Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Enhancing the Responsiveness of NFT Blockchain Smart Contract Authentication in Fog Computing Networks

Rivaldo Nugraha[1], Yudha Purwanto[2], Nyoman Bogi Aditya Karna[3]

Rivaldonugraha@student.telkomuniversity.ac.id[1]
Omyudha@telkomuniversity.ac.id[2]
Aditya@telkomuniversity.ac.id[3]

Introduction: The rapid growth of the Internet of Things (IoT) and decentralized technologies has led to an increased demand for efficient and secure authentication mechanisms, particularly in the context of Non-Fungible Token (NFT) blockchain smart contracts. Fog computing networks present a promising solution by distributing computational resources closer to end-users, thereby reducing latency and enhancing responsiveness. However, the challenge remains to optimize authentication processes for NFT smart contracts within these decentralized frameworks. This research seeks to address this gap by exploring innovative methods to enhance the responsiveness of NFT blockchain smart contract authentication in fog computing networks, ultimately contributing to more efficient and user-friendly decentralized applications. By leveraging the unique advantages of fog computing, this study aims to improve the overall performance and scalability of authentication systems, ensuring that they can effectively meet the demands of an increasingly interconnected and digital world.
Objectives: The primary goal of this research is to enhance the authentication speed of Non-Fungible Token (NFT) blockchain smart contracts within fog computing networks by measuring response times between IoT microcontrollers and decentralized applications (DApps) on fog nodes. Additionally, the study aims to analyze how the inclusion of multiple microcontrollers affects the system's scalability and overall performance. Ultimately, the research seeks to provide insights and strategies for optimizing authentication mechanisms in decentralized applications, contributing to improved efficiency and user experience.
Methods: The methodology of this research focuses on designing a microcontroller authentication system within a fog computing framework, utilizing a decentralized application (DApp) that operates on a fog node with

Ethereum NFT blockchain smart contracts for secure authentication. The study involves configuring three IoT microcontrollers, each assigned unique token IDs (16, 32, and 64), and conducting 100 authentication attempts for each microcontroller to measure response times. Data collection occurs in a controlled testing environment to ensure optimal network conditions, and the average response times (ART) are calculated by aggregating the recorded times from these attempts. Additionally, the study conducts comparative analysis to evaluate the impact of different token IDs on scalability, allowing insights into the system's responsiveness and efficiency in enhancing decentralized application performance within fog computing networks.

Results: The results of this research demonstrate the effectiveness of the designed microcontroller authentication system utilizing a decentralized service (DApp) on a fog node. The average response time (ART) values recorded for the microcontrollers with token IDs 16, 32, and 64 were 196.8 ms, 197.45 ms, and 198.46 ms, respectively, with an overall average ART of 197.57 ms. This performance is notably superior compared to the reference journal's average of 2.2 seconds, highlighting the advantages of placing the DApp on fog nodes closer to users, which significantly reduces response times. Additionally, the study revealed minimal differences in ART due to the scalability performance, indicating that optimal network conditions contributed to these variances.

Conclusions: This research illustrates that implementing a microcontroller authentication system on a fog computing framework can greatly enhance the responsiveness and efficiency of decentralized applications. The significantly lower average response times and the demonstrated scalability indicate that fog computing effectively addresses the challenges associated with authentication processes in IoT environments. These findings suggest that deploying DApps on fog nodes can lead to improved user experiences and more efficient system performance in blockchain-based applications.

Keywords: lorem, ipsum, dolor.

1. Introduction

IoT is a new perspective in which objects are interconnected through the internet and are able to interact, understand, and exchange data intelligently. The development of IoT began to increase along with the convergence between wireless networks and the internet. Along with the rapid growth of IoT, characterised by the emergence of IoT platforms for smart homes, smart health, and even smart cars, data communication responsiveness and security issues are also increasing.

To solve the problem of high responsiveness, previous research using fog computing shows that the use of fog computing accelerates IoT processing. In his journal entitled Migration from Cloud Computing to Fog Computing, Hindreen (2021) revealed that data communication requests sent first to the nearest fog node, instead of directly to the cloud, can minimise response time and reduce failures in authentication and data delivery.

In addition, in terms of IoT communication security, NFT blockchain technology has recently

been used to provide a distributed and cryptographically secure blockchain, which allows data traces to be immutable and guarantees data ownership and user privacy. This is supported by Dan Chirtoaca's (2020) research in his journal entitled Making Smart Contract References on the NFT Blockchain, which identifies the current most popular application domains for NFT smart contracts and compiles a list of features and extensions that are often used on these smart contracts. These features and extensions have been organised into a comprehensive suite of smart contracts that support the ERC721 standard.

Thus, the focus of this research is the utilisation of fog computing schemes to integrate IoT microcontrollers with NFT blockchain smart contract technology to perform authentication, so that faster connectivity can be established between IoT microcontrollers and fog nodes (fog computing) locally placed on the user side using Raspberry Pi. In this process, a smart contract using the NFT blockchain generates a token ID containing key management and contract identity, which is then stored in the EEPROM (Electrically Erasable Programmable Read-Only Memory) of the IoT microcontroller using ESP8266. In this way, IoT devices can authenticate based on the token ID that has been generated using NFT blockchain smart contracts with decentralised blockchain on the fog node device, where the fog node is installed decentralised services (DApp) that will communicate wirelessly.

