Promethon3: new way to look at the next generation of Internet

Intro

Ethereum uses a world state to hold states of each account . World state is implemented by using Merkle Tries which makes finding accounts and getting the hash of the whole state faster and with much less process complexity. If we didn't use Merkle Trie and used normal arrays instead, finding and hashing would have process complexity of O(n) but now it has O(log(n)) which is much more efficient.
But the size of this world state is growing rapidly and there hasn't been any efficient solution for shrinking it down.
There has been some ideas that never came to life that best of them are

- State rent (precursor to state expiry), original 2015 proposal: https://github.com/ethereum/EIPs/issues/35
- State expiry A state expiry and statelessness roadmap - HackMD

All of these solutions are experimented in this document
https://hackmd.io/@vbuterin/state_size_management

Process complexity of ethereum transaction handling is O(log(n)) and storage complexity of ethereum is O(n) so if we want to make any changes to the current version of ethereum and upgrade it, we should keep in mind that not to make those complexity worse because then our upgrade wouldn't be completely an upgrade and just be trade off between features.

Our project promises to reduce unused accounts without compromising any time complexity or adding complexity to smart contract development. This project fixes the logic of ethereum in which people can have their data saved forever just by giving a saving fee and even one thousand years later, nodes should hold that data even though the person that put it there doesn't need it anymore. And in order for us to achieve this without compromising any storage or time complexity, we useLinked Merkle tree (a Merkle tree which nodes form a linked list together).

In this approach we set an expiration date for each account and if that account wasn't accessed until that expiration date, it will be deleted but if it was accessed and used, the expiration date will be extended.

In order to delete an account that reached its expiration date, we had to go through all accounts in each time period; but this method would take time complexity of O(n), let n be number of accounts, which is a disaster in case of blockchain projects because the n can be a very big number and impossible to be iterated through.

So we decided to use a new kind of linked list and mapping in order to have all the accounts in specific order based on the time they have left to be deleted. So that accounts that should be deleted first, be in the leftist place of the list. In this method we only need to check the leftist account and if it hasn't reached its expiration date, it means that no other account needs to be checked because we are sure that they still have time to be deleted.

In this method, we have a world state database just like ethereum and we also have a new type of linked list combined with mapping to be able to reach the oldest accessed node with O(1) and not make any of the modification operations that ethereum uses.

Related Works

There have been a lot of attempts to overcome this issue around the world state of ethereum because it has been a concern for the network from the beginning.
Some of these related works are brought here:

- **The Stateless Client Concept**, original ethresear.ch post (2017):
  https://ethresear.ch/t/the-stateless-client-concept/172
  (see also EthHub)In reality, “stateless” does not really mean “no state”! What it actually means is that you made state someone else’s problem. This proposal takes a completely different approach that is not compatible with the current version of ethereum world state.

  It suggests that we don't need to have all the world state on our full node to validate blocks but instead we can have only the Merkle trie root hash and for verifying use Merkle branches. Despite all the benefits that this approach can bring us; it makes ethereum processes very dependent on other oracles and databases which can cause the network not to work flawlessly and the current user experience will be altered.

- **State expiry**
  https://notes.ethereum.org/@vbuterin/verkle_and_state_expiry_proposa

  State expiry suggests that we remove state that has not been recently accessed from the state every once in a while like each year.
  This approach requires the core to switch to Verkle-Trie so that when we want to bring back accounts that have been deleted, we won't need to use large amounts of data to prove its previous state.
  State expiry has a time period and every once in a time period, it cleans up the accounts that haven't been used in the previous time period, the time period suggested to be 1 year.

  One of the problems of State expiry is that it has process complexity of O(n*log(n)). This is a lot on the scale of ethereum. The other problem is that state expiry only solves problem of having too many accounts in the time between two periods(in this case, 2 years) and we can have a big number of useless accounts in the start of the period and our clients should carry those accounts for nearly a complete period (which in our case is 1 year).

  The other problem that should be addressed is the fact that how can we find the best period of time; because if we set the period for too short, the process will be massive because (n*log(n)) is a big number if you consider the fact that (n) in the current version of ethereum can be up to 2^60. And if we set the period for too long, we technically haven't solved any problem and we can have those useless accounts for too long.

