Local's Hit – Intermediate Report

Group 16: Frederick Dehner, Markus Drespling and David Lüttmann

1. Introduction

What is our distributed application about?

1.1 Problem Statement

How to implement a distributed system which meets all the challenges. We need to develop a heterogenized but open system. It also has to comply the security requirements of confidentiality, integrity and availability. Our distributed system must be scalable and resistant to faults. It also has to fulfill concurrency and transparency. As you can see from real life examples like the Apple AirPods, it is a difficult task to fulfill all named requirements while building a distributed system. A fundamental result in the theory of distributed systems is, that under certain conditions, including surprisingly benign failure conditions, it is impossible to guarantee that processes will reach consensus. [1] In the following report, we will describe how we tried to solve the named problems by implementing our program which is called local's hit.

1.2 Project Description

Now at times of corona virus pandemic and lockdown, it is hard to stay home and avoid social contacts. Especially students have to study at home, can't see their friends and could not life the typical "student life". Therefore, to stay connected within your student community we want to build a social media platform, where the user can anonymously share quotes with people.

It is best described with "a snapchat for short quotes (like twitter)". Each uploaded content is initially visible within a certain time period. Our goal is that all users, despite the (social) distance, are close to funny events, interesting finds or other occurrences. We want the users to forget about their problems and give them a platform to communicate without the barriers of appearance, prejudice, origin, income, and names. This is possible because of the anonymity and avoids racism and discrimination against individual members.

Therefore, we need a proxy server which communicates with the clients and displays the content. With several backend servers we provide the shared content on our frontend and replicate shared quotes. When entering our program, the client can choose in which region he wants to interact with the community. It is also possible for the users to react on posted quotes by give them an upvote or downvote.

At the following chapter the technical requirements will be focused. The decision of the specific characteristics and other possible options will also be described. This report is mainly based on the material from the distributed systems course in the summer semester 2020 including the lectures and the book "Distributed systems: concepts and design" by Coulouris Dollimore and Kindberg in the 5th edition from 2012.

2. Requirement Analysis

How would you address the project requirements?

2.1 Dynamic discovery of hosts

To establish any communication a necessary precondition is the identification of the participants. The communication initiator also must identify a role participant or group that it will send a message to. But the main question is, how can a new participant find someone when it has no knowledge about available participants? The answer will be described in the following chapter. Also, the implementation of the dynamic discovery of hosts in our project will be elucidated.

The dynamic discovery follows four steps. In the first step, if a new participant wants to join a distributed system, from which it has no knowledge of any specific recipient but only of a broadcast address, it sends a broadcast message. This broadcast message includes the address of the new participant. The second step includes the receiving of the broadcast message from potential recipients. To receive broadcast messages from new participants, each recipient continuously has to listen for broadcast messages. The recipients update their group views if they receive a new broadcast message. After the group view update each recipient respond with reply messages in a third step. This reply messages include the recipient's address. As a last step, the new participant collects all replies and creates its own group view.

To reach every recipient it is essential, to broadcast the message. In A Unicast only one recipient would be contacted. A Multicast is not restricted by size, but you cannot be sure to reach each member of the system by using a multicast.

There are two different types of transport protocols which can be used to send messages through a network. The UDP (User Datagram Protocol) and TCP (Transmission Control Protocol) differ in many aspects and are suitable for different purposes. The differences between these two protocols are described below.

UDP	TCP
UDP is a connection-free protocol. It does not create an overhead for establishing, maintaining, and closing a connection.	TCP is a connection-oriented protocol, i.e. the devices must establish a connection before data transmission, and it has to close it again after the transmission.
The successful transmission of data from source to destination cannot be guaranteed.	TCP is more reliable because it ensures the delivery of data to the destination router.
UDP only offers standard error detection mechanisms based on checksums.	TCP provides extensive mechanisms for error detection.
UDP is faster, easier, and more efficient than TCP.	TCP is slower than UDP.
UDP does not provide the sequencing of data.	TCP offers a sequencing of data, i.e. they arrive at the receiver in the "correct" order.
UDP does not offer retransmission.	TCP does offer retransmission.
The UDP header has a size of 8 bytes. It is a lightweight transmission protocol.	The TCP header has a size of 20 bytes. It is a heavyweight transmission protocol.

