Εθνικό Μετσόβειο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Κατανεμημένα Συστήματα

NOOBCASH

A SIMPLE BLOCKCHAIN SYSTEM

Ομάδα 17

Μάρκος-Γεώργιος Ραμός el18841

Βικέντιος Βιτάλης el18803

Ιωάννης Λύρας el17828

Github Repository:

https://github.com/VikentiosVitalis/distributed_systems

Noobcash - Περιγραφή ζητούμενου:

Το ζητούμενο της εργασίας είναι η ανάπτυξη ενός απλού συστήματος Blockchain.

Στο σύστημα καταγράφονται transactions (δοσοληψίες) μεταξύ των συμμετεχόντων και εξασφαλίζεται το consensus με χρήση Proof-of-Work.

Το Blockchain είναι ένας κατανεμημένος λογιστικός κατάλογος (distributed ledger). Κάθε καινούρια ομάδα καταχωρήσεων είναι ένα block και συνδέεται με τα προηγούμενα, δημιουργώντας μια αλυσίδα καταχωρήσεων. Τα blocks αυτά συνδέονται μονοσήμαντα μεταξύ τους μέσω συνάρτησης κατακερματισμού (hashing). Μέσω της συγκεκριμένης συνάρτησης προκύπτει η διαδικασία που ονομάζουμε Proof-of-work. Σε επίπεδο υλοποίησης βασίζεται στην αλγοριθμική επίλυση ενός υπολογιστικού προβλήματος του οποίου το αποτέλεσμα επιβεβαιώνεται εύκολα κι εξασφαλίζει την ακεραιότητα των στοιχείων του block.

Έτσι, το Blockchain λειτουργεί ως ένα αποκεντρωμένο (decentralized) λογιστικό καθολικό σύστημα, το οποίο είναι κοινό για όλους τους συμμετέχοντες. Αυτό συμβαίνει επειδή όλοι οι εμπλεκόμενοι αποθηκεύουν ένα αντίγραφό του, γεγονός που εξασφαλίζει την ασφάλεια και την διαφάνεια των συναλλαγών.

Η χρησιμότητα του Blockchain προκύπτει από το γεγονός ότι δεν είναι πλέον απαραίτητη η ύπαρξη μιας ενδιάμεσης αξιόπιστης αρχής (πχ. Τράπεζας), ενώ η αξιοπιστία των συναλλασσομένων μελών βασίζεται σε αλγοριθμική επιβεβαίωση.

Περιγραφή Συστήματος & Περιβάλλον Υλοποίησης:

Το σύστημά μας χωρίζεται σε backend, cli και frontend. Γίνεται χρήση Threads για την διαδικασία του mining.

Χρησιμοποιούμε το Visual Studio Code ως Integrated Development Environment (IDE) και το Git ως λογισμικό - εργαλείο απομακρυσμένης συνεργασίας. Αναπτύξαμε το σύστημα με το Flask framework σε γλώσσα προγραμματισμού Python3.8. Χρησιμοποιούμε, επίσης Python3.8, HTML5, CSS3 για το frontend και Python3.8 για το CLI.

Επικοινωνία με το σύστημα Okeanos μέσω Git:

Τρέχοντας το etc/update.sh script κατεβάζουμε το Git repository από το διαδίκτυο μέσω του κόμβου που έχει σύνδεση με το Internet και το μεταφέρουμε στους υπόλοιπους κόμβους του συστήματος. Χάρη στο paswordless ssh μπορούμε να μεταφέρουμε τα αρχεία χωρίς την χρήση κωδικών στα υπόλοιπα VMs μέσω της εντολής scp.

Εκκίνηση Εφαρμογής:

Στο app.py υλοποιείται το REpresentational State Transfer Application Programming Interface (REST API) του συστήματός μας το οποίο αποτελεί το συμβόλαιο επικοινωνίας μεταξύ των επιμέρους τμημάτων. Για την εκκίνηση του, καλείται η κλάση node.py και ξεκινάει η λειτουργία του Noobcash. Εισάγεται ο bootstrap node και στη συνέχεια οι κόμβοι παιδιά.

Για την εισαγωγή του bootstrap node πρέπει να δοθεί η εντολή:

> *'python3.8 distributed_systems-main/noobcash/app.py PORT IP NumberOfChildren isBootstrap'*

με τα κατάλληλα ορίσματα στο PORT, IP και NumberOfChildren.

