# Report for "Peer-to-Peer Backup" Simulation

Distributed Computing

Professor Matteo Dell'Amico

Delaram Doroudgarian - 5881909
Seyede Aida Atarzadeh Hosseini - 6936800

2024/2025

## 1. Summary of Assignment Goals

The main goal of this assignment is to simulate a peer-to-peer (P2P)[1] backup system that utilizes erasure coding for long-term data storage. P2P systems operate through decentralized collaboration among nodes, enabling resource sharing, scalability, fault tolerance, and efficient data distribution without relying on a central authority. Peers within this system connect and disconnect regularly, and some may experience failures resulting in the loss of stored data. Erasure coding is employed as a method to safeguard data against node failures by dividing it into n encoded blocks, with the ability to recover the data using only k blocks, thereby ensuring data integrity even when some blocks are lost.

By simulating the four participant components including **peers, data encoding, data storage and failure scenarios,** the simulation evaluates how well the peer-to-peer system can preserve data availability and reliability over time, accounting for factors like connectivity, redundancy, and the parameters of erasure coding.

## 2. Nodes Criteria

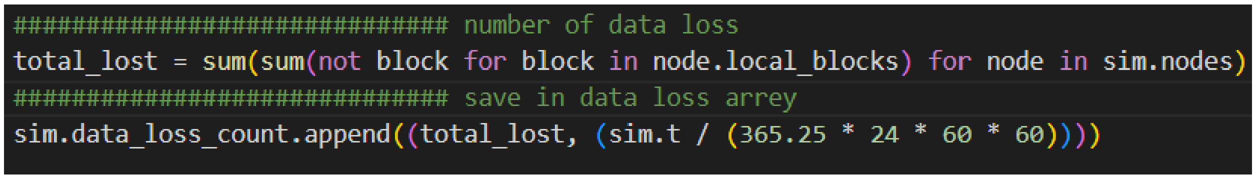
- name: A string representing the node's name.
- n: An integer indicating the number of blocks in which the data is encoded.
- k: An integer representing the number of blocks sufficient to recover the entire node's data.
- data_size: An integer indicating the amount of data to back up (in bytes).
- storage_size: An integer representing the storage space devoted to storing remote data (in bytes).
- upload_speed: A float representing the node's upload speed (in bytes per second).
- download_speed: A float representing the node's download speed (in bytes per second).
- average_uptime: A float representing the average time the node spends online.
- average_downtime: A float representing the average time the node spends offline.
- average_lifetime: A float representing the average time before a crash and data loss.
- average_recover_time: A float representing the average time after a data loss for the node to recover.
- arrival_time: A float representing the time at which the node will come online.

- selfish: A boolean indicating whether the node is selfish (default is False).

## 3. Implementation and Completion of Peer-to-Peer Backup

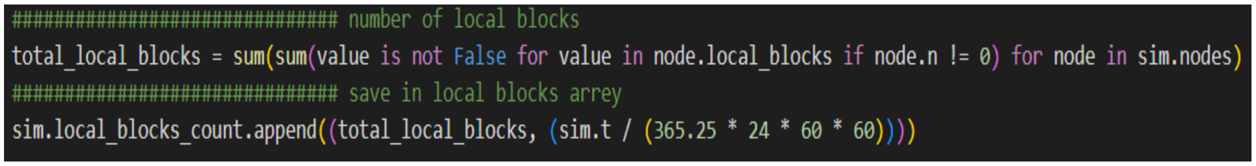
- First, we focused on completing the existing code to ensure it runs smoothly and correctly with both configurations. Then, to analyze the system, we decide to choose 3 metrics as follow:

1. The average of Data loss over time:
   When a node fails, it loses all its locally stored data. The number of lost blocks is computed by given code in below. This implementation is inside the Fail.process() function. The computed values are stored in sim.data_loss_count to be used for plotting.



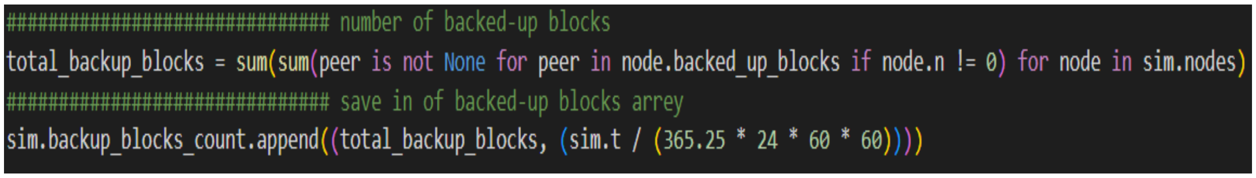
```
################################ number of data loss
total_lost = sum(sum(not block for block in node.local_blocks) for node in sim.nodes)
################################ save in data loss arrey
sim.data_loss_count.append((total_lost, (sim.t / (365.25 * 24 * 60 * 60))))
```

2. The average of local blocks over time:
   Every time a transfer is completed, the total count of locally stored blocks is updated using below code. This implementation is inside the TransferComplete.process() function. The computed values are stored in sim.local_blocks_count to be used for plotting.



```
############################## number of local blocks
total_local_blocks = sum(sum(value is not False for value in node.local_blocks if node.n != 0) for node in sim.nodes)
############################## save in local blocks arrey
sim.local_blocks_count.append((total_local_blocks, (sim.t / (365.25 * 24 * 60 * 60))))
```

