Characterizing Latency Performance in Private Blockchain Network

Xuan Chen, Kien Nguyen(✉), and Hiroo Sekiya

Graduate School of Science and Engineering,
Chiba University, 1-33, Yayoi-cho, Inage-ku, Chiba-shi, Chiba, Japan
{chenxuan,nguyen}@chiba-u.jp, sekiya@faculty.chiba-u.jp

Abstract. There has recently been an increasing number of blockchain applications in different realms. Among the popular blockchain technologies, Ethereum is an emerging platform featuring smart contracts with the public Ethereum associated to the Ether currency. Besides, the private Ethereum has been gaining interest due to its applicability to the Internet of Things. An Ethereum blockchain network includes distributed records that are immutable and transparent through replicating among network nodes. Ethereum manages information in blocks that are submitted to the chain as transactions. This paper aims to characterize latency performance in the private Ethereum blockchain network. Initially, we clarify two perspectives of latency according to the lifecycle of transactions (transaction-oriented and block-oriented latency). We then construct a real private blockchain network with a laptop and Raspberry Pi 3b+ for the latency measurement. We write and deploy a smart contract to read and write data to the blockchain and measure the latencies in a baseline and realistic scenario. The experiment results reveal the latencies-hop correlation, as well as the latencies' relation in different workloads. Moreover, the blockchain network spends averagely 63.92 ms (except the mining time) to take one transaction into effect in one hop.

Keywords: Private blockchain · Ethereum · Latency · Transaction-oriented · Block-oriented

1 Introduction

Blockchain technology has been gaining popularity with the applications in many realms, including finance [1], healthcare [2], and the Internet of Things (IoT) [3]. 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. Also, the records are immutable because they are reserved eternally under 51% rule [4]. Thus, using blockchain, we can build a network that allows nodes to share information without trusting each other. All information submitted to a blockchain is formed as transactions. The mining nodes, which hold a full copy of the blockchain, can verify transactions and generate new blocks through the proof-of-work (PoW) consensus protocol. The blockchain networks

S. W. Loke et al. (Eds.): MONAMI 2020, LNICST 338, pp. 238–255, 2020.
https://doi.org/10.1007/978-3-030-64002-6_16

are divided into two categories: public and private. The former is known as a "permissionless" network. Anyone can join and leave a public blockchain network unrestrictedly. On the contrary, the latter is a "permissioned" network. Nodes need to be permitted by an administrator to join a private blockchain network [5].

The open-source Ethereum [6] is one of the most popular blockchain platforms. The public Ethereum network (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 network with a self-configured genesis file. Ethereum features smart contracts that are self-executed computer programs without an external trusted authority. Ethereum executes the smart contracts in the Ethereum Virtual Machine (EVM) [8], in which the operations are activated by transactions. Once a node receives a new block, it verifies and runs functions triggered by the transaction. 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 sharing information among trustless IoT nodes. Those characteristics significantly improve the IoT devices' cooperation [9]. Besides, the blockchain can extend the scalability of the IoT system with the distribution of records. It hence avoids a single point failure due to decentralized nature. One of the promising IoT applications for private Ethereum is in smart-home scenarios [10,11], which allows home appliances to store, share or modify states cooperatively. Therefore, it is crucial to understand the Ethereum performance for those scenarios.

In this research, we characterize the latency of information propagation in a private blockchain-based IoT scenario. We clarify latencies in two propagation steps as the transaction-oriented and block-oriented latency while ignoring the mining process time. We then investigate those latency values in a testbed. The testbed includes a laptop and four Raspberry Pi 3b+ aiming to mimic a home-based IoT application. The devices 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. The measurement results show that the latency increases proportionally to the hop number in the 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 averagely 63.92, 117.39, 172.38, and 229.21 ms to propagate a single transaction effectively in one, two, three, four hops, respectively.

The remainder of the paper is organized as follows. Section 2 presents related works. In Sect. 3, we introduce the background and our methodology. Section 4 describes the experiment setting and evaluation results. Finally, Sect. 5 concludes the paper and introduces the future works.