  The last thing that should be noted is the fact that the current way that ethereum wants to store data is not perfect. When a new data is added to World state, it has a process complexity of O(log(n)) and it takes storage space of O(n) if n be the number of bits. An ethereum client receives an amount of gas to cover the work that it has done and storage that it has used up. But all this money is received by one client and the data

that it wants to store should remain in the blockchain forever or at least for one period of time of State expiry and in the meantime all other nodes should contain data even if they came to the network after they were stored.

- **State rent (precursor to state expiry)**, original 2015 proposal: (https://github.com/ethereum/EIPs/issues/35)

  This article suggests that we should stop charging for **fee = bytes** and use **fee = bytes * time** but the way it wants to do it is highly dependent on some unknown client that finds accounts that should be deleted and send delete transactions to the network and in the network, nodes calculate the total balance of the account by subtracting the rent from it's balance and see weather it reaches zero or less, if it reached, they delete it from the world state.

  The complete algorithm for this method is to use the current version of ethereum merkle-patricia-trie. And have a function in every node that takes an account and check if it should still stay in our storge by subtracting its **value** by **(bytes * (block passed since the last access) * (fee for each byte per block)** and if this equation equals to zero or less, then the account should be deleted.

  But all the nodes shouldn't check every account to see whether or not they should stay in the storage every block that passes because the time complexity would be massive and unreasonable. But for the network to work correctly, we must ensure that there will always be at least one node that searches for accounts that should be deleted and reports them to the network.

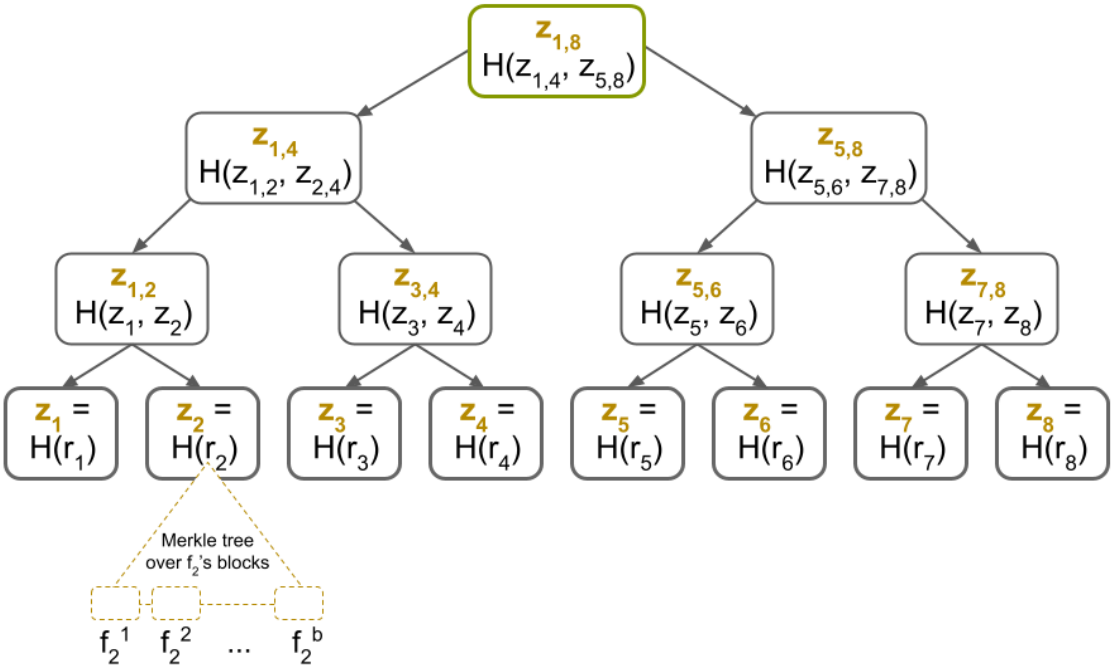
  This article didn't propose any solution for ensuring the existence of that node and didn't propose any solution to reduce the time complexity that checks every account to find the zero balance.

Priminaly Construction

The most important data structures used in our project is Merkle Patricia Trie:

Merkle Patricia Tree: Merkle tree or hash tree is a tree that is used for data verification. In most data structures when you have the hash of the whole database, if you change just one element in the database, in order to get the new hash of the whole database, you have to do the hash function for all the elements in the database which takes time complexity of O(n) but in Merkle tree, it takes only time complexity of O(log(n)). It is a tree data structure where each non-leaf node is a hash of its child nodes. All the leaf nodes are at the same depth and are as far left as possible.