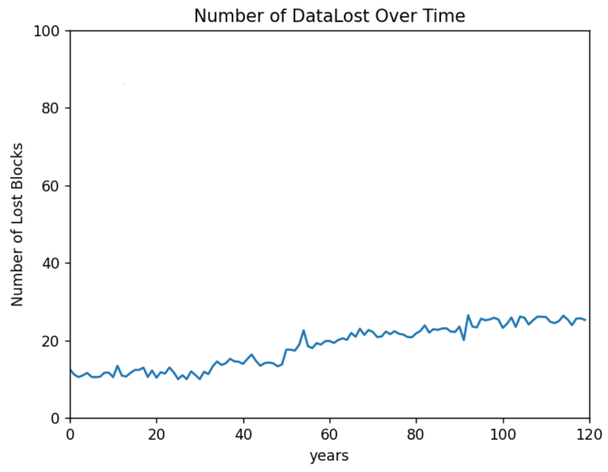
3. The average of Backed-up blocks over time:
   Every time a backup operation is completed, the count is updated using below code. This implementation is inside the TransferComplete.process() function. The computed values are stored in sim.backup_blocks _count to be used for plotting.

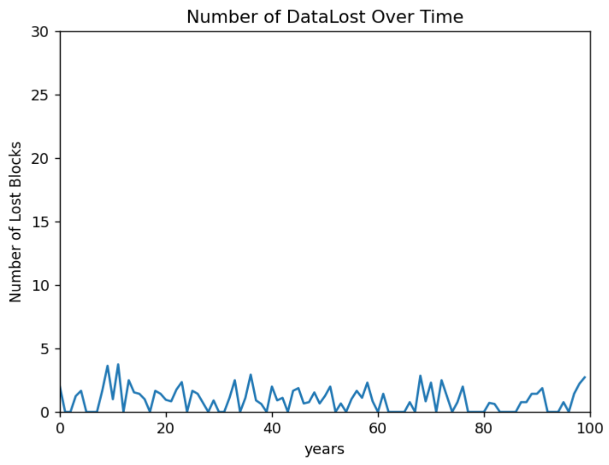


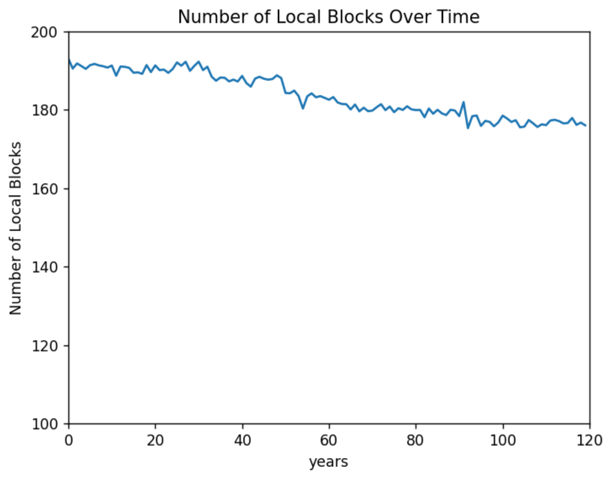
```
############################## number of backed-up blocks
total_backup_blocks = sum(sum(peer is not None for peer in node.backed_up_blocks if node.n != 0) for node in sim.nodes)
############################## save in of backed-up blocks arrey
sim.backup_blocks_count.append((total_backup_blocks, (sim.t / (365.25 * 24 * 60 * 60))))
```

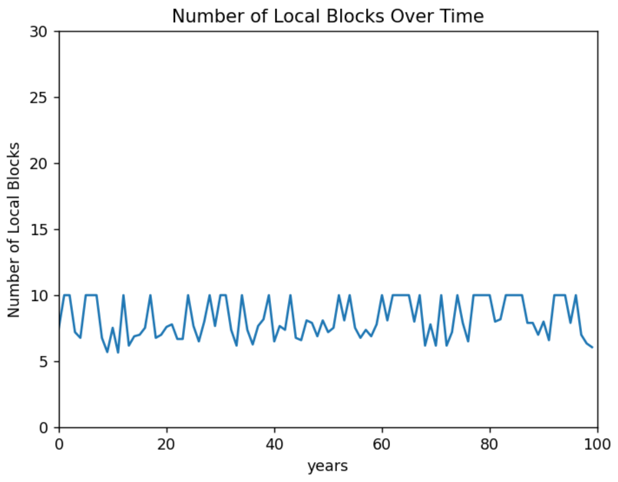
## 4. Plots of P-2-P and client-server configuration files

