KLE Technological University
Creating Value
Leveraging Knowledge
B. V. Bhoomaraddi Engineering & Technology College Campus, Hubballi - India

Blockchain-Powered Decentralized DNS for Enhanced Security

Guide: Jayalaxmi maam

NAME	ROLL NO	USN
KARTHIK KP	228	01FE22BCS183
SUSHMITA MATH	332	01FE22BCS184
HARSHITAA KATWA	348	01FE22BCS257
CHINMAY AVARADI	362	01FE22BCS306

Problem statement:

Design and implement a decentralized DNS using blockchain technology to create a more secure and resilient DNS infrastructure.

Problem description:

The decentralized DNS design using blockchain aims to enhance DNS security and resilience by eliminating single points of failure and mitigating risks like DNS spoofing and phishing. By leveraging blockchain's immutability and smart contracts, domain name records are securely stored, tamper-proof, and transparently verified, ensuring a trust-based, robust DNS infrastructure.

Objectives

1. To explore and assess the feasibility of implementing a decentralized DNS system using blockchain technology to mitigate the vulnerabilities present in traditional DNS
2. To design a blockchain-based architecture for DNS that distributes the domain registration and resolution process, eliminating central points of failure.
3. To create a secure smart contract framework that enables transparent and tamper proof domain registration and ownership management on the blockchain.
4. To analyse the performance of the decentralized DNS system in handling high traffic loads, scalability challenges, and gas optimization on the Ethereum network

Comparison Between Traditional DNS and Blockchain Based DNS

Aspect	Traditional DNS	Blockchain-Based DNS
Ownership Control	Only registrars and intermediaries have control.	Full ownership through smart contracts.
Validation Methods	Semi-automated with a high chance of human error.	Fully automated with strict adherence to rules.
Security Attacks Against	Prone to DNS spoofing and DDoS attacks.	Decentralized and cryptographically secure.
Transparency	Domain changes not clearly visible.	Records are immutable and auditable.
Dependency Trust on	Highly dependent on the registrars.	Smart contracts ensure trustless operation.

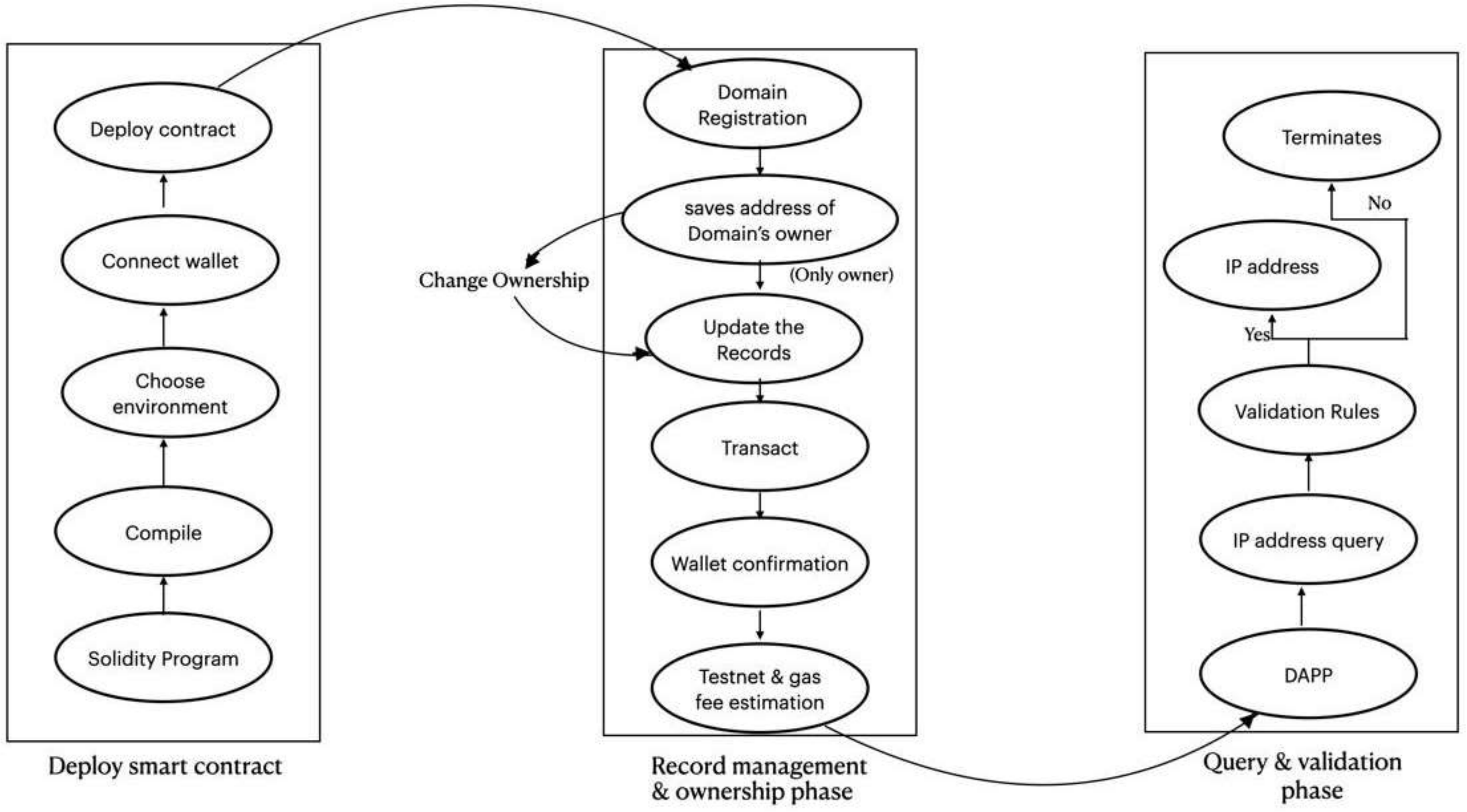
Functional Requirements

1. The system should enable DNS registration directly on the blockchain, eliminating reliance on centralized authorities.
2. Smart contracts should handle secure registration, updates, and deletion of DNS records.
3. The blockchain must support resolving DNS queries using stored data.
4. Only verified owners should be allowed to modify domain details, including IP addresses and canonical names.
5. The system must support secure and seamless ownership transfers of domains.

