



ΕΘΝΙΚΟΝ ΜΕΤΣΟΒΙΟΝ ΠΟΛΥΤΕΧΝΕΙΟΝ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένα Συστήματα

Αναφορά Εξαμηνιαίας Εργασίας

Δημήτριος Κυριακίδης
el17077

Χειμερινό Εξάμηνο 2022 - 2023

Περίληψη

Στην παρούσα αναφορά περιγράφεται η δομή της εξαμηνιαίας εργασίας, η διαδικασία ανάπτυξης αυτής, και η μεθοδολογία διεξαγωγής μετρήσεων με διαφορετικούς παραμέτρους στην υποδομή του εργαστηρίου με εικονικές μηχανές. Έπειτα σχολιάζονται τα αποτελέσματα και εξάγονται συμπεράσματα. Επίσης γίνεται σύντομη επισκόπηση του θεωρητικού υποβάθρου της τεχνολογίας blockchain. Τέλος επιδεικνύεται η λειτουργία του κατασκευασμένου συστήματος μέσω διεπαφής γραμμής εντολών και γραφικής εφαρμογής.

Όλα τα αρχεία κώδικα και αποτελέσματα της εργασίας έχουν δημοσιευθεί στο εξής github repository: github.com/DimK19/when-lambo. Αναλυτικά όλες οι μετρήσεις από τα πειράματα με διάφορα πλήθη κόμβων και τιμές difficulty και block capacity στο αρχείο excel "experiment_results.xlsx".

Ημερομηνία υποβολής 25/03/2022, εντός της κανονικής προθεσμίας.

Περιεχόμενα

Περίληψη	2
Κρυπτογραφία	4
Υπογραφή RSA	4
Συνάρτηση SHA-256	5
Εργασία	5
Περιγραφή κυριοτέρων κλάσεων και μεθόδων.....	6
Πολυνηματική εκτέλεση.....	9
Διαγράμματα activity UML.....	11
Πειράματα.....	13
Παράμετροι	13
Μετρώμενα μεγέθη	13
Υποδομή εργαστηρίου	14
Εκτέλεση	15
Αποτελέσματα	17
Συμπεράσματα.....	18
Παράρτημα	20
Επίδειξη λειτουργίας της γραφικής διεπαφής.....	20
Επίδειξη λειτουργίας CLI.....	23
Βιβλιογραφία	26

Κρυπτογραφία

Υπογραφή RSA

Ο αλγόριθμος RSA (Rivest-Shamir-Adleman) είναι ο ευρύτερα χρησιμοποιούμενος αλγόριθμος κρυπτογραφίας δημοσίου κλειδιού. Πλαισιώνεται από το σύνολο προτύπων PKCS (Public Key Cryptography Standards), το οποίο υλοποιείται από την ομώνυμη βιβλιοθήκη της Python. Ο αλγόριθμος έχει ως εξής:

Κάθε συμμετέχων ακολουθεί τα κάτωθι βήματα για την δημιουργία του δημοσίου και του ιδιωτικού του κλειδιού

1. Επιλέγει δύο διαφορετικούς, μεγάλους τυχαίους πρώτους αριθμούς p και q τέτοιους, ώστε η διαφορά $p-q$ να είναι επίσης μεγάλος αριθμός.
 2. Υπολογίζει το γινόμενο $n=pq$.
 3. Υπολογίζει την τιμή της συνάρτησης Euler $\varphi(n) = (p-1)(q-1)$.
 4. Επιλέγει έναν αριθμό $1 < e < \varphi(n)$, τέτοιον ώστε να είναι σχετικά πρώτος με το $\varphi(n)$.
 5. Υπολογίζει τον αριθμό d με την ιδιότητα $d \cdot e \equiv 1 \pmod{\varphi(n)}$.
- Το δημόσιο κλειδί είναι το (n, e) , και το ιδιωτικό το d .

Όπου η τιμή της συνάρτησης φ του Euler για ακέραιο n είναι το πλήθος των ακεραίων μεταξύ 1 και n σχετικά πρώτων με τον n . Αποδεικνύεται ότι αν $(m, n) = 1$ τότε $\varphi(mn) = \varphi(m)\varphi(n)$.

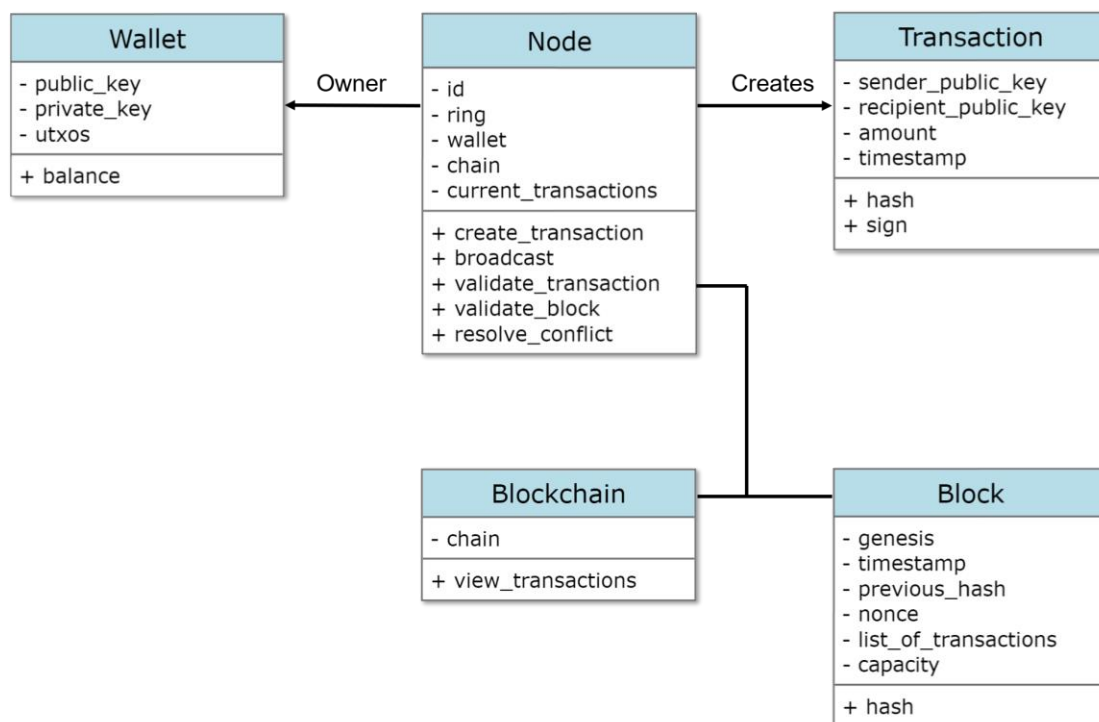
Στην συνέχεια, για την δημιουργία υπογραφής RSA για ένα μήνυμα m ο υπογράφων «A» υπολογίζει το $H = h(m)$, όπου h μία μονόδορμη συνάρτηση κατακερματισμού με πεδίο τιμών $[0, n-1]$, και αποστέλλει το $S_A = H^d \pmod{n}$. Οποιαδήποτε οντότητα γνωρίζει το δημόσιο κλειδί του A μπορεί να επιβεβαιώσει την υπογραφή του πάνω στο μήνυμα m ως υπολογίζοντας τις τιμές $(S_A)^e \pmod{n} = H$ και $H' = h(m)$ και ελέγχοντας αν είναι ίσες. Σε αυτήν την περίπτωση η υπογραφή έχει επαληθευθεί.

Συνάρτηση SHA-256

Η συνάρτηση κατακερματισμού SHA-256 ανήκει στην ευρύτερη οικογένεια των συναρτήσεων - αλγορίθμων κατακερματισμού SHA (Secure Hash Algorithm). Λαμβάνει ως είσοδο μηνύματα μήκους μικρότερου από 2^{64} bits και παράγει τιμή μήκους 256 bits. Το εισαγόμενο μήνυμα συμπληρώνεται για να αποκτήσει το κατάλληλο κάλυμμα και γίνεται επεξεργασία του σε τμήματα μεγέθους 512 bits. Τόσο ο αλγόριθμος SHA όσο και ο RSA είναι ντετερμινιστικοί.

Εργασία

Το σύστημα αναπτύχθηκε σε Python, με βασικές επιπλέον βιβλιοθήκες τις Flask, για χειρισμό της επικοινωνίας με HTTP requests, και PyCryptodome για τις συναρτήσεις κρυπτογράφησης. Η υλοποίηση του βασικού υποβάθρου που επιτελεί τις λειτουργίες της blockchain περιέχεται στον φάκελλο nbcoin και αποτελείται από πέντε βασικά αρχεία - κλάσεις, και το rest.py που υλοποιεί μεθόδους για την επικοινωνία μεταξύ των κόμβων και την εκτέλεση δοκιμών.



Η διεπαφή γραμμής εντολών (command line interface) έχει υλοποιηθεί στο αρχείο `cli.py`, και η γραφική διεπαφή εν είδει web εφαρμογής στον φάκελλο `client`. Το αρχείο `experiments.py` αποστέλλει παράλληλα στις διευθύνσεις όλων των εικονικών μηχανών (ή του τοπικού δικτύου ενός υπολογιστή για τις ανάγκες testing) τα κατάλληλα αιτήματα HTTP για την εκτέλεση των απαιτούμενων πειραμάτων. Η συμπεριφορά σε αυτά τα αιτήματα καθορίζεται όπως προανεφέρθη από το `rest.py`, το οποίο υλοποιεί ένα REST API και τρέχει έναν απλό HTTP server που δέχεται και αποστέλλει αιτήματα. Κάθε κόμβος του συστήματος προσομοιώνεται από έναν server.

Περιγραφή κυριοτέρων κλάσεων και μεθόδων

Οι πέντε βασικές κλάσεις που απαρτίζουν το σύστημα είναι οι Node, Wallet, Transaction, Block, και Blockchain. Αυτές περιέχουν τα στοιχεία και τις λειτουργίες των ομονύμων ονοτήτων, σύμφωνα με το πρωτόκολλο. Κάθε κλάση υλοποιεί την μέθοδο `as_dict`, η οποία επιστρέφει ένα dictionary object με τα σημαντικά μέλη αυτής. Χρησιμοποιείται για την εξαγωγή του hash ενός αντικειμένου, αλλά κυρίως για την αποστολή αντικειμένων μέσω HTTP, αφού πρώτα κωδικοποιηθούν σε μορφή JSON. Στους αντίστοιχους controllers στο `rest.py`, τα αντικείμενα αυτά αποκωδικοποιούνται ξανά ως αντικείμενα Python των αντίστοιχων κλάσεων καλώντας τους constructors των κλάσεων αυτών. Επίσης όλες οι κλάσεις κάνουν override την μέθοδο `__hash__`, ώστε να μπορεί να κληθεί η συνάρτηση `hash()` σε αυτά και να υπολογίζεται το hash ενός αντικειμένου λαμβάνοντας υπ' όψιν τα απαραίτητα διαφοροποιητικά στοιχεία (timestamp για transaction, nonce για block).