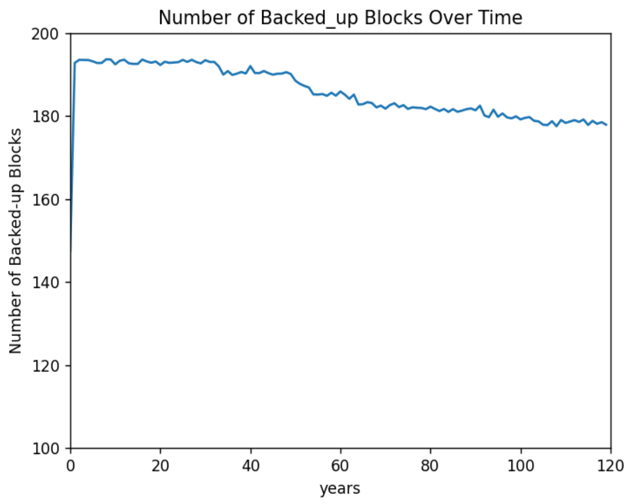
The left plots belong to P2P CFG while the right ones are the result of C-S CFG

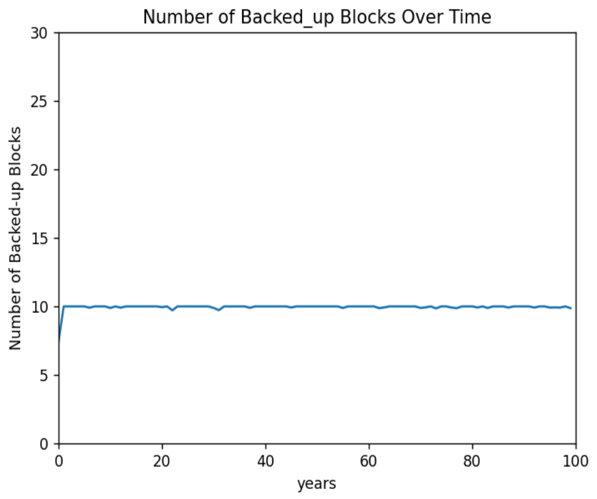












- Comparison of Data Loss Over Time :

  - In the left plot (P-2-P), initially, the number of lost blocks is low. Over time, this number increases, especially after 60 years, where more fluctuations are observed. In the later years, the number of lost data blocks stabilizes around 20 to 30 blocks.
    **Analysis:** As time progresses, the probability of data loss increases, possibly due to node failures or recovery processes.

  - In the right plot (client-server), the amount of data lost over time is very low, remaining close to zero in most years. In some instances, there are slight increases, but the system quickly stabilizes.
    **Analysis:** This trend indicates that the Client-Server configuration is highly stable in preventing data loss. This stability can be attributed to the role of servers as more reliable and stable nodes.

- Comparison of Local Blocks Over Time :

  - In the left plot, at the beginning, there are around 195 local blocks in the system. Over time, this number gradually decreases but remains above 120 local blocks.
    **Analysis:** The decline in local blocks indicates that some nodes are losing their data.

  - In the right plot the number of local blocks shows minor fluctuations between 6 and 10 blocks. The reason of these small variations is When a node fails, it loses its local data, causing a temporary decrease in the number of local blocks. After that, the recovery process starts, restoring the lost data and leading to an increase in the number of local blocks.
    **Analysis**: This behavior demonstrates the system's resilience against node failures and its ability to maintain data integrity.

- Comparison of Backed-Up Blocks Over Time

  - In the left plot, initially, there are around 195 backed-up blocks in the system. Over time, this number decreases especially after 50 years and by the end of simulation, reaches around 170.
    **Analysis:** The decrease in backed-up files can be a result of node failures, connectivity changes, and backup inefficiencies**.**

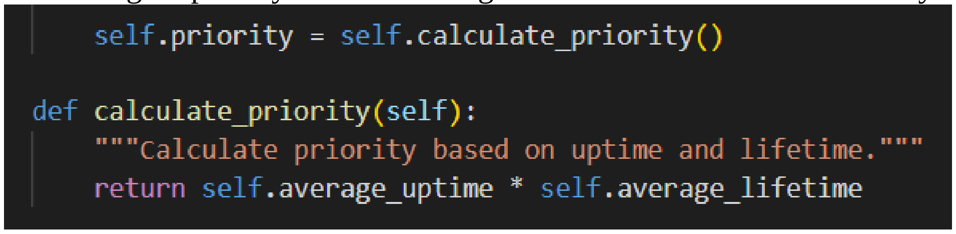
- The right plot is extremely stable, with the number of backed-up blocks remaining constant over time (around 10 blocks because there is 1 client and its number of blocks to be encoded (n) is 10). There are no noticeable changes in the trend.
  **Analysis:** This remarkable stability is due to the fact that, in the Client-Server model, servers are responsible for continuous and reliable backups. Unlike peer-to-peer nodes, servers are usually always available and do not exhibit variant behavior.

## 5. Implementation of P-2-P backup with Priority_Based extension

- We decided adding **priority-based node selection** during backup and restoration. This extension will implement a mechanism where nodes with higher reliability (higher uptime or longer lifetime) are prioritized for storing and restoring data.

  1) First we add Priority Calculation to the Node Class:

     - Each node is assigned a priority score based on its average_uptime and average_lifetime.
     - Higher priority scores are assigned to nodes with better reliability metrics.