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 [12]. The model may have issues, for example, when dealing with scalability [13]. Thus, it is essential to transfer a centralized model to a decentralized one [12]. The blockchain technology is attractive as one of the candidates for decentralizing IoT systems. In [14], the author discusses the possibility of blockchain to strengthen IoT. The work in [15] 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 [16]. In [3], 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 [17], 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 [11], 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 [10], 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 Raspberry Pi 3b, and a computer). The Raspberry 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 [18,19], 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.

Recently, there is an evaluation framework for analyzing private blockchains proposed in [20]. The authors divide a blockchain network into four layers. From top to bottom, they are application layer, execution engine layer, data model layer, and consensus layer. The authors use different workloads to evaluate

different layers of a private blockchain. They evaluated latency ("the response time per transaction") in the application layer on three main blockchain platforms: Ethereum, Parity, and Hyperledger Fabric. In terms of latency, Parity has the lowest latency, and Ethereum has the highest. In [21], which is an extension of [20], the authors implemented different workloads with varying numbers of transactions to the application layer. They analyzed the performance of two platforms: Ethereum and Hyperledger Fabric. In both papers, they considered the response latency of transactions in a single node. In this work, we propose two different latency types according to the lifecycle of transactions. Those two latencies describe a full view of information propagation in a private blockchain network.

3 Background and Methodology

3.1 Background

Ethereum blockchain is essentially a transaction-based state machine [8]. 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 [22]. 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 [23] is the official implementation of Ethereum nodes. The nodes form the Ethereum blockchain network following the P2P networking protocols named DEVp2p [24]. Devp2p includes a node discovery protocol and a RLPx transport protocol, which are based on UDP, TCP, respectively (as shown in Fig. 1).

Node Discovery Protocol. Geth implements its node discovery protocol based on a Kademlia-like Distributed Hash Table (DHT) [25] 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 [26]. 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

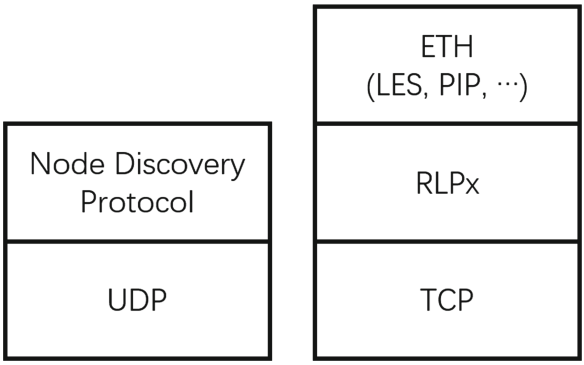

Fig. 1. Ethereum's DEVp2p protocols

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 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.

The RLPx Transport Protocol. The RLPx transport protocol is a TCP-based transport protocol used for communication among Ethereum nodes. Recursive Length Prefix (RLP) [27] 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 2 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.

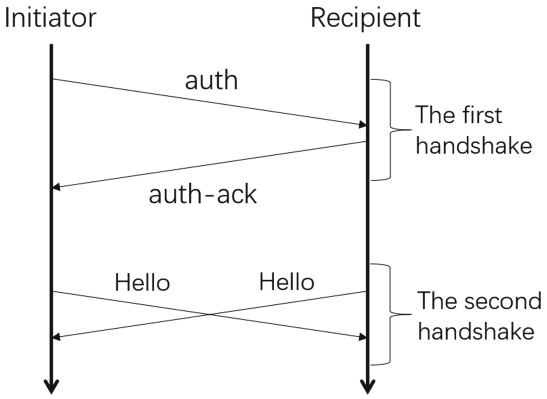

Fig. 2. Two handshakes in RLPx

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.

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 introduce the ETH in detail.

The latest version of ETH is *eth*/64 at the time of writing. 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

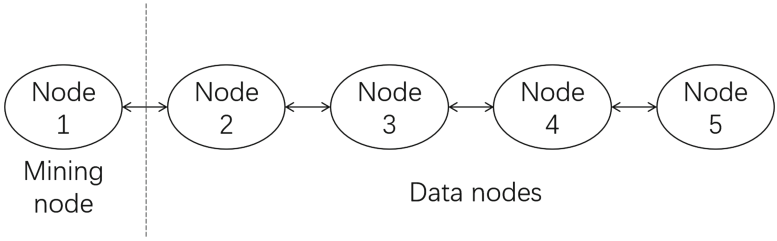

Fig. 3. The private blockchain structure in both experiments

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.

