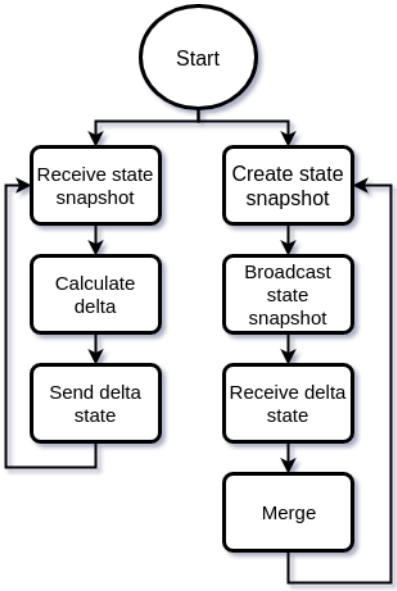


Figure 3.5: Flowchart of Delta-CRDT

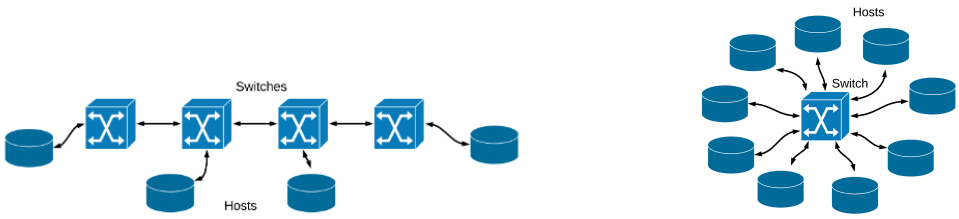
The entropy that Delta-CRDTs create and the subsequent need for garbage collection, explained in Section 2.6.3, is circumvented by the design of the Delta-CRDT in this project, seen in Figure 3.5. This is due to the broadcasting of state snapshots and direct communication of delta-states between nodes, i.e., no states are directly propagated between nodes thus no need for garbage collection. Furthermore, this design keeps the idempotent property of the CRDT which the Delta-CRDT proposed by Almeida, Shoker and Baquerro [4] does not. Reasoning about history is therefore not necessary with this design.

3.3 Virtual Test Environment - Mininet Network Emulator

Mininet is an emulator that can emulate complex network topologies [19]. The emulated network contains hosts, links, switches and controllers which are the building blocks of the network structure. Furthermore Mininet has different connection settings within the virtual network which for this project are used to create limitations in bandwidth and connection in addition to simulate the network partitioning that occurs in the real life scenario. The simulated partitioning allows for testing the systems with different levels of connectivity.

Mininet is designed for testing and development and it provides functions for concurrent development. In the setting of the project there is the possibility of hosts (vehicles) being added to the network dynamically. Mininet is scalable with the number of hosts that can be added to the topology [19].

Mininet has different levels of functionality. A command can be written at the command line to build predefined network topologies within the Mininet software. The different topologies within Mininet are *Tree, Simple, Reversed, Torus* and *Linear*. Mininet has the possibility for the user to build custom topologies by writing python scripts to declare the structure of the desired topology.



(a) *Linear* topology with four hosts and switches.

(b) *Simple* topology with nine hosts and a single switch.

Figure 3.6: Simple and Linear topologies.

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class CustomSingleTopo(Topo):
    def build(self, no_hosts = 4):
        hosts = [self.addHost("host%s" % (h+1)) for h in
                    range(no_hosts)]
        switch = self.addSwitch("Switch1")
        for host in hosts:
            self.addLink(host, switch)
topos = {'customsingletopo': (lambda: CustomSingleTopo())}
singleswitch = CustomSingleTopo(4)
network = Mininet(topo=singleswitch)
network.start()
CLI(network)
network.stop()
```

Source Code 1: Custom created network topology that simulates the *simple* single switch topology seen in Figure 3.6b

When designing a custom topology there are four building blocks that can be used; controllers, switches, hosts and links. The functionality of the building blocks can be customized by the designer in different ways. For links there is the possibility to decide bandwidth, percentage of packets lost, maximum packet queue size and delay in milliseconds stating the time it takes for the packets to go from A to B [19]. On hosts CPU power as well as the number of cores available for hosts to use can be declared. Switches have the option to choose which transfer protocol and what type of switch to be used. In addition to all the options available to choose from,

programming how the building blocks should be connected as well as the amount of each component can all be decided within the script.
Due to not being able to test the systems on the vehicles in the real-life scenario this thesis is based on a custom made topology that is similar to the predefined simple topology that is mentioned in 3.6b in section 2. In the real life scenario the vehicles are not able to talk directly between themselves so they use a cloud service to relay messages for them which is illustrated in Figure 3.8. In the virtual network the cloud is emulated by a switch.

(a) *Tree* topology with a depth of two and a fan-out of four.

(b) *Torus* topology with nine nodes and hosts.

Figure 3.7: Torus and Tree topologies.

In the Centrally managed system all hosts are not equal, in the real-life system there is the central host which can be seen in Figure 3.8 named "Cloud". A custom simple topology was made with $N-1$ slave hosts and one master host in the network with a total of N hosts. The master node represents the central database in this scenario. All hosts have limited processing power and a connection to the switch with limited bandwidth that disconnects on predefined times.

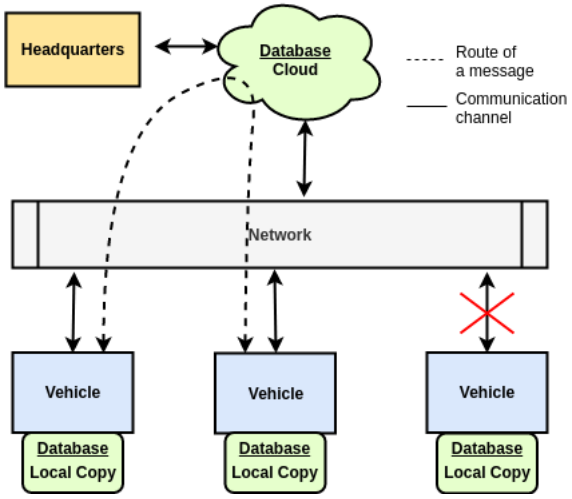


Figure 3.8: System design of the Centrally managed system used as of today.

To achieve this a Python script was written that is made up of two classes, CustomTopo and CustomTopology. CustomTopo extends the Topo class from the Mininet package. In CustomTopo there is a function called build which must exist in the Topo class as it is run when an object of the Topo class is created. In build the number of hosts, switches and links in the network topology is defined. Hardware performance of the hosts like number of cores, how many percent of CPU the host is allowed to use and IP address is also defined in the script. The number of cores was set to two, the IP address of the hosts set to "20.1.90.ID" where ID is the ID number of the host in range of $1, 2, 3, ..., n$.

The CustomTopology class sets up the network and the performance of the network. First an object of CustomTopo is created with the number of desired hosts. Then the performance of the links is defined followed by the switches. Next all different settings for the parts are combined with the addition of other settings like IP base into a network object by using the Mininet class. This object represents the network and it can be started and shutdown by built in functions.

Once the network is up and running individual hosts (nodes) can be selected in the network and run programs on them. This is used to run the script containing one of the different backend solutions that will be compared in this thesis. Once the hosts are running their backends there is the possibility to start a simple command-line interface for the network with the CLI() function within Mininet. The command-line interface enable the possibility to break link connections between the hosts and the switch by typing certain commands in the CLI. As we want to recreate the exact same behaviour in all tests we can not write every command individually. Instead once the hosts are running their backends we start a script that creates the connection behaviour we want and that provides reliable results for all tests.

3.4 Network Communication between Replicas

In all the different systems tested in this thesis the communication is done by using sockets running TCP. Several libraries that would have provided a sufficient backend solution were considered, SocketServer, SimpleHTTPServer and Sockets or a combination of SimpleHTTPServer and SocketServer. SimpleHTTPServer uses the SocketServer to send its messages and the SocketServer in turn uses Sockets. Based on the low bandwidth and the high rate of connection loss SimpleHTTPServer was out-ruled as the extra data added to each message by the HTTP protocol would increase message size for functionality that was deemed unnecessary for the systems [7, 10]. The SocketServer and the Sockets were left as options with Sockets being the better option for the backend core communication as it meant the bare minimum for the systems.

Each node act as both server and client in the network. In all systems nodes listen for connections from other nodes and connect to other nodes to send updates or states. The server part of the node was built by setting up a socket that has been configured for IPv4 with Transmission Control Protocol (TCP). An IP-address together with

a port is bound to the socket. When setting these parameters the socket can listen for incoming connections. A loop was created that checks for incoming connections, when connections are established threads are created to handle each new connection to avoid blocking new incoming connections. The code used in all systems created in this project can be seen in Source Code 2.

```
# AF_INET -> ipv4 and SOCK_STREAM -> tcp
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Starting server
sock.bind((IP, PORT))

# Listening for connections
sock.listen(5)
while True:
    # someone connected to the socket
    connection, connectioninfo = sock.accept()

    # Creates a new thread to handle each connection
    thread = Thread(target=handleconnection)
    thread.daemon = True
    thread.start()
```

Source Code 2: Backend communication on all nodes listening for incoming connections.

To be able to send data in an environment where connections may be lost at any time the recipient must be able to acknowledge if the entire message has been received. Since it is a stream of data and chunks of data can be sent at different speeds the receiver does not know how much data it should wait for. To counter this problem an "end of transmission byte" in the form of a semicolon is appended to the message which can be seen on line 1 in Source Code 3. Once the data is ready to be sent the size of the data is saved in order to estimate when all bytes of the message have been sent. The data is sent by the **send** function which returns the number of bytes sent. It transmits as many bytes as possible but there is no guarantee that all bytes are sent. Therefore after each call to **send**, the number of sent bytes is saved to the total bytes sent variable, then continues to send all the bytes that have not yet been sent. If **send** returns 0 the connection has been lost and the send attempt is canceled. This is displayed on line 6-7 in Source Code 3.

```
data = (DATATOSEND+ ";").encode()
datasize = len(data)
totalsent = 0
while totalsent < datasize:
    sent = sock.send(data[totalsent:])
    if sent == 0:
        break
    totalsent += sent
```

Source Code 3: How the systems send data between nodes.

As mentioned before the data stream that is sent is ended by a semicolon. The receiver continuously compares every incoming byte individually to the end statement. If the received byte is not the end byte nor *EMPTY/NULL* the byte is appended to the received message. If the end byte is received the loop is broken and the message is delivered. The order of the received bytes is maintained thanks to the sockets using a TCP connection. If an *EMPTY/NULL* byte is received the connection is broken and the entire message is dropped. The code for the recipients is illustrated in Source Code 4.

