Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Peer-to-peer networks under churn

Semester Thesis

Thomas Buob

tbuob@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Lucianna Kiffer
Prof. Dr. Roger Wattenhofer

December 29, 2023

Abstract

In the present time, peer-to-peer networks are everywhere. They define how we perform day to day tasks and how we use the internet. Currently, we can observe a shift from a centralized to a more decentralized world.
In this project, we will take a look at this shift from a very centralized to a more decentralized world by simulating a large peer-to-peer network. We will discuss the impact that churn has on networks, and how it changes a network over time. We will observe how it affects the eccentricity of nodes and analyze how decentralized a peer-to-peer network actually is.

Examining large peer-to-peer networks in the real world can be challenging, as the involved parties have a rather large interest in keeping as much information about the behavior of their network as hidden as possible. In this thesis we produce simulate a peer-to-peer network, produce our own data and compare it to real data from the Bitcoin peer-to-peer network.
The focus will be on eccentricity and degree of the individual nodes.

Contents

Chapter 1

Introduction

Bitcoin or Ethereum are large decentralized peer-to-peer networks, which remain stable and available regardless of the status of their individual nodes. These peer-to-peer networks have a lot of interesting properties, one of which being the rate of nodes joining or leaving the network itself. This is generally referred to as the churn-rate
The churn rate has a big impact on the performance and stability of the network, especially as both Bitcoin and Ethereum use distributed consensus algorithms to maintain their integrity.

Investigating the churn rate is a complex task that includes multiple issues that have to be addressed:

1. Scalability issues.
   A large directed graph does not scale well, as every operation on one node forces the graph to update the properties of all other nodes in the graph as well. Some of those properties do have a large time complexity.

2. Unknown properties.
   Blockchain networks like Bitcoin have a large percentage of peers behind network addressing translation (NAT). These nodes only provide outgoing links and can not be reached from other nodes.

3. Resource consumption.
   Running a simulation on a large graph consumes a lot of resources. Additionally, implementing algorithms that rely heavily on multiprocessing will produce a large overhead. Running multiple simulations at the same time can also impact each other due to hardware based bottlenecks, like cache size or memory read/write speed.

4. Strong connectivity.
   For graph algorithms and network connectivity, it is important that the graph stays strongly connected. Ensuring strong connectivity adds another computational overhead.

5. NAT nodes.
   The amount of unreachable peers in the Bitcoin network is unknown, but heavily impacts the behavior of the network. Unreachable peers contribute to the network by disseminating blocks and transactions.

The problems above will be addressed in the following chapters. We will show the impact unreachable peers have on the network and show graph properties by focusing on the eccentricity and degree of each node.

Chapter 2

Network Simulation

In this chapter, we will show what design decisions were taken and show how to run a simulation.

2.1 Problem of using a directed graph

As we heavily rely on the eccentricity of the nodes and their degree, it is important that those parameters can be calculated as efficiently as possible. Early simulations showed that two issues arose:

1. The graph becomes weakly connected. Calculating the eccentricity requires the graph to be strongly connected. Removing nodes and adding new nodes requires multiple additional operations on the graph to ensure that it is strongly connected again, even if already established peers have to be deleted to force new connections. This is not something we want, as it adds makes the process comparatively more expensive and possibly negatively impacts the collected data and its correctness.

2. Directed graph space complexity. A directed graph is a lot larger than an undirected graph. It requires incoming connections, outgoing connections, in degree, out degree and so on for every node. This added size puts a large overhead on multiprocessing operations.

To limit the above-mentioned issues, we decided to run the simulations with an undirected graph and adapt all operations that required the graph to be directed. We implemented a hash map that stored the outgoing connections for every node and thus providing direction for each edge. This reduced the size of the graph object and resolved the necessity to ensure strong connectivity.

2.2 Scalability Issue

The fundamental issue of simulating a large network is the exponentially growing time complexity. There exist multiple libraries that try to reduce that bottleneck with a multitude of optimizations. In our case, we use the NetworkX [1] library. This library is limited by pythons single core focused design, but provides good documentation and is rather user-friendly. Lately, it also has been updated to provide an interface for GPU based processing, which could improve the performance significantly. The bottleneck we experienced with it appears to be the graph access itself for networks with 10000 nodes or more.

This forces us to make some compromises. Calculating distances between nodes is very resource intensive, as it relies on the shortest path computation to calculate all pairwise distances for each individual node. The resulting time complexity for this is $O(b^d)$ (A*-algorithm)

Some improvements can be made by using multiprocessing, but this only provides a negligible difference for networks with more than 1000 nodes.
We will therefore mostly rely on calculating eccentricity and degree of each node.

The combination of eccentricity and degree gives us a good insight on the relevance of each node. Additionally, we will store all edges in every cycle, which can be used to calculate the age of connections between nodes.

2.3 Simulation

The simulation follows the following steps:

1. Define graph parameters
2. Create undirected graph
   2.1. add nodes
   2.2. randomly add edges until every node has k edges
   2.3. add all connections for every node to direction object
3. randomly select x-percent of all nodes
4. remove selected nodes and their edges
5. store what nodes have lost how many neighbors
6. add the same amount of nodes with label NAT
7. add k outgoing edges, but only towards non NAT nodes
8. reconnect the nodes that lost their neighbors, but only with non NAT nodes

9. give every node its churn probability
10. iterate through specified amount of cycles
11. store all produced data

2.3.1 What a cycle looks like

In 2.1 the process of a single cycle is depicted i.e. step 10 previously.