Το όρισμα true δηλώνει ότι ο κόμβος αυτός θα είναι ο bootstrap και στη συνέχεια εισάγουμε τόσα παιδιά όσα και το NumberOfChildren. Π.χ.:

> *'python3.8 distributed_systems-main/noobcash/app.py 5000 83.212.81.180 4 true 5'*

Εάν θέλουμε να τρέξουμε τα πειράματα που μας έχουν δοθεί μέσα από τα αρχεία 5nodes και 10nodes, δίνουμε τις ίδιες εντολές απλά με 5 ή 10 σαν τελευταίο όρισμα της παραπάνω εντολής.

Node:

Αντικείμενο	Ρόλος
bootstrapAddr	IP του αρχικού κόμβου (bootstrap)
port	Port του Node
ip	IP του Node
nodeNr	Αριθμός των nodes (εκτός bootstrap)
ipList	Λίστα με όλες τις {ID, IP, publicKey} των συνδεδεμένων κόμβων
buffer	Λίστα με transactions που είναι στην ουρά για εκτέλεση
blockchain	Το μέχρι τώρα blockchain
allBlockchains	Dictionary για αποθήκευση των chains σε περίπτωση conflict
currentBlock	Block που ήρθε μέσω /mine αλλά δεν είναι ακόμα στο Blockchain
validationBlocks	Λίστα των blocks που εισέρχονται προς επικύρωση

Διαδικασία λειτουργίας της κλάσης Node:

1. Αρχικά εισέρχεται ο bootstrap node, στον οποίο αρχικοποιούνται τα χρήματα που θα μοιράσει στα παιδιά. Δημιουργεί το genesis block, παράλληλα (χρήση thread) αναμένει την εισαγωγή όλων των παιδιών.

2. Κάθε φορά που εισέρχεται ένα παιδί στο σύστημα ενημερώνει τον bootstrap κόμβο σχετικά με την IP και το δημόσιο κλειδί του.

3. ΄Οταν εισέλθουν όλα τα παιδιά ο bootstrap ξεκινάει να τους αναθέτει IDs και τους στέλνει από 100 NBCs στον καθένα.

4. Όταν κάθε κόμβος λάβει την IP List και το Genesis block με τα νομίσματα, το self.nodeFlag γίνεται true και ξεκινάει η διαδικασία. Εκτελείται αενάως η συνάρτηση waitThread κι αποτελείται από τα επιμέρους μέρη:

    4.1. Εάν υπάρχει block από άλλο μηχάνημα καλούμε την συνάρτηση validateBlock, η οποία το ελέγχει ως προς την ορθότητα και το εισάγει στο blockchain, αλλιώς εκτελεί Consensus.

    4.2. Εάν δεν κάνουμε mine, λαμβάνουμε ένα transaction (δοσοληψία) από τον buffer. Σε περίπτωση incoming Transaction το εισάγουμε σε block ενώ σε περίπτωση outgoing Transaction, καλούμε την συνάρτηση createTransaction.

Συναρτήσεις κλάσης Node:

- addNode: Προσθήκη των {ID, IP, publicKey} του node που εισέρχεται στο ipList.
- setIPList: Αρχικοποιείται η IP List.
- setGenesis: Εισχωρείται το genesis block.
- broadcastNodes: Ενημερώνουμε όλους τους κόμβους σχετικά με την ύπαρξη του genesis block και στέλνεται η ipList.
- broadcastTransaction: Η συναλλαγή που μόλις έχει δημιουργηθεί αποστέλεται σε όλους τους άλλους κόμβους του συστήματος.
- broadcastConsensus: Στέλνουμε request για consensus σε όλους τους κόμβους και περιμένουμε ως απάντηση τις λίστες των blockchains.
- waitThread: Η συνάρτηση αυτή τρέχει συνεχώς από τη στιγμή και μετά που όλοι οι κόμβοι έχουν συνδεθεί στο σύστημα. Εκτελεί τα βήματα 4.1 και 4.2.
- createTransaction: Συνάρτηση δημιουργίας συναλλαγής. Δέχεται ως ορίσματα το ID του παραλήπτη και το ποσό που του αποστέλεται. Μέσω του wallet βρίσκει τα UTXOs και δημιουργεί τη λίστα inputTransactions. Εν συνεχεία, δημιουργεί ένα αντικείμενο κλάσης Transaction, ενημερώνονται οι κατάλληλες μεταβλητές, υπογράφεται η συναλλαγή, γίνεται broadcast σε όλο το δίκτυο και προστίθεται στο block και στο wallet.

