CS-438 - Decentralized Systems Engineering, Fall 2025

Group Project

Choose your own adventure !

Author	DEDIS lab, EPFL

Revision	October 14, 2025 - 1.3.0
Publish date	Friday, October 14, 2025
Project start	Monday, November 18, 2025
Due date	**Friday, December 19, 2025 @ 23:55**
	(Tolerance until December 22 @ 23:55)

Table of contents

Introduction

End result

As you've now built Peerster to be a small but complete decentralized system featuring gossiping, data sharing and consensus, you have gained the experience to develop decentralized systems with less guidance and more freedom. In short, it is time to choose your own adventure !

You will be working in groups (3-4 students) building a decentralized or distributed system on top of peersters, based on one or more research papers. This document will present many potential projects for you to choose from, but you can also elect to pursue your own project idea (see Project Tracks).

Whichever project you pick, by the end of the semester, you will have:

- implemented a decentralized system and designed a good testing suite for it
- evaluated your implementation's performance and limitations
- written a project report, including a brief literature review, your system's threat model, architecture and design, testing strategy, its evaluation and limitations.
- written an individual report (max 1 page) presenting your contributions as part of the project team
- a documented repository with a Makefile allowing us to run your system, execute your testing suites and reproduce your evaluations' results.

Objectives

Understand, implement, test and evaluate a non-trivial decentralized protocol or system, based on the existing literature and with limited guidance, leveraging what you have learned throughout the class and the code you've developed during the semester.

Grading

The group project of CS-438 accounts for 30% of your final grade. Your project grade will depend both on your individual and group performance. If your individual grade is passing, the group and individual grades will count for 50% of the project grade each, otherwise the grade will be your individual grade. In short, we aim to have a fair and just evaluation, rewarding good contributions to the teams' efforts and, conversely, ensuring that free-riding or slacking behaviour is severely penalized.

As such, we look at individual contributions to the overall project implementation and testing work, but also at effort that went into team coordination, system design, the writing of the final report, etc. To obtain a good grade, make sure that you contribute effectively to your team's project: (1) complete your individual contributions on time, (2) make sure that

your team regularly meets and completes the work on time, and (3) make sure you deliver all the expected artifacts on time, and done to the best of your abilities.
You will find a detailed grading rubric below, but please remember that “there are no winners in a losing team”.

Group (50%):

Architecture & design	5%	The team has identified the key components and how to integrate the components together, and documented it.
Implementation / Functionality	15%	The implementation is complete with respect to the project goals, it is robust and with good code quality. The CS-438 staff can use the software and it works. ⚠️ Just being able to run tests is not sufficient!
Testing	10%	The system is tested at the unit and integration level, addressing a few/the most important/almost all failure scenarios, with an appropriate level of test coverage. The CS-438 staff can run the tests and they pass.
Evaluation	10%	The system is evaluated across (some/most/all) relevant metrics with a sound experimental setup. The graphs and results provided in the report are reproducible based on the Makefile and README.
Final report (group)	10%	The final report provides an overall, accurate overview of the project, following the provided report template. Architecture & design, evaluation are graded separately.

Individual (50%):

Individual contributions	20%	The extent and quality of your contributions to the overall project’s implementation, testing, evaluation and report.
Teamwork	15%	Your degree of involvement in the project and its timeliness, including in support of your team. We take into account your team’s individual reports, your team’s interaction with the TAs and evidence from your code repository.
Individual report	15%	Your short (max 1 page) individual report, detailing your contributions to the project and how you collaborated with your team.

Project Phases

Phase 1: Group formation

As the first phase of the group project, each student is expected to form a group with 3–4 members. Please indicate your group on the Moodle group formation link. In this document, we provide you with different group project ideas. Go through each topic, and select the project that your team would like to work on. Make sure that the expected number of students indicated in each project matches the number of members in your group. Only one member per group marks the project preference on Moodle. Note that more than one group can select a given project topic.

We encourage you to form groups as soon as possible.

Phase 2: Project definition and planning

As the second phase of your project, first read the project description and the associated papers carefully. Discuss among yourself the best possible design. Note that we expect that your project builds on top of the Peerster / improves the Peerster in some non-trivial manner. You can discuss with the TAs and get feedback to improve your project design.