This tree is usually used in systems that require to compare the whole state of more than one storage and check whether they are similar or not.



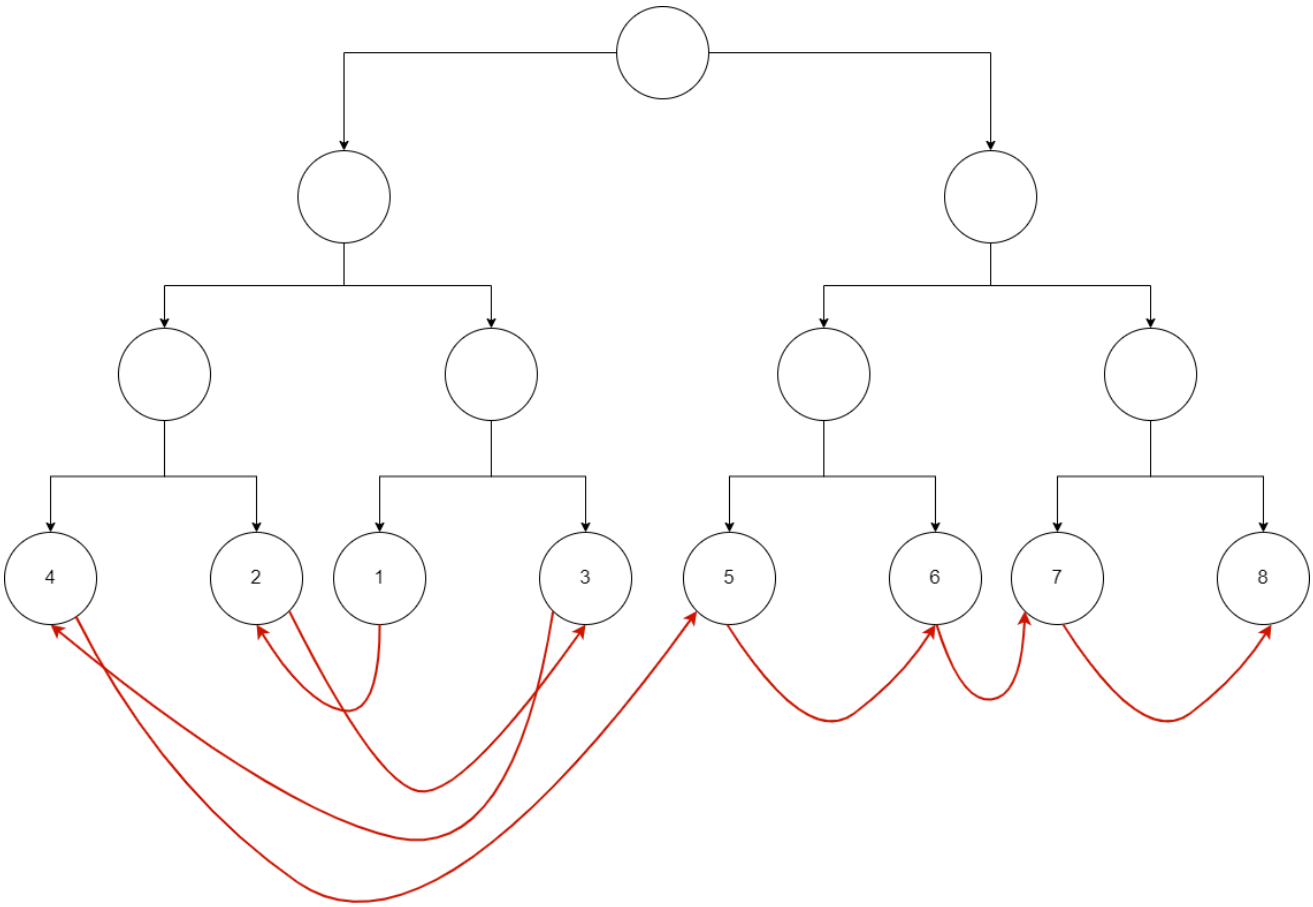
More info: https://www.geeksforgeeks.org/introduction-to-merkle-tree/

Merkle trees were invented by Ralph Merkle in 1988 in an attempt to construct better digital signatures.
Original paper: http://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkle.pdf

Linked Merkle Patricia Trie: A Linked Merkle Trie is a modified version of a Merkle Trie where the leaf nodes are linked together in a linked list fashion. Each leaf node in the trie contains a value and a "next" pointer that points to the next leaf node in the list. The leaf node with the

oldest interaction is considered the head of the list, while the leaf node with the newest interaction is considered the tail of the list.

