Peer-to-Peer Networking and Applications (2021) 14:3075–3091
https://doi.org/10.1007/s12083-021-01148-9

An experimental study on performance of private blockchain in IoT applications

Xuan Chen[1] · **Kien Nguyen**[1] · **Hiroo Sekiya**[1]

Received: 6 December 2020 / Accepted: 31 March 2021 / Published online: 12 May 2021

Abstract

Blockchain includes distributed records that are immutable and transparent through replicating among public or private networks. The open-source Ethereum is one of the emerging blockchain platforms featuring smart contracts. The private Ethereum has been obtaining interest due to its applicability in various applications, including the Internet of Things (IoT). Hence, understanding and quantifying blockchain performance is crucial to facilitate the blockchain application. In this paper, assuming IoT scenarios, we conduct an experimental study to investigate various performance parameters of private Ethereum networks. Initially, we clarify the latency processes according to the transaction lifecycle (i.e., transaction-oriented and block-oriented latency) and measure them in different deployments. Then, we track and report the performance of blockchain nodes during the processes of utilizing transaction. Our deployment networks include an indoor IoT blockchain network (i.e., with a laptop and several Raspberry Pi 3b+ (RPI 3b+)) and a private blockchain over the cloud. In both cases, we write and deploy a smart contract to read and write data to the blockchain and measure the performance in various scenarios. The experiment results reveal not only the blockchain node's performance but also the latencies-hop correlation, as well as the latencies' relation in different workloads. Notably, the latency values in the cloud deployment latency strongly depend on Round Trip Time (RTT) between the blockchain nodes.

Keywords Private blockchain · Ethereum · IoT · Performance · Latency

1 Introduction

Blockchain technology has been gaining popularity with various applications, such as finance [18], healthcare [28], and the Internet of Things (IoT) [29]. A blockchain network is a distributed ledger, which can be replicated and shared among its nodes. Blockchain networks are transparent because any node can view all historical records. The records are immutable because they are reserved eternally under 51% rule [3]. Thus, we can build a network that allows nodes to share information without pre-trusting each other using blockchain. All information submitted to a blockchain is moduled in transaction format. The mining nodes, which maintain a full copy of the historical transactions and blocks, can generate new blocks by verifying new transactions with the proof-of-work (PoW) consensus protocol. The blockchain networks are divided into two categories: public and private. The former is known as a "permissionless" network, to which anyone can join and leave unrestrictedly. On the contrary, the latter is a "permissioned" network, where the nodes need to be permitted by an administrator to join the network [27].

The open-source Ethereum is one of the most popular blockchain platforms [11]. On the one hand, the public Ethereum network (i.e., the Mainnet) is associated with the *Ether* cryptocurrency (i.e., similar to the well-known Bitcoin [7]). On the other hand, Ethereum allows users to deploy a private block network with a self-configured genesis file. Ethereum has smart contracts that

This article is part of the Topical Collection: *Special Issue on Blockchain for Peer-to-Peer Computing*
Guest Editors: Keping Yu, Chunming Rong, Yang Cao, and Wenjuan Li

✉ Kien Nguyen
nguyen@chiba-u.jp

Xuan Chen
chenxuan@chiba-u.jp

Hiroo Sekiya
sekiya@faculty.chiba-u.jp

[1] Graduate School of Science and Engineering, Chiba University, 1-33, Yayoi-cho, Inage-ku, Chiba-shi, Chiba, Japan

are automatically executed computer programs without external trusted authority. The execution of smart contracts is in the Ethereum Virtual Machine (EVM), in which transactions activate the operations. Once a node receives a new block, it verifies and runs functions triggered by the inside transactions. The blockchain mechanism ensures the execution outcomes are the same across nodes among a blockchain network.

The private Ethereum shows a substantial potential for the IoT system, which includes a group of devices. The devices typically cooperate and share information via wireless communication. In such a context, the smart contracts, which are immutable and transparent, can allow the sharing of information among trustless IoT nodes. Those characteristics significantly improve the IoT devices' cooperation [22]. Besides, the blockchain can extend the scalability of the IoT system with the distribution of records. It hence avoids a single point failure that is caused by the decentralized nature. One of the promising IoT applications for private Ethereum is in smart-home scenarios [20, 23], which allows home appliances to store, share or modify states cooperatively. Moreover, in some IoT applications, the devices from different geographical locations may need to cooperate over the Internet (via various cloud computing infrastructures) [25]. Therefore, it is crucial to understand the Ethereum performance for those scenarios.

This research introduces a new method to investigate the private Ethereum blockchain's performance by experimental study. This research's uniqueness first includes clarification of two propagation steps, which are the transaction-oriented and block-oriented latencies, in an indoor testbed and an Ethereum deployment over Google Cloud Platform (GCP). Moreover, we introduce a set of lightweight monitoring tools, which effectively gather the performance parameters of devices running the blockchain node in the two deployments. More specifically, the testbed includes a laptop and four Raspberry Pi 3b+ aiming to mimic a home-based IoT application. In the case of GCP deployment, we deploy three virtual machine instances distributed on different continents. In each case, the network nodes form a private Ethereum blockchain network wherein there is a preloaded smart contract to read and write strings to the blockchain. We measure the dissemination of writing-related transactions in a baseline and a realistic scenario following the new latency definition. Simultaneously, we use the monitoring tool to collect the blockchain node's performance metrics such as CPU, disk usage, memory usage, and network throughput. The delay measurement results show that the latency increases proportionally to the hop number in the indoor testbed's realistic scenario. Besides, the transaction-oriented latency is lower than the block-oriented latency when the number of transactions is small. The opposite is observed when the number of transactions is relatively large. The total latency (without the mining time) indicates that the system spends an average of 63.92, 117.39, 172.38, and 229.21 milliseconds to propagate a single transaction effectively in one, two, three, four hops, respectively. In GCP, we observe the single-hop transmission over long-distance on the private blockchain. The measurement results are highly associated with the Round Trip Time (RTT). In all experiments, our monitoring tool well captures the performance metrics.

The remainder of the paper is organized as follows. Section 2 presents related works. In Section 3, we introduce the background of information propagation protocols in Ethereum network. In Section 4, we present our methodologies of two procedures of latency and the constitution of the monitoring pattern in this work. Section 5 describes the experiment setting and evaluation results. Finally, Section 6 concludes the paper.

2 Related work

The current IoT is constructed on a central server model, in which all devices have to connect to the server to ensure the authentication and communication [5]. The model may have issues, for example, when dealing with scalability. Thus, it is essential to transfer a centralized model to a decentralized one [5]. The blockchain technology is attractive as one of the candidates for decentralizing IoT systems. In [15], the author discusses the possibility of blockchain to strengthen IoT. The work in [16] provides a comprehensive review and analysis of blockchain solutions for the IoT systems. It also shows the potential of integrating blockchain and IoT to solve current IoT issues.

There have already been many interested in combining the blockchain and IoT aiming to accelerate their adoption speeds [13]. In [14], the authors described a survey of the state-of-the-art combinations between the technologies. The blockchain enables a distributed peer-to-peer (P2P) network in which nodes don't need to trust other nodes through a trusted third party. This feature means that the nodes can reach a reconciliation faster and potentially increases the network scalability. Moreover, the utilization of smart contracts makes it possible for researchers and developers to fulfill the different demands of the IoT. With smart contracts, IoT devices can run multi-step processes automatically in a distributed method.

Up to date, there are many applications in which the blockchain has been beneficial for IoT networks. In [24], the authors proposed to use Ethereum to manage IoT devices. However, they only show proof of concept in a scenario with a limited number of IoT devices. In [20], the authors

presented an overview of the private Ethereum blockchain-based smart home system (SHS). The SHS is defined as an integration of home appliances and sensors, which obtain and share information for each other. The presented SHS used a smart home miner to manage the private blockchain, and several non-mining sensors to deliver data to local storage. In [23], the authors presented a more realistic smart home application with a private Ethereum blockchain, which composes of four major components (i.e., temperature and humidity sensors, a smartphone-based visualization application, a RPI 3b, and a computer). The RPI with sensors collects sensing data and calls a preloaded smart contract. The computer was used to manage a private blockchain. In both systems, there is always a computer to maintain the blockchain network. That is because IoT devices usually don't have enough capacity to conduct mining process nor store the full copy of the blockchain. However, both works did not investigate the performance of private blockchain. In [1] and [2], the authors introduce an architecture of a smart home containing several local private blockchain networks, which communicate with each other through an elected cluster header (CH). Each CH mines blocks and implement access control for its local private blockchain network.

The latency and performance of blockchain platform have been one of their most concerned shortcomings [21]. The work in [4] introduces an evaluation framework for analyzing private blockchain performance, which divides the blockchain into a four-layer modulo. The performance is evaluated on each layer. [26] inherites [4]'s evaluation, extends it with variations workloads. However, both the works interpret the latency with the adoption of a single node when the long-distance distributed nodes require a significant time to propagate data. Many studies have evidenced the performance of IoT nodes. [6] characterize latency in the private Ethereum network in three transaction life processes, which do not consider the distance among different nodes. [30] propose a log-based real-time framework to monitor Ethereum. In comparison, we have more intuitive parameters.