[2]

Because of the connection-free protocol and its fast, ease and efficient delivery with a small size the UDP is more suitable for broadcast and multicast transmissions. And therefore, we will use the UDP protocol in our project. [2]

2.1.1 Our Solution and the technical restrictions:

Every backend-server sends a service beacon at least every t=5 seconds over a UDP Datagram multicast (e.g. port 1200). Every server holds a list of the other servers. The client (in our case the user's device) only communicates with one backend server which is allocated by the leader. The process will be implemented with sockets.

2.2 Voting Algorithm/Leader Election

In this section the leader election will be considered by a voting algorithm. Therefore, the general issue of how to 'elect' one participant of a collection of processes to perform a special role will be analyzed. [1] This special role, further referred to as Leader, is required to prevent interference and ensure consistency when accessing the resources and for the sake of consistency it is necessary to choose just one server to fulfill that role. [1] Other Leader responsibilities, as learned in the lecture of the distributed systems course, are the coordination of activities, modification of data, handle of faults and the role as a sequencer. To ensure coordination, we will analyze a collection of algorithms that share an aim for a set of processes to coordinate their actions or to agree on one or more values. [1] In conclusion of the analysis, we will pick one algorithm which fits to our project requirements.

Voting Algorithms can act in a synchronous or an asynchronous system. In an asynchronous system we can make no timing assumptions. In a synchronous system, we shall assume that there are bounds on the maximum message transmission delay, on the time taken to execute each step of a process, and on clock drift rates. The synchronous assumptions allow us to use timeouts to detect process crashes. Therefore, it is impossible to guarantee in an asynchronous system that a collection of processes can agree on a shared value. [1]

An algorithm for choosing a unique process to play a particular role (Leader) is called an election algorithm. It is essential that all the processes agree on the choice. Afterwards, if the leader fails or wishes to retire and several surviving servers can fulfil that role then another leader election is required to choose a replacement. [1] Every process of the system can initiate a run of the election algorithm. But there are two requirements to be fulfilled. An individual process does not call more than one election at a time, but in principle N processes could call N concurrent elections. Secondly, the choice of the elected process must be unique, even if several processes call election concurrently. The process with the largest identifier will be chosen as the leader. An identifier may be every useful value if the identifiers are unique and totally ordered. [1]

Central requirements during any particular run of the algorithm are safety and liveness. Safety means, that a leader is chosen as the non-crashed process at the end of the run with the largest identifier. Liveness refers to the participation of all processes and their eventuality to crash or not to elect. [1]

To measure the performance of the following election algorithm the total network bandwidth utilization will be considered. [1]

2.2.1 Chang and Roberts Algorithm:

The algorithm of Chang and Roberts [1979] is suitable for a collection of processes arranged in a logical ring. Each process has a communication channel to the next process in the ring and all messages are sent clockwise around the ring. We assume that no failures occur, and that the system is asynchronous. The goal of this algorithm is to elect the leader, which is the process with the largest identifier. [1]

The algorithm can be separated in two parts. Initially, every process is marked as a non-participant in an election. Any process can begin an election by sending an election message containing its unique identifier and sends the message in a clockwise direction to its neighbour. Every time a process sends or forwards an election message, the process also marks itself as a participant. When a process receives an election message, it compares the identifier in the message with its own unique identifier. Four different possibilities can occur. Firstly, if the identifier in the election message is larger, the election message is forwarded. Secondly, if the identifier in the election message is smaller and the process is not yet marked as a participant of an election, the process replaces the identifier with its own unique identifier and sends an updated election message. Thirdly, if the identifier in the election message is smaller, but the process is already a participant of an election, the election message is discarded. And fourthly, if the identifier in the election message is the same as the unique identifier of the process, that process starts acting as a leader. [1]