Non-Functional Requirements

1. Use the blockchain's immutable nature and keccak256 hashing for tamper-proof storage and validation of domain-IP combinations.
2. Optimize the smart contract to minimize gas costs during storage and event-driven operations.
3. The solution must support a growing number of domains and transactions without performance degradation.
4. Ensure fast execution times for operations such as domain registration, query res olution, and ownership validation.

Architecture design



Architecture design

(a) Deploy Smart Contract Phase

•Solidity Program: Write the smart contract in Solidity for Ethereum-based blockchains.
•Compile: Compile Solidity code into bytecode executable by the Ethereum Virtual Machine (EVM).
•Choose Environment: Use tools like Remix IDE and deploy on a blockchain network (e.g., Sepolia Testnet).
•Connect Wallet: Connect a wallet like MetaMask to the blockchain for signing transactions and paying gas fees.
•Deploy: Deploy the compiled smart contract, storing it immutably on the blockchain.

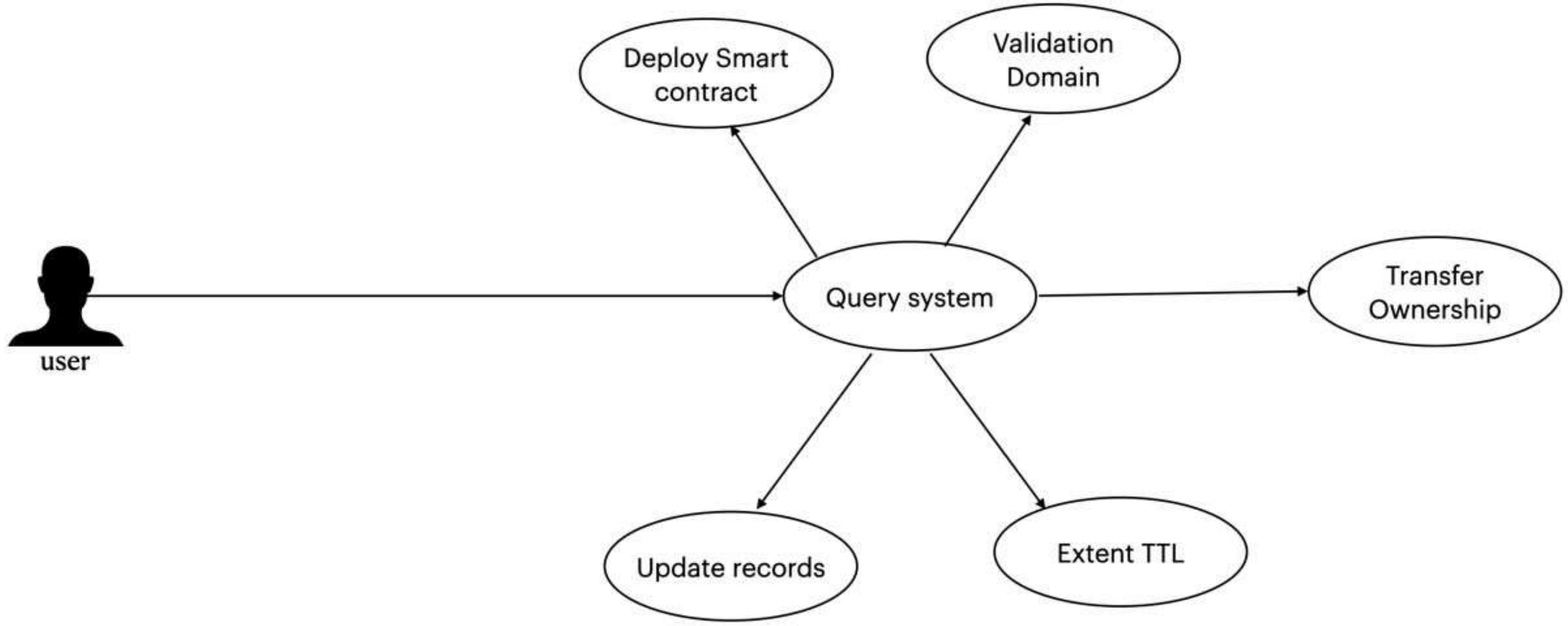
(b) Record Management and Ownership Phase

•Domain Registration: The smart contract registers domains and tracks ownership through the owner's address.
•Secure Ownership: The domain owner’s address is securely stored to prevent unauthorized changes.
•Update Records: Owners can update DNS records (e.g., mapping domain names to IP addresses).
•Transactions: Blockchain transactions are created for updates or ownership transfers, incurring gas fees.
•Wallet Confirmation: Wallet signs and confirms transactions, ensuring authorization.
•Testnet & Gas Fee: Transactions are tested on a testnet (e.g., Sepolia) to estimate gas fees and verify smart contract behavior.