With the increasing number of IoT devices used in various sectors, problems arise related to responsiveness in the authentication process of these devices. This research aims to integrate and test the responsiveness of IoT microcontroller authentication using ESP8266 with ID tokens from NFT blockchain smart contracts on fog nodes (DApp) in a fog computing scheme. In addition, this research also focuses on adding IoT devices to test the scalability performance of authentication with fog nodes (DApp). The main focus of this research is to test the responsiveness of authentication based on the response time between the IoT microcontroller and the fog node (DApp). In addition, this research also assesses the impact of adding three microcontrollers that are continuously connected to the fog node (DApp). The tests in this study are limited to edge devices used by fog node users in the fog computing scheme, which serves as a decentralised system (DApp) to manage the authentication of IoT devices without involving the cloud.

2. Objectives

George and Jayashree (2023) developed a data sharing mechanism that utilizes blockchain-based authentication and access control among service providers within a cloud model. This approach employs a lightweight blockchain to showcase the effectiveness of techniques aimed at preserving data privacy. The framework leverages Ethereum blockchain Smart Contracts to facilitate efficient data sharing between cloud owners and requesters, enhancing cloud data security without necessitating data sharing with third-party users, thus mitigating privacy concerns. As a result, the detection rate of legitimate users improved by 95%, search accuracy increased by 95%, and responsiveness decreased by 2.25 seconds for a dataset comprising 500 blockchains.

In a study by Javier Arcenegui et al. (2021), titled Secure Combination of IoT and Blockchain by Physically Binding IoT Devices to Smart Non-Fungible Tokens Using PUFs, the NFT

blockchain smart contract scheme was integrated into the SRAM of the ESP32 microcontroller, which possesses PUFs (Physically Unclonable Functions) properties. The code written in SRAM is immutable, providing a robust authentication mechanism for the provider server. The results indicated that mutual authentication between the IoT device and the server was established in 166.7 milliseconds via a UART (Universal Asynchronous Receiver Transmitter) serial connection.

Gupta (2023) emphasized the necessity for industry-wide collaboration to establish common standards for user authentication in IoT environments. This research provides a comprehensive analysis of user authentication in IoT through address-based NFT contracts and 9NM tokens developed on Satoshi core-based blockchains. The proposed framework aims to enhance the security of IoT networks while promoting a transparent and interoperable authentication ecosystem. Additionally, the paper discusses the creation of Web 3.0-based programming using JavaScript, Python, ASP.NET, and PHP for user authentication, facilitated by the presence of smart contracts within the user wallet.

Internet Of Things

The Internet of Things (IoT) is a transformative and rapidly evolving technology that has the potential to alter how we engage with our environment. It encompasses the interconnection of diverse physical objects, devices, and sensors via the Internet, enabling them to communicate and exchange data with one another (Li et al., 2024). According to Laghari et al. (2021), IoT functions as a system that links computer devices, mechanical and digital machines, objects, or individuals, each equipped with a unique identifier (UID), facilitating data transmission without the need for human-to-human or human-computer interaction.

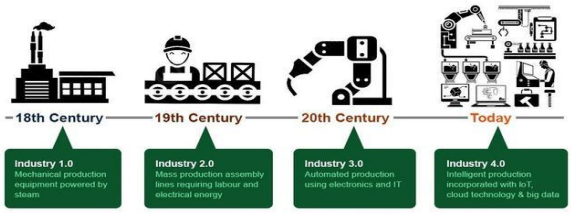

Figure 1 Industrial revolution (Source: kemenperin.go.id)

Additionally, Figure 1 illustrates the evolution of the Internet of Things (IoT) within the context of industrial development, extending to the Fourth Industrial Revolution, which is characterized by the integration of artificial intelligence (AI), cloud computing, and IoT technologies. IoT represents a significant advancement from the Third Industrial Revolution, which was defined by the use of computers and electronics in production processes. This phase facilitates the interconnection of nearly all objects, enabling them to be managed in the virtual realm (Visan, 2021). The most extensive expansion of the Internet is found in the realm of IoT, which has a profound impact across various industries and in our daily lives. For example, household appliances like refrigerators and air conditioners, along with remote-controlled devices such as televisions, can now be operated via smartphones. The IoT ecosystem consists of web-enabled smart devices that utilize integrated systems, including processors, sensors, and communication hardware, to gather, transmit, and respond to the data they collect. These devices reveal the sensor data they gather by interacting with IoT gateways or other edge

devices.

Nodemcu ESP8266

The ESP8266 microcontroller, often referred to as NodeMCU ESP8266, is a module designed for IoT applications and can serve as a key component in controlling systems such as sun tracking systems (Anshory, 2024). According to Sutikno et al. (2021), NodeMCU is an open-source firmware and development kit that streamlines the development of IoT-based application systems. This platform was developed to enable easier implementation of advanced applications for input/output (I/O) hardware, making it accessible for developers working on innovative IoT solutions.