The second part of the algorithm starts, when a process starts acting as a leader. The leader process marks itself as non-participant and sends an elected message with its election and identifier in a clockwise direction to its neighbour. When a process receives an elected message, it marks itself as non-participant, records the elected identifier and forwards the elected message unchanged. When the leader receives its own elected message, it discards that message and the election is over. [1]

The Chang and Roberts algorithm respects safety, as a process will receive an elected message with its own unique identifier only if his identifier is greater than others', and only when all processes agree on the same identifier. The algorithm also respects liveness. “Participant" and "non-participant" states are used so that when multiple processes start an election at roughly the same time, only one single leader will be announced. [3]

If only a single process starts an election, the algorithm requires 3n-1 sequential messages, in the worst case. [1] If every process starts the election, in a worst-case scenario $\frac{n^2+n}{2} + n$ [1] has to be sent during the election process until a leader is elected.

In the following we will point out the differences to other algorithms and explain why we decided to use the Chang and Roberts algorithm for our project.

2.2.2 Hirschberg-Sinclair algorithm

The Hirschberg-Sinclair (HS) algorithm of is also suitable for a collection of processes arranged in a logical ring. In contrast to the Chang and Roberts (CR) algorithm each process has a communication channel to both its neighbour processes in the ring and all messages are sent in both directions around the ring. There are a few more differences between the CR algorithm and the HS algorithm. At the HS algorithm all participants initiate the election, whereas every participant in the system can initiate an election at the CR algorithm. The CR algorithm works with only a single message, called the election message. The HS algorithm requires a reply message in addition to the election message. Also, the structure of the election messages differs from each other regarding the two algorithms. Whereas the election message of the CR algorithm only contains the largest identifier of previous participants, the election message of the HS algorithm contains the participant identifier, number of the current phase and a hop counter. This makes the messages and the comparison in the election more complex and takes time. Regarding the worst case of the election, the number of messages sent is also different comparing the two algorithms. We compared the case if every participant of the system is starting the election equally. The number of messages for the worst case of the HS algorithm can be calculated by $\left(\frac{\log(n)}{\log(2)} + 1\right) * 2n + n * \frac{\log(n)}{\log(2)}$ [1] with n as the number of participants.

[1] details on the derivation can be found in the appendix

The number of messages for the worst case of the CR algorithm can be calculated by $\frac{n^2+n}{2} + n$ with n as the number of participants. Regarding the two equations, the CR algorithm has to send less messages if the number of participants is 1 < n < 30. For all numbers of participants n > 30 the HS algorithm sends less messages. Even though the HS algorithm also respects safety as well as liveness, regarding our project, the CR algorithm is more suitable. That's because up to a number of thirty participants, the total network bandwidth utilization is lower with the CR algorithm. Furthermore, the election message is less complex with the CR algorithm and no reply message is required.

2.2.3 Bully-Algorithm

The main difference is that unlike the CR algorithm, the bully algorithm allows processes to crash during an election and assumes reliable message delivery. This is possible because the bully algorithm works with a synchronous system and uses timeouts to detect crash failures. Another difference is that the bully algorithm assumes full knowledge about the server's identifiers. Unlike the CR algorithm in which the participants of the system only know their direct neighbours, the participants in a bully algorithm know each server can communicate with all other servers. [1] So far, the bully algorithm seems to have great advantages in comparison with the CR algorithm. But regarding the requirements imposed at the beginning of this section, the bully algorithm cannot fulfill all of them. Firstly, the bully algorithm requires three different types of messages for the leader election. An election message, an answer message, and a final coordinator message. Secondly, regarding one server starting an election, it takes n^2 messages in the worst case to elect a leader with the bully-algorithm. [1] By way of comparison, the CR algorithm requires less messages in the worst case, because its total network bandwidth utilization is $3n - 1$. The bully algorithm sends less messages for 0 > n > 3 bur the CR algorithm in a range of 3 ≤ n < ∞. For a system of three or more participants, the CR algorithm is more suitable regarding the total network bandwidth utilization.