```
while True:
    byte = connection.recv(1)
    byte = byte.decode()
    if byte == ";":
        receivedMessage = True
        break
    elif byte:
        data += byte
    else:
        break
```

Source Code 4: Code for message reception on nodes.

3.5 Data Persistence

The databases that nodes use to persist data in the virtual network are SQLite databases. It has support for Python, has good documentation and is the type of database that the vehicles in the real life scenario use. The nodes have the exact same database layout as the vehicles with the exception of foreign keys (relations between tables). All tables in the database are using a primary key column that is an *auto increment* integer called _ID. This means that when an insert is performed to the table, a value is added which is one value higher than the _ID value of the last inserted.

In this thesis the functions the database can perform has been limited, these can be seen in the API "DbConnect" in Figure 3.3. This API allows both the State/Delta-CRDT systems and the Centrally managed system to create new databases from a hard-coded layout, add entries to tables, query the database to get the full state or desired parts of it, check if entries or databases exist, get snapshots (only used by Delta-CRDT), delete entries and update entries.

DbConnect contain no decision making, all decisions are done by the systems themselves, it just allows the systems to communicate with the database/databases.

4
Experimental Study

In this chapter the results from the gathered data representing the connectivity of the network, the behaviour of the vehicles in the real-life system, all seen in Section 4.1, in addition the results from the tests made in the simulation are discussed and shown. The results are used to determine the performance between the different implemented systems, Delta-CRDT, State-CRDT and a Centrally managed system each discussed in Sections 3.2.4, 3.2.3 and 3.1.

A Reference system has been created to be used to compare and draw conclusions on the performance of the other systems. The Reference system is a centrally managed system operating in a test environment with *perfect conditions*. *Perfect conditions* in the Mininet network simulator(virtual environment), explained in Section 3.3, is bandwidth of links set to 1 Gbit/s, a latency of 10ms and there are no connection loss in the system. The Reference system is ran in the virtual environment with the same number of nodes as the other systems 8, 13 and 16. The tests for the Reference system are made to be used as a reference when comparing the performance of the systems by measured metrics that will be defined in the following paragraph. The centralized system and the CRDT systems have ran the same test script with the same predefined settings of number of nodes, broadcast time interval and disconnect rate. The Reference system differ by having better bandwidth, latency and a disconnect rate of 0%.

In Section 4.1 information is given to clarify how the test script was created how the settings for the virtual nodes and virtual environment were determined based on data collected from a system in production at CPAC Systems. This is then succeeded by a section with the specifications of the hardware the virtual environment, systems and tests were run on.

The following sections present the results of each system of the performance metrics that were measured. In Section 4.2 the converge time metric's results are presented. Converge time is the time it takes in seconds, from the test start, for all nodes in the system to converge to the same state. The converge time gives a hint of the overall performance of the system, a high converge time suggest that the system have had difficulty to handle the load on the network or nodes have not been able to load with the local calculations. Because of how it is implemented the converge time will always be $ConvergeTime \geq TestDuration$.

Section 4.3 presents the message size, the number of bytes sent message from a node has, metric's results. The message size can be compared to see how much strain the system as a whole puts on the network, the bigger messages sent the longer it takes for the receiver to get the message and thus take up resources on the network. The message size is measured by the node before sending each message. The message size has a direct correlation to the size of the state on the nodes for multiple of the examined systems as the message sometimes is the entirety of the state.

Following Section 4.3 is the message latency metric which is presented in Section 4.4. The message latency is the time it takes from the time a connection is setup between two nodes until the entire message has been received at the receiving node. The message latency metric shows the performance of the network. High message latency times indicate that the network has been flooded with too many and/or too large messages. There is a direct correlation between the message size and the message latency if the bandwidth is constant.

Section 4.5 present how many bytes in total was sent during the test. The target of this metric is to get a more comprehensive view of how much data is being transferred in the system. The bytes sent provides another dimension to the amount of messages sent in the system as a whole and how the total amount of bytes is spread between the nodes in the system. Each time a message is sent the size of that message is added to the total bytes sent metric. At the end of the test it contains the total sum of messages sent.

Section 4.6. The merge latency is the time it takes for a node to perform an operation. Operation in this context can either be a received message from another node, which correlate to a state, or a local update which can be seen as a state with a single atom. When a state is received by a node it performs a *merge operation*, discussed in Section 3.2.2, to update its own state. The merge latency is the time from a state is received until the *merge operation* has been completed. This metric provide information on how the nodes in the systems handle the calculations required to perform their actions and if they are flooded with operations does this affect the latency negatively, if so, to what extent.

Finally, in Section 4.7 is discussion on the results of the *quality arrays*. *Quality arrays* is defined as arrays of success or failure of message deliveries from other nodes. *Quality arrays* is not a performance metric but rather a way to see that nodes actually have diverged during the test. In the section it will be discussed how the different systems diverge and how the nodes reacquire the same state after diverging.

The results presented and discussed in this section are a selection from the extensive amount of data generated by the tests. The results are summarized and the focus has been on the performance of the systems for tests made with 16 nodes as this is the most demanding setting for the systems, to see all the gathered results from the test they can be found at: https://github.com/perzonas/exjobb/tree/master/results.

4.1 Setup of Virtual Test Environment

The test environment consist of a virtual network created with Mininet, explained in 3.3, in which nodes are running a script that simulates local updates that are propagated differently by the nodes in each system.

Figure 4.1: The Chepstow quarry, where data has been collected, with the position of active vehicles marked with vehicle symbols. Picture from CPAC Systems [1].

The test data and parameters are based on the data produced by vehicles running in the Chepstow mine, seen in Figure 4.1. In this unreliable environment messages are dropped because vehicles disconnect from time to time. The data is used to get as accurate inputs as possible for the systems in addition to the overall accuracy of the simulation.

4.1.1 Local Updates

For consistent testing and testing the systems in a similar way to the real-life scenario, updates from each node are performed to mimic how the vehicles would receive them. Every node continuously read a row from its local file where each row represents an action performed by the vehicle that is to be applied to the local database. In order for all the updates not be applied at once there is a script that writes to N local files, N being the number of nodes for the test, at predefined times. This way the timing of updates can be controlled and all tests will be identical for all the systems. An illustration of the process is shown in Figure 4.2.

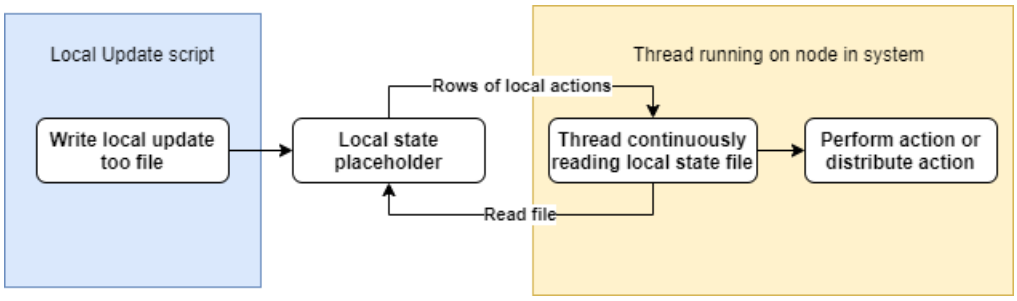


Figure 4.2: An illustration of how the local updates are created and perform on the systems in a consistent manner.

4.1.2 Number of Updates Per Vehicle

How much load in terms of updates to put on the systems was based on data gathered from 13 active vehicles. The data provided the number of actions from three different time intervals, Short Interval (48h), Medium Interval (1 week) and Long Interval (3 weeks), each with a different time frame (30min, 1 hour, 3 hours).

The service provided to extract the gathered data did not support the option to go further into depth i.e. to look at smaller time frames. It did not allows us to see how the updates were distributed during the time frames available to us. We focused on three different time intervals, can be seen on the last row of each interval's result seen in Table 4.1.

Table 4.1: Results from gathered data about the actions performed.

Short Interval (48h)	**Results**
Average updates per hour	421.888889
Average updates per hour per vehicle	32.452991
Highest number of updates in a 30 min window	737
Medium Interval (1 week)	
Average updates per hour	631.938967
Average updates per hour per vehicle	48.610690
Highest number of updates in a 1 hour window	3295
Long Interval (3 weeks)	
Average updates per hour	614.625571
Average updates per hour per vehicle	47.278890
Highest number of updates in a 3 hour window	9088

When looking at the results the interval with the highest number of updates in Table 4.1 is used as a guideline when creating the script that performs the local updates, discussed in Section 4.1.1. If the system is able to handle the higher number of updates it will be able to handle lower loads as well. Updates are performed in waves to be as close to real-life as possible, when a vehicle starts working in the beginning of its shift it does not perform all of its work directly, the work is spread

across the entirety of the day. Each wave of updates are different to see if the systems can handle a high number of actions in a short period of time in addition to lower loads.

Table 4.2: Time intervals displaying the number of updates performed by each node during testing.

Time (s)	0-50	50-95	95-135	135-170	170-200	200-225	225-245	245-260	260-270	275-280	>280
Updates	5	10	3	3	1	10	3	5	20	15	5

The behaviour of the vehicles in real-life can also be viewed as performed in waves as they perform their updates at different speeds, a digger loading a dumper truck performs more frequent updates than a dumper truck transporting material from point A to point B. The different waves can be seen in Table 4.2, in the end of the test (245<) more than 50% of the updates are performed to simulate a burst of updates with almost the same amount of updates as the average updates per hour in Table 4.1. The behaviour of the local updates should affect the results and thus provide interesting end results.

4.1.3 Network Performance

To be able to simulate a test environment as close to the real-life vehicles network data was collected from the vehicles in production. The data, containing network performance collected from the vehicles were structured as shown in Source Code 5.