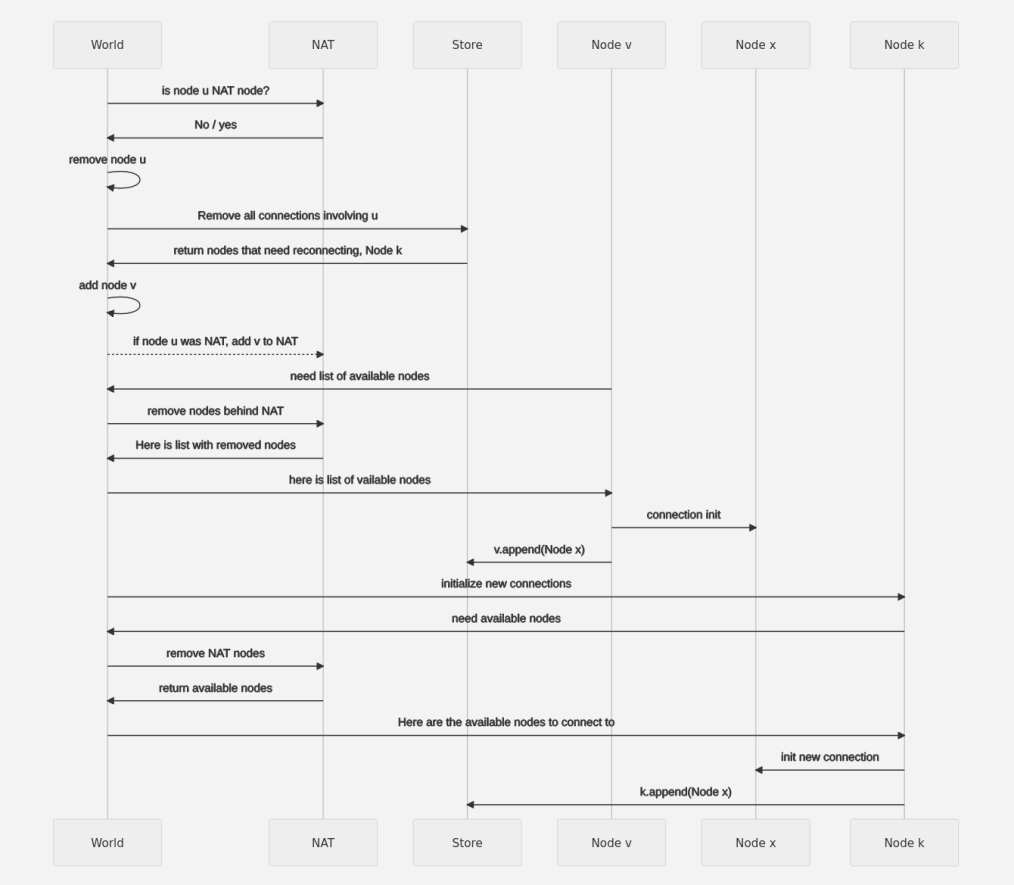


Figure 2.1: One cycle of removing and adding nodes

2.4 Short guide on how to run a Simulation

For ease of use, the code has been split onto multiple files. In this section, we will show how to use it and how to enable / disable features.

Setting parameters

```
size = #Nodes
k_outgoing = # outgoing connections per node
upper_limit = # cap on incoming connections
probabilities = {chance of churning per cylce *e2: amount nodes}
nat = how many nodes are behind nat (x out of 100)
```

Plotting data collected by monitoring node vs ground truth: For a settled network, what the monitor node observes should be similar to what the ground truth is. The use of a monitor node to collect data proved to be not reliable, as the monitor node requires the network to run multiple thousand cycles after settling to establish a big enough set of connections. If measurements want to be done via monitor node, it can be enabled in the plot eccentricity file by setting boolean to "True".

```
if os.path.isfile(filename_monitor):

    with open(filename_monitor, 'r') as file:
        file_content = file.read()
        objects=file_content.strip().split('-')
        ic(file_content)
        monitor = objects[0]
        ic(monitor)
    boolean = False
```

Using an undirected graph instead of a directed graph makes the computation more efficient. The downside of this is, that having two nodes connect to each other in both directions gets blocked. But the influence of this in a network this large is negligible.

