How to Cite:
Shah, J. R., & Sharma, D. H. (2022). The performance evaluation of permissionless distributed ledger technology ethereum: A case study blood trace application. *International Journal of Health Sciences*, *6*(S4), 8211–8236.
https://doi.org/10.53730/ijhs.v6nS4.10601

The performance evaluation of permissionless distributed ledger technology ethereum: A case study blood trace application

Jignasha R Shah
Resarch scholar, K J Somaiya College of Engineering
Corresponding author email: jignasha.shah@somaiya.edu

Deepak H. Sharma
Professor, K J Somaiya College of Engineering
Email: deepaksharma@somaiya.edu

***Abstract*---**Permissionless or public Distributed Ledger Technologies (DLTs) provide full decentralization and transparency. The performance evaluation of permissionless Distributed Ledger Technologies is difficult due to their dynamic nature. This paper focuses on the performance evaluation of permissionless Ethereum DLT. Public test networks highly resemble the main network and capture the dynamic nature of permissionless DLTs. The paper proposes the BloodTrace application to trace the donated blood from the donor to the patient. The performance is evaluated by deploying the BloodTrace smart contract on the public test network Ropsten. The data collection for the calculation of the performance parameters can be performed in two ways: i) RPC based method ii) Log-based method. RPC based method involves polling the blockchain for each transaction data. The log-based method involves the collection of logs and fewer RPC calls to the Blockchain. The probabilistic nature of transaction confirmation is also taken into consideration while calculating the Latency (Finality) by analysing the effect of block confirmations on the latency. The evaluation results are then compared with the results obtained by the Hyperledger Caliper Benchmarking tool as it is using RPC based approach for calculation of performance parameters.

***Keywords*---**ethereum, ropsten, test networks, performance benchmarks.

Introduction

Empirical Performance Analytical methods for evaluation of DLTs constitute Benchmarking, Simulation, Monitoring and Experimental Analysis [1]. Benchmarking uses well-documented workloads and a standard environment for performance evaluation. Benchmarking facilitates the comparison of permissioned DLTs as well as the comparison of the different versions of the same DLT platforms. The experimental analysis involves self-designed workloads [1]. Setting up a private network to measure the performance of a permissionless DLT will not yield the same results when implemented in the real world. Performance monitoring with the real-time workload is a better way to assess the performance of a permissionless DLT. Benchmarking, monitoring and experiment analysis necessitates setting up of the testing network. Simulators are great when there is no time or resources to set up the testing network. Scenarios are created and scenario-based testing is performed in a simulated environment. Simulations facilitate the fine-tuning of platform-specific parameters.

Dabbagh et al. [2] presented a generalized survey of empirical studies for evaluating the performance of permissioned Blockchains. This survey categorizes the existing research based on parameters such as Blockchain Platforms, latency, throughput, consensus algorithm, scalability, fault tolerance, evaluation framework, performance metrics, and network type. It also gives a quick rundown of consensus algorithms and DLT platforms. The platforms under consideration are primarily Hyperledger Fabric and Ethereum, each with its own set of versions, implementations, and consensus algorithms.

In most of the existing work, permissionless DLT Ethereum is set up as a private network and performance is evaluated in a controlled environment. The joining and leaving of nodes, different node configurations and geographically separated nodes are not taken into the account in the evaluation of Ethereum in a controlled environment. The test networks provided by various DLT platforms resemble the original DLT platform protocol. They make it simple to test the application before releasing it to the Mainnet. The paper measures the performance of Ethereum DLT using a log-based data collection approach and Ropsten test network with the BloodTrace application.

Most of the existing work calculate the latency as the difference between the time of the transaction submission and the time at which it was included in the block. The consensus is probabilistic in a permissionless DLT like Ethereum that uses the Proof of Work (PoW) algorithm. When 12 or more blocks attach to the block containing the transaction in Ethereum, the transaction is considered irreversible[1]. This should be taken into account when calculating the transaction's latency or finality.

[1] https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/

The contribution of the paper is as follows:

i) The performance Evaluation of Ethereum focuses on capturing the dynamic nature of the permissionless DLT.
ii) The calculation of latency (finality) considers the probabilistic nature of consensus by analysing the effect of block confirmations on latency.
iii) The results obtained with the log-based data collection method are compared to those obtained with Hyperledger Caliper benchmarking tool.

The paper is organized as follows: Section 2 discusses related work in performance measurement of Ethereum DLT, Section 3 presents the proposal and implementation of BloodTrace application, Section 4 discusses the performance evaluation of Ethereum in detail with the experimentation and results, and Section 5 gives conclusion and future work.

Literature Review

This literature review discusses performance evaluation of Ethereum DLT under benchmarking, monitoring, experimental analysis and simulation as shown in Fig.1. Benchmarking uses well-documented workloads and allows for a more accurate comparison of DLT platforms. The experimental analysis involves synthetic workload and a controlled environment. Monitoring requires live monitoring of a DLT platform with a synthetic workload. More suitable for permissionless DLTs. The simulation does not need any test network setup. The effect of any parameter can be examined. For example, we can vary the size of the block to see its effect on throughput and latency. Simulation provides less accurate results compared to other performance evaluation methods.

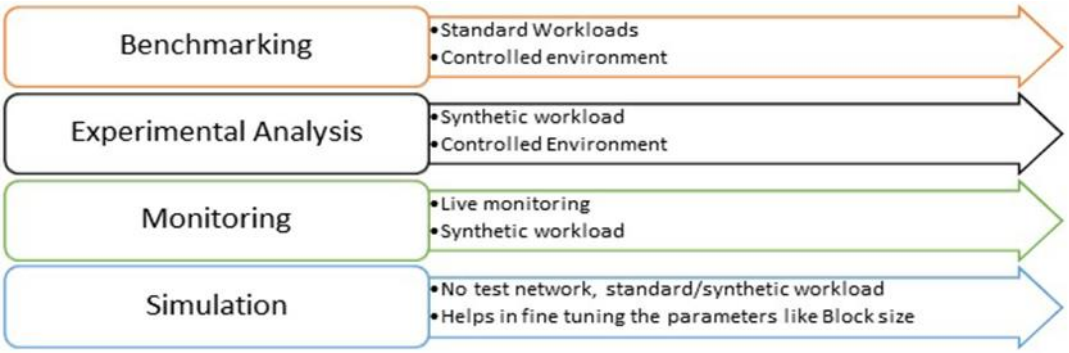

Fig.1. Classification of Empirical Analysis

Benchmarking

The BLOCKBENCH [3] authors have classified the DLT platform in the application layer, execution layer, data model layer and consensus layer. They have used some well-documented workloads that are defined layer-wise as shown in Fig. 2. Some of the workloads were used for distributed databases. A good Benchmark must provide well-documented workloads, reservation of resources, deployment of processes and services for the DLT, generation of workloads, evaluation of

metrics, generation of reports, reproducibility of experiments and definition of parameters/ process to measure them.

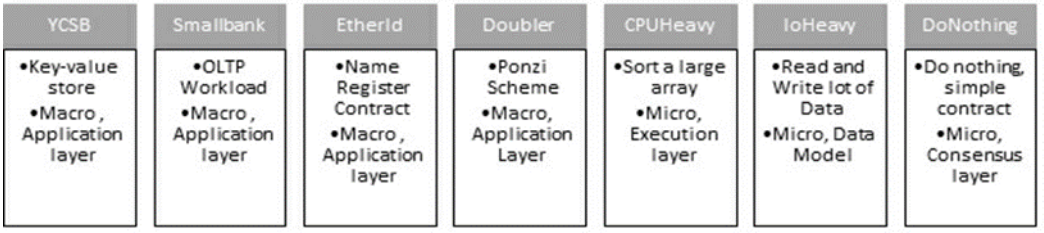

Fig. 2. Benchmarking Workloads

Experiment reproducibility allows researchers to share their findings with their peers. Developers should be able to run the same experiments on different platforms given the ability to generate a workload from history. The benchmarks are compared in Table 1. BLOCKBENCH uses the standard workloads and does not provide automatic deployment and resource reservation abstraction. BLOCKBENCH and Hyperledger Caliper [4] both do not have the capability of reproducing experiments. This will limit their performance evaluation.