```
1   {
2     "timestamp": "1555579143274",
3     "signalStrength": 18,
4     "networkOperator": "Tele2",
5     "dataStatus": 2,
6     "location": {
7       "longitude": 11.9994135,
8       "latitude": 57.647903
9     }
10  },
11  {
12    "timestamp": "1555579143274",
13    "signalStrength": 99,
14    "networkOperator": "unknown",
15    "dataStatus": -1,
16    "location": {
17      "longitude": 11.9994135,
18      "latitude": 57.647903
19    }
20  }
```

Source Code 5: Collected connection data.

The different fields of importance are an epoch *timestamp* which is the number of milliseconds from January 1st 1970 and *signalStrength*, the quality of the connection in Arbitrary Strength Unit (ASU). There are different ASU values depending on what kind of signal it is used for, in this case it is GSM (3G) [3]. With GSM (3G), ASU can have values between 0 and 31 as well as 99 where 99 means that the signal is undetectable (disconnected). *NetworkOperator* is ignored as it provides no useful information in this context. *DataStatus* displays if the vehicle is connected, 2, or disconnected, -1. *Location* is also ignored as only connectivity is of interest but could be used to find connectivity issues of certain areas.

The signal quality is measured in dBm and for the 3G network it is called Received Signal Strength Indication (RSSI) [9]. To change the ASU value into an RSSI value the following formula is used:

$$2 * ASU - 113 = RSSI[17]$$

The calculated RSSI value can then be used to gain knowledge about the quality of the signal. The signal quality is divided into 5 different levels [12], these are shown in Table 4.3.

Table 4.3: Table of the signal quality from RSSI value.

RSSI	QUALITY
<-70dBm	Excellent
-70dBm to -85dBm	Good
-86dBm to -100dBm	Fair
>-100dBm	Poor
-110dBm	No signal

As the signal quality declines the ability to transfer data is lost faster than the availability of telephone communications. Already at poor connection quality in Table 4.3 there is next to no data available [29]. The collected data show the following results:

- Total number of seconds measured: 6426
- Total number of seconds disconnected: 87
- Total number of seconds connected: 6339
- Downtime percentage: 1.354%
- Number of disconnects: 3
- Average time of disconnect: 29 seconds
- Average time between disconnects: 32min & 8 sec
- Highest signal strength recorded (ASU): 26
- Lowest signal strength recorded (ASU): 1
- Average signal strength (ASU): 16.1325
- Average signal strength in dBm (RSSI): -80.7349
- Average signal strength is classified as: Medium (good voice and data)

From the results it can be deduced how often and for how long connections should be removed between hosts, explained further in Section 4.1.4. In addition to this it can be interpreted that the average quality of the connection is sufficient for providing both data and voice to the nodes. There is no direct formula to provide the upload and download speeds for a 3G connection based on the signal strength in either ASU nor RSSI. To be able to find a bandwidth for the network, the connection speeds of a 3G connection were analyzed. There are many different possibilities for doing this depending on the generation of 3G used. It ranges from 0.1 Mbit/s up to 8 Mbit/s in upload speeds and 0.3 Mbit/s to 42 Mbit/s in download speeds [12, 16]. In this case the upload speed is the most important of the two since a node can not download any more than another node can upload. Thus the upload speed is setting a limit for the connections. During testing we will be keeping the speed at 1.5 Mbit/s, which is usually the throughput you receive from a 3G network [12]. We believe this is the correct choice for the scope of this thesis. The average latency of all the 3G networks in Australia is 150ms [16]. Ken Lo claims that the average latency for 3G overall is 100ms [12]. Based on this we will perform the test using a latency of 150ms to simulate bad connectivity and the systems will work even better with better latencies.

4.1.4 Connectivity

To be able to simulate connectivity loss, a script was made that runs in parallel with all the hosts and with the local updates script mentioned in Section 4.1.1. This script changes the status of the links between the hosts and the switch, the switch simulate the network which all nodes connect to. When the status is changed to *off* no packets can be sent across that specific link. The status can be swapped freely at any time from either *on* or *off*. The time a host is disconnected is brief as mentioned in Section 4.1.3.

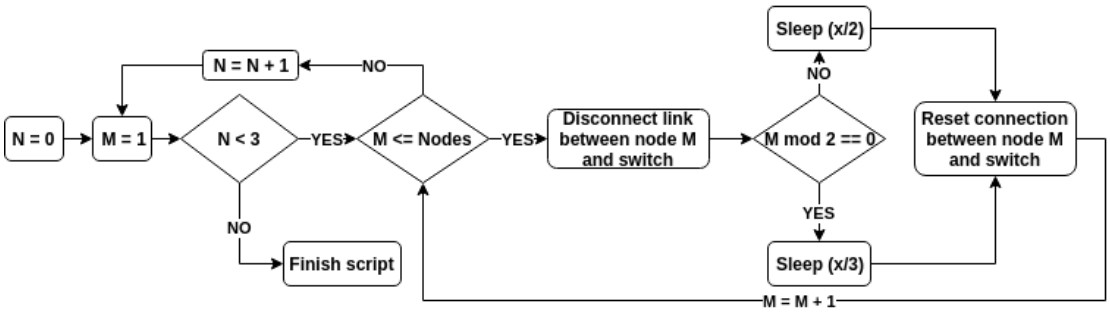


Figure 4.3: The basic idea behind the script performing systematic connects/disconnect on the nodes during the test.

The idea is that a test should be 300 seconds long and with 1.35% disconnect time that will result in a 4 second disconnect for the entire test. The 4 seconds does not have to be sequential and can be split up into multiple shorter sequences. In the test data there were 3 different disconnects all with similar length. To recreate the real

life scenario 3 disconnects were simulated each lasting for $\approx N/3$ seconds, where N is the the total time a node should be disconnected in seconds. For the real life test N equals 3. During the test two different disconnect patterns is used. Some nodes disconnect 3 times and other disconnect 2 times but for longer periods of time. It is good to have variety in the duration of disconnects as well as longer periods of downtime. In the real life data the disconnects are not equal and to choose a longer duration disconnects on some nodes will provide a more interesting result.

The tests will be based on the information gathered from the data. 4 second disconnect duration based on the 1.35% will be used as the base case. Additional tests with increased disconnect duration will be performed to simulate scenarios with lower connectivity than the connectivity deduced from the gathered data. This is to see if there are any changes in the performance of the different systems when they are challenged with lower connectivity.

4.1.5 Hardware

All the tests have been performed on a Lenovo Thinkpad T420. To get a fair representation of the results a single machine was used for all the tests. Using multiple machines could have potentially skewed the results for this thesis. The specifications are shown in Table 4.4.

Table 4.4: A table of the hardware specifications used during testing

LENOVO THINKPAD T420	
Processor	Intel® Core™ i5-2520M CPU @ 2.50GHz
Memory	8GB DDR3
Storage	Samsung MZ7PA 128GB SSD
Operating system	Ubuntu 18.04.2 LTS

4.2 Converge Time

In this section the converge time of the systems are presented and discussed. Converge time is the time it takes for the system to converge to the same state on all nodes from the start of the test. The converge time can not be less then 300 seconds because of how the test is built, there are local updates being performed for the first 300 seconds and the system can not converge before all local updates have been introduced.

4.2.1 Reference System Converge Time

The converge times of the Reference system tests are close to nothing. The test can not be completed before 300 seconds, as this is the length of the test. In Table 4.5 are the converge times of all the test done with *perfect conditions*. The results clearly states that the system converges in less than 0.3 seconds independent of the

number of nodes and broadcast interval. Table 4.5 also displays 2 patterns, that the converge time increases with the number of nodes in the system as well as with a decreasing broadcast interval even if only slightly.

Table 4.5: Table of the different converge times from tests on Reference system.

Number of Nodes	Broadcast Interval			
	5	**15**	**25**	**35**
8	300.0339	300.0163	300.0157	300.0165
13	300.15034	300.0346	300.0357	300.0337
16	300.27895	300.0843	300.0481	300.0882

4.2.2 Centralized Converge Time

The convergence times for the centralized system are shown in Table 4.6. All results are similar with converge times between 300 to 301 seconds with the exception of the results from the 16 nodes. The load on the system increases with the size of data and the number of nodes. For each additional node in the system the total data size in the system increases in addition to the number of nodes the master node needs to send its state to when broadcasting.

Table 4.6: Converge times gathered during tests with various setting for the Centralized system.

Disconnect rate	Number of Nodes	broadcast interval			
		5	**15**	**25**	**35**
4 Seconds/Test, 1.35%	**8**	300.0313	300.0287	300.0328	300.0303
8 Seconds/Test, 2.7%	**8**	300.0161	300.0173	300.0168	300.0179
16 Seconds/Test, 5.4%	**8**	300.0166	300.0209	300.0155	300.0166
4 Seconds/Test, 1.35%	**13**	300.0679	300.0647	300.0688	300.0651
8 Seconds/Test, 2.7%	**13**	300.0361	300.0376	300.0371	300.0355
16 Seconds/Test, 5.4%	**13**	300.0400	300.0354	300.0350	300.0386
4 Seconds/Test, 1.35%	**16**	335.2897	300.0917	300.0935	300.0958
8 Seconds/Test, 2.7%	**16**	358.6397	300.0492	300.0511	300.0519
16 Seconds/Test, 5.4%	**16**	427.7964	300.0621	300.0466	309.4215

The result show that somewhere between 13 and 16 nodes with a 5 second broadcast interval the system starts having problem to maintain the one second window all the other test have managed to converge in. The network is getting flooded with messages slowing down the message latency as discussed in Section 4.4. This affects the converge time negatively and the test also show that this is effected by the DR of the systems. A higher DR provides a higher converge time when the network is being flooded. When the network is not flooded the results are similar independently of the DR as can be seen in all results with 13 nodes or less. There is one outlier in the

result which is the 16 nodes with 5.4% DR. Because of the other test with 35 second broadcast interval being able to converge in $< 301 seconds$, it could be because of the timing of the broadcast is badly timed or one node lost the last message due to a disconnect and must wait for the next broadcast.

4.2.3 State-CRDT System Converge Time

The converge time for the State-CRDT system is growing fast with the increase of nodes as well as the broadcast interval. For 8 nodes the converge time is steady within $300 < ConvergeTime < 301$, but with 13 nodes results are already showing worse converge times. Looking at the test with broadcast interval of 5 seconds, the results are significantly worse, as the converge times range from 1400 to ≈ 3300 seconds, seen in Table 4.7. The network is not able to handle the amount of data sent between the nodes, all messages are transferred at slower speeds as more message are sent concurrently. Once the *message latency* > *broadcast interval* the network enters a vicious circle where each following broadcast will only slow the network down further. In the tests made, slowing down the network also leads to a slow down of all other actions performed by the nodes as everything is simulated on the same machine.

Table 4.7: Table of the converge times gathered during tests for the State-CRDT system.

Disconnect rate	Number of Nodes	broadcast interval			
		5	15	25	35
4 Seconds/Test, 1.35%	**8**	300.1140	300.1078	300.1079	300.1072
8 Seconds/Test, 2.7%	**8**	300.1311	300.0535	300.0576	300.0596
16 Seconds/Test, 5.4%	**8**	300.0593	300.0616	300.0579	300.0573
4 Seconds/Test, 1.35%	**13**	1953.8584	300.3037	300.3017	300.2878
8 Seconds/Test, 2.7%	**13**	1407.7268	300.1439	300.1490	300.1684
16 Seconds/Test, 5.4%	**13**	1400.2475	314.5570	300.1450	300.1416
4 Seconds/Test, 1.35%	**16**	4798.6765	985.2645	472.7948	300.4186
8 Seconds/Test, 2.7%	**16**	6001.0533	1171.3792	574.9403	300.2352
16 Seconds/Test, 5.4%	**16**	>16000[1]	1344.4049	640.9116	343.6800

The broadcast interval has a bigger effect on the converge time than the DR. There are vast improvements when the broadcast interval is decreased. For example in the test with 2.7% DR the converge time for 16 nodes starts at 6000 seconds for the 5 second broadcasts and decrease with each level of broadcast interval down to 300.2352 for the 35 second broadcast interval. That makes it a decrease of 95% in converge time, the same number is $\approx 96\%$ for the test with 5.4% DR when using the last gathered value from the test, 16000 seconds, before the computer crashed to calculate. The reason for the computer crashing is that during the test the system reaches a point where the message latency is longer than the broadcast interval

[1]The computer used during testing ran out of memory and crashed

rate. So for each additional broadcast the network is slowed down and for each message in transit a new thread is started. With more threads being added than threads getting killed due to finishing its task, the computer runs out of memory and crashes. Each times this happened the operative system needed to be repaired. The connections does not timeout because they are sent at very low speeds and not broken or lost. We could have implemented a maximum time for the message latency before dropping the connection on the receiver side, but then the systems might have stopped working completely as all messages could get dropped if the message latency would be higher than the threshold. If we would use different thresholds for each system the results would be skewed and invalid.

Comparing the converge times from the different systems the Reference system does not have any indication that it reaches any limits with high frequency broadcast during the 16 node test. The centralized systems starts getting worse converge times when tested with 16 nodes and a 5 second broadcast. In contrast, the State-CRDT is getting worse result with 13 nodes and 15 second broadcasts.

4.2.4 Delta-CRDT System Converge Time

The Delta-CRDT system manages to converge within one second for almost all tests with a broadcast interval of 15 seconds or more. During the two most demanding tests in DR for this broadcast interval with 16 nodes the system is not able to converge within one second of 300, shown in Table 4.8. This is an improvement compared to the State-CRDT system which already had an increased converge time of 314 seconds when using 13 nodes with 5.4% DR in addition to a converge time of 343 seconds with a 35 second broadcast interval with 16 nodes. The State-CRDT system gets better results when the broadcast interval is increased and the same can be seen for the Delta-CRDT system.