To still have a directed graph we store the outgoing connections in a hash map.

```
def create_undirected_graph(self,upper_limit):
        nodes = list(range(self.num_nodes))
        random.shuffle(nodes)
        for i in range(self.num_nodes):
            self.store[nodes[i]]=[nodes[(i+1)%self.num_nodes]]
            self.graph.add_edge(nodes[i],nodes[(i+1) % self.
   num_nodes])

        for node in self.graph.nodes:
            targets = set(nodes)
            targets.remove(node)
            for temp in targets:
                if self.graph.degree(temp)>=upper_limit:
                    targets.remove(temp)
            while len(self.store[node])< self.degree:
                target = random.choice(list(targets))
                if not node in self.store:
                    self.graph.add_edge(node,target)
                    self.store[node]=[target]
                elif not target in self.store[node]:
```

```
20                      self.graph.add_edge(node,target)
21                      self.store[node].extend([target])
22                  else:
23                      continue
24                  targets.remove(target)
```

In the following code:
Line 1 defines how many cycles are run. Then depending on the use case, if distances between certain nodes have to be measured, line 238 is important. Generally, measuring the distance between nodes should be avoided, as the time complexity explodes when doing so.

```
1  for i in range(250):
2          prob = run.drop_connections()
3          eccentricities = {}
4          distances_to_miners = {}
5          for index, key in enumerate(prob):
6              eccentricities[key] = GraphSetup.
   compute_eccentricities_parallel(run.graph.graph,prob[key])
7              distances_to_miners[key]=GraphSetup.
   compute_distance_to_miner_parallel(run.graph.graph,run.stable,
   prob[key])
8          with open(filename_10,'a') as file:
9              json.dump(prob,file)
10         maximum = 0
11         distance_to_miners =  []
12         val = []
13         dis = []
14         for ecc in eccentricities:
15             val.append(sum(eccentricities[ecc].values())/len(
   eccentricities[ecc]))
16             maximum = max(max(eccentricities[ecc].values()),maximum
   )
17         for dist in distances_to_miners:
18             dis.append(sum(distances_to_miners[dist].values())/len(
   distances_to_miners[dist]))
19         dia = maximum
20         store[i]=(run.graph.store)
21         eccentricity.append(val)
22         diameter.append(dia)
23         with open(filename_9,'a') as file:
24             json.dump(nx.node_link_data(run.graph.graph),file)
25         distances.append(dis)
```

As some of the simulations were running for multiple weeks, all graph information is being stored, so that future simulations can take that graph as a starting point and do not first have to stabilize. Stored are:

What nodes have what probabilities	'data/write_edgelist/node_probabilites'
The eccentricity of each node for each cycle	'data/eccentricities/eccentricity'
All edges each cycle	'data/write_edgelist/edge_list_values'
The avg degree of nodes of a certain churn probability	'data/edges/tables'
The store file containing the directions (outgoing connections)	'data/edges_store'
The monitor node if activated.	'data/monitor_monitor'

Chapter 3

Investigating Bitcoin and defining parameters

In this chapter, we will take a look at the distribution of connection durations for Bitcoin and define the parameters of our simulation accordingly.

3.1 Distribution of connection durations

By observing how long connections between nodes stay up, we can learn a lot about the network and its properties. For Bitcoin, there exist monitor nodes that have unlimited incoming connections available. Over time, a lot of nodes will connect to them, as every node tries to have all 8 outgoing connections filled, while only a limited amount of nodes that can be connected to exists. The Karlsruhe Institute of Technology [2] runs such a monitor node and plots the distribution of the connection durations in the plot below.

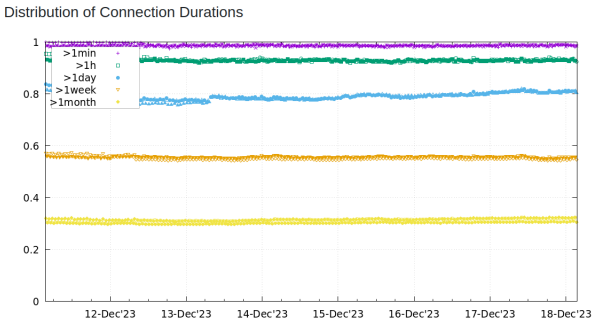


Figure 3.1: [2] Bitcoin distribution of connection durations

For our simulations, we decided to use the ground-truth as reference for the connection durations. Simulating a monitoring node did not prove to be beneficial, as the nodes would not connect to it in an appropriate time, unless we forced

them to do so by implementing a low upper bound on the amount of incoming connections per node.

The parameters for our simulations were determined by an educated guess and adjusted by trial and error until the results of the simulations resembled the observed data from 3.1.
In 3.1 we can observe, that more than 90% of peers are connected for at least one hour, more than 80% for more than one day and so on.

The probabilities for churn per cycle used for our simulations are:

```
probabilities = {1:#nodes/100, 0.17: #nodes/10, 0.007:#nodes
/5,0.001:#nodes/5,0.0003:#nodes/5, 0: rest}
```

Below we plot the distribution of connection durations for a simulation with those parameters.

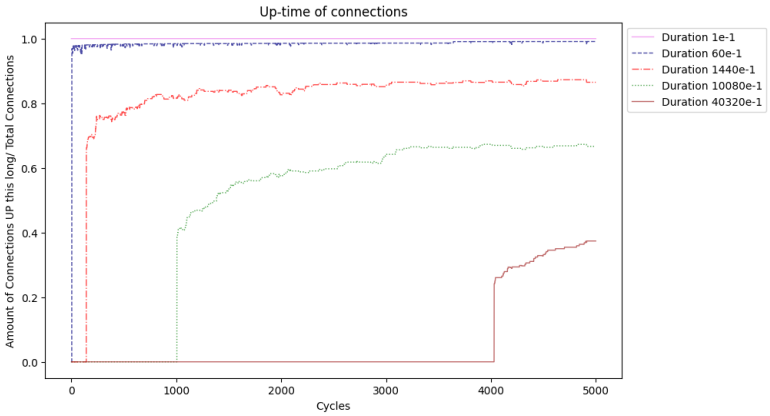


Figure 3.2: Distribution of Connection Durations of 6.7(a)

In 3.2 we observe, that more than 95% of all connections are online for longer than 1 hour, over 80% of all connections are online longer than one day and around 40% of all connections are online for more than one month.

