



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Κατανεμημένα Συστήματα

Noobcash: A Blockchain Implementation

Ιωάννης Μιχαήλ Καζελίδης
03117885

Ιωάννης Γκιορτζής
03117152

Μάριος Κερασιώτης
03117890



ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1	Εισαγωγή	3
2	Σχεδιασμός Συστήματος.....	3
2.1	Εποπτεία Συστήματος.....	3
2.2	Noobcash Core	4
2.2.1	Ανάλυση Κλάσεων	4
2.2.1.1	Κλάση Node	4
2.2.1.2	Κλάση Block	6
2.2.1.3	Κλάση Blockchain.....	6
2.2.1.4	Κλάση Wallet	6
2.2.1.5	Κλάση Transaction	6
2.2.1.6	Κλάση Ring.....	7
2.2.2	Αλληλεπιδράσεις Κλάσεων και Αλληλουχία Εντολών	7
2.2.3	REST API και Σχεδιαστικό Μοτίβο Αντάπτορα	9
2.3	Noobcash Client.....	9
2.4	Noobcash Tester.....	10
3	Αξιολόγηση Συστήματος	10
3.1	Περιβάλλον Πειραμάτων	10
3.2	Απόδοση	11
3.3	Κλιμακωσιμότητα.....	12
3.4	Σχολιασμός Πειραμάτων	12
4	Αναφορές.....	13

