Indonesian Journal of Electronics and Instrumentation Systems (IJEIS)
Vol.14, No.2, October 2024, pp. 209~220
ISSN (print): 2088-3714, ISSN (online): 2460-7681
DOI: https://doi.org/10.22146/ijeis. 99255

Ethereum Blockchain-Based Weather Data Storage Prototype

Eris Sulistiyani*[1], Bambang Nurcahyo Prastowo[2]
[1,2]Gadjah Mada University; Bulaksumur, Caturtunggal, Depok, Sleman Regency, Special Region of Yogyakarta
[1,2]Electronics and Instrumentation, Faculty of Mathematics and Natural Sciences, UGM, Yogyakarta
e-mail: ***[1]erissulis05@gmail.com**, [2]prastowo@ugm.ac.id

Abstract

The application of Ethereum Blockchain within IoT-based weather monitoring systems presents substantial potential for enhancing data security, integrity, transparency, and trust. This study is focused on the design, implementation, and evaluation of Ethereum Blockchain as a robust data security mechanism in an IoT weather monitoring system. The system is configured to monitor environmental parameters, specifically temperature and humidity, using DHT22 sensors, with data securely stored and processed through smart contracts on a locally deployed Ethereum network. The research utilizes the Proof of Authority consensus mechanism, assessing data transmission and storage latency across varying mining intervals. The findings reveal minimal transmission delays, whereas storage delays on the blockchain exhibit variability, influenced by the duration of the mining period. Specifically, longer mining intervals contribute to increased delays in data storage. These results underscore the necessity of optimizing the mining interval to ensure complete and synchronized data storage, thereby enhancing the accuracy and reliability of the weather monitoring system. This study demonstrates the efficacy of Ethereum Blockchain in addressing critical challenges related to data security and integrity within IoT applications, highlighting its potential as a promising solution for secure data management.

Keywords— *Etehereum, Proof of Authority (PoA), IoT, latency*

I. INTRODUCTION

The Internet of Things (IoT) enables various sensors, actuators, and microcontrollers to connect and communicate online through the internet. IoT represents a network of interconnected devices that collect and exchange data regarding usage patterns and the environment in which these devices operate [1]. The deployment of IoT has significantly contributed to technological advancements by facilitating real-time data acquisition for more efficient analysis. IoT applications span a wide range of fields, including agriculture, energy, environmental monitoring, smart homes, healthcare, and transportation. Among these applications, weather monitoring is a key area where IoT is utilized to gather meteorological information and provide weather forecasts for specific regions [2].

Accurate environmental assessment necessitates the monitoring of various weather parameters, such as air temperature, atmospheric pressure, humidity, and precipitation [3]. IoT devices equipped with sensors are employed to collect data on these weather parameters, which

Received August 19th,2024; Revised October 30th, 2024; Accepted October 31th, 2024

is then transmitted to a central database server. However, the transmission and storage of IoT data require robust security and privacy measures, including data integrity, confidentiality, and authentication. Many IoT devices and applications are not originally designed to address these security and privacy concerns, highlighting the need for efficient data transmission and storage mechanisms to safeguard connected devices from hackers and intruders.

Addressing security challenges in IoT systems necessitates careful consideration of hardware that can securely run an operating system, including loading trusted firmware onto smart devices to establish secure communication channels with backend systems. The security of information obtained from IoT devices can be ensured through the implementation of hash algorithms, cryptography, and digital signatures, which provide confidentiality, authentication, and data integrity [4].

Blockchain technology has emerged as a promising solution due to its transparency, immutability, security, and decentralization [5]. Decentralization offers an effective approach for managing the growing complexity of networks, which often require expensive and insecure information and communication infrastructure. Blockchain enables dynamic integration and management of network components [6]. In recent years, blockchain has been increasingly applied in IoT systems to provide a decentralized and secure method for storing and processing data, facilitating data exchange among interconnected IoT devices.

Several studies have explored the application of blockchain in enhancing security and data integrity within IoT systems. Kumar et al. [7] investigated the secure storage of IoT data using a combination of on-chain (Ethereum Blockchain) and off-chain (IPFS and Ethereum) storage, addressing technical challenges in securely storing IoT data. Chen et al. [8] proposed a blockchain-based community safety and security system integrated with IoT devices, aiming to enhance security through the decentralized nature of blockchain and the secure communication capabilities of IoT devices. Chaganti et al. examined a cloud-based agricultural security monitoring system that leverages Ethereum Blockchain to ensure the security of agricultural data by integrating various components such as sensors, AWS cloud, and smart contracts.

The application of Ethereum Blockchain for data security in IoT-based weather monitoring systems holds significant potential. The integration of blockchain with IoT can enhance data security, integrity, transparency, and trust across various applications, including IoT-based weather monitoring systems. This research aims to design, develop, and investigate the application of Ethereum Blockchain as a data security system within an IoT-based weather monitoring framework.

2. METHODS

This research propose the implementation of Ethereum Blockchain as a security system for data storage in an IoT-based weather monitoring system. Ethereum Blockchain has been widely applied theoretically and it can be integrated into IoT systems through smart contracts validated by the blockchain when IoT devices operate. The weather monitoring system collects data from sensors that measure environmental conditions, including temperature and humidity. In this study Ethereum Blockchain is applied to ensure that the data obtained from weather monitoring sensors is securely stored. The smart contract on Ethereum is designed with functions to fetch and store sensor data. The use of hash algorithms ensures security and maintains public transparency by recording transactions on the Ethereum network.