When excluding all converge times less than 301 seconds, the Delta-CRDT system also follow the same pattern as the Centralized and State-CRDT system where a higher DR increases the converge time of the system if all other parameters remain unchanged. For most tests the converge time of the Delta-CRDT system and Centralized system, and all test for the Reference system, are within the time interval $300 \leq ConvTime \leq 301$ seconds which is considered acceptable and a good converge time for the system used today. For the 5 seconds broadcast interval the Delta-CRDT system is having trouble to maintain the converge times of the Centralized system. Since more data is sent for the Delta-CRDT system the network gets flooded and slowed down compared to the Centralized system.

Since the Delta-CRDT system with 25 second broadcast interval, for all number of nodes, have a good converge time this would be a plausible solution to replace the Centralized system with a higher broadcast interval when only taking converge time into consideration. Even though the broadcast interval is higher than what the Centralized system can handle without losing convergence time the machines using Delta-CRDTs can still be updated equally often or even more so, as they can receive

Table 4.8: Table of the converge times gathered during tests with various setting for the Delta-CRDT system.

Disconnect rate	Number of Nodes	broadcast interval			
		5	15	25	35
4 Seconds/Test, 1.35%	**8**	300.1067	300.1109	300.1079	300.1084
8 Seconds/Test, 2.7%	**8**	300.0572	300.0564	300.0560	300.0573
16 Seconds/Test, 5.4%	**8**	308.3991	300.0551	300.0543	300.0540
4 Seconds/Test, 1.35%	**13**	384.1841	300.3399	300.2772	300.2781
8 Seconds/Test, 2.7%	**13**	420.1244	300.1711	300.1460	300.1455
16 Seconds/Test 5.4%	**13**	588.4631	300.1397	300.1426	300.1446
4 Seconds/Test, 1.35%	**16**	1204.7629	300.5428	300.4104	300.4158
8 Seconds/Test, 2.7%	**16**	1418.3686	333.5832	300.2087	300.2159
16 Seconds/Test, 5.4%	**16**	1758.2876	367.1295	300.2191	300.2161

updates from an arbitrary machine and updates can propagate along any path. If a slave node N in the Centralized system miss a message from the master node M , due to connectivity issues, it has to wait for the next broadcast from M before updating its own state. The time N has to wait to receive an update is equal to $broadcastinterval * 2$, there is the possibility that the next message may not reach node N either, giving the time to receive an update if the first message is dropped $\geq broadcastinterval * 2$.

For a node N in the Delta-CRDT system that time would be $> broadcastinterval$ as N does not have to receive the update from node M directly. The update can take alternative paths, $M \rightarrow K \rightarrow D \rightarrow N$ or $M \rightarrow K \rightarrow N$ where $M, N, K, D \in P$ where P is the set of all nodes in the system and $M \neq N \neq K \neq D$. Another node may receive the update and the receiving node can then broadcast the new updated state to the node that missed the original broadcast thus the message *can* be received at N after $\leq broadcastinterval * 2$.

4.3 Message Size

In this section the message size results measured during testing are shown and discussed. The message size is the size of each message sent by a node during the test. The message size metric's primary use is to see what improvement the Delta-CRDT provides. As described in the prelude to Chapter 4 the amount of data generated from the tests is extensive. The results of the message size tests are therefore presented by comprehensive boxplots as can be seen in Figure 4.4 and 4.5. All data can be viewed at *https://github.com/perzonas/exjobb/tree/master/results*.

Both Figure 4.4 and 4.5 present the tests with 16 nodes and 5 seconds broadcast interval. The difference is the disconnect rates which are 1.35% and 5.4% respectively. The reason for presenting the tests with the most nodes and lowest broadcast interval is because they were the tests the systems struggled the most with. Looking at

tests with longer broadcast interval only increases the average and median message size. This is due to the nodes accumulating more data between the broadcasts as the data generation is not dependent on the broadcast interval.

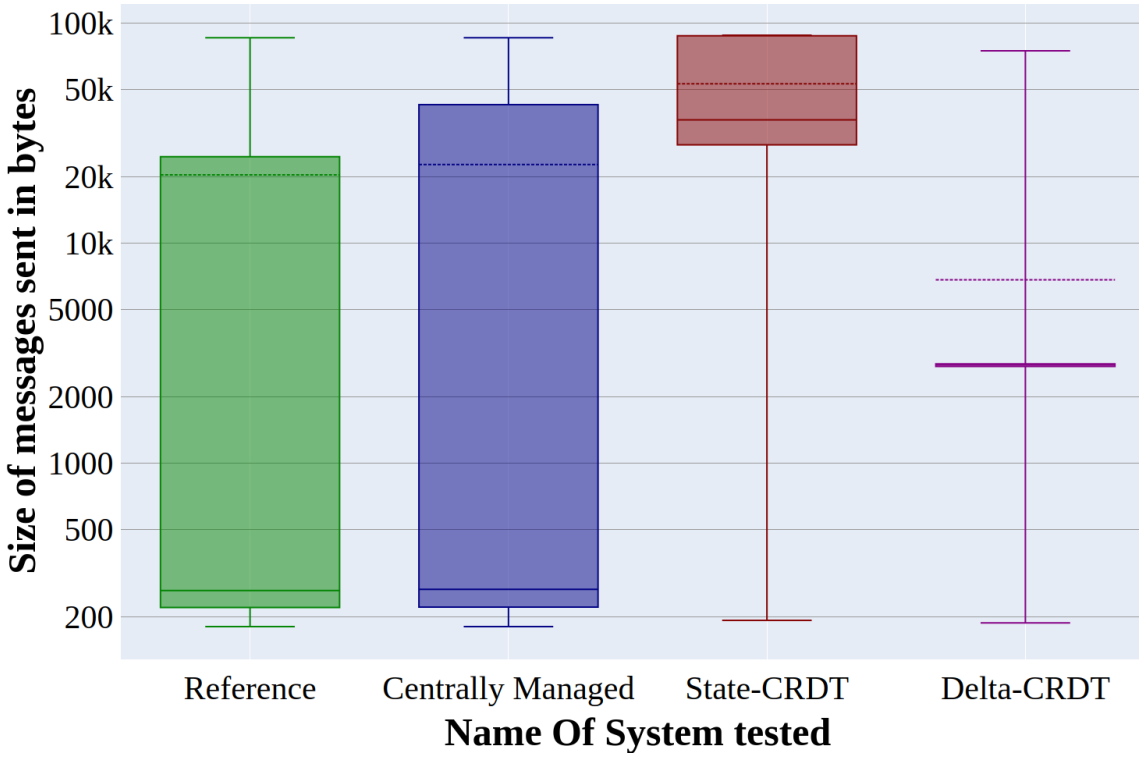


Figure 4.4: Size of messages sent by 16 nodes, 5s broadcast interval, 1.35% disconnect rate. The boxes include 25-75 percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

The results shown in Figure 4.4 and 4.5 are quite similar. The maximum size of messages remain the same despite an increase in disconnect ratio. This is expected since the data that is predefined to be produced in the test is a fixed amount. The increase in disconnect ratio does however affect the upper quartile of the size of the messages sent in the Centrally Managed system and the overall message size of the State-CRDT system. The reason is that the nodes that are disconnected are still producing data. Therefore, when they reconnect they send the accumulated data resulting in larger message sizes.

The Delta-CRDT system has the same mean (dotted line) regardless of which disconnection rate is being used. The Delta-CRDT system's nodes does accumulate data while being disconnected just like the Centrally Managed system and State-CRDT system. The reason that the Delta-CRDT system's mean message size remain the same even though the disconnect rate is increased is because when disconnected, nodes enable other nodes to send smaller states. The nodes that are disconnected are not adding to the delta-states that the nodes are sending to each other. Taking this to an extreme would be when there are only two out of all nodes that are not

disconnected. These two nodes will send smaller delta-states as these two will only send whatever new data they are producing. Thus during the time with only two nodes not disconnected the mean message size will become lower in comparison to the Centrally Managed system and State-CRDT system which will not lower their means.

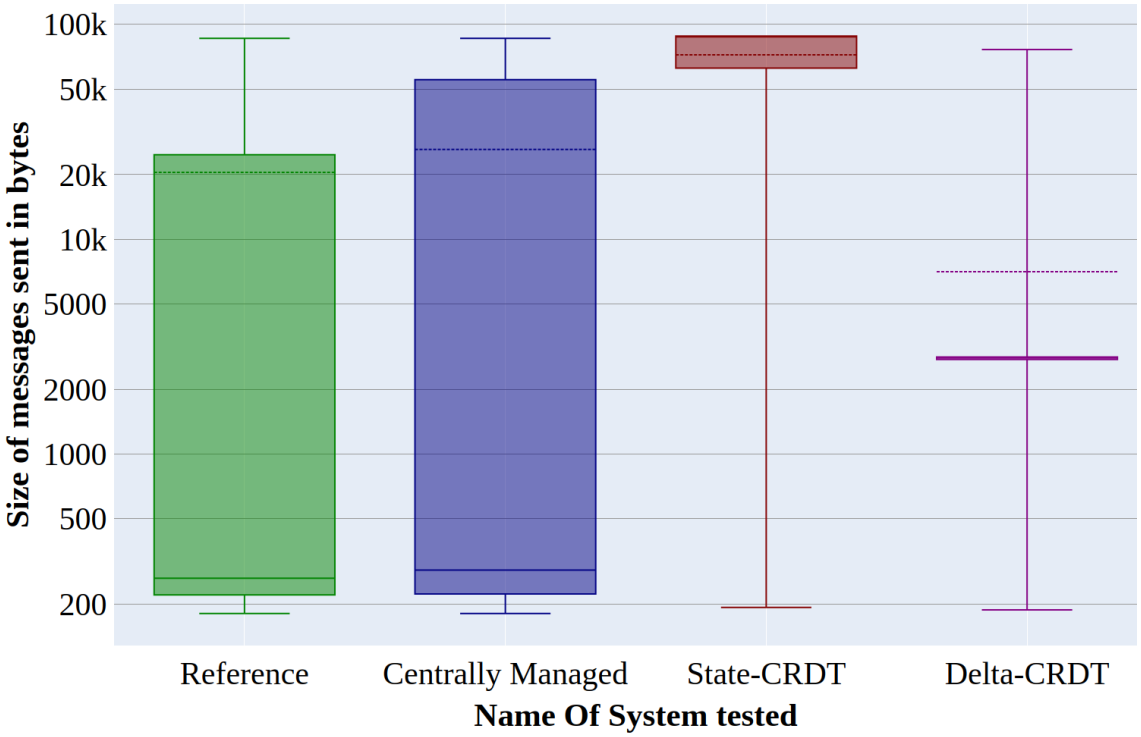


Figure 4.5: Size of messages sent by 16 nodes, 5s broadcast interval (except State-CRDT which has 15 second broadcast rate because of crash during testing), 5.4% disconnect rate. The boxes include 25-75 percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

The mean value of the Delta-CRDT is skewed due to the snapshots that the Delta-CRDT sends. These snapshots are consistently approximately 3000 bytes in size regardless of the size of the delta states that the nodes are sending thus lowering the mean. The same is applicable to the result of the Centrally Managed system where the slaves send really small messages lowering the mean though not to the same extent as the Delta-CRDT system. However, in the case of the Centrally Managed system the messages sent by the slaves are not overhead messages such as the snapshots the Delta-CRDT system is sending.

The mean for the Delta-CRDTs in Figure 4.4 and 4.5 is outside the box compared to the other systems where the mean is located within the box. The box of the Delta-CRDT is very thin around the median due to all the snapshots representing a large part of the total messages sent. The median is $\approx$ 3000 bytes and the largest

25% of the messages range from ≈ 3000 bytes to $\approx 80k$ bytes and the smallest 25% range from 180 bytes to ≈ 3000 bytes. The largest 25% of the messages has a ≈ 24 times bigger span compared to the lower 25%. The difference leads to the mean, in this case, climbing above the median and even outside the box.

The most fair comparison of the results of the different systems is the mean. As previously mentioned the mean of the Delta-CRDT System is skewed due to the snapshots which are overhead messages. To compensate for the snapshots, which are approximately half of all the messages sent by the Delta-CRDT system during the test, the mean should be doubled. This means that the mean of the Delta-CRDT system is about 15k bytes. Despite this it can be seen that the Delta-CRDT system perform approximately up to 50% better than the Centrally Managed system and approximately 4.5 times better than the State-CRDT system.

4.4 Message Latency

The message latency is the time it takes from the time a connection is setup between two nodes until the entire message has been received at the receiving node. The message latency metric shows the performance of the network. High message latency times indicate that the network has been flooded with too many and/or too large messages. The results from the readings of message latency are discussed in this section.

The message latency metric is the metric that gives the best overview of the state of the network. The links in the virtual environment have a constant bandwidth and the message size has a fixed max size which is the maximum state size. If the network would not get affected by flooding the *Maximum* value of all systems that send messages with the max size would have the same message latency. This is not the case as seen in Figure 4.6 where the Reference system has a lower *Maximum* compared to the Centrally managed system. The State-CRDT system's *Maximum* is greater than that of the Centrally managed system. The reason for the Reference system having a lower maximum is because of the higher bandwidth for the links inside the virtual Mininet environment which provides a lower message latency. The biggest message for the State-CRDT is the same as the biggest message for the Centrally managed system and they have the same bandwidth during the test yet the State-CRDT has a *Maximum* that is ≈ 10 time that of the Centrally managed system. This is a clear indicator of the load on the network/system as a whole for the State-CRDT.

Comparing the *Minimum*, all systems have approximately the same value. The *Minimum* value is either the first broadcast or a local update in the centrally managed systems. In the beginning of the tests the states are very small, as small as a single atom, and the network is at this stage cooping with the load. With all systems having the exact same potential smallest message and a constant bandwidth they have the same message latency. The Reference system should have a slightly lower

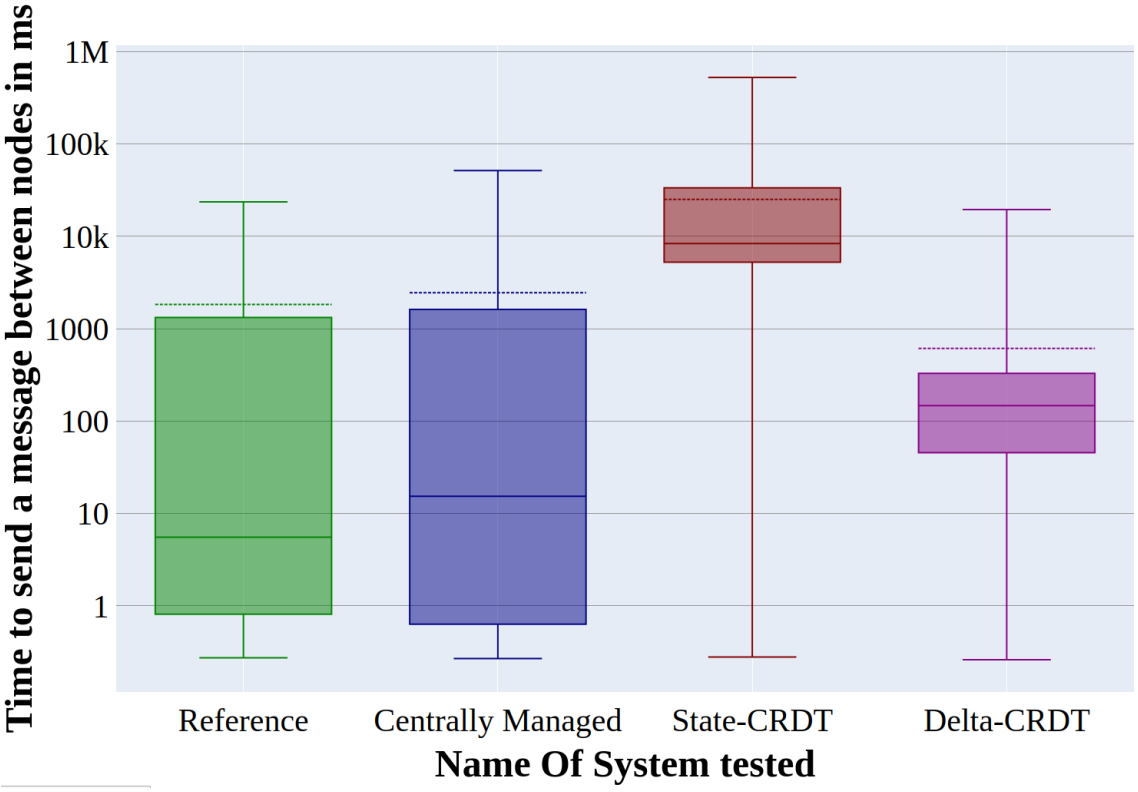


Figure 4.6: Latency to receive messages between nodes for systems with 16 nodes and 5s broadcast interval and 1.35% disconnect rate. The boxes include 25-75 percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

Minimum because of the higher bandwidth during the test, but the size of the message is small enough that the difference in message latency is negligible and because of that all systems have approximately the same *Minimum* value.

Continuing with the comparison of the *Mean* value in Figure 4.6. The Delta-CRDT has the lowest *Mean* which is unexpected because of the results in Figure 4.4 and 4.5 where the *Median* of the Reference and Centrally managed system are $\approx$ 10% of the Delta-CRDT. For these systems over 50% of their messages sent are smaller than 300 bytes compared to the $\approx$ 3000 bytes of the Delta-CRDT and this would have a positive effect on the *Mean* message latency. The *Mean* of the Reference and Centrally managed system is outside of the box in Figure 4.6 compared to Figure 4.4 which is unexpected as the Mean is higher than the latency of three quarters of messages sent compared to the *Mean* message size is less than the last quarter of sizes seen in Figure 4.4. The State-CRDT as in all previous test are the worst performing system. It is only the minimum, as discussed above, that is not worse than the rest of the systems. The broadcasting of the entire state from all nodes overflow the network and as the state grows the load increases which lead to longer message latencies.

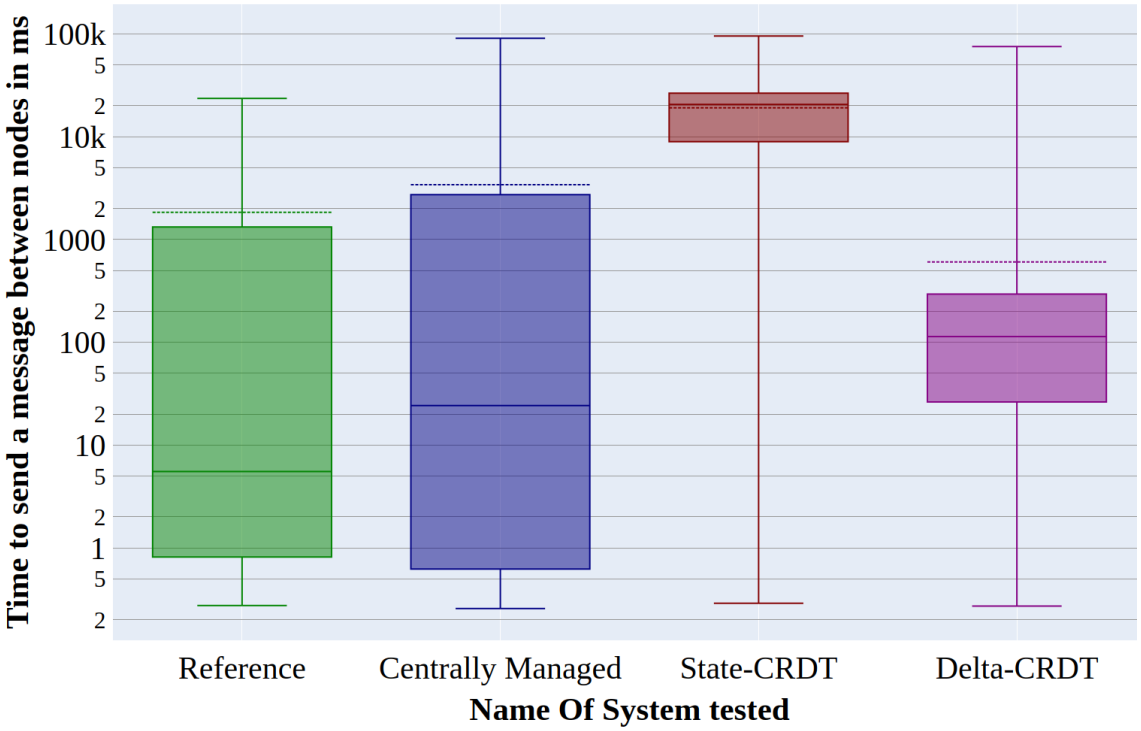


Figure 4.7: Latency to receive messages between nodes for systems with 16 nodes and 5s broadcast interval and 5.4% disconnect rate (except State-CRDT which has 15 second broadcast rate because of crash during testing). The boxes include 25-75 percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

Looking at the differences between Figure 4.7 and 4.8 the first thing that catches the eye is the change in *Maximum* values. For State-CRDT it is decreased by $\approx 85\%$ while the other two systems have an increase in *Maximum*. The reason behind the decrease of State-CRDT is the fact that the broadcast for this test is not the same in the figures, it is three times slower, 15 seconds, in Figure 4.7 the reason for this is mentioned in Section 4.2 as no data was able to be collected for the 5 second broadcast. This lowers the number messages in transit on the network which decreases the latency as seen for the State-CRDT. Delta-CRDT and the Centrally managed system both have a higher *Maximum* with increased disconnect rate. The size of the message have not changed considerably as discussed in Section 4.3, with the same message sizes and a fixed bandwidth the reason for the higher latencies are the load on the systems. With higher disconnect rates there are moments with higher load on the systems which gives a higher *Maximum*, in Section 4.2 it can be seen that the disconnect rate also affects the converge time negatively.

Despite having a longer broadcast interval the State-CRDT system is still the worst performing when looking at the *Mean* of the systems. The *Mean* of the Centrally managed system has increased by $\approx 40\%$, this is clear as *Maximum* in addition to the *Median* and *Q3* has increased. As the test has longer converge time compared to the test in Figure 4.6 and there are no more local updates being performed from

the slave nodes, the ratio between broadcasts and local updates change during the difference in converge time where broadcast represents 100% of the sent message. The broadcasts are from the master node and they are containing the entire state which is significantly larger than the local updates of the slave nodes which alter the results in just the way seen here. The *Minimum* and *Q1* remain unchanged, in terms of latency, as the lowest 25% of the messages are still local updates but the broadcasts are a larger part of the total messages sent. The following simplification of $LocalUpdate \leq BroadcastedState$ explains why the *Mean*, *Q3* and *Median* have increased as seen in Figure 4.7. While the *Mean* of the Centrally managed system have increased, for the Delta-CRDT it has decreased by $\approx 1.5\%$ despite having an increased maximum of $\approx 200\%$. It looks like the message latency of the Delta-CRDT system has decreased in all metrics in Figure 4.7 except *Minimum*, remains the same, and *Maximum* as discussed earlier. The minimum is close to unchanged as there is a threshold based on the structure of the messages which makes it impossible to reach a lower latency. Unlike the Centrally managed system the Delta-CRDT system changes the message size ratio to become smaller the longer the test operates as the differences between the nodes decrease with each message received, greatly simplified, as a node updates its state with the received $\delta - state$. As the local updates stop the difference between the nodes state decreases and with that the $\delta - states$ decrease making the messages sent smaller and thus decreasing the overall message latency.

