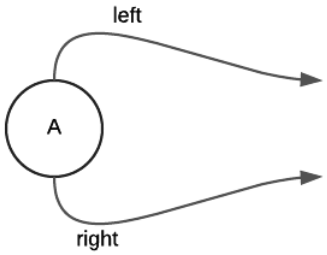
Mini Project 3

Dennis Thinh Tan Nguyen, Jacob Holm, Jacob Mullit Mniche,
Pernille Lous, Thor Valentin Aakjr Olesen, William Diedrichsen Marstrand

20. november 2015

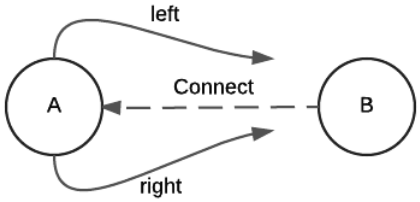
1 Question 1: Briefly explain your solution and why it works

A Node can hold the information on a right and left neighbor Node, by storing the IP and port of the specific Node.



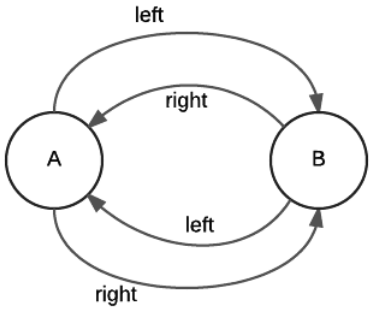
Figur 1: Single node before connection

The initial Node will have no Nodes stored for its left and right neighbor as is shown in Figure 1.



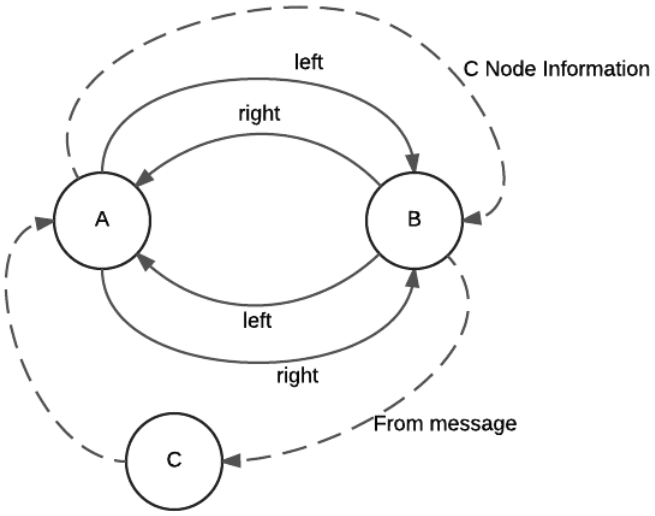
Figur 2: Incoming connection from Node B

When a new Node wants to connect it sends a From message containing information about its IP and port to the Node it wants to connect to. The existing Node (A) will check if it holds any information about a neighbor in its right or left variable Figure 2.



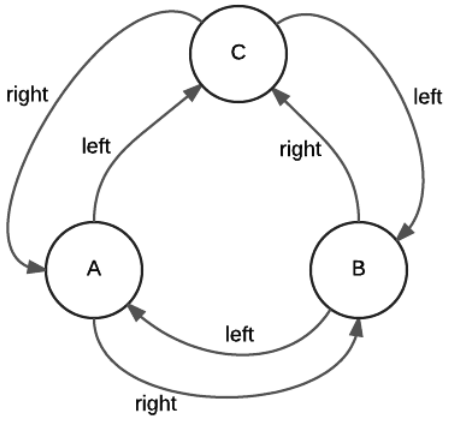
Figur 3: Nodes after connection

In Figure 2 Node A is not holding any neighbor information at all, and so it will store the information about the incoming Node (B) in both its right and left value Figure 3. The arrows shown illustrates that the Node pointing to another Node holds that Node's IP and port info in its left and/or right value. To store this information a connection is established between the Nodes, but it is closed immediately after, and so no permanent connection is held open. E.g. in Figure 3 both Node A and B store each others IP and port in both their right and left value. The Nodes will echo the Nodes according to their left and right values to see if they are alive.