Οι κυριότερες μέθοδοι των κλάσεων είναι οι ακόλουθες:

Node

- `register_bootstrap()`: αυτή η μέθοδος επιτρέπεται να κληθεί μόνο από αντικείμενο Node που ανήκει στον κόμβο bootstrap, και

απονέμει σε αυτόν το $id = 0$ και μεταφέρει με μία συναλλαγή το αρχικό ποσό των $100 \cdot TOTAL_NODES$ coins.

- `register_node_to_ring(n)`: αυτή η μέθοδος επίσης καλείται μόνο από τον κόμβο `bootstrap`, αφού λάβει αίτημα καταχώρησης από τον κόμβο `n` με τα δεδομένα του, τον προσθέτει στην λίστα κόμβων, του αναθέτει `id` και προσαυξάνει τον μετρητή κόμβων ώστε να ελέγχει τότε το δίκτυο είναι πλήρες και συνεπώς έτοιμο να τρέξει.
- `create_genesis()`: άλλη μία αποκλειστική μέθοδος του `bootstrap`, κατασκευάζει την συναλλαγή της αρχικής ποσότητας coins, την προσθέτει στο `genesis block`, το οποίο περιέχει μόνο αυτήν και δεν επαληθεύεται, και το προσθέτει στην νέα αλυσίδα.
- `initialize_network()`: συνάρτηση που αφορά μόνο τον `bootstrap` ενεργοποιείται αφού έχουν καταχωρηθεί αρκετοί κόμβοι (σύμφωνα με τις οδηγίες του μαθήματος στο `helios` γίνεται η υπόθεση ότι συναλλαγές αρχίζουν μόνο αφού εγγραφούν όλοι οι κόμβοι). Με αυτήν ο `bootstrap` αποστέλλει τα απαραίτητα συγκεντρωτικά δεδομένα (πληροφορίες διευθύνσεων κόμβων, αρχική αλυσίδα) σε όλους τους κόμβους.
- `send_genesis()`: η τελευταία βασική συνάρτηση που αφορά μόνο τον `bootstrap`, μετά την αρχικοποίηση του δικτύου κατασκευάζει μία συναλλαγή ύψους 100 για κάθε κόμβο και τις εκπέμπει (`broadcast`) στο δίκτυο.
- `send_registration_request()`: αφορά τους υπολοίπους κόμβους, στέλνουν αίτημα HTTP στην διεύθυνση του `bootstrap` (με το κατάλληλο `url` οπότε ενεργοποιείται το αντίστοιχο `endpoint` στο `rest.py`) για να καταχωρηθούν στο δίκτυο.
- `create_transaction(recipient, amount)`: δημιουργία νέας συναλλαγής από το `wallet` του `Node` που καλεί την μέθοδο προς το `public key` του παραλήπτη, όπως περιγράφεται από το διάγραμμα.
- `broadcast(m, param)`: αποστολή αιτήματος HTTP POST προς κάθε κόμβο στο `ring`, για το μήνυμα `m` του οποίου το είδος καθορίζεται από την παράμετρο (`transaction`, `block`, ή `chain`).

- `mine_block()`: η συνάρτηση αυτή ενεργοποιείται όταν η λίστα τρεχόντων συναλλαγών του κόμβου φθάσει σε μήκος το προκαθορισμένο μέγεθος `block`. Τότε η λίστα αυτή μαζί με το `hash` του προηγούμενου `block` και τον αριθμό `nonce` συνθέτουν το `hash` του `block`. Δοκιμάζονται διαδοχικά τιμές του `nonce` έως ότου το αποτέλεσμα πληροί το κριτήριο τα πρώτα ψηφία του να είναι 0. Αυτό καθορίζεται από την παράμετρο "MINING DIFFICULTY".
- `verify_signature(t)`: κατά την λήψη μίας συναλλαγής `t`, επαληθεύεται η υπογραφή της σύμφωνα με τον προαναφερθέντα αλγόριθμο.
- `validate_transaction(t)`: κατά την λήψη μίας συναλλαγής ακολουθούνται τα εξής βήματα για την επικύρωσή της: ελέγχεται αν η υπογραφή της είναι αληθής, και αν, σύμφωνα με τα δεδομένα του παρόντος κόμβου, ο αποστολέας έχει αρκετούς πόρους - UTXO για να πραγματοποιήσει την συναλλαγή.
- `validate_block(b)`: κατά την λήψη ενός `block` ακολουθούνται τα εξής βήματα για την επικύρωσή του: ελέγχεται αν είναι το `genesis block` οπότε δεν επικυρώνεται, ελέγχεται αν το πεδίο `hash` του `block` όντως ισούται με το `hash` του αντικειμένου, ελέγχεται αν δείχνει προς το `hash` του προηγούμενου `block` της τρέχουσας αλυσίδας, και επικυρώνεται ότι ικανοποιείται η συνθήκη `proof-of-work` για το `hash` (leading zeroes).
- `resolve_conflict()`: αν προκύψει ασυμφωνία εφαρμόζεται το πρωτόκολλο `consensus` του `blockchain`, σύμφωνα με το οποίο υιοθετείται η μεγαλύτερη σε μήκος έγκυρη αλυσίδα.
- `get_statistics()`: επιστρέφει για τον κόμβο τις μετρήσεις που εξετάζονται στο πλαίσιο της εργασίας, δηλαδή ρυθμαπόδοση (`throughput`) σε συναλλαγές ανά δευτερόλεπτο, και μέσο χρόνο εξαγωγής `block` σε δευτερόλεπτα. Το τελικό αποτέλεσμα είναι για την δεύτερη τιμή για όλο το σύστημα είναι ο μέσος όρος μεταξύ όλων των κόμβων, ενώ για το `throughput` του συστήματος μετριέται ο χρόνος πρώτης και τελευταίας συνολικά συναλλαγής. Οι υπολογισμοί για την εξαγωγή αυτών γίνονται στα καίρια

σημεία εντός των μεθόδων. Επίσης αποθηκεύει τα στοιχεία αυτά σε αρχείο κειμένου.

Transaction

- `sign()`: παράγει την υπογραφή RSA της συναλλαγής με τον αλγόριθμο που έχει περιγραφεί.

Wallet

- `__init__()`: κάθε κλάση περιέχει έναν κατασκευαστή, ο συγκεκριμένος χρήζει ιδιαιτέρας μνείας καθώς αρχικοποιεί το δημόσιο και το ιδιωτικό κλειδί με τον αλγόριθμο RSA. Κάθε κόμβος Node έχει ένα Wallet.
- `balance()`: υπολογίζει το συνολικό υπόλοιπο προσθέτοντας τις τιμές όλων των UTXO.

Blockchain

- `view_transactions()`: για το ζητούμενο της εργασίας, επιστρέφει όλες τις συναλλαγές καταχωρημένες στις αλυσίδα (δηλαδή σε επικυρωμένα block).

Στο αρχείο που υλοποιεί το REST API, `rest.py`, υπάρχουν controllers για κάθε επιθυμητή λειτουργία. Παραδείγματος χάριν, με αποστολή POST request στο endpoint `"/node/transaction/create"` με τα κατάλληλα δεδομένα, ο κόμβος στον οποίο έχει αποσταλεί το αίτημα πραγματοποιεί μία συναλλαγή, ενώ όταν εκπέμπει συναλλαγές τις αποστέλλει στο endpoint `"/transaction"` όλων των άλλων, το οποίο είναι υπεύθυνο για την διαχείριση εισερχόμενης συναλλαγής. Το πρόγραμμα επίσης δέχεται παραμέτρους για το όνομα του κόμβου και την διεύθυνση και την θύρα που θα τρέξει ο server.

Πολυνηματική εκτέλεση

Νήμα (thread) είναι μία ροή ελέγχου μέσα σε μία διεργασία, δηλαδή μία βασική μονάδα χρησιμοποίησης της CPU. Μοιράζεται με άλλα νήματα που ανήκουν στην ίδια διεργασία κώδικα, δεδομένα, και

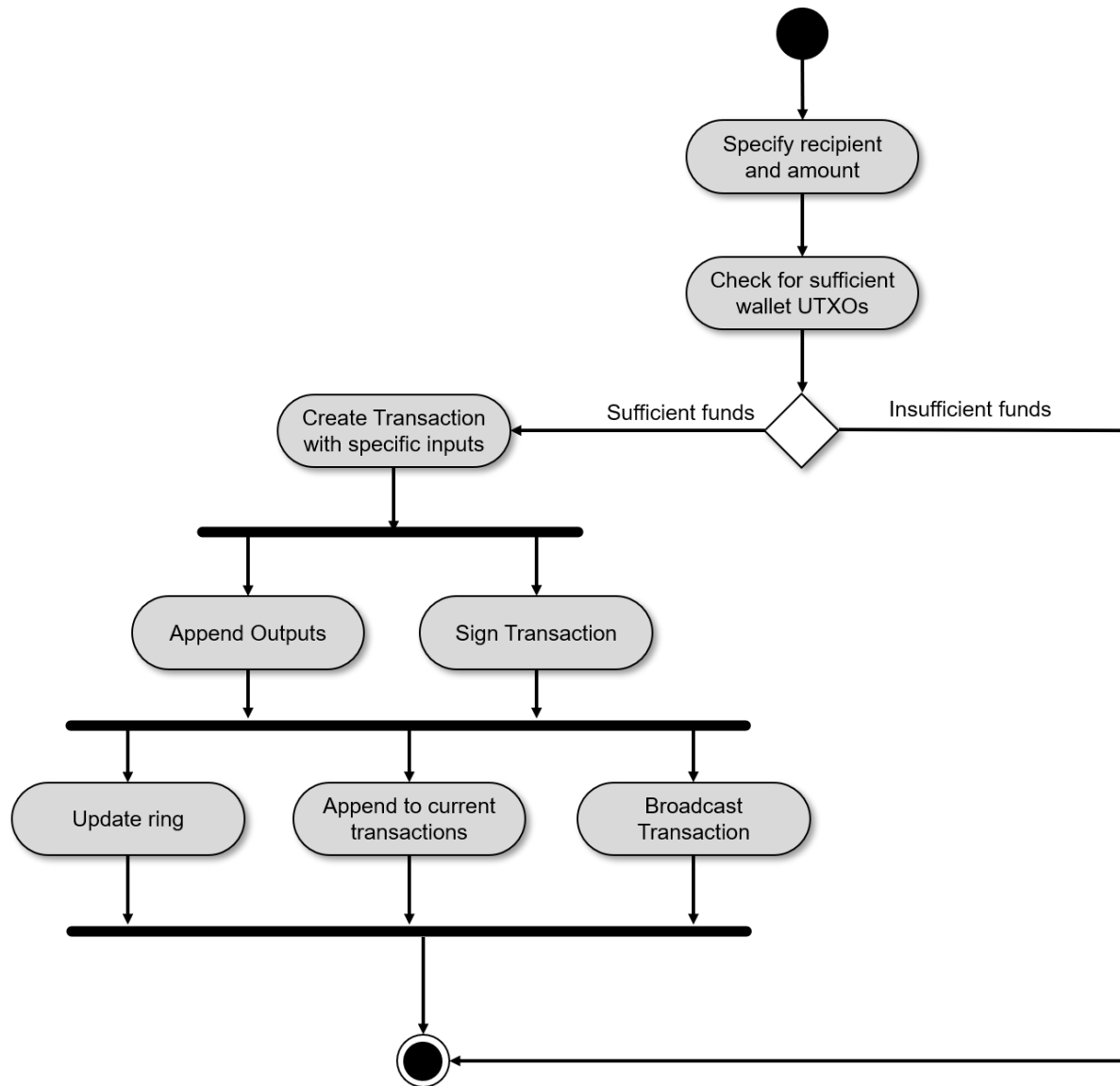
αρχεία, αλλά διαθέτει δικούς του καταχωρητές και στοίβα. Ορισμένες μέθοδοι έχουν υλοποιηθεί με multithreading. Η `experiment.py` που εκτελεί τα πειράματα αποστέλλοντας HTTP requests στους κόμβους πρέπει να το κάνει αυτό παράλληλα για όλους και όχι σειριακά. Επομένως ανοίγει τα διαφορετικά δεδομένα αρχεία `txt` συναλλαγών παράλληλα και αποστέλλει `create transaction` requests. Αυτό γίνεται με την βιβλιοθήκη `concurrent futures` της Python. Προκειμένου να μην υπερφορτώνεται το δίκτυο με αιτήματα, αλλά και για να προσδοθεί αληθοφάνεια στην προσομοίωση, μεταξύ των αιτημάτων αυτών επιβάλλεται χρονοκαθυστέρηση με την συνάρτηση `sleep`. Αυτό έχει επίδραση στο τελικό αποτέλεσμα, ωστόσο επειδή σε όλα τα πειράματα γίνεται με τον ίδιο τρόπο, η σχέση του με τις παραμέτρους είναι η ίδια και μπορεί να εξαχθεί.