Table 1. Comparison of Benchmarks

Framework	1	2	3	4	5	6	7	8
BLOCK BENCH	Throughput, latency, scalability, fault tolerance	Ethereum, Hyperledger Fabric, Parity and Quorum	48	No	No	No	No	Academic, First benchmark for private DLTs
Hyperledger Caliper	Throughput latency, resource consumption, success rate	Hyperledger Besu, Ethereum, Hyperledger Fabric and FISCO BCOS	scalable	Yes	No	No	Yes/No	Commercial, standard documentation and process, easily scalable
BCTMark	Power usage, CPU usage, Cost of smart contract	Ethereum Ethash (PoW), Ethereum Clique (PoA), and Hyperledger Fabric	16 on Grid 5000 3+ on Raspberry Pi	Yes	Yes	Yes	No	Academic, no standardized documentation, the combination of different frameworks to provide ease of use, Grid 5000 and Raspberry Pi
DLPS	Throughput, Latency	Ethereum (Geth and	64 AWS	Yes	Yes	Yes	Yes/Ye	Academic, no standardized

	Parity), Indy, Sawtooth, and Quorum	ec2 M5 nodes	s	documentation, Amazon AWS EC2 setup code

1—Metrics, 2---Platforms, 3---Number of nodes, 4---Deployment, 5--- Reproducibility, 6--- Resource Reservation, 7--- Definition of parameters /process to measure them, 8--- comments

Both BCTMark (Blockchain Technology benchMarking) [5] and DLPS (Distributed Ledger performance Scan) [6] provide the capability of reproducing the same experiments. All Benchmarks use an adapter-based approach for integrating new DLTs and new workloads. Hyperledger Caliper is a commercial, standardized framework for performance benchmarking. BLOCKBENCH, BCTMark and DLPS are academic frameworks. Benchmarking frameworks are useful for enterprise or permissioned DLTs, while monitoring and experimental analysis are more useful for public DLTs. The dynamic nature of Public DLTs makes them very difficult to emulate.

Experimental Analysis

In Experimental Analysis, self-designed workloads are used. The performance evaluation of Quorum is presented in [7]. Quorum is an Ethereum-based consortium ledger platform. Latency and throughput are measured using the Hyperledger Caliper tool and the plugin. Smart contracts generate read workload, write workload, and mix workload. The impact of using the RAFT and IBFT consensus algorithms was also investigated. Read workloads have the lowest latencies, whereas write workload latencies are heavily influenced by block time.

Performance analysis of private blockchains is presented in [8]. The performance analysis is done by varying the number of transactions for Hyperledger Fabric and Ethereum. The parameters are execution time, latency and throughput. JSON-RPC mechanism is used for retrieving the data from the blockchain. Hyperledger has more throughput and less latency compared to Ethereum, but Ethereum can handle more concurrent transactions than Hyperledger.

Dinh et al. [9] used BLOCKBENCH to conduct a comparative performance analysis on three mainstream private blockchains, namely Ethereum (Geth v1.4.18), Parity (v1.6.0), and HLF (v0.6.0-preview). Their findings can be summarized as follows: 1) HLF consistently outperforms Ethereum and Parity across all macro (e.g., throughput and latency) and micro (e.g., IOHeavy) benchmarks, but it cannot scale to more than 16 nodes; 2) The consensus protocols have been identified as major bottlenecks for HLF and Ethereum, while transaction signing is a bottleneck for Parity. The authors then compared the performance of two different versions of HLF v0.6.0 and v1.0.0 [9]. Rouhani and Deters [10] investigated Ethereum's performance on a private blockchain by examining two of the most popular Ethereum clients: PoW-based Geth and PoA-based Parity. According to the results, Parity is faster than Geth in terms of transaction processing under various workloads. Bez et al. [11] performed a preliminary quantitative analysis of Ethereum's scalability. The transaction

throughput was measured using synthetic benchmarks in an extensible test environment. They examined the effect of gas limit, difficulty and timestamp on throughput. The results show that a blockchain platform can rarely achieve decentralization, scalability, and security at the same time. Benahmed et al. [12] compared the performance of Hyperledger Sawtooth, EOS, and Ethereum. The authors defined three types of workloads, namely CPUHeavy, KVStore (Key-Value Store), and SmallBank. Using these workloads they tested CPU consumption, memory consumption, load scalability and network scalability in distributed ledgers. The results show that EOS outperforms the other two in terms of both resource consumption and speed, but with a bottleneck of centralization.

Wazen et al. [13] proposed an orchestration tool for node reservation, deployment, and blockchain configuration on the Grid '5000 platform. They have implemented a blockchain-based KYC application and used it for testing the tool. This tool allows the researchers to fine-tune the different parameters and analyze their effect on blockchain performance. They varied the number of mining nodes and analyzed the impact on throughput and latency. Edurado et al. [14] presented the analysis of different consensus algorithms using geographically distributed VPCs from Amazon AWS EC2 virtual machines. They emphasized the significance of latency in the choice of a consensus algorithm. Dynamic performance evaluation of DLTs using Data Envelopment Analysis (DEA) is presented in [15]. DEA is the non-parametric method to evaluate peer units with multiple inputs and outputs. With the help of DEA, the inputs will be converted to 1 and then compared. The DLTs are analyzed based on technical indicators, market indicators and popularity indicators. DEA and the global Malmquist Index are integrated to study the dynamic performance of DLTs. The performance of 31 public DLTs was observed and evaluated for 24 months [15]. Leal et al. [16] presented the performance evaluation of Ethereum by setting up the private network. The performance of the blockchain platform was empirically evaluated considering the following scenarios: (1) fixed and dynamic block gas limit configuration; (2) two consensus algorithms, namely PoW and PoA; (3) different PoA block periods for simple transactions (low gas limit) and complex transactions (high gas limit).

Monitoring

A performance monitoring framework with the log-based method is suggested in [17]. This framework is having low overhead and increased scalability. It does the real-time monitoring of the Blockchain platform. It uses overall performance parameters for comparative analysis DLTs and detailed parameters to measure the performance of the blockchain platform. A thousand open-source contracts were used to evaluate the performance of the Blockchain platforms Ethereum, Parity, Hyperledger and CITA (Cryptape Inter-enterprise Trust Automation). Overall parameters are Transaction per second (TPS), Average response Delay (ARD), Transactions per CPU (TCPU), Transactions per Memory Unit (TPMU), Transactions per Network Data (TPND), Transactions per Disk I/O (TPDIO), and detailed parameters are Peer Discovery Rate (PDR), RPC Response Rate (RRR), Transaction Propagation Rate (TPR), Contract Execution Time (CET) and Consensus cost Time (CCT). The performance of a permissionless DLT was measured by connecting one terminal to the permissionless Blockchain. The private LAN was used to conduct the experiments. Monitoring is the least

expensive and more appropriate method to measure the performance of permissionless DLTs.

Simulation

Simulators do not require the availability of SUT. They also assist in investigating the effects of fine-tuning some of the parameters. Alharby and van Moorsel [18] proposed and implemented BlockSim, a framework for developing discrete-event dynamic system models for PoW-based blockchain systems. This framework is composed of three layers: the incentive layer, the connector layer and the system layer. The authors investigated the performance of block creation under the PoW consensus algorithm using the tool. This simulator aided in comprehending the nuances of the PoW block generation process. The simulation results were compared to those of real-world systems such as Bitcoin and Ethereum.

Pandey et al. [19] proposed and created BlockSIM, a comprehensive open-source simulation tool for simulating private blockchain systems. This tool is intended to run scenarios to evaluate system stability and transaction throughput (TPS) for private blockchain networks. The comparison results between BlockSIM and a real-world Ethereum private network using PoA consensus shows the effectiveness of BlockSIM. Faria and Correia [20] presented BlockSim, an extensible discrete-event simulator, to evaluate various blockchain implementations. They used the simulator for Bitcoin and Ethereum. Other Blockchain platforms can be added easily. They discovered that increasing the block size had only a minor effect on propagation delay. The encrypted data had a greater influence on propagation delay.

Discussion

Benchmarking, Experimental analysis and monitoring for permissioned DLTs involve private network set up through cloud or physical machines. Not all benchmarking frameworks use standard workloads and their evaluation environments are also different. Some of the benchmarks have hardcoded configurations which makes them difficult to use. Table 2 shows the deployment models/evaluation environments for experimental analysis and monitoring. The number of nodes and node configurations are part of the evaluation environment. The evaluation framework indicates the use of the benchmark frameworks described in section 2.1.