Similar to a standard Merkle Trie, a Linked Merkle Trie maintains the time complexity of O(log(n)) for operations such as insertion, update, addition, and removal. This is achieved by utilizing the underlying Merkle Trie structure, where each non-leaf node is a hash of its child nodes.
The modification in a Linked Merkle Trie allows for efficient traversal of leaf nodes in a fast manner, enabling easy access to the oldest and newest interactions.ensures time complexity of O(log(n)) by using the combined linked-list and balanced binary trees (Like Merkle Patricia tries).



Solution

This project overcomes the issue of having too many unused accounts on any account-state-based blockchains like ethereum.

We made a rule for every account on the ethereum world state to be deleted if it hasn't been accessed for a certain time. The current method for saving data only costs one time on the execution of the transaction that saves the data and after that, the data will always stay on the blockchain and this causes the problem of having too many unused accounts and contract data because there is no cost for keeping the data.

We accomplish this by setting an expiration date for each account which indicates the number of blocks that are left for the network to delete the account. In this method, the network knows the last time every account was used by saving the block number in which there was a transaction that this account was modified on. This value is updated whenever a transaction that this account is modified on executes and saved on the blockchain. So the last modified value of each account represents its last time use.

To check if a certain account needs to be deleted, we only should reduce the current block height by the last modified value of the account and if it was more than the number of blocks that each account can stay on the world state without being deleted, it should be deleted but if it wasn’t more and was less or equal to that number, it should stay there on the blockchain.

Problem with this method is that checking each account to see whether or not they reached their expiration date requires iteration over the whole data and in ethereum case which is more than 100GB, is sth that surely can not be done in every block creation which is only around 12s.

Our solution is to arrange all the accounts in ascending order of their last modified value. If we arrange all the accounts in ascending order and put them in an array, then if we only control the account with the minimum last modified value, we can tell whether other accounts have reached their expiration date or not.

The problem with this approach is that if any account does a transaction and its position changes in the array, rearranging the accounts would require an execution with time complexity of O(n).

Executing any transaction in the current implementation of ethereum has the time complexity of O(log(n)) and changing it to O(n) would not be an improvement and execution of this many transactions with O(n) with the current hardware is impossible.

Solution for this problem is to use Linked Merkle Patricia tree instead of an array to hold the accounts with ascending order. Using Linked Merkle Patricia tree allows us to rearrange the accounts with time complexity of O(log(n))