Once you have a reasonable design of your project, then start the implementation and evaluation, where each group member will work on a different part of the project. Keep in mind that you will be expected to implement the distributed/decentralised building blocks yourself, without the help of external libraries, though you may be allowed to use external libraries subject to TA approval.

As part of your planning, we strongly encourage you to set goals for yourselves at each week of the five project weeks. As an example of what this might look like:

Week 1
- *Code skeleton is in place; Peerster integration points are defined*
- *First system tests are in place (we expect most will be failing)*

Week 2
- *Features X and Y are (partially) implemented and integrated with one-another*

Week 3
- *Basic implementation of the features is finished (happy paths)*
- *Initial (rough) evaluations (this will help us detect bugs) & debugging*

Week 4
- *Implementation of the features is finished (all corner cases are addressed)*
- *Most integration & system tests passing, first (complete) draft of report*

Week 5
- *Bug fixing, integration & system tests finished*
- *Report writing, incl. final evaluation*

You need to have a well-defined, TA approved project at the latest one week before the start of the project.

Phase 3: Project execution

This phase will take place in the last 5 weeks of the Fall semester, once your homework 3 is completed. During this phase, you will be carrying out the project. TAs will be available on a weekly basis to accompany and guide you as necessary. At the end of this phase, you will be expected to submit your code and reports.

Code Repository

Once your team is registered, you will receive a GitHub repository with write permissions for all members. After the deadline, write access will be revoked.

Build your project on an existing Peerster implementation from one member, and document (both in the report and the README) whose implementation was chosen as the foundation for the project.

Ensure you have a plan to distribute work between yourselves and that you clearly divide responsibilities amongst yourselves. Make sure to commit all changes to the repository and to set your Git author and e-mail correctly to reference your full name and EPFL e-mail address, this will enable us to assign authorship correctly.

Ensure the code builds with "go build", or a Makefile if necessary, document all the dependencies and provide all the necessary scripts for the CS-438 staff to reproduce your results.

Properly structure the project repository, aim for at least 70% test coverage, and include thorough documentation. Failure to meet these requirements will make you lose points.

Final Group Report Format

The group report we expect will have to adhere to the following format, and a Latex template will be provided for your convenience.

File Name: [group_number]_final_report.pdf
Page limit: 5 pages
Font: Times New Roman, 10pt

Overall structure

- **Introduction** Provide background and summary of the project.
- **Related Work** Discuss the 5 most relevant published works related to your project.
- **Threat Model** Describe the assumed threat model for the system.
- **Design** Present the main components of your system and how they interact.
- **Implementation** Present the main implementation choices in your system.
- **Testing** Summarise the testing strategy to ensure the system's correctness.
- **Evaluation** Present the main results and include graphs to support your findings.

- **Limitations & Future Work** Limitations of your work and how they could be addressed.

Individual Report

The individual report (max 1 page) will need to present succinctly your contributions to the project, in terms of development work and teamwork, as well as your interactions with the rest of the team: how did you organize? Were you pair-programming? Who was leading the team? etc.

Do not hesitate to include what you learned in the project and any feedback for us (max 2-3 lines). We won't take it into account for grading, but we'll use it to improve the class.

Tools to support you

You may find some of the following tools to be helpful as part of your project:

- Simulating adversarial networks:
  Linux NetEm

- Running lots of nodes:
  Docker, Kubernetes

- Visualising messages:
  Polypus (included in Peerster)

- Monitoring / graphs:
  Prometheus + Grafana

- Load testing:
  JMeter, Gatling, Locust, Artillery

- Property-based and generative testing:
  Rapid (Go library), Jepsen

Project tracks

In this section, we will list the set of projects. Please go through the entire list, and select a project that your team wants to work on. Make sure that the number of members in your team matches the expected number of students for that project. Should you want to bring modifications to the project, including to add an extra member, you will need to get those changes approved by a TA.

If you choose to propose your own project, start defining it early, iterate on it with your team and your teaching assistants (this can take up to 5 weeks), and make sure to have it fully defined, endorsed by a TA and approved by the lecturers one full week before the start of the project at the latest. Failing to meet this deadline with an adequate project proposal will disqualify the project and you will be expected to do one of the projects outlined hereafter.

As soon as you have studied up on the project and gained sufficient knowledge, please meet your TAs during the Friday exercise sessions, in person, and present your project plan, including how you plan to share the work amongst yourselves. Depending on your plan, the TAs may ask you to adjust some components of your projects, or the distribution of work, to help you ensure its success.