3.2 Methodology

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. 3, 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. 4. 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. 4, 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.

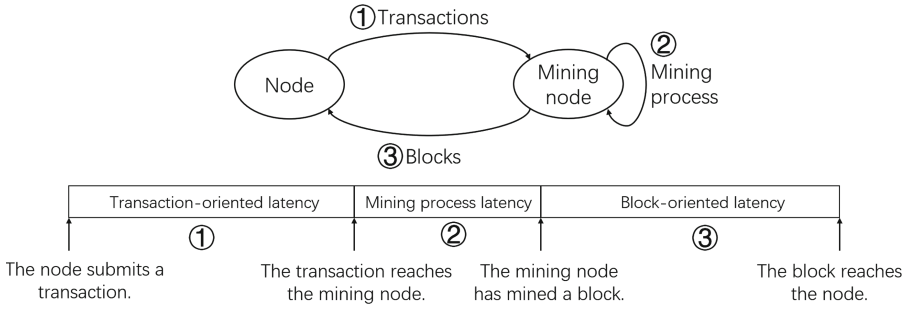

Fig. 4. Lifecycle of transactions in Ethereum blockchain network

Transaction-Oriented Latency. In Ethereum, the workflow of transmitting a transaction between a node and its peer is shown in Fig. 5. 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.

Block-Oriented Latency. The mining node selects transactions from txpool and packs them into a block, which is then propagated to its peer. Figure 6 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 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.

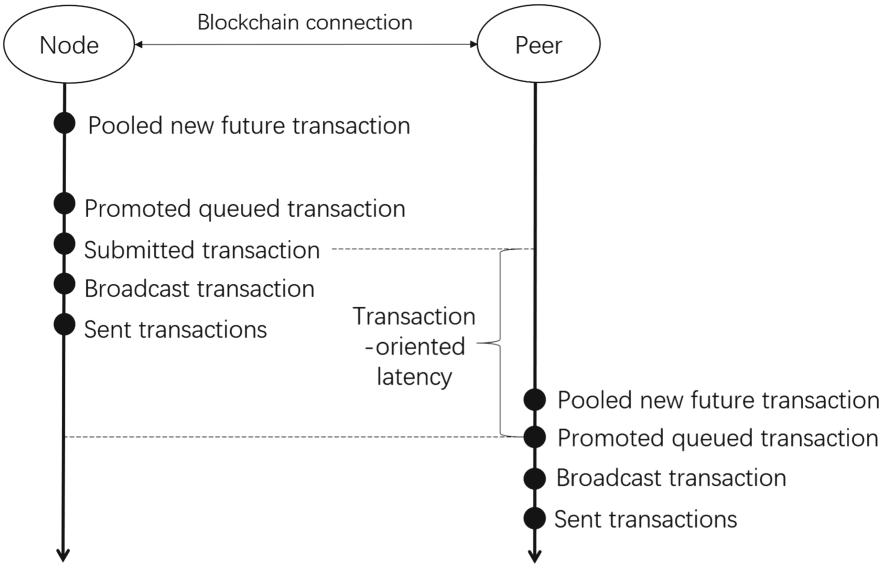

Fig. 5. The workflow of transmitting a new transaction in Geth

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 Evaluation

This section describes our evaluation of the two latencies. We first construct the experiment environment with several specific preparations. We then conduct experiments and report the results.

4.1 Experiment Setup

We build our testbed with a laptop computer and four IoT devices. Each IoT device is a single-board Raspberry Pi 3b+ that could run tasks as normal computers. The hardware configuration is shown in Table 1. Figure 7 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 using the software setting described in Table 2. 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,