Table 2. Deployment models for Experimental Analysis and Monitoring

Reference	Platforms	Parameters	Evaluation environment	Evaluation Framework
Baliga et al. [7]	Quorum	Throughput, Latency	3 nodes (RAFT) 4 nodes (IBFT) 8 vCPUs per machine, 4 cores, 3.6 GHz, 16 GB RAM	Caliper
Pongnumkul et al. [8]	Fabric, Ethereum	Throughput, Latency	1 node, AWS EC2, Intel E5-1650, 8 core CPU, 15 GB RAM, 128GB SSD	Synthetic

Dinh et al. [9]	Fabric, Ethereum	Throughput, Latency	8 nodes, E5-1650, 3.5 GHz, 32 GB RAM, 2 TB HD	BLOCKBENCH
Rouhani and Deters [10]	Geth and Parity	Execution Time	1 node, core i7-6700 CPU, 24 GB RAM	Synthetic
Bez et al. [11]	Ethreum	Throughput	1/2/4/8 miners with 16 clients, START1-s Server, 2 cores, 2 GB RAM, 50 GB SSD	Synthetic
Benahmed et al. [12]	Hyperledger Sawtooth, EOS, Ethreum (Geth)	Resource consumption and scalability	6 nodes, Xeon X7350 CPU 16 cores, 2.93 GHz, 64 GB RAM	Synthetic
Wazen et al. [13]	Ethereum	Throughput, Latency	100 clients, miners varying from 13 to 25 on Grid '5000 testbed	Synthetic
Edurado et al. [14]	SpeedyChain	Latencies for adding blocks and transactions at each stage	2 EC2 VMs on AWS, one running gateway and other simulating 15 devices	Synthetic
Zheng et al. [17]	Ethereum, Parity, Hyperledger Fabric and CITA	Overall and Detailed parameters	10 nodes in a LAN, i7-4790 3.60GHz CPU, 8GB RAM 1 node to the public network	Synthetic

It can be seen that Ethereum's performance is evaluated by establishing private networks. Except for concurrent transactions, Hyperledger Fabric outperforms Ethereum in every way. The PoW consensus algorithm provides decentralization and security but lacks in performance. Hyperledger Fabric uses CFT (Crash Fault Tolerant) Consensus. Other algorithms like PoS (Proof of Stake) and PoA (Proof of Authority) perform better than PoW but do not provide full decentralization.

Each study's evaluation environment is unique. It is difficult to replicate studies that use a synthetic evaluation framework and self-designed workloads. The influence of the evaluation environment cannot be ignored. The differences in the evaluation environment make comparing these studies difficult. It can be seen from the table that the evaluation is done in a controlled environment and with very limited nodes. This kind of evaluation provides useful insight into the performance of permissioned DLTs. But it will not capture the dynamic nature of permissionless DLTs. The next section discusses the proposal and implementation of the BloodTrace smart contract which will be used to measure the performance of Ethereum.

Proposal and Implementation of BloodTrace Smart Contract

Proposal

Tracing Blood from donor to patient increases transparency, communication speed and the willingness to donate blood. This tracing also helps in times of epidemic and pandemic to find the origin of disease. In the existing non-blockchain system, it is possible to trace blood to a donor, but not possible to trace blood from the donor to the patient. Before donated blood reaches the patient, it travels through multiple places and is separated into transfusable components plasma, platelets and red cells. Fig. 3 shows the journey of blood from donation to transfusion [21].

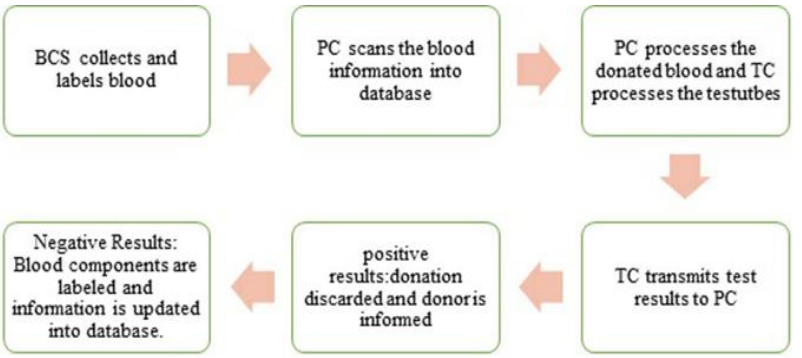

Fig. 3. Journey of Donated Blood

The BCS (Blood Collection Service) collects 1 pint of blood from the donor and some blood in the test tubes for testing. The blood and test tubes are marked with the same bar code. They are sent to the Processing Centre (PC) and Testing Centre (TC) respectively. The PC separates the blood into different components Plasma, Red blood cells and Platelets. TC tests the blood in the test tubes for different diseases. If the tests are positive, the donation is discarded and the donor is informed. If tests are negative, the different components of blood are stored in the blood bank shelf or hospital shelf with labels. The database is maintained for information about donors and blood. After this process, the blood loses visibility. In a non-blockchain solution, there is a system for tracking blood to the donor, but there is no system for tracking blood from the donor to the patient who receives transfusion.

The use of Blockchain can increase visibility and transparency in Blood donation tracing. Whenever any exchange of blood takes place, this exchange will be stored as a transaction on the Blockchain. A full and common history can be made available for donated blood. The proposal builds a PoC system for tracing Blood from donation to transfusion. Fig. 4 shows the participants in a Blockchain-based system to trace the blood from a donor to the patient.

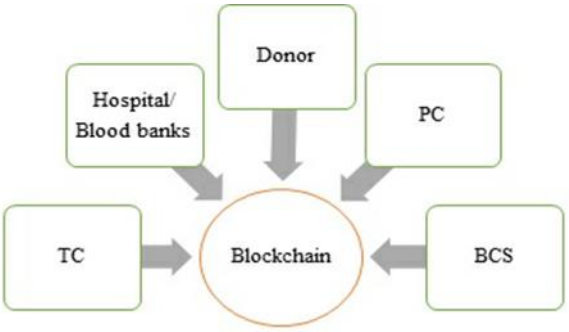

Fig. 4. Participants in BloodTrace System

BCS collects the blood from the donor and collects information about the donor. The TC will perform different tests on the blood from test tubes and upload the result. These tests also include testing for the type of blood and Rh value. The PC will separate the blood into different transfusable components, labels them and store them. The PC needs to update this information on the Blockchain along with donor id and Blood type. The hospitals where transfusion is needed can request the type of blood along with the component of the blood. The Processing Centre provide the requested blood and stores this exchange as a transaction on the blockchain and also informs the donor about the use of his/her donated blood. A donor can view his test results and personal details. Fig. 5 shows the interactions among the stakeholders and Blockchain. Fig. 6 shows the sequence diagram of BloodTrace portraying the communication among all the parties.

