Blockchain Interaction in Sending Ethereum

Author: Sahil Bansal

Department of Computer Science and Engineering,

Panipat Institute of Engineering and Technology, Haryana, India

Abstract-

If you are actively using internet and like to surf what's going around, you definitely come up with the word blockchain. So, I believe that blockchain has the power to revolutionize the internet. basically, what blockchain do is it makes the whole system completely decentralized which means there is no one in the middle you can directly send funds from one address to another address irrespective of banks. In this paper a model is proposed through which you can store your and send ethers over the Ethereum blockchain with the minimum gas consumed. We have referred certain papers to come up this project, Transaction fees optimization in the Ethereum blockchain and Blockchain challenges and opportunities.

Keywords:
Ethereum, Solidity, Smart-contracts, Blockchain, Ethers, State Transition System, Merkle Trees.

List of Abbreviations:
BC: Blockchain, Eth: Ethereum, EVM: Ethereum Virtual Machines, pow: proof of works, pos: proof of states, dApps: decentralized applications.

1. Introduction:

If you've read anything about technology or digital transformation, you've probably heard a lot about blockchain. After removing the hype, I believe that blockchain technology, like big data and the internet, will change many businesses. Looking at the benefits of blockchain, one of the most significant advantages is that you receive a history of all transactions/activity; it's essentially a database with history, if you will. Furthermore, there is no central community in charge of the blockchain; it is fully decentralised, thus there is no single point of access for hackers. So, in essence, "blockchain is a distributed ledger structured as a linked list of blocks." Each block has an order set in terms of transactions A typical approach secures the link from a block to its predecessor using cryptographic hashes."

So, taking this into account we have made a system through which we can send digital currency from one account to another account irrespective of any fraudulent it's completely decentralized and all the transactions that are done can be stored on Ethereum blockchain forever it means no one can change it or replace it *from the blockchain. So, we prepared this model using sol as our backend work and for front-end we use react and java script along with HTML and Css. Then we deploy our sol smart contract on the Ethereum virtual machine. As we know that Ethereum blockchain is very popular blockchain with over 3,00,000 nodes are connected with the blockchain which results in the slow transaction and high gas fees. so, we proposed a model in which you can store your transaction over a Ethereum blockchain with minimum cost and in less amount of time.

2. Literature review

The Blockchain showed up in the New York Times in 2013 alongside Beck Coin, which was additionally named "Expression of the year" by Oxford Dictionary. Bitcoin began with Bitcoin (AShared Electronic Cash System) published by Satoshi Nakamoto in October 2008.Since January 2009, it has been utilized as a web-based virtual cash that unreservedly exchanges between nodes by means of P2P without a middle person. Recently many researchers have published their research in this field with their own perspective or point of view. *Satpal Singh Kushwaha and Sandeep Joshi* has showed how attackers breach the security of smart contract they stated that even a small mistake can cause you a loss in millions. They also provide some techniques to keep in mind while writing smart contracts in order to prevent the security breach. *Li Duan and Kejia Zhang* has provide the preserved methods in detail for providing security for different types of attacks on blockchain. Michael Pacheco and Gustavo A. Oliva has presented their word on the blockchain transaction's whether the Ethereum transaction must be fast or slow the result of their research is to make the system which is time and cost effective. In 2020 *Seong-kyu Kim and Jun-ho-huh* has developed an auto chain platform which uses expert automatic algorithm used for house renting it's basically a Dapp which stores all the house record and buyers' details on the Ethereum blockchain. Rabia *Musheer Aziz and Mohammed Farhan Baluch* has designed a machine learning algorithm which uses machine learning approach to detect fraud on Ethereum Mainent