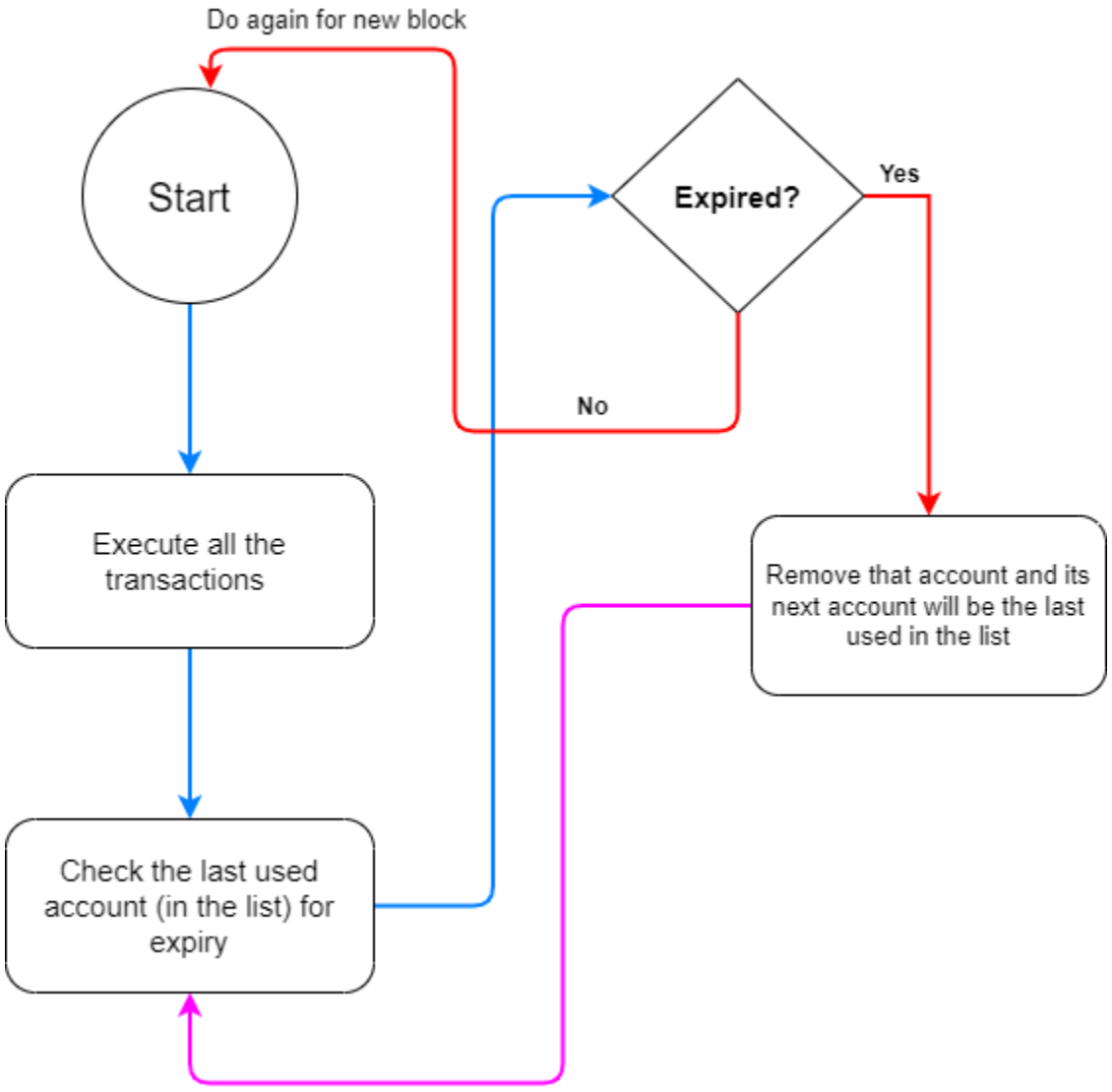
We also can achieve this by other trees of sorting with low time complexity but using this approach, would stop lots of storage duplications.

components

- First we should have all the accounts in the merkle patricia trie and our leaf nodes have the value of last modified, address of previous and the next account in the list of accounts.
- We should have a variable somewhere that stores the addresses of the first and last accounts in the linked list of accounts.

Algorithm of executing a block

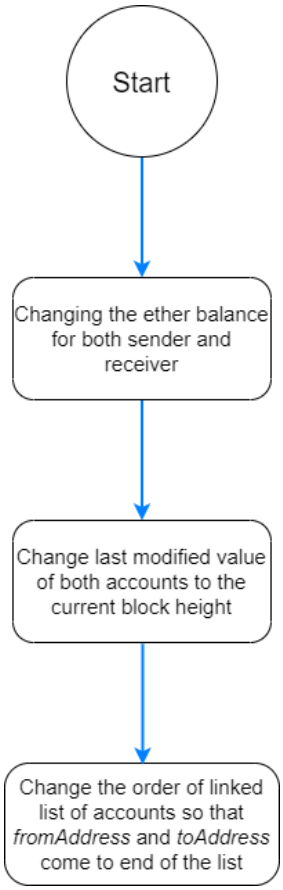
1. Execute all the transactions
2. Find and delete the accounts that should be deleted



Algorithm of executing a transaction

Every transaction has a **sender**, **receiver** and a **value**.

