# CS438: NameCoin on Peerster

Rassene M'sadaa
rassene.msadaa@epfl.ch

Oussama Gabouj
oussama.gabouj@epfl.ch

Ahmed Abdelmalek
ahmed.abdelmalek@epfl.ch

## I. Executive Summary - 1 page (strict)

**Project background** The Domain Name System (DNS) is an essential part of the Internet. It converts domain names that people can read into IP addresses. However, traditional DNS relies on centralized authorities, which makes it susceptible to censorship, abuse of authority, and single failure points. Decentralized DNS systems, particularly those that utilize blockchain technology, tackle these vulnerabilities by distributing control across a network of users. This method improves security, reduces the likelihood of outages, and offers increased resistance to censorship, resulting in a more resilient and independent Internet framework.

**Project description** This project extends the efforts of Homework 1, which created a gossip-oriented broadcasting and anti-entropy system for peer synchronization within the Peerster decentralized protocol. We use the codebase of Oussama as the foundation of the project. Expanding on this, the project incorporates a blockchain for domain registrations, updates, transfers, and expiration, using a Proof-of-Work (PoW) method for security. Clients can create, update, or transfer domains by transmitting transactions to the server, where they are validated by the transaction and blockchain layers. Finally, the DNS layer processes client requests for domain resolution.

**Main building blocks of the project** The initiative comprises four primary layers:

*Peerster Layer:* Enables gossip-based dissemination and anti-entropy for effective peer synchronization within the decentralized network.

*Blockchain Layer:* The Blockchain Layer oversees domain registration, modifications, transfers, and expiration by employing a Proof-of-Work (PoW) consensus method for security. This layer ensures that all transactions are permanently documented in blocks, offering a clear and unchangeable record and guaranteeing the integrity of the domain system.

*Transaction Layer:* The Transaction Layer verifies and handles transactions. Verifies elements such as domain ownership and transfer of name domains before mining legitimate transactions on the blockchain, ensuring safe domain activities.

*DNS Layer:* The DNS Layer connects with clients to resolve domains. Checks the status of the domain by querying the blockchain and obtains accurate domain information, ensuring secure and precise handling of domain requests.

**Component descriptions** In our design, there are 4 types of messages. Between Miners: *BlockChainMessages* containing a new block mined and N previous blocks. Between Miners and Clients: *TransactionMessages* broadcasted by a client when he needs a new transaction, this message will be answered by *TransactionAckMessages* when the block containing the transaction is added in the blockchain. And finally *DNSRequestMessage* and *DNSResponseMessage* for classic requests of IP address of a given domain name. For consensus, a miner always accepts a valid, longer blockchain and replaces blocks from the point of divergence (resolving forks). If a received BlockChainMessage sequence is too far ahead, the miner saves it for later processing. If a new block completes this sequence, it is processed; otherwise, outdated sequences are ignored. When no correspondence is found, it indicates an excessively long fork, which requires determining the optimal number of blocks per BlockChainMessage.

**Implementation summary**: We faced various obstacles to guaranteeing the system's correctness. Without a Certificate Authority (CA), we directly associate IPs with public keys, facilitating secure identity validation. Managing forks posed a significant challenge, which we addressed by implementing a mechanism for block addition and rollback that ensured new chains were validated while invalid ones were eliminated. Moreover, future blockchain messages that are not associated with the peer blockchain modifications were handled via an indexed pool, preventing duplication and enabling efficient retrieval and validation of branches. Finally, confirmations are delayed until k blocks are mined. This ensures all miner's blockchain converge to the same chain.

**Testing and Evaluation Summary** Our system underwent three levels of testing:

1) **Unit Tests:** We tested individual methods within modules (*BlockChain*, *Crypto System*, *DNS*, *Transactions*) to ensure correct behavior and resistance to edge cases for each seperate component.
2) **Integration Tests:** We integrated all functionalities in these tests by validating the Proof-of-Work mechanism and ensuring full end-to-end operation for domain tasks such as registration, updates, expiration, and resolution. We also assessed the system's resilience to churn, catch up and Byzantine attacks.
3) **Performance Tests:** We evaluated the mining and domain operation throughput, consensus times with varying node counts, and network overhead for domain updates to ensure efficiency.

## II. Related Work

The Domain Name System (DNS) serves as the backbone of the Internet, translating human-readable domain names into machine-friendly IP addresses. Traditional DNS systems rely on centralized entities, which introduce critical weaknesses such as single points of failure and censorship risks.