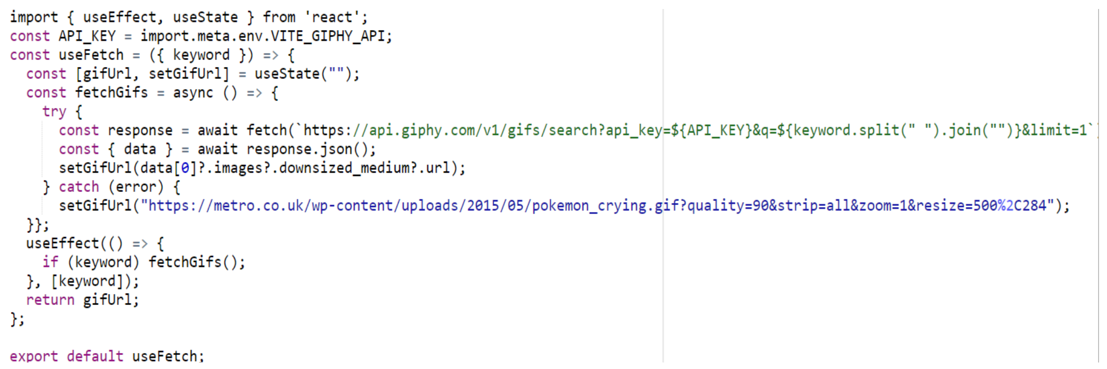
3. METHODOLOGY

3.1 React Hooks

React Hooks are new functionsalities introduced in React 16.8. Hooks enable developers to leverage state and other capabilities without having to write a class. Developers can reuse stateful code using Hooks without having to update the component system. Furthermore, Hooks help to improve rendering performance and provide a new approach to mix existing React ideas such as props, state, and so on.

The following are some common built-in Hooks in React:

- useState: acknowledges an underlying state worth and returns an ongoing stateful worth as well as the capability used to refresh the state.
- useEffect: acknowledges a capability that executes secondary effects that are not allowed in the capability part's fundamental body. The principal contention to useEffect is a capability, while the subsequent contention is discretionary. The capability is executed after each time the render is done; be that as it may, there are a couple of ways of changing the prerequisites for running the incidental effects to smooth out the interaction.
- useContexts: gets a React setting object as a contention. createContexts and returns the worth "prop" of the.Provider> part as the setting's ongoing setting esteem. useContexts empowers you to collaborate with the Contexts API and convey information without providing any prop.



```
import { useEffect, useState } from 'react';
const API_KEY = import.meta.env.VITE_GIPHY_API;
const useFetch = ({ keyword }) => {
  const [gifUrl, setGifUrl] = useState("");
  const fetchGifs = async () => {
    try {
      const response = await fetch(`https://api.giphy.com/v1/gifs/search?api_key=${API_KEY}&q=${keyword.split(" ").join("")}&limit=1`
      const { data } = await response.json();
      setGifUrl(data[0]?.images?.downsized_medium?.url);
    } catch (error) {
      setGifUrl("https://metro.co.uk/wp-content/uploads/2015/05/pokemon_crying.gif?quality=90&strip=all&zoom=1&resize=500%2C284");
  }};
  useEffect(() => {
    if (keyword) fetchGifs();
  }, [keyword]);
  return gifUrl;
};

export default useFetch;
```

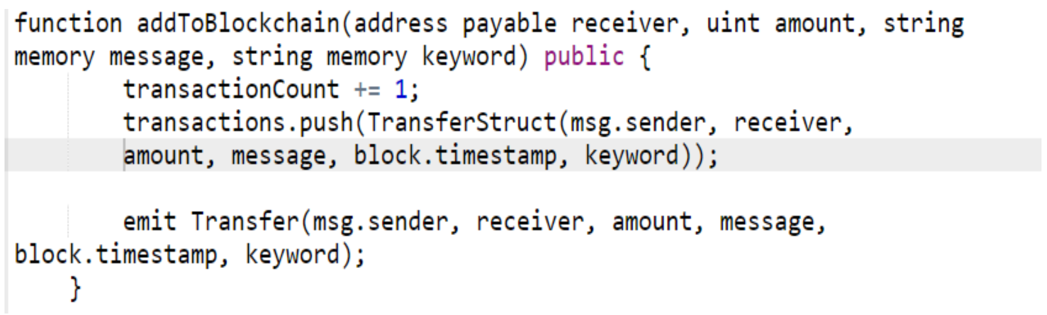
listing1: react hooks.

A key standard is that Hooks must be conjured from inside React utilitarian parts at the high level, as well as from custom Hooks.

3.2 Writing Sol smart contract

There are three primary types of Transactions, functionss that control what happens within a contract

- addsToBlockchain: This is the most useful functions in smart contract that is displayed in Listing 3.