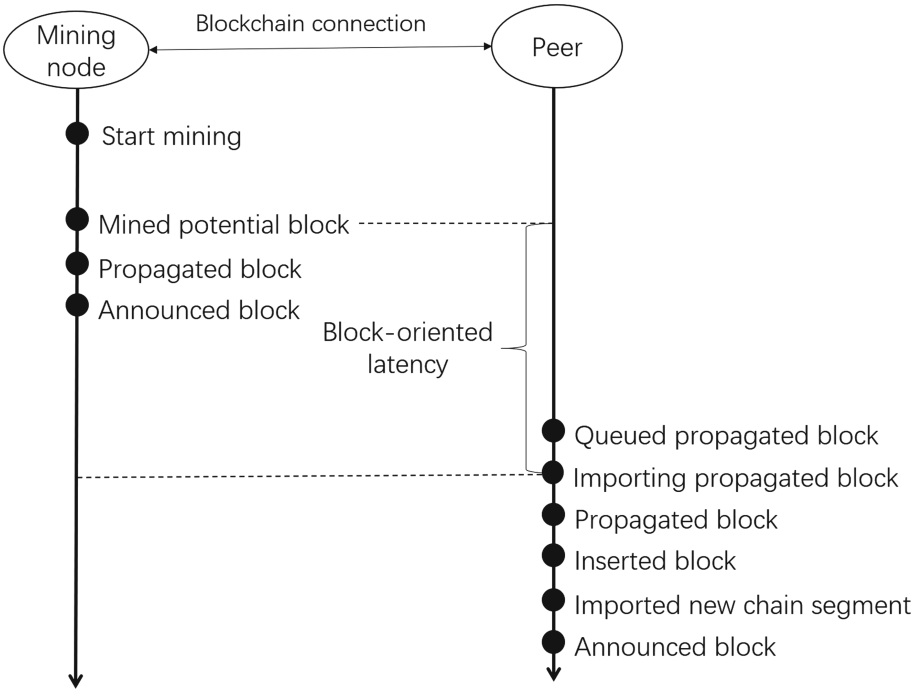

Fig. 6. The workflow of transmitting a new block in Geth

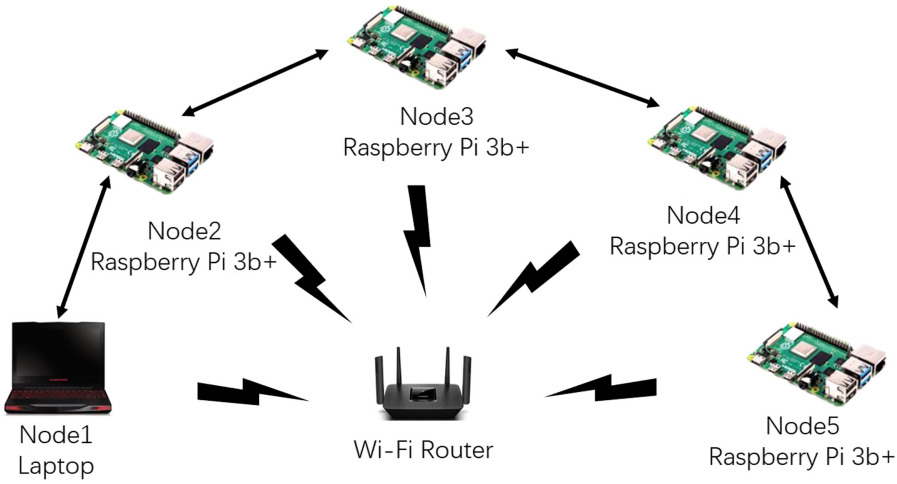

Fig. 7. Our deployment of the private block chain network (The lightning marks represent the underlying network connection, while the solid arrows represent the blockchain connection.)

Table 1. Hardware configuration

Raspberry Pi 3b+	
Processor	4x Cortex-A53 1.4 GHz
Memory	1 GB
Storage	16 GB MicroSDHC
Thinkpad laptop	
Processor	4x Corei5-7200U 2.5 GHz
Memory	8 GB and 2 GB Swap

Table 2. Software configuration

Thinkpad laptop	Ubuntu 16.04 LTS	Geth 1.9.10
Raspberry Pi 3b+	Ubuntu Mate 18.04	Geth 1.9.10

which allows the blocks contain sufficient transactions. Our private blockchain network has a linear structure that indicates by the arrows in Fig. 7. In this scenario, the laptop runs as a mining node, which has enough power to mine continuously. On the other hand, the Raspberry Pi 3b+ serves as a data node, which concentrates on information sharing. To run our experiments, we have to prepare two important issues as follows.