Regarding latency in the blockchain network, there are many interpretations of the definition according to different scenarios. For a traditional payment blockchain system [7], a transaction may need several blocks to be accepted to the main chain depending on the attached transaction fee, where the speed of generating blocks is essential. Latency is explained as the mining time. In several papers, latency is explained as the time between submitting a transaction and the first confirmation of acceptance [17]. Because after the first confirmation, the transaction becomes more convinced to gain widespread adoption. While, for our distributed IoT system, the network transmission latency can't be ignored. Both the first confirmation and authorized block propagation are achieved on the long-distance transmission. Unlike others, in [6], we initially present a method of characterizing the latency of information propagation in the private Ethereum blockchain. This paper extends the work to include the IoT and the deployment on the Google cloud platform. Moreover, we newly add the monitoring tools and monitoring results.

3 Background

Ethereum blockchain is essentially a transaction-based state machine [11]. The information of present state composes of account balances, data of smart contracts, etc. Any nodes in the blockchain network can submit transactions to modify the state machine. The submitted transaction on a node will broadcast to all other nodes. Each node maintains a transaction pool (txpool) to keep all pending transactions. A mining node will select some transactions from txpool to form a block and reach a consensus with the PoW algorithm [31]. After that, it will form and broadcast the block to other nodes. All other nodes need to confirm the correctness of the hash value contained in the block header. If the hash is validated, the block will be appended to the local blockchain database. When the block modification has been done by most of the nodes, they will reach a consensus for the state modification. Geth [9] is the official implementation of Ethereum nodes. The nodes form the Ethereum blockchain network following the P2P networking protocols named DEVp2p [8]. Devp2p includes a node discovery protocol and a RLPx transport protocol, which are based on UDP, TCP, respectively (as shown in Fig. 1a).

3.0.1 Node discovery protocol

Geth implements its node discovery protocol based on a Kademlia-like Distributed Hash Table (DHT) [19] for efficiently locating and storing content in a P2P network. In private Ethereum blockchain network, nodes are expected to join the network manually by the administrator, the node discovery protocol is used to routing peers. Every node keep a 256-bit identity or "node ID" randomly generated from Secp256k1 elliptic curve. The logical distance between two nodes is defined as the bitwise XOR of two nodes ID (a and b) as the following equation:

$$distance(a, b) = a \oplus b \tag{1}$$

Every node is also expected to maintain an Ethereum Node Records (ENR) containing up-to-date information of itself, including node ID, IP address, TCP and UDP port, etc. Nodes keep information about other nodes in their "neighborhood". The information of neighbor nodes are stored in a routing table. According to the integer value of

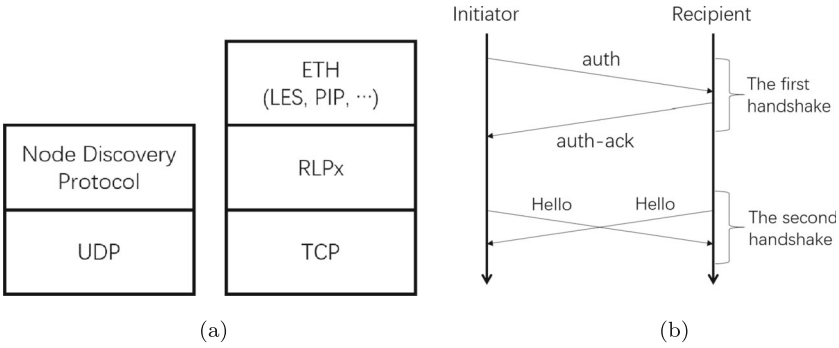


Fig. 1 Transmission protocols in Ethereum network

the distance, Ethereum divides the routing table to several 'k-buckets.' For each $0 \leq i < 256$, every node keeps a k-bucket for nodes of distance between 2^i and 2^{i+1} from itself. The current protocol uses $k = 16$, which means every k-bucket contains up to 16 node entries. The node entries are sorted in an update order —most recently updated at the tail and least recently updated at the head.

3.0.2 The RLPx transport protocol

The RLPx transport protocol is a TCP-based transport protocol used for communication among Ethereum nodes. Recursive Length Prefix (RLP) is a protocol to encode arbitrarily nested arrays of binary data to serialize messages in Ethereum. Based on RLP, RLPx enables nodes to transfer encrypted, serialized data.

In RLPx, two nodes need to perform a two-phase handshake to initialize the session before transmitting essential messages. Figure 1b presents an overview of two handshakes. The first handshake pertains to the exchange of public keys that are used for the subsequent communication. The subsequent messages are therefore encrypted and authenticated. The second handshake pertains to the negotiation on the subsequent capabilities with a $Hello$ message instantly after the first handshake. An RLPx connection is established by creating a TCP connection and agreeing on a pair of an ephemeral key for further encrypted and authenticated communication. The process of creating a session keys between the 'initiator' (the node which opened the TCP connection) and the 'recipient' (the node which accepted it) is the first handshake. The initiator generates the ephemeral key with a shared secret and sends an $auth$ message containing the encrypted shared secret to the recipient. The recipient decrypts and generates the same key with the shared secret. Then it responds an $auth - ack$ message to the initiator. All messages following the first handshake are framed.

After the ephemeral key is negotiated, both sides of the connection send a $Hello$ message or a $Disconnect$ message, which is considered the second handshake. The $Disconnect$ message Inform the peer that a disconnection is imminent. The sender can append a single byte of reason code in the message on this disconnection. Alternatively, the $Hello$ message exchanges their supporting capabilities and the corresponding version. Two sides of nodes negotiate which capability to use in the subsequent communication with $Hello$ messages.

3.0.3 Ethereum wire protocol

Based on the RLPx protocol, Ethereum utilizes different capabilities in different clients or conditions. The most widely used subprotocol is the Ethereum Wire Protocol (ETH), which is used to exchange blockchain information between "full" nodes. The Light Ethereum Subprotocol (LES) is a protocol used by the "light" nodes, which only download block headers and fetch other parts of the blockchain on demand. It provides full functionalities of safely accessing the blockchain. Clients running LES do not mine blocks. Therefore they do not take part in the consensus process. The Parity Light Protocol (PIP) is a variation of LES for Parity Ethereum clients. We use the ETH version $eth/64$ in this work. After the nodes agree to use ETH, they need to exchange $Status$ messages. The $Status$ message includes the Total Difficulty (TD) and the hash of their latest block. A node, which has a lower TD after exchanging the $Status$ messages, will start synchronization immediately.

Transactions are propagated with one or more $Transactions$ messages. Nodes utilize the $NewBlock$, and $NewBlockHashes$ messages to propagate a new block. The $NewBlock$ message includes the full block, which is sent to a small set of connected nodes (the square root of the total number of peers). Other peers are sent with a $NewBlockHashes$ message, which contains the hash of the new block. Those peers can request the block body with $GetBlockBodies$ message if they don't receive it from other nodes after a period of time.

4 Methodology

4.1 Latency characterization

In the scope of this research, we consider the IoT-based application of the private Ethereum network. It is generally that some IoT devices may not have enough capacity to conduct the mining process. However, the mining process is indeed essential for the blockchain. Therefore, we come up with a scenario, as in Fig. 2, where a powerful node serves as the mining one for several other nodes. The nodes establish a blockchain network, which has a linear structure. In our work, rather than using the node discovery protocol, we create the connections manually. Node 1, which performs the mining tasks for the pending transactions, generates blocks for the other nodes. Except for Node 1, the other nodes submit transactions and wait for outcomes from the mining node. On creating a TCP connection between two nodes, they exchange existent information in k-bucket and initialize a blockchain connection on the ETH protocol. Transactions and blocks are propagated along with the blockchain connection, hop by hop.

We can present a lifecycle of a transaction following the three steps, as shown in Fig. 3. The lifecycle indicates the duration from the submitted moment to the time of becoming effective. First, transactions are submitted to the txpool, and then disseminated to the mining node. Second, the mining node executes the PoW algorithm and generates blocks. Note that the transactions involved in the mining process are packed into blocks. Third, the blocks are broadcasted and validated to all nodes in the blockchain network.

After the validation, the transactions will finally be efficacious. In this work, we intentionally ignore the period of the mining process and focus on the other two others, namely the propagation of transactions and the propagation of blocks. As indicated in Fig. 3, we define the leftmost process as transaction-oriented latency and the rightmost one as block-oriented latency. The transactions and blocks transmissions are triggered by $Transactions$ and $NewBlock$ messages, respectively. They are followed by several steps implemented in Geth. We analyze the Geth log at the highest verbosity to clarify the workflow of the two processes in the private Ethereum network. The definitions are presented as follows.

4.1.1 Transaction-oriented latency

In Ethereum, the workflow of transmitting a transaction between a node and its peer is shown in Fig. 4. After a transaction is submitted to a node, it is pushed into a queue, waiting to be verified. When the node finishes the verification (i.e., at the $Promoted\ queued\ transaction$ point), the transaction is submitted and added into the txpool (i.e., at the $Submitted\ transaction$ point). Afterward, the node broadcasts the new transaction to its peer in $Transactions$ message. The peer node first queues the received transaction at the $Pooled\ new\ future\ transaction$ point. It then verifies the transaction after the $Promoted\ queued\ transaction$ point. Subsequently, the transaction is added to the txpool of this peer. This peer repeats the process to propagate the transaction to the next peer. We define the transaction-oriented latency as the interval between the submission moment in one node and the promotion time in its peer. The transaction-oriented describes the time consumption for a transaction to be propagated in different hops. With a lower value of the transaction-oriented latency, a submitted transaction can reach the entire network quicker.