Thirdly, the bully algorithm is not guaranteed to meet the safety condition as two servers can announce themselves as the leader concurrently. Unfortunately, there are no guarantees in message delivery order and these could lead to different conclusions on which of the servers if the leader. [1]

Despite the fact that the CR algorithm tolerate no failures, with a reliable failure detector it is in principle possible to constitute the ring when a process of the system crashes. For our project to understand the properties of election algorithms in general and with a small number of servers, a ring-based algorithm is useful. [1]

2.2.4 Our Solution and the technical restrictions

As we have mentioned above, for our project we will use the CR-algorithm for the leader election. Firstly, we have to implement an asynchronous uniform non-anonymous ring. Therefore, it is necessary, that each participant has a unique identifier. We use the participants IP address as the unique identifier. This means, that the participant with the highest IP address is the elected leader. For the ring implementation, we assume that the participants of our system are all members of the same group and have the same view of the group. The ring is formed by sorting the list of the IP addresses. Afterwards, to find out its neighbours, each participant has to get the next higher IP address for its clockwise neighbour and its next lower IP address for its counterclockwise neighbour. As we use the CR-algorithm, the link between the process are unidirectional and every server can send the election message to its clockwise server only. The functionality of the CR algorithm has already been explained above, but in the architecture design section and in the code repository there will be examples of its implementation. With a CR leader election algorithm, we provide that we have at least and at most one leader. This one is the first backend-server, which responds to client requests.

2.3 Logical time and reliable multicast with causal ordering

2.3.1 Logical time

As we cannot synchronize clocks perfectly across a distributed system and the physical time does not work to order events that occur at different processes, we have to implement scheme which is similar to physical causality. Therefore we implement a logical clock to provide how a local process updates its own clock if an event like a new quote is posted. Also we have to determine how a local process update its clock when it receives a message from another process for example a new comment. To provide causally consistent ordering we introduce vector clocks. The vector clocks are used to causally order messages. Each process maintains a vector with a sequence number which tells us how the partial order happened. Basically, each counter represents the number of messages received from each of the other processes. So, if a client sends a message. the sender process increments its own counter and attaches its vector clock. Also, the vector time clock is suitable for our causal ordered multicast.

2.3.2 Causally reliable multicast

At the beginning we implement two regions (South and North Germany) Each of both as a closed group. A client has to select a group at the front end and then he is able to post quotes which will be multicast to the corresponding group. Therefore, we need coordination and agreement. Aim is for each group of processes, to receive copies of the message sent by the client to the group with regards to delivery guarantee. This delivery guarantee consists of an agreement on the number of messages which every process in the group receives and an agreement towards the delivery ordering across the members of the group. A reliable multicast is needed to provide the quotes to every user which is in the closed group. In addition to the reliable multicast we add up the causal ordering to implement like and reply functions. A causally ordered multicast is needed to guarantee a relationship between posting and like & reply function.

Why we did not choose basic multicast:

➔ Ack implosion → multicast process buffers fill rapidly → as a result we have to retransmit messages which further results to more acks and more waste of network bandwidth.