Chapter 4

Distance

In this chapter, we will investigate the distance between churning and non-churning nodes. This will provide us with some additional context for the coming chapters and allow us to make predictions on what to expect.

In 4.1, the probabilities used are $0.0, 0.1, 0.2, ..., 0.9$ for 100 nodes each in a network with 1000 nodes in total.

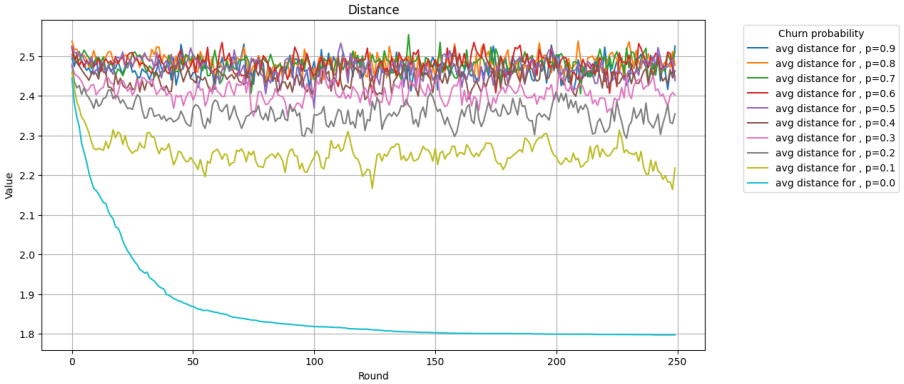


Figure 4.1: avg distance of node to miner (stable node)

We can see how the distance of between the nodes with $p = 0$ converges to a distance of 1.8, while the distance to the churning nodes are relatively stable at a further distance. Based on 4.1, we can now make the assumption, that non churning nodes over time will find each other and form the center of the graph. The higher the churn probability of a node is, the further away it should be from this center. Due to this, we expect the non churning nodes to have more incoming connections than the other nodes.

Chapter 5

Unreachable nodes

In the Bitcoin P2P-network, a substantial amount of nodes are behind a NAT. This means that they have only outgoing connections and that they are very hard to detect in the network. Even thouth these unreachable peers only have outgoing connections, they do still play an important part in the network. One of their main goals is to disseminate transactions and blocks [3] and to provide redundancy. While the exact number of how many nodes are behind NAT is unknown, estimates are about 50 to 70%. In the following paper [4] the assumption is that more than 84% of all Bitcoin nodes are behind NAT.

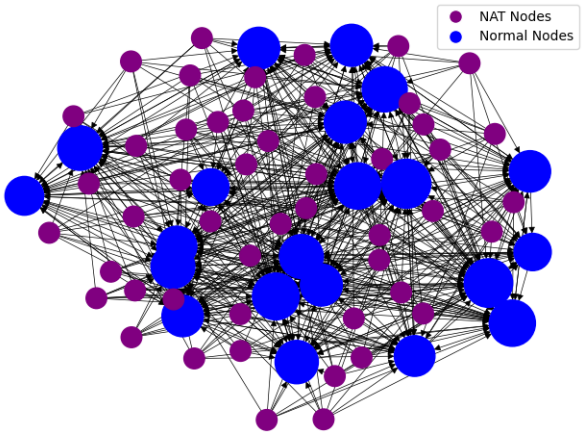
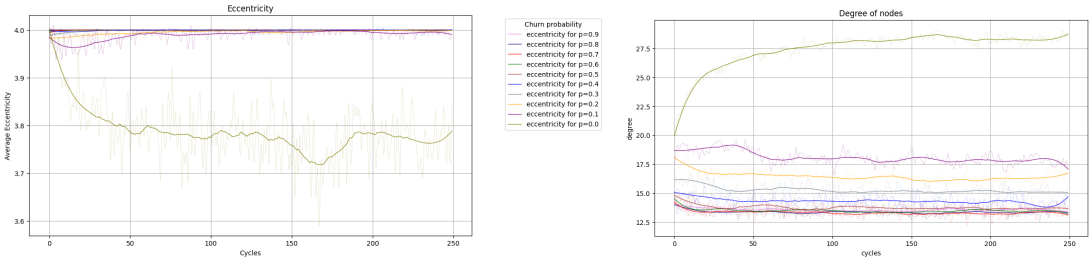