2.1. Software, Hardware, and Runtime Environment

The implementation of the Ethereum Blockchain systems for IoT based weather monitoring requires specific hardware such as laptop/pc, Raspberry Pi 4B, and DHT22 sensor.

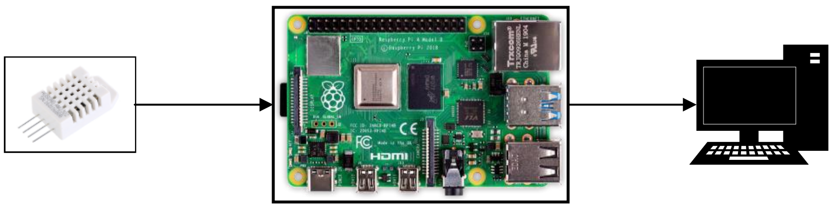
The The software tools used in this research include:

a. Publisher MQTT (mqtt_publisher.py): Program on Raspberry Pi 4B for collecting data from the DHT22 sensor and publishing it to the broker.
b. Subscriber MQTT (mqtt_subscriber.py): Program on a local computer that subscribes to the MQTT broker and receives sensor data.
c. Smart Contract (iot.sol): A Solidity-based smart contract that stores sensor data on the blockchain, manages read and write operations, and enforces conditions for transaction execution.
d. Deploy Program (deploy.py): Used to deploy the smart contract on the Ethereum network.
e. Data Storage Program (store.py): Responsible for storing sensor data on the Ethereum network by interacting with the smart contract.
f. Data Retrieval Program (retrieve.py): Fetches block data and transactions at specific addresses on the Ethereum network.
g. Web Backend (app.py): Flask-based program acting as a server intermediary between the web interface and the blockchain.
h. Web Interface (index.html): Template file used by app.py to display sensor data and data hash values on the web interface.

The runtime environment for this research consists of simulated Ethereum Virtual Machine that running on a local computer connected to the blockchain network, smart contract deployment to implemented on the local Ethereum network, and sensor data collection and transmission using MQTT and stored on the Ethereum network, with data displayed on a local host dashboard.

2.2. Weather Monitoring Design

The weather monitoring system is designed with components including the DHT22 sensor and Raspberry Pi 4B. The Raspberry Pi 4B is connected to a monitor to display sensor data, including temperature, humidity, and rainfall. Data collection is done using a Python program on Raspberry Pi Debian, utilizing the integrated Wi-Fi module on the Raspberry Pi 4B. The design of the IoT-based weather monitoring device is illustrated in Figure 1



.

Figure 1 Weather Monitoring Hardware Design

2.3. System Architecture Design

The implementation of Ethereum Blockchain is designed to secure data in the IoT-based weather monitoring system. The process starts with connecting the DHT22 sensor to Raspberry Pi 4B to collect temperature and humidity data. This data is then sent to the MQTT broker with IP 10.6.6.13 using the mqtt_publisher.py program. On the local computer, the MQTT subscriber receives the data along with a timestamp. Ethereum network runs on the local machine, where the Smart Contract is deployed using the deploy.py program. The sensor data is then stored on the

Ethereum network in real-time using the store.py program, and the data, along with the transaction hash, is displayed on a local web interface using the Flask-based app.py and index.html programs. The overall system architecture is illustrated in Figure 2.