Άλλη μία μέθοδος που είναι σημαντικό να υλοποιηθεί με αυτόν τον τρόπο είναι η `mine_block`, καθώς πρέπει ο κόμβος να μπορεί να εξυπηρετεί αιτήματα καθώς κάνει `mining`, δηλαδή την αναζήτηση του κατάλληλου `nonce` με επαναληπτικές εφαρμογές `hashing`.

Βέβαια πρέπει να διευκρινισθεί ότι η υλοποίηση νημάτων της Python επιτρέπει στην πραγματικότητα μόνο ένα νήμα να τρέχει κώδικα Python ανά πάσα στιγμή. Τα `threads` της Python είναι γνήσια `operating system threads`, αλλά όλα πρέπει να κατέχουν ένα μοναδικό κλειδωμά αμοιβαίου αποκλεισμού (`mutex lock`), το “Global Interpreter Lock” (GIL) της Python virtual machine, για να εκτελέσουν κώδικα Python. Λόγω αυτής της σχεδιαστικής επιλογής, τα μέρη του προγράμματος που εκτελούν κώδικα Python δεν μπορούν να κατανεμηθούν σε πολλούς πυρήνες ενός επεξεργαστή. Ωστόσο, τα μέρη που καλούν κώδικα C εκτελούνται σε `threads` ανεξάρτητα από το GIL. Επομένως στην εργασία ήταν σκόπιμο να εκτελούνται με πολυνηματικό τρόπο οι μέθοδοι που καλούν κώδικα C, που είναι όλες όσες χρησιμοποιούν κρυπτογραφικές μεθόδους από την βιβλιοθήκη `PyCryptodome`: `hash`, `sign transaction`, `verify signature`, `mine block`.

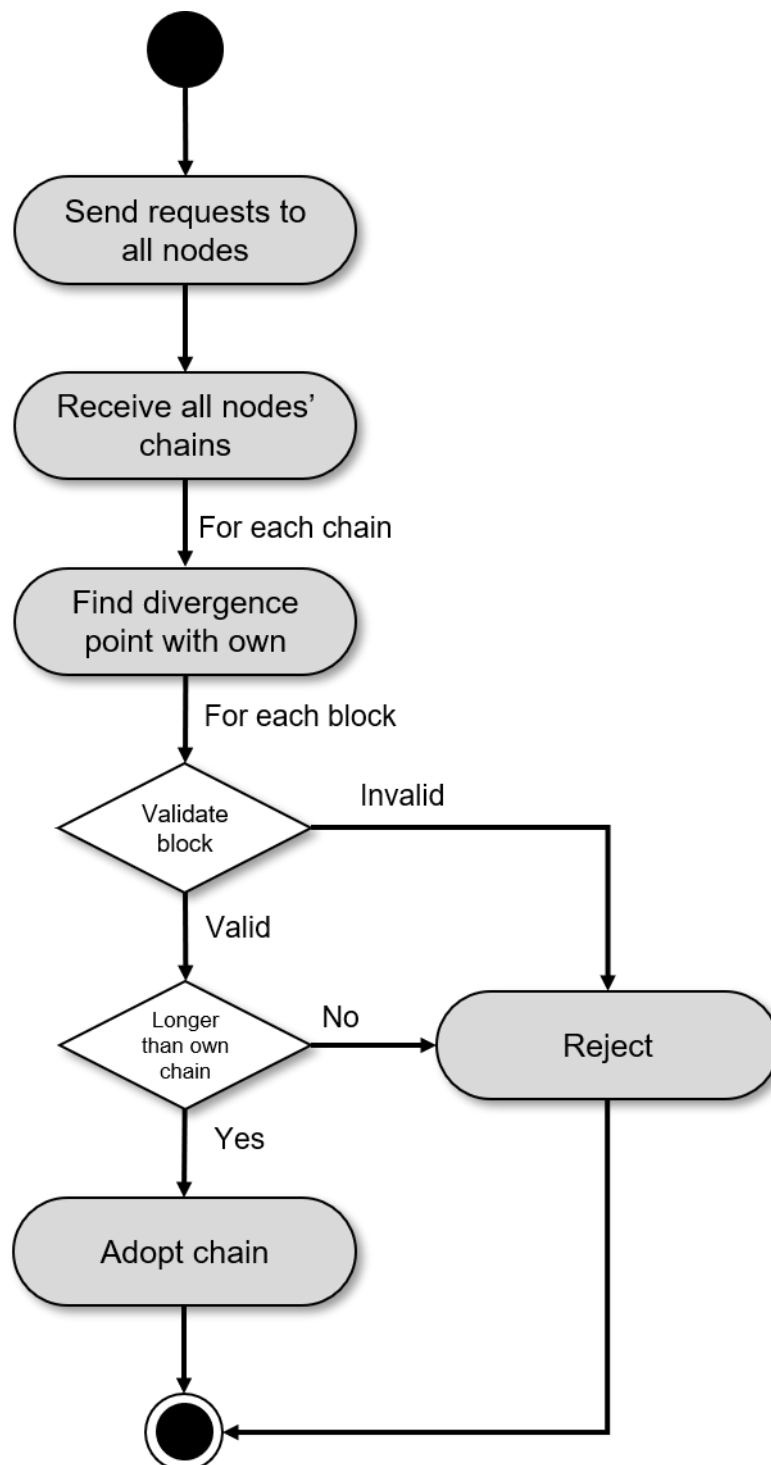
Διαγράμματα activity UML

Create Transaction:



Οι οριζόντιες γραμμές μεταξύ των λειτουργιών υποδηλώνουν ότι οι εντολές αυτές σε λογικό επίπεδο γίνονται στο ίδιο στάδιο, παράλληλα, είναι δηλαδή ανεξάρτητες.

Resolve conflict:



Πειράματα

Εκτελέσθηκαν πειράματα με το σύστημα στην υποδομή του εργαστηρίου με σκοπό την αξιολόγηση της απόδοσης και της κλιμακωσιμότητας ως προς το πλήθος κόμβων του συστήματος.

Παράμετροι

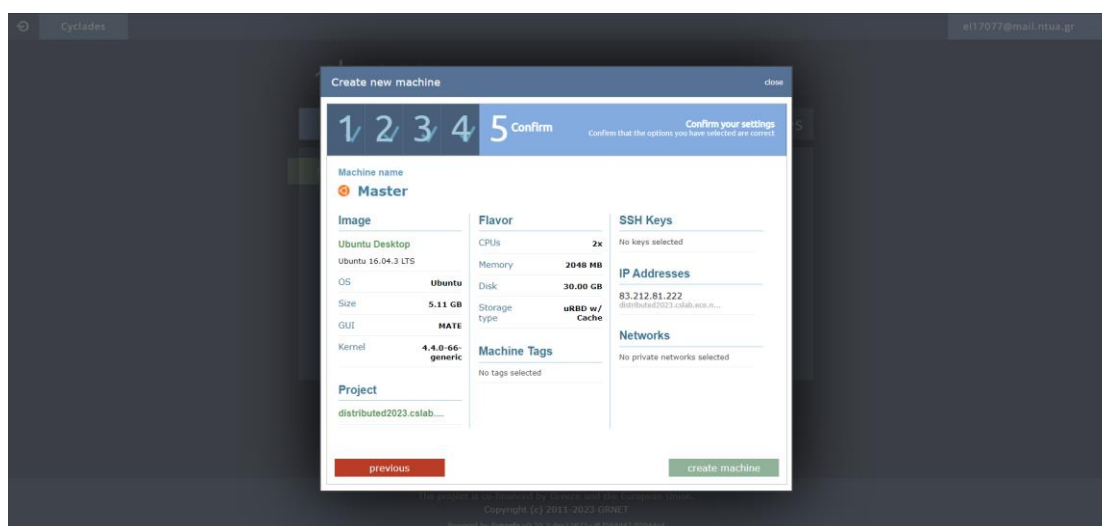
Οι παράμετροι του συστήματος που διαφοροποιούνταν μεταξύ των διαφορετικών εκτελέσεων είναι το πλήθος κόμβων $N \in \{5, 10\}$, η χωρητικότητα block σε συναλλαγές, $C \in \{1, 5, 10\}$, και η δυσκολία εξόρυξης, δηλαδή το απαιτούμενο πλήθος διαδοχικών μηδενικών στα πρώτα ψηφία της δεκαεξαδικής αναπαράστασης του hash ενός block ώστε να επικυρωθεί το "proof of work", $D \in \{4, 5\}$. Πραγματοποιήθηκε ένα πείραμα για κάθε δυνατό συνδυασμό αυτών, δηλαδή συνολικά 12 πειράματα.