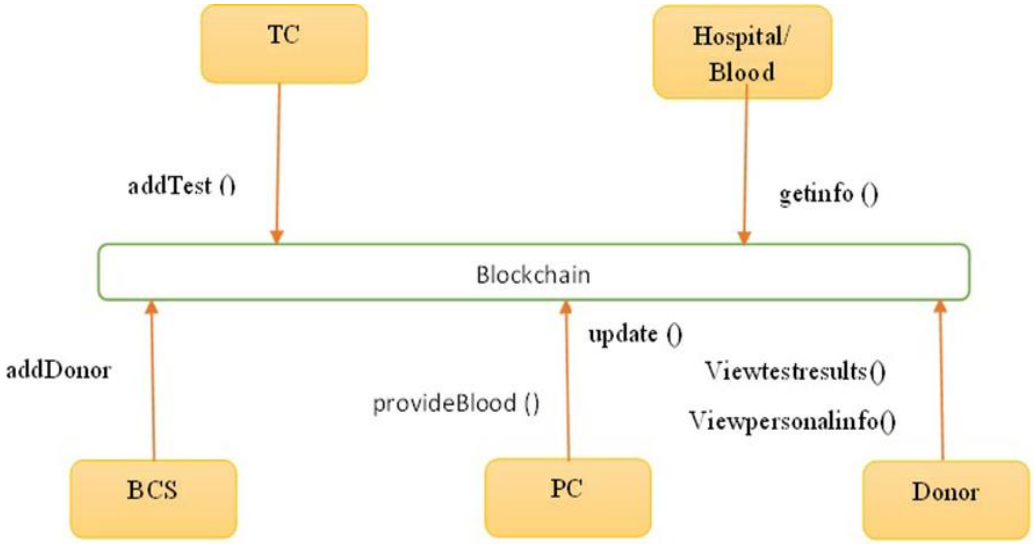

Fig. 5. Interactions of stakeholders with Blockchain

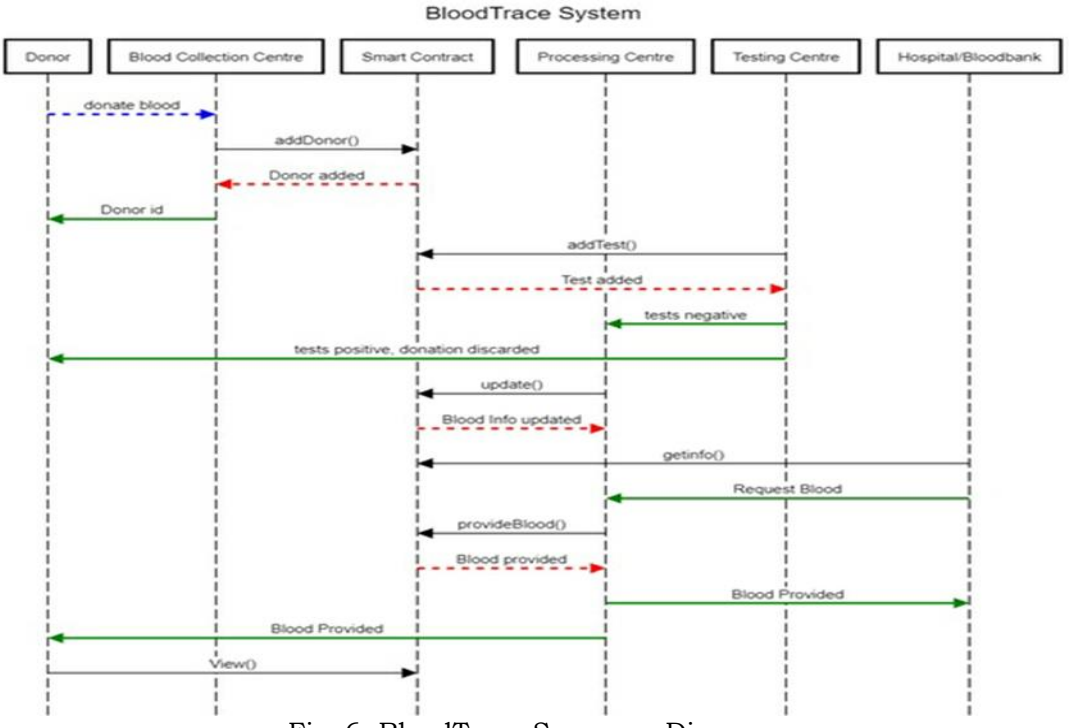

Fig. 6. BloodTrace Sequence Diagram

Implementation

The functions describing the working principles of the BloodTrace system are defined as follows:

1) Access Control List
The contract owner can add Blood collection centres (BCS), Processing centres (PCs), Test centres (TCs) and hospitals. Each centre and hospital will be identified by its Ethereum address. The following functions will help to make Access Control List (ACL). The modifier onlyOwner will allow only the contract owner to add the centres and hospitals. Listing 1 shows the code for the same.

Listing 1

```
modifier onlyOwner { require(msg.sender == owner); _; }
function addbcs(address _bcaddress) public onlyOwner{
        bcsusers.push(_bcaddress);
  }
  function addpcs(address _pcaddress) public onlyOwner {
      pcsusers.push(_pcaddress); }
function addtcs(address _tcaddress) public onlyOwner
  {
      tcsusers.push(_tcaddress);
  }
  function addhosps(address _hcaddress) public onlyOwner {
      hospsusers.push(_hcaddress);
  }
```

2) addDonor and addTest
BCS collects the blood from the donor and adds the donor details and TCs add the test details. Listing 2 shows the code for the same.

Listing 2

```
function addDonor(string memory _dname, address _daddress, string memory _dgender, uint _dage, uint _dweight, string memory _bloodtype) public
   {
       for(uint8 k=0;k<bcsusers.length;k++)
          {if(msg.sender==bcsusers[k])
         {
            donorCount ++;

donors[donorCount]=Donor(donorCount, _dname, _daddress, _dgender, _dage, _dweight, _bloodtype,false);
            donorusers.push(_daddress);
         }
       }
   }
```

```
function addTest(uint _did, bool _testresultspositive, bool _donationdiscarded) public
   {
         for(uint8 k=0;k<tcsusers.length;k++)
           {if(msg.sender==tcsusers[k])
           {

testresults[_did]=Testresults(_did, _testresultspositive,_donationdiscarded);
           }
         }
   }
```

3) update() and getinfo()
The PC updates the blood information through the update() function. By providing a blood group, anyone can check the availability of the blood and processing centre number using the getinfo() function. Listing 3 shows the code for the same.

Listing 3

```
function update(uint _pcid, uint _did, string memory _bloodtype, uint _plasmaunit, uint _redbloodcell, uint _platelet) public
  {
     for(uint8 k=0;k<pcsusers.length; k++)
         {if(msg.sender==pcsusers[k])
        {
            blood[_bloodtype]=Blood(_pcid, _did, _bloodtype, _plasmaunit, _redbloodcell, _platelet);

bloodinfo[_bloodtype].totalplasma += _plasmaunit;
            bloodinfo[_bloodtype].totalredcell += _redbloodcell;

bloodinfo[_bloodtype].totalplatelet += _platelet;
            bloodinfo[_bloodtype].pcid=
```

```
function getinfo(string memory _bloodtype) public view returns(uint, uint, uint, uint)
  {
       uint _pcid;
       uint _totalplasma;
       uint _totalredcell;
       uint _totalplatelet;
       _pcid=bloodinfo[_bloodtype].pcid;

_totalplasma=bloodinfo[_bloodtype].totalplasma;

_totalredcell=bloodinfo[_bloodtype].totalredcell;

_totalplatelet=bloodinfo[_bloodtype].totalplatelet;
       return(_pcid, _totalplasma, _totalredcell, _totalplatelet);
   }
```

```
_pcid;
        }
      }
```

4) provideBlood()

The PC can provide blood components on request from the hospital and inform the donor about the donation. The hospital needs to provide blood type and the component. Listing 4 shows the code for the same.

Listing 4

```
function provideBlood(uint _hid, uint _pid, string memory _bloodtype, string memory _component, uint _quantity) public {
    for(uint8 k=0;k<pcsusers.length;k++) {
        if(msg.sender==pcsusers[k]){

            for(uint8 k = 0; k < donorusers.length; k++){
                uint8 j=k+1;
                if(hashCompareWithLengthCheck(donors[j].bloodtype, _bloodtype){
                    bloodprovided[_bloodtype] = Bloodprovided(_hid, _pid, j, _bloodtype, _component, _quantity);
                    donors[j].donated=true;
                    if(hashCompareWithLengthCheck(_component, "plasma"))
                        bloodinfo[_bloodtype].totalplasma -= _quantity;
                    else if(hashCompareWithLengthCheck(_component,"redcell"))
                        bloodinfo[_bloodtype].totalredcell -= _quantity;
                    else if (hashCompareWithLengthCheck(_component, "platelet"))
                        bloodinfo[_bloodtype].totalplatelet -= _quantity; } }}}
```

5) Viewdonorpersonaldetails() and Viewtestresults()

The donor can view his/her details and test results. Listing 5 shows the code for the same.

Listing 5

```
function Viewdonorpersonaldetails(uint _did) public view returns( string memory _dname, address _daddress, string memory _dgender, uint _dage, uint _dweight, string memory _dbloodtype, bool _ddonated){
    for(uint8 k=0;k<donorusers.length;k++){
        uint8 j=k+1;

if(msg.sender==donorusers[j].daddress){
            _dname=donors[_did].dname;
            _daddress=donors[_did].daddress;
            _dgender=donors[_did].dgender;
            _dage=donors[_did].dage;
```

```
function Viewtestresults(uint _did) public view returns(bool, bool)
    {
      for(uint8 k=0;k<donorusers.length;k++)
          { uint8 j=k+1;

if(msg.sender==donorusers[j].daddress)
          {

return(testresults[_did].testresultpositive, testresults[_did].donationdiscarded);
          }
        }
```

```
            _dweight=donors[_did].dweight;

_dbloodtype=donors[_did].bloodtype;
            _ddonated=donors[_did].donated; }
        }
    }
```

The smart contract also provides ACL (Access Control List) for authenticating the participants of the BloodTrace application. The proposed BloodTrace system is a proof-of-concept system. For real-life implementation, the donor information and the test results can be stored on IPFS in encrypted form. The hash provided by IPFS can be stored on the ledger with donor id. The proposed System traces the blood from donation to transfusion.