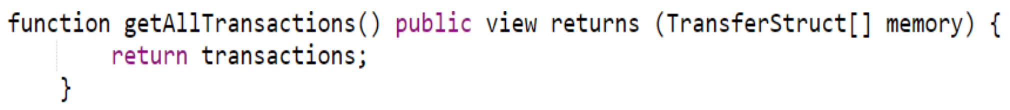


```
function addToBlockchain(address payable receiver, uint amount, string
memory message, string memory keyword) public {
        transactionCount += 1;
        transactions.push(TransferStruct(msg.sender, receiver,
        amount, message, block.timestamp, keyword));

        emit Transfer(msg.sender, receiver, amount, message,
block.timestamp, keyword);
    }
```

Listing 2. functions addsToBlockchain.

The functions is a public functions that returns nothing. When the addsToBlockchain functions is invoked from React, parameters such as receiver, amount, message, and keyword are sent to it.

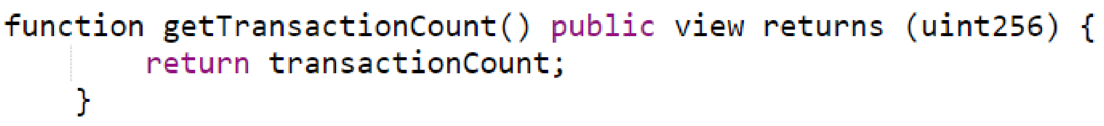
- getAllTransactions: Listing 3 shows the getting all transactions functions.



```
function getAllTransactions() public view returns (TransferStruct[] memory) {
        return transactions;
    }
```

Listing 3. getAllTransaction functions.
It is a public functions, which is also a read-only that returns list of transactions in the form of array which is stored in an array

- getTransactionCount: Getting the transaction count functions is shown in Listing 5.



```
function getTransactionCount() public view returns (uint256) {
        return transactionCount;
    }
```

Listing 4. getTransactionCount functions
It is a read-only public functions which returns the number of transactionCount.

3.3 Connecting smart contract to React

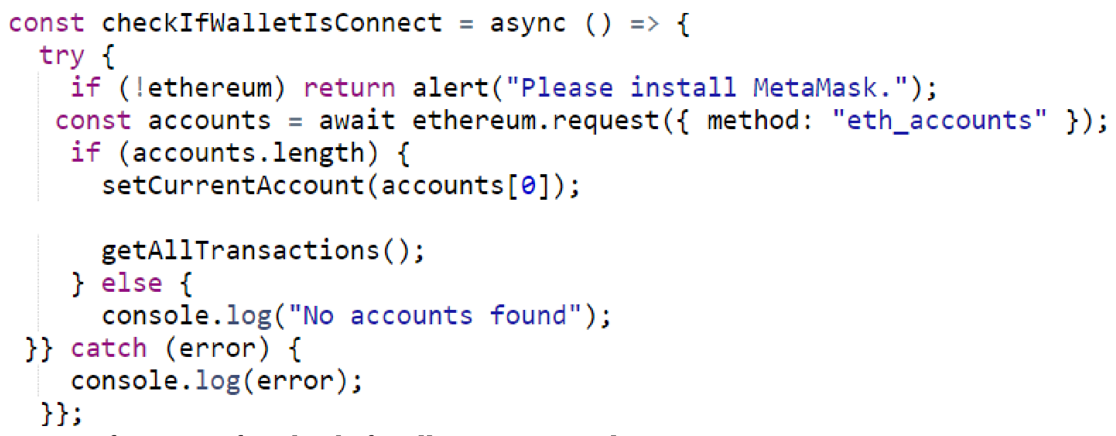
All interactions are saved in the TransactionContexts.js file. To connect to the blockchain, the React contexts API was utilised, which allows the logic to be written in one location rather than across all components.

To interact with a smart contract, the functions shown in Listing 8 was used to retrieve an instance of an Ethereum contract.

```
const getEthereumContract = () => {
    const provider = new ethers.providers.Web3Provider(ethereum);
    const signer = provider.getSigner();
    const transactionContract = new ethers.Contract(contractAddress,
contractABI, signer);

    return transactionContract;
}
```

Listing 5. Functions to fetch the contract.

- checkIfWalletIsConnected: at the start of the programme, the async method is called to see connected account in rendering the UI. Listing 9 displays the functions's code.