4.1.2 Block-oriented latency

The mining node selects transactions from txpool and packs them into a block, which is then propagated to its peer. Figure 5 shows the workflow of propagating a newly mined block, which begins at the $Mined\ potential\ block$ point. Nodes in the network use the $NewBlock$ message to send the full block to its peers at the $Propagated\ block$ point. After the peer receives the block, the block is pushed into a queue at the $Queued\ propagated\ block$ point. The peer imports the block at the $Importing\ propagated\ block$ point then starts to process it. To reach all the nodes as soon as possible, the peer first passes the block to other nodes (i.e., at the $Propagated\ block$ point). At this moment, the peer has already started repeating the block transmission process to the next peer. Then the peer verifies the block

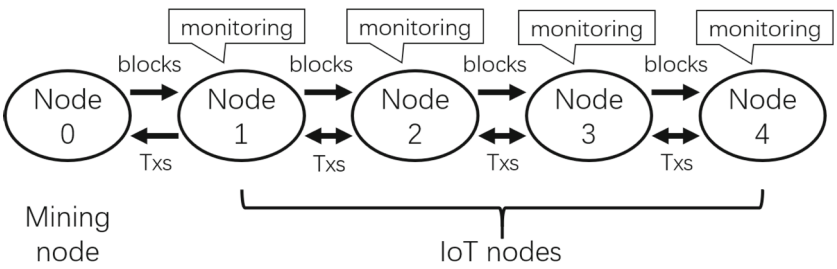


Fig. 2 The private blockchain structure in this work

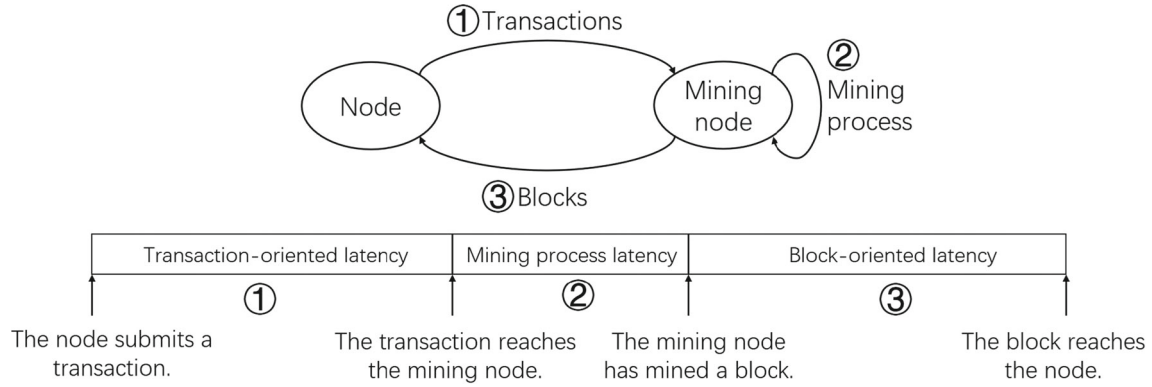


Fig. 3 Lifecycle of transactions in Ethereum blockchain network

and inserts it in its local database at *Inserted block* point. The block finished its lifecycle at the *Imported new chain segment* point. The peer announces the ownership of the block to avoid duplicating transmission. We define the interval from mined a block to the import of the block in one of the peers as block-oriented latency. The block-oriented latency describes the time consumption for a block to be propagated in different hops. A block contains several verified transactions. After the block is propagated and appended to a peer, those transactions are formally accepted and come into effect. Thus, this latency describes how fast a block outstretches the network.

4.2 Monitoring tools

We aim to monitor the performance parameters of the devices running Ethereum implementation. Since the monitoring method is expectedly lightweight, therefore we exploit the Linux internal commands to get the performance values. In the following, we introduce the performance metrics as well as the associated tools.

Disk space usage: Ethereum full nodes have to keep the entire historical transactions, blocks, and smart contract codes to locally regenerate the state tire. That historical information is permanently retained. Thus, Ethereum kept them in a key-value LevelDB[1] to disk space. Hence it is worthy of investigating the disk usage. We use the Linux tool named `df`[2] to get the values of disk space usage.

Memory usage: Ethereum executes transactions and changes the state tire in a specialized EVM,[3] which is kept in the memory space, along with pending transactions and other ephemeral data. Therefore, the memory space relates to the capacity of achieving blockchain functionalities. Moreover, the execution of the JavaScript library also requires disk space and memory space for caching. The memory usage value can be obtained with the `free`[4] tool.

CPU utilization: It has been generally known that the execution of the PoW algorithm needs much more computational power. While the verification of transactions and blocks also requires calculating hash functions. Therefore, besides the miner, we also monitor the non-mining Ethereum nodes over the CPU utilization. We use a bash script to capture the CPU time statistics from the `/proc/stat` file two times and then calculate the immediate CPU usage.

Network throughput: Ethereum nodes communicate with each other continuously to exchange states, propagate transactions and blocks through its underlying network. Therefore, network throughput plays an important role in understanding the whole blockchain performance. We log and read the accumulated bytes of data transmitted and received from the `/proc/net/dev` file from the Linux system.

The above commands are composed to a file as the monitoring pattern shown in Fig. 6. It is integrated with the cron daemon on each device (i.e., in *crontab*), a time-based Linux build-in scheduler utility, which can execute the monitoring pattern periodically (i.e., every minute). This monitoring pattern has a negligible impact on the running blockchain system comparing with the remote procedure call (RPC) based way as discussed later. Additionally, the script can be extended with other parameters such as I/O speed, load average, and CPU temperature for further information.

5 Evaluation

5.1 Blockchain network environments

First, we build an indoor testbed with a laptop computer and four IoT devices. Each IoT device is a single-board RPI 3b+ that could run tasks as normal computers. The hardware

[1] https://github.com/ethereum/leveldb

[2] https://linux.die.net/man/1/df

[3] https://ethereum.org/en/developers/docs/evm/

[4] https://linux.die.net/man/1/free

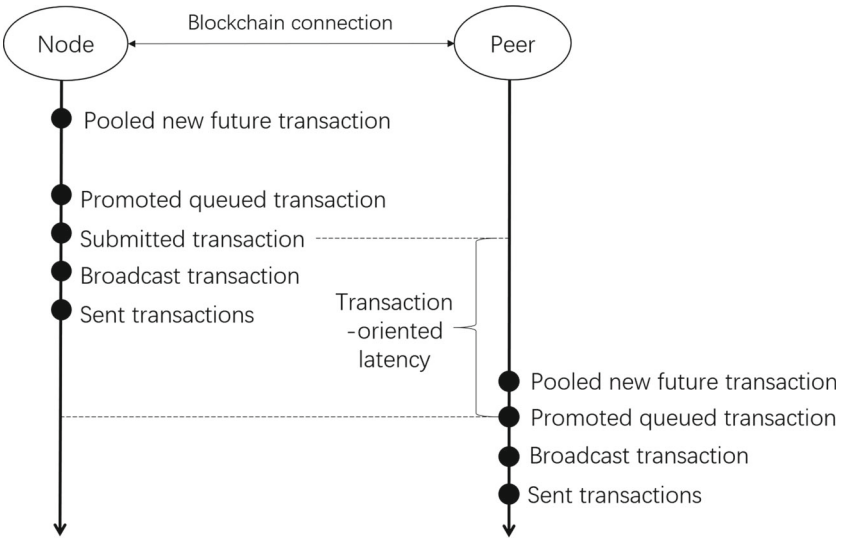


Fig. 4 The workflow of transmitting a new transaction in Geth

and software configurations of the local blockchain nodes are shown in Table 1. Figure 7b shows the physical deployment of the devices. The devices connect to the same TPlink Wi-Fi router to build the underlying network. On top of the underlying network, we set up the Ethereum blockchain network. We necessarily create a custom genesis file to launch the blockchain client in private deployment. Moreover, in the genesis file, we need to set a proper level of $difficulty$, which can ensure the data nodes receive responses in a reasonable time. Another critical parameter is the block gas limit, which allows the blocks contain sufficient transactions. Our private blockchain network has a linear structure that indicates by the arrows in Fig. 7a. In this scenario, the laptop runs as a mining node, which has enough power to mine continuously. On the other hand, the RPI 3b+ serves as a data node, which concentrates on information sharing.

Second, we construct a private Ethereum network on the cloud. As illustrated in Fig. 8c, we select GCP and create three distributed virtual machine (VM) instances. The nodes have different locations, as shown in Fig. 8a. We allocate those nodes with light computational resources to simulate the IoT devices. The configuration of the nodes and their operating systems are in Table 2. In this case, the underlying network of private Ethereum is the Internet. We use the same genesis file as the indoor testbed and form the private blockchain network over the cloud as in Fig. 8b. In this case, the underlying network of private Ethereum is the Internet.