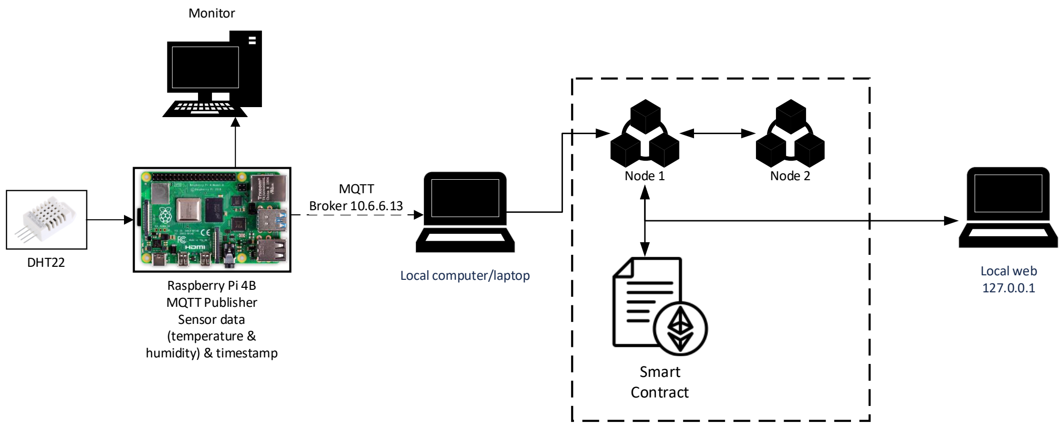


Figure 2 System Design

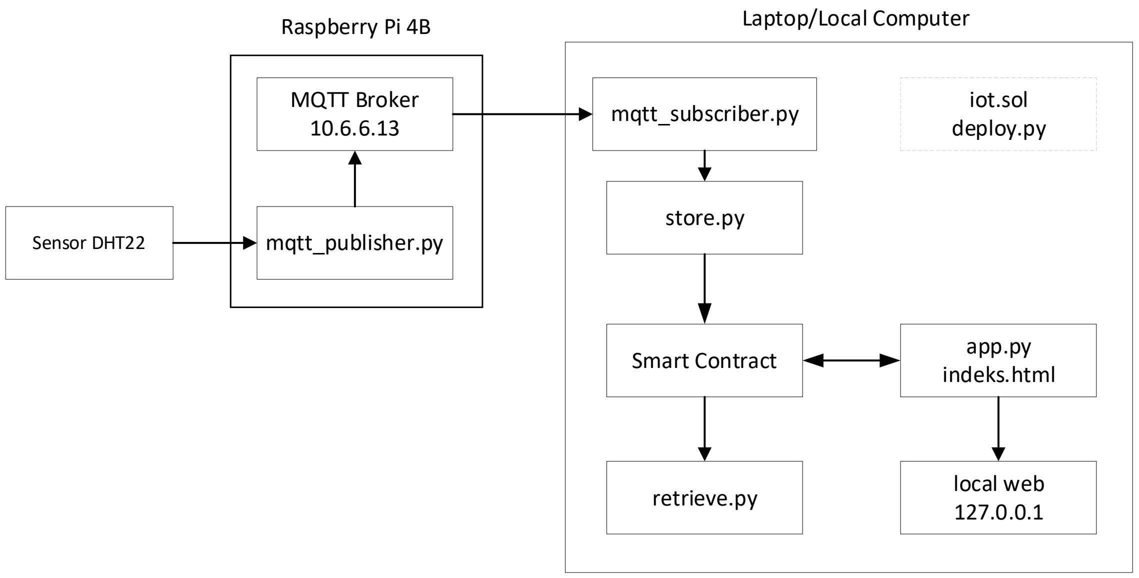


Figure 3 Block Diagram

The entire process begins with mining on node 1, where the Smart Contract is deployed on the Ethereum network. Data collected by the Raspberry Pi 4B from the DHT22 sensor is sent to node 1 via MQTT, where it is stored using the store.py program. Node 1 validates the blockhash, ensuring data security. The stored data is then retrieved and displayed on the local web interface. All of the process is illustrated in Figure 3.

2.4. *Ethereum Blockchain Development*

The Ethereum Blockchain is implemented on a local network, following these steps:

a. Geth Installation: Go Ethereum (Geth) is used to run Ethereum nodes, mine, and manage accounts. The version used is Geth 1.13.15-stable-c5ba367e, supporting Proof of Authority (PoA) consensus.

b. Account Creation: New Ethereum accounts are created using `geth account new`, with specific nodes acting as validator and non-validator nodes.

c. Genesis Block Initialization: The first block in the Ethereum Blockchain is initialized using the genesis.json file with the geth init genesis.json command.
d. Node Deployment: Nodes are deployed using geth –networkid [network id] to identify the private network used in this research.
e. Smart Contract Deployment: The smart contract, written in Solidity, is compiled and deployed on the Ethereum network using ABI and BIN files.

2.5. Ethereum Network Configuration

To create the first block on the Ethereum private blockchain, we need to configure the Genesis Block. The Genesis Block is initialized using the genesis.json file. In the alloc section, initial ether balances are assigned to specific accounts. For Node 1 and Node 2, each account is allocated 50,000 ETH. The extraData field is used to store specific network information or identifiers, and in this Genesis file, the address of Node 1 is added as a validator node.

```
{
  "config": {
    "chainId": 5335,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "muirGlacierBlock": 0,
    "berlinBlock": 0,
    "londonBlock": 0,
    "clique": {
      "period": 5,
      "epoch": 30000
    }
  },
  "difficulty": "1",
  "gasLimit": "9000000000",
  "alloc": {
    "0xd273F775d819136cfE8e85BeC3cde5309b67908C": {
      "balance": "50000000000000000000000"
    },
    "0x3DB911686aD77068Dc5d7cB96eCD86Db5d934501": {
      "balance": "50000000000000000000000"
    }
  },
  "coinbase": "0x0000000000000000000000000000000000000000",
  "timestamp": "0x00",
  "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000d273F775d819
136cfE8e85BeC3cde5309b67908C000000000000000000000000000000000000000000000000
0000000000000"
}
```

The local Ethereum Blockchain network is run on a laptop's localhost. Node 1 and Node 2 are created on the localhost (127.0.0.1), with different ports assigned to each. Node 1 uses port 30304 for peer-to-peer communication, and Node 2 uses port 30305. For HTTP access, Node 1 uses HTTP port 8552, and Node 2 uses HTTP port 8554, which handle HTTP API requests from Geth (Go Ethereum).

In the Ethereum network, an enode (Ethereum Node) serves as a URI (Uniform Resource Identifier) to identify and connect with other nodes. The enode includes information such as the public key, IP address, and port needed to communicate with other nodes in the Ethereum network.