To address these limitations, decentralized DNS (dDNS) systems were introduced. In particular, Namecoin [1] pioneered the use of blockchain technology for domain registration and management. By leveraging the blockchain's decentralized ledger, Namecoin eliminates the issues mentioned above. Google researchers have explored dDNS alternatives, highlighting their potential benefits and limitations. For example, studies such as Blockstack [2] and BlockDNS [3] highlight the importance of balancing decentralization with scalability, performance, and security in dDNS networks.

Blockchain technology was first introduced in Bitcoin [4], which implements a decentralized ledger secured through the Proof-of-Work (PoW) consensus mechanism. PoW ensures that miners verify transactions by expending computational resources, creating a secure and tamper-proof system.

Despite its advantages, blockchain systems face various security threats. Gervais et al. [5] analyze key vulnerabilities, including 51% attacks and selfish mining, which threaten the integrity of PoW networks. Similarly, decentralized DNS systems are not immune to threats. Zhang et al. [3] discuss risks associated with domain hijacking and consensus failures in dDNS networks, emphasizing the need for robust protocols to mitigate such vulnerabilities.

## III. Threat Model

In this design, the attacker aims to disrupt the system through several methods. In the following, we outline these threats, their potential impacts, and the protections or limitations of our implementation.

**Injecting False Transactions:**

- *Goal:* Inject a fraudulent transaction to gain money or a DNS name without payment.
- *Mitigation:* Transactions are secured by authentication and integrity through cryptographic signatures. Invalid transactions are rejected by miners.

**Disturbing System Behavior:**

- *Man-in-the-Middle (MITM) Attack:*
  - For Miners: The attacker impersonates a client or miner to inject invalid blocks or transactions. This has no impact, as miners validate blocks and transactions against their blockchain, rejecting invalid ones. Message integrity and authenticity are ensured through hashes.
  - For Clients: An attacker may intercept and modify private messages such as transaction acknowledgments or DNS responses. But as we take a threshold of these messages to confirm them, our model is resistant.
- *Packet Dropping:* As they receive messages via broadcast, dropped packets are recovered through the anti-entropy protocol.
- *Overwhelming the Network:*
  - *Goal:* Flood miners with excessive messages to hinder performance.
  - *Mitigation:* Miners can quickly detect and reject invalid blocks in `BlockChainMessages`. However, advanced `BlockChainMessages` may lead to increased memory usage, reducing system efficiency under sustained attacks.

**51% Attack:**

- *Goal:* Control the majority of mining power to manipulate the blockchain, allowing double spending, censorship, or rewriting the transaction history. With majority control, an attacker could create and propagate fraudulent blocks that overwrite legitimate transactions and prevent new transactions or DNS updates from being added to the blockchain.
- *Limitations:* Our design assumes that no single entity or coalition can control more than 50% of the mining power. However, if this assumption is breached, the system cannot guarantee integrity.

## IV. Design

### A. Peer Types and Data Structures

We have two distinct peers and two primary data types:

- **Peer Types:**
  - Miners: Responsible for mining blocks, processing transactions, and resolving DNS requests.
  - Clients: Interact with miners to perform transactions and DNS queries.
- **Primary Data Types:**
  - `UTXOToken` includes: **Amount** (token value), **Transaction Signature** (creating transaction), and **IsNew** (boolean flag for new tokens).
  - `Domain` includes: **Name** (domain name), **Expiration Block** (expiry block number), and **IPAddress** (domain owner's IP).

### B. Message Types

Our design includes four types of messages:

1) **Between Miners:** `BlockChainMessages` contains a fixed number of blocks. Each block includes: **Index**, **Previous Hash**, **TransactionMessages** (constant number), **Merkle Root**, **Nonce**, **Timestamp**, **BlockHash**, and **Miner IP** (proof of mining ownership).
2) **Between Miners and Clients:** `TransactionMessages` include: **Type** ("name new", "name firstupdate", or "name update"), **Domain**, **Salted Hash**, **Salt**, **Sender IP**, **Receiver IP**, **Inputs** (UTXOs to process the transaction), and **Outputs** (UTXOs for receiver price, sender change, and miner fee). `TransactionAckMessages` include: **Type**

(of the transaction), **Owner IP**, and **UTXOs** (money received).

3) **For DNS Requests (between Miners and Clients):** `DNSRequestMessage` includes: **DomainName** and **Owner IP** (requester's IP). `DNSResponseMessage` includes: **DomainName** and **Owner IP** (or empty if the domain is available).

### C. Miners Workflow

When receiving a **DNSRequestMessage** Figure 1, The miner searches for the requested domain in the blockchain. If the domain is not owned or is expired, an empty DNSResponseMessage is sent back. However, if the domain is valid and not expired, the miner retrieves the corresponding IP address and sends it in the DNSResponseMessage.