• insertBlockchain: Καλείται από την waitThread, προσθέτει το δεδομένο transaction στην λίστα transactions του blockchain. Εάν το block έχει φτάσει το όριο χωρητικότητας δημιουργεί ένα mine thread.

• getBalance: Επιστρέφεται το υπόλοιπο των χρημάτων του κόμβου.

• validateTransaction: Επικύρωση της συναλλαγής μέσω της υπογραφής της και του ελέγχου των inputs και outputs.

• validateBlock: Επικύρωση του στοιχείου του current hash του block μέσω hash function. Εν συνεχεία, παύση του mine κι εισαγωγή του block στο Blockchain εάν ταιριάζει, αλλιώς consensus.

• resolveConflict: Η συνάρτηση καλείται όταν ένα κόμβος λάβει ένα block το οποίο δεν μπορεί να κάνει validate γιατί το πεδίο previous_hash δεν ισούται με το hash του προηγούμενου block. Αυτό μπορεί να σημαίνει ότι έχει δημιουργηθεί κάποια διακλάδωση, η οποία πρέπει να επιλυθεί. Περιμένει να λάβει τα blockchains από όλα τα άλλα nodes στο σύστημα που έρχονται ως απάντηση στο broadcastNodes. Αφότου τα λάβει, επιλέγει εκείνο με μεγαλύτερο μήκος και το εγκαθιστά ως δικό του blockchain.

• validateChain: Επαληθεύει όλα τα blocks του blockchain.

Wallet:

Αντικείμενο	Ρόλος
publicKey	Δημόσιο Κλειδί
privateKey	Ιδιωτικό Κλειδί
transactions	Λίστα με όλα τα transactions
tr_dict	Dictionary που λειτουργεί ως συνάρτηση από tid σε index της λίστας transactions
balances	Ποσό χρημάτων όλων των κόμβων
unspentOutputs	Λίστα με τα UTXOs του node μας

Συναρτήσεις κλάσης Wallet:

- getMoney: Επιστρέφει τις UTXOs που χρειάζονται για να καλύψουμε το amount και το σύνολο των χρημάτων αυτών.
- addTransaction: Απαλοίφει τα input transactions, δηλαδή θέτει unspent = false στα input transactions. Προσθέτει τα transactions outputs στα unspent outputs του wallet μας και προσθέτει το transaction στη λίστα transaction και στο tr_dict.

Transaction:

Αντικείμενο	Ρόλος
sender	ID του αποστολέα της συναλλαγής
receiver	ID του παραλήπτη της συναλλαγής
amount	Το ποσό που αποστέλεται (NBCs)
inputs	Λίστα με UTXOs που σπαταλήθηκαν
outputSender	UTXO με το υπόλοιπο που προστίθεται στον λογαριασμό του αποστολέα
outputReceiver	UTXO με το ποσό που προστίθεται στον λογαριασμό του παραλήπτη
signature	Η υπογραφή της συναλλαγής
tid	Το id της συναλλαγής

Συναρτήσεις κλάσης Transaction:

- toJSON: Μετατροπή του transaction σε json μορφή για την ευκολότερη μεταφορά της συναλλαγής στο δίκτυο.
- verifySignature: Επαλήθευση της υπογραφής της συναλλαγής.

Blockchain:

Αντικείμενο	Ρόλος
maxTransactions	Συναλλαγές ανά block
blockchain	Λίστα με τα blocks του Blockchain
transactions	Προσωρινή αποθήκευση συναλλαγών
stopMine	Σημαία για παύση διαδικασίας mining

Συναρτήσεις κλάσης Blockchain:

- mine: Ξεκινάει την διαδικασία της εξόρυξης.
- broadcastBlock: Στέλνει σε όλους τους κόμβους το νέο block της αλυσίδας.
- convert_chain: Μετατροπή του block σε json μορφή για την ευκολότερη μεταφορά των blocks της αλυσίδας στο δίκτυο.
- genBlock: Επιστρέφει το genesis block.
- addBlock: Προσθέτει block στην αλυσίδα.
- getLastHash: Παίρνει το hash του τελευταίου block της αλυσίδας.

Block:

Αντικείμενο	Ρόλος
index	Ο αύξων αριθμός του block
timestamp	Το timestamp της δημιουργίας του block
transactions	Οι συναλλαγές που περιέχονται στο block
nonce	Η μεταβλητή που συνεισφέρει στην λύση του Proof-of-Work
previous_hash	Το hash του προηγούμενου block στο Blockchain
current_hash	Το τρέχον hash του block
goal	Δυσκολία αλγορίθμου εξόρυξης

Συναρτήσεις κλάσης Block:

- set: Αρχικοποιεί ένα block από json αρχείο.
- hashing: Λαμβάνει το json string του block κι εκτελεί την συνάρτηση hash.
- convert_block: Μετατροπή του block σε json μορφή για την ευκολότερη μεταφορά του πάνω από το δίκτυο.
- mine_block: Εκτελεί την διαδικασία hashing μέχρι να βρεθεί το string με πρώτα DIFFICULTY στοιχεία να είναι '0'(mining).

