Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 10, Issue. 4, April 2021, pg.28 – 34

Creating Payment Application and Cryptocurrency on the Ethereum Blockchain

[*]Mr. M. Thirunavukkarasu[1]
Harsha Surya Abhishek Kota[2]; Kommireddy Venkata Srinivasa Reddy[3]
[1]Assistant Prof, Department of Computer Science and Engineering, SCSVMV University, Kanchipuram
[2]B.E., Department of Computer Science and Engineering, SCSVMV University, Kanchipuram
[3]B.E., Department of Computer Science and Engineering, SCSVMV University, Kanchipuram
[1] mthiru@kanchiuniv.ac.in; [2] khsabhishek1335@gmail.com; [3] srinivasareddykommireddy111@gmail.com
DOI: 10.47760/ijcsmc.2021.v10i04.005

Abstract: Blockchain allows transactions to be performed much faster than is possible in traditional centralized systems. To provide decentralized payment application to the users. It can be very cost effective. Using Ethereum and Solidity programming language we are creating payment application. Which will include adding functionalities such as depositing ether, gaining tokens, and withdrawing ether in exchange of tokens and also a creator fee. By using Ethereum blockchain technology we can create our own ERC20 token project. ERC20 token is a cryptocurrency built on top of the Ethereum blockchain.
Keywords: Blockchain, Cryptocurrency, Ethereum, Smart contract, Ether, ERC

I. Introduction

Blockchain technology was born with the invention of Bitcoin—a new form of **peer-to-peer** (**P2P**) electronic cash—back in 2008. The Bitcoin white paper was released on October 31, 2008 and the first release of Bitcoin came on January 3, 2009 [1]. The blockchain space is still a toddler in terms of adoption and the tools available. It has some unique properties that did not exist before in any of the previous systems or software applications. The main property of blockchain is building up trust between two or different parties without requiring any intermediaries, which opens another period in programming. Ethereum is one of the implementations of blockchain. Ethereum is an open source, public, and distributed computing platform. On Ethereum, developers can deploy smart contracts written in the Solidity language and build a decentralized application—also called dApp, Đapp, Dapp, or Dapp [1]. Smart contracts are little programs where engineers can characterize the standards of the trust that they planned to code. One of the staggering properties of smart contracts is immutability—whenever they are conveyed on the blockchain, their code can't be changed. This immutable property makes it difficult to program smart contracts and anticipate blunders/bugs in advance. A blockchain is a timestamped arrangement of changeless transactions that is overseen by a group of PCs utilizing unique computer algorithms.

These immutable records are not claimed by any single entity. A blockchain is a decentralized P2P organization of nodes. Every node in a blockchain shares a similar duplicate of information, additionally called the digital ledger. Every node present in the organization utilizes a similar algorithm to arrive at an agreement. A blockchain, by configuration, is resistant to the change of information [2]. The ledger can record exchanges between two parties in an evident and lasting manner. At whatever point there is an adjustment in the ledger utilizing exchanges, changes are circulated to every one of the nodes, to check and refresh their own duplicate of the ledger. When an exchange is put away and checked by every node in the network, at that point it isn't practical to change the exchange without modifying every one of the ensuing and past blocks. That is the

reason blockchain exchanges are irreversible, as blockchain exchanges and their information are add as it. Every PC that takes part in this P2P network is known as a node. Every node keeps up the records of exchanges in various successive blocks. The P2P network is additionally utilized in torrents like BitTorrent; nonetheless, torrent networks are not as blockchains, as they are intended to shares records as it were [3]. Blockchain technology is also called Decentralized Record Innovation (DLT), as every node in the organization keeps a similar duplicate of the record.

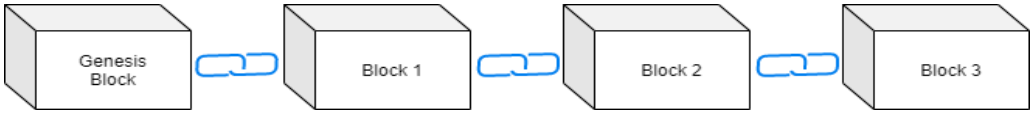

Fig 1 Chain of Connected Blocks

In the preceding diagram, each block is connected with a link (also known as a chain). The chain is usually recognized as the chain of all the blocks. The connection between two blocks is executed by having a record of the cryptographic hash of the past block in each block, with the goal that we can visit the chain backward sequential request [3].