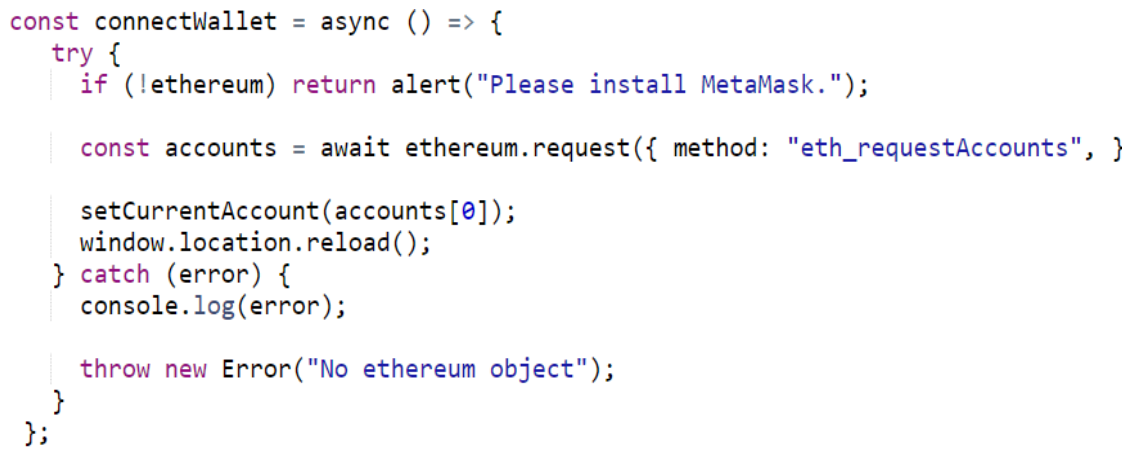


```
const checkIfWalletIsConnect = async () => {
  try {
    if (!ethereum) return alert("Please install MetaMask.");
   const accounts = await ethereum.request({ method: "eth_accounts" });
    if (accounts.length) {
      setCurrentAccount(accounts[0]);

      getAllTransactions();
    } else {
      console.log("No accounts found");
 }} catch (error) {
    console.log(error);
  }};
```

Listing 6. functions for checkIfWalletIsConnected

Method eth accounts in the functions that returns an list of arrays containing the address of current connected account. If there is a connected account, the first thing is to set the current account, and the functions also returns all transactions record in the getAllTransaction() functions.

- connectWallet: As demonstrated in Listing 10, the use of this functions is to connect with the MetaMask wallet.



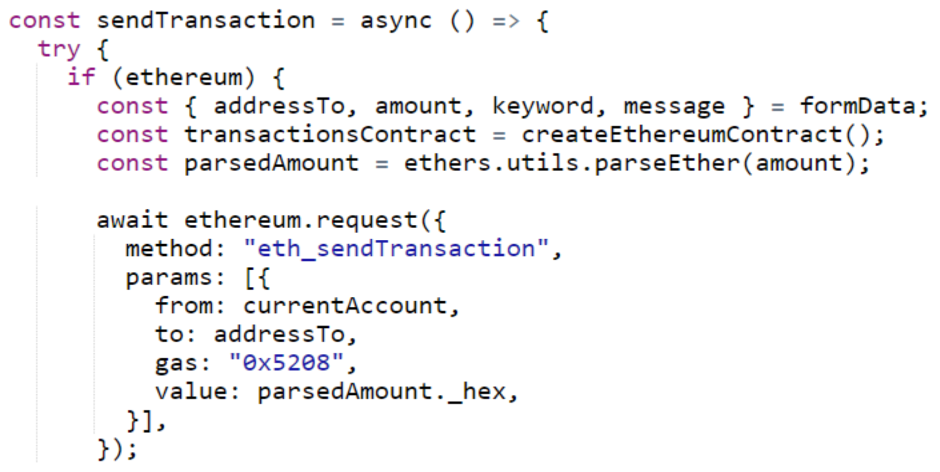
```
const connectWallet = async () => {
   try {
     if (!ethereum) return alert("Please install MetaMask.");

     const accounts = await ethereum.request({ method: "eth_requestAccounts", }

     setCurrentAccount(accounts[0]);
     window.location.reload();
   } catch (error) {
     console.log(error);

     throw new Error("No ethereum object");
   }
 };
```

Listing 7. connectWallet functions

When this function is invoked using the method eth request Accounts, MetaMask is launched in web browser. It gives an array which containing all the address that are connected to the programme, and the functions is used to set the current address to the first address.