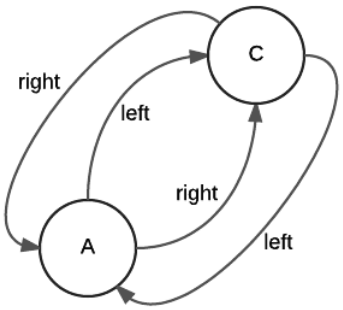
Figur 4: Incoming Node connection from Node C

When another Node (C) wants to connect to one of the existing Nodes (A), the requested Node (A) will send the IP and port info to the Node according to its left value (B) assuming that it exists. When the Node (B) receives the IP and port it will change its right value to this (C), and send a From message to the Node (C) Figure 4.



Figur 5: Nodes after connection

Now the initially requested Node (A) will change its left value to match the incoming Node (C) and lastly the incoming Node (C) will set its right value to match the requested Node (A) and its left value to the incoming Node (B) Figure 5.



Figur 6: Nodes after 3rd Node disconnect

If one of the Nodes (B) disconnects, the Node which is connected to this node through its right value (A) will send a message to its neighbor Node (C) on the left value asking if this Node's left value is the same as the requesting Node's (A) right value. If this is not the case the message will go to the next Node and the next and so on until finding a match. Then the matching Node (C) will get the info of the requesting Node (A) and connect its left value to it. The requesting Node (A) will then store this Node's (C) info in its right value, and the circle is complete again. If a Node holds any resources one of its neighbors will always have a reference to these resources. When the Node is disconnected, the neighbor Node will inherit these resources, and pass on a reference for these resources to one of its neighbors.

The solution works under the assumption that whenever a Node is disconnected from the system, the neighbors to this Node will have sufficient time to reconnect themselves and send new resource references to their neighbors. If this condition is upheld the system will work, because the resources will always be stored in 2 places unless only a single Node exists, which will then hold all the resources.

2 Space Consumption

If this assumption is upheld, the network will be able to handle N Nodes and disconnection of N-1 Nodes.

We both explain the Space Consumption for each Node and the network in general.
The resources in our network are strings inserted by the user. The space consumption therefore depends on both the structure and what the user inserts into the network. A message's space consumption is 4bytes + variable length of the string.

Since the Nodes do not know which messages there have passed through the Nodes before, the Nodes do not use space consumption to store these resources.

N is information consisting of a key and a value.

Object / Actor	Researcher	Manager
Researcher	R	R
Paper	RU	RU
Project	R	R
Task	RU	RU
Team	R	CRUD
API	R	R

2.1 Best case

Each Node The very best case for each Node will be zero if nothing is stored in the Node, but this is not likely and the best case is considered as O(N). When the Node is created and the user makes a PutMessage the Node will use space consumption O(N).

For the network The lowest space consumption for the network will be if there is only one Node in the network. In this case none of the resources are duplicated and will only take place in this Node; this will held O(N) amount of resources.
However, this is not the best solution since there will be no backups of any resource and a heavy amount of resources are stored in just one Node. This will have a bad effect on how the structure distribute further inserted resources - this is explained in subsection "Worst case".

2.2 Average case

Each Node It is difficult to calculate the exact average case since it is depending of the usage of the network. This network is unstructured and the average case depends on how and how many Nodes are added/removed. This further depends on the users input.
What we can say is that for each Node it will be between (0-N) but it can never

be more than N.