First, we deploy a smart contract written in Solidity [28] version 0.4.25. The smart contract, which simulates the reading and writing of information in the IoT system, has two functions, namely writing a string to the blockchain and reading the current string. Usually, Ethereum will charge the sender some *Ether* based on gas consumption and the gas price of the transaction. However, the nodes in a private network suppose to share information for free. In our private deployment, the gas price is set to zero. That means the nodes can submit transactions to write for free. Moreover, the reading function doesn't consume any gas. Thus, nodes can read the string from the smart contract without paying.

Second, we synchronize the system time on the nodes. We expect to measure the latency on the accuracy level of a millisecond. 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, which can adjust the time to microsecond accuracy. There are many target server can be utilized. We selected the closest available NTP pool server in Japan[1] as the synchronization target to get the highly accurate time. By running *ntpdate jp.pool.ntp.org* command several times, the time error can be adjusted less than one millisecond.

In our measurements, we use the Web3.js library [29] on every data node to send transactions. Those nodes call the writing function in a transaction. We preload a JavaScript file to the mining node. The file enables the start of a mining

[1] IP address: 133.243.238.243 or 133.243.238.163.

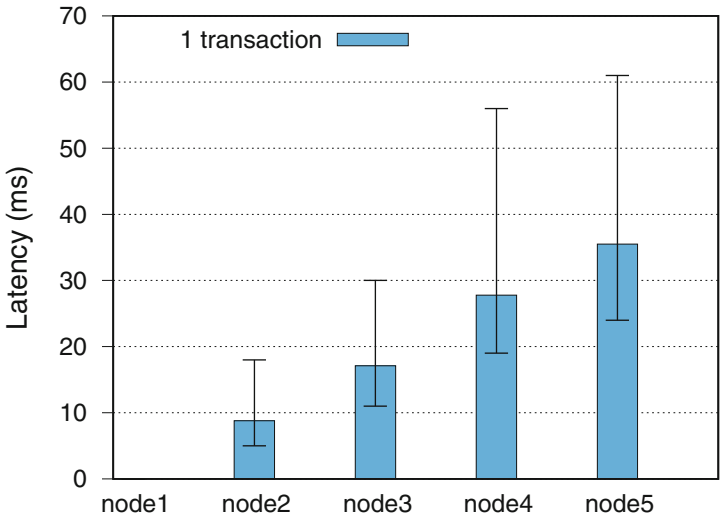

Fig. 8. Transaction-oriented latency in the baseline scenario

process after receiving a transaction. Moreover, it will stop mining after finish process all transactions in the txpool. Since the data nodes send transactions discontinuously in our scenario, this functionality can save computing power and reduce the number of empty blocks.

The *verbosity* is set to five in all nodes. That allows the Geth output has detailed information, including all steps (i.e., in Fig. 5 and Fig. 6) with a timestamp. We record those outputs from the console to a log file and collect them together with the *scp* (secure copy) tool. We then process the timestamp of the claimed steps for each type of latency using our self-written bash scripts. We calculate the minimum, average, and maximum value of different types of latency in each set of experiment.

4.2 Result

We first measure the so-called baseline scenario, which includes the transaction-oriented latency of transmitting a single transaction, and the block-oriented latency of transmitting an empty block. Then we add workloads to simulate latency in a realistic scenario. In transaction-oriented latency, we set two workloads: sending 10 and 100 transactions per time. We send transactions in each data node a hundred times for each workload. All transactions are sent without waiting until the previous one verified, and they arrive at the mining node within different hops. In block-oriented latency, we set three workloads: sending blocks containing 1, 10, and 100 transactions. The mining node collects transactions from data nodes and do the mining process. The latest block is propagated to data nodes within different hops. We send blocks from the mining node a hundred times for each workload. Because Geth will omit the verification process