As we probably are aware, we can share records and pictures over the web with somebody. At the point when we share those records or pictures with someone else, we really share a duplicate of the document [4]. When that duplicate is sent someone else, you and the other individual have the same duplicate of that report. However, things like cash, bonds, and shares should not be shared as duplicates, as we accomplish for records/pictures when we need to move them over the web. In the event that you attempt to send cash P2P, without utilizing any mediators, the two players wind up having similar duplicates of the cash. This is known as the twofold spending issue [2].

Blockchain solved this double-spending problem. Presently, for value transfers between two gatherings, neither of them needs to rely on a middleman (confided in party). The two of them can do safe exchanges straightforwardly. Blockchain's decentralized organization and agreement algorithm guarantees the security of exchanges and forestalls the double-spending issue.

1.2 Ethereum

Ethereum was proposed in 2013 by Vitalik Buterin, a cryptocurrency analyst and software engineer. The primary creation discharge dispatched in 2015. Ethereum is an execution of blockchain, and is an open source, public, and decentralized processing stage. Ethereum acquired ubiquity as a result of its smart contract's highlights. Engineers can program their smart contracts and execute them on the blockchain stage so the maximum capacity of the blockchain's various properties can be utilized. As talked about previously, Bitcoin utilizes blockchain technology for payments utilizing bitcoins as a cash. Bitcoin also upholds a scripting language, with which you can compose little scripts [5]. Bitcoin scripts have restricted functionalities and it is hard to compose complex scripts on it. Its scripting language is additionally not Turing-complete (it doesn't uphold loops). Bitcoin exchanges are slow; regularly, it gets takes 10 minutes or more. With Ethereum, composing complex smart contracts and using different properties of blockchain turns out to be simple. Any software engineer who knows Java, JavaScript, and other programming languages can code in the Solidity language. Solidity is a Turing-complete language. Exchanges on Ethereum are processed a lot quicker when contrasted with Bitcoin exchanges. Utilizing these possibilities, we can compose decentralized applications that would run on the Ethereum blockchain [5].

Ethereum architecture comprises of different elements that make its blockchain network. An Ethereum network has every one of the nodes associated with one another utilizing the P2P network protocol; every node keeps the most recent duplicate of the Ethereum blockchain record. A client can collaborate with the Ethereum network through the Ethereum client. The Ethereum client can be a work area/versatile/website page.

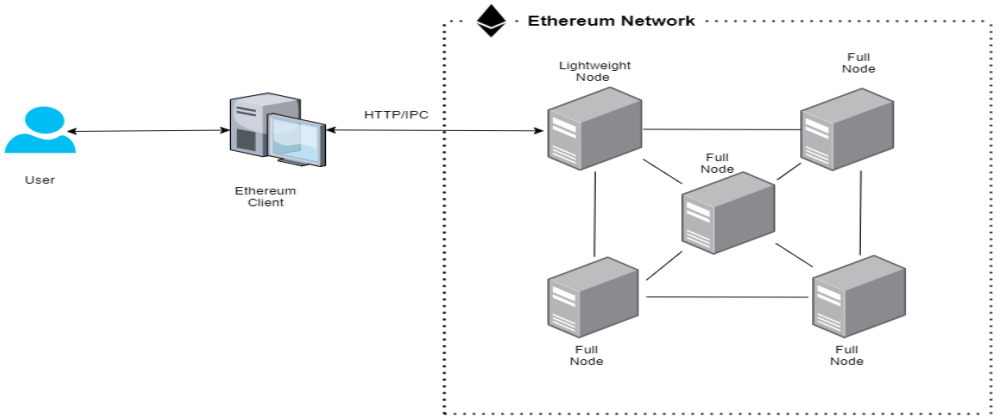

Fig 2 Ethereum High-Level Architecture

Ethereum has its own digital money called ether (image: ETH). Ether is a fungible coin, which implies that a coin can be partitioned into more modest units. For instance, 1 ETH can be partitioned into a limit of 18 decimal places; the littlest worth is called wei. Ether is a crypto fuel for the Ethereum network [6]. For any exchange you perform on the Ethereum network, a few gas is burned-through to execute that exchange, and the gas is to be paid in ether as it were. This exchange fee that you pay to deal with your exchange goes to the miners.

II. Proposed Method

To Provide decentralized payment application to the users. Blockchain allows transactions to be performed much faster than is possible in traditional centralized system. Using Ethereum and Solidity language we are creating payment applications. Which include functionalities like depositing ether, gaining tokens, and withdrawing ether in exchange of tokens and also a creator fee. By Ethereum blockchain technology we can create our own cryptocurrency using ERC20 token.