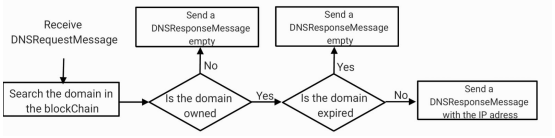


Fig. 1: When receiving a DNS Request

When receiving a **TransactionMessage** Figure 2: The new transaction is verified as shown in Figure 3. Once enough transactions are received, a block is mined, and a BlockChainMessage is broadcast. If a new block is received during mining, the mining process is canceled.

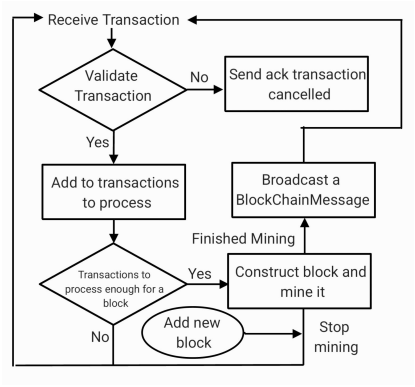


Fig. 2: When receiving a transaction

When receiving a **BlockChainMessage** (Figure 4), if the index of the received blocks is:

- Higher than 2 indexes beyond the last block in the blockchain, the blocks are added to a pool for future processing.
- Equal to or older than the last block in the blockchain, the blocks are discarded.The received blockchain is outdated.
- Otherwise, we detect a fork. We Then identify the most common recent block in both chains. Then, we cache the peer's blockchain and attempt to add the new longest chain by validating it block by block. If the entire chain is valid, we adopt the cached blocks and verify that the transactions match the new longest chain. If they do, we proceed. Otherwise, we return the invalid blocks to the pool. If validation fails, we discard the longest chain and revert to the peer's chain.

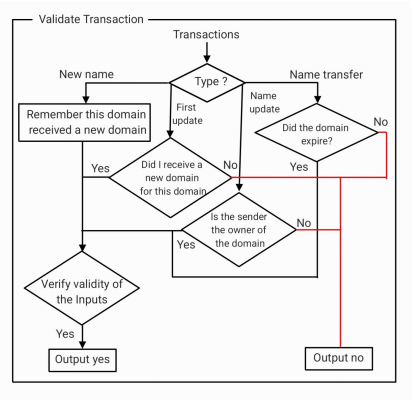


Fig. 3: Validate transaction

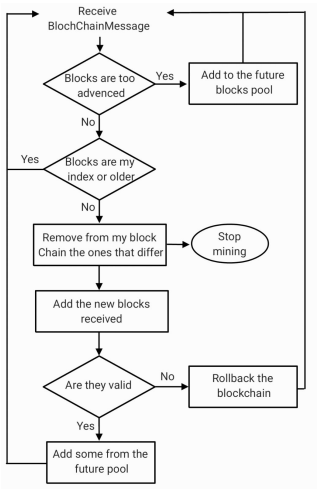


Fig. 4: When receiving a BlockChainMessage

### D. Clients Workflow

1) A client can add a new domain by sending a transaction of type "new name." Once the acknowledgment is received, the client sends a transaction of type "first update."
2) A client can update a domain they already own after it has expired by sending a transaction of type "name update."
3) A client can purchase the domain of another client by paying through a transaction of type "name transfer."
4) A client can send a DNS Request to miners, and receive a DNS Response indicating the IP address of the domain owner, or empty if the domain is not owned or has expired.

**Here is an example of scenario** (Figure 5):

## V. Implementation

During our implementation, we encountered multiple issues that we had to find a solution for.

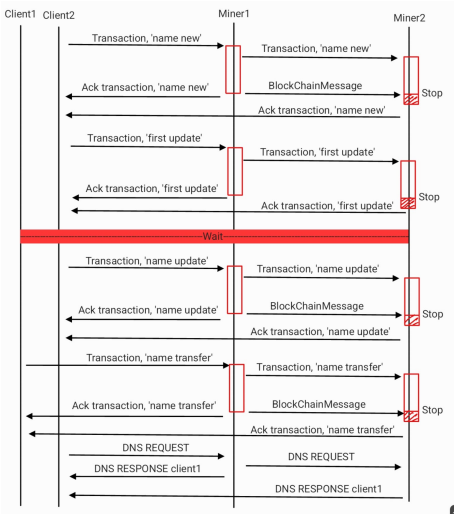


Fig. 5: Scenario

**Absence of a CA** In our implementation we noticed that we needed a trusted CA to provide us with the public key of each peer. Our first choice was to send in the transaction the full public key of the concerned parties, but we noticed that after marshaling and unmarshaling the public key was modified enough so that ".equal" doesn't work anymore. We decided to provide a map to each node in the initialization with all the IP addresses to the public keys.