Μετρώμενα μεγέθη

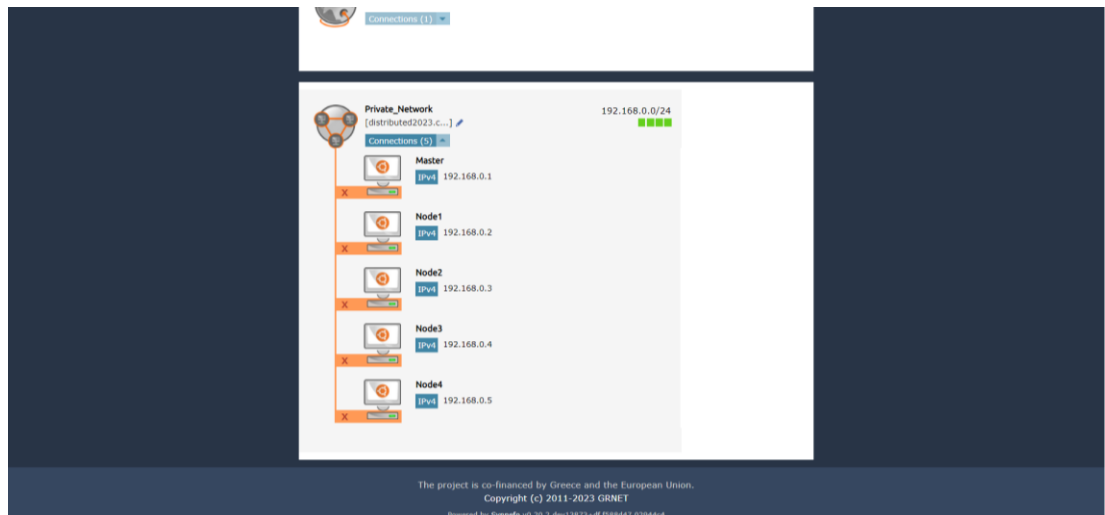
Για κάθε κόμβο χωριστά μετράται ο μέσος όρος χρόνων πρόσθεσης νέου block στην αλυσίδα, δηλαδή εξόρυξης - υπολογισμού του hash του block με επαναληπτικές προσαυξήσεις του αριθμού nonce έως ότου ικανοποιείται η προϋπόθεση proof of work, μετάδοσης αυτού και επαλήθευσής του. Το μεγαλύτερο μέρος του χρόνου αυτού δαπανάται για τον υπολογισμό του proof of work. Στα συνολικά αποτελέσματα παρουσιάζεται ο μέσος όρος των μετρήσεων αυτών για κάθε συνδυασμό παραμέτρων. Για το σύστημα συνολικά μετράται η ρυθμαπόδοση - throughput, χρησιμοποιώντας το πεδίο timestamp που αρχικοποιείται σε κάθε αντικείμενο Transaction κατά την δημιουργία του. Μετράται σε συναλλαγές ανά δευτερόλεπτο, και υπολογίζεται ως το πλήθος των συναλλαγών στην αλυσίδα διά το χρονικό διάστημα μεταξύ της πρώτης και της τελευταίας χρονικώς συναλλαγής στην αλυσίδα κατά την ολοκλήρωση του πειράματος

Υποδομή εργαστηρίου

Τα πειράματα εκτελέσθηκαν σε εικονικές μηχανές της πλατφόρμας infrastructure-as-a-service "okeanos-knossos". Διατέθηκαν πέντε μηχανές με μία μοναδική public IP για πρόσβαση στο διαδίκτυο, και με την δυνατότητα δημιουργίας ενός ιδιωτικού εσωτερικού δικτύου μεταξύ των. Τα χαρακτηριστικά κάθε μίας είναι τα εξής: OS Ubuntu Desktop 16, 2 CPUs, 2GB RAM. Η εγκατάσταση των απαραίτητων εργαλείων σε κάθε μία έγινε μεταφέροντας την public IP μεταξύ των μηχανών ώστε να δίνεται πρόσβαση στο διαδίκτυο. Έπρεπε να εγκατασταθούν τα απαραίτητα εργαλεία μέσω apt install για την μεταγλώττιση της Python 3.9 μετά την λήψη του πηγαίου κώδικα της αυτής από τον επίσημο ιστότοπο, και ύστερα για την εγκατάσταση των προσθέτων βιβλιοθηκών μέσω pip. Αυτό έγινε για κάθε μηχανή. Στο τέλος οι μηχανές συνδέθηκαν μέσω εσωτερικού δικτύου ώστε να επικοινωνούν μεταξύ των κατά τα πειράματα. Αυτή η λειτουργία παρέχεται από την γραφική διεπαφή του ωκεανού. Για την πρόσβαση στις μηχανές και την χρήση αυτών χρησιμοποιήθηκε το πρόγραμμα remote desktop "X2Go".



Επισκόπηση δημιουργίας του "master node" που διαθέτει IP. Κάθε VM του ωκεανού διαθέτει και ένα domain name.



Το ιδιωτικό δίκτυο.

Εκτέλεση

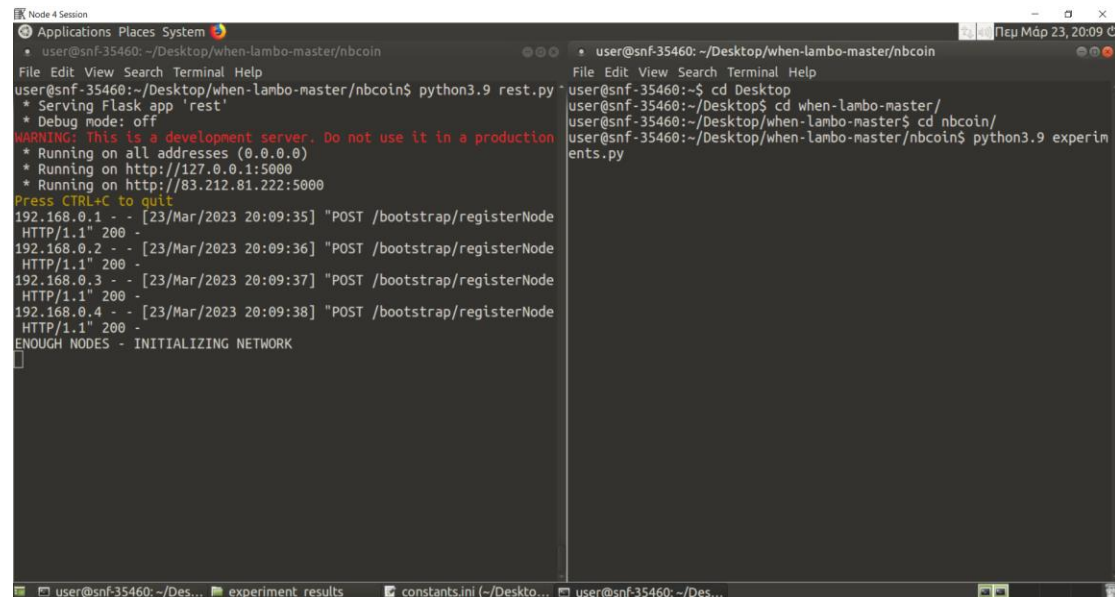
Για τα πειράματα με 5 κόμβους, ένα instance της εφαρμογής έτρεχε σε κάθε VM. Για τα πειράματα με 10 κόμβους, σε κάθε VM έτρεχαν δύο, κάθε ένα ως διαφορετική διεργασία και με τον HTTP server σε διαφορετική θύρα, οπότε προσομοιώθηκαν επαρκώς οι 10 διαφορετικοί κόμβοι, εφόσον τα δύο instances ανά μηχανή ήταν πρακτικά ανεξάρτητα μεταξύ των, καθώς η μηχανή διέθετε 2 CPUs.

Η «ενεργοποίηση» ενός κόμβου γίνεται εκκινώντας τον HTTP server εκτελώντας το αρχείο `rest.py`. Οι παράμετροι του εκάστοτε πειράματος περιέχονται στο αρχείο `constants.ini`. Από αυτό αναγιγνώσκονται από τις διεργασίες οι κατάλληλες τιμές σε κάθε πείραμα, οπότε η μοναδική αλλαγή που χρειάζεται το σύστημα μεταξύ πειραμάτων είναι η ενημέρωση των τιμών στο αρχείο αυτό σε κάθε μηχανή.

Η ίδια η διεξαγωγή των πειραμάτων γίνεται από το αρχείο `experiments.py`, το οποίο περιέχει λίστα με όλες τις διευθύνσεις που τρέχουν οι HTTP servers των κόμβων. Αποστέλλει εν παραλλήλω αιτήματα σε όλους, στο endpoint του REST API `/experiment` το οποίο αναλαμβάνει να ανοίξει το δεδομένο αρχείο συναλλαγών που

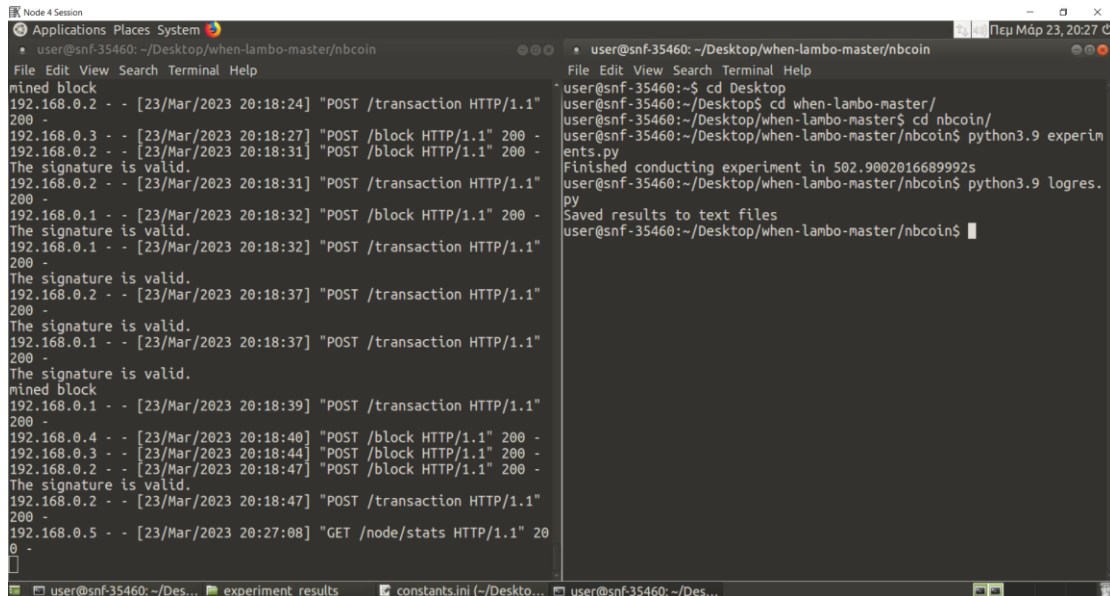
αντιστοιχεί στον κάθε κόμβο (τα αρχεία αυτά διακρίνονται ανά πλήθος κόμβων και αναγνωριστικό κόμβου) και να αποπειραθεί να πραγματοποιήσει τις εμπεριεχόμενες συναλλαγές. Τέλος το αρχείο `logres.py` εκτελείται μετά την ολοκλήρωση του πειράματος και αποστέλλει μοναδικό αίτημα σε κάθε κόμβο που καλεί την μέθοδο `get_statistics` και ως εκ τούτου αποθηκεύεται για κάθε κόμβο στο σύστημα αρχείων της αντίστοιχης μηχανής αρχείο κειμένου με τα στατιστικά για το πείραμα.

Όπως προανεφέρθη, προκειμένου κατά την εκτέλεση των πειραμάτων να προσομοιωθούν πραγματικές συνθήκες blockchain κρυπτοσυναλλάγματος, αλλά και ώστε να μην υπερφορτωθεί το δίκτυο από αιτήματα, εισάγεται μικρή χρονοκαθυστέρηση με την συνάρτηση `time.sleep`, τυχαίας τιμής μεταξύ 0 και 6 δευτερολέπτων. Αυτό έχει επίδραση στον χρόνο εκτέλεσης, αλλά καθώς εφαρμόζεται με τον ίδιο τρόπο σε όλα τα πειράματα, εξάγονται αναλλοίωτα τα συμπεράσματα για την επίδραση των υπό μελέτη παραμέτρων.