III. Developing Payment Decentralized Application with Ethereum

Dapp or a Decentralized application is an application that is run on a blockchain. Ethereum gives its clients some adaptability that permits them to make such applications. In this part, we will focus in making a Dapp and figure out how to carry out it in different parts of our lives.

In this section, we will figure out how to do the accompanying:

i) Create a project.
ii) Development and deploying our project.
iii) Changing our application with a superior payment technique.

Creating a project

This part will show us how to make new project. It will expect us to investigate the code and the Solidity syntax structure. Deploying the undertaking will assist us with seeing that it is so natural to change this task into a better payment application. To improve comprehension of the idea, we will take a look at few particulars on bug fixing and furthermore figure out how to troubleshoot smart contracts and some capacity applications.

To start, we need to make new folder. For the wellbeing of show, we name this as Payment application.

C:\Users\abhishek\payment application

The quick subsequent stage is run Truffle. The accompanying command is utilized for this:

C:\WINDOWS\system32>truffle

This shows the rundown of orders that can be utilized for different processes for Truffle. The Truffle init order permits us to instate another Truffle project.

All through the project, we will utilize a convenient Truffle unpack order to develop and set up a standard undertaking. For this, we will utilize the webpack box, which is finished by running the truffle unpack webpack.

C:\Windows\System32\my_project>truffle unbox webpack

All the containers are recorded on developing rundown, among which there are some Respond boxes and numerous others. Whenever this is done, we can begin making our project in a remix editor. We can start by building up the project structure. The unpacking will have made a few documents and folders for us. It will likewise install a webpack, which can be configured through the webpack.config.js record. This document permits us to design how we will limit our JSON, CSS, and JavaScript, and everything in the middle. We will likewise have a truffle.js document, which is utilized to characterize our networks. The truffle.js document will be defined with a development network that defaults to a localhost on port 7545. We will be also making an application organizer that contains our principal project. This folder contains a file of HTML and some CSS and JavaScript that imports our smart contracts. The contracts envelope contains **ConvertLib** which is a straightforward library for exhibition purposes that is imported within the primary document, **MetaCoin.sol**. The file extension unmistakably portrays that the language utilized here is Solidity, which one may consider a tongue of JavaScript that was created for Ethereum. We can likewise see a movements and test organizer. Each fills its own need, which we will talking about during the project.

Developing and deploying our project

We will currently be deploying the application that we had created. We will do as such by beginning a development blockchain, designing our deployment, deploying our smart contracts, and building the application. First and foremost, we will need to run our advancement blockchain. This should be possible by composing ganache-cli in the terminal window.