Given the above, do not wait for the last minute if you want to bring changes to the project or propose your own project. Talk to the CS-438 staff early in the semester and give yourself time to iterate on the project definition.

Keep in mind that each project's testing and performance evaluation goals are the requirements used to assess you, and *fully* addressing them puts you on the path to a great grade. However, you may need to expand on them to achieve a good test coverage and a solid evaluation.

PBFT Consensus (3 students)

Description

Practical Byzantine Fault Tolerance (PBFT) is a consensus algorithm designed to provide high fault tolerance and reliability in distributed systems. It achieves consensus through a series of communication steps among nodes, including a pre-prepare, prepare, and commit phase, ensuring all honest nodes agree on the same sequence of requests even in the presence of faulty or malicious nodes.

In this project you will replace the Paxos consensus layer (HW3) of Peerster with PBFT. Replacing Paxos with PBFT makes Peerster robust against the malicious nodes that can commit omission faults and equivocations.

Relevant Paper(s)

- Practical Byzantine Fault Tolerance (Miguel Castro and Barbara Liskov)

Feature Implementation

- ☐ Stable leader based replication protocol
- ☐ Leader election module
- ☐ Group membership change

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Byzantine failures are provoked (and recovered from) as part of end-to-end tests
- ☐ The system's resilience to crash faults at various stages in the protocol is tested
- ☐ Scenarios with nodes churn are tested

Performance evaluation

- ☐ Performance under a stable leader (throughput vs latency)
- ☐ Performance in leader failure scenarios
- ☐ Performance under adversarial network conditions
  (you can use Linux NetEm to simulate adversarial networks)

Hotstuff Consensus (3 students)

Description

HotStuff is a consensus algorithm designed to improve upon traditional Byzantine Fault Tolerance (BFT) mechanisms by enhancing simplicity, efficiency, and performance in distributed systems.

In this project you will replace the Paxos consensus layer (HW3) of Peerster with Hotstuff. Replacing Paxos with Hotstuff makes Peerster robust against the malicious nodes that can commit omission faults and equivocations.

Relevant Paper(s)

- HotStuff: BFT Consensus in the Lens of Blockchain (Maofan Yin, Dahlia Malkhi)
- DiemBFT v4: State Machine Replication in the Diem Blockchain (The Diem Team *)

Feature Implementation

- ☐ Stable leader-based replication protocol
- ☐ Synchronizer module
- ☐ 2 chain and 3 chain replication

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Byzantine failures are provoked (and recovered from) as part of end-to-end tests
- ☐ The system's resilience to crash faults at various stages in the protocol is tested
- ☐ Scenarios with nodes churn are tested

Performance evaluation

- ☐ Performance under a stable leader (throughput vs latency)
- ☐ Performance in leader failure scenarios
- ☐ Performance under adversarial network conditions
  (you can use Linux NetEm to simulate adversarial networks)

QuePaxa Consensus (3 students)

Description

QuePaxa is a novel (2023) consensus protocol designed to overcome the limitations of traditional leader-based algorithms such as Paxos. Unlike Paxos that rely on timeouts to check the leader's progress, which can fail under adverse conditions like network delays or denial-of-service (DoS) attacks, QuePaxa uses a randomised asynchronous consensus core to ensure robustness without sacrificing efficiency.

In this project you will replace the Paxos consensus layer (HW3) of Peerster with QuePaxa. Replacing Paxos with QuePaxa makes Peerster robust against adversarial network conditions.

Relevant Paper(s)

- QuePaxa: Escaping the tyranny of timeouts in consensus (Pasindu Tennage, B. Ford)

Features

- ☐ QuePaxa consensus protocol (requires 2 students)
- ☐ Multi-Armed-Bandit-based leader change

Testing

- ☐ Unit testing and integration testing of both core features
- ☐ The system's resilience to crash faults at various stages in the protocol is tested
- ☐ Scenarios with asynchronous network conditions are tested
- ☐ The impact of hedging delay is tested

Performance evaluation

- ☐ Performance under a stable leader (throughput vs latency)
- ☐ Performance in leader failure scenarios
- ☐ Performance under adversarial network conditions
  (you can use Linux NetEm to simulate adversarial networks)

Secure DHT (3 students)

Description

Peerster implements a distributed searching protocol and a distributed objecting store. The versions of searching and storing protocols you implemented in the homeworks are only safe when all the nodes are “honest” – no node in the network violates the rules in the protocol. However, in real life distributed protocols, security is a common problem. S/Kademlia is a protocol that is secure against malicious node behaviours.