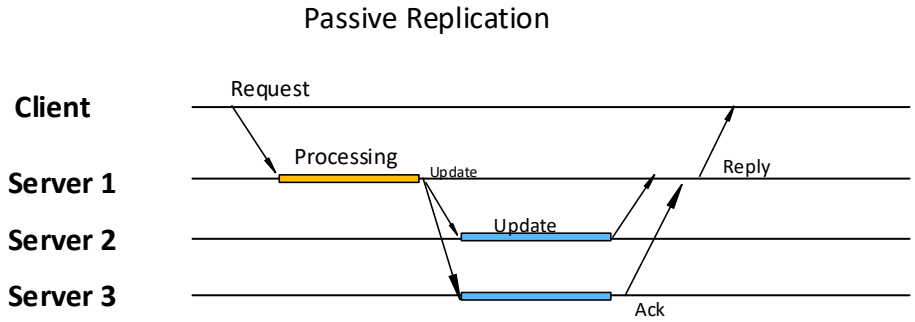
With the reliable multicast we keep a history of messages for at-most-once delivery and everyone repeats the multicast upon a receipt of a message for agreement & validity. So, we achieve:

- “Agreement: If a correct process multicasts message m then it will eventually deliver m.
- Validity: If a correct process delivers message m, then all other correct processes in group(m) will eventually deliver m.”[1]

Because of reliable multicast, every user receives every posting eventually. Besides the reliable multicast we implement a casual ordering for replies and votes. So, every posting from a given use will be received in the same order. This concerns only particular users. With casual ordering we guarantee a relationship between posted quote and the reply.

2.4 Replication

2.4.1 How does it work? – passive replication



In the passive, also known as primary back model of replication, we have one designated primary Replication Manager who is responsible to manage the communication between client and the backup servers. The clients send over the front end a request to the primary only. After receiving the request, the primary executes the request, atomically updates the other copies, sends updates to the backup servers and finally response to the client. "If the primary fails, one of the backups is promoted to act as a primary." [1]

2.4.1.1 Correctness criteria's

Linearizable

Obviously the linearizability is achieved through a correct primary which sequences all the operations. In case of primary failure, the system can preserve linearizability with a new primary election, where a backup server gets promoted to the new primary server. The new primary must take over exactly where the last primary left off.

Sequential Consistency

"Replicated data is sequentially consistent if for any execution there is some interleaving of the series of operations of issued by all the clients which satisfies the following two criteria:

- The interleaved sequence of operations meets the specification of a (single) correct copy of the objects.
- The order of operations in the interleaving is consistent with the program order in which each individual client executed them." [1]

Crashes:

If the primary crashes, backups will receive new view with primary missing. A new primary gets elected.

If Front End re-sends request either the reply is known and resent or the execution proceeds as normal.

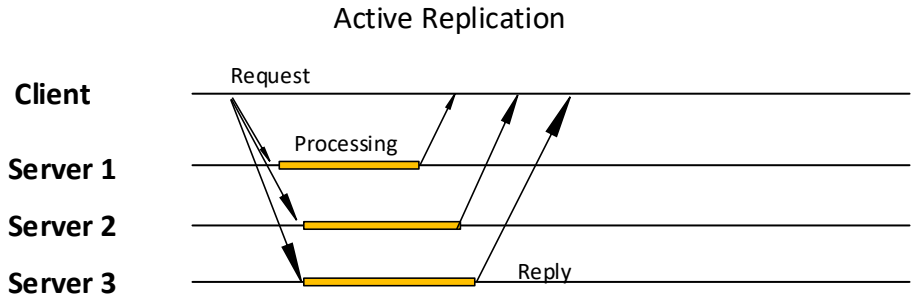
2.4.1.2 Advantages

- All operations passes through a primary that linearize operations. Each request is atomically taken in order. Therefore, we do not need a coordination extension of FIFO etc.
- Works even if execution is non-deterministic.
- If view – synchrony is given, no consultation with the backups is needed, because they all have processed the same set of messages.

2.4.1.3 Disadvantages

- Delivering state change can be costly.
- View-synchrony and leader election could be expensive.
- Failure response is delayed (in comparison to active replication).

2.4.2 How does it work? – Active Replication



Active replication is a protocol which has no centralized control. Therefore, all replica managers are state machines that have equally rights and are organized as a group. They receive each request, process it, updated their state, and send a response back the client. In case of a replica manager crash, there will be no impact due to the remaining responding replication managers. Failure of replica managers are not visible for the client. But all replicas have to be in the same state for the client. This is mostly ensured by a total ordered multicast.