Comparison with an existing blockchain-based solution

D. Hawashin et al. [22] proposed a full-fledged blockchain-based blood donation system that allows tracing of the blood from the donor to the patient. The system also traces the blood during transportation. They have used smart contracts deployed on Ethereum private blockchain. They have used QR codes for tracing the blood. Testing of Blood and updating of test results are not included in the system. S. Sadri et al. [23] proposed a blood donation supply chain management system using Ethereum smart contract. They are not using off-chain storage like IPFS. S. Lakshminarayanan et al. [24] suggested a blockchain-based blood management system using Hyperledger Fabric. A web application is also built for ease of access. Table 3 shows the comparison of the proposed system with the existing system. Blood donation management involves participants from all over the country. It is our view that a public blockchain should be used to provide full decentralization, transparency and security. Using IPFS will reduce the storage requirements on blockchain but it will not provide privacy to the donor. The hash provided by IPFS can be stored on the ledger. This hash is publicly visible and can be used to retrieve the donor data from the IPFS. The next section discusses the deployment of the smart contract on Ethereum and the performance evaluation of Ethereum.

Performance Evaluation of Ethereum

The performance of permissionless DLTs can be evaluated in two ways: i) by setting up the private network or ii) by monitoring the DLT. Ethereum is a permissionless DLT that provides full decentralization and security using the PoW algorithm. The performance results availed by establishing a private network are far less accurate and do not capture the dynamic nature of a permissionless DLT.

Table 3. Comparison with existing systems

Features	D.Hawashin et al. [22]	S.Sadri et al. [23]	S.Lakshminarayan et al. [24]	Our system
Blockchain Platform	Ethereum private network	Ethereum private network	Hyperledger Fabric	Ethereum Public network

ACL	Yes	No	No	Yes
Traceability	Yes	Yes	Yes	Yes

Performance measurement of a permissionless DLT necessitates the selection of i) a real public network, ii) performance parameters and iii) a data collection method. This performance evaluation is client-centric and application-specific. But it will work for all similar applications. Client-centric evaluation is important for the practical implementation of DLT in real life. The client would like to know the overall performance of an application and may not be interested in knowing the detailed parameters.

Public Network

The deployment of a smart contract on the Ethereum network can be achieved in many ways. Fig. 7 shows the various ways to establish the network for the deployment of smart contracts. It is expensive to use the real DLT platform for performance measurement. The real ethers are needed to use the Ethereum mainnet for testing and performance evaluation. The permissionless DLTs like Ethereum provide testnets for testing the application using fake ethers. There are multiple testnets for Ethereum such as Ropsten, Gorli, Rinkeby and Kovan[2]. Rinkeby, Gorli and Kovan are PoA based testnets. The Ropsten is a PoW based network that resembles the mainnet. The total blocks on the testnet[3] are 10403804 on 09/06/2021 at 19:51. The nodes are geographically dispersed and dynamic.

Parameters

It can be seen from the literature review that performance parameters throughput and latency are used for comparative analysis of DLTs most of the time. In most of the existing work, the definitions of these parameters are not given. Hyperledger Caliper uses the performance parameters defined in the white paper [25]. The definitions are as follows:

Transaction throughput – total committed transactions/total time in seconds. It is the rate at which valid transactions are committed by Blockchain SUT in a defined period.
Read Latency – Time when the response received – submit time
It is the time between when the read request is submitted and when the reply is received.
Transaction Latency (Finality) – Confirmation time@network threshold – submit time
It is the network-wide view of the amount of time taken for a transaction's effect to be usable across the network. Finality measures how long one has to wait to be given a reasonable guarantee the transaction written in Blockchain is irreversible (will not be orphaned)

[2] https://docs.ethhub.io/using-ethereum/test-networks/
[3] https://ropsten.etherscan.io/

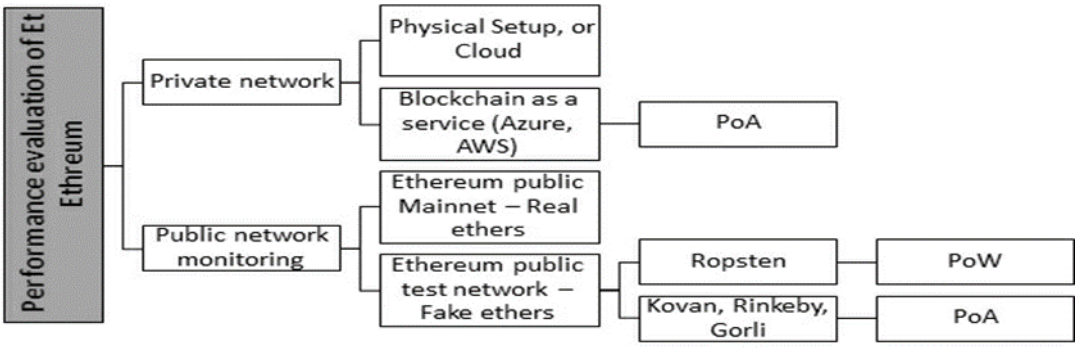

Fig. 7. Deployment of smart contract

In Ethereum, A new block is generated every 15 secs. The 12 Block confirmations (means 12 blocks are getting connected to the block containing the transaction) are needed for achieving the finality in Ethereum[4]. A transaction is said to be confirmed only when the finality is achieved. The latency should be calculated as the time between the transaction submission time and transaction finality time. In Ethereum, when a transaction is included in the block and the block is mined (not achieved finality), the block number is assigned to the transaction. Some of the researchers consider this as a confirmation of the transaction and calculate latency accordingly. This does not consider the situation when a block is lost in the fork and it is not part of the longest chain. This work proposes to use the following parameters for evaluation of Ethereum: *Transaction Throughput and* Transaction Latency (Finality).

Data Collection Methods

There are two ways to collect the data from the permissionless DLT:

RPC based Method the Web3 library can be used to get information from the Ethereum network. It uses RPC based method to extract data from the network. The RPC based method induces overhead and affects the calculation of performance parameters.
Log-based Method This method involves the collection of logs from the network. These logs are then used for the calculation of parameters [17]. The authors claimed that the log-based method has less overhead than RPC based method. This work proposes to compare RPC based and the log-based method. Fig. 8 shows the simple model for the evaluation of permissionless Ethereum.

The steps for using the model are:

- Deploy application/smart contract on the public test network and send transactions
- Collect logs from the test network
- Calculate the parameters.

[4] https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/

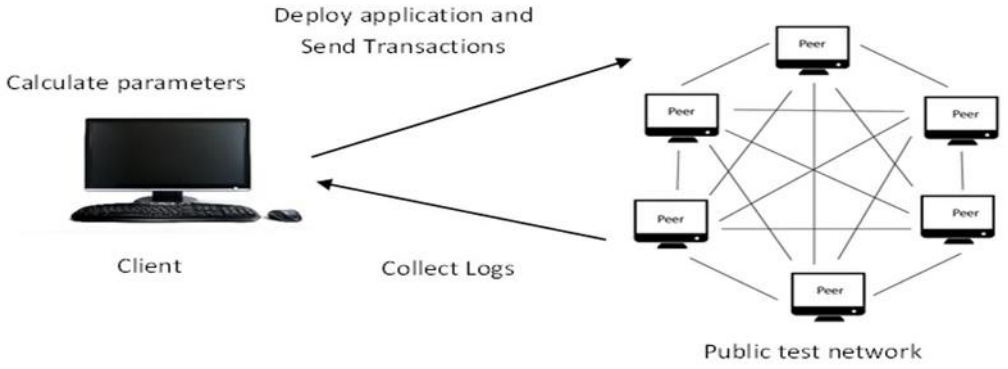

Fig. 8. A model for performance evaluation of permissionless DLT Ethereum

Many applications/smart contracts are running in a public test network at any given time, just as they are on the mainnet. Public test networks capture the dynamic nature of a permissionless DLT. They also provide the client with the perception of real-time interaction with the smart contract.