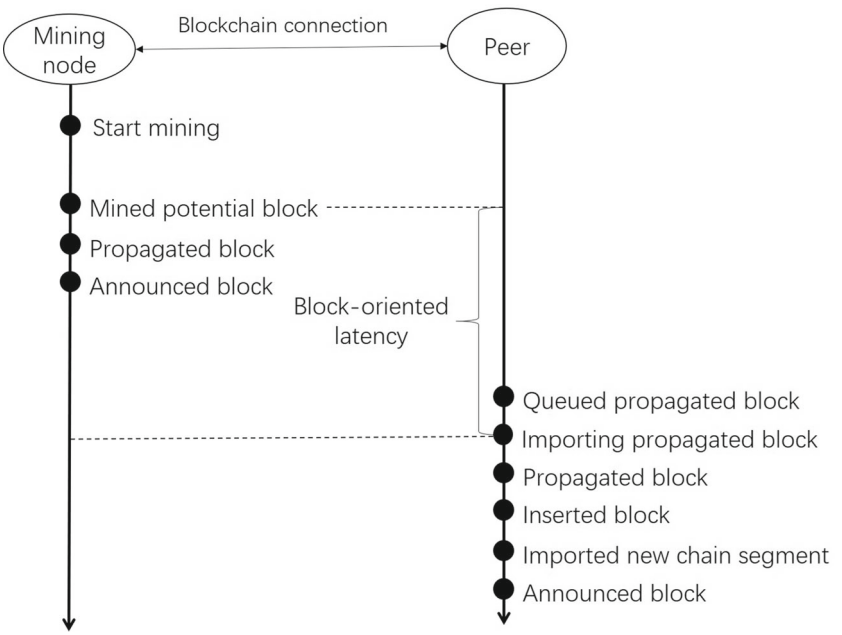


Fig. 5 The workflow of transmitting a new block in Geth

Fig. 6 Screenshot of monitoring pattern

Node 1 and 2 serve as data nodes, while node 3 in the middle serves as the mining node. We use a PC (i.e., local host) with SSH connections to the nodes to control them and collect results.

To run both experiments properly, we have to prepare three important issues as follows.

1) Deploy a smart contract written in Solidity version 0.4.25. The smart contract, which executes the interaction functionalities of reading and writing information from the blockchain. Usually, Ethereum performs a charging module from the sender based on gas consumption and the current gas price. However, the nodes in a private network suppose to share information without any actual charge. In our private deployment, the gas price is set to zero to eliminate the cryptocurrency cost.

2) Synchronize the system time on all nodes. We expect to measure the latency on an accuracy level of milliseconds. Thus, we choose *ntpdate* to synchronize the system time on all devices. The *ntpdate* command sets the local system time by polling the NTP (Network Time Protocol) servers specified to determine the correct time, adjusting the time to microsecond accuracy. We selected a static NTP pool server[5] as the synchronization target to get the highly accurate time.

3) Install the Web3.js library [10] on every data node to generate transactions calling the pre-loaded smart contract automatically. Web3.js is a collection of JavaScript libraries, which provides a complete Ethereum JSON-RPC implementation to interact with a local or remote Ethereum node via HTTP, IPC, or WebSocket. We open an IPC port for each data node and send transactions every fixed period. Also, there are other libraries available, such as Web3.py[6] or Web3J[7]

We record the output of the Geth clients with the highest *verbosity*, which allows Geth produce the most detailed information, including all steps (i.e., in Figs. 4 and 5) with a timestamp. Eventually, we collect the log files together with the *scp* (secure copy) tool and extract the timestamp of the claimed steps for each type of latency using our self-written bash scripts. We calculate the minimum, average, and maximum values of different types of latency in each set of experiments.

5.2 Evaluation in indoor testbed

The latency is evaluated in two scenarios, namely, baseline scenario and realistic scenario. The former scenario includes the transaction-oriented latency of transmitting a single transaction, and the block-oriented latency of transmitting an empty block since Geth will omit the verification process on receiving an empty block. The latter scenario is equipped with additional workloads. For transaction-oriented latency, we set two workloads: sending 10 and 100 transactions per time. For block-oriented latency, we set three workloads: sending blocks containing 1, 10, and 100 transactions.

5.2.1 Baseline scenario

The results in the baseline shows the latency of conveying a minimal amount of information in this local private blockchain network. For the transaction-oriented latency, we measure the time consumption for a transaction to be transferred from each data node to the mining node 100

Table 1 Configuration of the local nodes

RPI 3b+	
Processor	4x Cortex-A53 1.4 GHz
Memory	1 GB
Storage	16 GB MicroSDHC
OS	Ubuntu 18.04 LTS
Geth version	Geth 1.9.10-stable
Thinkpad laptop	
Processor	4x Corei5-7200U 2.5GHz
Memory	8 GB and 2 GB Swap
OS	Ubuntu 18.04 LTS
Geth version	Geth 1.9.10-stable

[5] IP address: 133.243.238.243

[6] https://web3py.readthedocs.io/en/stable/

[7] https://docs.web3j.io/

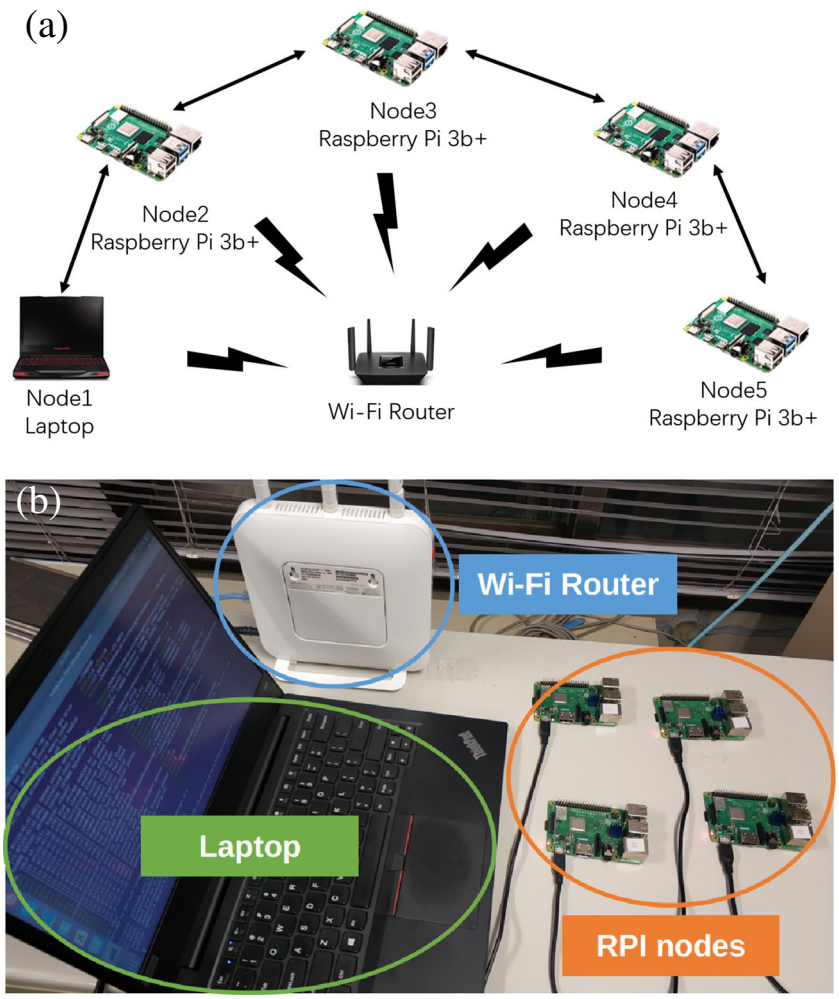


Fig. 7 Deployment of the private blockchain in the indoor testbed

times. Figure 9 shows the minimum, average, and maximum value of the measurement. The latency has a positive relationship with the number of hops. We notice that the average value increases approximately nine milliseconds for each hop. Since blockchain transfers messages via the network connection, we use *Ping* commands to assessed the RTT values between two devices 100 times. The average RTT is 5.047 milliseconds (min: 2.33 ms, max: 21.3 ms), which means averagely a transaction consumes 4 ms to be processed and 5 ms to be transferred to the next node. Moreover, we notice the error bar extends in node 4 and node 5, especially the maximum value. It means the latency tend to diversify and become more significant with more hop.

For the block-oriented latency, we measure the time consumption for an empty block to be transferred from the mining node to each data node 100 times. We show the measurement results in Fig. 10. Again, the latency has a positive relationship with the number of hops. However, the nodes only transfer and validate the block header with no transaction inside a block. There is no need for transaction verification and subsequent state modification. Hence, latency in the baseline scenario can be regarded as a reference to the following realistic scenario

5.2.2 Realistic scenario

The results in the realistic scenario show the latencies with workloads. For the transaction-oriented latency, we measure the time consumption of transmitting 10 and 100 transactions. Figure 11 shows the results of two workloads. Comparing to the baseline, we can see that at each node, processing more transactions spends more time. For each workload, the latency also has an approximately linear increase along with the number of hops. Notice that when we transfer ten times the transactions, the latency is not ten times the previous one. It is because the blockchain receives transactions continuously. They don't wait until a transaction is verified to accept the next one. Moreover, the error bar is bigger when the number of transactions is larger. The reason is related to the RTT. The blockchain network transmits each transaction