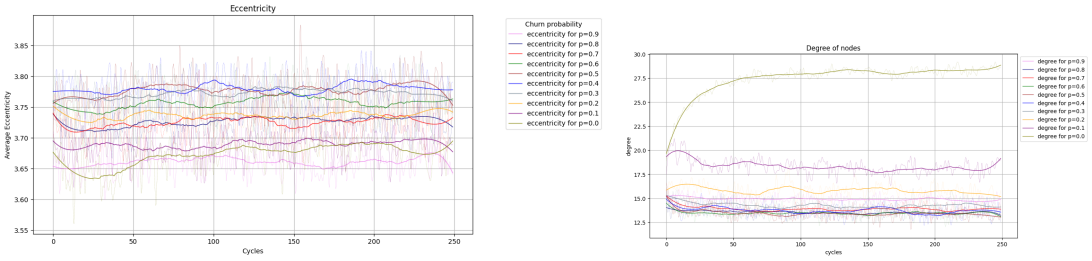
Figure 5.1: Example of a Network with 70% of the nodes being behind NAT

In this chapter, we will show how varying the amount of NAT nodes influences the Graph.



(a) 1000 nodes, no unreachable nodes

(b) distances, only reachable nodes



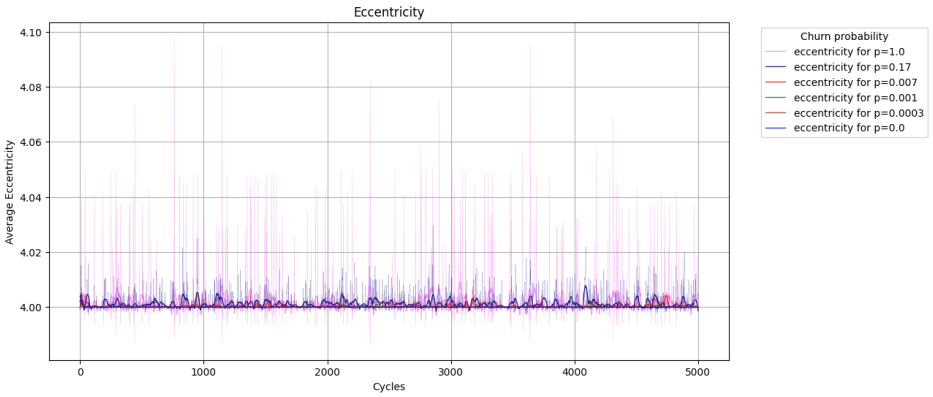
(c) 1000 nodes, 70% unreachable nodes

(d) distances

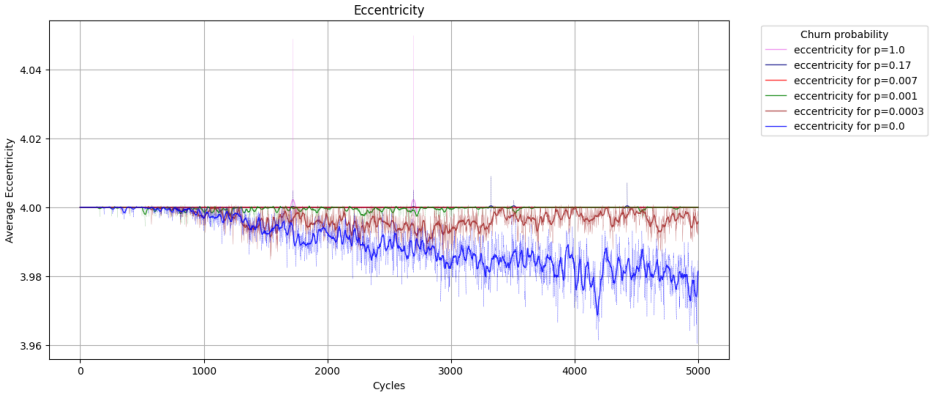
Figure 5.2: 1000 nodes, with and without unreachable nodes

In the plots above, we can see, how making a substantial amount of the nodes unreachable impacts the eccentricity of all nodes. While the average degree of the nodes stay around the same.

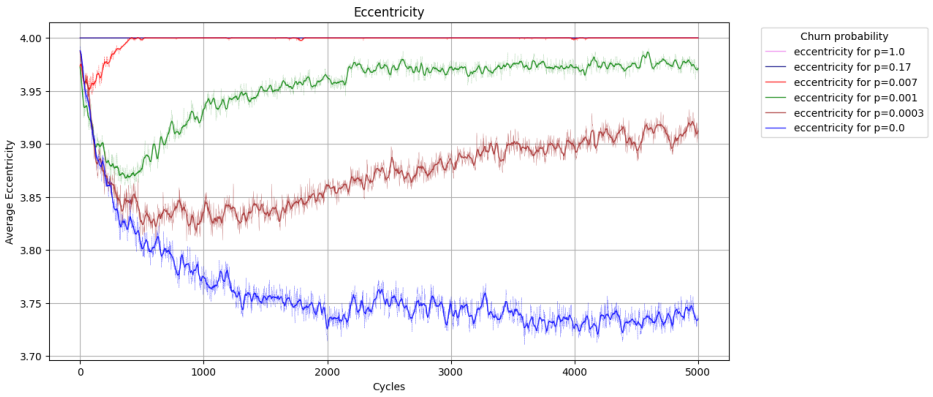
In our simulation, the NAT nodes will be selected at random in the building phase of the graph. They still have a probability to go offline and be replaced with a node with the same properties. The graphs in the following pages are aimed at providing an insight into the Bitcoin network, therefore the number of outgoing connections of each node is set to 8.