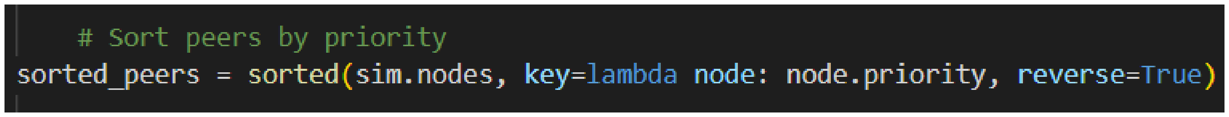


```
    self.priority = self.calculate_priority()

def calculate_priority(self):
    """Calculate priority based on uptime and lifetime."""
    return self.average_uptime * self.average_lifetime
```

  2) Then Sort the nodes based on priority for backing up:

     - During backup, the uploader prioritizes nodes with higher scores as targets for storing data.

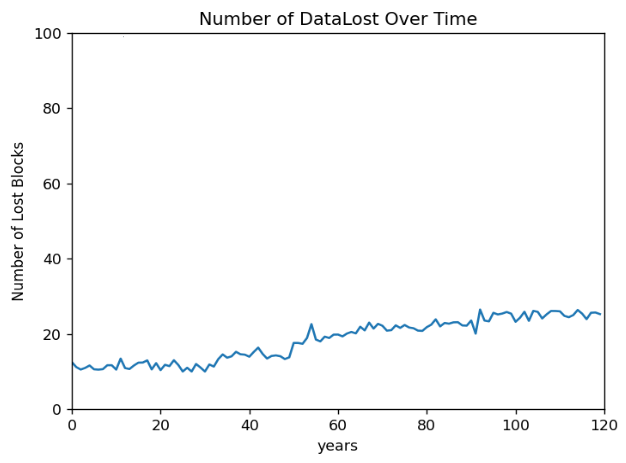


```
    # Sort peers by priority
sorted_peers = sorted(sim.nodes, key=lambda node: node.priority, reverse=True)
```

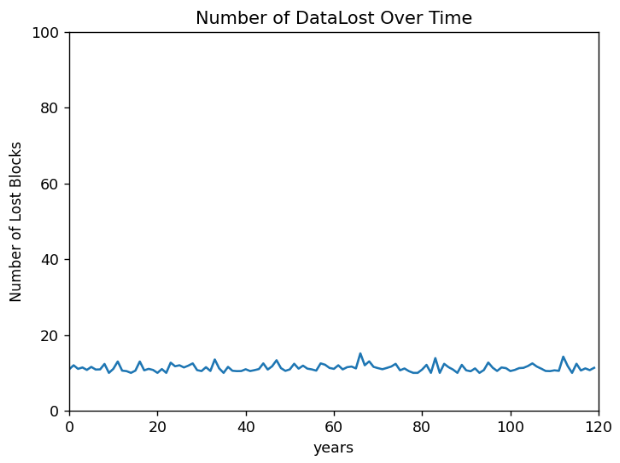
- **Benefits:**

  - Improves the efficiency of data backup and restoration.
  - Reduces the likelihood of data loss by utilizing reliable nodes for critical operations.

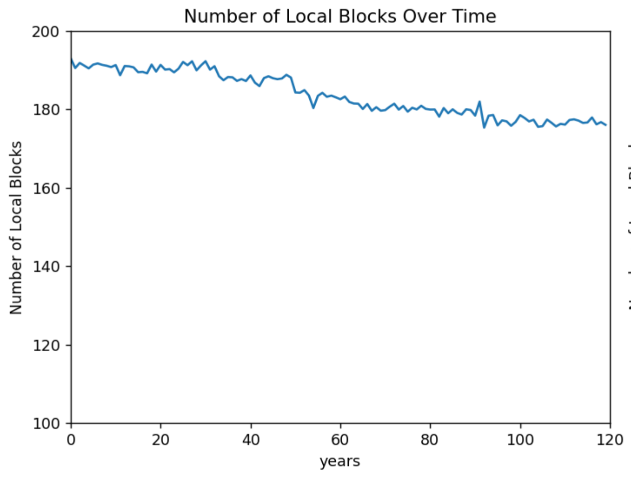
## 6. Plots of p-2-p Backup with Priority_Based extension