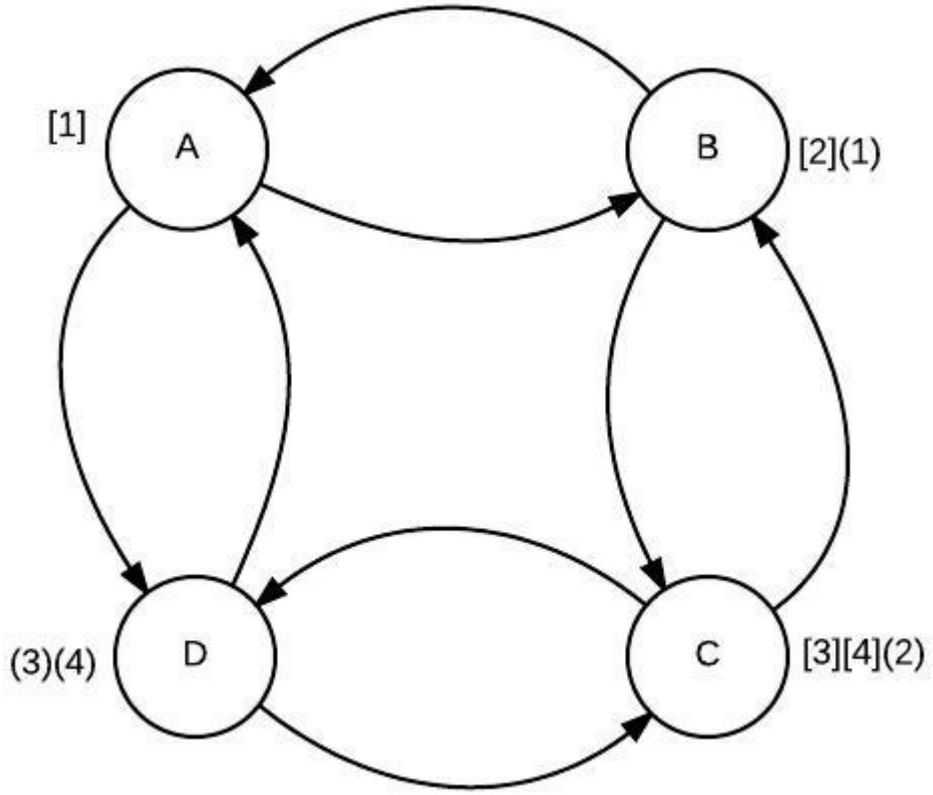
For the network For the network in general it is most likely that the space consumption is (N*2) since every time a need Node is created, it will backup the resources stored on its left side.

2.3 Worst case

Each Node When all of the resources are duplicated the worst case must be the double of the best case. Therefore, the worst case will take (N*2) space consumption for each node.

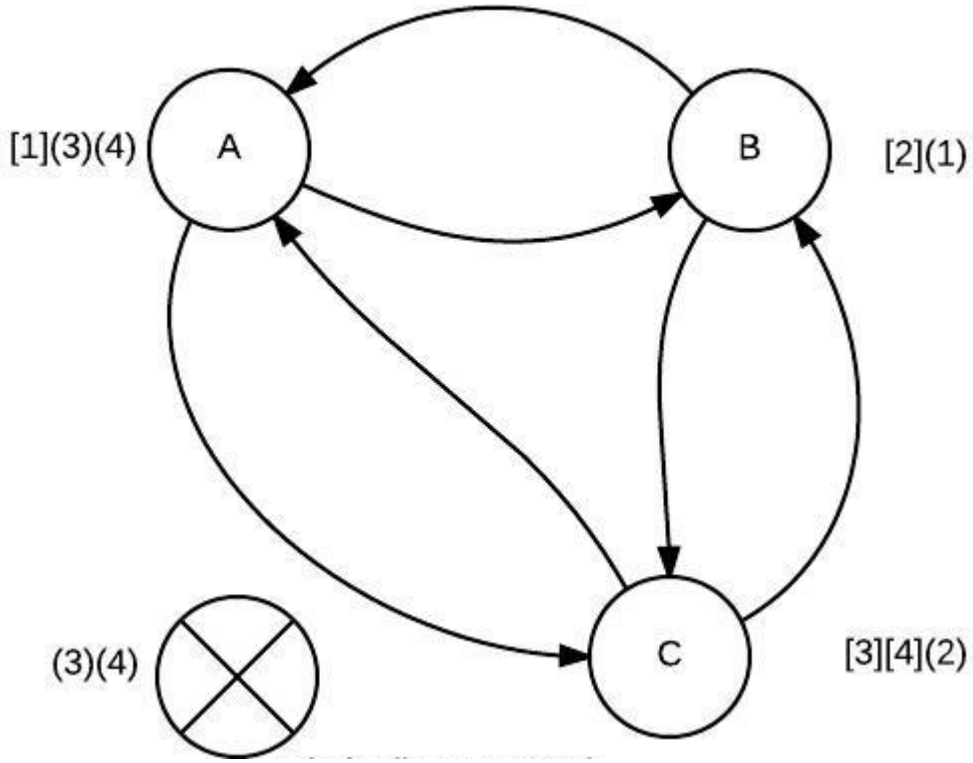
For the network According to how our network works, there will be another worse case scenario:

If the network consist of x-numbers of Nodes, then there is an amount of resources stored.