(a) 30% of nodes behind NAT



(b) 50% of nodes behind NAT



(c) 70% of nodes behind NAT

Figure 5.3: Network with 2000 nodes with different % NAT

In the plots above, we can clearly see how increasing the amount of nodes behind NAT reduces the overall eccentricity and makes the graph more compact, reducing its diameter from 5 to 4.

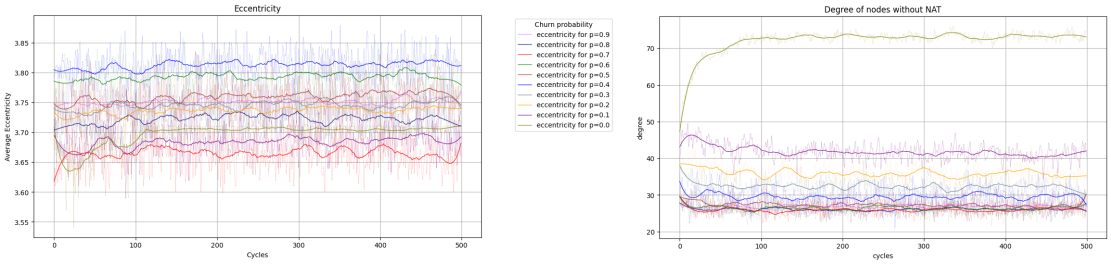
What this means for the overall network is, that the reachable nodes are closer to each other and form the center of the network, while the unreachable nodes are the furthest away from the core.

Chapter 6

Impact of limiting degree

In this chapter, we investigate the impact that limiting the amount of incoming connections a node can have has on the graph. We expect the nodes to converge in their behavior the more we limit the in-degree. The thought behind this is, that the fewer nodes are available to connect to, the less it will matter when a node joins and for how long it stays in the graph.

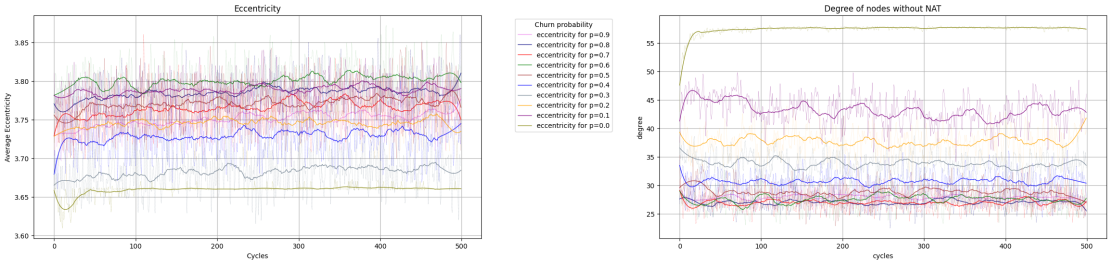
We will investigate the impact for a graph where all nodes are available and for a graph, where 70% of the nodes will be considered to be unreachable. To achieve the best visual effect, we will use a uniform distribution of churn probabilities on a graph with 1000 nodes.