As also mentioned in Section 4.3 the mean is outside of the box in Figure 4.4 and 4.5. The same can be seen in Figure 4.6 and 4.7. The message latency and the message size has a clear relationship, which explains why the Delta-CRDTs mean is outside the box for the message latency as well as for the message size. In addition to the Delta-CRDT, the Reference system and Centrally Managed system means are also outside the box. Both of these systems are centralized and the majority of nodes are slaves which only pass on their local updates which are very small compared to the messages sent by the master node which is the size of the entire state of the system at the time the message is sent. As the master node's messages are much larger than the majority of the messages, seen by looking at the top whisker of the box plot, the mean gets right-skewed or as for our vertical box plots the means are top-skewed.

4.5 Bytes Sent

The bytes sent is the number of bytes that each node has attempted to send in each system during the test. It does not take into account that the sent bytes are received correctly or if the bytes are lost due to a bad connection, they are still added to this metric. The bytes sent metric measures the amount of data that is needed to be sent for each system to reach consistency. The results gathered in this metric differ in magnitude in a way that prevents it from being displayed in a readable manner with box-plots which is why in this section the different metrics of the box-plots instead are displayed in a list to make it readable. The results are from tests with 16 nodes and 5 seconds broadcast interval.

There is a clear difference between the centralized and the decentralized systems which clearly can be seen in all measurements in Table 4.9. The decentralized systems have more than tenfold the amount of any of the centralized systems in all measurements except for the *Maximum* of the Delta-CRDT. The reason for this is the nature of the decentralized systems, the decentralized system pays the price of having higher availability by having increasing information sent in the system. A detailed discussion of this can be found in Section 2.3. The reason for the centralized systems having such good performance for the bytes sent metric is that only one node in the system is sending large amounts of data, the master node. This lowers the availability of the system as a broken connection to the master node makes that specific node disconnected from the system as a whole.

Table 4.9: Bytes sent by 16 nodes with 5 seconds broadcast interval and 1.35% disconnect rate.

	Reference System	**Centrally Managed**	**State-CRDT**	**Delta-CRDT**
Minimum	18.846k	18,846k	108,571M	23,358M
Quarter 1	18.856k	18,858k	123,030M	25,524M
Median	18.883k	18,906k	173,638M	27,042M
Mean	1.512M	2,425M	170,423M	27,015M
Quarter 3	18.910k	19,155k	208,992M	28,067M
Maximum	38.306M	47,078M	240,695M	31,021M

Looking at Table 4.9 all metrics except the *Maximum* value are low when comparing the centralized systems to the decentralized systems. The reason for this is that the measured value of the master node is the *Maximum* for the centralized systems. All the other measurements, except the mean, are values from a slave node. The slave nodes as discussed earlier only send messages when they have received a local update and local updates are single atom states that are sent to only the master. This makes 15 out of the 16 nodes in the test only send local updates and is another reason why the results of the centralized systems are lower than the decentralized systems. There is a small difference in the results between the Reference system

and the Centrally managed system, the *Mean* of the Centrally managed system is $\approx 60\%$ higher than the Reference system. The reason for the mean being higher is mainly because of the *Maximum* being greater, as mentioned the *Maximum* is the bytes sent by the master node. The reason for the master node sending more bytes in the Centrally managed system can be concluded to the results in Section 4.2. The system has a longer converge time which leads to more broadcasts by the master node. The master node does not have any retry mechanism and therefore a dropped message will not lead to a greater number of bytes sent.

The State-CRDT is as in all the previous results the worst performing system, seen in Table 4.9. In the bytes sent metric it is more clear than in any other test. The mean of the State-CRDT system is more than 100 times greater than the Reference system which has the best *Mean* value in terms of performance. The reason for the great difference is the CRDT system is decentralized and because of how it is designed. Each node broadcasts its state to all other nodes in the system. Each node acts as a master node in a centrally managed system, one can see that even when comparing the *Maximum* value of the Reference system, which as discussed is the master node, with the mean value they should be close to similar if the systems would have the same efficiency. However, as discussed in Section 4.2, the State-CRDT has a longer converge time which leads to more broadcasts by each node.

Delta-CRDT outperforms State-CRDT in all measured performance metrics. Delta-CRDT also has a lower spread between the highest and lowest bytes sent by a node. The Delta-CRDT does not have better performance than the Centrally Managed systems when looking at any other aspect than the *Maximum* value. Even though the Delta-CRDT system had a higher converge time than both central systems, the master node in both system has more bytes sent compared to the worst-performing node in the Delta-CRDT despite it broadcasting its state more often. But as the slave nodes in the centrally managed system sent such a small amount of data compared to the master the *Mean* and *Median* value of the Delta-CRDT is worse.

Table 4.10: Bytes sent by 16 nodes with 5 seconds broadcast interval and 5.4% disconnect rate (except State-CRDT which has 15 second broadcast rate because of the 5 second test crashing during testing).

	Reference System	**Centrally Managed**	**State-CRDT**	**Delta-CRDT**
Minimum	18.846k	18,844k	52,395M	28,233M
Quarter 1	18.856k	18,861k	76,869M	29,881M
Median	18.883k	18,911k	88,763M	31,092M
Mean	1.512M	3,060M	83,906M	30,884M
Quarter 3	18.910k	19,117k	93,575M	31,877M
Maximum	38.306M	57,742M	99,447M	33,820M

When comparing the result in Table 4.10 the centralized systems are still outperforming the decentralized systems. The Centrally Managed system has an increased *Maximum* value with $\approx 20\%$ which in turn has increased the *Mean* since it can be seen that the rest of the data are approximately the same values. The slave nodes send single atom states so their messages are small, thus the effect of a bad connection is less noticeable compared to the master that broadcasts larger states. There is also an increase in converge time which indicates more broadcasts by the master node leading to more bytes sent while the slave nodes maintain the same number of messages sent because of just sending local updates which are a fixed number that is the same for all tests.

Continuing by looking at Delta-CRDT system all results have increased compared to Table 4.9, the main reason for this is the increase in converge time like discussed above. The *Mean* still remains ≈ 10 times higher than the Centrally Managed system but the difference of *Maximum* value has increased with $\approx 8M$ bytes. The difference of bytes sent for any node in the Delta-CRDT system has increased in the same manner as the master node in the Centrally Managed system. The Centrally Managed system still has better performance when looking at the *Mean* and *Median* but the Delta-CRDT system has not lost the same amount of performance when comparing the *Mean* in Table 4.9 to Table 4.10 which may suggest that at an unknown disconnect rate, higher than 5.4%, the Delta-CRDT could surpass the Centrally Managed system.

The results from the State-CRDT system is from a test with a 15 second broadcast interval unlike the rest of the results which only have 5 second broad cast rate. The reason for using that test for the State-CRDT system is that data could not be stored during the test as the computer crashed and the data was lost. The results are not the desired but they can still be used to get a rough estimation of the performance. Even with lower broadcast interval the State-CRDT still has the worst performance in all measurements. The *Mean* is $\approx 166\%$ higher than Delta-CRDT, and over 26 times higher than the Centrally Managed system. Because of each broadcast containing the entire state the total bytes sent increase rapidly compared to the other systems. Combining this with the results of the converge time in Section 4.2 explains why the system has a mean that is much higher than the other systems. Making a rough estimating what the mean could be if it had the same broadcast interval as the other systems it would send three times more messages. This would give a *Mean* that is three times higher, not taking into account that the network will be slowed down further, compared to the 15 second broadcast interval system. Comparing the rough estimation of $estimate \geq 3 \times Mean$ and the rest of the systems it would be ≈ 10 times worse than Delta-CRDT and ≈ 100 times worse than the Centrally managed system.

To summarize when comparing the mean, which gives a fair image of the performance, of the systems it is clear that the decentralized systems perform worse in the amount of data sent during the tests. Each node in the decentralized system acts as the master node in the centrally managed systems, more broadcasted states leads

to higher total bytes sent within the systems. All systems had worse results with a higher disconnect rate, one definitive reason for this is the increase in converge time which leads to more broadcasts. The State-CRDT had the worst result out of all, even when being compared with a broadcast interval three times slower than the other systems the State-CRDTs perform the worst.

4.6 Merge Latency

The merge latency metric is the time it takes for a node to perform an operation. Operation is either an external merge between a received state or a local update and its local state. A local update is a state with a single atom so the system handles it as a merge. The local updates simulate the machines performing actions and owing to this their state is updated with a single atom. An important note is that in the centralized systems the slave nodes do not directly perform the local updates received locally. When received they are instead being forwarded to the master node where they are merged and then distributed across slave nodes. As mentioned before the results displayed in this section is also an outtake of the total results and are focused on the test with 16 nodes and 5 seconds broadcast interval.

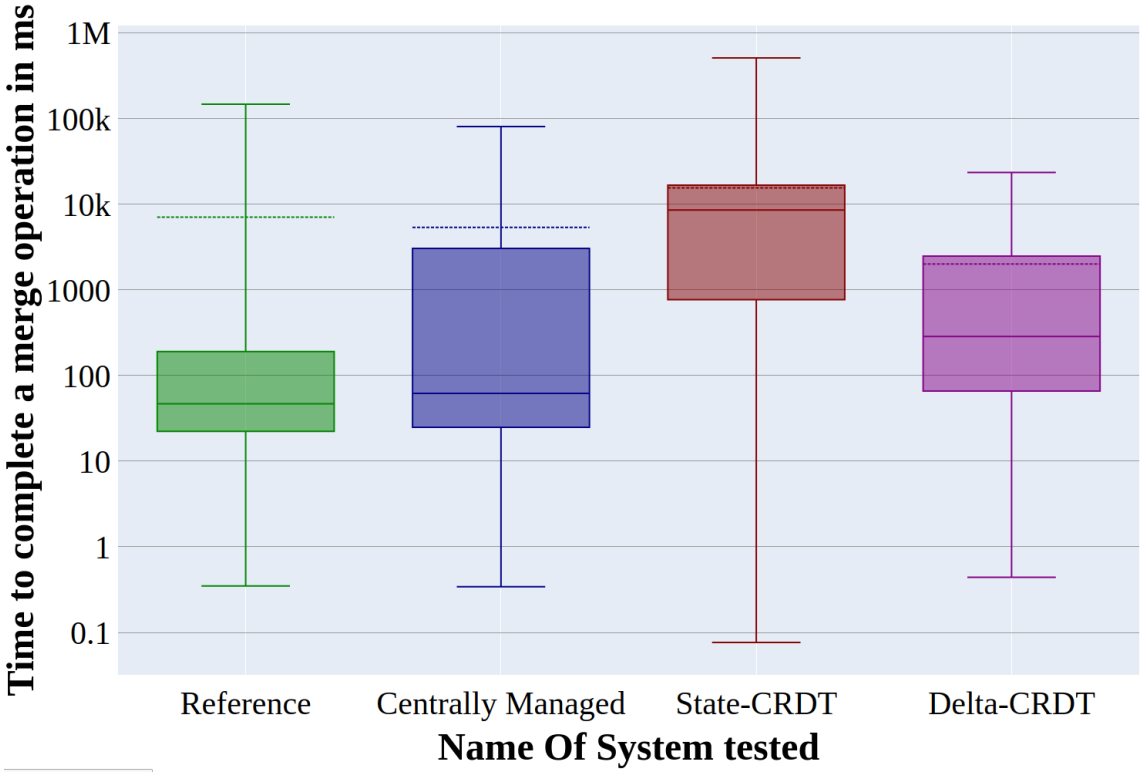


Figure 4.8: Latency to perform a merge operation in systems with 16 nodes and 5s broadcast interval and 1.35% disconnect rate. The boxes include 25-75percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

Looking at the results gathered where the disconnect rate is 1.35%, the mean (dotted line) of the Delta-CRDT system is the lowest of all the systems as seen in Figure 4.8. The Reference system is the only system in this figure that does not have any disconnects present across its nodes. Despite this it has the second highest mean merge latency, only system with worse mean is the State-CRDT system. The State-CRDT system was as expected the worst in mean and it has the highest merge latency out of all the systems. This is because all nodes in the State-CRDT system broadcast their entire states with each broadcast which overflows the network with data which in turn create larger merge operations. Each node receives $N - 1$ states, assuming no nodes are disconnected, where N is the number of nodes in the system. Each state requires a merge operation. The reason for the Delta-CRDT system having the lowest mean is because of the δ-state. The δ-states only contain information not already possessed by the node that sent the snapshot. Therefore the δ-states are smaller than the state sent by nodes in the State-CRDT system and the master node in both centrally managed systems. Thanks to this Delta-CRDTs have the lowest max value, closest is the Centrally Managed system which is ≈ 3.5 times higher. It shows clearly on the *max* value as a smaller state requires less time to merge. This can also be seen in the difference between *min* and *max* which is explained by the difference in state size across the nodes in the system. In the beginning of the test the states are small and thus making for fast merge latencies.

Looking at the boxes in Figure 4.8 it shows that the Reference system has the lowest median as well as q3 out of all system, since it is a centrally managed system the reason for the majority of the operations, $> 75\%$, are below 500 ms compared to the State-CRDT which has $> 75\%$ greater than ≈ 700 ms. This is also reflected by the low median of the Reference system. The Centrally managed system also has a lower median than the CRDT systems. Both the Reference system and the Centrally managed system use the same node network hierarchy which explains these results. The same number of local merge operations are performed in all systems but the amount of merge operations with the received states (state merge), which are always the same or larger than the local merge, differ in the different systems. In the centrally managed system it is performed $N - 1$ state merges each broadcast while in the CRDT systems, which are decentralized systems, $(N - 1)^2$ state merges are performed which leads to a higher median.