In this project, you will implement S/Kademlia DHT on top of Peerster. You can either keep the existing data store and the search protocol of Peerster, or, remove them.

Relevant Paper(s)

- S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing (Ingmar Baumgart, Sebastian Mies)

Feature Implementation

- ☐ Secure `nodeId` assignment, reliable sibling broadcast and mutual authentication
- ☐ Secure data replication, and routing table management (bucket refreshing and secure node introduction)
- ☐ Secure routing: secure lookup protocol, redundant lookups

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ The ability to add a new node securely
- ☐ The sibling broadcast mechanism
- ☐ Mutual authentication
- ☐ Data replication with at least 5 nodes
- ☐ Routing protocol in the presence of different percentages of malicious nodes

Performance evaluation

- ☐ Fraction of successful node lookups with respect to the path length
- ☐ Fraction of successful node lookups with respect to the number of adversarial nodes
- ☐ Impact of adaptive buffer for the successful node lookups
- ☐ Performance under churn scenarios (% failure, recovery times)

Secure Routing (4 students)

Description

Tor enhances online privacy and anonymity by routing internet traffic through a decentralised network of volunteer-operated servers. Each user's data is encrypted in multiple layers and sent through a series of randomly selected relays, known as nodes. As the data passes through each node, one layer of encryption is peeled off, revealing the next destination but not the entire path or the source. This method ensures that no single node knows both the origin and destination of the data, significantly enhancing privacy.

In this project you will implement onion routing and Tor on top of Peerster. To demonstrate its use, you will integrate Tor with the "private messaging" feature of Peerster, such that the identity of the 2 nodes engaged in a private conversation are not known by other nodes.

Relevant Paper(s)

- Tor: The Second-Generation Onion Router (R. Dingledine, N. Mathewson, P. Syverson)

Feature Implementation

- ☐ Circuits and streams
- ☐ Rate limiting, fairness and congestion control
- ☐ Rendezvous Points and hidden services (NB: this feature is larger than the other 2)

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ The setup of a new circuit and streams
- ☐ Rate limiting of clients that send too many messages
- ☐ End-to-end two-way communication between 2 clients using at least 3 Tor nodes

Performance evaluation

- ☐ Latency and throughput, depending on the number of intermediate nodes in a circuit
- ☐ Overhead of congestion control and rate limiting

Anonymous reputation (3 students)

Description

Reputation systems help users evaluate information quality and incentivize civilised behaviour, often by tallying feedback from other users such as “likes” or votes and linking these scores to a user’s long-term identity. This identity linkage enables user tracking, however, and appears at odds with strong privacy or anonymity. AnonRep is a practical anonymous reputation system offering the benefits of reputation without enabling long-term tracking. AnonRep users anonymously post messages, which they can verifiably tag with their reputation scores without leaking sensitive information.

In this project, you will build a reputation system for Peerster, where each Peerster node votes other nodes accounting for their good / bad behaviours. First, you will build a minimal social media like application on top of Peerster (using the existing broadcast based messaging), where each user can create posts. This application lets each Peerster node vote on the posts from other nodes, and there you will use AnonReps to provide anonymity.

Relevant Paper(s)

- AnonRep: Towards Tracking-Resistant Anonymous Reputation (Ennan Zhai and Bryan Ford)

Feature Implementation

- ☐ Mix-Net (1 student)
- ☐ Verifiable Shuffles and Linkable Ring Signatures (1 student, part-time)
- ☐ AnonRep Workflow (1-2 students)
  Consider dividing the 5 phases of the AnonRep workflow between you

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Basic cryptographic functionalities (mixing, encryption, and decryption)
- ☐ Interactions between MixNet and the rest of your system
- ☐ End-to-end integration tests covering the whole AnonRep workflow

Performance evaluation

- ☐ Run time vs number of clients
- ☐ Computational and bandwidth overhead
- ☐ Number of clients vs the delay of announcement phase

E-Voting (3 students)

Description

This project involves developing a web-based open-audit voting system modelled after the Helios e-voting system. It aims to provide a secure, transparent, and verifiable voting process suitable for low-coercion environments such as online communities, student governments, and local clubs.