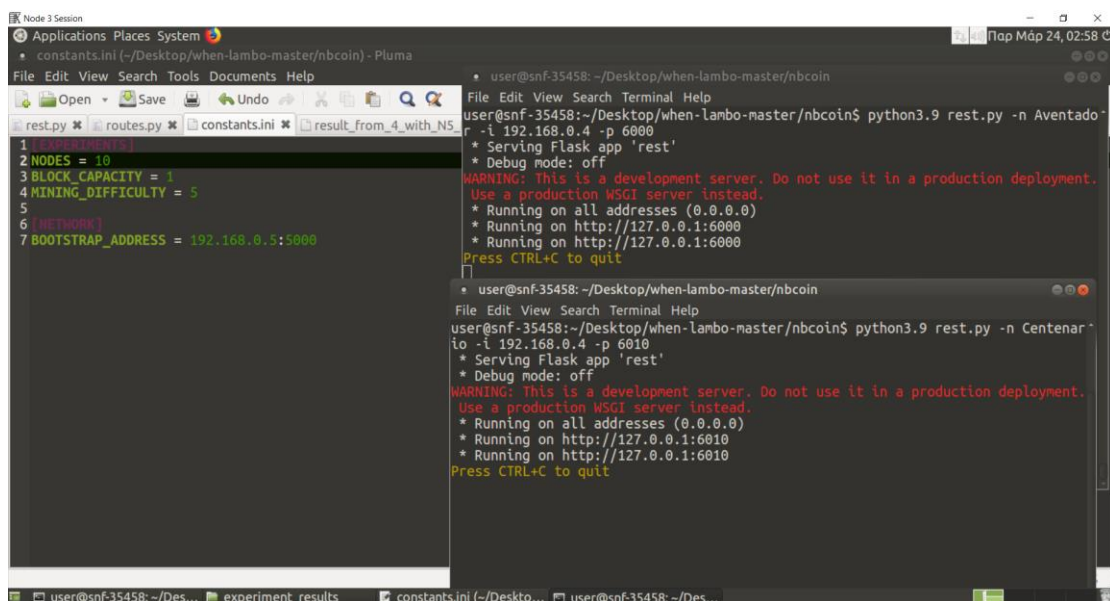
```
user@snf-35460: ~/Desktop/when-lambo-master/nbcoin
File Edit View Search Terminal Help
user@snf-35460:~/Desktop/when-lambo-master/nbcoin$ python3.9 rest.py
* Serving Flask app 'rest'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://83.212.81.222:5000
Press CTRL+C to quit
192.168.0.1 - - [23/Mar/2023 20:09:35] "POST /bootstrap/registerNode
HTTP/1.1" 200 -
192.168.0.2 - - [23/Mar/2023 20:09:36] "POST /bootstrap/registerNode
HTTP/1.1" 200 -
192.168.0.3 - - [23/Mar/2023 20:09:37] "POST /bootstrap/registerNode
HTTP/1.1" 200 -
192.168.0.4 - - [23/Mar/2023 20:09:38] "POST /bootstrap/registerNode
HTTP/1.1" 200 -
ENOUGH NODES - INITIALIZING NETWORK
```

Κατά την έναρξη εκτέλεσης ενός πειράματος κάθε κόμβος αποστέλλει αίτημα καταχώρησης στον bootstrap. Σύμφωνα με τις οδηγίες έχει γίνει η παραδοχή ότι το σύστημα αρχίζει την λειτουργία του αφότου συνδεθούν όλοι.



```
File Edit View Search Terminal Help
mined block
192.168.0.2 - - [23/Mar/2023 20:18:24] "POST /transaction HTTP/1.1" 200 -
192.168.0.3 - - [23/Mar/2023 20:18:27] "POST /block HTTP/1.1" 200 -
192.168.0.2 - - [23/Mar/2023 20:18:31] "POST /block HTTP/1.1" 200 -
The signature is valid.
192.168.0.2 - - [23/Mar/2023 20:18:31] "POST /transaction HTTP/1.1" 200 -
192.168.0.1 - - [23/Mar/2023 20:18:32] "POST /block HTTP/1.1" 200 -
The signature is valid.
192.168.0.1 - - [23/Mar/2023 20:18:32] "POST /transaction HTTP/1.1" 200 -
The signature is valid.
192.168.0.2 - - [23/Mar/2023 20:18:37] "POST /transaction HTTP/1.1" 200 -
The signature is valid.
192.168.0.1 - - [23/Mar/2023 20:18:37] "POST /transaction HTTP/1.1" 200 -
The signature is valid.
mined block
192.168.0.1 - - [23/Mar/2023 20:18:39] "POST /transaction HTTP/1.1" 200 -
192.168.0.4 - - [23/Mar/2023 20:18:40] "POST /block HTTP/1.1" 200 -
192.168.0.3 - - [23/Mar/2023 20:18:44] "POST /block HTTP/1.1" 200 -
192.168.0.2 - - [23/Mar/2023 20:18:47] "POST /block HTTP/1.1" 200 -
The signature is valid.
192.168.0.2 - - [23/Mar/2023 20:18:47] "POST /transaction HTTP/1.1" 200 -
192.168.0.5 - - [23/Mar/2023 20:27:08] "GET /node/stats HTTP/1.1" 200 -
```

Εικόνα κατά την ολοκλήρωση του ιδίου πειράματος. Φαίνεται ο πεπλεγμένος τρόπος με τον οποίο αποστέλλονται και λαμβάνονται αιτήματα από κάθε κόμβο.

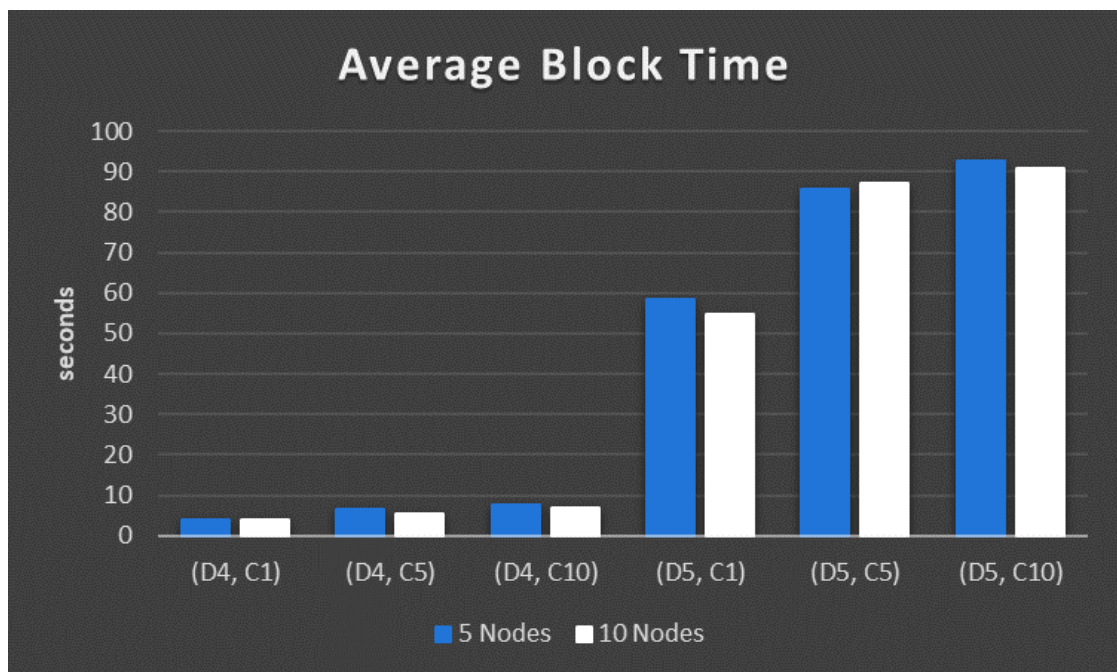
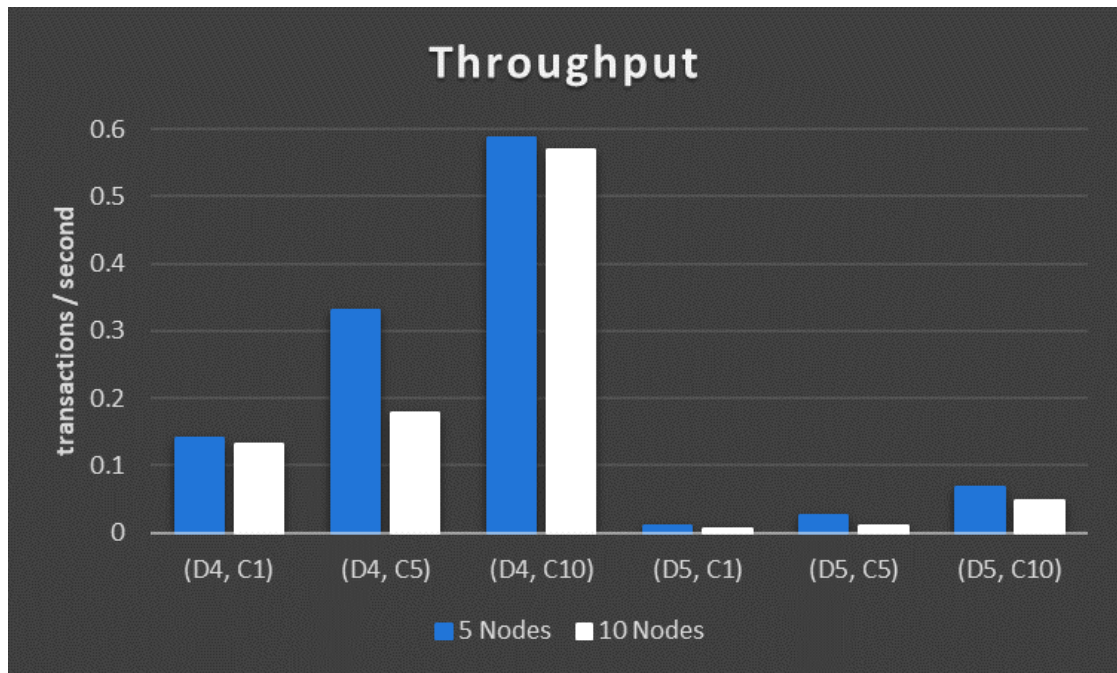


```
File Edit View Search Tools Documents Help
rest.py routes.py constants.ini result_from_4_with_N5
1. MINING_DIFFICULTY = 10
2. NODES = 10
3. BLOCK_CAPACITY = 1
4. MINING_DIFFICULTY = 5
5.
6. NETWORK
7. BOOTSTRAP_ADDRESS = 192.168.0.5:5000
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:6000
* Running on http://127.0.0.1:6000
Press CTRL+C to quit
user@snf-35458: ~/Desktop/when-lambo-master/nbcoin
user@snf-35458:~/Desktop/when-lambo-master/nbcoin$ python3.9 rest.py -n Aventado
-i 192.168.0.4 -p 6000
* Serving Flask app 'rest'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:6010
* Running on http://127.0.0.1:6010
Press CTRL+C to quit
```

Παράδειγμα εκτέλεσης με δύο κόμβους ανά VM. Με την ίδια μέθοδο έγιναν και οι δοκιμές τοπικά κατά την ανάπτυξη της εφαρμογής.