Transactions_input:

Αντικείμενο	Ρόλος
previous_output_id	ID της συναλλαγής που προέρχεται το ποσό

Transactions_output:

Αντικείμενο	Ρόλος
tid	ID της συναλλαγής που προέρχεται το ποσό
receiver	Address του παραλήπτη
amount	Ποσό συναλλαγής
unspent	Boolean αν έχουν σπαταληθεί τα χρήματα του TXO

- fromdict: Φορτώνουμε τις τιμές από dictionary.
- __str__: Επιστρέφει το Transaction Output ως string.

Command Line Interface (CLI):

Για την εκκίνηση του CLI, εκτελούμε στον φάκελο ‘noobcash’ την εντολή

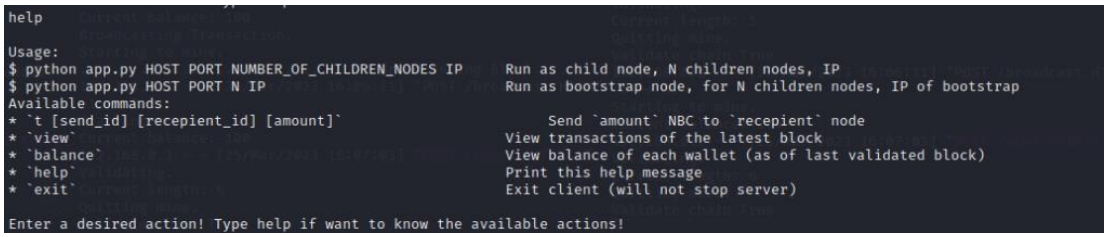
‘python3.8 distributed_systems-main/noobcash/cli.py 5000 83.212.81.180‘

αφού έχουμε τρέξει:

‘python3.8 distributed_systems-main/noobcash/app.py 5000 83.212.81.180 4 true’, δηλαδή εκκινήσει το σύστημα.

Οι λειτουργίες που υλοποιήθηκαν:

- Δημιουργία νέου Transaction.
- Προβολή των transactions που περιέχονται στο τελευταίο επικυρωμένο block του noobcash blockchain.
- Εμφάνιση των υπολοίπων όλων των wallets.
- Εμφάνιση οδηγιών σχετικά με τη χρήση της διεπαφής.



Μέσω της εντολής ‘help’ φαίνονται, όπως και παραπάνω, οι διαθέσιμες λειτουργίες που ζητούνται για το Noobcash Client.