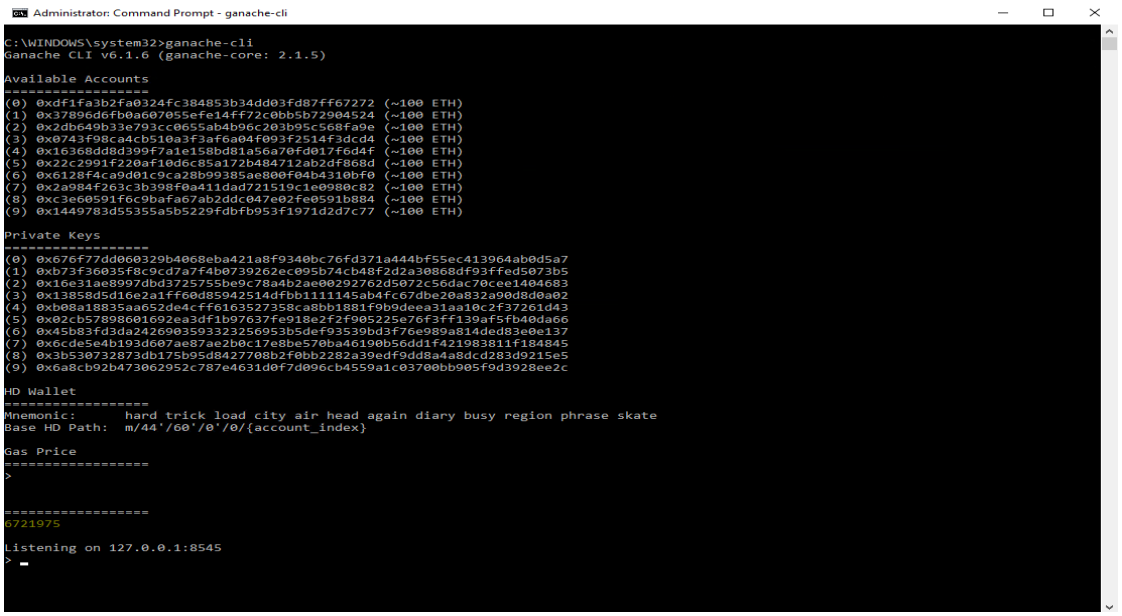

Fig 3 Ganache-cli Output

Ensure we write down or recall the localhost port number. It is of significance, as we will connecting it in the later phases of this project. Ganache-cli creates 10 accessible accounts and afterward the relating private keys. These private keys are utilized to scramble the exchanges that are being sent from every individual account. At the base, we will see a Mnemonic. These twelve words are vital. Continuously make sure to save these words since we will expect them to import our private keys and similar accounts into MetaMask. Pushing forward, we need to ensure that the project settings are relating to the hostname and the port on which our blockchain was made. To do this, we need to return to our editor, discover the truffle.js document that is arranged in the root folder, and change the port number here from 7545 to 8545 to coordinate with the port number that is facilitating our hosting blockchain. There will be no compelling reason to roll out any improvements to the host as that is only the localhost. At that point, continue to deploying our project on the terminal window, in the payment application folder.

C:\Windows\System32\my_project>truffle-cli compile

"In the OS that we are utilizing the Windows System we need to roll out certain improvements in the npm Package folder which will be situated at C:/clients/ [YOUR USERNAME]/appdata/wandering/npm. The way might be shift contingent on where every client has installed the npm package distinctive machine. We need to rename the truffle.cmd file to truffle-cli.cmd in the npm package."

The form output documents have now been written in the contracts folder (otherwise called the build folder). The quick following stage is migrated and deploy these smart contracts to our advancement blockchain. To do this, we utilize the accompanying command: C:\Windows\System32\my_project>truffle-cli migrate

The former command makes our agreements and manages exchanges for every one of them. Here, we can notice the production of the main agreement post—a few factors were refreshed and a few exchanges occurred. A similar cycle emphasizes to make different agreements each having their own factors and exchanges.

Returning to our editor of choice, we will discover these contracts in our migration folder. The way toward deploying starts with the migration smart contract and afterward proceeds onward to the ConverLib smart contract. MetaCoin is the last smart contract to be sent. An all the clearer arrangement can be accomplished on the off chance that we look the deploy_contracts.js file that lies in the migration folder on our editorial manager.

Changing our application with a better payment application

This part will focus in improving our code. This will incorporate adding functionalities, for example, depositing ether, acquiring tokens, and pulling out ether in return of tokens and furthermore a creator fee. We will deal with the very code that was utilized for the past area and keep on expanding on it. As we would prefer not to give away free tokens in return for stored ether. We start by setting a creator. To do this, we should characterize an address maker and a creatorFee. The collectedFees is the thing that one may call a pot. This is utilized to gather the creatorFees. The change rate is the rate that is utilized to multiply the quantity of tokens.

For instance, in the event that we have one ether, we will get five tokens in return. Allow us to consider that one has 0.1 ether; the transformation actually works with numbers at the backend. The currency_multiplier is utilized to use to estimations of our tokens and ether. The littlest category is a wei. We likewise need to introduce the maker as the message. sender when the smart contract is being made. The creator needs some exceptional elements of its own, like one for the withdrawal of fees. This expects us to make a modifier called onlyCreator. This adds the condition that if the message. sender is

the maker or assuming the transaction. origin comes from the maker; just will the code be executed. This is finished by adding an underscore. The subsequent stage is making a function called collectFees. For the good of the code, we'll disclose it, however to such an extent that lone the creator can call this function.

Presently we will move the funds to the creator and set the collectedFees to 0. Presently, what we would like to add is a few functions so that at whatever point someone sends a coin, we need a piece of that coin to have the option to go to the creator, and for this we will make the deposit and the withdraw functions. The deposit function will be a payable function. A payable function is utilized at whatever point we need to get ether. On the off chance that we do neglect to utilize it, it will simply show an error. The conversion rate utilized has effectively been set in the contract function. Keep in mind, our tokens will work equivalent to one ether; that implies one symbolic will be multiplied by 10 to the power 18. So, when we store any tokens, it will be increased by the CURRENCY_MULTIPLIER.