1. We execute the transaction like ethereum does, verify and modify both sender and receivers value
2. We get to the last modified value for both of the sender's and receiver's accounts from the Merkle Patricia Trie.
3. We change both of those last modified block values to the current block value.
4. Get the address of the account before and after the sender's and receiver's account and change the value of account before the sender's account to account after the sender's account and change the value of account before the receiver's account to account after the receiver's account.
5. Set the accounts in a specific order like first put the from account and then the sender's account
6. Set the value of *afterAccount* for the *lastAccount* to address the from account.
7. Set the value of *beforeAccount* for the *from* account to address of *lastAccount*.
8. Set the value of *afterAccount* for the *from* account to address of *toAccount*.
9. Set the value of *beforeAccount* for the *to* account to address of *fromAccount*.
10. Set the value of *lastAccount* to the address of *toAccount*.



Algorithm of finding and deleting the accounts that should be deleted

All of the accounts in the world state can stay there for a certain period of time without being modified and for better consensus throughout the distributed system, the time unit is block number. And the number of blocks that one account can stay in the world state without being deleted is called expiration date.

Modified here means an account to receive or send transactions. We check the unmodified accounts to see whether or not they reached the expiration date by the following formula.

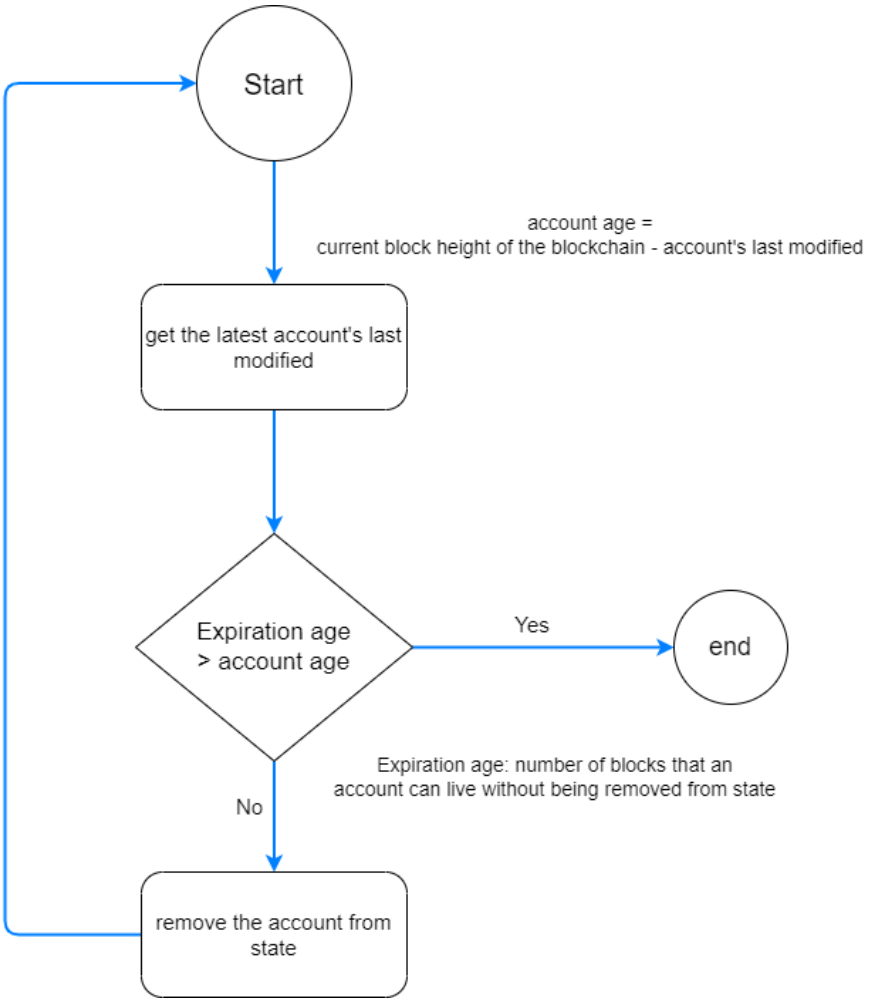
1. Go to the left of the red black tree as much as possible and find the node with the least value possible
2. Get the **account remaining time** by the following formula:

account remaining time = current block height of the blockchain - account's last modified

3. Check the following statement:

$$expiration\ date\ > account\ remaining\ time$$

4. If it was true, there is no account to remove because all the accounts have been modified less than the expiration date.
5. If it was not true, this account should be deleted so we delete it from both Merkle patricia trie and the red black tree
6. We do this algorithm again from the first step until we reach to an account with total value more than zero



Evaluation

Executing transactions and blocks and storing data in the storage are some of the most important operations for each full node in the ethereum network.

the current version of ethereum does each one these operations with the following complexity:

- Executing blocks: execution of each block that has m transactions in it, takes time complexity of O(m).
- Executing transactions: execution of transaction takes time complexity of O(log(n)) where n is the height of the Merkle patricia trie which won't exceed more than 40.
- Storing data in the storage: storing data for Merkle trie of the n accounts, has storage complexity of O(n).