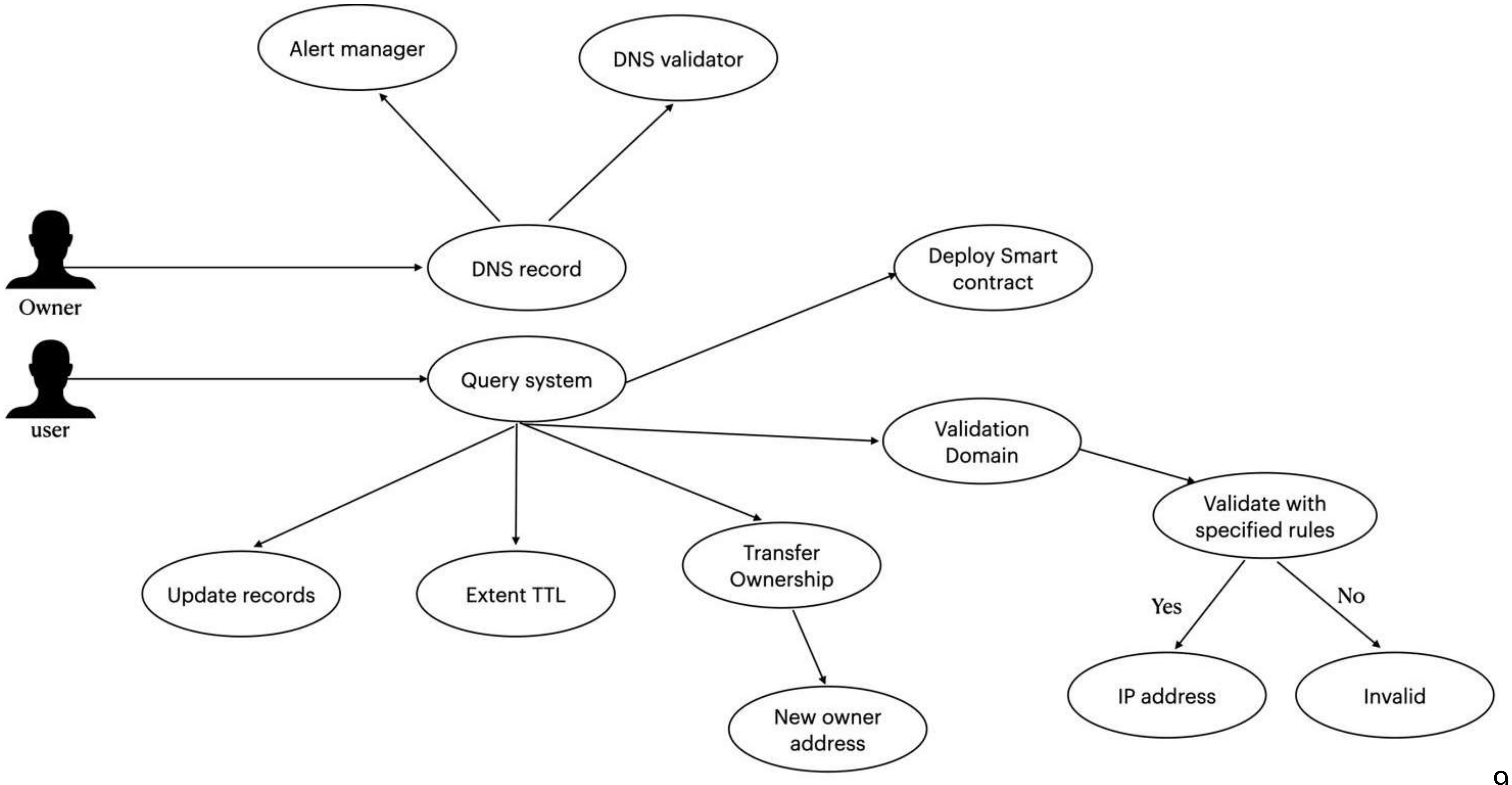
(c) Query and Verification Phase

•DApp: A front-end decentralized application (DApp) allows querying of domain information.
•IP Address Query: DApp fetches the IP address linked to a domain via the smart contract.
•Validation Rules: Ensures queried IP matches the correct domain, validating DNS record integrity.
•Response: Returns the IP address if validation succeeds; otherwise, the query terminates without returning invalid data.