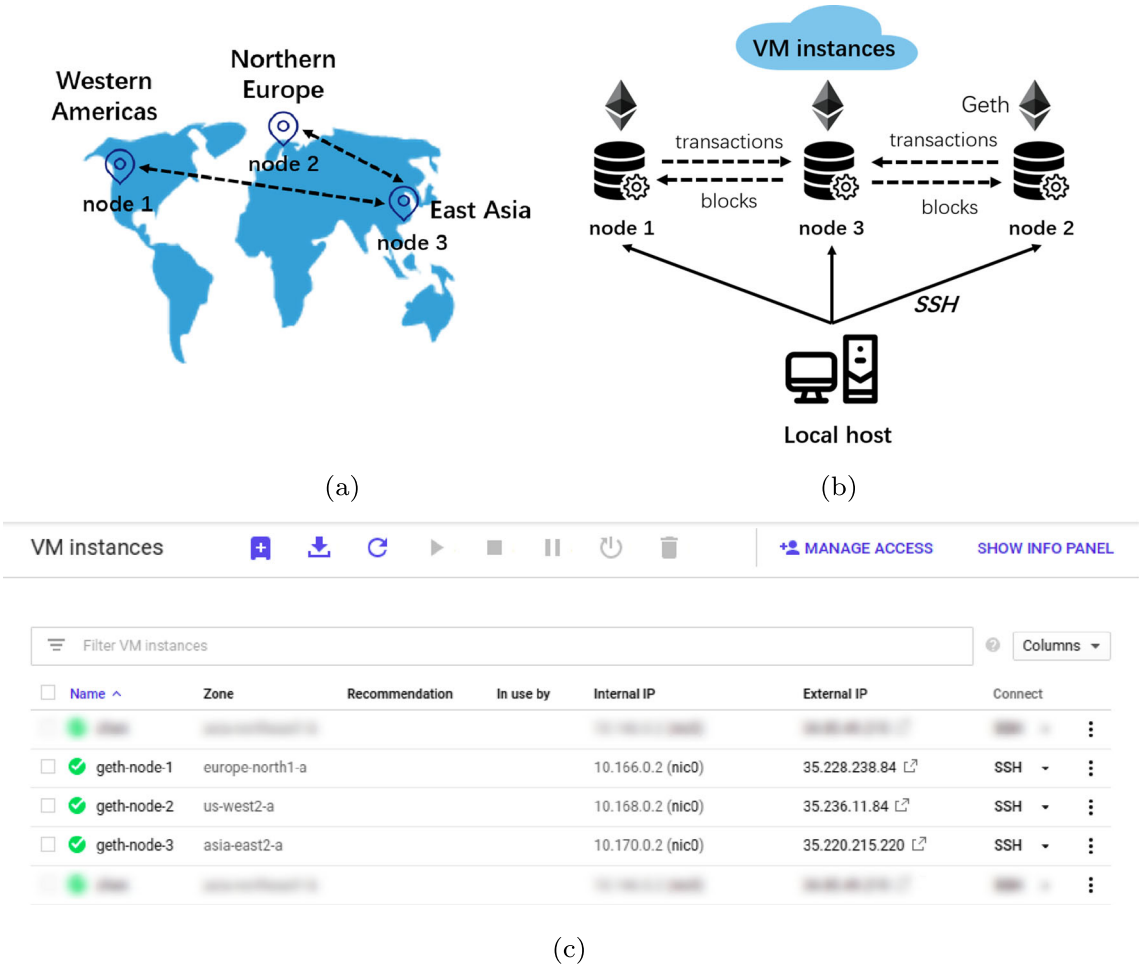


Fig. 8 Deployment of private Ethereum blockchain on the cloud

independently. Therefore, there more transactions are sent, the more RTTs will be added to the latency. The RTT value is affected by the network condition. Thus, the latency is more diverse when transmitting more transactions.

For the block-oriented latency, we measure the time consumption of transmitting a block with 1, 10, and 100 transactions. Figure 12 shows the results of the latency in those three scenarios. Comparing to the baseline, we can observe that at each node, transmitting a block with more transactions spends more time (the relationship of block size and transaction is shown in Fig. 13.) For each workload, the latency value linearly increases following the number of hops. A block with one transaction consumes approximately 50 ms more than an empty block. That is the time for the client to start the verification process. Moreover, the latency in the one transaction case is close to that in the 10 transactions scenario, which means to verify a transaction is

Table 2 Configuration of cloud nodes

Node	CPU	Memory	OS	Location
Node 1	2 vCPUs	2 GB	Ubuntu 18.04	Northen Europe
Node 2	2 vCPUs	2 GB	Ubuntu 18.04	Western America
Node 3	4 vCPUs	4 GB	Ubuntu 18.04	East Asia

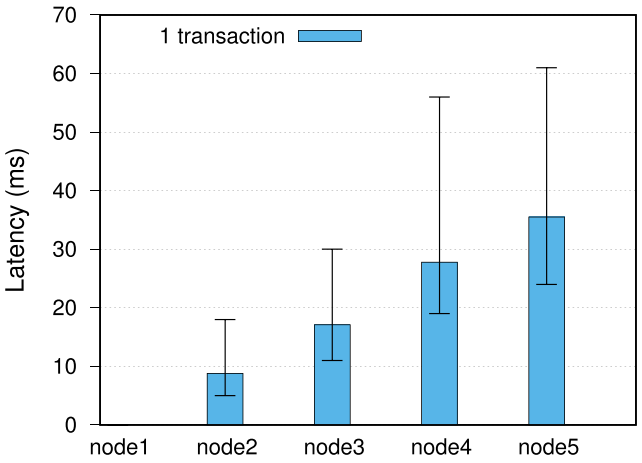


Fig. 9 Transaction-oriented latency in the baseline scenario

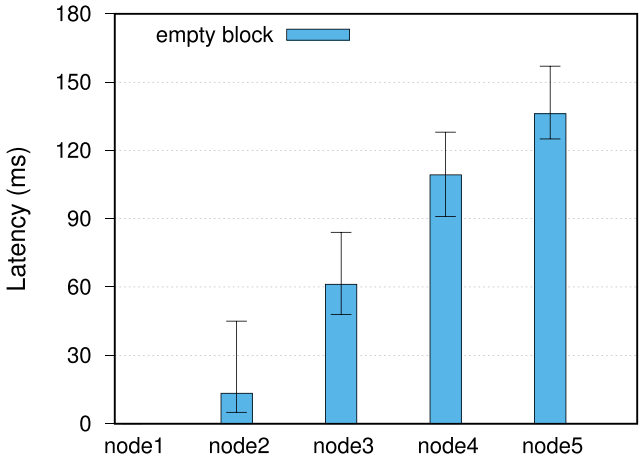


Fig. 10 Block-oriented latency in the baseline scenario

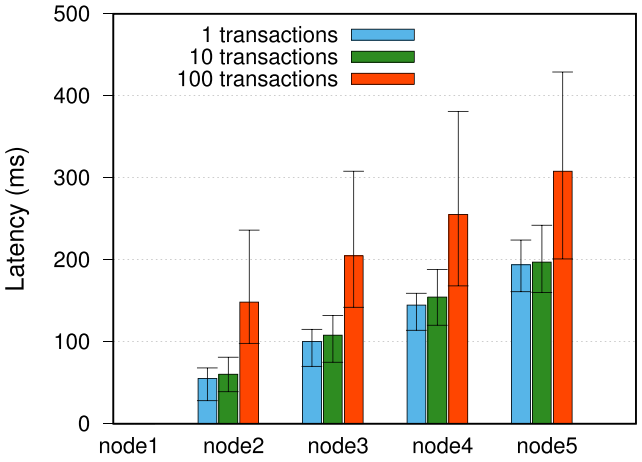


Fig. 12 Block-oriented latency in the realistic scenario

quicker than initialize the verification process. Additionally, we can observe that, for the latency in 100 transaction-case. The first hop from the mining node to the data node consumes averagely 148.32 milliseconds. Other nodes spend roughly 50 milliseconds for each hop. It is because each node propagates blocks before processing them locally. Ethereum holds this mechanism to propagate blocks to the entire network as soon as possible. We also notice that the transaction-oriented latency is lower than the block-oriented latency when transmitting 1 or 10 transactions, while the block-oriented latency is more significant than the transaction-oriented latency when transmitting 100 transactions.

5.2.3 Performance monitoring

We enable the monitoring tools on each local RPI node to inspect resource consumption for two hours, during which those nodes send one transaction per second with the miner engaged. The network involved 647 blocks containing about 28800 transactions over the duration. The performance parameters supporting the network are list as following:

Disk space usage: The increment of used disk space on each local node is shown in Fig. 14a. Throughout the monitoring, the involved transactions, blocks, plus execution results are stored in disk space. The size of a block depends on the number of transactions inside, as Fig. 13 indicates. An empty block takes 537 Bytes space, and each transaction takes about 200 Bytes (various on different types of transactions.) However, the local RPI nodes record the Geth log activities, which primarily occupy the disk space increment. By decreasing the verbosity of the client output, the RPI nodes can significantly save disk space usage.