2.6. Smart Contract Implementation

After the Ethereum network is set up, the Smart Contract is developed using Solidity (version 0.8.0). The contract ensures sensor data is stored securely and is immutable. The contract code is provided below:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract IoTData {
  struct Data {
    uint256 timestamp;
    string data;
  }

  Data[] public dataList;

  function storeData(string memory _data) public {
    dataList.push(Data(block.timestamp, _data));
  }

  function retrieveData(uint256 _index) public view returns (uint256, string memory) {
    require(_index < dataList.length, "Index out of bounds");
    Data storage data = dataList[_index];
    return (data.timestamp, data.data);
  }

  function getDataCount() public view returns (uint256) {
    return dataList.length;
  }
}
```

After compilation, ABI and BIN files are generated for contract deployment. Smart contract deployed in the Ethereum network and the sensors data can be stored in the Ethereum network as transactions that stored within recipient as contract address. To deploy the Ethereum Blockchain network, in this case will be create 2 nodes. The following commands is used to run node1 and node2.

```
geth --datadir node1 --syncmode "full" --port 30304 --http --http.addr "127.0.0.1" --http.port 8552 --
http.api "personal,eth,net,web3,txpool,miner,admin" --networkid 5335 --unlock
0xd273F775d819136cfE8e85BeC3cde5309b67908C --allow-insecure-unlock --password pwd.txt --
authrpc.port 8553 --ipcdisable --config config.toml --mine --miner.etherbase
0xd273F775d819136cfE8e85BeC3cde5309b67908C

geth --datadir node2 --syncmode "full" --port 30305 --http --http.addr "127.0.0.1" --http.port 8554 --
http.api "personal,eth,net,web3,txpool,miner,admin" --networkid 5335 --allow-insecure-unlock --
authrpc.port 8555 --ipcdisable --config config.toml
```

2.7. Store and Retrieve Data

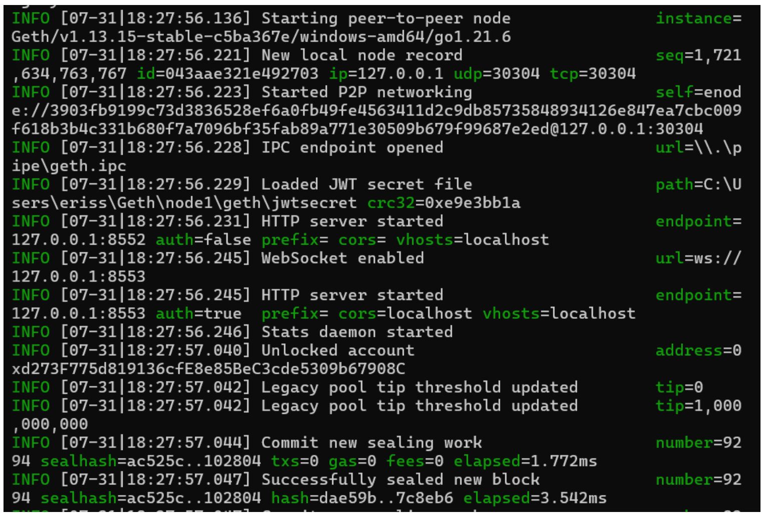
Once the subscriber receives the data, it is stored on the locally running Ethereum Blockchain network using the `store.py` program. This program logs the data, including date, time, temperature, and humidity, as transactions on the Ethereum network, generating a unique transaction hash. Web3 is used to facilitate this process. The stored data can later be retrieved using the `retrieve.py` program, which ensures transparency by displaying all stored data without any modifications. The `store.py` connects to Node 1, while `retrieve.py` connects to Node 2.

3. RESULTS AND DISCUSSION

This study focuses on implementing an Ethereum network to store IoT weather monitoring data, specifically temperature and humidity readings.

3.1. Results of the Ethereum Blockchain Network Implementation

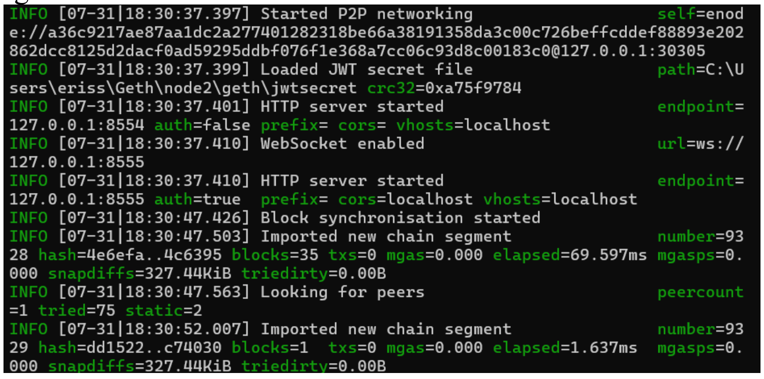
Once the command for Node 1 is executed, Node 1 starts the Ethereum network based on the genesis file and the specific commands written for it. The implementation result for Node 1 is shown in Figure 4.