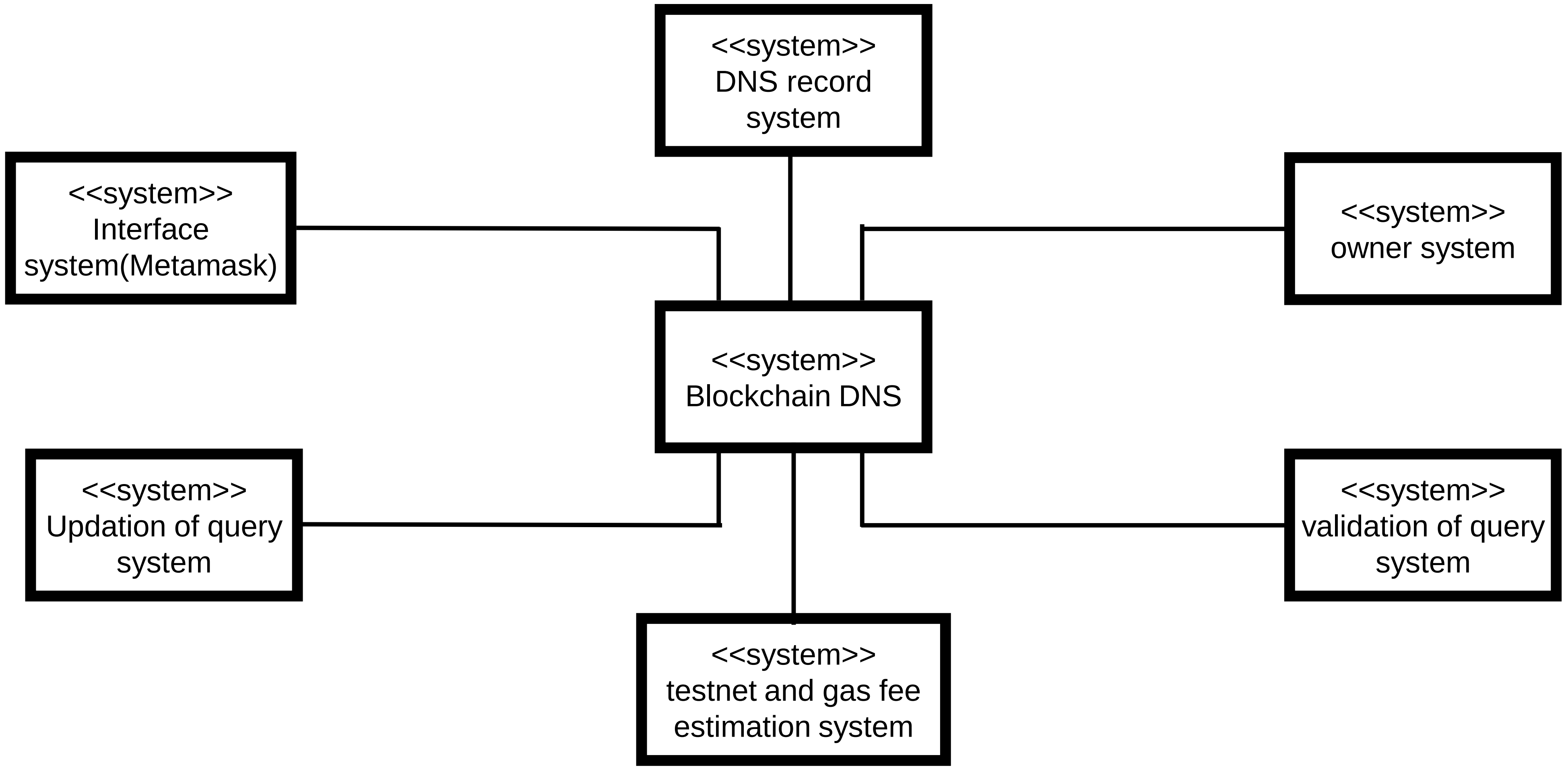
High-level design



Low-level design



System Modelling


<<system>> DNS record system
<<system>> Interface system(Metamask)
<<system>> owner system
<<system>> Blockchain DNS
<<system>> Updation of query system
<<system>> validation of query system
<<system>> testnet and gas fee estimation system

System Modelling

Workflow Explanation:

(a) A user uses the Interface System (MetaMask) to perform operations such as registering or updating a domain.

(b) Validation of Query System verifies the genuineness and authorization of the request made by the user.

(c) Based on the type of operation, the request is sent to either DNS Record System or Updation of Query System.

(d) The Owner System validates ownership and permission before making any updations.

(e) The Testnet and Gas Fee Estimation System calculate transaction costs, offering real-time fee estimations.

(f) All interactions get processed and are stored within the Blockchain DNS, so they remain open to full transparency, are immutable, and are secure.

Workflow Explanation:

(a) A user uses the Interface System (MetaMask) to perform operations such as registering or updating a domain.

(b) (b) Validation of Query System verifies the genuineness and authorization of the request made by the user

(c) (c) Based on the type of operation, the request is sent to either DNS Record System or Updation of Query System

(d) (d) The Owner System validates ownership and permission before making any updations.

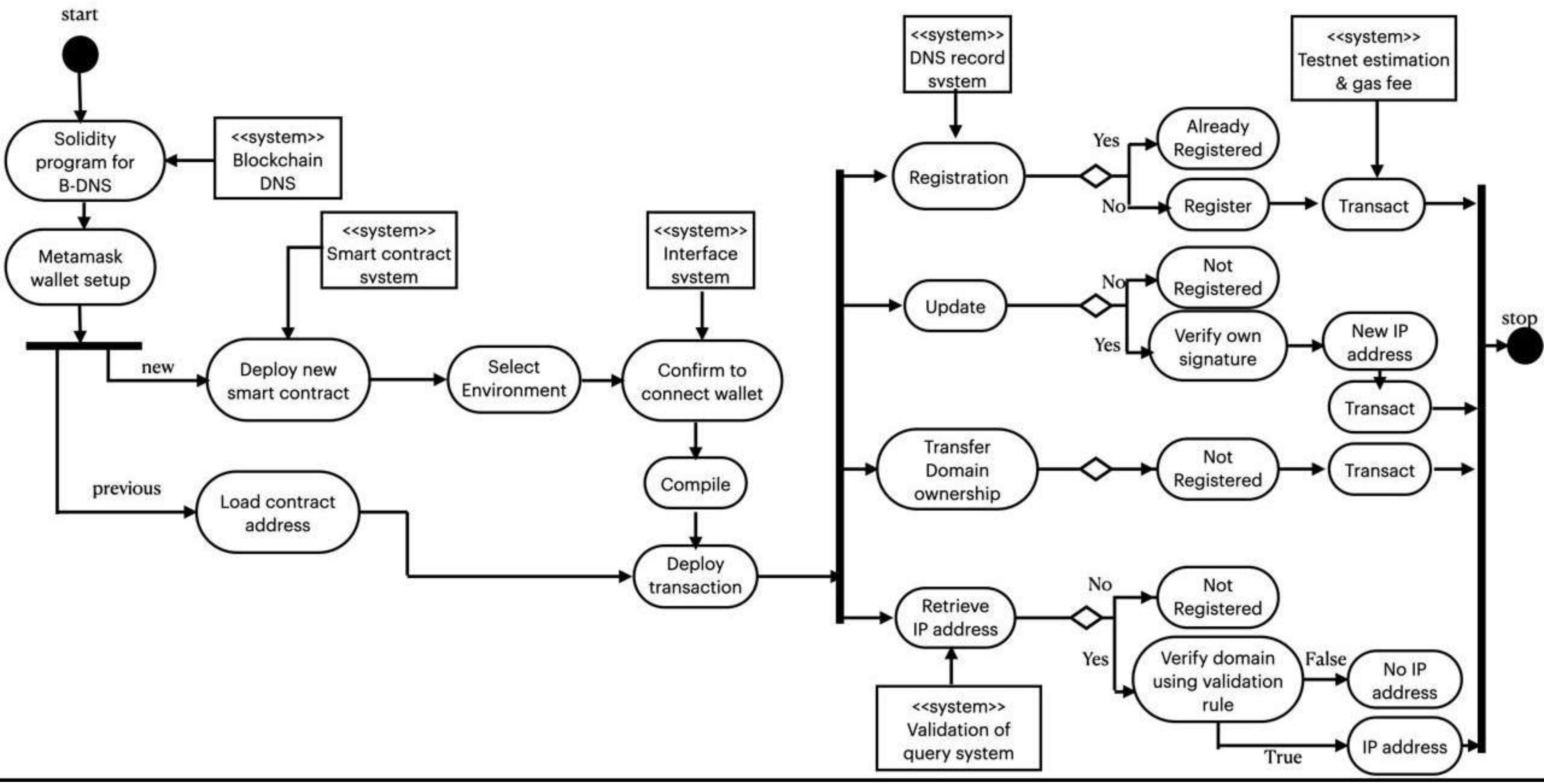
(e) (e) The Testnet and Gas Fee Estimation System calculate transaction costs, offering real-time fee estimations.

(f) (f) All interactions get processed and are stored within the Blockchain DNS, so they remain open to full transparency, are immutable, and are secure.