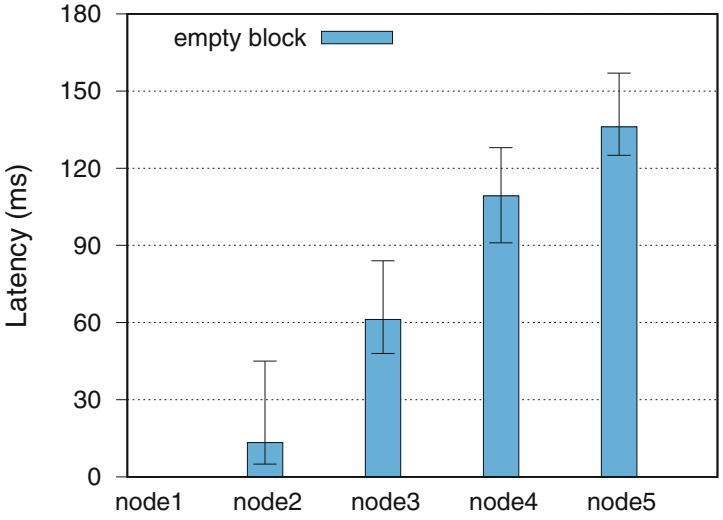

Fig. 9. Block-oriented latency in the baseline scenario

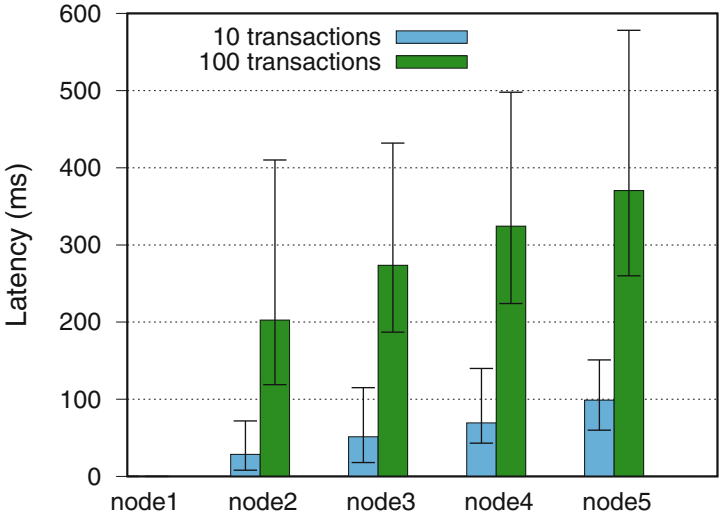

Fig. 10. Transaction-oriented latency in the realistic scenario

when receiving an empty block. We set the workload of transmitting blocks with one transaction to observe the time consumption to start the verification process.

Baseline Scenario. The results in the baseline shows the latency of conveying a minimal amount of information in this private blockchain network. For the transaction-oriented latency, we measure the time consumption for a

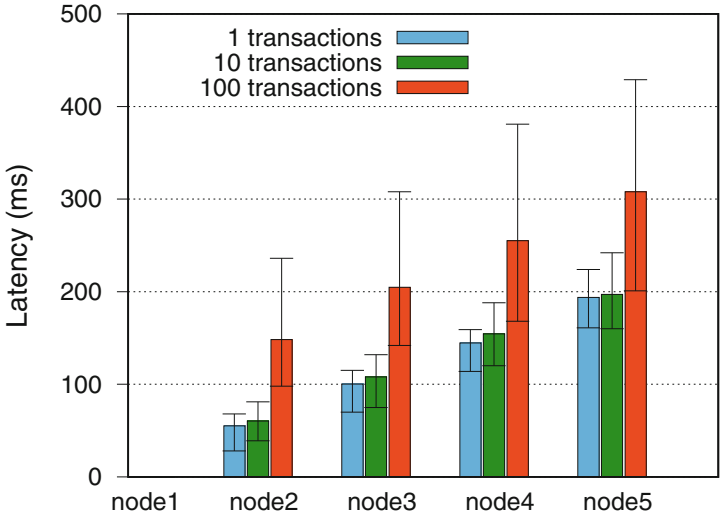

Fig. 11. Block-oriented latency in the realistic scenario

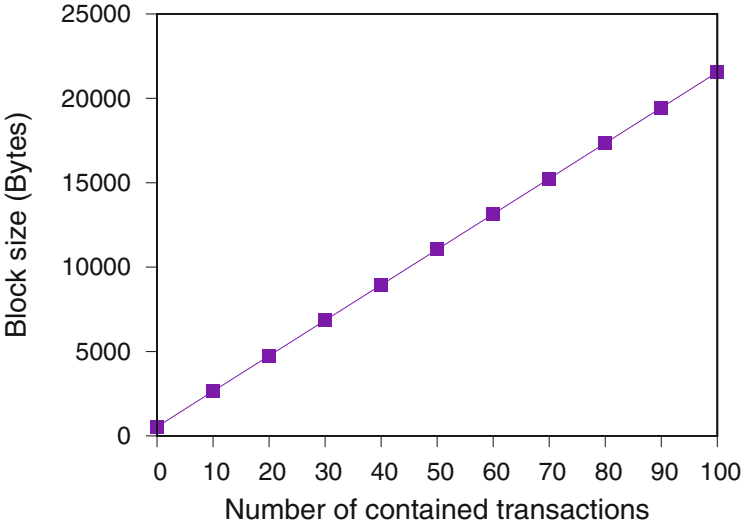

Fig. 12. Relationship between the block size and the number of transactions inside the block