Deployment of the Blood Trace on Ropsten

The deployment of a smart contract on the Ropsten testnet can be accomplished by using Remix IDE and Metamask Wallet. Remix IDE[5] is an open-source application that facilitates the development, testing and debugging of smart contracts written in solidity language. Metamask is the wallet to store Ethers and ERC-20 tokens enabling the users to make transactions on Ethereum mainnet and testnets. The Ropsten test network highly resembles the Ethereum mainnet and facilitates the use of fake ethers. The faucet[6] helps in getting fake ethers to the Metamask wallet. The BloodTrace smart contract was deployed on the Ropsten test network.

Experimentation and Results

The performance parameters are measured using Hyperledger Caliper Tool and the log-based method. Hyperledger Caliper It is a framework for DLT performance benchmarking [26]. Hyperledger Besu, Ethereum, Hyperledger Fabric, and FISCO BCOS are the supported platforms. Transaction/Read Throughput, Transaction/Read Latency, and Resource Consumption are supported metrics. The Caliper generates load for the System Under Test (SUT) and monitors it. A network configuration file, a benchmark configuration file, and workload modules are used. The transaction rate, number of rounds, and SUT monitoring settings are included in the benchmark configuration file. This file is independent of the underlying SUT. The network configuration file contains information about the SUT, such as its type, as well as the endpoint addresses of the nodes and smart contracts. Workload modules contribute to the creation of transaction content. Because each round can be associated with a separate module, it is simple to separate workload implementation based on the SUT's phase or behaviour. The

[5] http://remix.ethereum.org/
[6] https://faucet.ropsten.be/

Caliper can support multiple SUTs owing to a connector-based interface. The Caliper makes use of a single manager process and multiple worker processes. The manager process starts off the SUT and generates the performance report. Workloads are generated by worker processes, which can be configured on a single machine or distributed across multiple machines in the network. The worker processes are responsible for Caliper's scalability.

Experimentation

The Hyperledger Caliper was used with one client having the configuration i5 Intel Core, 4 cores, 1.60 GHz, 8 GB RAM, 500 GB hard disk. The Caliper@cli was downloaded and installed [26]. The benchmark configuration file was set up for addDonor and viewDonor functions of the smart contract with 1 worker and the rate of the transaction were set at 50 transactions per second(tps). The network configuration file was set up with the address of the Ropsten network through the Infuria node. The block confirmations were set up as twelve and then zero. The readings were averaged for three runs to incorporate the fluctuations in internet speed and the dynamic nature of Ropsten DLT. Hyperledger Caliper shows large variations in the values of the parameters for the same client and the same internet speed connection.

Read Throughput and Read Latency

The viewDonor function is a read query and does not require consensus. Table 4 shows the read latency and throughput values with the number of transactions. The average throughput is almost 100% and the average latency is 0.22. The read operation includes time to send requests from the client to the Infuria node. The read latency and read throughput graphs are shown in Fig. 9.

Write Throughput and Write Latency

The addDonor function will write on the ledger. Tables 5 and 6 show the write throughput and write latency values for the addDonor function with twelve block confirmations and zero block confirmations respectively. There were 3 runs for each batch of transactions. Fig. 10 shows the graphs for the same.

Transactions	Max Latency(s)	Min Latency(s)	Avg. Latency(s)	Throughput
100	0.26	0.21	0.22	46.8
200	0.39	0.21	0.23	47.9
500	0.25	0.21	0.22	49.1

Table 4. viewDonor latency and Throughput values

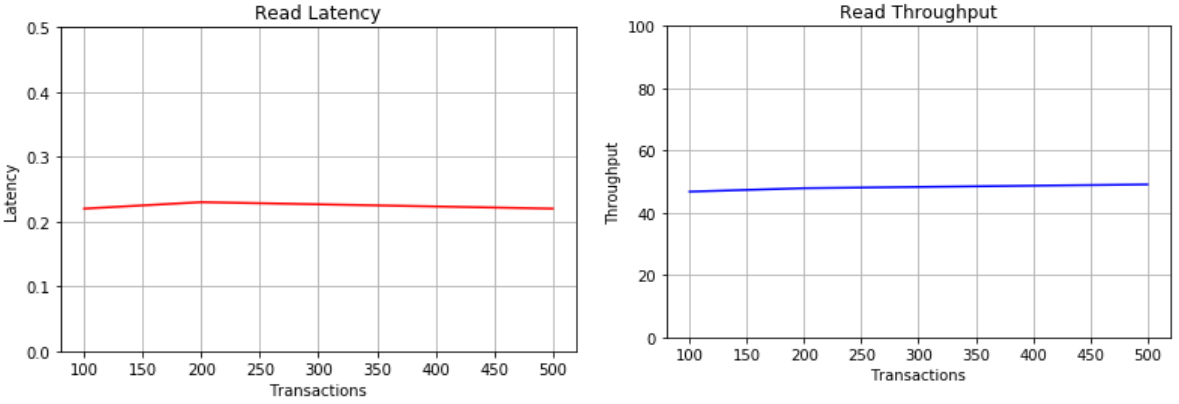

Fig. 9. Read Latency and Read Throughput using Hyperledger Caliper

Block Confirmations

Ethereum is a permissionless DLT with probabilistic consensus PoW. The average block time of Ethereum is 15 secs. The transaction needs to be part of the block which is on the longest chain. To make sure a block B1 is part of the longest chain, 12 more blocks need to get attached to block B1. This makes sure that transactions in block B1 are irreversible. Block confirmations must be part of latency calculations [25].

The latency increases with the number of transactions but the variation is less in throughput. Also, it can be seen that the block confirmations do not affect the latency and throughput. A network configuration file allows to set up the block confirmations, but varying this parameter does not show variation in throughput and latency. The slight variation is due to internet speed fluctuations. In the case of a few runs, the minimum latency is less than 10 secs, which is also not consistent with 12 block confirmations.

Log Based Method

This method involves the collection of logs for the calculation of performance parameters and it is inspired by the idea presented in the paper [17]. Collecting logs will provide details about a batch of transactions at the same time, whereas polling the DLT for each transaction confirmation will increase the overhead.

Table 5. addDonor Write latency and Write throughput (12 Confirmations Hyperledger Caliper)

Transactions	Max Latency(s)	Min Latency(s)	Avg. Latency(s)	Avg Latency over 3 runs	Throughput	Avg throughput over 3 runs
	24.03	17.83	20.26		3.9	
100	34.57	17.14	27.66	21.75	2.8	3.4
	25.98	10.86	17.33		3.6	
200	56.33	26.82	38.79	35.25	3.3	3.5

	60.87	9.50	36.05		3.1	
	44.40	13.57	30.92		4.1	
	112.79	3.40	69.75		4.1	
500	103.02	14.73	64.89	58.84	4.4	4.7
	78.70	10.05	41.89		5.6	

Table 6. addDonor Write latency and Write throughput (0 Confirmations Hyperledger Caliper)

Transactions	Max Latency(s)	Min Latency(s)	Avg. Latency(s)	Avg Latency over 3 runs	Throughput	Avg throughput over 3 runs
	17.11	4.53	14.95		5.4	
100	20.48	15.84	17.39	17.00	4.5	4.5
	25.68	3.69	18.67		3.7	
	41.05	11.06	21.97		4.4	
200	54.22	24.71	34.14	40.58	3.4	3.3
	95.07	4.74	65.64		2.0	
	124.44	10.69	70.89		3.7	
500	115.75	15.42	50.41	58.197	4.0	4.2
	91.44	16.92	53.29		5.0	