**Handling Forks in consensus process** As it happens that we receive a blockchain that is ahead of ours, we will have to take it as our own (Figure 6).

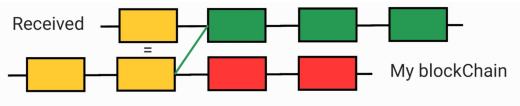


Fig. 6: Solving a fork when a BlockChainMessage is received

However, this raises three issues:

- If the chain to be added is valid, we must manage the transactions. The transactions in the blocks removed from our blockchain must be re-mined, while those from the received chain must be discarded.
- To verify a block in our implementation, it must first be added to the blockchain. This means that to verify the entire received chain, we must temporarily add all the blocks. However, if the last block fails the validity check, we must roll back to our original blockchain. To address this, we've implemented a security mechanism that uses temporary caching, ensuring the real cache is only updated once the entire chain is validated.
- To ensure that the client only receives transaction acknowledgment once their block is part of the majority blockchain, the miner will send the acknowledgment only after K additional blocks have been added. The client will then confirm the acknowledgment only if it is received from at least N miners.

**Managing the advanced blockChainMessages received** when a miner arrives late and has to catch up a lot of blockChainMessages and that he has to store them knowing that there will be a lot of duplicates. We created a type of pool that is a map from the index to a list of blocks, when we add a chain to the pool only the non-duplicated blocks will be added in the good index key. To retrieve them we will have to

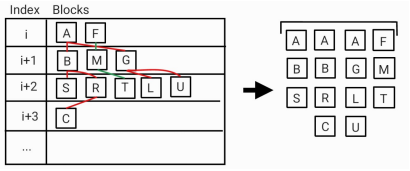


Fig. 7: Storing and retrieving branches from the pool

choose the longest possible branch from that index and test its validity, if this branch is not valid, we try the next longest. To achieve this we did a recurrent algorithm that retrieves from this pool all the possible branches of blockChain and puts them in order. Like figure (Figure 7)

## VI. Testing

This section outlines the unit and integration tests conducted for different components of the system. We focused on implementing unit tests to ensure each function's correctness and then we implemented the integration tests to validate the system's interaction. Our tests achieved 73.4% coverage, and an HTML coverage report is available on the CI.

### A. Unit Testing

These tests focus on the key features of each layer.

**Blockchain Layer** The Blockchain layer includes features related to block creation, hash calculation, and maintaining the integrity of the chain. We wrote unit tests for the following features: *Merkle Tree Creation and Verification*, *Block Hash Calculation*, *Adding and Retrieving Blocks*, *Fetching the Latest Block*, *Proof-of-Work Mechanism* and *Test the pool of advanced block Chains*

**Transaction Layer** The Transaction layer is responsible for handling user transactions, ensuring their validity, and managing their inclusion in blocks. The unit tests focused on: *Transaction Validation:* Tests to verify that each transaction has the amount of Inputs equal to outputs and that the Inputs come from a other transaction's output in the Block Chain and *Transaction Hashing for integrity*

**Domain Services Layer** The Domain Services layer handles the core business logic, including domain transfers and state transitions. The tests included: *Domain Transfer Validity:* Tests to ensure that domain ownership is correctly transferred and *State Transitions:* Tests to verify that the clients send the type of messages in good order.

*1) Cryptography Module:* The Cryptography module is essential for securing transactions and data. The unit tests for this module focused on: *Encryption and Decryption*, *Digital Signatures*, and *Hashing Functions*.

### B. Integration Testing

Integration tests were designed to validate the interaction between the different layers and ensure that they work together as expected. The following key scenarios were tested:

**Multiple Clients and Miners Handling Transactions:** This test simulates multiple clients submitting new domains concurrently while miners process these transactions. The goal is to ensure that all miners maintain consistency by having the same transactions in their blocks.

**Consensus and Block Validation:** This test focuses on ensuring that when new transactions are broadcasted, we have a valid consensus mechanism by comparing blockchains across miners to ensure they match and converge.

**Persistence of Transactions During Churn:** This test simulates node churn, where some miners disconnect and some clients ask for the domain. This verifies that no transactions are lost and that the blockchain is consistent during node drops.

**Catch-Up Mechanism:** In this test, late-joining miners are simulated to ensure they can catch up with the latest block . This validates that all miners, regardless of when they join the network, can synchronize with the current blockchain state.