transaction to be transferred from each data node to the mining node 100 times. Figure 8 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 (Round Trip Time) between two devices 100 times. The average RTT is 5.047 ms (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. 9. Again, the latency has a positive relationship with the number of hops. However, with no transaction, there is no need for confirmation process. Hence, the latency reflects transmitting a block header.

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 10 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 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 11 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. 12). 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 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 ms. Other nodes spend roughly 50 ms for each hop. It is because the first hop includes the verification time in the mining node, while the others don't. 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.

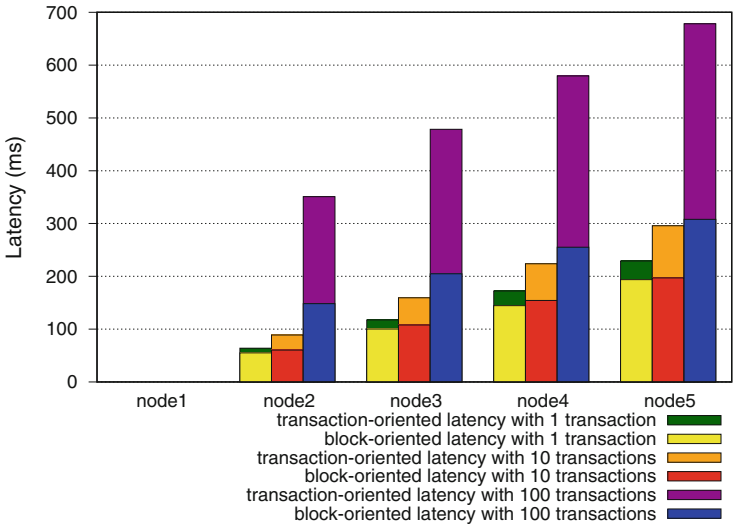

Fig. 13. The total latencies without mining time

Next, we consider the total latency (without the mining process) from submitting transactions to the transactions becomes effective. The total latency is a sum of the transaction-oriented latency and the block-oriented latency. The results are shown in Fig. 13, in which the private Ethereum blockchain consumes averagely 63.92, 117.39, 172.38, and 229.21 ms to propagate one transaction for one, two, three, four hops. Moreover, it also needs 350.83, 478.46, 579.44, and 678.27 ms to propagate 100 transactions for the same hop conditions. We can observe that the transaction-oriented latency is larger than the block-oriented latency when handling 100 transactions.

5 Conclusion and Future Work

The private blockchain network is the potential technology for many IoT applications; hence understanding the blockchain performance is essential. In this paper, we aim to characterize the latency in a real deployment of the private Ethereum. First, we have defined the two latency types (i.e., transaction-oriented and block-oriented latency) and their measurement methodology. We have then measured them in the baseline and realistic scenarios. The results give us an observation of the latencies' relationship. The transaction-oriented latency is lower than the block-oriented latency in the 1 or 10 transaction scenario. While the transaction-oriented latency becomes more significant than the block-oriented latency with the number of transactions increases to 100. Besides, we have considered the total latency, which integrates the two latencies without the consideration of mining latency. The private Ethereum blockchain consumes averagely 63.92, 117.39,

172.38, and 229.21 ms to spread one transaction for one, two, three, four hops, respectively. In the same scenario, the maximum total latency is less than 700 ms.

In the future, we plan to measure the performance of blockchain networks with a more complex structure and more nodes. We also plan to investigate the proof-of-authority (PoA) [30] consensus protocol, which doesn't require mining blocks, for the private Ethereum.