The minimum value of all the systems are less than 1 ms, the small difference in minimum merge latency could be a difference in background load on the system by the operating system or another background software. The State-CRDT system has the largest range between the *max* and *min* value, followed by the Reference system. The Centrally managed system has a lower *max* value than the Reference system which is unexpected, but comparing the the max values of the CRDT systems in Figure 4.8 and Figure 4.9 there is a pattern that a higher disconnect rate decreases the *max* values of the systems. The reason for this could be that the all parts of the simulation is done in software. The network and all the local computations by

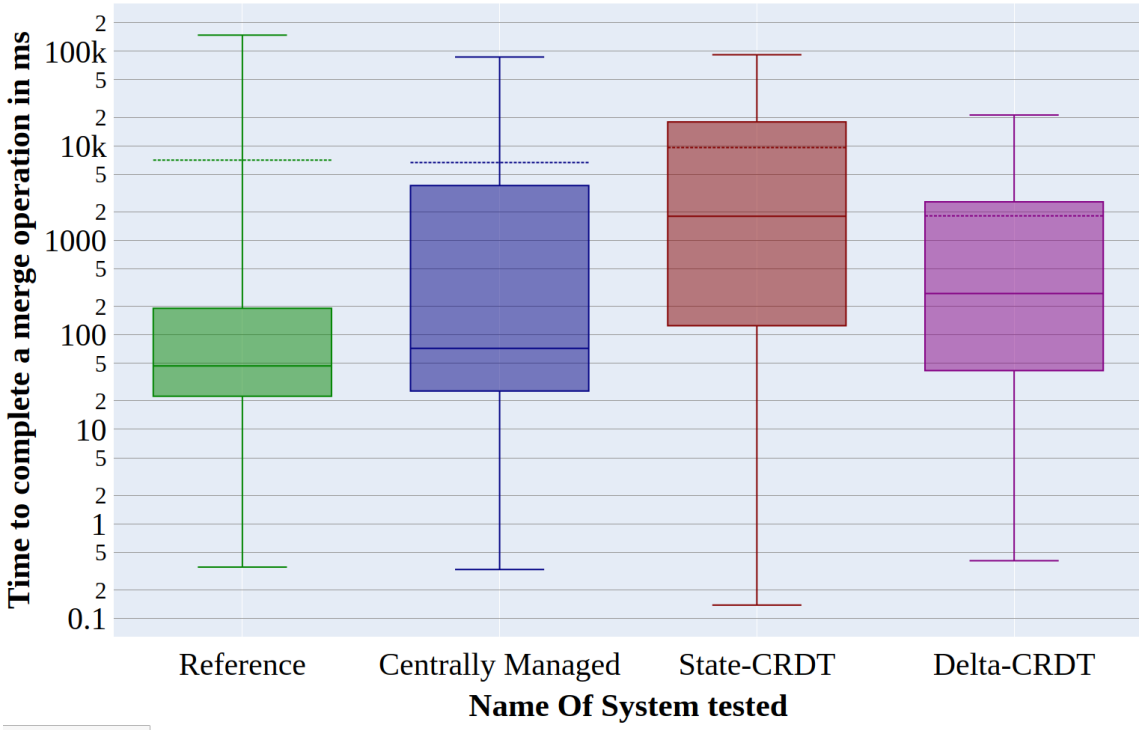


Figure 4.9: Latency to perform a merge operation in systems with 16 nodes and 5s broadcast interval and 5.4% disconnect rate (except State-CRDT which has 15 second broadcast rate because of crash during testing). The boxes include 25-75 percentiles, the whiskers are max and min with outliers included and the dotted lines represents mean and solid lines mean.

nodes are sharing CPU and memory. When a connection disconnects it takes load of the system which increases the available resources for the rest of the system. This would also explain the reason that the Reference system has a higher *Mean* merge latency than the Centrally managed system.

With the higher disconnect rate seen in Figure 4.9 the biggest change in the results are the *max* value of the State-CRDT system. In the test with the lower disconnect rate the State-CRDT had a *Maximum* of $\approx 500k$ and with the higher disconnect rate in Figure 4.9 that value is $\approx 100k$. The reason for the decrease for the State-CRDT is the same as discussed earlier, the broadcast interval for the State is not 5 seconds but is instead 15 seconds. With the lower number of messages the overall load on the system is decreased leaving more CPU processing power for merging which decrease the merge latency as the latency is measured in real time and not in logical time.

The Centrally managed system has minimal changes in Figure 4.9 compared to Figure 4.8. All values have increased except the minimum value which remains approximately the same unlike the CRDT systems which see a decrease in all measurements in the box plot. The difference between the CRDT systems and the Centrally managed system is that they are of the decentralized replication model.

They are sending more messages compared to the Centrally managed system which in theory should slow down the system and flood the network. With higher disconnect rate more messages are dropped and that results in a lower load on the system which gives faster operation speed as the operations are measured in real time when comparing the same system with higher and lower disconnect rates.

Both Figure 4.8 and 4.9 display mean values that are skewed as in Section 4.3 and 4.4 for the two centralized systems. The Delta-CRDT does not have a skewed mean for the merge latency metric. The maximum merge latency for the Delta-CRDTs are the lowest out of all the systems, seen at the top whisker in Figure 4.8 and 4.9. Because the maximum of the Delta-CRDT is closer to the upper quartile of the box the mean is within the box. The two centralized systems both have low median and upper quartile compared to the upper whisker. Because the master node receives all local updates individually from all the slave nodes it performs a high amount of merge operations with single atom state which provide a small merge latency. The slave nodes perform merge operations with the entire states of the system that is received from the master node. These two factors are the reason for the difference that is clearly seen in the result in Figure 4.8 for the Reference system. 75% of the recorded latencies of merge operations for the Reference system are below 200 ms and the maximum latency measured ≈150k ms. The maximum latency for a merge operation represents 750 merge operations at the upper quartile for the Reference system. Because of the nature of the centralized systems the mean for the merge latency is top skewed.

The order in which the *Mean* distribution does not alter with higher disconnect rate as seen in Figure 4.9 compared to Figure 4.8. Delta-CRDTs maintain the lowest mean, followed by the Centrally managed system, the Reference system, and lastly the State-CRDT system. The significant change in mean can be seen in the State-CRDT which has decreased $\approx$ 50% from $\approx 20k$ in Figure 4.8 to $\approx 10k$ seen in Figure 4.9. The reason for this was discussed earlier with the State-CRDT having a different broadcast interval in Figure 4.9. The Centrally managed system has a $\approx$ 15% increased *Mean* while the Delta-CRDT has a decrease with $\approx$ 10%. As the size of the state on the master node grows the time it takes for a slave node to perform a merge operation increases. With a longer converge time more broadcasts are performed by the master node and thus the slaves are performing more merge operations with larger data sets than the average seen in Figure 4.8. The Delta-CRDT, on the other hand, performs merge operations with the δ-state, with longer converge time more δ-states are merged, and with each merge, the next δ-state will decrease in size until it stops receiving δ-states when convergence is reached between two nodes. With smaller δ-state over time the merge latency also decreases with longer converge times as is seen in Figure 4.9.

To summarize, State-CRDT does not perform well in the merge latency metric either, the system sends big states which take too long to merge compared to the other systems and the system being decentralized is not helping for this metric. The Reference and the Centrally managed system are merging many small messages

which provides a lower median than the decentralized systems because the master node performs merges for every single atom local updates the slave nodes send. The Delta-CRDT performs the best when considering the *Mean*, because of the $\delta - state \leq State$ the merge operations are faster than they would be for a slave node in a centralized system or any node in the State-CRDT system. Provided this the *Mean* for Delta-CRDT is the best in both Figure 4.8 and 4.9.

4.7 Quality Arrays

Quality arrays show the amount of lost messages and which messages are lost in addition to if messages are lost in sequence or individually. Figure 4.10 and 4.11 show visual representations of which messages a node has either failed to receive (0) or received (1). Each line represents messages that were received or failed to be received from a particular node in the network. Each column represent a broadcast round. The system is either "in sync" if all nodes are in the same state or "out of sync" if one or more nodes' states differ.

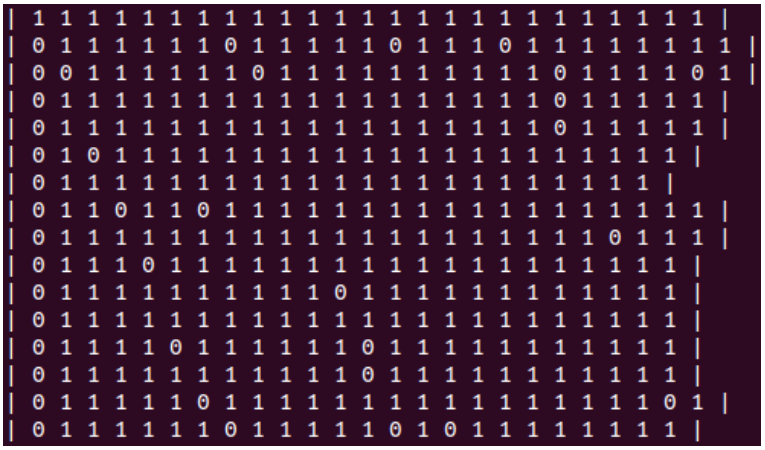


Figure 4.10: Quality arrays created by a node where each line is an array of messages received or not received from other nodes in the system. 1 is received message while 0 is lost message. State-CRDT system, 16 nodes, 15s broadcast interval, 5.4% disconnect rate.

Since its not reasonable to show all results from all tests the results will be summarized. Unsurprisingly a higher disconnect rate leads to more lost messages throughout all the tests which in turn leads to the nodes diverging. When looking at the converge times in for example Table 4.8 in Section 4.2.4 it can be seen that the tests with 16 nodes, 15 seconds broadcast interval and 2.7% and 5.4% disconnect rate have increased converge times, 334 and 367 seconds respectively. However, when studying the quality arrays in Figure 4.11 it can be seen that the node that created the arrays (top row or node 1) have been trying to send more broadcasts than it has received from other nodes. This means that these results are again affected of the network being flooded with too many messages. Another indicator of this can be seen by looking at the converge time of the tests in Table 4.8 with 16 nodes, 25

seconds broadcast interval and 2.7% and 5.4% have the normal 300 seconds converge time. The longer broadcast interval have enabled the network to keep up while the converge time is unaffected by the disconnect rate.

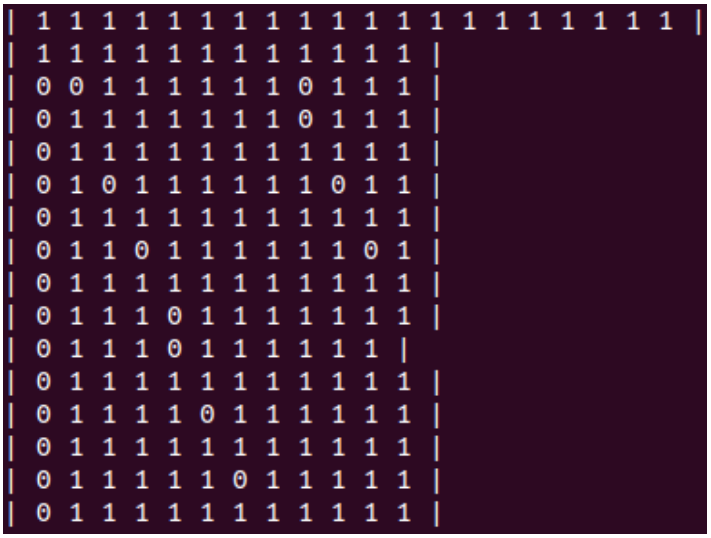


Figure 4.11: Quality arrays created by a node where each line is an array of messages received or not received from other nodes in the system. 1 is received message while 0 is lost message. Delta-CRDT system, 16 nodes, 15s broadcast interval, 5.4% disconnect rate.

The same applies to Figure 4.10 though in this case the reason the converge times are longer is because the messages the nodes in the State-CRDT system send are too large and therefore overloading the network. In both Figure 4.10 and 4.11 it can be seen that the first column contains a majority of lost messages (0). This is because this specific node has disconnected during this time and missed all or most of the first broadcasts from the other nodes in the system.

On a more general note, as the nodes are all sending their messages at the same time a message not received by a node means that the system is out of sync for at least one round. The slave nodes in the Centrally Managed system will try to resend their messages until they succeed so on the master node it will look like all messages were received without fail. However, if a slave node disconnects it will fail to receive the states sent by the master node. Furthermore, the other nodes will not receive any updates from the disconnected slave node. Once the disconnected slave node reconnects and manages to receive a state it will be in sync with the rest of the system again after just a single round due to the master node sending the whole state. This means that it takes one broadcast round for the slave nodes to be back in sync after a disconnect period has happened. In Table 4.11 the maximum dropped messages in sequence for the Centrally managed system is 2, with the broadcast rate of 15 seconds the time for the slave nodes to be back in sync would therefore take $15 \times 3 = 45$ seconds.

Table 4.11: Summary of the gathered results from the quality arrays. This is a representation of the quality array displaying (the maximum percentage of drop messages by a node, maximum number of dropped messages in sequence by a node). Tests are done with 16 nodes and 15 second broadcast rate for all systems.

	1.54% disconnect rate	**5.4% disconnect rate**
Centrally managed	(0.4%, 1)	(6.6%, 2)
State-CRDT	(3.7%, 2)	(9.2%, 2)
Delta-CRDT	(1.9%, 1)	(12%, 2)

The State-CRDT system and the Delta-CRDT system also only require a single round to be in sync if certain criteria are met. The criteria consist of at that least one node, N, in the system has received messages from all other nodes during a broadcast round and N is able to send its state during the following broadcast round to all other nodes M with the following property $M \neq N$ inside the system. These criteria are in fact the same required for the centrally managed systems as in the CRDT systems when the node N represents the master node. Expanding on this reasoning it can be seen that as long as some node in the CRDT systems receive a state from a node and is able to propagate during the following round the system will stay in sync. This creates redundancy for when nodes disconnect in the middle of a broadcast round meaning some nodes not receiving the broadcasted state.

Building on this the results shown in Table 4.11 show the maximum dropped messages in sequence for the Centrally managed system and the CRDT systems are two for 5.4%. For the Centrally managed system this would result in the system being out of sync for 45 seconds, the time for three broadcast rounds to occur. The CRDT system on the other hand has redundancy as discussed and will because of this return to a synced state faster than the Centrally managed system as the CRDT system will be out of sync for ≤ 45 seconds.

One major difference between how the Centrally Managed system and the CRDT systems diverge is that the nodes in the CRDT systems apply their own updates to their own states directly. The slave nodes in the Centrally Managed system require the master node to apply the update. This means that the nodes in the CRDT systems are more prone to have diverging states thus being out of sync. However, this is a crucial technicality for the CRDT systems to function as they are decentralized systems with no single entity to decide on a state such as in the Centrally Managed system.

5
Discussion

Starting this chapter is Section 5.1 the discussion of the questions that were asked at the beginning of the project. In Section 5.2 the particular design of the Delta-CRDT with Snapshots in this project is discussed upon as it differs from the proposed solution by Almeida, Shoker and Baquero [4]. Following is Section 5.3 where the main hurdles for CRDT systems are being discussed. Lastly is Section 5.4 which is thoughts on improvements that can be made in order to boost performance of the CRDT systems.

5.1 Discussing the Questions Asked for this Project

At the beginning of this project there were questions asked, stated in Section 1.2. Here we present reflections of the results presented in Chapter 4 based on previously mentioned questions.

5.1.1 Can CRDTs be used in a real-life environment with large data sizes and low number of updates on a low bandwidth network?

Technically the State-CRDT can be used but definitely should not. Looking at the results from Table 4.9 it can be seen that the State-CRDT sends multiple times more data than the Centralized system and the Delta-CRDT. The huge amount of data sent by the State-CRDT together with the broadcast rate is the reason the network becomes overloaded. The overloaded network results in extreme converge times for the State-CRDT with some tests never converging at all which can be seen in Table 4.7. Even though there are configurations that allow the State-CRDT to converge with reasonable times, the ever growing states of the State-CRDT will eventually cause these configurations to also overload the network.
The Delta-CRDT is a viable option as its performance wise far superior to the State-CRDT and is also better than the Centralized system on all metrics except total bytes sent and converge time. The complexity of the Delta-CRDT is higher than the State-CRDT. However, the Delta-CRDT in this project uses snapshots which keeps the difference in complexity compared to the State-CRDT to a minimum. This is expanded upon in both Section 5.1.4 and 5.2.