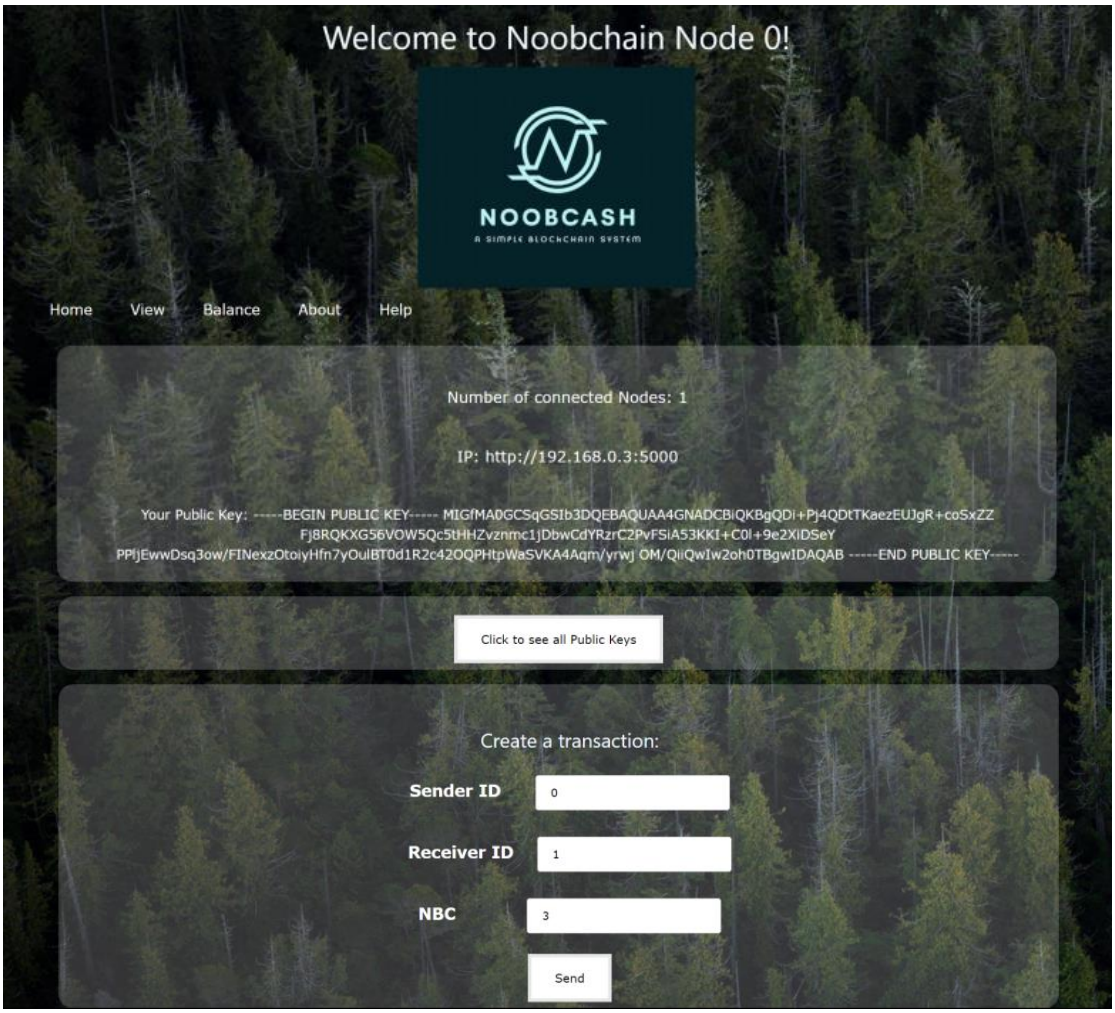
Web Application:

Η Web εφαρμογή διαθέτει τις λειτουγίες απο αναφέρθηκαν παραπάνω. Για την εκκίνηση της χρειάζεται η εντολή:

'python3.8 distributed_systems-main/noobcash/cli.py 5000 83.212.81.180', η οποία εκκινεί το σύστημα και μετά επισκεπτόσμαστε σε οποιονδήποτε browser τον υπερσύνδεσμο http://83.212.81.180:5000/

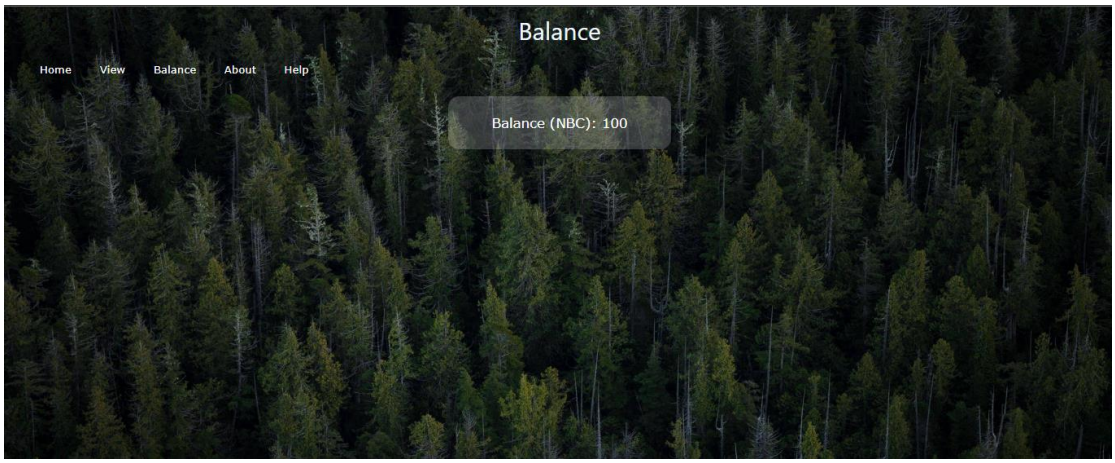
Ακολουθεί η παρουσίαση των διαθέσιμων σελίδων:

• Αρχική Σελίδα(/home) - Δημιουργία Transaction:

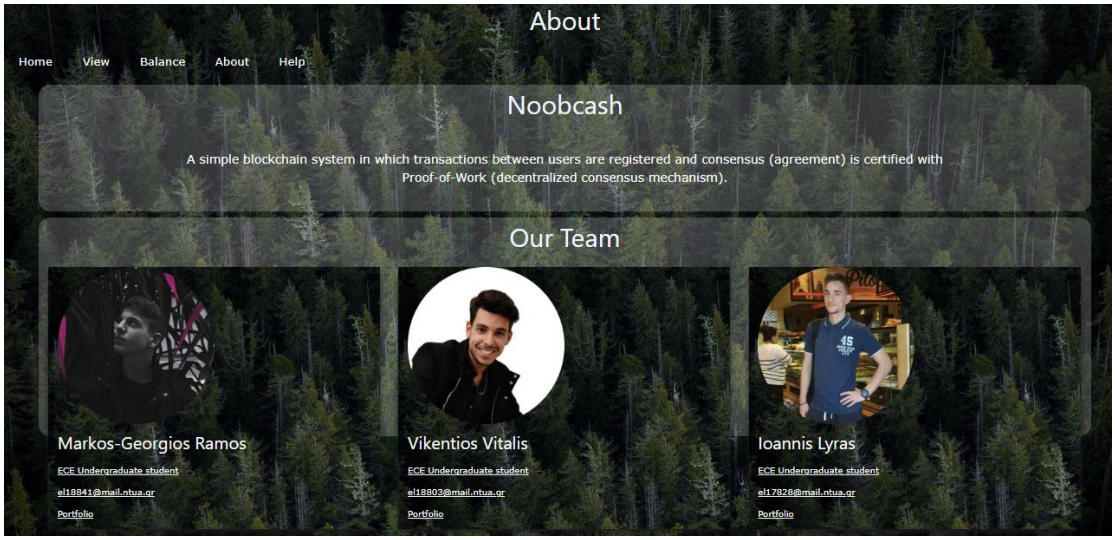


• Σελίδα View (/view):

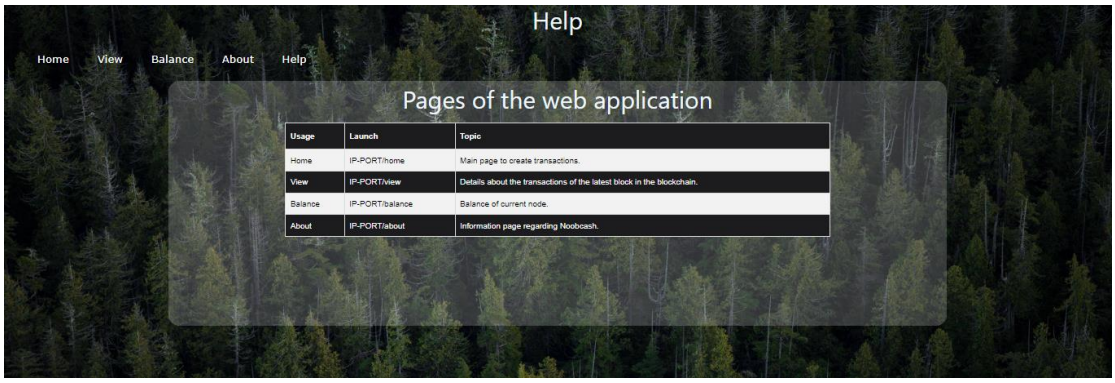
• Σελίδα Balance (/balance)



• Σελίδα About (/about)