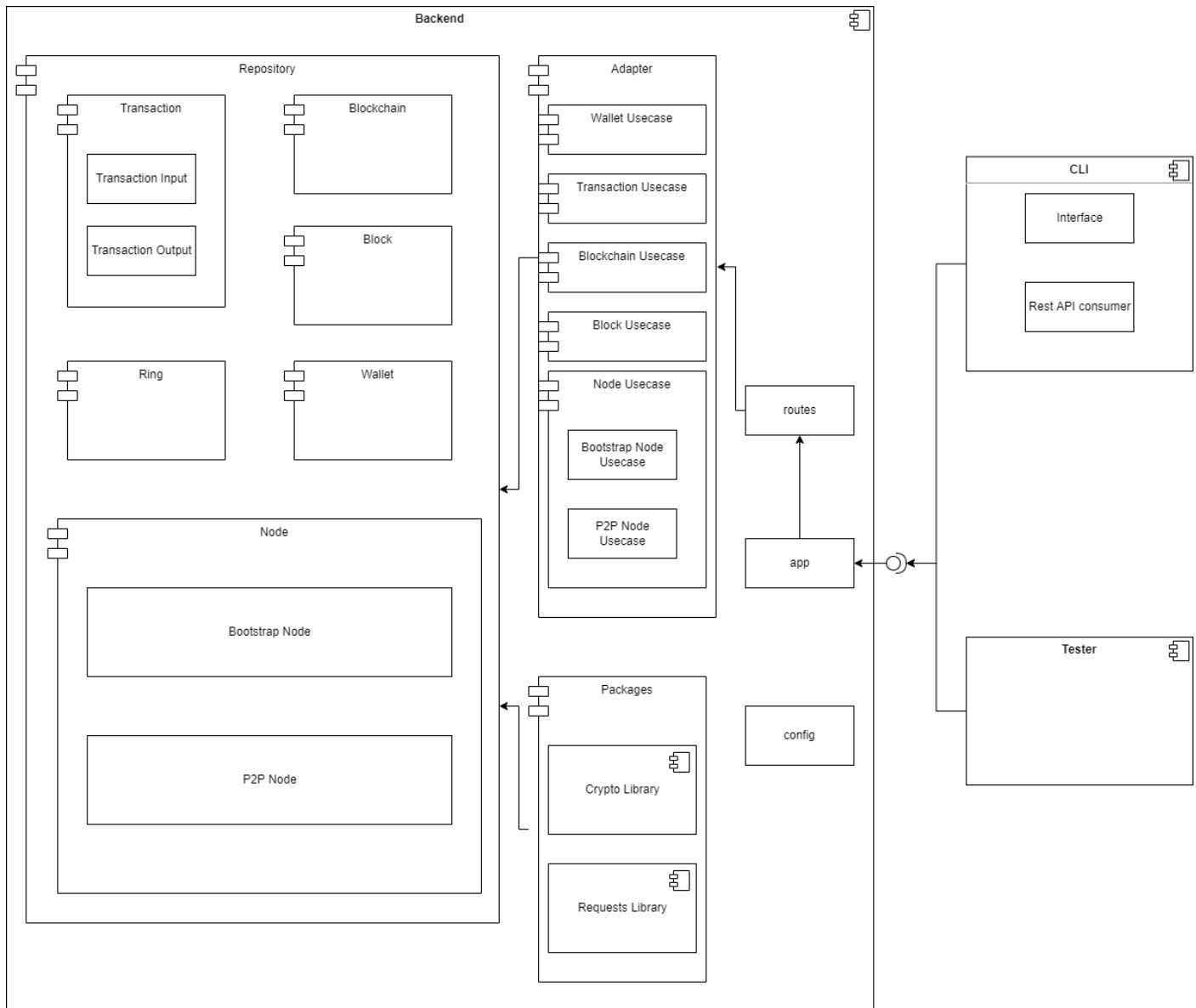
1 ΕΙΣΑΓΩΓΗ

Η ανάπτυξη της τεχνολογίας και η διάδοση του διαδικτύου στην σύγχρονη εποχή έχουν ως αποτέλεσμα πολλές πτυχές της ζωής μας να γίνονται ηλεκτρονικά, μια από τις οποίες είναι και η δημιουργία χρηματικών συναλλαγών. Στην προσπάθεια ικανοποίησης των αναγκών αυτών με γνώμονα την ασφάλεια και την φερεγγυότητα, δημιουργήθηκε το blockchain, μια τεχνολογία πάνω στην οποία βασίστηκαν τα περισσότερα σύγχρονα κρυπτονομίσματα. Το blockchain αποτελεί μια κατακεντρωμένη βάση που δίνει την δυνατότητα στους χρήστες της να συνδέονται σε ένα ομότιμο δίκτυο (P2P network) και να εκτελούν δοσοληψίες μεταξύ τους με ασφάλεια, χωρίς να χρειάζεται να μεσολαβήσει μια έμπιστη τρίτη οντότητα (Trusted Third Party, TTP), όπως μια τράπεζα στην περίπτωση των χρηματικών συναλλαγών. Η τεχνολογία αυτή αναπτύχθηκε για να υποστηρίξει κυρίως ένα κρυπτονομίσμα ονόματι Bitcoin, το οποίο από την περίοδο που εμφανίστηκε έχει αλλάξει τον τρόπο με τον οποίο διεξάγονται πλέον οι συναλλαγές. Σκοπός, λοιπόν, της εργασίας αυτής είναι η δημιουργία του Noobcash, ενός απλού συστήματος blockchain, στο οποίο θα καταγράφονται οι δοσοληψίες μεταξύ των κόμβων του συστήματος και θα εξασφαλίζεται η συμφωνία (consensus) μεταξύ τους με χρήση Proof-of-Work, αποκτώντας έτσι ομοιότητες με το Bitcoin σε αδρά σημεία.

2 ΣΧΕΔΙΑΣΜΟΣ ΣΥΣΤΗΜΑΤΟΣ

2.1 ΕΠΟΠΤΕΙΑ ΣΥΣΤΗΜΑΤΟΣ

Το σύστημα αποτελείται από τρία υποσυστήματα: το [Noobcash Core](#), το [Noobcash Client](#) και το [Noobcash Tester](#). Το Noobcash Core είναι ουσιαστικά το backend του συστήματος, και αποτελεί το βασικότερο σύστημα όλων. Συγκεκριμένα, περιέχει τον κώδικα που επιτρέπει στον χρήστη να δημιουργήσει το σύστημα Noobcash, να συνδεθεί σε αυτό και να δημιουργήσει συναλλαγές. Τα υπόλοιπα συστήματα, Noobcash Client και Noobcash Tester, επικοινωνούν με το Noobcash Core μέσω REST API και είτε προσφέρουν στον χρήστη μια διεπαφή που του επιτρέπει να εκτελεί βασικές λειτουργίες που του χρειάζονται, είτε ελέγχουν το σύστημα και δίνουν μετρικές για αυτό. Περαιτέρω πληροφορίες για κάθε ένα από αυτά τα υποσυστήματα θα δοθούν στις επόμενες ενότητες. Πιο συγκεκριμένα, ωστόσο, φαίνεται στο παρακάτω διάγραμμα η βασική δομή του συστήματός μας και το πώς συνδέονται τα επιμέρους κομμάτια που το αποτελούν.