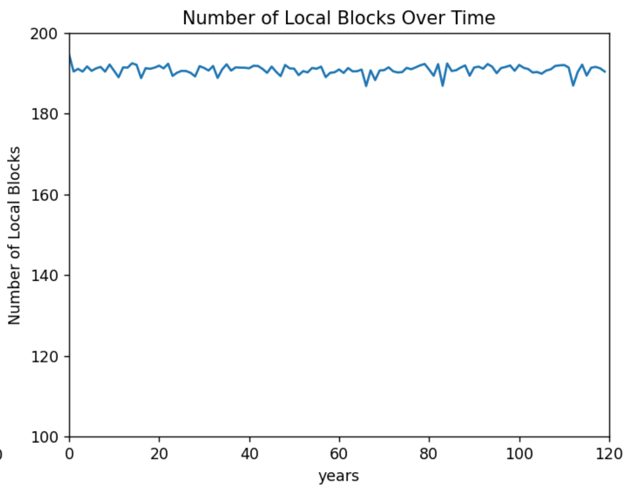
without priority_based extension



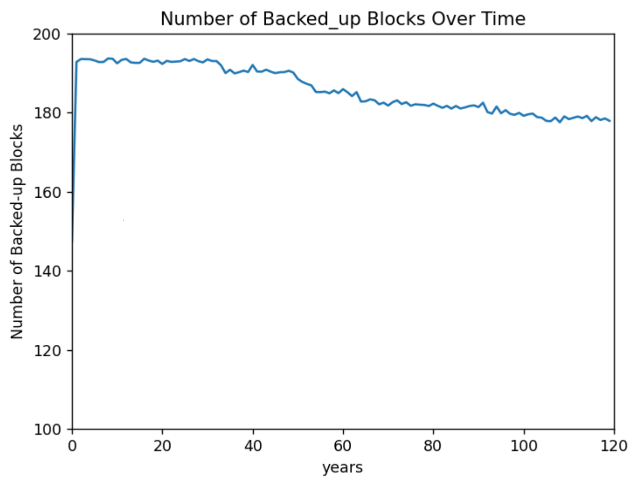
with priority_based extension



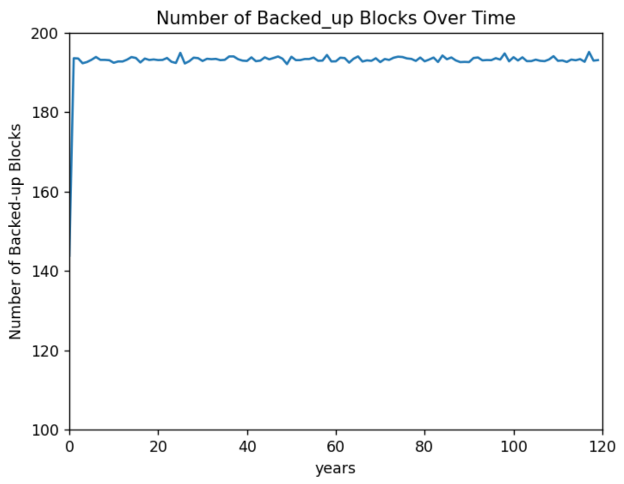
without priority_based extension



with priority_based extension



without priority_based extension



with priority_based extension

- Comparison of Data Loss Over Time :
  - The left chart (without the priority-based extension) shows that the number of lost blocks increases over time, reaching more than 30 lost blocks by the end of the simulation.
  - The right chart (with the priority-based extension) shows that the number of lost blocks is significantly lower, and the increase in lost data is much more gradual. This indicates that the priority-based extension improves data recovery and reduces data loss.
- Comparison of Local Blocks Over Time :
  - In the left chart (without the priority-based extension), the number of local blocks decreases from about 190 blocks to below 170 blocks. This means that some nodes are unable to keep their data.
  - In the right chart (with the priority-based extension), the number of local blocks remains relatively stable. This indicates that prioritizing more reliable nodes for storage helps nodes maintain their data, increasing the stability of local blocks.
- Comparison of Backed-Up Blocks Over Time
  - In the left chart (without the priority-based extension), the number of backed-up blocks gradually decreases over time. This decline shows that some blocks are lost due to node failure or the unavailability of some nodes because of the connectivity changes, so the backup process is ineffective.

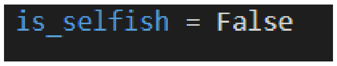
- In the right chart (with the priority-based extension), the number of backed-up blocks remains almost constant. This shows that using the extension has improved the efficiency of data storage across nodes, making it easier to recover data when needed.

- **Overall,** adding the priority-based extension to the P2P backup system significantly improves performance. This approach enhances system stability, increases node reliability, and prevents long-term data loss, making the system more stable and efficient.

## 7. Implementation of p-2-p backup with Selfish Node extension

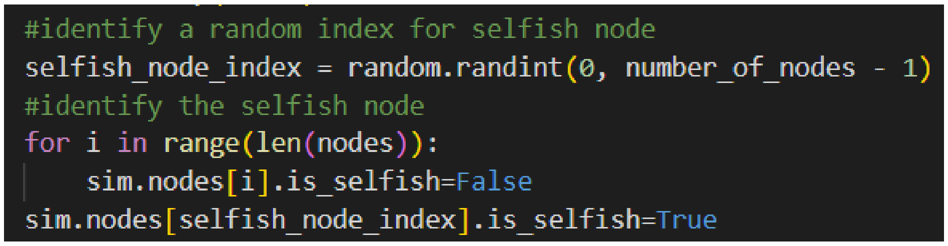
- After implementing previous extension, we decided to use from the **Selfish Node extension** during backup and restoration. There are different definitions for selfish node base on system's needs. In our scenario, the selfish node doesn't upload anything and doesn't backup for other nodes.
  Selfish behavior is not proposed as a solution to improve efficiency but rather as a challenge for analyzing and designing more stable and resilient systems.