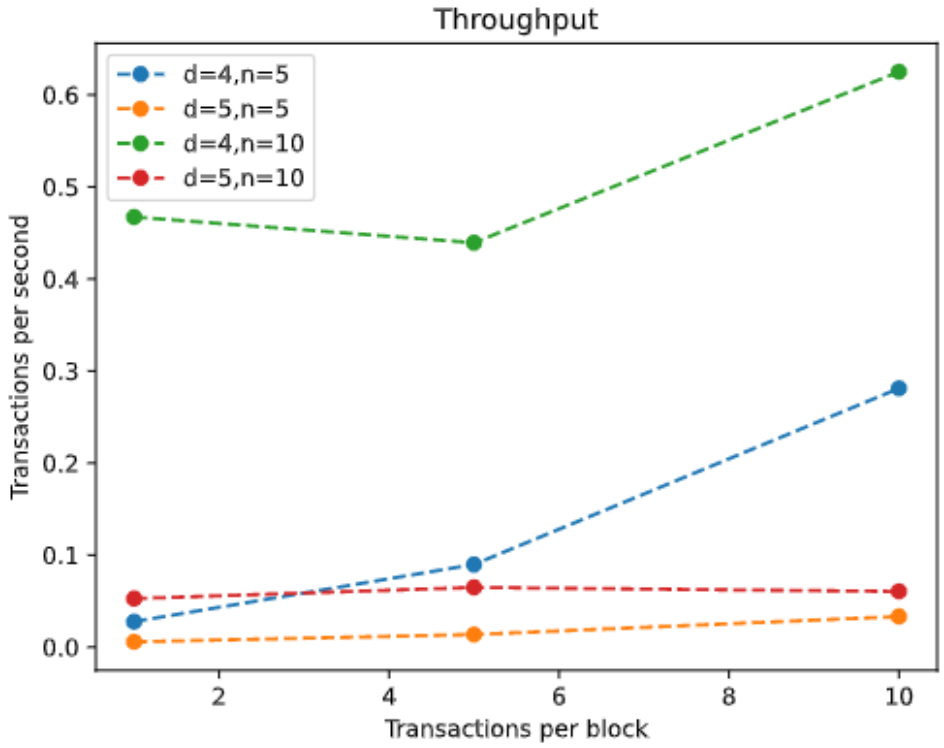
• Σελίδα Help (/help)