Διάγραμμα 1 Component Diagram

2.2 NOOBASH CORE

Η κύρια εφαρμογή είναι η υλοποίηση του Noobcash Core, που είναι ουσιαστικά το backend του συστήματός μας. Η πρώτη διεργασία Noobcash Core πρέπει να ξεκινήσει ως bootstrap διεργασία και πρέπει να γνωρίζει τον αριθμό όλων των άλλων διεργασιών Noobcash Core. Οι υπόλοιπες δημιουργούνται ως απλές διεργασίες γνωρίζοντας μόνο την διεύθυνση της bootstrap διεργασίας (κόμβου). Τέτοιες διεργασίες θα τις πούμε P2P διεργασίες (κόμβους). Η αρχικοποίηση των διεργασιών αυτών φαίνεται σε επόμενη ενότητα (αλληλεπίδρασης κλάσεων), στο διάγραμμα 2. Οι βασικές οντότητες με τα χαρακτηριστικά τους που απαρτίζουν το Noobcash Core και κατ' επέκταση το σύστημά μας φαίνονται παρακάτω.

2.2.1 Ανάλυση Κλάσεων

2.2.1.1 Κλάση Node

Η κλάση Node αποτελεί την κύρια κλάση του συστήματος, αντιπροσωπεύει τον κάθε χρήστη και αποτελεί τον συνδετικό κρίκο μεταξύ όλων των υπόλοιπων κλάσεων και λειτουργιών. Μάλιστα, για να είναι ξεκάθαρος ο διαχωρισμός μεταξύ του bootstrap και των p2p κόμβων, η κλάση Node αποτελεί κλάση-γονιός για τις ξεχωριστές κλάσεις-παιδιά Bootstrap και P2P, οι οποίες κληρονομούν τις βασικές λειτουργίες ενός Node, αλλά σε αυτές

υλοποιούνται και οι εξειδικευμένες λειτουργίες του κάθε ρόλου. Αναλυτικότερα για τις κλάσεις αυτές θα δούμε στην συνέχεια. Πιο συγκεκριμένα τώρα, ο κάθε κόμβος Node κατέχει ένα wallet, ένα αντίγραφο του blockchain, το ring που περιέχει πληροφορίες για τους υπόλοιπους κόμβους που είναι συνδεδεμένοι στο σύστημα και μια ουρά με pending transactions, δηλαδή transactions τα οποία έχουν γίνει validated (είτε εισερχόμενα από άλλους κόμβους είτε δημιουργημένα από τον ίδιο τον κόμβο) αλλά δεν αποτελούν μέρος του blockchain, εφόσον δεν έχει γίνει ακόμα mine ενός block που τις περιέχει. Τέλος, περιέχει και ένα αντικείμενο με τα βασικά στοιχεία του, που βρίσκονται και στο ring, τα οποία είναι το id του, το host name του, το port στο οποίο λειτουργεί, το public key του, καθώς και το σύνολο των unspent transactions μαζί με το balance του wallet.

Την λειτουργικότητα της κλάσης Node την παρέχουν οι ακόλουθες συναρτήσεις:

- **broadcast:** Πραγματοποιεί broadcast σε όλο το υπόλοιπο δίκτυο είτε επισυναπτόμενα δεδομένα, είτε ζητάει για να πάρει επισυναπτόμενα δεδομένα.
- **create_transaction:** Δημιουργεί συναλλαγή ενός ποσού από Noobcash Coins (NBC) στον επιθυμητό παραλήπτη σε περίπτωση που έχει το απαιτούμενο ποσό.
- **register_transaction:** Καταχωρεί μια συναλλαγή που δημιουργήθηκε από άλλο κόμβο σε περίπτωση που είναι valid και την βάζει στην ουρά των pending transactions για να προστεθεί σε block.
- **validate_transaction:** Επαληθεύει την ορθότητα ενός transaction. Η επαλήθευση περιλαμβάνει τόσο την επαλήθευση της υπογραφής όσο και τον έλεγχο ότι ο αποστολέας διαθέτει όντως το ποσό που μεταφέρει στον παραλήπτη.
- **mine_block:** Αυξάνει το nonce του block κατά 1 έως ότου το hash του nonce να αρχίζει από συγκεκριμένο αριθμό από μηδενικά, που καθορίζεται κάθε φορά από την παράμετρο mining difficulty.
- **set_blockchain:** Θέτει το blockchain που λαμβάνει στην θέση του υπάρχοντος blockchain.
- **handle_pending_transactions:** Ελέγχει αν έχει καλυφθεί το μέγιστο capacity ενός block, και σε αυτήν την περίπτωση το δημιουργεί με όσες συναλλαγές απαιτούνται από την ουρά των pending transactions, το κάνει mine, το προσθέτει στο blockchain και ύστερα το κάνει broadcast στο υπόλοιπο δίκτυο.
- **update_transactions:** Ενημερώνει τα UTXOs του wallet και το balance και τα UTXOs των κόμβων του ring.
- **register_incoming_block:** Καταχωρεί το εισερχόμενο block στο blockchain και διαχειρίζεται κατάλληλα τα transactions της ουράς των pending transactions.
- **_request_blockchain:** Ζητά το blockchain από όλους τους υπόλοιπους κόμβους του δικτύου.
- **_request_ring_and_transactions_from_node:** Ζητά το ring και τα pending transactions του κόμβου με το δοσμένο id.
- **resolve_conflict:** Σε περίπτωση που το index του block που μας ήρθε δεν είναι το index που περιμέναμε ή το block δεν είναι valid και άρα δεν μπορούμε να το βάλουμε στην αλυσίδα μας, καλείται η resolve conflict έτσι ώστε να βρούμε τον κόμβο με την μεγαλύτερη valid αλυσίδα και να πάρουμε από αυτόν την αλυσίδα, το ring και τα pending transactions του.