Acknowledgment. This work was supported 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. Singh, S., Singh, N.: Blockchain: future of financial and cyber security. In: Proceedings of 2nd IEEE International Conference on Contemporary Computing and Informatics (IC3I), pp. 463–467 (2016)
2. Zīle, K., Strazdiņa, R.: Blockchain use cases and their feasibility. Appl. Comput. Syst. **23**(1), 12–20 (2018)
3. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the internet of things. IEEE Access **4**, 2292–2303 (2016)
4. Lin, I.C., Liao, T.C.: A survey of blockchain security issues and challenges. IJ Netw. Secur. **19**(5), 653–659 (2017)
5. Wüst, K., Gervais, A.: Do you need a blockchain? In: Proceedings of IEEE Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 45–54 (2018)
6. Ethereum. https://ethereum.org/. Accessed 10 Mar 2020
7. Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: Proceedings of IEEE P2P 2013, pp. 1–10 (2013)
8. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151**(2014), 1–32 (2014)
9. Xu, Q., Jin, C., Rasid, M.F.B.M., Veeravalli, B., Aung, K.M.M.: Blockchain-based decentralized content trust for docker images. Multimedia Tools Appl. **77**(14), 18223–18248 (2018)
10. Xu, Q., He, Z., Li, Z., Xiao, M.: Building an ethereum-based decentralized smart home system. In: Proceedings of IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 1004–1009 (2018)
11. Aung, Y.N., Tantidham, T.: Review of ethereum: smart home case study. In: Proceedings 2nd IEEE International Conference on Information Technology (INCIT), pp. 1–4 (2017)
12. Atlam, H.F., Alenezi, A., Alassafi, M.O., Wills, G.: Blockchain with internet of things: benefits, challenges, and future directions. Int. J. Intell. Syst. Appl. **10**(6), 40–48 (2018)
13. Beck, R., Stenum Czepluch, J., Lollike, N., Malone, S.: Blockchain-the gateway to trust-free cryptographic transactions (2016)
14. Kshetri, N.: Can blockchain strengthen the internet of things? IT Prof. **19**(4), 68–72 (2017)
15. Lo, S.K., Liu, Y., Chia, S.Y., Xu, X., Lu, Q., Zhu, L., Ning, H.: Analysis of blockchain solutions for IoT: a systematic literature review. IEEE Access **7**, 58822–58835 (2019)

16. Rathore, H., Mohamed, A., Guizani, M.: A survey of blockchain enabled cyber-physical systems. Sensors **20**(1), 282 (2020)
17. Huh, S., Cho, S., Kim, S.: Managing IoT devices using blockchain platform. In: Proceedings of IEEE 19th International Conference on Advanced Communication Technology (ICACT), pp. 464–467 (2017)
18. Dorri, A., Kanhere, S.S., Jurdak, R.: Blockchain in internet of things: challenges and solutions. arXiv preprint arXiv:1608.05187 (2016)
19. Dorri, A., Kanhere, S.S., Jurdak, R., Gauravaram, P.: Blockchain for IoT security and privacy: the case study of a smart home. In: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 618–623. IEEE (2017)
20. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: a framework for analyzing private blockchains. In: Proceedings of ACM International Conference on Management of Data, pp. 1085–1100 (2017)
21. Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance analysis of private blockchain platforms in varying workloads. In: Proceedings of 26th IEEE International Conference on Computer Communication and Networks (ICCCN), pp. 1–6 (2017)
22. Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: architecture, consensus, and future trends. In: Proceedings of IEEE International Congress on Big Data (BigData Congress), pp. 557–564 (2017)
23. Go Ethereum (2013). https://geth.ethereum.org/. Accessed 12 Apr 2020
24. Devp2p (2020). https://github.com/ethereum/devp2p. Accessed: 13 Apr 2020
25. Maymounkov, P., Mazières, D.: Kademlia: a peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_5
26. Secp256k1. https://en.bitcoin.it/wiki/Secp256k1. Accessed 09 Apr 2020
27. RLP. https://github.com/ethereum/wiki/wiki/RLP. Accessed 11 Apr 2020
28. Solidity (2020). https://github.com/ethereum/solidity. Accessed 13 Apr 2020
29. Web3.js (2020). https://github.com/ethereum/web3.js/. Accessed 11 Apr 2020
30. Clique Poa Protocol (2020). https://github.com/ethereum/EIPs/issues/225. Accessed 11 Apr 2020