Now that we know the complexity of each operations in the ethereum, we should note that if we want to propose a new solution for unused accounts, we shouldn't worsen the complexity of any of these algorithms because the ethereum network highly depend on them and big complexity like O(n) for executing transaction can fail the network and nodes won't operate like now because n is a very large number.

Evaluation of the new proposed solution is as follow:

- Executing transaction: executing transaction on the current ethereum network requires finding and modifying data on the Merkle patricia trie. In our new proposed way, some more steps are added to this computation and those steps are mentioned below:
    1. We get to the last modified value for both of the sender's and receiver's accounts from the Merkle Patricia Trie: O(1) for each account because we already had access to their state when we wanted to modify their value. 2 * O(1) in total
    2. We change both of those last modified block values to the current block value: O(1) for each account and 2 * O(1) in total
    3. Get the address of the account before and after the sender's and receiver's account and change the value of account before the sender's account to account after the sender's account and change the value of account before the receiver's account to account after the receiver's account: 2* O(1) for each account and 4 * O(1) in total
    4. Set the value of *afterAccount* for the *lastAccount* to address of from account: O(1)
    5. Set the value of *beforeAccount* for the *from* account to address of *lastAccount*: O(1)
    6. Set the value of *afterAccount* for the *from* account to address of *toAccount*: O(1)
    7. Set the value of *beforeAccount* for the *to* account to address of *fromAccount*: O(1)
    8. Set the value of *lastAccount* to the address of *toAccount*: O(1)

So to sum upm, it is: 2 * O(1) + 2 * O(1) + 4 * O(1) + O(1) + O(1) + O(1) + O(1) = O(1)

- Executing blocks: for execution of blocks, first ethereum executes all the transactions inside it and after that, nodes do the consensus algorithm to keep the network safe. In our approach, everything stays the same as ethereum but one part is included and that is deleting the accounts that haven;t been accessed in a certain number of blocks that passed.
  To get the account that has been modified the oldest we should start from the head of the trie and the first account has the oldest modification which is the least *lastModified* value.
  There should be a loop that iterates over the first of the linked list of the accounts and deletes the accounts that have reached their expiration date and stops the loop if it reaches to an account that hasn't reached its expiration date.

  The new algorithm only has the mentioned loop in addition to the original ethereum algorithm. If in the current state we have m accounts that need to be deleted, the loop iterates m times and in every iteration, it deletes an account which has time complexity of O(1) that n is the number of total accounts. So the time complexity is O(m * 1).
  M can be as big as n but this happens when the network is going to be abandoned forever. So m is approximately n/total number of blocks until the blockchain ends.

- Storing data in the storage: for storing n accounts in the Merkle patricia trie we have storage complexity of O(n) and is the same as the current version of ethereum.

Evaluation in comparison to state expiry:

N = number of accounts

M = number of accounts that should be deleted in each block ≃ n/ total blocks in the blockchain in all time(m is very low)

T = number of transactions in each block

	Our approach	State expiry
Executing transaction	O(log(n))	O(log(n))
Executing block	O(t) + O(m)	O(t)
Storage complexity	O(n)	O(n)
Additional complexity on each state(one year)	none	O(n)

Benefits

According to the leader of the go-ethereum project (the official node client for ethereum network), at the time of writing this article, the Ethereum world state is near 185 GB reference. By the data gathered from a big query, this world state holds for over 261615030 accounts.

Three out of every five of these accounts do not have any ether(158195507 accounts) and all of the EOA(externally owned account) addresses between these have useless data and are abandoned. But because of the current rules in the ethereum network, they are not deleted and their data should stay in the network for eternity.

So when 60% of ethereum world state consists of useless data and this useless data is growing rapidly. Our solution is to use a rent mechanism to ensure that everything that states in the network is useful and valuable for its owner. And when the owner doesn't need the data any more, they can stop paying for the rent and the data will be erased automatically.