Αποτελέσματα

Συγκεντρωτικά όλες οι μετρήσεις των πειραμάτων σε πίνακες, μαζί με τους συνολικούς χρόνους εκτέλεσης κάθε ενός στο αρχείο excel experiment_results στο github repository της εργασίας (σελίδα 2).



Συμπεράσματα

Ως προς τον βαθμό «δυσκολίας»: η παράμετρος αυτή φαίνεται πως επιφέρει την μεγαλύτερη αλλαγή στα αποτελέσματα, τόσο στους 5 όσο και στους 10 κόμβους. Με δυσκολία 4, δηλαδή 4 απαιτούμενα μηδενικά στα πρώτα ψηφία του hash για επικύρωση, το throughput είναι πολύ υψηλότερο και ο χρόνος block πολύ χαμηλότερος σε

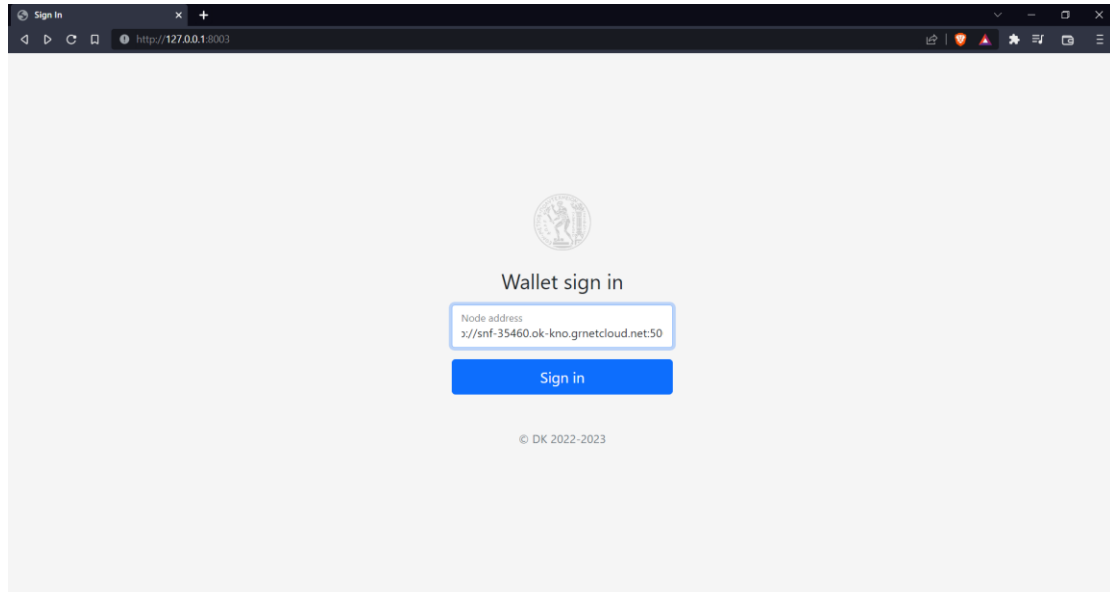
σχέση με την δυσκολία 5, *ceteris paribus*. Επειδή η συνθήκη *proof of work* αφορά την δεκαεξαδική αναπαράσταση του hash, και η συνάρτηση SHA-256 είναι κρυπτογραφική συνάρτηση κατακερματισμού άρα έχει την ιδιότητα να κατανέμει ομοιόμορφα τις εισόδους στο πεδίο τιμών της, η πιθανότητα εύρεσης συμβολοσειράς hash με 5 διαδοχικά μηδενικά στην αρχή είναι 16 φορές μικρότερη από ό,τι με 4. Πράγματι, οι χρόνοι block για $D = 5$ είναι περίπου 16 φορές μεγαλύτεροι από αυτούς για $D = 4$, *ceteris paribus*, αφού όπως έχει εξηγηθεί ο χρόνος εξόρυξης είναι το κυριότερο μέρος του block time. Αυτό το χρονικό κόστος έχει την ανάλογη επίδραση στην ρυθμαπόδοση, ωστόσο για την ασφάλεια του συστήματος σύμφωνα με το πρωτόκολλο blockchain, είναι επιθυμητό.

Ως προς το πλήθος κόμβων: φαίνεται πως αυτό δεν επηρεάζει τον χρόνο block, πράγμα αναμενόμενο καθώς η διαδικασία που συνιστά το μεγαλύτερο μέρος του χρόνου αυτού, δηλαδή η εξόρυξη, επιτελείται ανεξάρτητα σε κάθε κόμβο. Το throughput, για ίδιες τιμές difficulty και capacity, είναι λίγο μεγαλύτερο στο σύστημα με 5 κόμβους. Αυτό οφείλεται στο γεγονός πως κάθε μήνυμα (συναλλαγή ή block) πρέπει να αποστέλλεται σε όλους τους κόμβους και να επαληθεύεται από αυτούς προτού επικυρωθεί, οπότε περισσότεροι κόμβοι συνεπάγονται κάποια χρονοκαθυστέρηση στο δίκτυο. Ωστόσο το σύστημα λειτουργεί ικανοποιητικά. Συνεπώς το κόστος της κλιμακωσιμότητας είναι μείωση στην ρυθμαπόδοση.

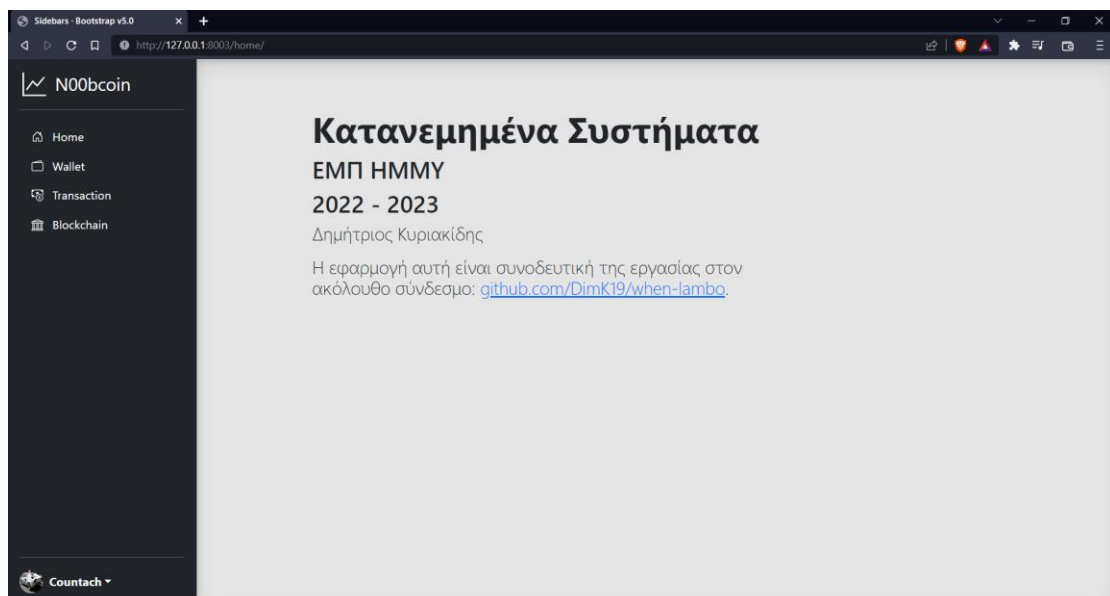
Ως προς την χωρητικότητα block: μεγαλύτερη χωρητικότητα block σε συναλλαγές οδηγεί σε υψηλότερες τιμές throughput, καθώς απαιτείται λιγότερο συχνά η εξόρυξη νέου block. Όμως, αυξάνει ελαφρώς τον χρόνο block, πιθανώς επειδή ο υπολογισμός του hash για ένα block που περιέχει περισσότερα δεδομένα, και στην συνέχεια η επικύρωση αυτών από τους υπόλοιπους κόμβους, απαιτεί περισσότερο χρόνο.

Παράρτημα

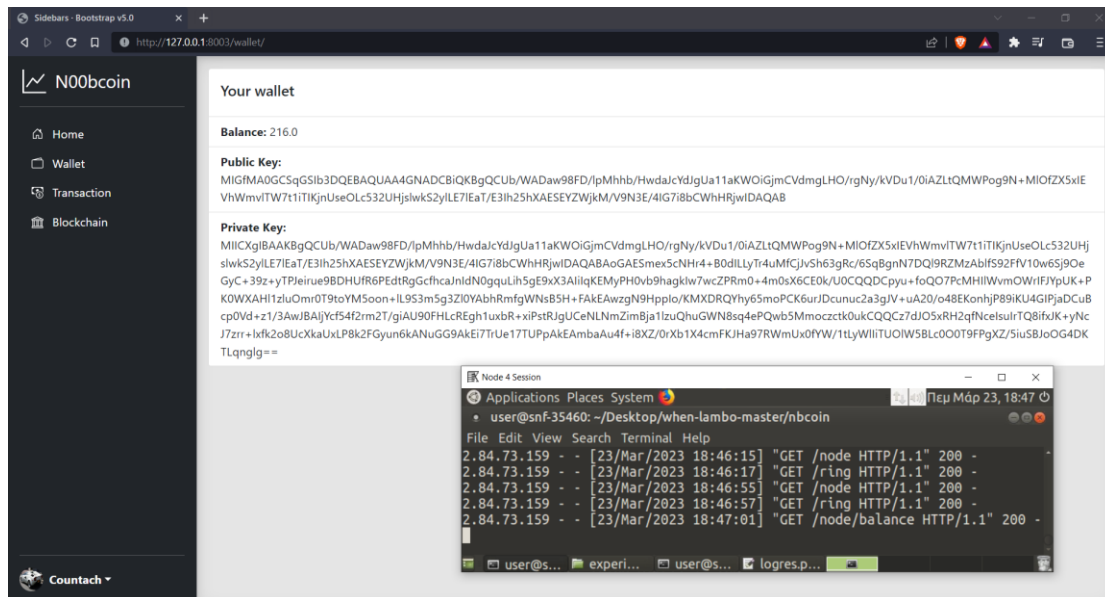
Επίδειξη λειτουργίας της γραφικής διεπαφής



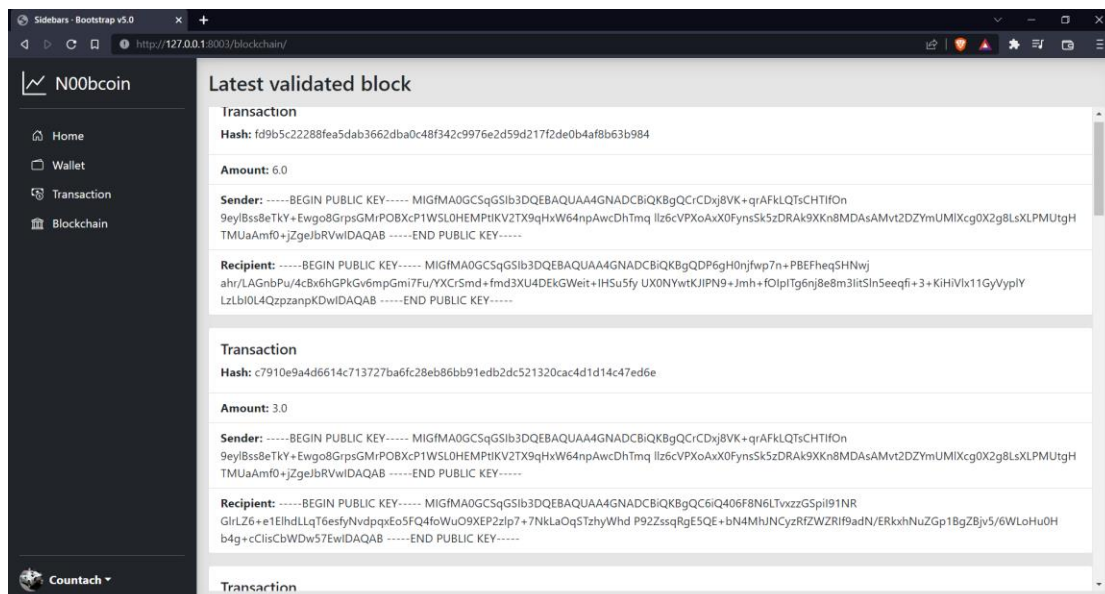
Σύνδεση στον γραφικό client που τρέχει στον τοπικό υπολογιστή, εισάγοντας την διεύθυνση της μίας VM που διαθέτει public IP.