Despite the Delta-CRDT having better performance than the Centralized system currently in use in the particular real-life scenario this thesis is based upon it would not perform better in practice. This is because the vehicles in the real-life scenario

does not have vehicle-to-vehicle (V2V) communication. Using CRDTs would not provide an improvement, especially as all communication goes through an offsite cloud. CRDTs together with V2V communication would create redundancy and create ways for data to propagate even to vehicles that are not connected to the cellular network. If the cloud service becomes unavailable the replicas will not converge until the cloud service is running again even if CRDTs are in use.

The propagation of data via V2V communication would be preferable in a setting where a cellular network does not cover everywhere. For example, if a vehicle is stationary in a place where there is no connection to the cellular network CRDTs in combination with V2V communication would be ideal. The stationary machine can operate freely while mobile vehicles that occasionally are connected to the network or other vehicles can propagate new information to the stationary vehicle when they get in range of V2V communication.

5.1.2 Which type of CRDTs are suitable for this environment?

Of the two different fundamental types of CRDTs, operation-based and state-based, the operation-based CRDT was rejected during the research stage of the project which is explained in Section 3.2. The sub-types of state-based CRDTs, in this project called State-CRDT (explained in Section 2.6) and Delta-CRDT (explained in Section 3.2.4), are both technically viable. However, considering the performance only the Delta-CRDT is actually viable following the reasoning in Section 5.1.1.

5.1.3 How does the performance of the CRDT systems compare to the existing system?

This is presented in detail for each tested metric in the corresponding subsection in Chapter 4. The State-CRDT is as expected the worst system and floods the network with its many and large messages. The converge times in Table 4.7 clearly display how bad the State-CRDT performs compared to the Centralized system depicted in Table 4.6. The Delta-CRDT shows impressive means in all metrics except the total number of bytes which is an effect of it being a decentralized system. Furthermore, the converge time of the Delta-CRDT, Table 4.8, is worse than the Centralized system. The Delta-CRDT has improved availability compared to the Centralized system at the cost of longer convergence time. This is as mentioned further presented in Chapter 4.

5.1.4 What are the trade-offs between the different types of systems?

As always with distributed systems there are trade-offs to be made in the system as discussed in Section 2.4. The Centralized system sacrifice availability for simplicity in implementation and low communication cost. CRDTs focuses on high availability but at the cost of communication cost. It is clear that the CRDT systems send

more data compared to the Centralized system by looking at Table 4.9 and 4.10, but they are decentralized and have higher availability by allowing local updates to always be performed on the system despite not having connection. Furthermore, by being decentralized it does not have the single point of failure that is present in the Centralized system. If the master node disconnects in the Centralized system no local updates can be added to the state, in addition no slaves would receive any updates from the master node and thus gain no new information about what has happened in the system.

There are trade-offs between the State-CRDT and the Delta-CRDT as well. The State-CRDT is more robust to losses compared to the Delta-CRDT but it is far more demanding in communication. The Delta-CRDT send a higher amount of messages back and forth by sending snapshots combined with δ-states which increases the chances to drop a update from another node. If either the snapshot or the response is lost the entire process is dropped as the implementation does not re-send any dropped messages which is further discussed in Section 5.3. Looking at the number of dropped messages in Table 4.11 Delta-CRDT has the highest number of dropped messages at highest disconnect rates (5.4%) on the network. In no test it has been seen any node miss three or more messages in a row for any system. Taking this in consideration combined with the data about the converge time of the Delta- and State-CRDT depicted in Table 4.8 and 4.7 it is clear that the weakness of the State-CRDT does not outweigh the weakness of the Delta-CRDT. The Delta-CRDT still function as intended despite the higher disconnect rates, the losses does not affect the performance in the same manner as the communication cost does for the State-CRDT at higher disconnect rates.

The focus was in the real-life scenario to increase the availability within the system. As discussed in Section 5.1.1 the vehicles did not have the ability to use V2V communication which mitigates the point of using a decentralized system instead of a centralized system. Considering that there were no hardware limitations and the vehicles actually had V2V communications the Delta-CRDT would be an option if the availability of the system was higher valued than consistency. The Delta-CRDT as mentioned eventually converge, not as fast as the Centralized system, but it converges for all tests independently of disconnect rates and provide higher availability by always being operational even without connection on top of not having a single point of failure.

5.2 This Project's Delta-CRDT with Snapshots Design

In the solutions that Almeida, Shoker and Baquero propose, previously explained in Section 3.2.4, in their paper [4], the δ-mutators they use produce delta-states whenever local updates are made which are then broadcasted either instantly or accumulated and sent as a group. States that are received are directly propagated to other nodes. This propagation requires nodes to reason about history and creates the

need for garbage collection of messages. By utilizing state snapshots there will only be information being transmitted over the network that nodes do not posses with the exception of two or more replicas are at the same state. When $A_{state} = B_{state}$ and both receive a state snapshot from a node C it will result in $\delta_{snapshot_A} = \delta_{snapshot_B}$. C will receive the same delta-state twice. Furthermore, since a state snapshot may result in an empty delta-state nothing has to be sent at all, keeping the load on the network to a minimum and allowing other services to use more of the network.

The use of snapshots does however still increase the complexity of the Delta-CRDT. While reasoning about history and garbage collection is not needed there is added complexity through the calculation of what is to be included in the delta-state when a snapshot has been received. Due to the simple database structure used in this project calculating the delta-state is simple. If the database structure has relations between entries and tables the calculation immediately becomes much more complex. Although not implemented in this project, the snapshots have further potential to be used to measure the level of congestion on the network in addition too lowering the amount of data being sent. By adding a timestamp to the snapshot the receiving node can easily see how long it took for the snapshot to be sent and subsequently determine if sending a delta-state should be delayed.

5.3 CRDT Implications

As the CRDT systems use grow-only sets, to keep complexity low, the total amount of data that is being sent between nodes, even with the Delta-CRDT system, puts more strain on the network compared to the Centralized system. The amount of data that the CRDT systems send grows with the size of the states. The Centralized system sends either just a single update or as the master node, sends its entire database to slave nodes, similar to the State-based CRDT, though in this case there is only one node. If the state of each replica is not continuously reset or minimized the performance of the system will diminish as the states grow leading to more and more messages being lost due to the unreliable network.

The risk of losing connection while trying to send a message increases with the message size assuming that the bandwidth remains constant. The slave nodes in the Centralized system will try to resend the same message multiple times if the message is lost. This is done because if the message is never received at the master node that update will never be added to the state of the system. For the other systems, including the slave nodes of the Centralized system, the order of messages received does not prevent convergence. State-based CRDT and Delta-CRDT performs a non order dependent merge, explained in section 3.2.3. The slave nodes does an update similar to State-CRDT which is also order independent. All *order independent action* messages sent are not re-sent in case of a connection loss. The next message sent will be the new state which, if it is a different state, always contains the previous state. This was a design choice, we felt that there were no point in re-sending an "old" state. The other option would be to retry until the state is sent which would take up resources.

The bottleneck created because of the network for State-CRDTs should be considerate and outweigh the efficiency of the merge. Because of this we did not expect that the performance of the State-CRDT would be anywhere near the Centralized system. The results show exactly what we anticipated, when implemented in a setting where the states are big the State-CRDT perform badly. They perform better at simple tasks as counters.

5.4 Future Work

5.4.1 Possible CRDT Optimizations

There are several improvements that can be made on both the State-based CRDT and the Delta-CRDT in order to reduce the congestion on the network, shorten merge time as well as optimize how data is propagated on the network.

A potential optimization to decrease message size for State-and Delta-CRDTs and to reduce the number of dropped messages are to split the state up into different smaller states. As each node has the state of all the other nodes in separate databases. When the sending node queries its databases to get its full state, it will instead send the query from each individual database combined with empty states from the rest, Figure 5.1 shows an illustrating of how the state can be split. The receiving node will perform a merge in the same way without any modifications or failure.

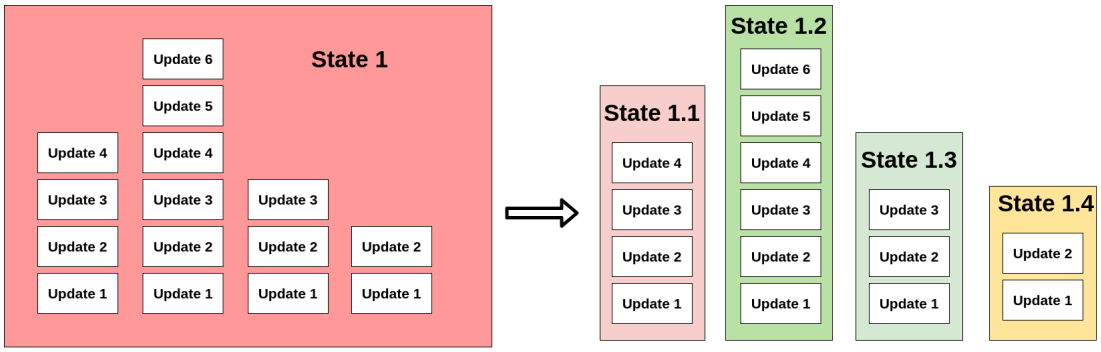


Figure 5.1: Splitting of a large state to gain smaller states to decrease message sizes before sending across the network.

In terms of performance we believe this state priority would give a performance increase in terms of number of message lost due to connection loss and the time it takes to deliver a message. As shown in Figure 5.1 a demonstration of how a combined state containing 4 different nodes states could be split up into 4 smaller states that can be sent across the network to gain higher success rate of delivery.

To add a form of prioritization of what data to replicate each node can save the last state it broadcasted. The next state to be sent is then compared to the saved state

to see which sub state, of the ones mentioned above and illustrated in Figure 5.1, have the most new updates. The sub states are then sent in the order of which sub state had received the most updates since the last broadcast i.e. the most urgent update.

By keeping the last sent state you get a way of prioritizing which information is most urgent to send. A drawback of the implemented system is that the sender is not keeping track of which message is dropped and what destination that message had. The CRDT systems simply broadcast their states and hopes the messages are received. By looking for changes between the last state sent and the present state there is the possibility that the priority of states are miss calculated due to the fact that the node does not know which states the receiver have received earlier. If the system is designed to keep track of what message is dropped to which node, the system could save one *last state sent* for each node and only update that state when it knows the message has been received.

Another improvement for the CRDT systems were thought of when looking at the results where the network often got congested where the number of nodes and the broadcast rate were the big factors to the congestion. Systems where the number of nodes is unpredictable will make it hard to find a good balance between how often nodes should broadcast their states or snapshots to not congest the network while also not having replicas diverge too much leading to longer merge times. A solution to this, with inspiration taken from how TCP works, would be for the nodes to dynamically change how often they broadcast by checking how congested the network is. When there is congestion the broadcast rate would decrease until there is less congestion.

5.4.2 Designing Databases to Suit CRDTs

The design choice of having a database for every node on all nodes, meaning that there are N^2 databases in the system where N is the number of nodes, gave way to snapshots being able to be used and acted as a performance improvement but also created quite a bizarre scenario where nodes would possess data that couldn't be used. The reason behind this was relations between the tables in the original structure. A node A could want to make an update that should have a relation to an entry in a table in a database A_B that A possess. Because of the rule that A only is allowed to make updates to A_A and there not being any functionality for cross database relations this would result in A knowing something which it is not allowed to use. To resolve this would require an immense increase of complexity when merging in order not to break the rules and thus the system.

The performance improvement of having multiple databases on each node is because this results in multiple grow-only sets (one database is a set). When merging a state each entry need only be checked against one set instead of N sets i.e. having all information in one database. Multiple databases together with the structure of the states being sent between nodes would allow to safely operate the merge in

parallel with one thread for each database this taking advantage of modern multi-core processors.