  1) First we specify one selfish node:
     - Defining the 'is_selfish' variable in config file with default value of False



```
is_selfish = False
```

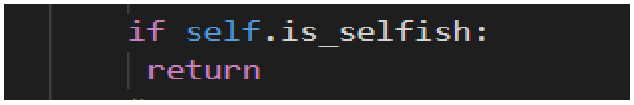
     - Choosing 1 node randomly and changing its selfish attribute to True.



```
#identify a random index for selfish node
selfish_node_index = random.randint(0, number_of_nodes - 1)
#identify the selfish node
for i in range(len(nodes)):
    sim.nodes[i].is_selfish=False
sim.nodes[selfish_node_index].is_selfish=True
```

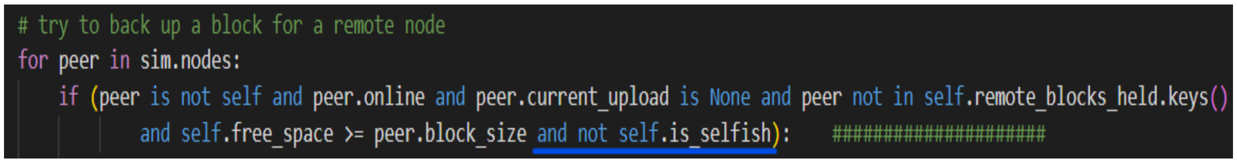
  2) Then into the "schedule_next_upload" function, selfish node doesn't upload anything.

     - it will be checked if a node is selfish, it returns.



```
    if self.is_selfish:
        return
```

3) Also into “schedule_next_download” function, selfish node doesn’t backup any block for remote nodes.

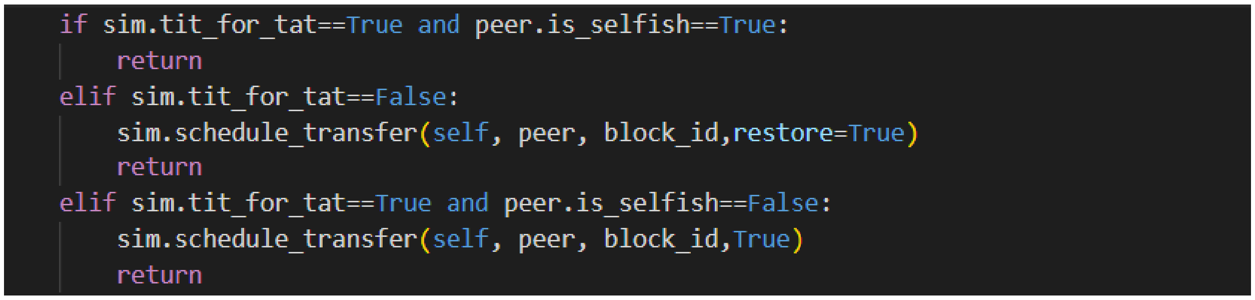


```
# try to back up a block for a remote node
for peer in sim.nodes:
    if (peer is not self and peer.online and peer.current_upload is None and peer not in self.remote_blocks_held.keys()
            and self.free_space >= peer.block_size and not self.is_selfish):    #####################
```

- **Combining the selfish node extension with tit-for-tat strategy :**

  As we expect, the existing of a selfish node should affect on system’ performance negatively. So we decided to combine the selfish node extension with Tit-for-Tat strategy to avoid the selfish behavior of some nodes. according to Tit-for-Tat strategy, there should be cooperation among all nodes which means the cooperative nodes don’t interact with non-cooperative nodes (e.g. selfish nodes).

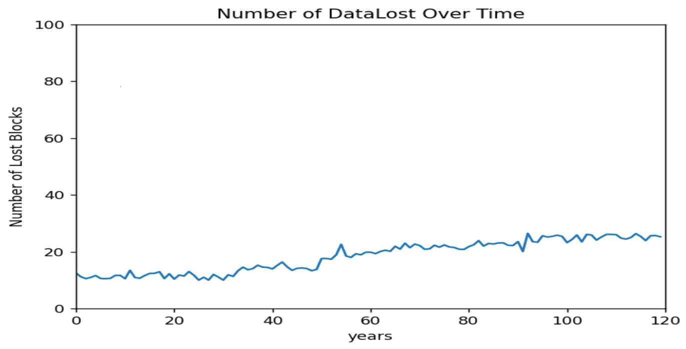
  4) In “schedule_next_upload”, the cooperative nodes only upload for non-selfish nodes:
     - For both restoration and backing up, the origin nodes check the peer isn’t selfish, otherwise, they don’t upload any blocks for that selfish node.