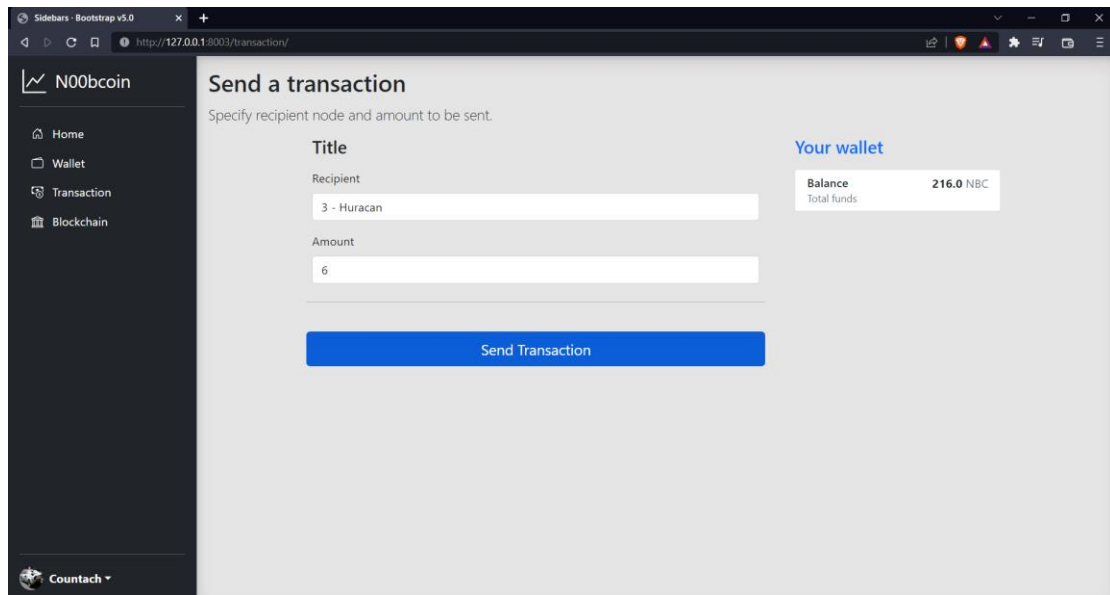
Αρχική σελίδα.



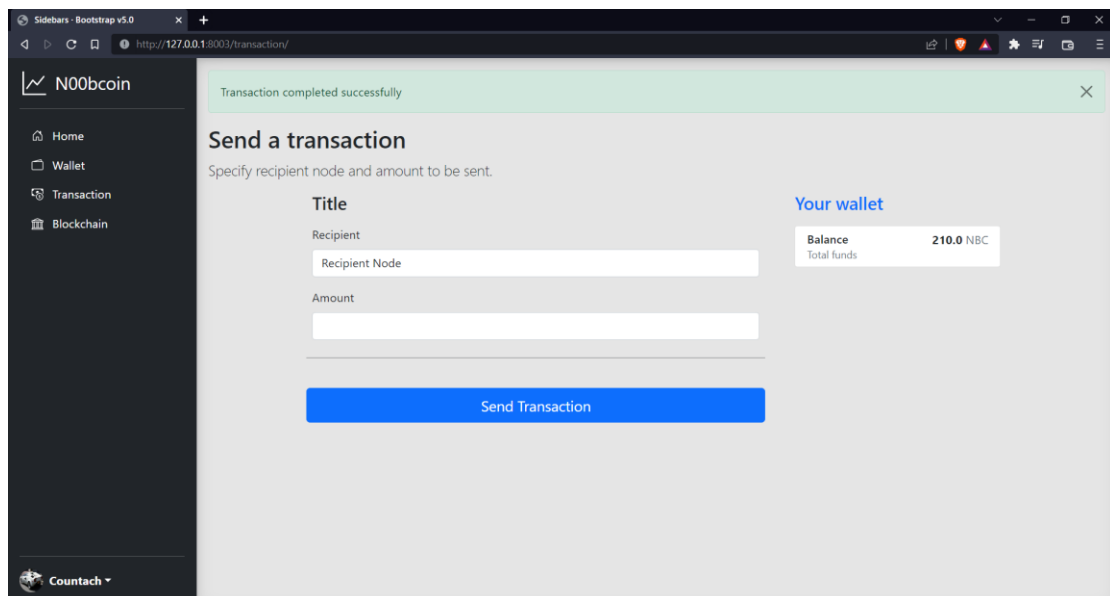
Σελίδα πληροφοριών wallet. Στο μικρό παράθυρο φαίνεται η γραμμή εντολών της εικονικής μηχανής όπου τρέχει ο HTTP server και τα αιτήματα που έχει λάβει από τον client.



Κατάλογος συναλλαγών στο τελευταίο επικυρωμένο block της αλυσίδας. Η εικόνα αυτή προέρχεται ύστερα από την ολοκλήρωση του πειράματος για $N = 5$, $D = 4$, $C = 10$.

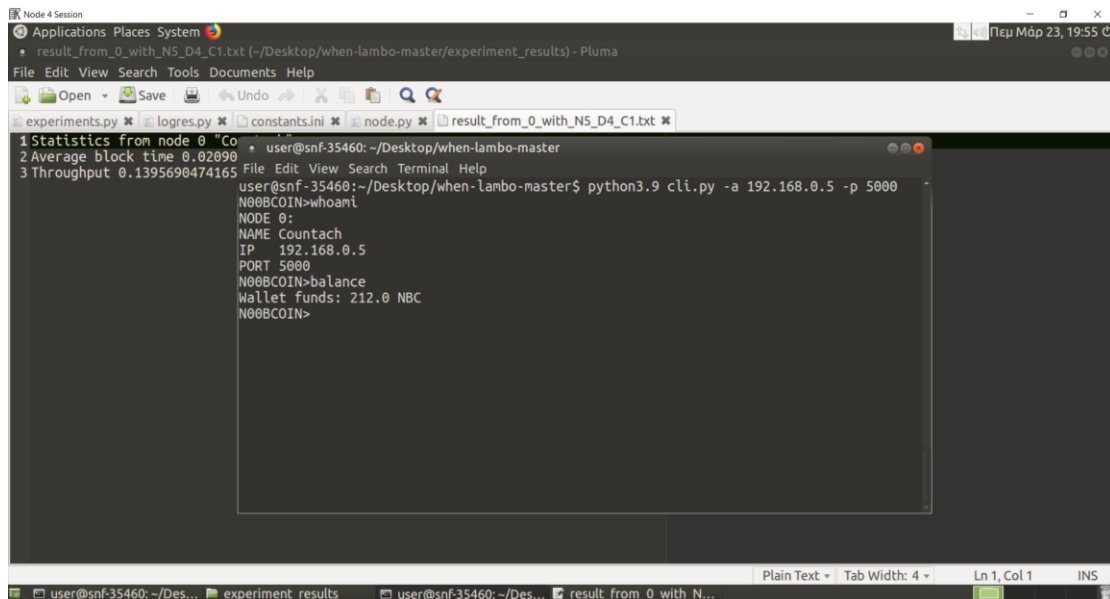


Σελίδα αποστολής συναλλαγών. Δίνει την επιλογή καθορισμού παραλήπτη, μέσω ονόματος και id το οποίο αντιστοιχίζεται στο public key, και ποσού.



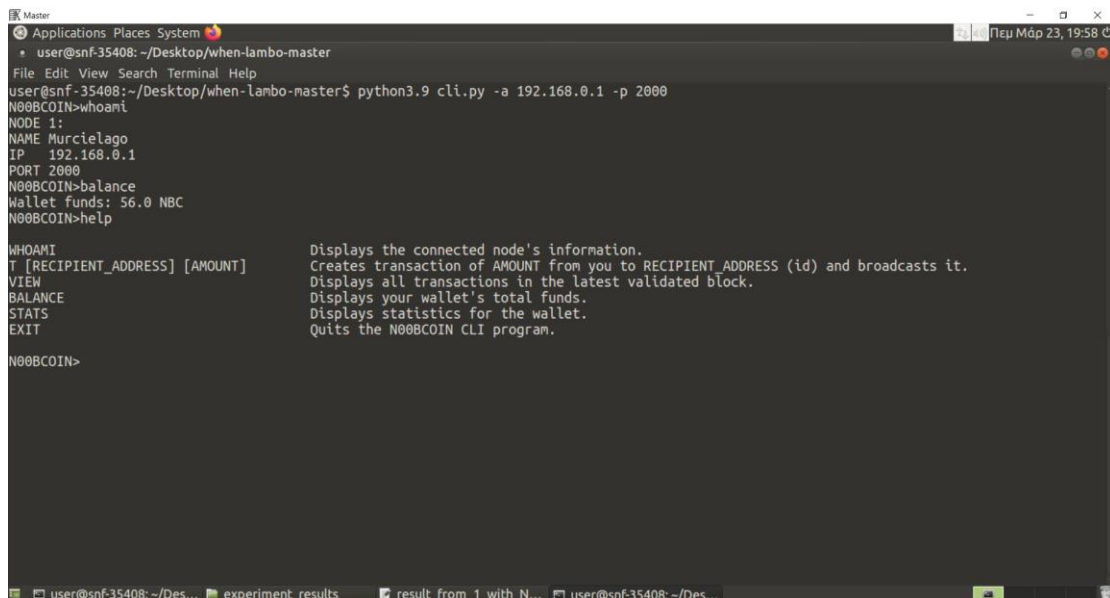
Μετά την ολοκλήρωση της συναλλαγής ενημερώνεται το υπόλοιπο του wallet.