```
INFO [07-31|18:27:56.136] Starting peer-to-peer node               instance=
Geth/v1.13.15-stable-c5ba367e/windows-amd64/go1.21.6
INFO [07-31|18:27:56.221] New local node record                    seq=1,721
,634,763,767 id=043aae321e492703 ip=127.0.0.1 udp=30304 tcp=30304
INFO [07-31|18:27:56.223] Started P2P networking                   self=enod
e://3903fb9199c73d3836528ef6a0fb49fe4563411d2c9db85735848934126e847ea7cbc009
f618b3b4c331b680f7a7096bf35fab89a771e30509b679f99687e2ed@127.0.0.1:30304
INFO [07-31|18:27:56.228] IPC endpoint opened                      url=\\.\p
ipe\geth.ipc
INFO [07-31|18:27:56.229] Loaded JWT secret file                   path=C:\U
sers\eriss\Geth\node1\geth\jwtsecret crc32=0xe9e3bb1a
INFO [07-31|18:27:56.231] HTTP server started                      endpoint=
127.0.0.1:8552 auth=false prefix= cors= vhosts=localhost
INFO [07-31|18:27:56.245] WebSocket enabled                        url=ws://
127.0.0.1:8553
INFO [07-31|18:27:56.245] HTTP server started                      endpoint=
127.0.0.1:8553 auth=true  prefix= cors=localhost vhosts=localhost
INFO [07-31|18:27:56.246] Stats daemon started
INFO [07-31|18:27:57.040] Unlocked account                         address=0
xd273F775d819136cfE8e85BeC3cde5309b67908C
INFO [07-31|18:27:57.042] Legacy pool tip threshold updated        tip=0
INFO [07-31|18:27:57.042] Legacy pool tip threshold updated        tip=1,000
,000,000
INFO [07-31|18:27:57.044] Commit new sealing work                  number=92
94 sealhash=ac525c..102804 txs=0 gas=0 fees=0 elapsed=1.772ms
INFO [07-31|18:27:57.047] Successfully sealed new block            number=92
94 sealhash=ac525c..102804 hash=dae59b..7c8eb6 elapsed=3.542ms
```

Figure 4 Ethereum network run on node1

The node performs mining at 5-second intervals, as specified in the genesis.json file. Node 1 begins peer-to-peer communication by activating its enode and searching for connected peers. If no peers are found, the peer count remains at 0. After Node 1 is running, Node 2 is then started with its specific command. The result of the Node 2 implementation is shown in Figure 6.2. Once running, Node 2 starts peer-to-peer communication using its enode. If Node 1 and Node 2 successfully connect, Node 2 will immediately begin synchronizing blocks from the existing Ethereum Blockchain network. If the blocks match the latest block in the network, Node 2 will import new chain segments from Node 1.



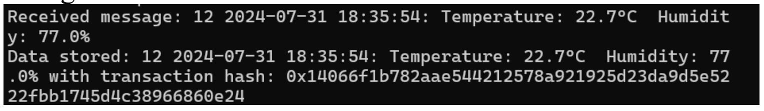
```
INFO [07-31|18:30:37.397] Started P2P networking                    self=enod
e://a36c9217ae87aa1dc2a277401282318be66a38191358da3c00c726beffcddef88893e202
862dcc8125d2dacf0ad59295ddbf076f1e368a7cc06c93d8c00183c0@127.0.0.1:30305
INFO [07-31|18:30:37.399] Loaded JWT secret file                    path=C:\U
sers\eriss\Geth\node2\geth\jwtsecret crc32=0xa75f9784
INFO [07-31|18:30:37.401] HTTP server started                       endpoint=
127.0.0.1:8554 auth=false prefix= cors= vhosts=localhost
INFO [07-31|18:30:37.410] WebSocket enabled                         url=ws://
127.0.0.1:8555
INFO [07-31|18:30:37.410] HTTP server started                       endpoint=
127.0.0.1:8555 auth=true  prefix= cors=localhost vhosts=localhost
INFO [07-31|18:30:47.426] Block synchronisation started
INFO [07-31|18:30:47.503] Imported new chain segment                number=93
28 hash=4e6efa..4c6395 blocks=35 txs=0 mgas=0.000 elapsed=69.597ms mgasps=0.
000 snapdiffs=327.44KiB triedirty=0.00B
INFO [07-31|18:30:47.563] Looking for peers                         peercount
=1 tried=75 static=2
INFO [07-31|18:30:52.007] Imported new chain segment                number=93
29 hash=dd1522..c74030 blocks=1  txs=0 mgas=0.000 elapsed=1.637ms  mgasps=0.
000 snapdiffs=327.44KiB triedirty=0.00B
```

Figure 5 Ethereum network on node 2

The network operates on a local network with a chain ID of 5335, as configured in the genesis file, and uses the Proof of Authority (PoA) consensus mechanism. This indicates that the network is not running on the main Ethereum network or any other public Ethereum network.

3.2. Data Storage

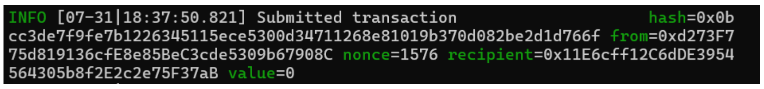
The running Ethereum network is used to implement data storage from sensor readings transmitted via MQTT. The storage program (store.py) is combined with a user interface program (app.py), as shown in Figure 6.