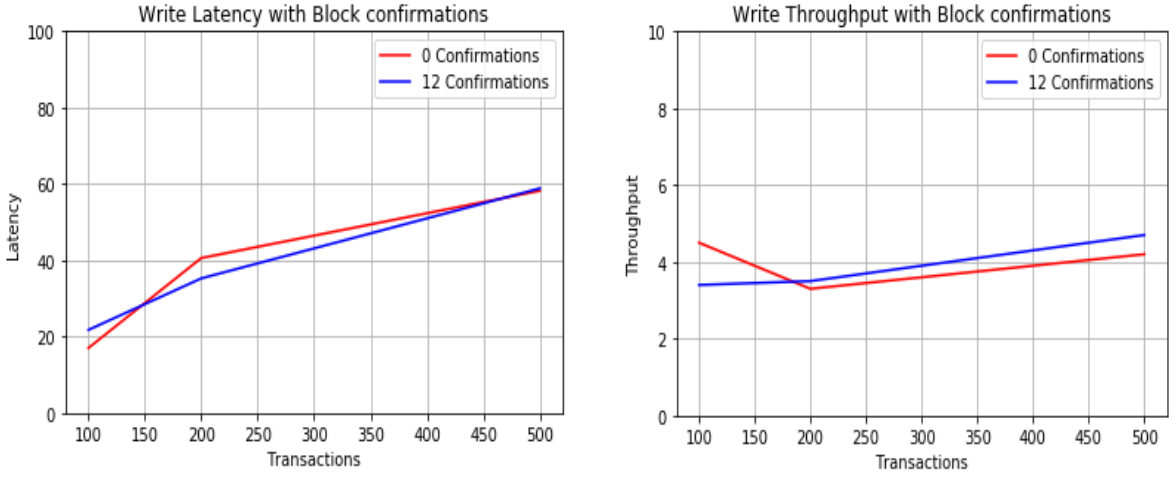

Fig. 10. Write Latency and Write Throughput using Hyperledger Caliper

A javascript is written to generate load based on the number of transactions and the transaction rate parameters. The transaction logs are collected with the Ropsten API[7]. As soon as the block is collated, a transaction receipt is generated. This transaction receipt contains the timestamp of the collated block. Most of the existing research considers this collated timestamp as a confirmation timestamp. But if a fork happens and future blocks are not making this block as the ancestor, then the block is not confirmed and the transactions inside the block

[7] https://ropsten.etherscan.io/apidocs

are also not confirmed. Minimum 12 block confirmations are needed as discussed in section 4.2. Table 7 provides the values of the parameters based on 0,6,12 confirmations. Fig. 11 shows the graphs for the same.

Table 7. Values of the parameters using Log-based method with different block confirmations

Number of transactions	Block Confirmations	Throughput	Max Latency(s)	Minimum Latency(s)	Average latency(s)
100	0	7.7	48	4	19.26
200	0	5.5	48	3	20.12
500	0	4.6	48	4	23.12
100	6	7.1	127	50	87.52
200	6	5.6	127	50	89.05
500	6	4.6	127	50	95.81
100	12	7.4	236	50	177.48
200	12	5.8	236	50	191
500	12	4.7	236	50	218.87

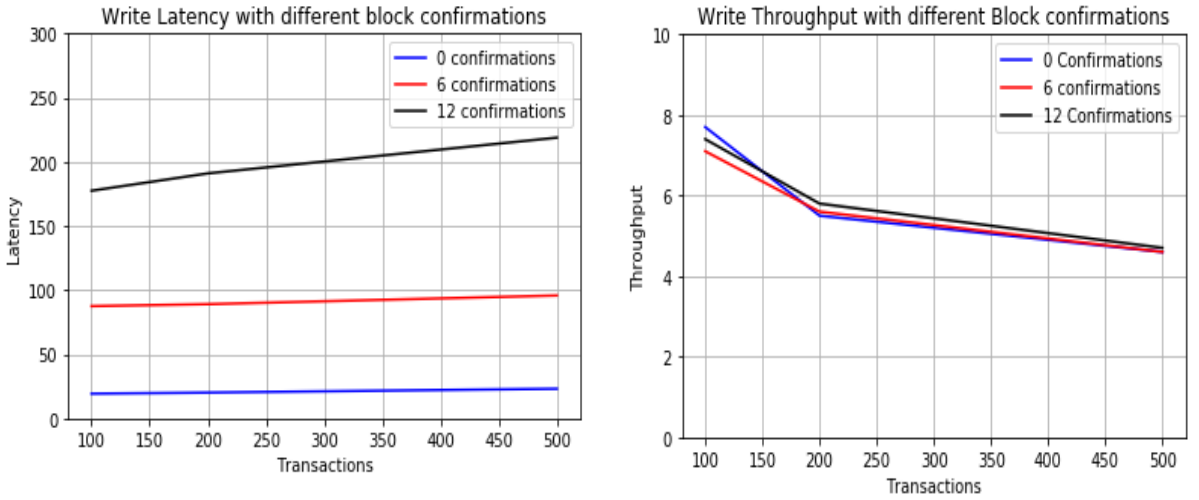

Fig. 11. Write latency and throughput with Log-based method

It can be seen from the graph that as the number of transactions is increasing, the latency increases and throughput decreases. Throughput depends on the number of transactions in the block and average block creation time**.** Block creation time is the time between creating the blocks. The block creation time varies from 10-15 seconds in Ethereum. As soon as the transaction is included in the block, a transaction receipt is generated with the block creation timestamp. All the transactions in the block are having the same timestamp. That means, when a block is confirmed, all the transactions in the block are confirmed. If the batch of transactions is included in the same block, the latency of all the transactions in the block will remain the same. If the transactions are dispersed in different blocks, the throughput will decrease and latency will increase. The

throughput is calculated as the total number of transactions in the confirmed blocks to the total average block creation time.

Latency also increases with the number of block confirmations. There is a linear increase in latency from zero confirmations to 12 confirmations Fig. 12 shows the latency with 12 confirmations for Hyperledger Caliper and Log based method. Hyperledger Caliper does not reflect the block confirmations in the calculation of latency.

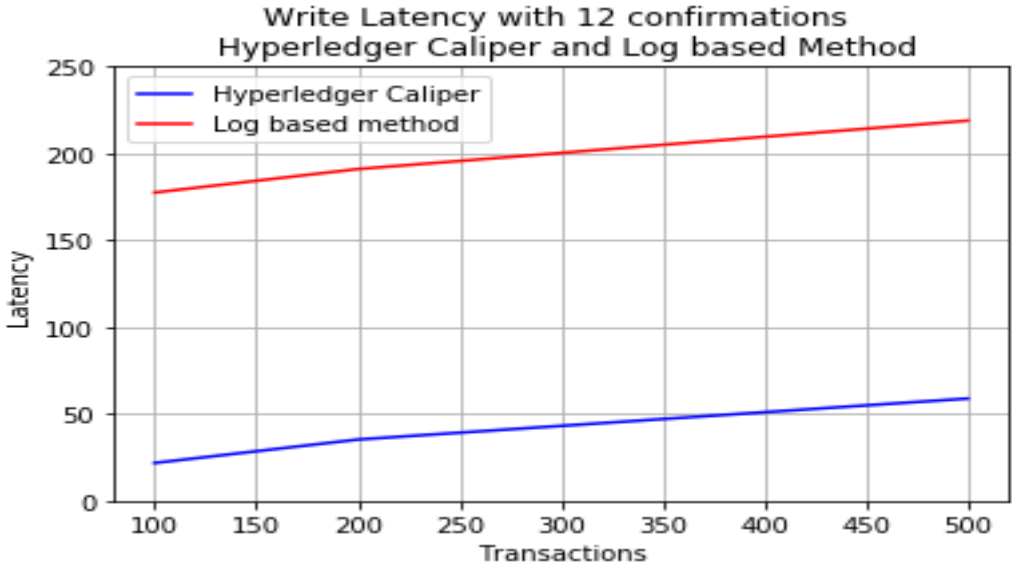

Fig. 12. Write latency Hyperledger Caliper and Log based Method

A new block is created every 10-15 seconds in Ethereum which leads to faster inclusion of the transaction into the block. It also leads to the generation of orphan blocks (uncle blocks). A client has to make sure that his/her transaction is included in the block which is part of the longest chain. So, the transaction confirmation in Ethereum takes more time. A client has to wait for twelve block creation time to get the confirmation of the transaction. Many clients select fewer transaction confirmations for less important transactions. Table 8 shows the different Block confirmations (bc) along with the latency (L) values for a batch of 100 transactions. The Linear Regression is used to predict latency values for the given block confirmations as there is a linear relationship between block confirmations and latency.