Memory usage: In our experiments, all the RPI nodes consume 256 KB swap space, which is negligible. Thus, we present the memory variation on each non-mining node in Fig. 14b. It indicates that the non-mining RPI nodes need approximately 650 MB memory. The memory is consumed by Geth and Web3.js library. Geth keeps

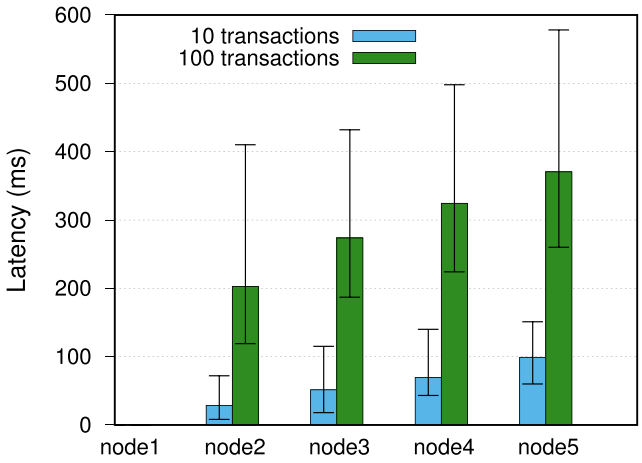


Fig. 11 Transaction-oriented latency in the realistic scenario

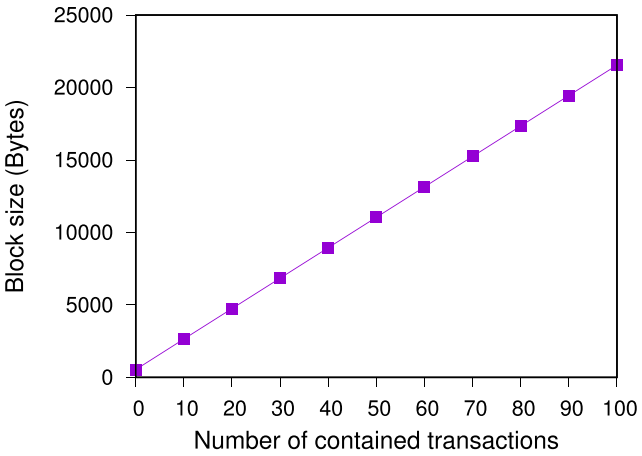


Fig. 13 Relationship between the block size and the number of transactions inside the block

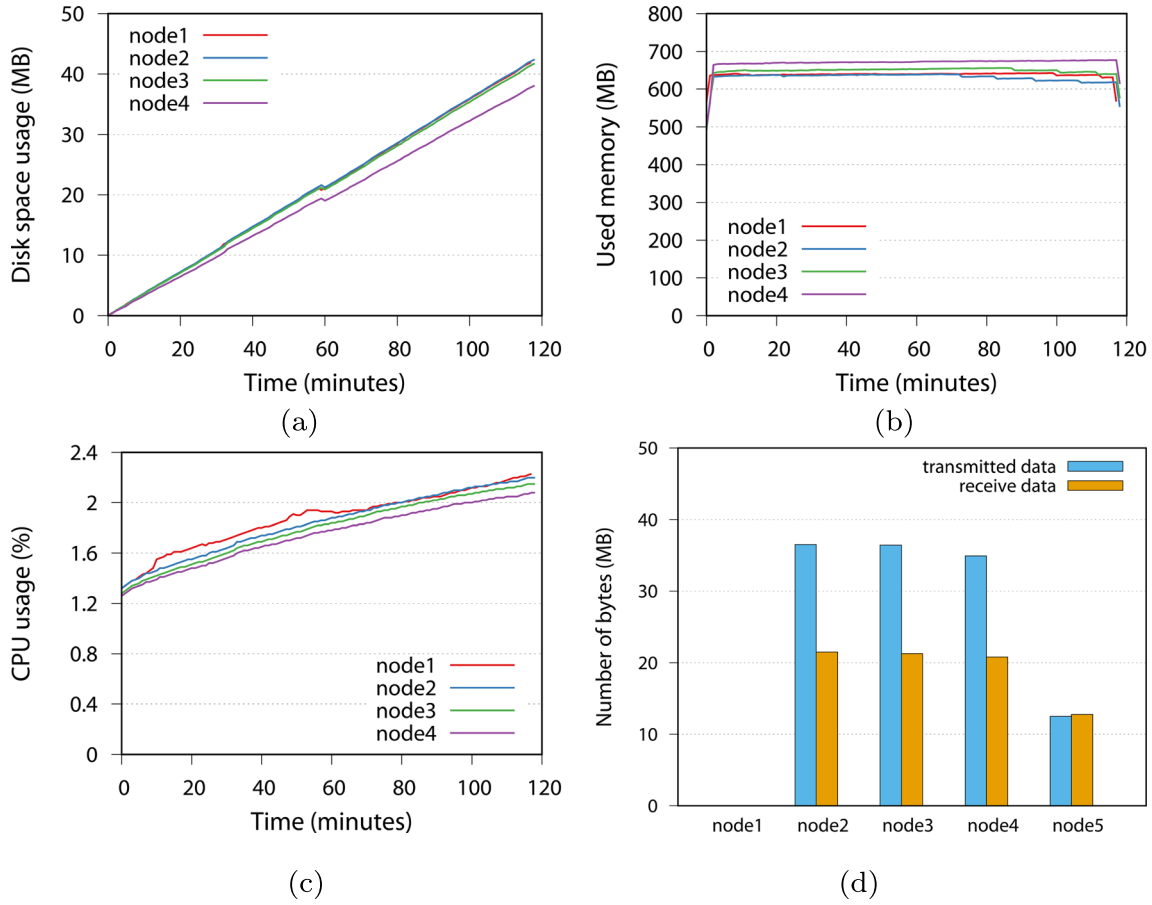


Fig. 14 Performance monitoring results of non-mining node in the indoor testbed

unmined transactions and the state trie in memory space, which is dynamically adjusted by mined transactions' execution results. Notice that transactions are produced on each RPI node locally. Alternatively, we can install the RPC remotely from another device to reduce memory consumption. However, the measurement latency will likely become inaccurate.

CPU utilization: The CPU utilization rate of each node is shown in Fig. 14c. We use `iostat` command to capture the results. The RPI nodes consume approximately 2% of the CPU because the verification of blocks and transactions do not require massive calculation. On the contrary, according to our observation, the mining process consumes almost all of the CPU on the laptop. The CPU utilization is slowly growing because `iostat` command calculates the average CPU time since it was booted. While, from a long-lasting view, the CPU utilization is stable on RPI nodes.

Network throughput: We first monitor the idle situation, in which the nodes are connected without mining or sending transactions. The nodes keep confirming the peer connections and checking the highest block. We then investigate the running situation, in which the miner starts mining, and all non-mining nodes keep submitting transactions one per second. Figure 14d shows the amount of transmitted and received data on the wireless interface of each RPI in the running situation. The first three nodes transmit more data than received because they have two peers to propagate, while node 4 only has one. We calculate the total throughput of both situations in Table 3. The throughput of node 4 in both situations is approximately half of the other nodes.

Table 3 Throughput on RPI nodes (Bytes/s)

	Node 1	Node 2	Node 3	Node 4
Idle	232.63	219.65	222.94	122.78
Running	4837.23	4623.88	4759.25	2042.71

Table 4 Average memory space usage of three states (MB)

States	Idle	Connected	Running
Node 1	198.46	245.48	687.78
Node 2	195.84	244.89	682.54
Node 3	246.83	293.92	378.97

Table 5 Total network throughput (MB/s)

States	Connected	Running
Node 1	0.185	1.929
Node 2	0.191	1.859
Node 3	0.236	3.738

We first monitor the idle situation, in which the nodes are connected without mining or sending transactions. The nodes keep confirming the peer connections and checking the highest block. We then investigate the running situation, in which the miner starts mining, and all non-mining nodes keep submitting transactions one per second. Figure 14d shows the amount of transmitted and received data on the wireless interface of each RPI in the running situation. The first three nodes transmit more data than received because they have two peers to propagate, while node 4 only has one. We calculate the total throughput of both situations in Table 3. The throughput of node 4 in both situations is approximately half of the other nodes (Tables 4 and 5).

5.3 Evaluation in cloud-based network

5.3.1 Latency evaluation

We evaluate the two types of latency of single-hop in the blockchain network deployed on the cloud (i.e., the structure is in Fig. 8b). The difference between this network and the indoor one is that the round trip time (RTT) is distinct in long-distance data transmission. We initially measure the RTT using the *Ping* command. The average RTTs are presented in Table 6. Before the latency measurement, we also synchronize every node's system time to an NTP server for accuracy. Then we estimate transaction-oriented latency and block-oriented latency from node 1 and 2 to node 3.

Transaction − oriented latency: Since Ethereum propagates each transaction separately and continuously, we collect the transaction-oriented latency of 1000 transactions in this evaluation. The latency values on each node are shown in Fig. 15. Averagely, Node 1 needs 149.53 ms to send a transaction to node 3 on average, while node 2 needs 71.23 ms. Compared with the local network, the latency is more stable, taking advantage of robust network infrastructure over the cloud platform. Moreover, the average latency value is almost half of the RTT.

Table 6 Average latency to node 3 (ms)

	node 1	node 2
Ping	297.36	139.67
Transaction-oriented latency	149.53	71.23
Block-oriented latency (empty)	151.82	73.17
Block-oriented latency (filled)	181.01	121.57

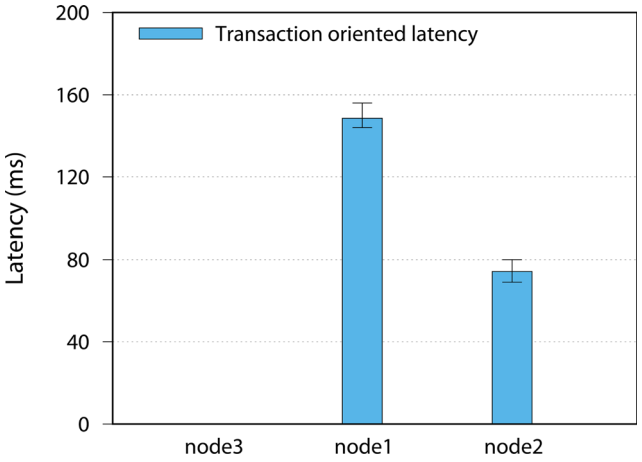


Fig. 15 Transaction-oriented latency in the cloud deployment

Block − oriented latency: In this evaluation, we first measure the block-oriented latency of transmitting an empty block, revealing block-oriented's baseline. Then, we measure the latency in the case of non-empty blocks. The results are presented in Fig. 16. An empty block contains only the block header, which can be loaded in few TCP packets. Thus, the block-oriented latency of empty blocks has a similar performance with transaction-oriented latency in Table 6, near half of the RTT. With transactions filled to the block body, a block's size varies from 1000 to 10000 bytes. The block-oriented latency becomes larger.