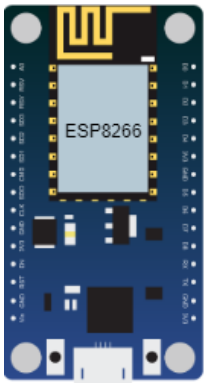

Figure 2 Nodemcu ESP8266

Figure 2 illustrates the NodeMCU ESP8266, which is more compact than comparable microcontrollers like the Arduino Uno, while also providing reliable and affordable Wi-Fi connectivity through its TCP/IP capabilities. The ESP8266 features the ESP-12E module, which houses an ESP8266 chip powered by a 32-bit Tensilica Xtensa LX106 RISC microprocessor, capable of operating at frequencies between 80 MHz and 160 MHz. For data storage, the ESP8266 is equipped with internal FLASH memory, offering a capacity of 4MB. The ESP8266 also serves as a data acquisition platform, connecting to various sensors to capture data. Utilizing this sensor data and calculating the required angles, the ESP8266 generates appropriate control signals to position the servo motor accurately (Visan, 2021).

Decentralized Application (DApp)

A decentralized application (DApp) is an innovative technology aimed at resolving challenges related to trust, privacy, and security. Unlike traditional applications that rely on centralized servers, DApps operate by hosting part of their backend services and databases on a peer-to-peer (P2P) network (Min and Cai, 2022).

According to Renu and Banik (2021), a DApp is specifically hosted on a P2P blockchain network, with Ethereum serving as a prominent platform for various DApps across sectors like insurance, energy, finance, and healthcare. However, it is important to note that many DApps are only partially decentralized. Typically, a DApp comprises a two-tier architecture: the first tier is the client-side application (front-end), while the second tier consists of the server-side application (back-end), where smart contracts are deployed on the blockchain network.

Ethereum was the pioneering blockchain platform to offer a complete programming language for smart contracts, as well as a dedicated DApp development environment, with Solidity being the standard language used for building DApps on this platform.

Smart Contract NFT

NFTs, or non-fungible tokens, fundamentally operate as smart contracts or other types of on-chain code, suggesting that they can function as a form of legal contract (Ellul and Revolidis, 2023). These blockchain-based tokens securely establish ownership rights over digital assets. Just as individuals can appreciate physical artwork in a museum without owning it, NFTs enable representation of ownership or control over digital assets such as art, music, games, and collectibles (Ante, 2021).

Solidity

Solidity is a high-level, object-oriented programming language specifically designed for implementing smart contracts on the Ethereum platform. It is influenced by languages such as C++, Python, and JavaScript, targeting the Ethereum Virtual Machine (EVM) (Ethereum, 2024).

Emphasizing security, Solidity enables developers to create smart contracts that are resilient to attacks and vulnerabilities, which is crucial in the blockchain ecosystem. Researchers have found that Solidity offers numerous advantages, including security, flexibility, integration, and consistency in development. These features empower developers to build secure and efficient decentralized applications within a blockchain environment.

Node.js

Node.js is gaining popularity among web developers due to its ability to efficiently handle asynchronous requests and support a larger number of clients than other frameworks. One of its key features is npm (Node Package Manager), the default package manager for the Node.js JavaScript runtime, which is the largest package manager, exceeding over one million packages (Ntantogian et al., 2021).

According to Manggal et al. (2022), Node.js serves as a server-side platform well-suited for real-time applications because of its event-driven architecture and unlimited I/O capabilities. It boasts performance that is up to ten times faster than traditional I/O services. Additionally, Node.js includes the "npm" management package, which facilitates the distribution and installation of third-party libraries, enhancing existing development projects.

Remote Procedure Call (RPC) Blockchain

In the development of microservices architecture, selecting the appropriate RPC technology is a crucial decision that can significantly impact the nonfunctional requirements of the entire architecture. Commonly utilized RPC technologies include Remote Procedure Call (RPC) and Representational State Transfer (REST) (Zhang et al., 2023). RPC is a protocol that allows a program to communicate with programs located on other computers or clients in a network without needing to understand the underlying network details. It enables remote systems to call processes as if they were local. To set up RPC encryption, the property `hadoop.rpc.protection` should be configured with the value `privacy` in the `core-site.xml` file (Zulfikri et al., 2021).

RPC plays a vital role in the blockchain ecosystem, facilitating efficient interactions between external applications and blockchain nodes. Grasping this concept is essential for leveraging the full potential of blockchain technology and developing applications that can communicate securely and effectively in a decentralized environment.

Fog Computing

Fog Computing is a decentralized computing architecture that provides intelligent processing, storage, and control closer to nearby data devices (Mulyana et al., 2021). According to Sabireen and Neelanarayanan (2021), Fog Computing can be seen as an extension of cloud computing that operates closer to devices that generate IoT data. It acts as an intermediary between end devices and the cloud, bringing storage, network, and computing services nearer to edge devices. The primary characteristics of Fog Computing include adaptability, real-time communication, physical distribution, compatibility, and heterogeneity.

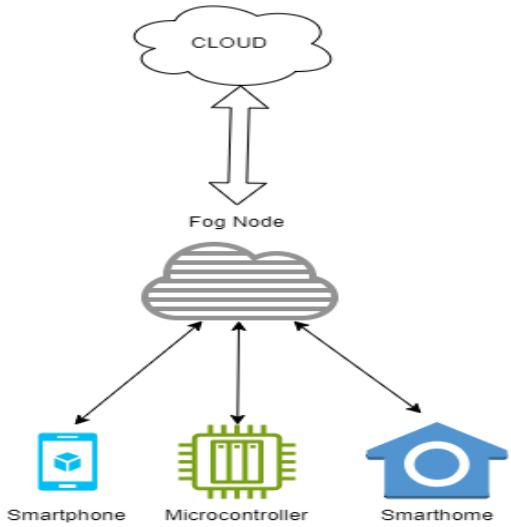

Figure 3 Fog Computing Schema

In Figure 3, the Fog Computing Scheme illustrates the significance of fog nodes within this framework. Fog nodes, situated on the user side, play a crucial role in minimizing latency and enhancing data processing speed. This improvement is possible because the fog computing architecture distributes the workload at the layer nearest to the user, specifically at the fog node. Consequently, this scheme enables the development of applications that are more responsive and adaptable to fluctuating network conditions.