(a) Eccentricity

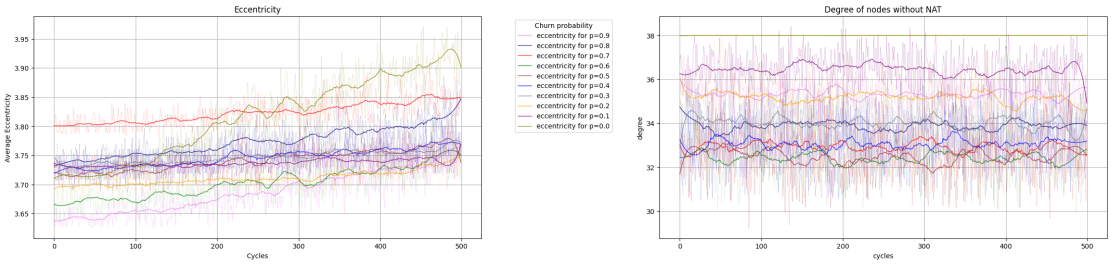
(b) Degree

Figure 6.1: Eccentricity and distance under max 70 incoming connections



(a) Eccentricity (b) Degree

Figure 6.2: Eccentricity and distance under max 50 incoming connections

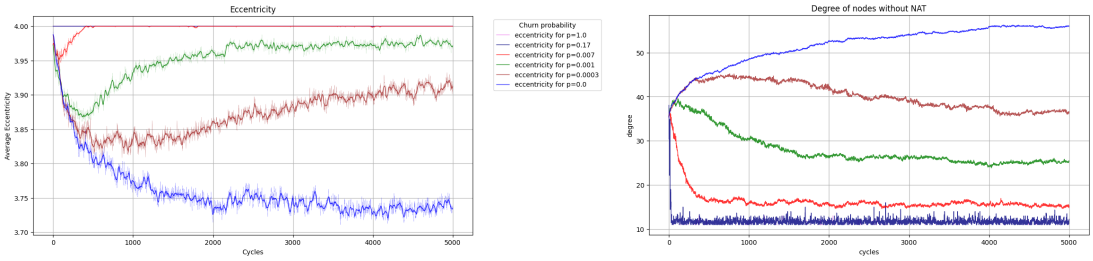


(a) Eccentricity (b) Degree

Figure 6.3: Eccentricity and distance under max 30 incoming connections

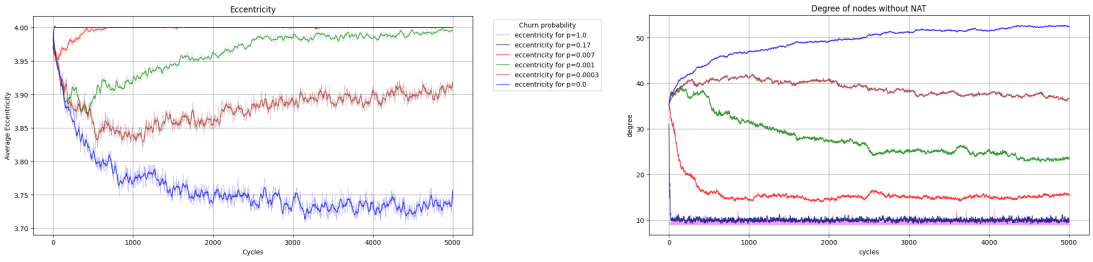
In the plots above, we can clearly observe how limiting the amount of incoming connections changes the network behavior and forces the churning nodes to take a more central role in the network. Without an artificial limit on the in-degree, the nodes that are always up will quickly fill up their incoming degree and dominate the network.

When simulating the Bitcoin P2P network, where a substantial part of the network is online for a much longer duration, the difference is less visible, unless we approach the minimum of the required degree that the graph needs to stay connected.



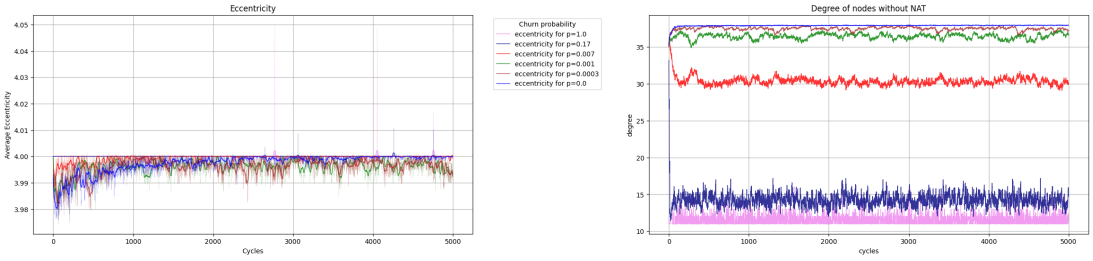
(a) Eccentricity (b) degree of nodes

Figure 6.4: (a) Eccentricity and (b) the degree of the nodes with an upper bound of 70 incoming connections, 2000 nodes



(a) Eccentricity (b) degree of nodes

Figure 6.5: (a) Eccentricity and (b) the degree of the nodes with an upper bound of 50 incoming connections, 2000 nodes



(a) Eccentricity (b) degree of nodes

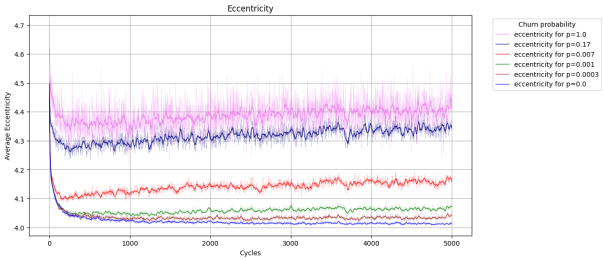
Figure 6.6: (a) Eccentricity and (b) the degree of the nodes with an upper bound of 30 incoming connections, 2000 nodes

6.1 Centrality of nodes

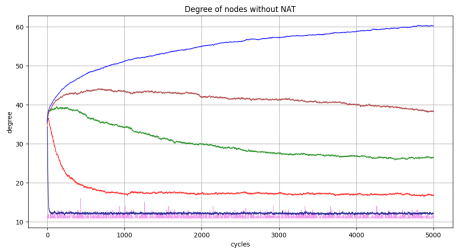
Over time, the network starts to stabilize. During this process, all nodes behind NAT are moved to the outside of the graph and all other nodes become more centralized.

$$C_D(v) = \frac{\text{degree of node } v}{\text{total number of nodes - 1}}$$

In the plots involving the average degree of the churning nodes, we can see, that the non-churning nodes become the central part of the network, with the highest centrality in the graph. From the plot with a cap on 50 incoming connections, we can see that the stable nodes approach that limit (50 incoming + 8 outgoing connections) after multiple thousand cycles, when the network stabilizes.