Table 8. Block confirmations and Latency (log-based method)

Block Confirmations	0	2	4	6	8	10	12
Latency	19.26	42.23	64.12	87.52	118	147.56	177.48

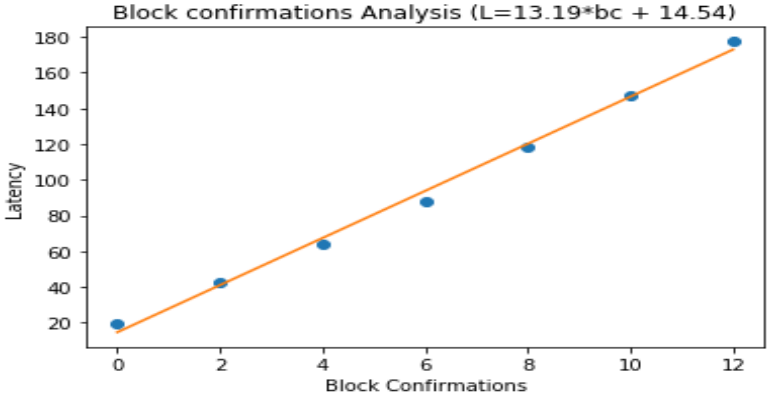

Fig. 13. Block confirmations Analysis

This performance evaluation of Ethereum has some limitations. The number of nodes and node configurations are not known and they are dynamic. So, it is difficult to compute resource consumption parameters and fine-tune the platform-specific parameters. But it also provides the perspective in terms of real-time performance as the public test networks in Ethereum has dynamic topology like the real mainnet.

Conclusion and Future Work

Conclusion

The paper defined the model and performance parameters for performance measurement of permissionless Ethereum using a public testnet. The paper also described the implementation of the BloodTrace application and deployment on the Ropsten. The performance parameters were calculated using Hyperledger Caliper and Log based method. The paper also included block confirmations in the calculation of the latency and compared it with Hyperledger Caliper. Measuring the performance of a permissionless DLT in a controlled environment will not yield accurate results. The topology of permissionless DLTs is unknown. The evaluation environment is ever-changing. Public test networks closely resemble the main network and capture the dynamic nature of permissionless DLT.

Log-based data collection method induces less overhead compared to RPC based method. Still, some RPC calls are required to collect some data. But polling the blockchain for each transaction is not required. This reduces the overhead. The log-based data collection method can be extended for other DLTs as well. Block confirmations play an important role in ensuring that the transaction is irreversible. Hyperledger Caliper does not show the effect of block confirmations on the latency, while the log-based method clearly shows the effect of block confirmations on the latency.

Future work

The future work includes using the log-based data collection method for performance evaluation of other DLTs and comparing their performances.

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. “Ethereum | Hyperledger Caliper.” https://hyperledger.github.io/caliper/v0.4.2/ethereum-config/ (accessed: Jul. 31, 2021).
2. “Getting Started | Hyperledger Caliper.” https://hyperledger.github.io/caliper/v0.3.2/getting-started/ (accessed: Apr. 26, 2021).
3. “Hyperledger Blockchain Performance Metrics White Paper.” https://www.hyperledger.org/learn/publications/blockchain-performance-metrics (accessed: Apr. 26, 2021).
4. Baliga, I. Subodh, P. Kamat, and S. Chatterjee, “Performance Evaluation of the Quorum Blockchain Platform.” ArXiv, abs/1809.03421.
5. C. Fan, S. Ghaemi, H. Khazaei and P. Musilek, "Performance Evaluation of Blockchain Systems: A Systematic Survey," in IEEE Access, vol. 8, pp. 126927-126950, 2020, doi: 10.1109/ACCESS.2020.3006078.
6. C. Faria and M. Correia, ``BlockSim: Blockchain simulator,'' in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 439-446.
7. D. Hawashin et al., "Blockchain-Based Management of Blood Donation," in IEEE Access, vol. 9, pp. 163016-163032, 2021, doi: 10.1109/ACCESS.2021.3133953.
8. D. Saingre, T. Ledoux and J. -M. Menaud, "BCTMark: A Framework for Benchmarking Blockchain Technologies," 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), 2020, pp. 1-8, doi: 10.1109/AICCSA50499.2020.9316536.
9. E. H. P. de Arruda, R. C. Lunardi, H. C. Nunes, A. F. Zorzo and R. A. Michelin, "Appendable-block Blockchain Evaluation over Geographically-Distributed IoT Networks," 2020 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), 2020, pp. 1-6, doi: 10.1109/BlackSeaCom48709.2020.9235000.
10. F. Leal, A. E. Chis, and H. González–Vélez, “Performance Evaluation of Private Ethereum Networks.” SN Computer Science, vol. 1, no. 5, 2020, doi: 10.1007/s42979-020-00289-7.
11. J. Sedlmeir, P. Ross, A. Luckow, J. Lockl, D. Miehle, and G. Fridgen, “The DLPS: A New Framework for Benchmarking Blockchains.” Proceedings of the 54th Hawaii International Conference on System Sciences, 2021, doi: 10.24251/hicss.2021.822.
12. M. Alharby and A. van Moorsel, ``Blocksim: A simulation framework for blockchain systems,'' *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, pp. 135-138, 2019.

13. M. Bez, G. Fornari, and T. Vardanega, ``The scalability challenge of ethereum: An initial quantitative analysis,'' in Proc. IEEE Int. Conf. Service-Oriented Syst. Eng. (SOSE), Apr. 2019, pp. 167-176.
14. M. Dabbagh, K. Choo, A. Beheshti, M. Tahir and N. Safa, "A survey of empirical performance evaluation of permissioned blockchain platforms: Challenges and opportunities", Computers & Security, vol. 100, p. 102078, 2021. Available: 10.1016/j.cose.2020.102078.
15. P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, ``A detailed and realtime performance monitoring framework for blockchain systems,'' in *Proc. 40th Int. Conf. Softw. Eng. Softw. Eng. Pract. (ICSE-SEIP)*, 2018, pp. 134-143.
16. Padmiswari, A. A. I. M., Wulansari, N. T., Antari, N. W. S., Damayanti, I. A. M., Indrayoni, P., & Indrawan, G. S. (2021). The effectiveness of soaking duration on blood cockles (Anadara granosa) with activated charcoal towards reducing metals lead (Pb). *International Journal of Health & Medical Sciences*, *4*(3), 304-308. https://doi.org/10.21744/ijhms.v4n3.1756
17. Redcrossblood.org, "What Happens to Donated Blood," 2020. Accessed on: July 1, 2020. [online] Available: https://www.redcrossblood.org/donate-blood/blood-donation-process/what-happens-to-donated-blood.html.
18. S. Benahmed, I. Pidikseev, R. Hussain, J. Lee, S. M. A. Kazmi, A. Oracevic, and F. Hussain, ``A comparative analysis of distributed ledger technologies for smart contract development,'' in *Proc. IEEE 30th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2019, pp. 1-6.
19. S. Lakshminarayanan, P. N. Kumar, and N. M. Dhanya, "Implementation of Blockchain-Based Blood Donation Framework." IFIP Advances in Information and Communication Technology, pp. 276-290, 2020, doi: 10.1007/978-3-030-63467-4_22.
20. S. Pandey, G. Ojha, B. Shrestha, and R. Kumar, ``BlockSIM: A practical simulation tool for optimal network design, stability and planning,'' in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 133-137.
21. S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance Analysis of Private Blockchain Platforms in Varying Workloads." 2017 26th International Conference on Computer Communication and Networks (ICCCN), 2017, doi: 10.1109/icccn.2017.8038517.
22. S. Rouhani and R. Deters, ``Performance analysis of ethereum transactions in private blockchain,'' in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2017, pp. 70-74.
23. S. Sadri, A. Shahzad and K. Zhang, "Blockchain Traceability in Healthcare: Blood Donation Supply Chain," 2021 23rd International Conference on Advanced Communication Technology (ICACT), 2021, pp. 119-126, doi: 10.23919/ICACT51234.2021.9370704.
24. Suryasa, I. W., Rodríguez-Gámez, M., & Koldoris, T. (2021). Health and treatment of diabetes mellitus. *International Journal of Health Sciences*, *5*(1), i-v. https://doi.org/10.53730/ijhs.v5n1.2864
25. T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, ``BLOCKBENCH: A framework for analyzing private blockchains,'' in *Proc. ACM Int. Conf. Manage. Data (SIGMOD)*, 2017, pp. 1085-1100.
26. T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, ``Untangling blockchain: A data processing view of blockchain systems,'' IEEE Trans. Knowl. Data Eng., vol. 30, no. 7, pp. 1366-1385, Jul. 2018.

27. W. M. Shbair, M. Steichen, J. François and R. State, "Blockchain orchestration and experimentation framework: A case study of KYC," NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1-6, doi: 10.1109/NOMS.2018.8406327.
28. Z. Zhou, R. Li, Y. Cao, L. Zheng and H. Xiao, "Dynamic Performance Evaluation of Blockchain Technologies," in IEEE Access, vol. 8, pp. 217762-217772, 2020, doi: 10.1109/ACCESS.2020.3040875.