We will at that point proceed onward to the withdraw function. There must be some measure of alert while carrying out this, since we first need to increase with a money multiplier that has been set in the contract function. At that point we will continue to deduct the balance of the agent of the exchange with the sum that should be removed. The sum will be isolated by the conversion rate.

3.1 Developing and Deploying our own cryptocurrency

As a matter of first importance, we need some information to go into the editor. In the MetaCoin.sol, we need to characterize a name, an image, the quantity of decimal spots, and INITIAL_SUPPLY. We will likewise allot our INITIAL_SUPPLY to the total Supply in the constructor. We will likewise give the maker of the token the INITIAL_SUPPLY. We realize that we need totalSupply since, in such a case that we go to StandardToken, found under node modules | zeppelin-solidity | contracts | token | ERC20 | StandardToken, at that point we can see that this imports ERC20 and BasicToken. Presently, in the event that we go to the ERC20 folder, we can see that this is an interface, however BasicToken.sol carries out this interface. So, we should go to the BasicToken.sol folder. It will have totalSupply, balances. It has an Exchange work that utilizes SafeMath from the SafeMath.Solidity record. This a little Solidity file that will ensure that at whatever point we play out a numerical activity, our information and our yield information will be right, and that it will not ruin anything, on the grounds that once information is inside the blockchain, it stays in the blockchain. Along these lines, subsequent to characterizing this, we can simply feel free to deploy it. We will Go through the accompanying steps to deploy the ERC20 tokens, our first cryptocurrency.

i) First, we ensure that Ganache is running.

ii) When Ganache is running, copy the mnemonic and paste it in the metamask.io Reestablish Vault. We have done this previously.

iii) Press alright. we should see nothing, at first. Check for the network. In the event that we are on our main network, we should associate with http://localhost:7545—this is the port that Ganache is tuning in on. we can do this by tapping the drop-down arrow next to the principal organization and choosing Custom RPC. Type in the localhost URL referenced beforehand and click Save. We should see that MetaMask has in a flash connected. We can examine our account; we ought to have some ether.

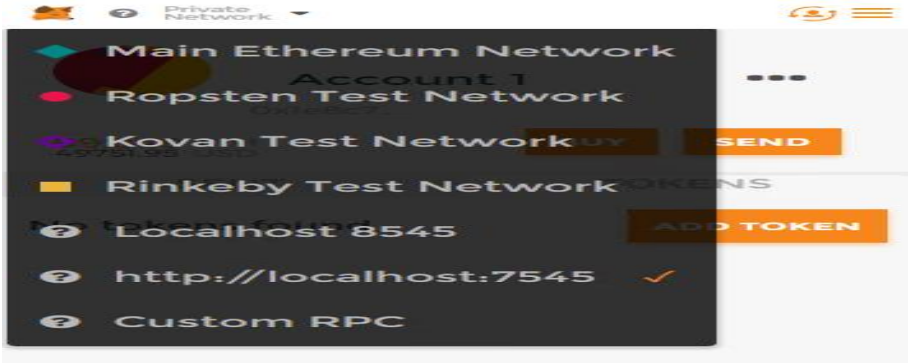

Fig 4 MetaMask Outlook

iv) Go to the migrations folder and make ensure that it has var MetaCoin.

v) We should go to MetaCoin.sol. Note that the INITIAL_SUPPLY will be separated by the measure of decimals indicated.

vi) Change to the terminal window. Type the truffle-cli move order. We should to have the option to see the exchanges that have come up effectively. Copy the contract token.


```
E:\chp3>truffle-cli migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
  ... 0xb60980748020609c66540dbc20e849072b6ae2424c5e3bad783309ab9903215a
  Migrations: 0x5dfba046aeb3ed90848030d2490cc5d291106c48
Saving successful migration to network...
  ... 0x8bc5118ca888e23515d7016dd5c28b82b4bf9811638974aba0423c44d28c760c
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying MetaCoin...
  ... 0xc3d1f13316587a5d112081c2bd73411f75607315aae526bcf194908e96f64f66
  MetaCoin: 0xb53fef326f9ef4b72dce78a88bc2cb9e0b14825f
Saving successful migration to network...
  ... 0x94d530561588fdbd5476a9a3a720493c7a530ac80f3725c9808f6bf9149a6b9d
Saving artifacts...
```