5.3.2 Performance monitoring

Disk space usage: During the mining process, the blocks are periodically produced (i.e., about 10 seconds per block). The number of blocks and the corresponding increment of disk space usage on each node is shown in Fig. 17a. Geth

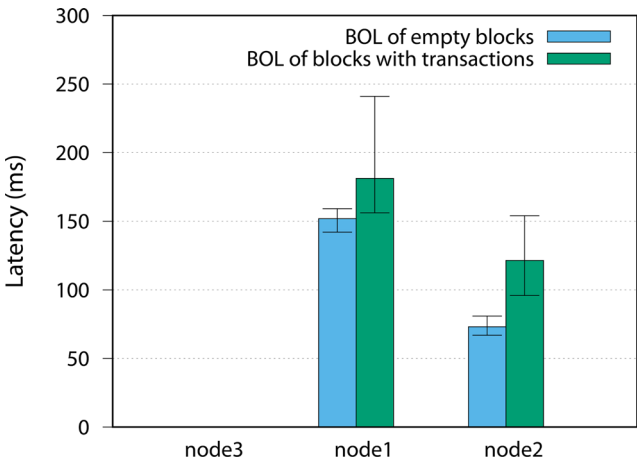


Fig. 16 Block-oriented latency in the cloud deployment

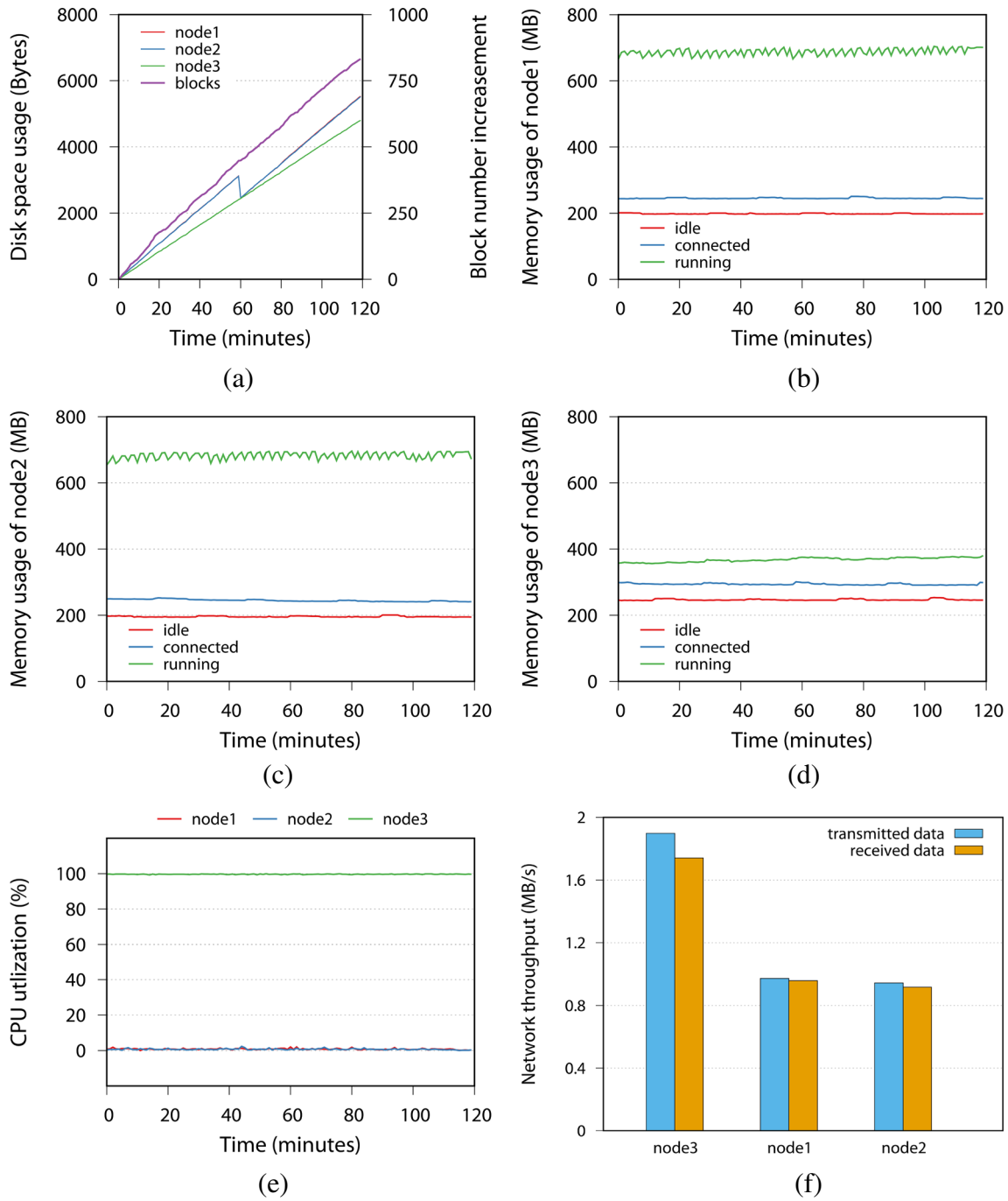


Fig. 17 Performance monitoring results in the cloud-based network

uses its specialized key-value database [12] to store account address and its mapped value. It also uses Merkle Patricia Trie (MPT) to store sequenced transactions. Because we keep calling the same smart contract without introducing new accounts or codes, the subsequent transactions come into effect by modifying the previous value. Therefore, disk space usage is mostly lower than the accumulated block size (i.e., 1152 bytes per block on average), revealing the merit of private Ethereum blockchain with predefined functionalities. Moreover, node 3 keeps a steady growth, while node 1 and node 2 have the same tendency with a point of inflection related to the database behavior.

Memory usage: In the experiment, we did not observe the devices consuming swap space. Thus, we present the memory variation, which is the RAM space usage, on each node in Fig. 17b, c and d with three states. We boot up our devices without doing anything in *idle* state, representing the baseline memory usage. Then, we start the

Geth clients and connect without sending transactions or mining. Nodes only maintain the connection with necessary communication, which consumes minimal memory space in *connected* state. Then we start Web3.js and the mining work, which turn to the *running* state. The average value is shown in Table 4. We can notice that the used memory space increment from *idle* to *connected* is roughly 50 MB, represents the memory space a node run statically consumes. Afterward, the mining process occupies 85.05 MB, while the Web3.js occupies 439.98 MB on average. The results indicate the current popular interaction method, Web3.js, consumes appreciable memory space.

CPU utilization: Figure 17e shows the immediate CPU utilization of each node, which is the percent of CPU time used by *user*. The mining node consumes approximately 99.67% of the CPU time because solving Pow problems requires significant calculation power. On the contrary, the non-mining nodes consume 0.84% and 0.79% of the CPU time, respectively, which also need to verify received transactions and blocks. The results prove that the Ethereum PoW problem is hard-to-solve and easy-to-verify.

Network throughput: Considering the network throughput, we also evaluated the *connected* and *running* states. The total network throughput is concluded in Table 5. According to the ETH protocol, nodes continuously confirm the connection and share the latest block in the *connected* state. Those data are also emitted when transferring transactions or blocks. Therefore, the data transmitted in the *running* state is mostly caused by transactions and blocks. Moreover, due to node 3 having two times the number of peers than node 1 and 2, the total throughput also approximately doubled. While we look detailed into transmitted and received data in Fig. 17f, node 1 and 2 send transactions at the same frequency. They have a nearly equal transmission and receiving. However, node 3 receives a little bit lower than transmitted, for it rejects some redundant transmission.

6 Conclusion and future work

The private blockchain with many advanced features is the potential technology for many IoT applications. However, in general, the blockchain deployment is considerably heavy to the IoT devices. Hence, it is crucial to understand the blockchain performance in IoT scenarios. This paper introduces an experimental study that reveals various performance parameters of private Ethereum networks in IoT scenarios. Our first focus is the latency performance. We have clarified two types of latency (i.e., transaction-oriented and block-oriented latency) and presented the measurement methodology. The method is effective to get the latencies in the indoor testbed and the cloud-based blockchain network. The results give us an observation of the latencies' relationship with the number of hops. Additionally, we implement lightweight monitoring tools, which use Linux internal commands. We use the tools to get various performance parameters of the nodes running the Ethereum implementation.

In the future, first, we plan to extend our work by considering complex structures instead of linear ones in this work. The complex structure will likely appear in real applications, where the node discovery protocol randomly forms the blockchain network topology. Each node, which has a different number of peers, will propagate full blocks to a portion of their peers while transmitting block header to the remaining peers. The path of block propagation becomes uncertain, and the latency also diverse in this circumstance. Second, we are going to investigate the latency performance of executing more complex smart contracts. In a real deployment, we need to quantify the impact of the various execution's impacts on the latency. Finally, we will extend the monitoring system to include more blockchain, system-related parameters.