(a) 10000 nodes, 100 incoming connections

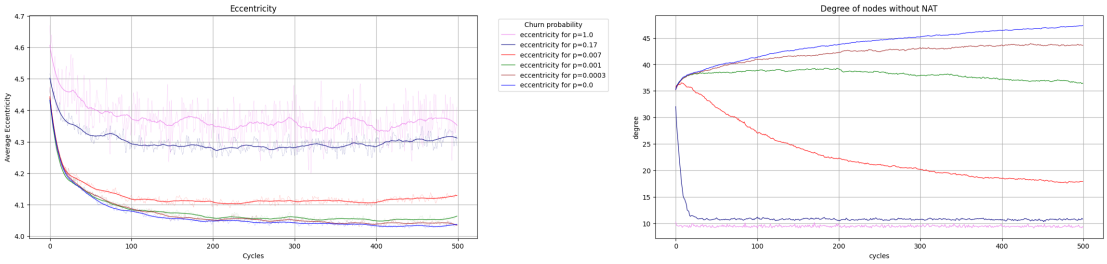


(b) 10000 nodes, 100 incoming connections degree

Figure 6.7: 10000 nodes, 70% of nodes behind NAT

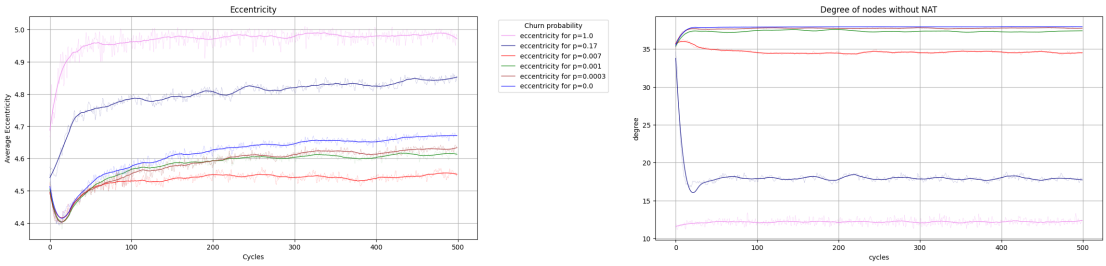
6.2 Simulation with 10000 Nodes and artificial upper bound

In the Bitcoin P2P-network, when a new node comes online, it quickly gets its incoming connections filled. In the following, we use 10000 nodes to simulate the real size of Bitcoin network. Like in graph 6.4,6.5,6.6 we introduce an artificial upper bound on incoming connections and investigate its impact.



(a) eccentricity (b) degree

Figure 6.8: 70% behind NAT, upper bound 50 incoming connections



(a) eccentricity (b) degree

Figure 6.9: 70% behind NAT, upper bound to 30 incoming connections

In 6.9(b) we can observe, how the stable nodes hit their upper limit after a short amount of time. Interesting is how the nodes that have a low churn rate first have a strongly decreasing eccentricity as they start to find each other. But as time goes on, the more volatile nodes are placed further away from the center cluster of the graph, causing the overall eccentricity to raise. This is very visible when we look at a simulation that was run over a long period of time 6.10(15000 cycles, 2000 nodes)

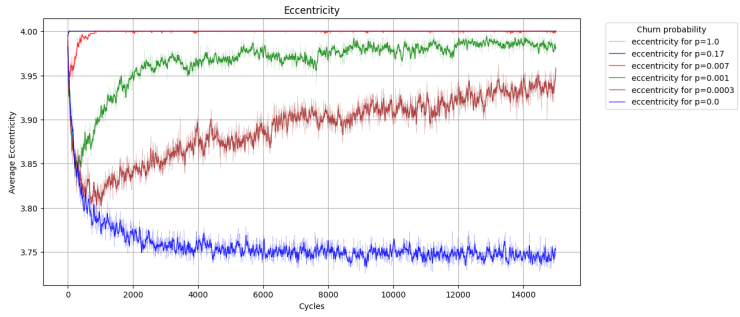


Figure 6.10: 15000 Cycles, 2000 nodes

Chapter 7

Conclusion

Simulating a large P2P network provided us with a deeper insight of how a real P2P network behaves and how churning nodes affect it. Through the collected data we have showed, how non-churning nodes find each other over time and how this affects their eccentricity and the eccentricity of the other nodes. This process also strongly affected the degree of the different nodes, as the longer a node stays online, the higher its degree becomes. Important was also to show how big of an impact unreachable nodes have on the network.

The simulation of such big P2P network has proven to be quite challenging, as at such large scale hardware bottlenecks and the exponential runtime of many graph algorithms become the main problem. By minimizing the amount of required graph accesses and graph operations, the desired scaling could still be achieved.
Every graph operation using the library would update the parameters of every single node. This could be partly avoided by minimizing all graph operations and moving graph-properties to external data structures. This design process proved to be beneficial and allowed us to collet a large amount of data.

Additional, one step that could still be taken is implementing the GPU cuda backend referenced in the documentation of the networkx [1] library.

Bibliography

[1] N. developers. networkx. [Online]. Available: https://networkx.org/

[2] F. D. S. und Netzdienste KASTEL. (2023) Bitcoin monitoring. [Online]. Available: https://www.dsn.kastel.kit.edu/bitcoin/index.html

[3] H. H. Matthias Grundmann. What peer announcements tell us about the size of the bitcoin p2p network. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000151755

[4] L. Z. F. G. Dawei Xu, Jiaqi Gao and J. Zhao. (2023, Nov.) Statistical and clustering analysis of attributes of bitcoin backbone nodes. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0292841