Implementation

- The goal of this project Implementation is to create a blockchain-based decentralized DNS has to be developed as a safe and transparent mechanism using Ethereum smart contracts.
- Therefore, the foremost objective is developing an immutable mechanism for domain registration, management, and association of IP addresses in addition to e-mail validation with tamper proofing. Hence, Ethereum's robust ecosystem made it suitable to deploy smart contracts. The system is coded in Solidity and deployed to the Remix IDE for testing using the MetaMask wallet.
- This system deploys to the Sepolia test net before migrating it to the Ethereum mainnet for final deployment and testing.
- Functions include registering domain names, ownership transfer, deletion, and changes to the associated IP addresses. Access control modifiers, for instance, only Owner, will protect sensitive operations with regard to ownership so that the change cannot be unauthorized.
- Events are emitted for transparency and trackability, enabling external systems to monitor activities like domain registration and updates. This approach ensures decentralization, increases security, and reduces risks like domain hijacking and DNS spoofing, thereby making the system reliable and efficient for real-world use.
- Remix IDE helped in developing and testing the contract, and MetaMask made it relatively easy to sign transactions and interact with the contract by offering a user-friendly interface , and Sepolia let me safely test at zero cost before deploying on Ethereum’s mainnet . This methodology provided a secure, fully functional contract ready for real-world applications, offering a decentralized yet efficient solution for domain and email registration.

Activity Diagram


start
Solidity program for B-DNS
<<system>> Blockchain DNS
Metamask wallet setup
new
previous
<<system>> Smart contract system
Deploy new smart contract
Select Environment
<<system>> Interface system
Confirm to connect wallet
Compile
Load contract address
Deploy transaction
<<system>> DNS record system
Registration
Yes
No
Already Registered
Register
<<system>> Testnet estimation & gas fee
Transact
Update
No
Yes
Not Registered
Verify own signature
New IP address
Transact
Transfer Domain ownership
Not Registered
Transact
Retrieve IP address
<<system>> Validation of query system
No
Yes
Not Registered
Verify domain using validation rule
False
True
No IP address
IP address
stop

Experimental Setup

Component/Tool	Purpose	Configuration/Details
Remix IDE	Development and testing of smart contracts	Web-based interface; used for Solidity programming and debugging smart contracts.
MetaMask Wallet	Transaction signing and gas fee payments	Integrated with Sepolia test net for testing and Test Ethers.
Sepolia Testnet	Blockchain network for testing	Simulates Ethereum mainnet; uses test Ether for risk-free transactions.
Solidity Language	Writing smart contracts for blockchain DNS	Version: [Specify version used]; En sures compatibility with Ethereum EVM.

Algorithm for Registration

Algorithm for the Registration Process of a Domain:

Input: Domain name, IP address, canonical name, and duration

Output: The registration of a domain with all the metadata attached

Steps:

- Validate the length and format of the domain
- If the domain is less than 4 in length or greater than 255 in length, then throw "Domain length is invalid."
- If the domain doesn't start and end with an alphanumeric character, then throw "Domain must start and end with an alphanumeric character.".
- For each character in the domain (except the first and last):

  If the character is not alphanumeric and not hyphen, throw "Invalid character in domain".
- If there exist consecutive hyphens, throw "Consecutive hyphens are not allowed".
- Checks whether the domain is already registered or not

  The owner of the domain is not null, then throw "Domain already registered".

Algorithm for Registration

.

Validate the IP address format

IP address given is not a valid IPv4 address, then throw "Invalid IPv4 address format".

Create a unique hash for the domain:

Compute domainHash = keccak256(domain + ipAddress).

Store domain details:

Store the domain metadata including the domain hash, owner, timestamp, IP address, expiry time, and canonical name.

Store the mapping cnameToDomain[canonicalName] = domain.

Emit the registration event:

Emit a DomainRegistered event with domain, domain hash, owner, and canonical name.

Algorithm for validation

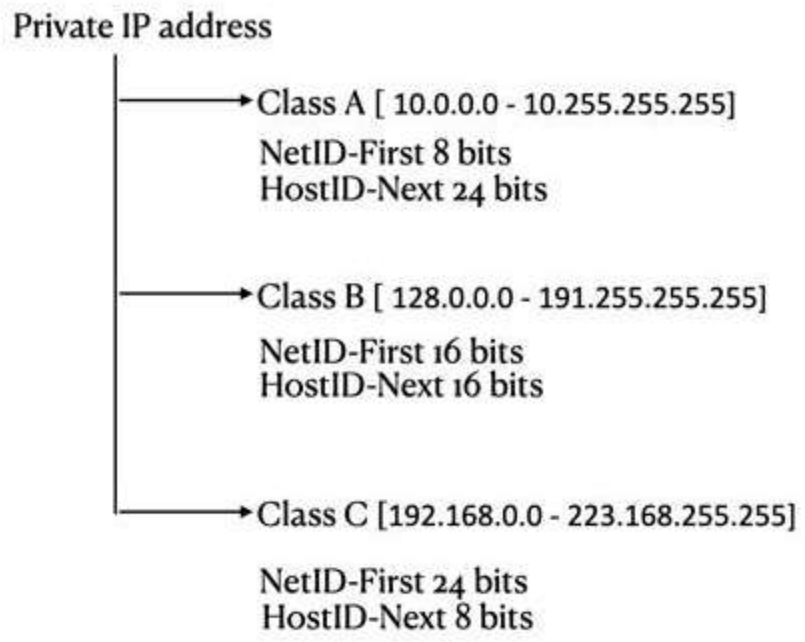
Algorithm: Domain Validation Process

```
Input: A string input (it might be a domain or canonical name)
Output: Validation status, associated domain, canonical name, and IP address
Check whether the input is a registered domain or canonical name:
If domains[input].owner =! null:
domain ← input.
Else if cnameToDomain[input] =! null :
domain ← cnameToDomain[input].
Else:
return (false, "Input is neither a registered domain nor a canonical name", "", "").
Obtain the domain record:
```