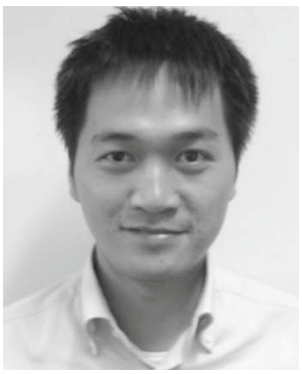
Acknowledgments This work was funded in part by Vingroup Joint Stock Company (Vingroup JSC), Vingroup and supported by Vingroup Innovation Foundation (VINIF) under project code VINIF.2020.DA09 in part by JSPS KAKENHI Grant Number 19K20251, 20H04174. Additionally, Kien Nguyen is supported by the Leading Initiative for Excellent Young Researchers (LEADER) program from MEXT, Japan.

References

1. Ali D, KS S, Raja J (2016) Blockchain in internet of things: challenges and solutions. arXiv:160805187
2. Ali D, KS S, Raja J, Praveen G (2017) Blockchain for iot security and privacy: the case study of a smart home. In: IEEE international conference on pervasive computing and communications workshops. IEEE, pp 618–623
3. Ali O, Jaradat A, Kulakli A, Abuhalimeh A (2021) A comparative study: blockchain technology utilization benefits, challenges and functionalities. IEEE Access 9:12,730–12,749
4. Anh DTT, Ji W, Gang C, Rui L, Chin OB, Lee TK (2017) Blockbench: a framework for analyzing private blockchains. In: Proc. ACM international conference on management of data, pp 1085–1100
5. Atlam HF, Alenezi A, Alassafi MO, Wills G (2018) Blockchain with internet of things: benefits, challenges, and future directions. International Journal of Intelligent Systems and Applications 10(6):40–48
6. Chen X, Nguyen K, Sekiya H (2020) Characterizing latency performance in private blockchain network. In: International conference on mobile networks and management. Springer, pp 238–255
7. Christian D, Roger W (2013) Information propagation in the bitcoin network. In: Proc. IEEE P2P 2013, pp 1–10
8. Ethereum Devp2p https://urldefense.proofpoint.com/v2/url?u=https-3A_github.com_ethereum_devp2p&d=DwIGaQ&c=vh6FgFnduejNhPPD0fl_yRaSfZy8CWbWnIf4XJhSqx8&r=8IEA8RSOvQ9oXtWAG1eT2mTMD_NJ-ANv3H9feUw_xMw&m=cGA8VFmG65hQ_4fSZEBNK42nE5WuDzD36r9U-poJdu0&s=TWhUlf32mvkIK

hH937nE0mpE9Id9IYFCqLPDr9wUbvA&e= (Access date: 2021 Mar.)
9. Ethereum geth client https://urldefense.proofpoint.com/v2/url?u=https-3A_geth.ethereum.org_&d=DwIGaQ&c=vh6FgFnduejNhPPD0fl_yRaSfZy8CWbWnIf4XJhSqx8&r=8IEA8RSOvQ9oXtWAG1eT2mTMD_NJ-ANv3H9feUw_xMw&m=cGA8VFmG65hQ_4fSZEBNK42nE5WuDzD36r9U-poJdu0&s=Ji3dWNInBj-S6VVIi9XshyZoxIuEvdTAmZG-gL5DvgU&e= (Access date: 2021 Mar.)
10. Ethereum javascript api https://urldefense.proofpoint.com/v2/url?u=https-3A_web3js.readthedocs.io_en_v1.3.0_&d=DwIGaQ&c=vh6FgFnduejNhPPD0fl_yRaSfZy8CWbWnIf4XJhSqx8&r=8IEA8RSOvQ9oXtWAG1eT2mTMD_NJ-ANv3H9feUw_xMw&m=cGA8VFmG65hQ_4fSZEBNK42nE5WuDzD36r9U-poJdu0&s=Yg6adc0aKjdcutz0r4sG2yywM_L1EsTNRnCaSNCSnLw&e= (Access date: 2021 Mar.)
11. Gavin W (2014) Ethereum: a secure decentralised generalised transaction ledger. Ethereum project yellow paper 151(2014):1–32
12. Google Leveldb https://urldefense.proofpoint.com/v2/url?u=https-3A_github.com_google_leveldb&d=DwIGaQ&c=vh6FgFnduejNhPPD0fl_yRaSfZy8CWbWnIf4XJhSqx8&r=8IEA8RSOvQ9oXtWAG1eT2mTMD_NJ-ANv3H9feUw_xMw&m=cGA8VFmG65hQ_4fSZEBNK42nE5WuDzD36r9U-poJdu0&s=K2YKFupTwNc1n5At_Xw6yefNWy31oVDEzYE0KVWrHYo&e= (Access date: 2021 Mar.)
13. Heena R, Amr M, Mohsen G (2020) A survey of blockchain enabled cyber-physical systems. Sensors 20(1):282
14. Konstantinos C, Michael D (2016) Blockchains and smart contracts for the internet of things. IEEE Access 4:2292–2303
15. Kshetri N (2017) Can blockchain strengthen the internet of things? IT Professional 19(4):68–72
16. Kuang LS, Yue L, Yen CS, Xiwei X, Qinghua L, Liming Z, Huansheng N (2019) Analysis of blockchain solutions for iot: a systematic literature review. IEEE Access 7:58,822–58,835
17. Kyun KS, Zane M, Siddharth M, Joshua M, Andrew M, Michael B (2018) Measuring ethereum network peers. In: Proceedings of the internet measurement conference 2018, pp 91–104
18. Liu Q, Yu L, Jia C (2020) Mover: stabilize decentralized finance system with practical risk management. In: Proc. conference on blockchain research & applications for innovative networks and services (BRAINS). IEEE, pp 55–56
19. Maymounkov P, Mazieres D (2002) Kademlia: a peer-to-peer information system based on the xor metric. In: Proc. springer international workshop on peer-to-peer systems, pp 3–65
20. Nandar AY, Thitinan T (2017) Review of ethereum: smart home case study. In: Proc. 2nd IEEE international conference on information technology (INCIT), pp 1–4
21. Novo O (2018) Blockchain meets iot: an architecture for scalable access management in iot. IEEE Internet of Things Journal 5(2):1184–1195
22. Quanqing X, Chao J, Mohamed RMFB, Bharadwaj V, Mi AKM (2018) Blockchain-based decentralized content trust for docker images. Multimedia Tools and Applications 77(14):18,223–18,248
23. Quanqing X, Zhaozheng H, Zengxiang L, Mingzhong X (2018) Building an ethereum-based decentralized smart home system. In: Proc. IEEE 24th international conference on parallel and distributed systems (ICPADS), pp 1004–1009
24. Seyoung H, Sangrae C, Soohyung K (2017) Managing iot devices using blockchain platform. In: Proc. IEEE 19th international conference on advanced communication technology, pp 464–467
25. Shi N, Tan L, Li W, Qi X, Yu K (2020) A blockchain-empowered aaa scheme in the large-scale hetnet. Digital Communications and Networks
26. Suporn P, Chaiyaphum S, Thajchayapong S (2017) Performance analysis of private blockchain platforms in varying workloads. In: Proc. 26th IEEE international conference on computer communication and networks (ICCCN), pp 1–6
27. Wüst K, Gervais A (2018) Do you need a blockchain? In: Proc. IEEE Crypto valley conference on blockchain technology (CVCBT), pp 45–54
28. Yu K, Tan L, Shang X, Huang J, Srivastava G, Chatterjee P (2020) Efficient and privacy-preserving medical research support platform against covid-19: a blockchain-based approach. Consumer Electronics Magazine
29. Yu KP, Tan L, Aloqaily M, Yang H, Jararweh Y (2021) Blockchain-enhanced data sharing with traceable and direct revocation in iiot. Transactions on Industrial Informatics
30. Zheng P, Zibin Z, Xiapu L, Xiangping C, Xuanzhe L (2018) A detailed and real-time performance monitoring framework for blockchain systems. In: Proc. IEEE/ACM ICSE-SEIP, pp 134–143
31. Zibin Z, Shaoan X, Hongning D, Xiangping C, Huaimin W (2017) An overview of blockchain technology: architecture, consensus, and future trends. In: Proc. IEEE international congress on big data (BigData congress), pp 557–564

Xuan Chen received the B.S. degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2018. He is currently pursuing the M.S. degree in graduate school of science and engineering from Chiba University, Chiba, Japan. His current research interests include the blockchain as a service and the underlying P2P networks.



Kien Nguyen received the B.E. degree in electronics and telecommunication from the Hanoi University of Science and Technology (HUST), Vietnam, in 2004, and the Ph.D. degree in informatics from the Graduate University for Advanced Studies, Japan, in 2012. He is currently an Assistant Professor with the Graduate School of Science and Engineering, Chiba University. His research covers a wide range of topics, including the Internet, the Internet of Things technologies, wired and wireless communication. He has published more than 100 publications in peer-reviewed journals and conferences and three patents. He is a senior member of IEEE and a member of IEICE. He also involves in IETF activities.

Hiroo Sekiya was born in Tokyo, Japan, in July 1973. He received the B.E., M.E., and Ph.D. degrees in electrical engineering from Keio University, Yokohama, Japan, in 1996, 1998, and 2001, respectively. Since April 2001, he has been with Chiba University, Chiba, Japan, where he is currently a Professor with the Graduate School of Science and Engineering. His research interests include high-frequency high-efficiency tuned power amplifiers, resonant dc/dc power converters, dc/ac inverters, and digital signal processing for wireless communications. He is a Senior Member of the Institute of Electronics, Information and Communication Engineers (IEICE), Japan, and a member of Institute of Electronics, Information Processing Society of Japan (IPSJ) and the Research Institute of Signal Processing (RISP), Japan.