Relevant Paper(s)

- Helios: Web-based Open-Audit Voting (Ben Adida)

Feature Implementation

- ☐ Ballot preparation and casting
  A simple interface for voters to prepare and cast their ballots. This will include the encryption of votes and the generation of hash commitments for auditability.
- ☐ Vote shuffling and anonymization
  A mixnet-based vote shuffling mechanism to anonymize votes before tallying. This will ensure that votes are not traceable back to individual voters, maintaining voter privacy.
- ☐ Audit and verification
  The tools for voters and auditors to verify the integrity of the election process. These should allow for auditing individual votes and verifying the overall election results through cryptographic proofs.

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ The voter can audit the ballot before submission
- ☐ The correctness of the vote shuffling and anonymization process
- ☐ The verification tools detect misbehaving shufflers
- ☐ An end-to-end election scenario with 100+ voters

Performance Evaluation

- ☐ Time taken for ballot encryption and casting by individual voters.
- ☐ Time and resource usage of the vote shuffling process
- ☐ Time and resource usage for the audit and verification process, both at the individual ballot level and at the whole-election level.

CRDT (3-4 students)

Description

This project involves developing a rich-text collaboration tool based on Peritext, a CRDT algorithm that supports asynchronous editing and merging of rich-text documents. The goal is to enable multiple users to work on independent copies and merge changes while preserving their formatting and intent.

Since CRDTs make sense in a local-first software context, you will need to implement an editor and the CRDT in the client. You may use any technology you are comfortable with to achieve this result, such as Kotlin Multiplatform, typical web stacks (HTML, TypeScript) or others, as long as you document the necessary steps to build and run your system.

Relevant Paper(s)

- Peritext: A CRDT for Collaborative Rich Text Editing (G. Litt, S. Lim, M. Kleppmann, P. Van Hardenberg)
  See also https://www.inkandswitch.com/peritext/

Feature Implementation

- ☐ Rich-text editor, allowing various formatting options like bold, italic, underline, and colors, expressing changes as CRDT operations.
- ☐ Asynchronous collaboration, enabling users to edit text concurrently, exchanging CRDT operations across the Peerster network, merging changes seamlessly.
- ☐ Conflict detection and resolution, handling overlapping changes and ensuring the strong eventual consistency of the final document

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Editor's correct behaviour during CRDT updates
- ☐ Property-based testing ensuring independent edits are correctly merged
- ☐ Correctness of the conflict detection and resolution for overlapping changes
- ☐ Strong eventual consistency of the CRDT, such that operations arriving in different orders still lead to the same resulting state

Performance Evaluation

- ☐ Time taken for initial document loading and rendering in the editor.
- ☐ Throughput of the number of merge operations vs. the number of parallel changes
- ☐ Storage / bandwidth overhead of the CRDT
- ☐ Performance characteristics at scale, with >1000s of operations

Memory efficient DHT using Erasure coding (3 members)

Description

In a traditional DHT, data items are replicated across multiple nodes to ensure high availability. If a node identified by the finger table entries fails, backup nodes can be used to retrieve the data. However, this approach leads to significant waste of storage space.

Instead, we can employ erasure coding to improve efficiency. By breaking each data item into multiple chunks using erasure codes and storing these chunks across different nodes, we optimize both storage and bandwidth. The DHT can then be used to locate the nodes containing these chunks, allowing parallel retrieval and data reconstruction. This method reduces both storage overhead and bandwidth usage.

Relevant Papers

- Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications (Ion Stoica, Robert Morri)
- Erasure coding for distributed storage: an overview (S. B. Balaji, M. Nikhil Krishnan)

Feature Implementation

- ☐ Chord DHT replication, routing, and finger table updation
- ☐ Erasure coding based replication and reformation

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Chord routing table and data replication
- ☐ Resilience to churn in Chord, with different levels of churn
- ☐ End-to-end scenarios verifying data recovery, including with chunks losses that do not affect the quorum required for erasure codes.

Performance evaluation

- ☐ Item download latency with respect to the number of nodes
- ☐ Item download latency with respect to the churn rate
- ☐ Item download latency with respect to the erasure coding parameters

DynamoDB (3 members)

Description

Key-value stores are highly useful as they enable a wide range of use cases, from caching to session management and real-time data processing. These stores can be implemented in various ways, with Amazon DynamoDB being a popular example due to its scalability and performance. In this project, you will implement an Amazon DynamoDB-like key-value store on top of Peerster, leveraging its peer-to-peer communication capabilities to create a decentralised and robust storage system.