3. Methods

This research employs a method that focuses on designing a microcontroller authentication system using a decentralized application (DApp) on a fog node, specifically leveraging the Ethereum NFT blockchain smart contract within a fog computing framework. The study evaluates key parameters such as the authentication response time between the microcontroller and the DApp, as well as scalability by incorporating three microcontrollers with unique token

IDs (16, 32, and 64), each authenticated 100 times. The data collection involves setting up these microcontrollers to interact with the DApp, recording the time taken for each authentication request to measure response times accurately. A controlled testing environment is maintained to monitor network conditions, ensuring optimal performance during authentication requests.

For data analysis, the research calculates the average response time (ART) for each microcontroller by aggregating and averaging the response times from the 100 authentication requests. Comparative analysis is performed to assess the impact of different token IDs on response times, revealing average ART values of 196.8 ms, 197.45 ms, and 198.46 ms for token IDs 16, 32, and 64, respectively, with an overall ART of 197.57 ms. This performance is benchmarked against a reference journal's average of 2.2 seconds, highlighting the efficiency of the fog computing scheme. The study also evaluates scalability by analyzing differences in ART between the various token IDs, attributing variances of 0.65 ms and 1.01 ms to optimal network conditions. Overall, the findings underscore the potential of fog computing to enhance the responsiveness and scalability of decentralized applications.

4. Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Laoreet id donec ultrices tincidunt arcu. Sollicitudin aliquam ultrices sagittis orci a scelerisque. Sit amet aliquam id diam maecenas ultricies mi.

System Specifications

In conducting this research, researchers used the following system specifications

Table 1 Hardware Specifications

No.	Name	Specifications	Function
1.	Raspberry Pi 4 Model B	64 bit quad-core cortex-A72 Processor, 8 GB LPDDR4 RAM, 256 GB microsd, 802.11b/g/n/ac wireless	As a decentralized service (DApp) that manages smart contracts and authentication access control
2.	Nodemcu microcontroller (ESP8266)	ESP-12E module, ESP8266 chip, Tensilica Xtensa 32-bit LX106 RISC microprocessor, 4 MB Internal Flash Memory	As an IoT microcontroller to perform authentication requests on decentralized services (DApp)
3.	Lenovo G40-45 series 80e1 laptop	AMD A8 Processor, 12 Gb RAM and 128 Gb Memory	As middleware to perform coding and as a medium for connecting between devices

Table 1, titled "Hardware Specifications," presents a detailed list of the hardware utilized in this study. It includes the NodeMCU ESP8266 microcontroller, which is employed to request authentication from the decentralized service (DApp). Additionally, the Raspberry Pi Model B serves as the decentralized service (DApp), functioning as a smart contract manager and access control system for authentication. Lastly, a Lenovo laptop is used as middleware for coding and acts as a connecting medium between the devices.

Table 2 Software Specifications

No.	Platform Name	Function
1.	Windows 10	As bridge software between users and computer devices
2.	Arduino IDE	Platform used to develop IoT microcontrollers using the C programming language base
3.	Node.js version 20.11.0	Framework used in conducting system development for decentralized service backend (DApp)
4.	Solidity	Framework used in performing access control for smart contracts on decentralized services (DApp)
5.	VNC	As a remote desktop to access the raspberry pi module used as a decentralized service (DApp).

Table 2 presents details of the software used in this study. The Windows 10 operating system serves as a bridge between the user and the computer device. Arduino IDE was used as the platform for program development on the IoT microcontroller, with the applied programming language being C. For the development of the backend side in the decentralised service (DApp), Node.js was used as the main framework that manages the interaction with the blockchain. On the other hand, Solidity plays a role in managing access control, including the creation and management of ID tokens to be used by clients in Ethereum's NFT smart contracts, which operate on decentralised services (DApp) located on fog nodes in the fog computing scheme. Finally, the VNC software allows remote access to the Raspberry Pi module that serves as the decentralised service (DApp).

RPC (Remote Procedure Call) Configuration

Infura is an RPC used in this research where Infura itself is a network service provider vendor to be able to communicate with blockchain networks. Infura itself provides endpoints that can be used by developers to create their own decentralized services that are real time connected to the network infrastructure of various blockchains (Endo & Moller, 2020). In RPC infura also has 2 endpoint services, namely using https or using the websocket that has been provided according to the needs of the developer.