2.4.2.1 Correctness criteria's

Sequential Consistency

Active replication achieves sequential consistency since the replica managers are all state machines. So, they end up with the same state. As result the client request are served in FIFO order which ensures sequential consistency

Linearizability

"The active replication system does not achieve Linearizability. This because the total order in which the replica managers process request is not necessarily the same as the real-time order in which the clients made their requests." [1]

2.4.2.2 Advantages

- No need to send state changes because all requests delivered in total order.
- No need to change existing servers → all servers are capable to process request.
- Read request could possibly be sent directly to replicas.

2.4.2.3 Disadvantages

- Requires total order multicast.
- Requires deterministic execution.
- Front ends may send read-only request to replica manager → losing fault tolerance that comes with multicasting request → Front end can easily mask the failure of a replica manager

2.4.3 Why did we choose passive replication?

We choose the passive replication because we already follow a casually ordered multicasting approach. In case of the totally – ordered multicast, we would have probably a big Hold-back queue due to many client requests. Our goal is to achieve a high availability and good performance therefore we also implement a variation of the stated model above: The clients are able to submit read requests to backups, therefore we are capable off-loading work from the primary. Consequently, we lose the guarantee of linearizability but at least the clients receive a sequentially consistent service.

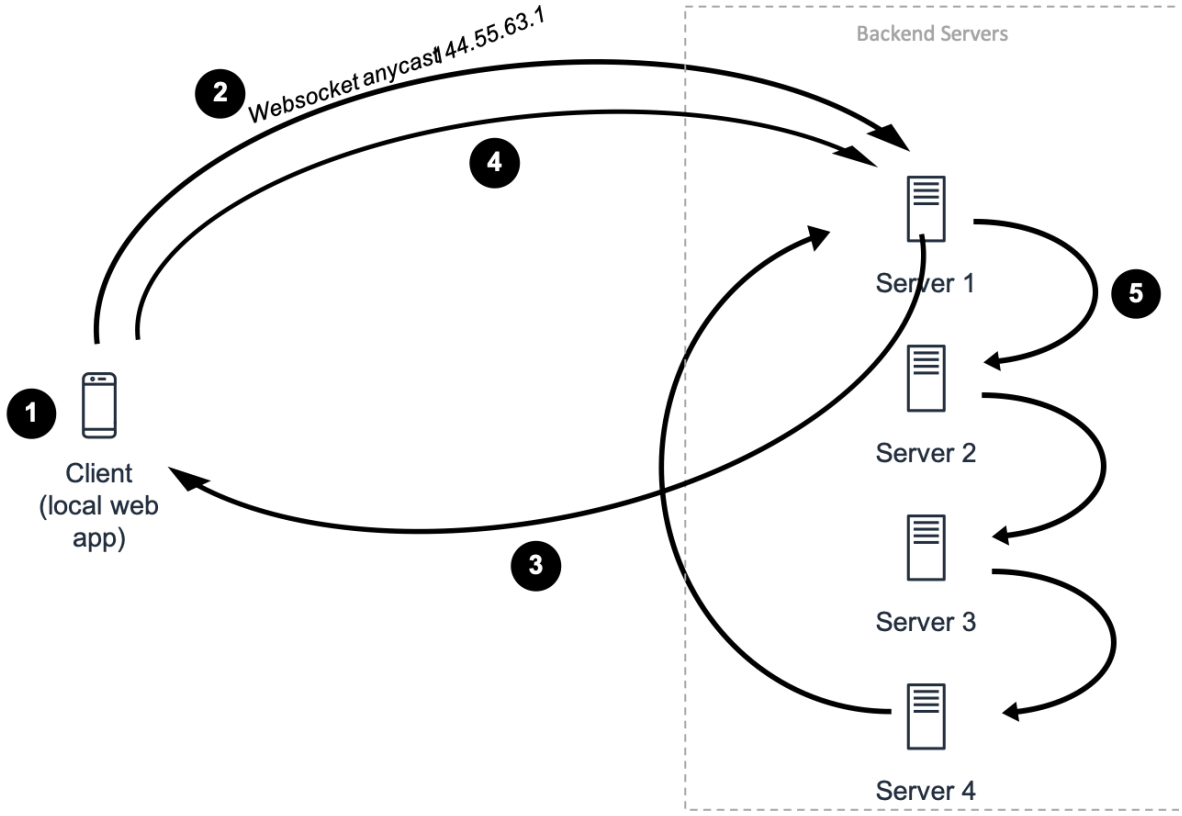
Another disadvantage of the active replication is the requirement of replicated objects have to be deterministic, which results in redundant processing. Whereas the passive replication does not waste resource and permits non-deterministic operations. But we have to deal with an increased latency of an invocation. Also, we have to deal with the update state and its distribution to the backup servers.