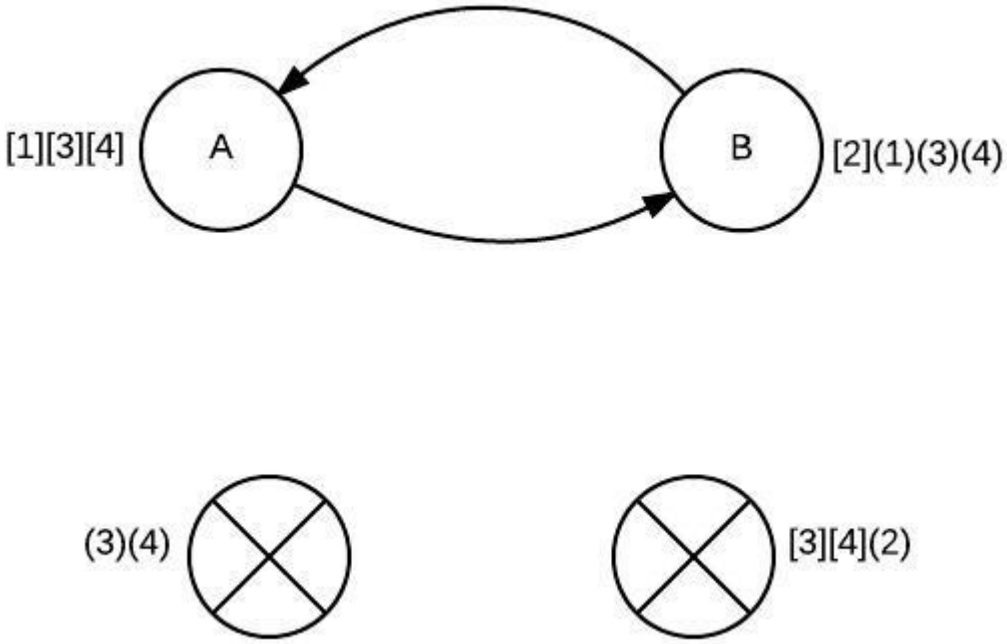


Figur 7: A network with no Nodes disconnected yet

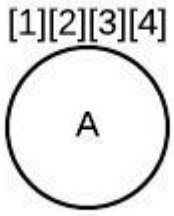
When a Node is disconnected from the network it will sent a backup of the resource to right side from itself. If the Nodes continues to disconnect without creating new Nodes, in the end the network will only consists of one Node.



Figur 8: A network with one Node disconnected.



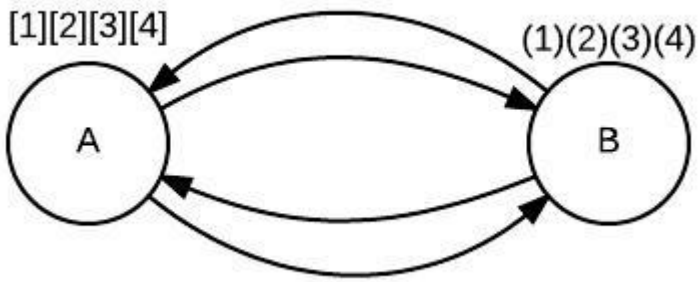
Figur 9: A network with two Nodes disconnected. The resources are now stored in neighbor B.



Figur 10: A network with only one Node left. This Node stores all the resources from the other pre-existing Nodes. It will make its own backup to its original resources.