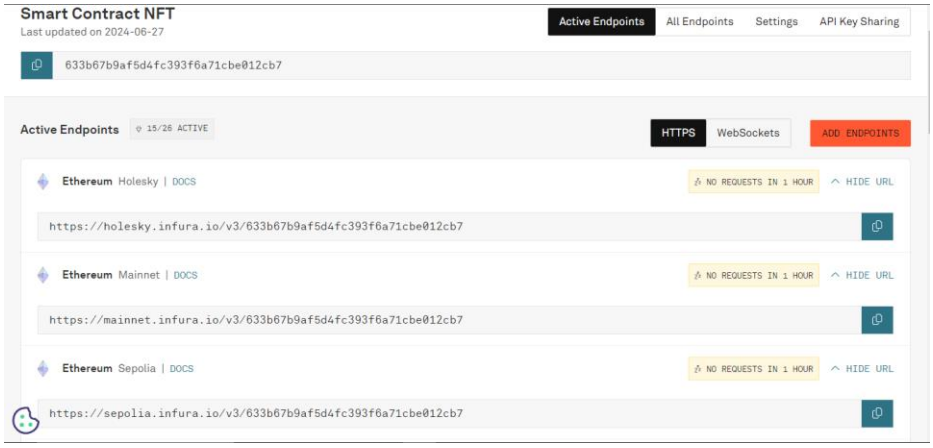

Figure 4 Infura Configuration

In this context, the RPC configuration is carried out through Infura, as illustrated in Figure 4: Infura Configuration. Smart contract testing is conducted using the Sepolia testnet, with a predetermined endpoint. Sepolia is a Proof of Authority (PoA) based testnet, meaning that the validators or nodes participating in the blockchain network have been predetermined and verified. This model facilitates the identification and resolution of potential issues before deploying smart contracts on the main Ethereum network (mainnet). Thus, using Sepolia as a testnet allows for safer and more efficient testing before transitioning to implementation on the main network.

Pseudocode System

Blockchain DApp Pseudocode

```
import Wallet, AbiCoder, JsonRpcProvider, Interface from ethers for interact to blockchain node rpc
import config, abiCode, byteContract, updateConfig, addDbTokenId from ./data for interact with database
import interval from node-interval-return for handle interval

abiIface <- create Interface from abiCode
abiCoder <- create new AbiCoder
wallet <- null
syncTokenId <- false

export define function main()
  url_rpc, key_wallet, interval_sync_token_id_local <- get from config
  If wallet is null:
    wallet <- create new Wallet from key_wallet & JsonRpcProvider(url_rpc)

  define function nftOwnerOf(tokenId)
    data <- encode ownerOf function with tokenId
    call <- send call to NFT contract
    res <- decode call result to address
    return status and result

  define function nftGetTokenURI(tokenId)
    data <- encode getTokenURI function with tokenId
    call <- send call to NFT contract
    res <- decode call result to string
    return status and result

  define function nftGetStoreTokenId()
    data <- encode getStoreTokenId function
    call <- send call to NFT contract
    res <- decode call result to uint256 array
    convert res to number array
    return status and result

  define function nftDeploy()
    tx <- send transaction with byteContract
    wait for tx to finish
    update config with contract address
    return status and tx

  define function nftMint(to, tokenId, tokenURI)
    data <- encode safeMint function
    tx <- send transaction to NFT contract
    return status and tx

  If interval_sync_token_id_local != 0:
    If syncTokenId is false:
      syncTokenId <- true
      call interval(interval_sync_token_id_local, true, () => {
        [_, res] <- call await nftGetStoreTokenId()
        call addDbTokenId(res)
      })

  return variable nftOwnerOf, nftGetTokenURI, nftGetStoreTokenId, nftDeploy, nftMint
```

Figure 5 DApp Pseudocode

Based on the pseudocode presented in Figure 5: Pseudocode DApp, the process begins with importing parameters that load various packages or libraries, enabling the execution of specific syntax. Next, several variables are initialized, including wallet and syncToken ID, to store corresponding values. Following this, the value of the `url_rpc` variable is assigned, which is used to connect to the Infura endpoint. Additionally, other variables, such as the wallet key and `interval_sync_token_id_local`, are sourced from the `config.json` file. This structure ensures that all necessary components are properly set up for the decentralized application (DApp) to function effectively.

Then if the system does not detect the available wallet, the system will create a new wallet through the RPC provider network, namely infura and connect it using key_wallet. Where then using NFT operation functions including:

1. nftOwnerOf(token id) where this function retrieves the NFT owner data based on the token id by calling the ownerOf function on the NFT smart contract.

2. nftGetStoreToken id() This function retrieves the list of token ids stored in the NFT smart contract.

3. nftDeploy() function is used to deploy a new NFT smart contract by sending a transaction containing the contract code bytes, so that after the transaction is complete the contract address is updated.

4. nftMin() function is used to create a new NFT token id calling the safeMint function on the NFT smart contract.

Next, synchronize the token id from the blockchain network infrastructure with the decentralized service that has been created using the interval_sync_toke_id_local function where this function has the task of periodically calling leftFootStore Token id and adding the results in the local token id database.