Επίδειξη λειτουργίας CLI



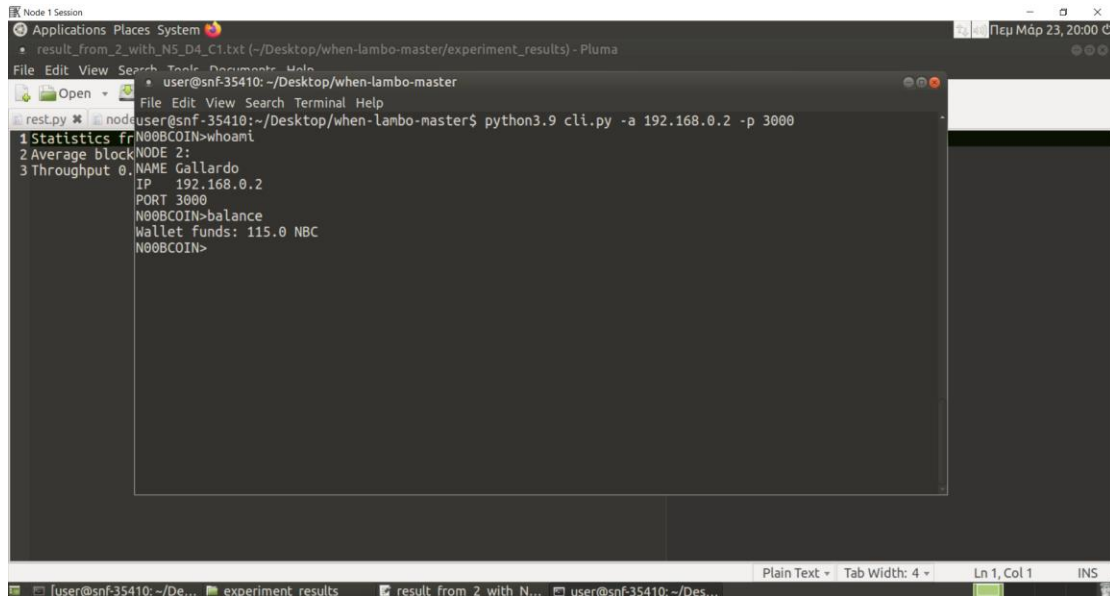
The screenshot shows a desktop environment with a Node.js application window titled "Node 4 Session". The application window has a menu bar (File, Edit, View, Search, Tools, Documents, Help) and a toolbar. It contains several tabs: experiments.py, logres.py, constants.ini, node.py, and result_from_0_with_N5_D4_C1.txt. The experiments.py tab is active, displaying statistics from node 0: 1 Statistics from node 0 "Co", 2 Average block time 0.02090, and 3 Throughput 0.1395690474165. A terminal window is open in the foreground, showing the command `python3.9 cli.py -a 192.168.0.5 -p 5000` being executed. The output shows node information for NODE 0: NAME Countach, IP 192.168.0.5, PORT 5000, N00BCOIN>balance, and Wallet funds: 212.0 NBC. The terminal window also shows the command `N00BCOIN>whoami` and its output.

Εκτέλεση του CLI στην εικονική μηχανή και σύνδεση με τον κόμβο που τρέχει σε αυτήν, καθορίζοντας θύρα και διεύθυνση IP στο εσωτερικό ιδιωτικό δίκτυο. Λόγω αυτής της διαφοράς με το ιδιωτικό δίκτυο χρειάζεται μικρή αλλαγή στο πρόγραμμα όταν τρέχει στο δίκτυο VMs από όταν τρέχει τοπικά για δοκιμές.



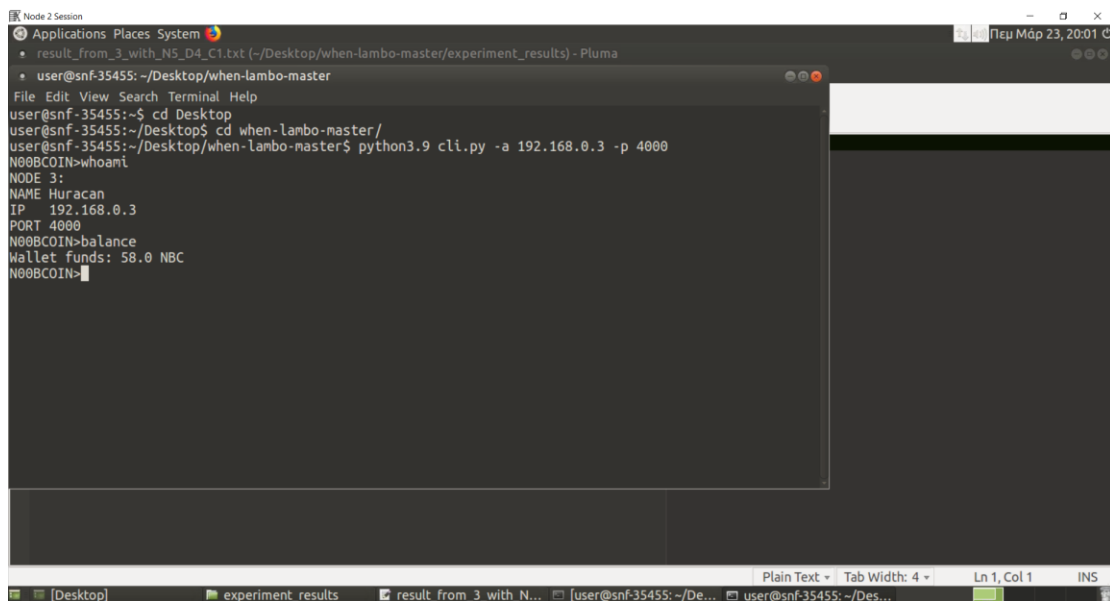
The screenshot shows a terminal window titled "Master" with a user prompt `user@snf-35408: ~/Desktop/when-lambo-master`. The user enters the command `python3.9 cli.py -a 192.168.0.1 -p 2000`. The output shows node information for NODE 1: NAME Murcielago, IP 192.168.0.1, PORT 2000, N00BCOIN>balance, and Wallet funds: 56.0 NBC. The user then enters `N00BCOIN>help`, which displays a list of commands and their descriptions: WHOAMI (Displays the connected node's information.), T [RECIPIENT_ADDRESS] [AMOUNT] (Creates transaction of AMOUNT from you to RECIPIENT_ADDRESS (id) and broadcasts it.), VIEW (Displays all transactions in the latest validated block.), BALANCE (Displays your wallet's total funds.), STATS (Displays statistics for the wallet.), and EXIT (Quits the N00BCOIN CLI program.). The user then enters `N00BCOIN>`.

Το ίδιο για τον δεύτερο κόμβο.



```
Node 1 Session
Applications Places System
• result_from_2_with_N5_D4_C1.txt (-/Desktop/when-lambo-master/experiment_results) - Pluma
File Edit View Search Terminal Help
Open user@snf-35410: ~/Desktop/when-lambo-master
rest.py * node user@snf-35410:~/Desktop/when-lambo-master$ python3.9 cli.py -a 192.168.0.2 -p 3000
1 Statistics for N00BCOIN>whoami
2 Average block NODE 2:
3 Throughput 0. NAME Gallardo
IP 192.168.0.2
PORT 3000
N00BCOIN>balance
Wallet funds: 115.0 NBC
N00BCOIN>
```

Για τον τρίτο κόμβο με την αντίστοιχη IP και θύρα.



```
Node 2 Session
Applications Places System
• result_from_3_with_N5_D4_C1.txt (-/Desktop/when-lambo-master/experiment_results) - Pluma
File Edit View Search Terminal Help
Open user@snf-35455: ~/Desktop/when-lambo-master
user@snf-35455:~$ cd Desktop
user@snf-35455:~/Desktop$ cd when-lambo-master/
user@snf-35455:~/Desktop/when-lambo-master$ python3.9 cli.py -a 192.168.0.3 -p 4000
N00BCOIN>whoami
NODE 3:
NAME Huracan
IP 192.168.0.3
PORT 4000
N00BCOIN>balance
Wallet funds: 58.0 NBC
N00BCOIN>
```

Για τον τέταρτο κόμβο.


```
rest.py * result_from_4_with_N5 user@snf-35458:~/Desktop/when-lambo-master$ python3.9 cli.py -a 192.168.0.4 -p 6000
1 Statistics from node 4 "AveN008COIN>whoami
2 Average block time 4.829688;NODE 4:
3 Throughput 0.14090572311202;NAME Aventador
IP 192.168.0.4
PORT 6000
N008COIN>balance
Wallet funds: 59.0 NBC
N008COIN>
```

Τέλος για τον πέμπτο κόμβο. Μπορεί να επαληθευθεί έχοντας κάνει αυτήν την επισκόπηση η ορθότητα της εκτέλεσης ελέγχοντας ότι το άθροισμα των τελικών υπολοίπων των κόμβων ισούται με την αρχική ποσότητα coins στο δίκτυο (εφόσον δεν αμείβονται με επιπλέον συνάλλαγμα οι κόμβοι που κάνουν επιτυχή εξόρυξη). Πράγματι, $212 + 56 + 115 + 58 + 59 = 500$.

Βιβλιογραφία

Έντυπη

Burmester, Mike, κ.α. *Σύγχρονη Κρυπτογραφία*. Αθήναι, Παπασωτηρίου, 2011.

Coulouris, George, κ.α. *Κατανεμημένα Συστήματα Αρχές και Σχεδίαση*. Μετεφρασμένο από τον Κωνσταντίνο Κοντογιάννη. Αθήναι, Da Vinci, 2020.

Jones, Gareth, and Mary Jones. *Elementary Number Theory*. London, Springer, 1998.

Kleppmann, Martin. *Designing Data-Intensive Applications: The Big Ideas behind Reliable, Scalable, and Maintainable Systems*. Sebastopol, O'Reilly, 2017.

Lutz, Mark. *Learning Python*. 5th ed., Sebastopol, O'Reilly, 2017.

Lutz, Mark. *Programming Python*. 4th ed., Sebastopol, O'Reilly, 2010.

Pilone, Dan, and Neil Pitman. *UML 2.0 in a Nutshell*. Sebastopol, O'Reilly, 2005.

Pressman, Roger, και Bruce Maxim. *Τεχνολογία Λογισμικού Μία Ποσοτική Προσέγγιση*. 8η εκδ., Μετεφρασμένο από τον Αγαμέμνονα Μήλιο. Αθήναι, Τζιόλας, 2018.

Silberschatz, Abraham, κ.α. *Λειτουργικά Συστήματα*. 9η έκδ, Μετεφρασμένο από τον Ιωάννη Σαμαρά. Αθήναι, Γκιούρδας, 2013.

Silberschatz, Abraham, κ.α. *Συστήματα Βάσεων Δεδομένων*. 6η έκδ, Μετεφρασμένο από την Μαίρη Γκλαβά. Αθήναι, Γκιούρδας, 2018.

Ηλεκτρονική

Kumar, Rahul. "How To Install Python 3.9 on Ubuntu 18.04". TecAdmin.net, tecadmin.net/how-to-install-python-3-9-on-ubuntu-18-04/. Accessed 21 Mar. 2023.

Nakamoto, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System". N.p., 31 Oct. 2008, bitcoin.org/bitcoin.pdf. Accessed 25 Mar. 2023.

N.a. "Cyclades User Guide". Okeanos-Knossos, GRNET, okeanos-knossos.grnet.gr/support/user-guide/. Accessed 20 Mar. 2023.

N.a. "Documentation". PyCryptodome, [readthedocs, pycryptodome.readthedocs.io/en/latest/](https://readthedocs.org/projects/pycryptodome/en/latest/). Accessed 18 Mar. 2023.