Furthermore, the architecture given by the passive replication enables us the use of a primary copy replication. Primary copy replication is used in context of transactions, in our case all client requests are directed to a single primary replica manager. As a result, for primary copy replication, we control concurrency through the primary.

3. Architecture Design

What is the initial idea for the architecture design?

3.1 Architectural Models



There is much more to come… We are working on it.

4. Implementation

There is much more to come… We are working on it.

4.1 Code Repository

There is much more to come… We are working on it.

5. Discussion and conclusion

There is much more to come… We are working on it.

5.1 Future Work

In a further step or stadium of our project we can think about uploading images instead of text or a vicinity-based service, so that the content is only visible for users in a limited area of approximately 10 kilometers around the person who uploaded the content. But for the first implementation we focused on the technical topics of the lecture. These two ideas require high amount of client-based technology like GUI programming or location-based services.

6. Bibliography

[1] G. F. Coulouris, J. Dollimore, T. Kindberg, G. Blair, A. K. Bhattacharjee, and S. Mukherjee, *Distributed systems : concepts and design*. 2012.

[2] F. A. Dalwigk, “UDP vs. TCP - Algorithmen verstehen,” 2019. [Online]. Available: https://www.cybersicherheit.guru/tcp-vs-udp/. [Accessed: 07-Jun-2020].

[3] “Chang and Roberts algorithm,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Chang_and_Roberts_algorithm. [Accessed: 07-Jun-2020].

7. Appendix

7.1 Proof of total network bandwidth utilization (n)

Hirschberg-Sinclair algorithm worst case with 8 participants:

Current phase (k)	Number of messages (n)	Type of message	Check if $d = 2^k$
k=0	n*2	Election	$2^0 = 1$
k=0	n	Reply	$2^0 = 1$
k=1	n*2	Election	$2^2 = 2$
k=1	n	Reply	$2^1 = 2$
k=2	n*2	Election	$2^2 = 4$
k=2	n	Reply	$2^2 = 4$
k=3	n*2	Election	$2^3 = 8$
In general:			
k(max)	n*2	Election	$2^k = n$

Getting k: $2^k = n \rightarrow \frac{\log(n)}{\log(2)} = k$

Regarding the election messages: $(k+1) * 2n$ | (k=0 must be considered!)

Regarding the Reply messages: $k * n$ | (k(max) must not be considered!)

➔ $\left(\frac{\log(n)}{\log(2)} + 1\right) * 2n + \frac{\log(n)}{\log(2)} * n$

Chang and Roberts Algorithm worst case with 8 participants:

Number of iterations	Number of messages (n)
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	8

For n iterations: $\frac{n^2+n}{2}$ | ”Gaußsche Summenformel”

For the n+1 iteration: n | = maximum iteration in worst case

➔ $\frac{n^2+n}{2} + n$