Microcontroller Pseudocode

```
// Include Arduino libraries
import library WiFiClient
import library WiFiUdp
import library ESP8266HTTPClient
import library EEPROM
import library ArduinoJson

// Define constants
ssid <- "PURI DEWATA 2"
password <- "googletranslate"
delayLoop <- 1000

// Setup function
function setup()
  Open serial monitor
  Connect to WiFi using ssid and password
  Initialize EEPROM
  print "[INFO] connecting to WiFi..."

  While WiFi is not connected:
    Wait 1 second

  print "[INFO] connection WIFI has success"

  // First Write and commit value to EEPROM (only once)
  EEPROM.write(0, 100)
  EEPROM.commit()

  print "[INFO] dateDelay: " + String(dateDelay)

// Main loop
function loop()
  If WiFi connected:
    dateNow <- get current time
    If dateNow >= dateDelay:
      dateDelay <- dateNow + delayLoop

      Create new client and http
      url <- API URL
      tokenId <- read from EEPROM at position 0

      http start request to url

      http add Content-Type json header
      payloadAuth <- create auth json rpc request
      print "[INFO] payload request auth: " + payloadAuth
      startTime <- get start time
      http send POST request with payloadAuth
      httpCode1 <- response code of request
      endTime <- get end time
      gapTime <- calculate time difference between start and end
      print "[INFO] statusCode request auth: " + httpCode1;

      Parse response json into doc2
      If parsing fails:
        print "[ERROR] failed parse JSON request auth"
        Exit function

      status <- get status from doc2
      If status is true:
        payloadAddBenchmark <- create addBenchmark json rpc request
        print "[INFO] payload request addBenchmark: " + payloadAddBenchmark
        http add Content-Type json header
        http send POST request with payloadAddBenchmark
        httpCode2 <- response code
        print "[INFO] statusCode request addBenchmark: " + httpCode2;

      http close request

// Call setup and loop functions
```

Figure 6 Microcontroller Pseudocode

Based on **Figure 6: Microcontroller Pseudocode**, which illustrates the pseudocode for microcontroller clients, the process begins by importing necessary libraries, including `WiFiClient`, `WiFiUdp`, `ESP8266HTTPClient`, `EEPROM`, and `ArduinoJson`. Next, constants such as `ssid` and `password` for the WiFi network are defined, along with `delayLoop`, which sets the time interval for the main loop.

In the `setup` function, the microcontroller connects to the WiFi network and initializes the EEPROM to store the token ID value, subsequently writing the token ID to the EEPROM and committing the changes. Within the `main loop` function, a conditional check is performed to see if the microcontroller is connected to WiFi. If it is, the `dateDelay` is updated by adding `delayLoop`, followed by the creation of new client and HTTP objects. This main loop contains the steps necessary for authenticating the microcontroller client with the server (DApp), which may include tasks such as sending authentication requests, receiving responses, and managing the communication process between the microcontroller and the DApp. In this main loop are the steps to authenticate the microcontroller client with the server (DApp), namely:

1 Specifies the API URL and reads the token id that has been stored in the microcontroller's EEPROM.

2 Initiates an HTTP POST request to the specified URL.

3 Adds a json header to the HTTP request.

4 Create a payload for JSON-RPC authentication requests.

5 Get the HTTP response code and calculate the time required for the request (authentication).

6 Prints the status information of the response code.

7 Parses the JSON response into the doc2 variable.

8 If the parsing fails, then print the error.

9 If the status is true or successful, it creates the payload for the addBenchmark JSON-RPC request.

10 Adds a JSON header to the HTTP request and sends a POST request with the payload for the Benchmark add.

11 Gets the HTTP response code and prints the status information of the response code.

Testing Data

System testing in this study was carried out in 3 stages and used 3 microcontroller devices to be tested:

1 In the first stage, the token id 16 microcontroller was tested 100 times for authentication to the decentralized service (DApp).

2 The second stage in the process of testing token id 16 100 times is carried out scalability (adding microcontroller devices), namely microcontroller token id 32 so that the authenticated microcontroller becomes 2 pieces, namely microcontroller token id 16 and microcontroller token id 32.

3 The third stage is carried out scalability again by adding a third device in the process, namely the token id 64 microcontroller so that the authenticated microcontroller becomes 3 pieces, namely the token id 16 microcontroller and the token id 32 microcontroller and the token id 64 microcontroller.

So that in testing the microcontroller system token id 16, token id 32 and token id 64 obtained data for each 100 authentication test data between each microcontroller and decentralized services (DApp). As in table 4.3 Testing Results Data as follows:

Table 3 Testing Result Data

No.	Token id 32	Token id 64	Token id 128
1	172	184	323
2	209	189	171
3	239	195	212
4	185	180	206
5	206	196	219
6	188	223	185
7	185	205	218
8	183	234	208
9	227	191	225
10	208	196	212
11	242	253	201
12	263	277	210
13	287	173	277
14	208	255	199
15	232	246	223
16	258	270	247
17	287	196	273
18	165	216	300
19	175	240	179
20	204	265	241
21	276	187	233
22	192	192	178
23	213	233	195
24	244	178	190
25	188	215	202
26	189	174	201
27	176	176	173
28	194	175	108
29	182	168	205
30	208	189	184
31	194	210	191
32	181	175	207
33	208	190	215
34	233	191	197
35	182	240	177
36	195	237	170
37	194	181	182
38	179	178	206
39	183	203	212
40	195	217	227
41	188	187	187
42	190	202	218
43	174	206	195
44	174	219	183
45	176	212	166
46	194	229	192
47	122	189	167
48	174	174	176
49	182	199	189
50	179	178	198
51	222	178	193
52	169	188	178