```
Received message: 12 2024-07-31 18:35:54: Temperature: 22.7°C  Humidit
y: 77.0%
Data stored: 12 2024-07-31 18:35:54: Temperature: 22.7°C  Humidity: 77
.0% with transaction hash: 0x14066f1b782aae544212578a921925d23da9d5e52
22fbb1745d4c38966860e24
```

Figure 6 Data sensor stored

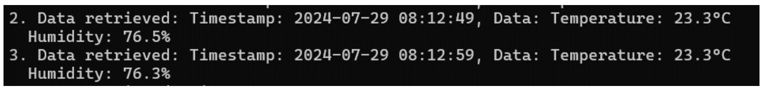
Data sent by the publisher from the Raspberry Pi and received by the subscriber on the laptop is directly stored on the running Ethereum Blockchain network. Each piece of data sent is stored on the blockchain as a transaction by Node 1. Data storage occurs whenever data is received by the subscriber. An example of successful data storage on the Ethereum network is shown in Figure 6.5. Each successful transaction stored at the contract address generates a transaction hash and a "nonce" value, indicating the order of stored data in the blockchain. The hash value is displayed on the web interface along with the data sent by the Raspberry Pi. If the Raspberry Pi does not send any data, the Ethereum network will continue running without storing information, resulting in no transactions by Node 1. The app.py program displays the most recently stored data transaction on the network, as shown in Figure 7.



```
INFO [07-31|18:37:50.821] Submitted transaction                    hash=0x0b
cc3de7f9fe7b1226345115ece5300d34711268e81019b370d082be2d1d766f from=0xd273F7
75d819136cfE8e85BeC3cde5309b67908C nonce=1576 recipient=0x11E6cff12C6dDE3954
564305b8f2E2c2e75F37aB value=0
```

Figure 7 Transaction on Ethereum Network from node 1

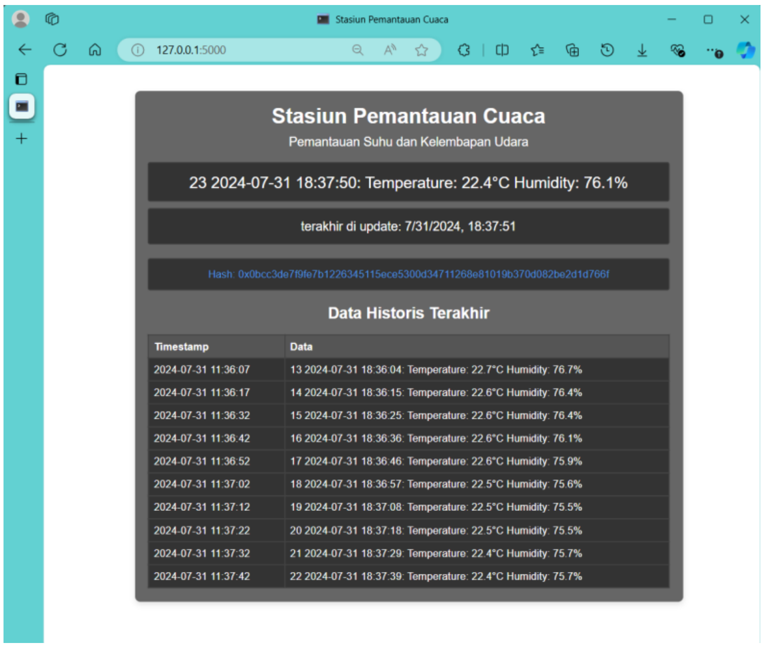
To read data stored on the blockchain, the retrieve.py program can be used, which reads data from the Ethereum Blockchain as long as the network is running. The data retrieved includes the first data stored on the blockchain up to the last data at the time the retrieve.py program is executed. The result of the retrieve.py implementation is shown in Figure 8. The retrieve program reads all stored data from the start of the Ethereum Blockchain network, allowing any changes in data to be visible and recorded.