Fig 5 Cryptocurrency/ERC 20 migration

vii) In the wake of copying this, you can go to MetaMask. Access the upper left symbol, click on Add Token, and paste the token address in the token address area. It will naturally recover the symbolic image and the predetermined number of decimals. Snap on Add. You will see your first tokens, preceded by your symbol.

viii) Presently you can make an account by tapping the client symbol above. You will currently have a subsequent account. Escape the symbolic tab and you will see that you have 0 tokens on the subsequent account.

ix) You can simply utilize the code you previously needed to send these tokens over. The difference between the MetaCoin from the last model and this one is that now the tokens are really keeping a genuine norm, specifically, the ERC20 standard. It expresses that you need a name, a symbolic image, decimals, and an INITIAL_SUPPLY.

x) To transfer tokens run, the npm run dev commands.

xi) Open the internet browser and explore to localhost:8080:

xii) Go to the editor. The JavaScript function will in any case attempt to call the getBalance function, yet this has been changed. You can see this in the event that you go to the BasicToken Solidity document; you will see that you have balanceOf. So, we should go to the app | JavaScripts | app.js and afterward on to the refreshBalance function. Here, we change the meta.getBalance to meta.balanceOf. Likewise, go to the sendCoin function and change meta.sendCoin to meta.transfer. Ensure that you have a similar sort of method signature in BasicToken.sol under the ERC20 folder:

xiii) Presently go to the browser on the localhost:8080 page. Change to the main account. There it is. You will not see the decimals set up, yet that is only a frontend stuff.

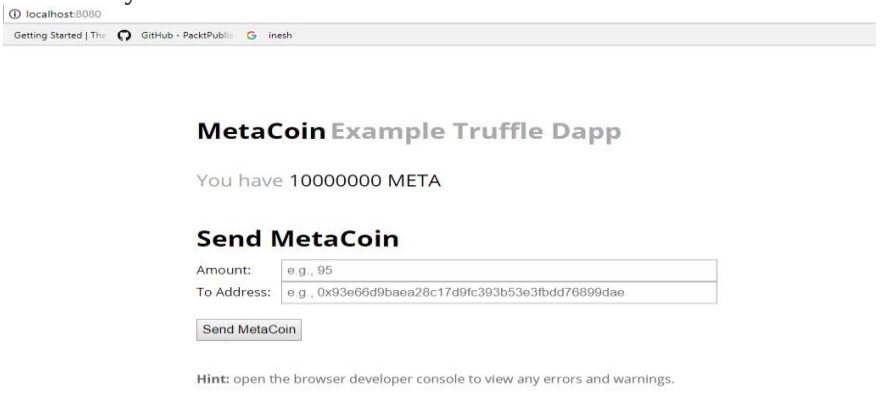

Fig 6 Final Browser view

IV. Conclusion

We investigated how to make an Ethereum based payment application. We additionally deployed and tested it. We at that point dug into the Solidity language syntax which isn't just huge but on the other hand is broad. JavaScript codes were additionally investigated. We had fix bugs utilizing our own frameworks just as remotely. furthermore, we likewise figured out how to make our own cryptocurrency ERC20 tokens, and afterward send and test them. We likewise figured out how to utilize these tokens and execute rationale when somebody gets these tokens. But it also shows that the performance of the proposed models is impacted when the network is too sparse, as the major improvement is caused by the network correlation features. At last, it shows that the performance of the proposed models is affected with regards to the security, as the significant improvement is brought about by the blockchain innovation features.

References

[1] Oriel Rivers et al., *A Study on Cyber Attacks and Vulnerabilities in Mobile Payment Applications,* Journal of The Colloquium for Information Systems Security Education, Volume 7, No. 1, (2020).
[2] Florian Ginez, *CFA Bitcoin Versus Traditional Payment Systems: Is One more Effective than the Other, WisdomTree, ISSN 2397-284X, (2019).*
[3] Dongfang Zhao et al., *Cross-Blockchain Transactions,* WisdomTree, ISSN 2397-284X, (2020).
[4] Qiheng Zhou et al., *Solutions to Scalability of Blockchain: A Survey,* IEEE, ISSN 0018-9219, (2020).
[5] Sudeep Tanwar et al., *Blockchain for Industry 4.0: A Comprehensive Review,* IEEE, ISSN 0018-9219, (2020).
[6] VictorChang et al., *How Blockchain can impact financial services – The overview, challenges and recommendations from expert interviewees,* (2020).