53	177	180	181
54	182	186	201
55	169	211	196
56	186	167	197
57	194	159	184
58	185	184	239
59	188	228	179
60	204	193	207
61	198	208	193
62	194	156	175
63	180	177	166
64	213	209	191
65	172	204	175
66	188	189	187
67	213	175	195
68	202	171	218
69	190	177	156
70	177	177	174
71	186	170	180
72	260	215	195
73	209	164	199
74	185	168	205
75	198	204	216
76	182	174	203
77	186	185	203
78	191	213	182
79	178	224	204
80	202	195	184
81	205	207	204
82	181	200	197
83	189	193	198
84	191	193	176
85	189	185	211
86	179	196	176
87	220	205	190
88	216	185	206
89	206	185	194
90	184	184	170
91	213	178	196
92	195	215	174
93	181	186	185
94	172	188	212
95	196	215	185
96	179	182	196
97	181	175	204
98	197	189	194
99	173	179	181
100	171	188	188

Analysis of Data Testing Results

In analyzing the results of testing the responsiveness of authentication between each microcontroller connected to a decentralized service (DApp) using the smartcontract nft token id with response time parameters. Where the response time parameter of this research is to find an actual value to find out how long it takes the microcontroller to authenticate and get the Average Response Time (ART) value based on the data that has been obtained in the test.

4.5.1 Token id 16

To get the average response time value on token id 16 from the test results are carried out using the following formula:

$$\mathrm{ART} = \frac{\sum_{i=1}^{n} \mathrm{Ri}}{n}$$

So that the following value is obtained:

$\mathrm{ART} = \sum_{i=1}^{100} \mathrm{R100} = 19680 \rightarrow \frac{19680}{n(100)} = 196.8$ milliseconds

Description:

ART = Average Response Time

n = number of observations

i = observation index

R_i = Response time at observation i

Token id 32

To get the average response time value on token id 32 from the test results are carried out using the following formula:

$$\mathrm{ART} = \frac{\sum_{i=1}^{n} \mathrm{Ri}}{n}$$

So that the following value is obtained:

$\mathrm{ART} = \sum_{i=1}^{100} \mathrm{R100} = 19745 \rightarrow \frac{19745}{n(100)} = 197.45$ milliseconds

Description:

ART = Average Response Time

n = number of observations

i = observation index

R_i = Response time at observation i

4.5.3 Token id 64

To get the average response time value on token id 64 from the test results are carried out using the following formula:

$$\mathrm{ART} = \frac{\sum_{i=1}^{n} \mathrm{Ri}}{n}$$

So that the following value is obtained:

$\mathrm{ART} = \sum_{i=1}^{100} \mathrm{R100} = 19846 \rightarrow \frac{19846}{n(100)} = 198.46$ milliseconds

Description:

ART = Average Response Time

n = number of observations

i = observation index

R_i = Response time at observation i

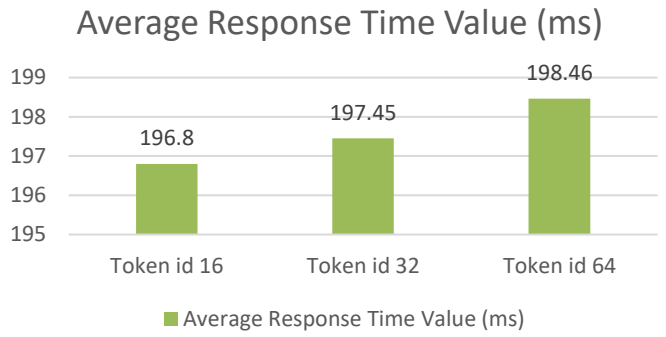

Figure 7 Comparison of Average Response Time of Each Testing Stage

The average authentication response time value for each microcontroller and each stage as shown in Figure 7 Comparison of Average Response Time (ART) Values for Each Testing Stage:

1 In the first stage, the authenticated microcontroller is the token id 16 microcontroller which has authentication data as much as 100 test data and gets an average response time (ART) value of 196.8 milliseconds.

2 In the second stage, in the process of testing, scalability (adding microcontrollers) is carried out by adding a 32 token id microcontroller. So that the authenticated microcontroller becomes 2 pieces, namely the token id 16 microcontroller and the token id 32 microcontroller and the authentication test results are obtained as many as 100 test data results each. So that the average value of authentication response time for microcontroller token id 16 for 196.8 milliseconds and microcontroller token id 32 for 197.45 milliseconds.

3 In the third stage, in the process of testing, scalability is carried out again (adding microcontrollers) by adding a token ID 64 microcontroller. So that the authenticated microcontroller becomes 3 pieces, namely the token id 16 microcontroller, token id 32 microcontroller and token id 64 microcontroller where token id 64 has 100 test data. So that the average value of authentication response time for microcontroller token id 32 for 196.8 milliseconds, and microcontroller token id 64 for 197.45 milliseconds.

Based on the Average Response Time (ART) values for each microcontroller connected to the decentralized service (DApp) at regular intervals, the response times remain under 200 milliseconds. This indicates that the fog computing scheme, in which the decentralized service (DApp) functions as a smart contract manager and access control for authentication, significantly enhances responsiveness. The proximity of the decentralized service (DApp) to

the user contributes to faster authentication access.

Additionally, the results reveal a difference in authentication scalability performance between microcontrollers with token IDs 16 and 32, which shows a variance of 0.65 milliseconds, and between microcontrollers with token IDs 32 and 64, which indicates a difference of 1.01 milliseconds. This variation occurs because the first request benefits from a more optimal network path or operates under ideal network conditions, resulting in a quicker response time compared to subsequent requests.