- sendTransaction: This function is used to send the transaction on Ethereum blockchain. This function provide access of all the function that is used in sending ethers and storing data over Ethereum blockchain.

.

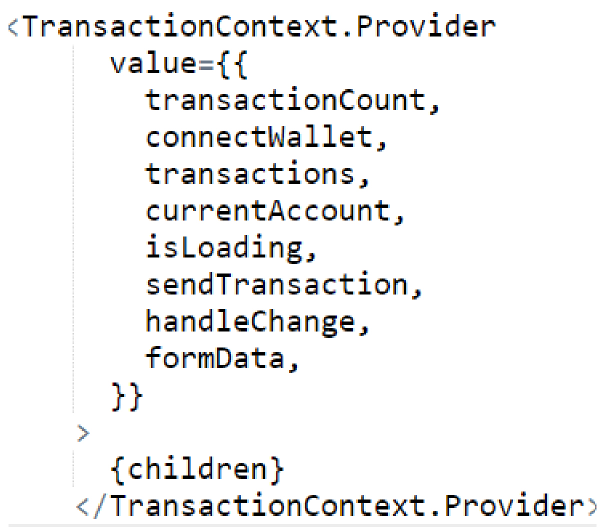


```
const sendTransaction = async () => {
  try {
    if (ethereum) {
      const { addressTo, amount, keyword, message } = formData;
      const transactionsContract = createEthereumContract();
      const parsedAmount = ethers.utils.parseEther(amount);

      await ethereum.request({
        method: "eth_sendTransaction",
        params: [{
          from: currentAccount,
          to: addressTo,
          gas: "0x5208",
          value: parsedAmount._hex,
        }],
      });
```

Listing 8. Sending Transaction action.

To obtain the transaction hash or id, the addsToBlockchain method was invoked. If the transaction is complete, the loading state was set to true. And once it finishes its work, the loading state is altered to false, and the transactions was successfully recorded.

In TransactionContexts.jsx, TransactionProvider the parental functions to three above capabilities, requirements to get {children} boundary from the props and returns TransactionContexts.Provider



```
<TransactionContext.Provider
      value={{
        transactionCount,
        connectWallet,
        transactions,
        currentAccount,
        isLoading,
        sendTransaction,
        handleChange,
        formData,
      }}
    >
      {children}
    </TransactionContext.Provider>
```

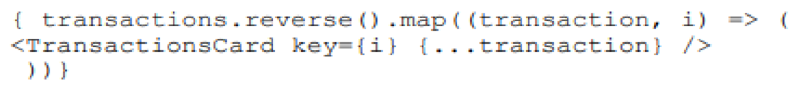
Listing 9. Component for TransactionContexts.Provider.

3.4 Recovering the lists of transaction

In TransactionContexts, two functions was created. The objective of jsx is to list the transactions.

- checkIfTransactionsExist: determines all the current transactions which is stored in system.
- getAllTransactions: it basically returns all the transaction that was done in a Sol smart contract using the functions.

use Effect Hooks is utilized by the boundaries check If Wallet Is Connected and check If Transactions Exist since they should be named toward the beginning of the app to accumulate information and rendering the UI.



```
{ transactions.reverse().map((transaction, i) => (
<TransactionsCard key={i} {...transaction} />
 ))}
```

Listing 10. How transaction was rendered in the card.

All current account transaction extracted after TransactionContexts. Each transactions is shown in a transaction Ethereum card with its characteristics during the loop.

3.5 Gas fees

On the Ethereum blockchain, "gas" is the fee or pricing value needed to complete a transaction or carry out a contract. The gas is used to distribute Ethereum virtual machine (EVM) resources so that decentralised applications like smart contracts can self-execute in a safe but decentralised manner. It is priced in small fractions of the cryptocurrency ether (ETH), also known as gwei and occasionally termed nanoeth.

The network's miners, who can refuse to execute a transaction if the gas price does not match their threshold, and users of the network looking for processing capacity establish the exact price of the gas through supply and demand.

3.6 how to reduce gas spent on Ethereum transaction

You can use the following patterns in your code to cut back on gas use.

- Short- circuiting:

We can employ the short-circuiting technique when an operation uses either || or &&. In order for the higher-cost action to be bypassed (short-circuited) if the first operation evaluates to true, the lower-cost operation must come first in this pattern.

```
// f(x) is low cost
// g(y) is expensive

// Ordering should go as follows
f(x) || g(y)
f(x) && g(y)
```

Listing11: short circuiting.