```
2. Data retrieved: Timestamp: 2024-07-29 08:12:49, Data: Temperature: 23.3°C
  Humidity: 76.5%
3. Data retrieved: Timestamp: 2024-07-29 08:12:59, Data: Temperature: 23.3°C
  Humidity: 76.3%
```

Figure 8 Data retieved from node 2

The web interface is accessed through http://127.0.0.1:5000. The result of the web implementation for the user interface, which is based on Flask and runs on localhost, is shown in Figure 9.



Stasiun Pemantauan Cuaca

Pemantauan Suhu dan Kelembapan Udara

23 2024-07-31 18:37:50: Temperature: 22.4°C Humidity: 76.1%

terakhir di update: 7/31/2024, 18:37:51

Hash: 0x0bcc3de7f9fe7b1226345115ece5300d34711268e81019b370d082be2d1d766f

Data Historis Terakhir

Timestamp	Data
2024-07-31 11:36:07	13 2024-07-31 18:36:04: Temperature: 22.7°C Humidity: 76.7%
2024-07-31 11:36:17	14 2024-07-31 18:36:15: Temperature: 22.6°C Humidity: 76.4%
2024-07-31 11:36:32	15 2024-07-31 18:36:25: Temperature: 22.6°C Humidity: 76.4%
2024-07-31 11:36:42	16 2024-07-31 18:36:36: Temperature: 22.6°C Humidity: 76.1%
2024-07-31 11:36:52	17 2024-07-31 18:36:46: Temperature: 22.6°C Humidity: 75.9%
2024-07-31 11:37:02	18 2024-07-31 18:36:57: Temperature: 22.5°C Humidity: 75.6%
2024-07-31 11:37:12	19 2024-07-31 18:37:08: Temperature: 22.5°C Humidity: 75.5%
2024-07-31 11:37:22	20 2024-07-31 18:37:18: Temperature: 22.5°C Humidity: 75.5%
2024-07-31 11:37:32	21 2024-07-31 18:37:29: Temperature: 22.4°C Humidity: 75.7%
2024-07-31 11:37:42	22 2024-07-31 18:37:39: Temperature: 22.4°C Humidity: 75.7%

Figure 9 User interface

When the Ethereum network is running and sensor data is sent, the data is displayed on the web interface as shown in Figure 9. The displayed data includes the timestamp of data sent from the Raspberry Pi, the sensor readings, the transaction hash of the data storage on the Ethereum Blockchain, the last updated timestamp, and additional data already stored on the network. The displayed data is updated every second. The data sent by the publisher, received by the subscriber, and retrieved data is stored in a CSV file as a backup, updated every minute.

3.3. Network Latency

During the data storage process, which begins with data transmission by the publisher, reception by the subscriber, and storage on the blockchain until the data is retrieved, there is a time delay in each process. To measure network latency, tests were conducted by varying the data transmission interval from the Raspberry Pi. The test was performed with transmission intervals of 0, 5, 10, 15, 20 and 30 seconds, using 20 batch sensor readings. To ensure any delays in the data storage process on the Ethereum Blockchain, tests were also conducted on the Ethereum network with different mining periods, specifically 5 seconds and 10 seconds.

Figure 10 and Figure 11 showed the most stable delay observed during the data transmission process without any data loss. This stability is achieved by carefully optimizing the interval between data transmission from the publisher and the mining period within the Ethereum Blockchain network. The graph shows that when the transmission interval is synchronized with the blockchain's mining period, data is consistently stored without loss, and the delay remains within an acceptable range. This result underscores the importance of aligning data transmission intervals with the blockchain's processing capabilities to ensure data integrity and real-time synchronization.

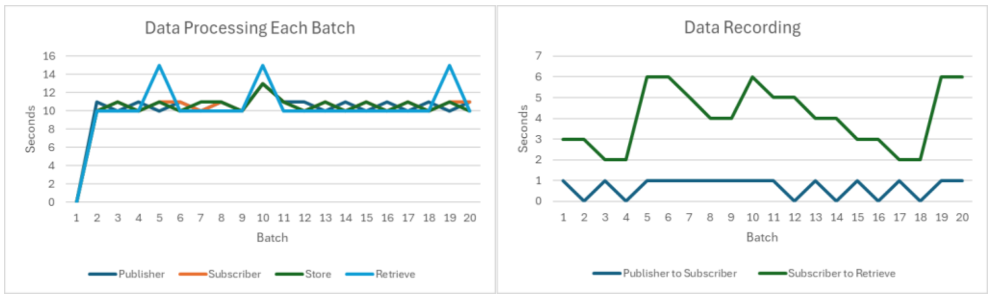


Figure 10 Latency for 5 seconds mining period within 10 seconds data transmission

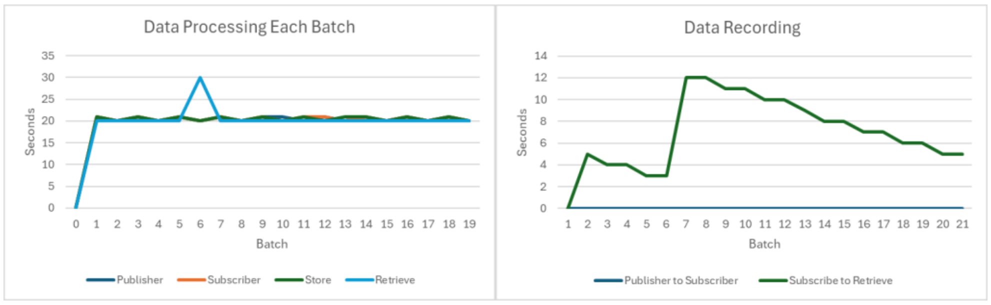


Figure 11 Latency for 10 seconds mining period within 20 seconds data transmission

The results indicate that when the data transmission interval exceeds the mining period, all transmitted data is successfully stored in the blockchain. However, the delay in storing data varies depending on the transmission interval. This study tested mining periods of 5 and 10 seconds, revealing several key points regarding real-time data transmission and transaction recording:

1. Publisher-Subscriber Delay: There is a consistent delay of about 1 second between the publisher and subscriber. This delay does not significantly impact data storage in the Ethereum Blockchain when using a 5-second mining period. The subscriber has approximately 4 seconds to process and prepare the data for storage. In contrast, with a 10-second mining period, data transmission from the publisher to the subscriber occurs without noticeable delay, and only occasionally experiences a 1-second lag.
2. Subscriber-to-Storage Delay: With a 5-second mining period, the delay from the subscriber to data storage varies between 2 to 6 seconds. For the 10-second mining period, this delay ranges from 3 to 12 seconds. This variability is crucial as it sometimes exceeds the mining period. When this happens, the transaction cannot be included in the current block and must wait for the next one. Such delays can lead to inconsistencies in blockchain data recording, potentially affecting the real-time nature of the system. Delays that exceed the mining period can cause significant temporal lags in data availability on the blockchain.