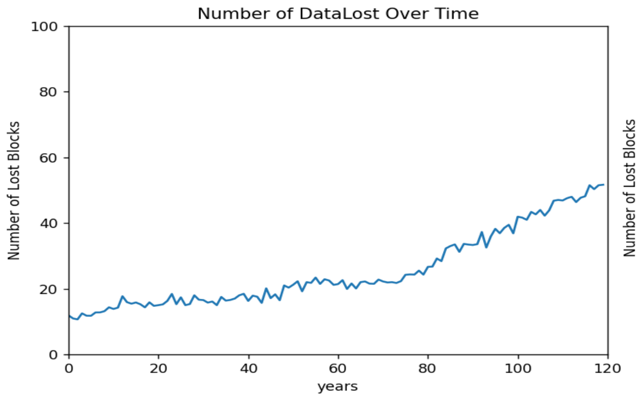
```
    if sim.tit_for_tat==True and peer.is_selfish==True:
        return
    elif sim.tit_for_tat==False:
        sim.schedule_transfer(self, peer, block_id,restore=True)
        return
    elif sim.tit_for_tat==True and peer.is_selfish==False:
        sim.schedule_transfer(self, peer, block_id,True)
        return
```

- **Benefits:**
  - Maintains the network efficiency by preventing selfish nodes from exploiting the network by limiting their access to resources if they do not contribute.
  - promotes mutual cooperation by prioritizing nodes that actively participate, ensuring fairness and balanced interactions.

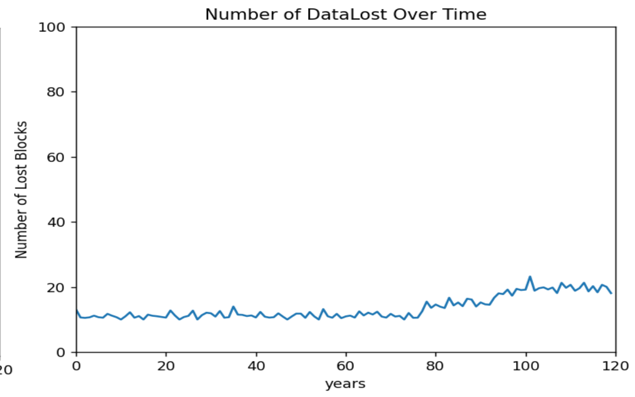
## 8. Plots of p-2-p Backup with Selfish Nodes extension