5. Discussion

This research focuses on designing a microcontroller authentication system that utilizes a decentralized service (DApp) on a fog node, leveraging the Ethereum NFT blockchain smart contract within a fog computing framework. The study evaluated parameters such as the authentication response time between the microcontroller and the DApp, as well as scalability by incorporating three microcontrollers, each authenticated 100 times. The findings indicated that the average response time (ART) values for microcontrollers with token IDs 16, 32, and 64 were 196.8 ms, 197.45 ms, and 198.46 ms, respectively. The overall average ART was 197.57 ms, significantly faster than the reference journal's average of 2.2 seconds. This enhancement can be attributed to the fog computing scheme, which situates the DApp on fog nodes nearer to the user, thereby reducing response times compared to a direct connection to the cloud. Additionally, the scalability performance revealed a difference in ART of 0.65 ms between microcontrollers with token IDs 16 and 32, and a difference of 1.01 ms between token IDs 32 and 64, with these variances resulting from optimal network conditions during the first request.

Future research is recommended to involve sensors and the cloud in a fog computing scheme using NFT blockchain smart contracts, in order to determine the responsiveness of data from microcontrollers sent by sensors to the cloud through fog nodes.

References

1. Ante, L. 2021. Non-fungible Token (NFT) Markets on the Ethereum Blockchain: Temporal Development, Cointegration and Interrelations. BRL Working Paper Series, (22); 1-23.
2. Anshory, I., Jamaaluddin, Fahruddin, A. 2024. Monitoring Solar Heat Intensity of Dual Axis Solar Tracker Control System: A New Approach. Case Studies in Thermal Engineering, 53; 1-8.
3. Chirtoaca, D., Ellul,, J., Azzopardi, G. 2020. A Framework For Creating Deployable Smart Contracts For Non-Fungible Tokens On Ethereum Blockchain. IEEE Computer science. 100-105.
4. Ellul, J, Revolidis, I. 2023. Non-Fungible Tokens (NFTs), Smart Contracts and Contracts: The need for Legal and Technology Assurances. Available at SSRN; 1-24.
5. Endo, N.T., Moller, A. 2020. NodeRacer: Event Race Detection for Node.js Applications. IEE Computer Society, 120-130.
6. Ethereum. 2024. Solidity Documentation. BASICS. 1-412.
7. George, G. M. 2023. Ethereum Blockchain-Based Authentication Approach for Data Sharing in

Cloud Storage Model. Cybernetics and Systemilisecond: An International Journal. 54(6); 961-984
8. Gupta, M. 2023. Integration of IoT and Blockchain for user Authentication. Scientific Journal of Metaverse and Blockchain Technologies, 1(1); 72-84.
9. Arcenegui, J., Arjona, R., Roman, R., Baturone, I. 2021. Secure Combination and Blockchain by Physically Binding IoT Devices to Smart Non-Fungible Tokens Using PUFs. MDPI, 21(9).
10. Laghari, A., A., Wu, K., Laghari, R., A., Ali, M., Khan, A., A. 2021. A Review and State of Art of Internet of Things (IoT). Archives of Computational Methods in Engineering, 1-19.
11. Li, C., Wang, J., Wang, S., Zhang, Y. 2024. A Review of IoT Applications in Healthcare. Nourocomputing, 565; 1-12.
12. Mulyana, A., Haryanti, T., Pradipta, R., F. 2021. Comparison Analysis of Fog Computing and Cloud Computing in Concern Data Processing of Weather Climate Based on IoT. Journal of Applied Electrical Telecommunications, 8(2); 1127-1137.
13. Ntantogian, C., Bountakas, P., Antonaropoulus, D. 2021. NodeXP: NOde.js Server-Side JavaScript Injection Vulnerability DEtection and EXPloitation. Journal of Information Security and Applications, 58; 1-11.
14. Renu, S.A, Banik, B.G. 2021. Implementation of a Secure Ride-Sharing DApp Using Smart Contracts on Ethereum Blockchain. International Journal of Safety and Security Engineering. 11(2); 167-173.
15. Sabireen, H., Neelanarayanan, V. A Review on Fog Computing: Architecture, Fog with IoT, Algorithmization and Research Challenges. ICT Express, 7(2); 162-176.
16. Sutikno, T., Purnama, H.S., Pamungkas, A., Fadlil, A., Alsofyani, M.I., Jopri, M.H. 2021. Internet of Things-Based Photovoltaics parameter Monitoring System using NodeMCU ESP8266. International Journal of Electrical and Computer Engineering (IJECE), 11(6); 5578-5587.
17. Visan, I., Daiconu, E., M. 2021. Home Automation System Using Esp8266 Microcontroller and Blynk Application. Scientific Bulletin of the Electrical Engineering Faculty, 2(45); 59-62.
18. Zhang, L., Pang, K., Xu, J., Niu, B. 2023. High Performance Microservice Communication Technology Based on Modified Remote Procedure Call. Scientific Report, 13; 1-17.
19. Zulfikri, R.A., Widjajarto, A., Almaarif, A. 2021. Implementation and Analysis of Data Confidentiality Hardening on Hadoop Infrastructure with Encryption Method for Data Security. e-Proceeding of Engineering, 8(5); 9235-9242