- Unnecessary libraries:
Libraries can contain a sizable amount of code that is unnecessary for your contract because they are frequently only imported for a select few usage. It is best to include functionality imported from a library into your contract if you can do so securely and successfully.

```
import './SafeMath.sol' as SafeMath;

contract SafeAddition {
  function safeAdd(uint a, uint b) public pure returns(uint) {
    return SafeMath.add(a, b);
  }
}
```

Listing 12. Example containing library

```
contract SafeAddition {
  function safeAdd(uint a, uint b) public pure returns(uint) {
    uint c = a + b;
    require(c >= a, "Addition overflow");
    return c;
  }
}
```

Listing 13. Example without having library

- Explicit functions visibility:

Explicit functions visibility frequently has advantages for gas optimization and smart contract security. To save gas each time the functions is called, specifically naming external functions compels the functions parameter storage location to be set as calldata.

- Proper data types:

Some data types in Sol smart contract are more expensive than others. Knowing the type that is most effective to employ is crucial. Here are some guidelines for data type usage.

- When possible, type uint should be substituted for type string.
- Storage of type uint256 requires less gas than uint8.
- Type bytes are preferable to byte[].
- Use the fewest number of bytes from bytes 1 to bytes 32 if the length of bytes can be restricted.
- It is less expensive to use type bytes32 than type string.

4. Conclusion

The project's purpose was to research decentralised technology blockchain and its main components, such as smart contracts, as well as to develop a cryptocurrency transfer application that allows users to send transactions in a quick and simple manner.

Consequently, a React application was created that connects to the blockchain and interacts with smart contracts using Solidity. Each transaction submitted by the user was accompanied with a Gif and was permanently recorded on the Ethereum network. The application served as a basic illustration of how blockchain may be used to provide transparency and trust while also reducing transaction time and cost. Overall, the project was a success because the author met the objectives outlined at the outset. this dissertation According to this study, incorporating the blockchain concept into a React application aided in efficiently assimilating the theories. The application, which is currently hosted on localhost, can be deployed to the domain for further development. Using this application, the user can send Ethereum via the blockchain using their own Meta mask account. Because bitcoin is frequently associated with a significant risk of scam or deception, it is advised that the application's security and dependability be improved.

Using this application, a user can send transaction from one account to another account across the globe easily quickly and without spending much on gas fees. In this project we uses many ways through which we can reduce the cost of sending Ethereum in simple words we write the cost efficient functions for reducing the gas fees which the fees used for sending Ethereum. We implement all those techniques inside our sol smart contract.

5. References

1. Tikhomirov, S. (2018). Ethereum: State of Knowledge and Research Perspectives. In: Imine, A., Fernandez, J., Marion, JY., Logrippo, L., Garcia-Alfaro, J. (eds) Foundations and Practice of Security. FPS 2017. Lecture Notes in Computer Science(), vol 10723. Springer, Cham. https://doi.org/10.1007/978-3-319-75650-9_14

2. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8

3. Andreev, O.: Proof that proof-of-work is the only solution to the Byzantine generals' problem (2014). http://nakamotoinstitute.org/mempool/proof-that-proof-of-work-is-the-only-solution-to-the-byzantine-generals-problem/

4. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 142–157. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_10

5. Bonneau, J., Miler, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Research perspectives and challenges for Bitcoin and cryptocurrencies. Cryptology ePrint Archive, Report 2015/261 (2015). http://eprint.iacr.org/2015/261

6. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. CoRR, abs/1703.06322 (2017)

7. Buchman, E.: Understanding the Ethereum trie (2014). https://easythereentropy.wordpress.com/2014/06/04/understanding-the-ethereum-trie/

8. Buterin, V.: Long-term gas cost changes for IO-heavy operations to mitigate transaction spam attacks (2016). https://github.com/ethereum/eips/issues/150

9. Castro, M., Liskov, B.: Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)

10. Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., Shi, W.: Decentralized execution of smart contracts: agent model perspective and its implications (2017). http://fc17.ifca.ai/wtsc/Decentralized%20Execution%20of%20Smart%20Contracts%20-%20Agent%20Model%20Perspective%20and%20Its%20Implications.pdf