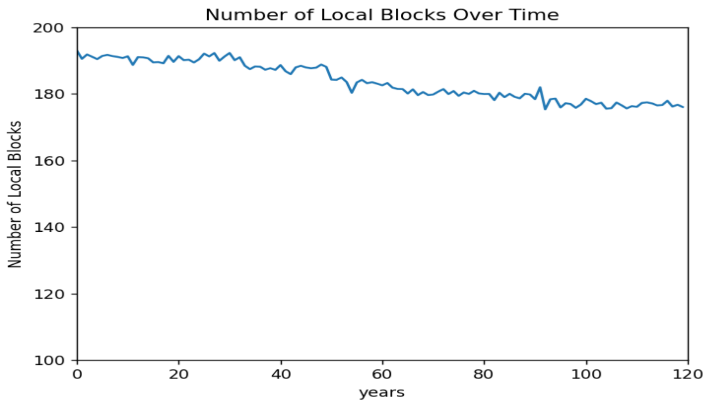
without selfish node extension



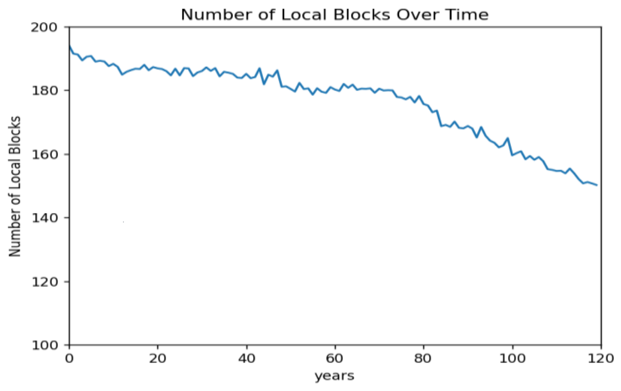
with selfish node extension

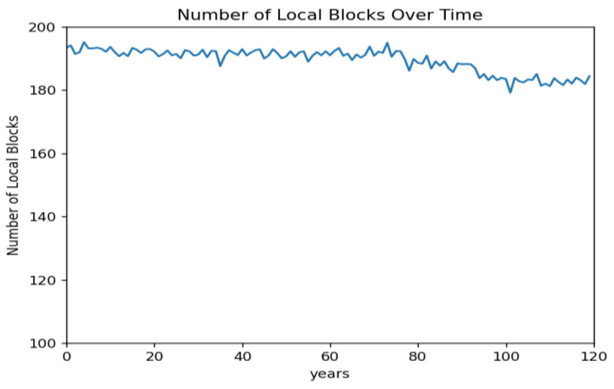


combination of selfish node and TfT strategy



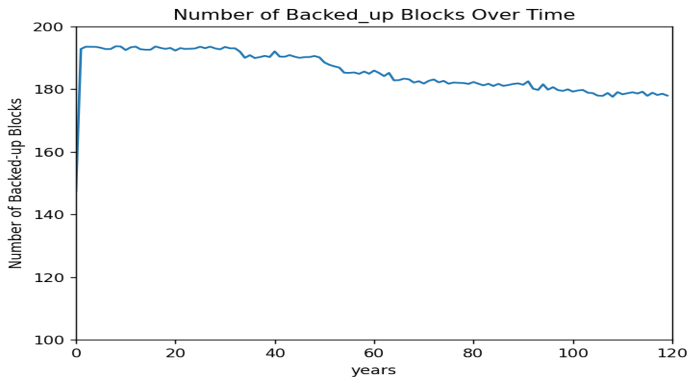
without selfish node extension



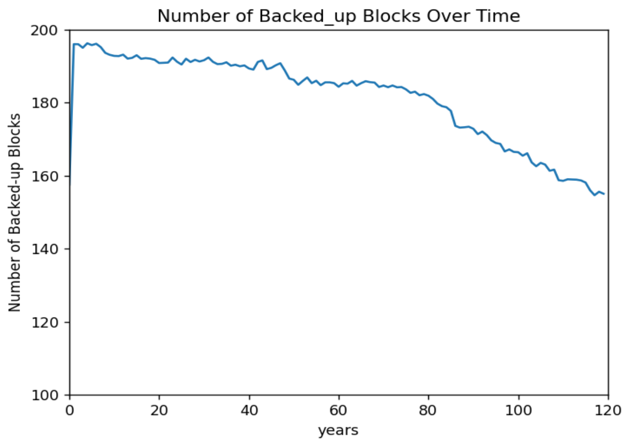


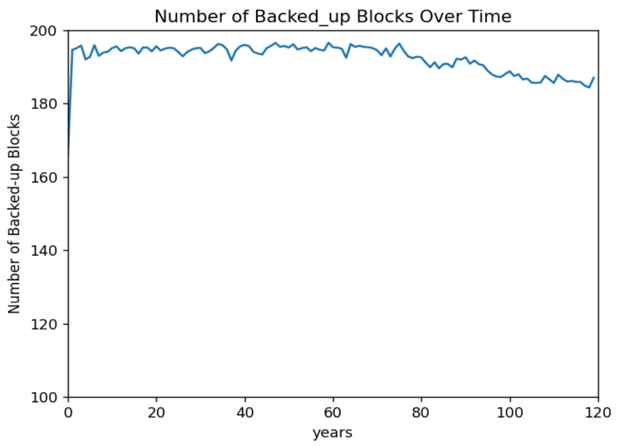
with selfish node extension

combination of selfish node and TfT strategy



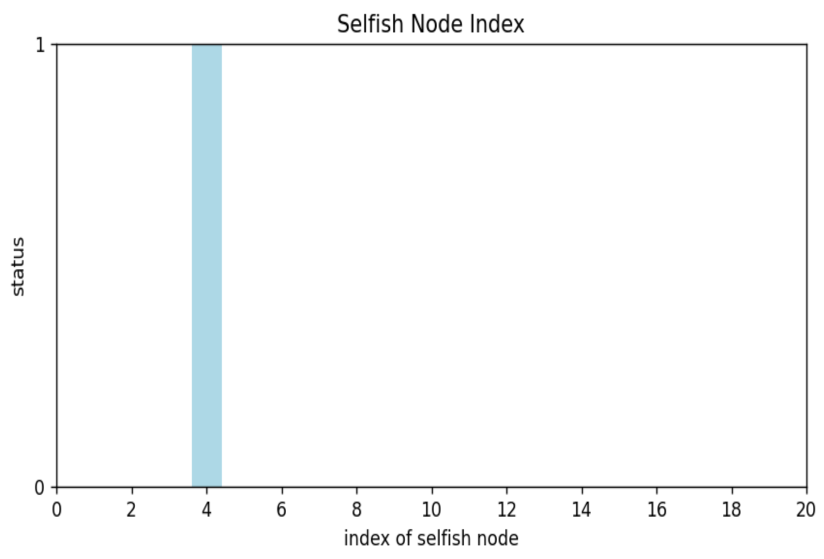
without selfish node extension





with selfish node extension

combination of selfish node and TfT strategy



- Comparison of Data Loss Over Time :

  - In the “without selfish node extension” graph, the data loss gradually increases over time.

  - In the “with selfish node extension” graph, data loss is higher as time progresses.

  - In the “combination of selfish node and TfT strategy” graph, data loss is lower compared to the selfish-only scenario.

- Comparison of Local Blocks Over Time :

  - In the “without selfish node extension” graph, the number of local blocks gradually decreases but remains at an acceptable level.

  - In the “with selfish node extension” graph, the decrease in local blocks is more significant, as selfish nodes rely on receiving data but do not contribute to upload others' data.

  - In the “combination of selfish node and TfT strategy” graph, local block stability is better than in the selfish-only scenario.

- Comparison of Backed-Up Blocks Over Time
  - In the “without selfish node extension” graph, the number of backed-up blocks gradually decreases but remains stable.
  - In the “with selfish node extension” graph, the decrease is more significant.
  - In the “combination of selfish node and TfT strategy” graph, the number of backed-up blocks remains more stable compared to the selfish-only case.
- **Overall,** as the selfish node doesn’t upload any restored block for other nodes, the number of data loss increases. This is also a reason for reduction of local and backed-up blocks because the uploading part of system becomes inefficient. With Selfish Node + TFT, the system becomes more stable, as TFT promotes cooperation among nodes, preventing selfish nodes from over-exploiting the network. In networks where selfish nodes may exist, applying a Tit-for-Tat (TFT) strategy can decrease their negative impact, ensuring better stability of backed-up and restored data over time.