Algorithm for validation

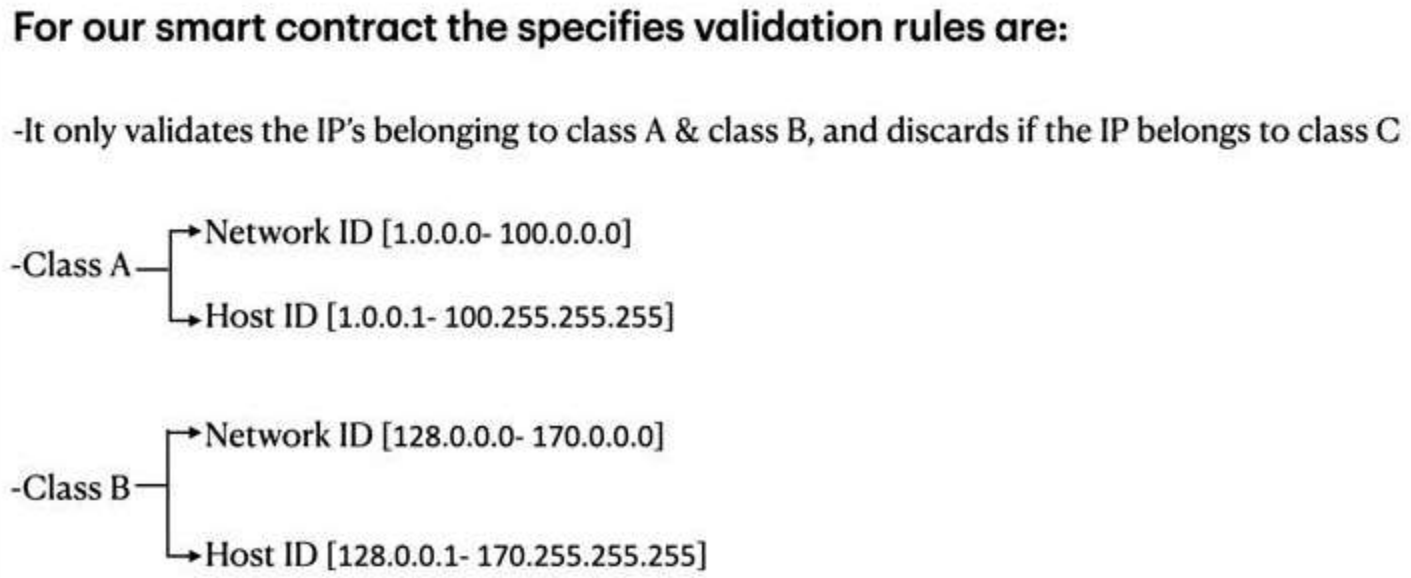
```
domainRecord ← domains[domain].
Check if the domain has expired:
If CURRENT_BLOCK_TIME > domainRecord.expiryTime:
RETURN (false, "Domain has expired", "", "");
Validate IP address class:
Validate Domain hash value
If(match) then
Set ipClassValidation ← validateClassAorB(domainRecord.ipAddress).
If ipClassValidation ≠ "Valid Class A IP range" and ipClassValidation ≠ "Valid Class B IP range":
RETURN (false, ipClassValidation, "", "");
Return the validation result:
RETURN (true, domain, domainRecord.canonicalName, domainRecord.ipAddress).
```

Range of private IP addresses for different classes



Private IP address

- Class A [10.0.0.0 - 10.255.255.255]
  NetID-First 8 bits
  HostID-Next 24 bits
- Class B [128.0.0.0 - 191.255.255.255]
  NetID-First 16 bits
  HostID-Next 16 bits
- Class C [192.168.0.0 - 223.168.255.255]
  NetID-First 24 bits
  HostID-Next 8 bits



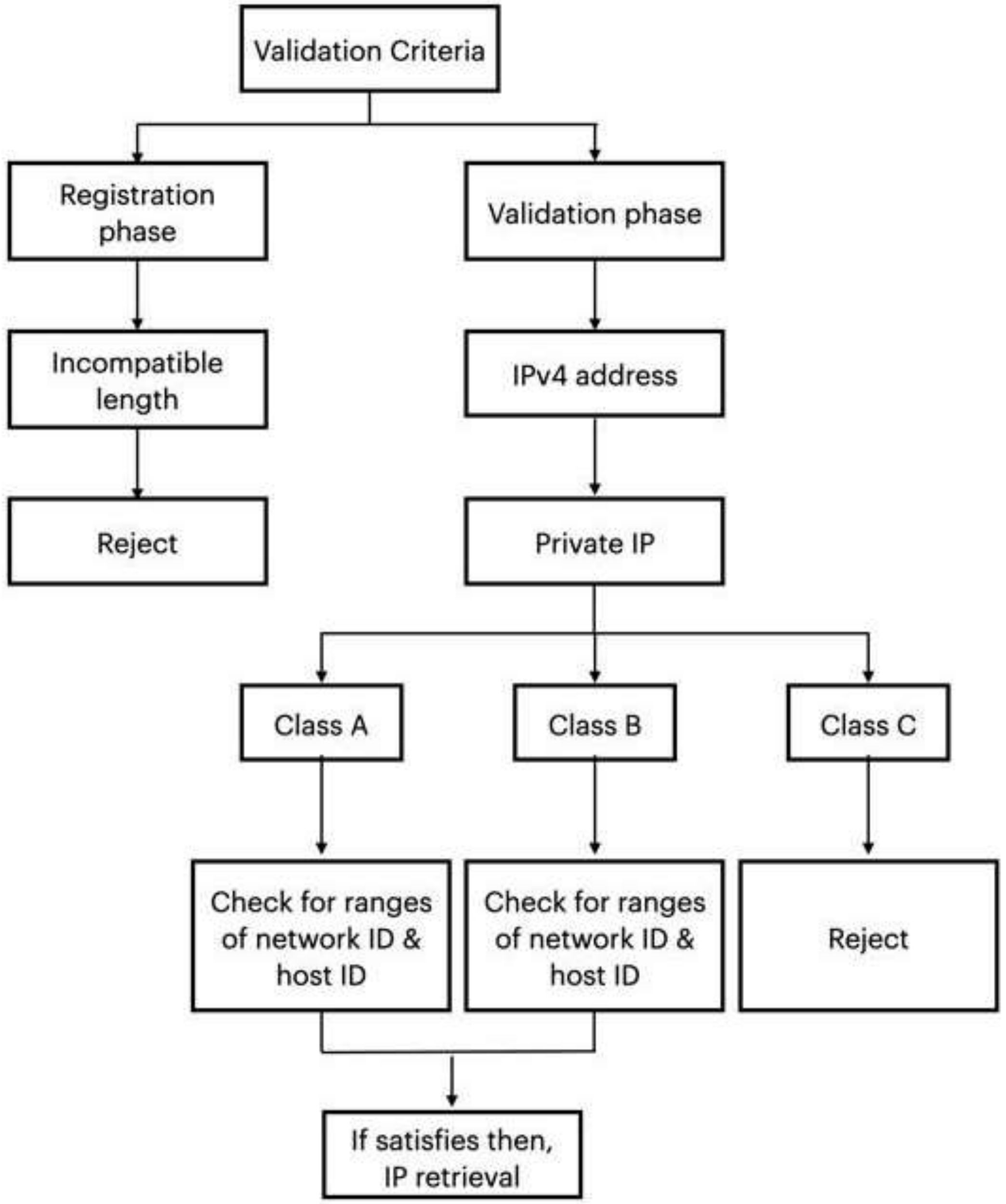
For our smart contract the specifies validation rules are:

-It only validates the IP's belonging to class A & class B, and discards if the IP belongs to class C

-Class A
- Network ID [1.0.0.0- 100.0.0.0]
- Host ID [1.0.0.1- 100.255.255.255]

-Class B
- Network ID [128.0.0.0- 170.0.0.0]
- Host ID [128.0.0.1- 170.255.255.255]

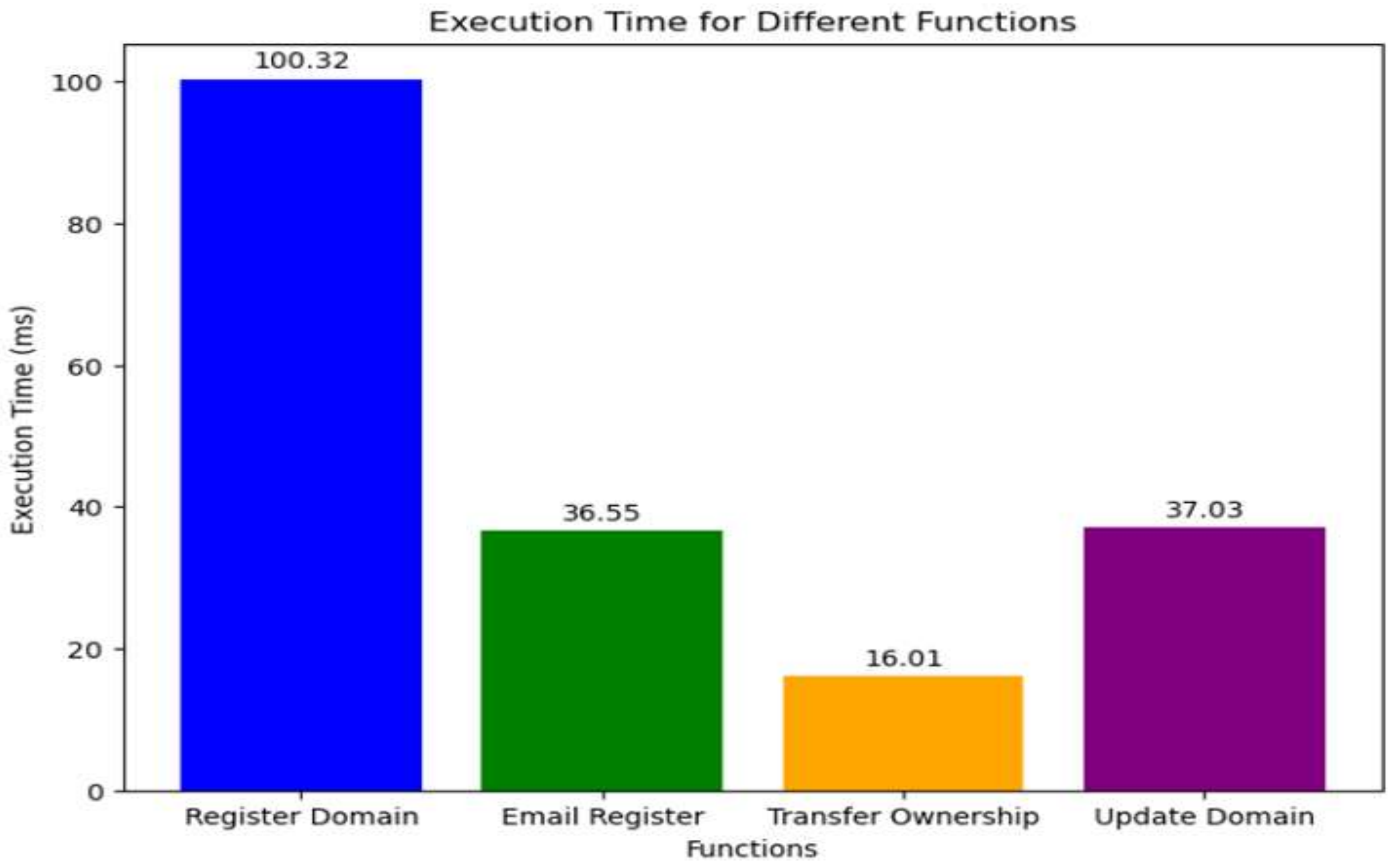
Validation rules



Testing

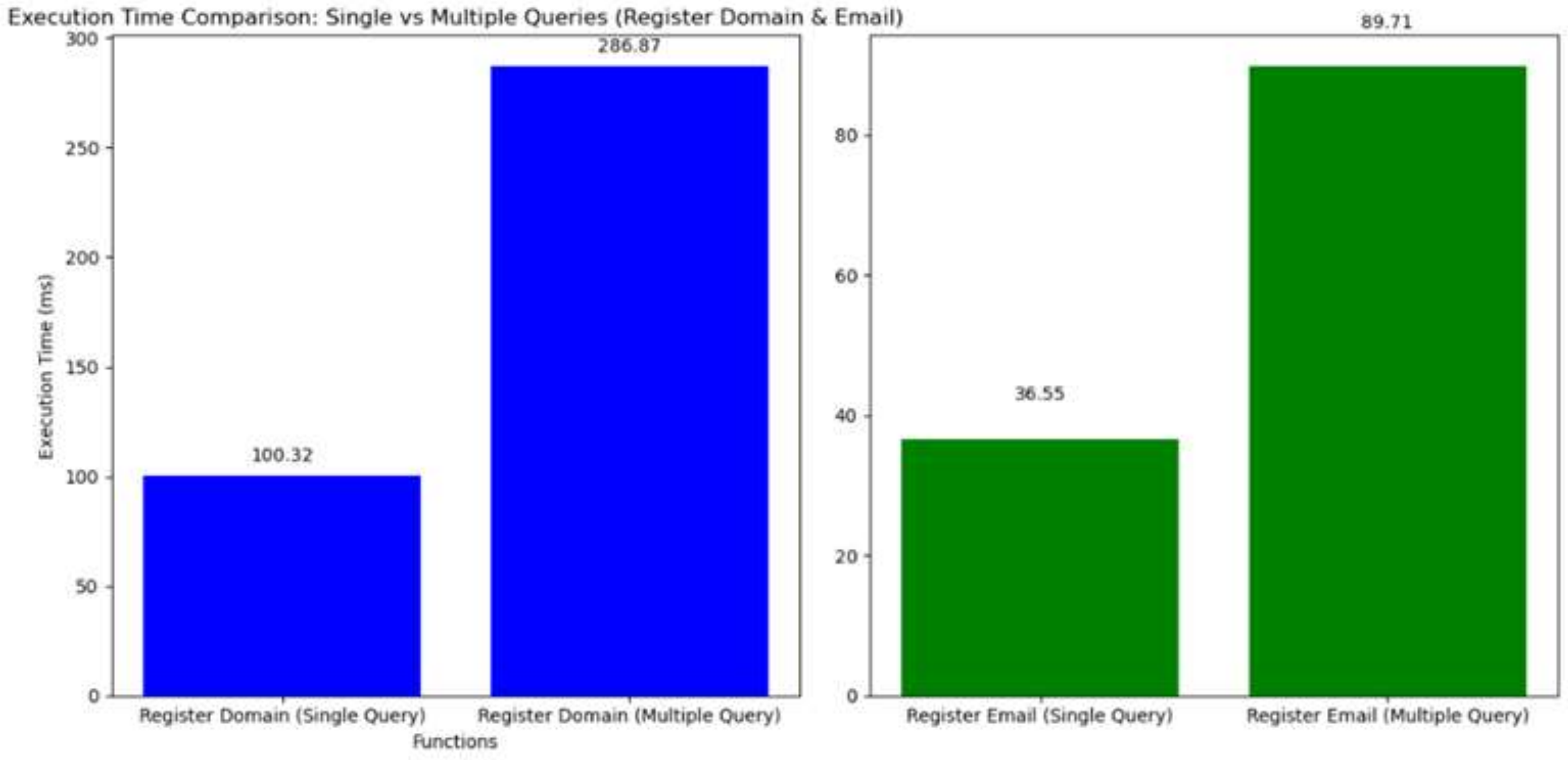
Test Case ID	Function	Description	Inputs	Expected Result
UTC-01	validateIPv4Address	Test IP vali dation logic	192.168.1.1, 256.256.1.1	Returns true for valid IPs, false for invalid ones.
UTC-02	generateDomainHah	Test domain hash generation	"example.com", "1.1.1.1"	Returns a consistent hash for the same in put.
UTC-03	is Alphanumeric	Test character validation	’a’, ’1’, ’-’, ’@’	Returns a consistent hash for the same in put.
UTC-04	validateClassAorB	Test Class A/B IP vali dation	1.0.0.1, 200.0.0.1	Returns valid/invalid class range information.

Evaluation of time for functions



Execution Time for Function to deploy in testnet

Evaluation of time for functions



Single Query vs Multiple Queries

Result Discussion

- The Blockchain-Powered Decentralized DNS was successfully implemented using Remix IDE and deployed on the Ethereum blockchain with MetaMask integration.
- IP address validation was implemented for Class A and Class B ranges, rejecting invalid or out-of-range addresses.
- Ownership verification was enforced for domain modifications and seamless support for domain ownership transfers, ensuring decentralized control.