## VII. Evaluation

We evaluated the performance of our system using four distinct tests as follows:

**Mining Latency and Throughput** The test initializes a set of miners and clients, simulates domain registrations, and ensures that after the blockchain stabilizes, all nodes have the same number of blocks, with the final block being consistent across all nodes. We extract all blocks, record the mining duration for each block, and compute the total mining duration, average block latency, and throughput for different difficulties.

**Transaction Latency and Throughput** In this test, a predefined number of nodes are set up, and the number of clients is varied. Each client generates two transactions, which are propagated across the network. We ensure that every node in the network has the expected number of transactions in its blockchain. Once this condition is met, we record the total time and compute the average latency and throughput.

**Consensus Mining Latency and Throughput** The test configures varying numbers of miners and clients, simulates domain registration transactions and monitors the time for all miners to achieve consensus.

**Network Overhead, Bandwidth, and Latency** In this test, we vary the number of clients and miners. For each combination, each client generates two transactions. We compute the total number of packets sent across the network by all nodes, categorizing them by message type. Additionally, we calculate the total data transmitted and bandwidth usage.

Here, we present the results of our performance evaluation. These results are derived from the performance tests and extracted from the HTML file generated by the CI system.

Table I summarizes latency data (MB/s) for different domain and miner configurations. It highlights a general increase in latency with higher miner and domain counts due to increased synchronization complexity.

| Domains vs miners | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|
| 5 | 0.14 | 0.15 | 0.04 | 0.10 | 0.16 |
| 10 | 0.11 | 0.10 | 0.08 | 0.07 | 0.25 |
| 15 | 0.09 | 0.12 | 0.10 | 0.03 | 0.79 |
| 20 | 0.09 | 0.30 | 0.32 | 0.32 | 0.37 |

TABLE I: Latency data for different numbers of domains

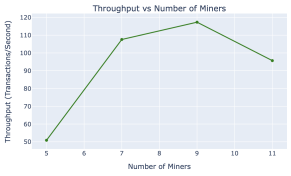


Fig. 8: Image 1

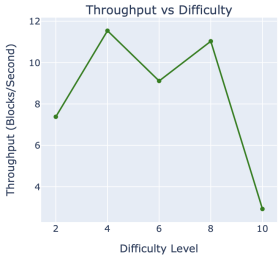


Fig. 9: Image 2

From figure 8, we can see that throughput increases with the number of miners, peaking at 8 miners, and then declines slightly, indicating optimal miner configurations. Figure 9: The throughput decreases significantly with higher mining difficulty, highlighting the need to balance mining difficulty (security) and throughput for optimal performance.

These results collectively illustrate key trade-offs and optimal configurations in system performance.

## VIII. Limitations & Future work

**Creating UTXOs for Clients:** To simplify the creation of initial UTXOs for clients, each UTXO includes a boolean flag set to `true` if it is newly created. This allows miners to bypass transaction history verification for these UTXOs, streamlining testing and implementation. However, this approach is inherently insecure and not suitable for production systems.

**Handling Duplicate Domain Names:** If two clients attempt to create a domain with the same name (via a ”new name” transaction) before performing a ”first update,” the miner cannot detect the conflict. This is because the domain names are salted and hashed, making them indistinguishable.

**Block Consensus Mechanism:** Miners achieve consensus by sending a fixed number of blocks per `BlockChainMessage`. If the received blockchain is longer, the miner adopts it. However, if the fixed number is insufficient to resolve a long fork, the miner may fail to update its blockchain. While sending partial blockchains looks less efficient than transmitting the entire chain, dynamically adjusting the block count based on bandwidth could improve efficiency in the long run.

**Trusting Clients for Domain Transfers:** For domain transfers, the system trusts the transferring client and does not require explicit consent from the seller, which could lead to potential misuse.

## REFERENCES


[1] "Namecoin: A decentralized naming system," in Proceedings of Namecoin. [Online]. Available: https://namecoin.org

[2] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in USENIX Annual Technical Conference, 2015, pp. 181–194. [Online]. Available: https://www.usenix.org/conference/atc15/technical-sessions/presentation/ali

[3] J. Zhang, J. Chen, T. Yu, and W. Chen, "Blockdns: A secure and efficient decentralized dns," in Proceedings of the IEEE International Conference on Blockchain, 2018, pp. 1420–1427. [Online]. Available: https://ieeexplore.ieee.org/document/8451414

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] A. Gervais, G. O. Karame, H. Wüst, V. Glykantzis, D. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 3–16. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978341