3. Overall Process Delay: The total delay from the publisher to data storage ranges from 3 to 7 seconds for the 5-second mining period. While most data can be processed within the 5-second window, some transactions may still experience delays. Similarly, for the 10-second mining period, the total process delay ranges from 3 to 12 seconds. This delay could lead to scenarios where data is not immediately synchronized on the blockchain, which is critical for applications requiring up-to-date information.

The overall delay in the data flow, especially from the subscriber to data storage, directly impacts the efficiency and timeliness of transaction recording on the blockchain. Minimizing this delay is crucial to ensure that transactions consistently meet the mining period, thus maintaining the integrity and real-time nature of data processing.

Implementing a blockchain-based data storage system for weather monitoring presents both opportunities and challenges. Managing data from approximately 200 weather stations in Indonesia requires careful consideration of latency, transaction costs, and data management strategies. The main challenge is ensuring timely data availability and optimizing the system to handle large data volumes efficiently. Given that real-time data is typically transmitted to the BMKG server every 10 minutes, this interval provides sufficient time for data collection and storage before being added to the blockchain. Efficient data storage mechanisms could involve storing only metadata and recorded hashes in the blockchain. Additionally, transaction costs or gas fees on the Ethereum Blockchain must be considered. Reducing gas fees can be achieved by minimizing the total number of required transactions and combining data from multiple stations into a single transaction.

4. CONCLUSION

This research implemented an Ethereum Blockchain with a Proof of Authority consensus to ensure the integrity and transparency of IoT data from weather monitoring stations, transmitted via the MQTT protocol at intervals of 0, 5, 10, 15, and 20 seconds with mining periods of 5 and 10 seconds. The findings indicate that the maximum delay in data transmission from the publisher to the subscriber was consistently around 1 second. The delay in storing data on the Ethereum Blockchain from the subscriber showed greater variability, ranging from 2 to 6 seconds for a 5-second mining period and 3 to 12 seconds for a 10-second mining period.

The study also revealed that a 5-second mining period with transmission intervals of 0 and 5 seconds resulted in different storage delays compared to intervals of 10, 15, and 20 seconds, which had similar variability. Similarly, for a 10-second mining period, data storage delays stabilized with similar variability when the transmission intervals were 15, 20, and 30 seconds. The delay in data storage is closely related to the mining period applied on the Ethereum Blockchain network, impacting the completeness and timing of data storage from IoT devices. To avoid incomplete data storage, it is recommended to set a mining period slower than the data transmission interval from IoT devices, allowing for more synchronized data transmission and storage, leading to more accurate and complete data on the blockchain.

5. FUTURE WORKS

Future research should consider developing the Ethereum Blockchain network on a public network to expand its applicability to broader systems. Additionally, it would be beneficial to conduct studies involving larger data volumes or more varied IoT devices to assess the performance of the Ethereum Blockchain network in storing data on a larger scale.

REFERENCES

[1] K. S. Shashidhara, S. Pradeep Kumar, V. B. Ganjihal, S. S. Phatate, S. S. Shetty, and R. Vinay, “IoT Enabled Weather Monitoring System,” in 2022 IEEE North Karnataka Subsection Flagship International Conference, NKCon 2022, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/NKCon56289.2022.10126649.

[2] A. S. Bin Shahadat, S. I. Ayon, and M. R. Khatun, “Efficient IoT based Weather Station,” in Proceedings of 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering, WIECON-ECE 2020, Institute of Electrical and

Electronics Engineers Inc., Dec. 2020, pp. 227–230. doi: 10.1109/WIECON-ECE52138.2020.9398041.

[3] R. K. Kodali and S. Mandal, "IoT Based Weather Station," 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies, pp. 680–683, 2016.

[4] S. R. Alam, S. Jain, and R. Doriya, "Security threats and solutions to IoT using Blockchain: A Review," in Proceedings - 5th International Conference on Intelligent Computing and Control Systems, ICICCS 2021, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 268–273. doi: 10.1109/ICICCS51141.2021.9432325.

[5] H. Wang and J. Zhang, "Blockchain Based Data Integrity Verification for Large-Scale IoT Data," IEEE Access, vol. 7, pp. 164996–165006, 2019, doi: 10.1109/ACCESS.2019.2952635.

[6] M. B. Mollah et al., "Blockchain for Future Smart Grid: A Comprehensive Survey," Jan. 01, 2021, Institute of Electrical and Electronics Engineers Inc. doi: 10.1109/JIOT.2020.2993601.

[7] V. Kumar, C. Ramesh, and " Storing, "Storing IOT Data Securely in a Private Ethereum Blockchain Storing IOT Data Securely in a Private Ethereum Blockchain Repository Citation Repository Citation," 2019, doi: 10.34917/15778410.

[8] C. L. Chen, Z. Y. Lim, and H. C. Liao, "Blockchain-based community safety security system with iot secure devices," Sustainability (Switzerland), vol. 13, no. 24, Dec. 2021, doi: 10.3390/su132413994.