- The system facilitated the registration of email addresses linked to IP addresses, requiring organizational email domains like @kletech.ac.in.

Result Discussion

- Security measures included ownership checks, expiration checks, and keccak256 hashing for domain-IP combinations, ensuring tamper-proof unique identifiers.
- The contract was optimized for gas efficiency through minimal storage use and event-driven operations.
- Remix IDE provided efficient and user-friendly performance with real-time contract compilation, fast execution on test networks like Rinkeby and Sepolia, and reliable gas usage estimates.

References

1. Z. Li, J. Chen, F. Lin, C. He, and X. Cheng, *"B-DNS: A secure and efficient DNS based on the blockchain technology*," IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 1674–1686, 2021.
2. F. S. Ali and A. Kupcu, "*Improving PKI, BGP, and DNS using blockchain: A systematic review,*" arXiv preprint arXiv:2001.00747, 2020.
3. Y. Shen, J. Zhang, X. Li, and S. Sun, "*DNS service model based on permissioned blockchain*," Intelligent Automation and Soft Computing, vol. 27, no. 1, pp. 259–268, 2021.
4. W.-B. Hsieh, J.-S. Leu, and J.-I. Takada, "*Use chains to block DNS attacks: A trusty blockchain-based domain name system*," Journal of Communications and Networks, vol. 24, no. 3, pp. 347–356, 2022.
5. T. Gao and Q. Dong, "*DNS-BC: Fast, reliable and secure domain name system caching system based on a consortium blockchain*," Sensors, vol. 23, no. 14, p. 6366, 2023.

Thank You