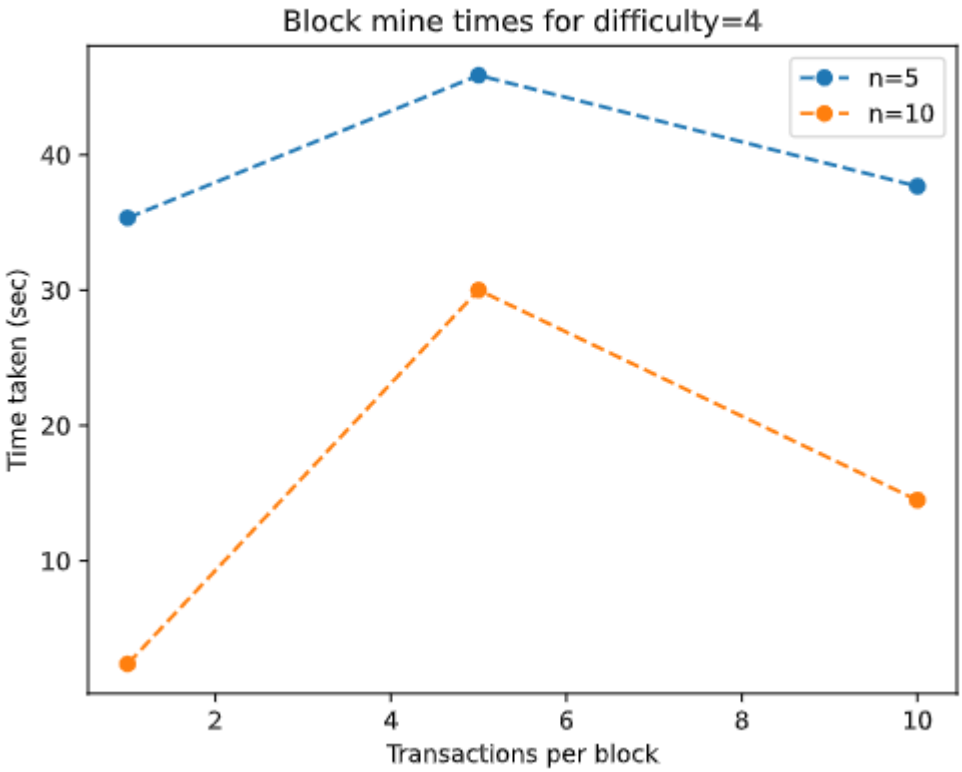
Πειράματα & **Κλιμακωσιμότητα:**

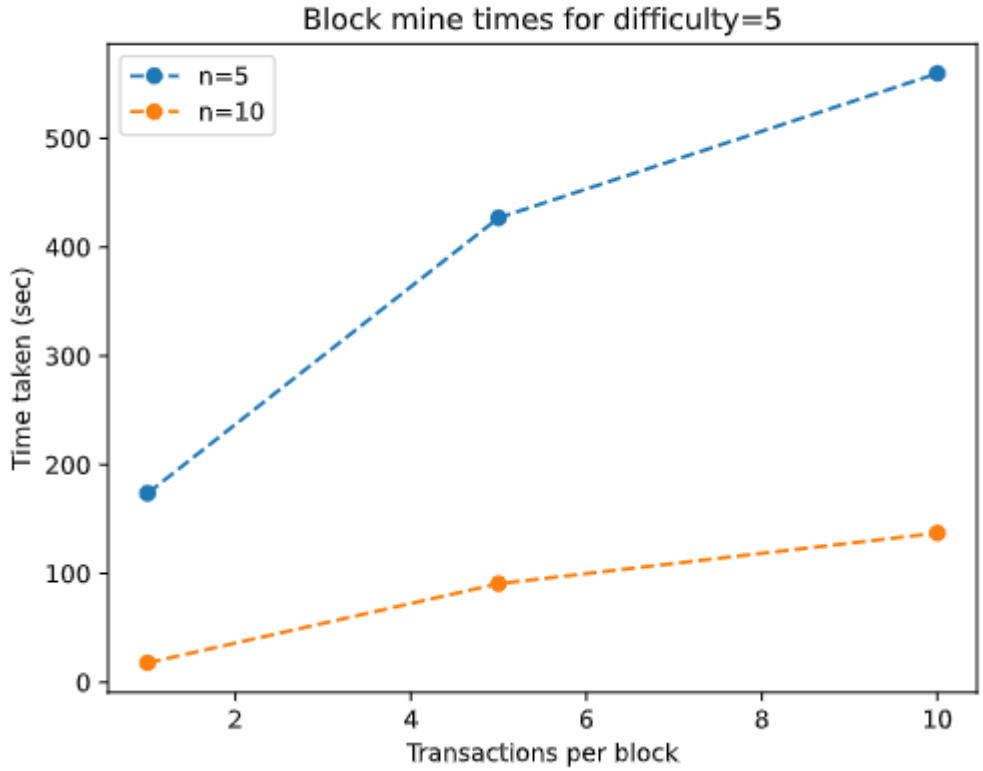
Μελετήθηκε η απόδοση και η κλιμακωσιμότητα του συστήματός μας σε 5 Virtual Machines που μας δόθηκαν. Προκειμένου να εξάγουμε ορθά συμπεράσματα εκτελέσαμε πειράματα με 5 και 10 nodes. Εκκινήσαμε 1 και 2 Noobcash clients σε κάθε ένα από τα machines αντίστοιχα. Όταν τα μηχανήματα λάβουν το μήνυμα από τον bootstrap κόμβο γεμίζουν τον buffer τους με τις εντολές των αρχείων input. Έτσι, υπολογίζουμε τους ζητούμενους χρόνους εκτελώντας πολλές συναλλαγές και εξορύξεις.

• Throughput (ρυθμαπόδοση) του συστήματος: Transactions που εξυπηρετούνται στην μονάδα του χρόνου.



• Block mine time: Ο μέσος χρόνος που απαιτείται για να προστεθεί ένα νέο block στο Blockchain.





Συμπεράσματα:

Παρατηρούμε ότι το Throughput το block mine time των συναλλαγών Difficulty 5 είναι μία τάξη μεγέθους μεγαλύτερο από αυτών με Difficulty 4. Λογική συνέπεια του γεγονότος ότι η πιθανότητα να έχουμε στοχείο με 5 συνεχόμενα μηδενικά στην αρχή είναι 10 φορές μικρότερη σε σχέση με την πιθανότητα να έχουμε 4 συνεχόμενα μηδενικά.

Επίσης, όταν έχουμε 10 nodes βλέπουμε σημαντική μείωση του μέσου block mine time σε σχέση με 5 nodes. Αυτό οφείλεται στο ότι η πιθανότητα για συγκεκριμένο χρονικό διάστημα 10 nodes να μην έχουν λύσει block είναι σημαντικά μικρότερη από την πιθανότητα για το ίδιο χρονικό διάστημα 5 nodes να μην έχουν λύσει block.

Το transaction Throughput αποτελεί άμεσο αποτέλεσμα του block mine time και του transactions per block.

Έτσι, παρατηρούμε ότι η γραφική παράσταση του transaction Throughput ανεβαίνει για μεγαλύτερο transaction block με σταθερό Difficulty. Επίσης, το transaction Throughput είναι αντιστρόφως ανάλογο του block mine time (Όσο γρηγορότερα κάνουμε mine blocks, τόσο γρηγορότερα προσθέτουμε transactions στο σύστημα).

Εν κατακλείδι, τα blockchains με 10 nodes αποδίδουν σημαντικά καλύτερα από εκείνα με 5 nodes.