Designing the CRDTs at the same time as the database structure opens up for making the CRDTs have the best performance possible. In this thesis we received a database structure not designed for the usage of CRDTs, as the goal of the thesis was to try the systems on a real-life environment we tried to make the CRDTs fit the database instead of creating a database to fit the CRDTs, this was very time consuming and ended up slowing down the progress of the implementation. If we would redo the project we would create the CRDT system and then make a new database to fit the CRDT system based on the columns in the old database.

6
Conclusion

This research aimed to examine the viability of replicating data by applying CRDTs in the same environment as an already existing centralized system by answering the following questions: Can CRDTs be used in a real-life environment with large data sizes and a low number of updates on a low bandwidth network with connection losses? Which type of CRDTs are suitable for this environment? How does the performance of the CRDT systems compare to the existing system?

State-CRDT was the best type of CRDT in an environment with unreliable network. Two different types of CRDTs were studied, State-based and Operation-based. After considering the data in the real-life system the decision was made to use a State-Based CRDT design. The decision was made because an Operation-Based CRDT would only emulate the State-CRDT and also requires messages to be delivered in correct order. Due to the network being highly unstable this not be guaranteed and thus was not an option for this environment.

Delta-CRDT as a complement to State-CRDT after gathered real-life data present weakness for State-CRDT. In order to test the CRDT systems in a simulated version of the real-life scenario, connectivity and data about operation frequency was collected from vehicles operating inside the Chepstow quarry. The gathered data was used to create a virtual network topology with connectivity issues and limited bandwidth similar to the real-life scenario. Furthermore, a script was created mimicking the behaviour of the vehicles operations to gain accurate behaviour for the systems. Evaluation of the performance of the systems was made by measuring the following; message latency, merge latency, converge time, bytes sent, message size and quality arrays. However, as the amount of data was large it was predicted that State-CRDTs would not be sufficient. Therefore, a Delta-State CRDT was created to mitigate the weakness of the State-Based CRDT's, which is large state sizes.

The State-CRDT system present shortcomings in all metrics and should not be considered a valid replacement to the Centralized system in this scenario. As expected before viewing the results State-CRDTs does not perform well where the state grows quickly in size compared to the other systems. State-CRDTs sends the entire state with each message, Figure 4.4, which in turn puts heavy load on the network and higher message latencies as seen in Figure 4.6. Sending entire states also effect the merge latency of the system shown in Figure 4.8, as each merge operation checks all elements in the received state for changes. One

flaw in the Reference system is the lack of availability, a problem that the State-CRDTs solves but at the cost of sending more messages and as mentioned above, large messages affects the total bytes sent in the system negatively as seen in Figure 4.9. The effect of the State-CRDTs being inferior to the Centralized system in all measurement is clearly seen when comparing the converge times in 4.7 and 4.6. To conclude the performance of the State-CRDTs it is lesser than the Centralized system in the order of it not being applicable in this scenario and maintaining desired functionality.

The Delta-CRDT performed better than the Centralized system in most of the metrics with the exception of converge time and total bytes sent. The Delta-CRDTs has the property of sending snapshots with the least amount of information possible to inform other nodes of its current state and then only sending the required information to reach convergence unlike the State-CRDT. With that said the Delta-CRDT outperforms the State-CRDT system in all metrics by quite a margin as seen in Figures 4.8, 4.6, 4.4, and in Table 4.9 and 4.8. Compared to the Centralized system the Delta-CRDT shows both pros and cons. Because of the δ-states and the snapshots the average message size of the Delta-CRDT is lower ($< 50\%$) than that of the Centralized system, despite the Centralized system having a lower median message size as depicted in Figure 4.4. In similar fashion the mean message latency have the same relation for the Delta-CRDT in relation to the Centralized system ($< 50\%$) as illustrated in Figure 4.6. Which is as expected as there is a correlation between these metrics as discussed in Section 4.6. Even though the Delta-CRDT has on average smaller message sizes compared to the Centralized system the total bytes sent is significantly higher for the Delta-CRDT as evidenced in Table 4.9, this is one of the sacrifices the decentralized systems make to be able to maintain high availability by sending more messages which explains the higher total bytes for both CRDT systems compared to the Centralized system. The merge latency of the Delta-CRDT only takes into account the δ-states, it does not include snapshots unlike the other metrics, when a snapshot is received the merge operation is not performed. Despite the snapshots not being included in the result the Delta-CRDT still surpasses the Centralized system in mean merge latency as detailed in Figure 4.8. The Centralized system maintain a lower median also in this metric because the nature of the Centralized system where all slaves send their local updates, single atom states, to the master which merges these states. With such good results for the Delta-CRDT compared to the Centralized system it was very unexpected to see the results in the converge time metric. The Delta-CRDT system has $\approx 1000\%$ longer converge time than the Centralized system for the test with the highest disconnect rate and highest number of nodes as seen in Table 4.8 and 4.6, when excluding the 300 seconds the local updates are being performed. To summarize the performance of the Delta-CRDT, the Delta-CRDT has great mean values in merge latency, message latency and message size compared to the other systems but under perform when it comes to the time it takes to converge and the total amount of data sent.

Delta-CRDT gets better merge and message latency means with higher disconnect rates while the other systems display higher means for all metrics. When testing the systems with different disconnect rates, there was a clear pattern across all the different systems, higher disconnect rate lead to higher means in all metrics in addition to longer converge times. However, the Delta-CRDT system did not show higher means for merge and message latency. The reason for the Delta-CRDT displaying better mean for these metric are the longer converge time combined with the δ-states. This is due to the δ-states are smaller the less divergence there is between the nodes. As the local updates stop occurring after a fixed amount of time the divergence shrinks the longer the test continues, thus without local updates that make the states diverge the δ-state will shrink over time and provide better merge and message latency in a unique way for the Delta-CRDT compared to the other systems.

To conclude, If availability is essential in the system the Delta-CRDT could replace the current Centralized system unlike the State-CRDT which does not maintain enough performance. As always with distributed systems there are trade-offs depending on which kind of system you choose to implement as discussed in Section 2.4. The results show that CRDTs can be applied to this real-life environment but with great limitations for the State-CRDT system seen in Figure 4.7 and 4.9. Looking at the converge time in Table 4.7 it is clear that the State-CRDT system is not a viable option as the test could not be completed with the most stressful environment settings compared to the rest of the systems. The Delta-CRDT system is able to converge for all tests as seen in Table 4.8. The performance of the Delta-CRDTs is proven to be able to maintain performance compared to the Centralized system in most metrics even at higher disconnect rate this can be seen in Figure 4.7, 4.9 and 4.5. Higher disconnect rate is a weakness of the Delta-CRDT compared to the State-CRDT as there are more messages sent that can be lost but the Delta-CRDT instead has better communication cost, as evidenced in Table 4.10. The flaw of the CRDT systems is the time of inconsistency (converge time) that is the trade-off for availability further discussed in Section 5.1.4, in Table 4.8 we can see that for all tests independently Delta-CRDT reach convergence. To be able to answer the question if Delta-CRDT can replace the existing system one must first decide if availability or consistency is most crucial. If availability is required Delta-CRDT can replace the current system and the system will always eventually converge, but if convergence is crucial the current system should be considered the best option even if the Delta-CRDT eventually converge.

Bibliography

[1] CPAC Systems AB. "Image from Rock2Road service by CPAC Systems AB, consent has been given by CPAC systems to use image." Göteborg, 2019.

[2] W. Lloyd, M. Freedman, M. Kaminsky, D. Andersen. "Don't settle for eventual". In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles - SOSP '11*, page 401, New York, New York, USA, 2011. ACM Press.

[3] "ASU - Arbiträr styrkaenhet" [Online], 2020. Available: *https://www.lte-anbieter.info/technik/asu.php* [2020-04-20].

[4] P. Almeida, A. Shoker, C. Baquero. Efficient State-Based CRDTs by Delta-Mutation. In *Networked Systems*, pages 62–76. Springer, Cham, 2015.

[5] E. Brewer. "Towards robust distributed systems (abstract)". In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*, page 7, New York, New York, USA, 2000. ACM Press.

[6] E. Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, Feb 2012.

[7] D. Luecke. "HTTP vs Websockets: A performance comparison – The Feathers Flightpath" [Online], 2018. Available: *https://blog.feathersjs.com/http-vs-websockets-a-performance-comparison-da2533f13a77* [2019-04-18].

[8] J. Lindström, M. Musa Elbushra. "Real-Time Eventual Consistency". *International Journal in Foundations of Computer Science & Technology*, 4(4):63–76, Jul 2014.

[9] Esmail. "Understanding Cellular Networks, Signal Strength, Factors Affecting Coverage and Methods of Capacity Augmentation - Telecom Talk" [Online]. Technical report, Telecomtalk, 2019. Available: *https://telecomtalk.info/understanding-cellular-networks/177052/* [2019-05-20].

[10] A. Gupta. "REST vs WebSocket Comparison and Benchmarks" [Online], 2014. Available: *http://blog.arungupta.me/rest-vs-websocket-comparison-benchmarks/* [2019-04-18].

[11] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, G. DeCandia, D. Hastorun. "Dynamo". In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles - SOSP '07*, volume 41, page 205, New York, New York, USA, 2007. ACM Press.

[12] K. Lo. "Download Speeds: Comparing 2G, 3G, 4G & 5G Mobile Networks" [Online], 2018. Available: *https://kenstechtips.com/index.php/download-speeds-2g-3g-and-4g-actual-meaning* [2019-05-20].

[13] D. Bermbach, J. Kuhlenkamp. “Consistency in Distributed Storage Systems”. In *Networked Systems*, pages 175–189. Springer, Berlin, Heidelberg, 2013.

[14] N. Preguica, J.M. Marques, M. Shapiro, M. Letia. “A Commutative Replicated Data Type for Cooperative Editing”. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 395–403. IEEE, Jun 2009.

[15] W. Golab, R. Rahman, A. Auyoung, K. Keeton, X. Li. “Eventually Consistent: Not What You Were Expecting?” *Queue*, 12(1):30, 2014.

[16] Powertec Telecommunications Pty Ltd. “Improving Mobile Internet Speeds” [Online]. Technical report, Powertec Telecommunications, 2019. Available: *www.mobilenetworkguide.com.au/pdf/Mobile-Network-Guide-to-Improving-Mobile-Data-Speeds.pdf* [2019-05-20].

[17] LTE provider.info. “RSSI (and ASU), RSCP, RSRP and EcIo | mytelebasket” [Online], 2015. Available: *https://mytelebasket.blogspot.com/2015/08/rscp-rssi-and-ecio.html* [2019-05-20].

[18] F. Mattern. *Virtual time and global states of distributed systems*. Citeseer, 2019.

[19] Mininet Team. Mininet Overview [Online], 2018. Available: *http://mininet.org/overview/* [2019-02-12].

[20] A. Gotsman, H. Yang, C. Ferreira, M. Najafzadeh. “’Cause I’m strong enough: reasoning about consistency choices in distributed systems”. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2016*, volume 51, pages 371–384, New York, New York, USA, 2016. ACM Press.

[21] A. Lakshman, P. Malik. “Cassandra”. *ACM SIGOPS Operating Systems Review*, 44(2):35, Apr 2010.

[22] A. Rotem-Gal-Oz. “Fallacies of Distributed Computing Explained” [Online], 2014. Available: *https://www.researchgate.net/publication/322500050* [2019-04-18].

[23] Gotsman, H. Yang S. Burckhardt, A. “Understanding Eventual Consistency” [Online]. Technical report, Microsoft, 2013. Available: *https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/distrmm-long-3-25-13.pdf* [2019-03-14].

[24] Y. Saito, M. Shapiro. “Optimistic replication”. *ACM Computing Surveys*, 37(1):42–81, Mar 2005.

[25] M. Thampi. “Introduction to Distributed Systems”. *Introduction to Distributed Systems*, Nov 2009.

[26] M. van Steen, A. Tanenbaum. “A brief introduction to distributed systems”. *Computing*, 98(10):967–1009, Oct 2016.

[27] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, B. Welch. “Session guarantees for weakly consistent replicated data”. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, pages 140–149. IEEE Comput. Soc. Press, 2019.

[28] B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Wolff. A brief history of the internet. *ACM SIGCOMM Computer Communication Review*, 39(5):22, Oct 2009.

[29] "Understanding Phone and Data device Signal Levels - Wireless ISPs" [Online], 2018. Available: *https://forums.whirlpool.net.au/archive/2473085* [2019-05-20].

[30] M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski. "A comprehensive study of Convergent and Commutative Replicated Data Types". *inria-00555588*, page 50, Jan 2011.

[31] M. Shapiro, N. Preguiça, C. Baquero, M. Zawirski. "Conflict-Free Replicated Data Types". In *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400. Springer, Berlin, Heidelberg, Oct 2011.