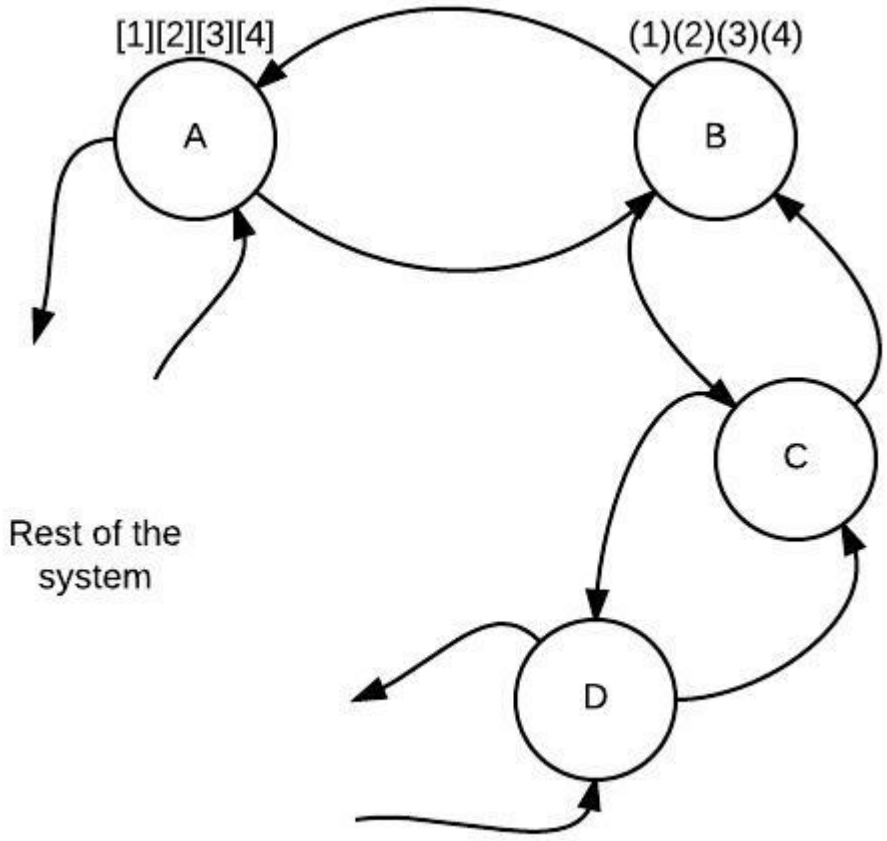
When there is only one Node left in the network, this Node(A) will store all the resources of the whole network in this particular Node.
In this case, when we create a new Node(B), this Node(B) will request the first Node(A) for resources to backup, and the first Node(A) will send its resources.

The network will then consist of two Nodes with a heavy amount of resources.



Figur 11: Node B will take a backup of A.

The worse case appears after this scenario, when we add x-numbers of Nodes. When we create Node(C), this Node will ask Node(B) if it needs any backup - but this is not the case, since Node(A) and Node(B) already have duplicated resources. Node(C) and every other Node created after this will never store any resources.
This mean we can create x-number of Nodes, but the resources will cluster in Node(A) and Node(B) no matter what.



Figur 12: All the resources are stored in Node A and B and they will cluster here. No resource is stored in another Node than A and B no matter how many Node we add to the network.

3 Number and Size of Messages

3.1 PUT message from a client

When a message is being PUT it is always stored at the Node to which it is send. Therefore both the best and worst case of a PUT message is 1 and thus the average-case number of messages is also 1.
A PUT message holds 3 fields two of which is constant (int for a key, and boolean for the version) and a String which can vary in size. And as such the size of all PUT messages is linear in the length of the message String, inputted by the user.

3.2 Successful GET message

The best case for the number of messages from a successful GET message is 2, first the request for a resource and then the message with the resource. The worst-case number of messages is $(N-1)+2 = N+1$ where N is the number of Nodes and +2 for the initial GET message and the returned resource. The average-case number of messages is dependent on the number of Nodes in the network in the way that when the number of Nodes increases the potential of not finding the resource at the first Node becomes lower and the number of potential messages needed to find the resource increases as well.

3.3 Unsuccessful message

If a GET message is asking for a resource with a non existing key, the GET message will go around in the system for ever. This means that in this case both the best- worst- and average-case number of messages will be infinitely high increasing until the system stops running.

4 Scalability

Handling clustering of resources

The resources in the Nodes are never redistributed to create an even allocation of the resources, which means that the system can end up with big clusters of resources in single Nodes. This could have been avoided by having some kind of capacity indicator for each Node. When a resource is PUT in a Node, it could then ask its neighbors if they had less resources than itself. If this is the case, the resource would be passed on to this Node. This would continue, until arriving at a Node with less resources than its neighbors or until the resource arrives at the initial Node. In this way the resources would be somewhat evenly distributed across the network.

Handling GET message for none existing resource

If a request comes in asking for a key which does not exist, the GET message will go around forever. This constitutes a scalability problem if one assume that the users of the system does not always ask for a key in the system. A possible scenario could be an adversary who wants to crash the system and therefor generates an arbitrarily high number of messages asking for non existing resources and eventually the number of messages going around in the system will make it crash. This could have been avoid by letting the GET message know if it had already visited a Node, for instance by saving some of the Node information in the message when the Node is requested. This would mean that the message would be dropped after asking the last Node for the resource.

Better routing

The current system is unstructured which makes the searching for resources less efficient than if the system had been structured. Since a ring topology has been used to create the system an obvious choice would be to use the Pastry overlay, which would have improved the routing by the use of a distributed hash table.