Relevant Papers

- Dynamo: Amazon's Highly Available Key-value Store (Giuseppe DeCandia, Deniz Hastorun)

Feature Implementation

- ☐ Partitioning keys and implementing writes
- ☐ Membership and failure detection

Testing

- ☐ Unit testing and integration testing of all 3 core features
- ☐ Data items replication
- ☐ Data items search and retrieval
- ☐ Tolerance to different levels of membership churn

Performance evaluation

- ☐ Average and 99.9 percentiles of latencies for read and write requests
- ☐ Impact of buffering for latency
- ☐ Impact of hotspots (data items that are concurrently read by many clients)

NameCoin on Peerster (4 members)

Description

Namecoin is a decentralised system for registering domain names on a blockchain. Its goal is to create a naming system without centralized control, where users can register and manage domains without interference.

Relevant Papers

There are several online resources available about NameCoin, but no good reference paper. We thus provide you with this overview of the system, to complement online resources.

Feature Implementation

System Implementation

- ☐ Blockchain: At the core, we need a blockchain system similar to Bitcoin but tailored for domain registration. In particular, as opposed to Bitcoin, in this project you won't need transaction scripting (domain operations can be hardcoded), wallet management, a mempool, or chain pruning. Furthermore, Peerster already provides a communication layer for broadcasting and unicasting messages, which we will use for propagating blockchain transactions and blocks.
- ☐ Transactions: We need to define special transactions for registering, updating, and transferring domain names. Each transaction needs to include domain metadata (like the domain name and its associated IP address) and fees paid in tokens.
- ☐ Proof of Work (PoW): We need to implement a PoW system to secure the blockchain. This involves creating a mechanism where miners solve cryptographic puzzles to add blocks to the chain.
- ☐ Tokens: We need a token system to handle registration fees and miner rewards. These tokens can be created when blocks are mined and used to pay for domain operations like registration and updates.
- ☐ Expiration and Renewal: Domains should expire after a set time unless the owner renews them by sending a new transaction. This helps prevent inactive domains from being permanently locked by inactive users.

(Continues on the next page)

How to Use It

- [ ] Registering a domain: A user broadcasts a `name_new` transaction to register a domain. `name_new` transaction includes **only** the salted hash of the domain name - e.g., `Hash(random_salt, `example.bit`)` - and a small fee paid in tokens.

  This is a front-running protection mechanism, as it prevents miners (and others) from observing the transaction and reserving the domain for themselves in a transaction that's included faster than the original one.
  After a confirmation period (a few blocks), the user sends a `name_firstupdate` transaction, revealing the salt, the domain name in plain text and linking it to an IP address or other arbitrary data. Once confirmed, the domain is registered on the blockchain and associated with the user's namecoin address.

- [ ] Updating a domain: To update the data associated with a domain (e.g., change the IP address), the user submits a `name_update` transaction, which includes the new data and a small fee. This transaction is added to the blockchain, and the updated data becomes active once it's confirmed.

- [ ] Domain expiration: Domains automatically expire after a certain number of blocks unless the user sends a `name_update` transaction before expiration. If a domain expires, it becomes available for anyone else to register.

- [ ] DNS server: The system will include a simple DNS server that looks up domains by querying the blockchain. Users can query the DNS server, which will resolve the domain to its associated IP address or data stored on the blockchain.

Testing

- [ ] Unit testing and integration testing of all core features
- [ ] The end-to-end use case: domain registration, domain update, domain expiration, domain ownership change, and Namecoin-based DNS resolution.
- [ ] The Proof-of-Work mechanism
- [ ] The persistence of submitted transactions in case of churn
- [ ] System robustness against byzantine attacks, including omission and equivocation.
- [ ] System robustness against adversarial network conditions

Performance Evaluation

- [ ] Latency and throughput of mining (blocks per second)
- [ ] Latency and throughput of name registration and updates (transactions per second)
- [ ] Consensus and mining time with respect to the number of nodes in the network
- [ ] Network/bandwidth overhead for domain updates, including the effect on the overall network speed and efficiency, depending on the number of nodes and domains.