2.2.1.1.1 Κλάση Bootstrap

Η κλάση Bootstrap αποτελεί παιδί της κλάσης Node και ως εκ τούτου περιέχει τις λειτουργικότητές της, αλλά υλοποιεί και νέες μεθόδους που μόνο αυτή η κλάση χρησιμοποιεί. Πιο συγκεκριμένα, όταν αρχικοποιείται ο bootstrap κόμβος, ενημερώνει το ring με τα στοιχεία του, δημιουργεί το genesis block με transaction που έχει το απαιτούμενο ποσό από NBC με την βοήθεια της συνάρτησης **_create_genesis_block**, και στην συνέχεια το προσθέτει στο blockchain. Κύρια λειτουργία του bootstrap είναι να αναθέτει id στους νέους κόμβους του συστήματος και να τους προσθέτει στο ring, κάτι το οποίο υλοποιεί με την **register_node**. Στην συνέχεια, όταν όλοι οι απαιτούμενοι κόμβοι συνδεθούν στο σύστημα, με την βοήθεια της συνάρτησης **_init_blockchain** αρχίζει η αρχικοποίηση του συστήματος, η οποία περιλαμβάνει τόσο την αποστολή του ring του και το blockchain του σε όλους τους κόμβους του συστήματος, αλλά και την αποστολή των συναλλαγών με τα αρχικά χρήματα σε αυτούς, κάτι το οποίο υλοποιείται αντίστοιχα με την βοήθεια των συναρτήσεων **_broadcast_current_state** και **_send_first_transactions**.

2.2.1.1.2 Κλάση P2P

Η κλάση P2P αποτελεί και αυτή παιδί της κλάσης Node και ως εκ τούτου περιέχει τις λειτουργικότητές της, αλλά υλοποιεί και νέες μεθόδους που μόνο αυτή η κλάση χρησιμοποιεί. Πιο συγκεκριμένα, όταν αρχικοποιείται ο p2p κόμβος ζητάει από τον bootstrap να του στείλει το id του, κάτι το οποίο υλοποιείται με την βοήθεια της συνάρτησης **`_get_node_info_from_bootstrap`**, που κάνει register το node στο ring. Άλλη μια μέθοδος που υλοποιεί είναι η **`set_ring`** που πρέπει να διαθέτει μόνο ο P2P Node για να πάρει το ring από τον bootstrap όταν συνδεθούν όλοι οι κόμβοι στο σύστημα.

2.2.1.2 Κλάση Block

Η κλάση Block είναι ένα instance του blockchain, αποτελεί την βασική του μονάδα και περιέχει όλες εκείνες τις πληροφορίες που θα πρέπει να αποθηκευτούν τελικά στην αλυσίδα. Εμπεριέχει μια λίστα από transactions, το μέγιστο πλήθος των οποίων καθορίζεται από την παράμετρο capacity. Πιο συγκεκριμένα, κάθε Block περιέχει την μεταβλητή index, δηλαδή την «θέση» του στο blockchain, την timestamp, που δηλώνει τον χρόνο δημιουργίας του block και μια λίστα από transactions που περιέχει. Επίσης, περιέχει και έναν ειδικό αριθμό που λέγεται nonce, ο οποίος αρχικοποιείται ως 0 και δηλώνει τον αριθμό των επαναλήψεων που απαιτούνται μέχρι το current hash να ξεκινάει από τόσα μηδενικά όσα καθορίζονται από την παράμετρο mining difficulty. Όσο πιο μεγάλος είναι αυτός ο αριθμός, τόσος περισσότερος χρόνος απαιτείται για να βρούμε το nonce και κατά συνέπεια τόσο πιο ασφαλές είναι το σύστημά μας. Τέλος, περιέχει δύο hash, το previous hash που δείχνει στο block με δείκτη index - 1 και το current hash που αρχικοποιείται κατά την δημιουργία του αντικειμένου με χρήση της μεθόδου **`calculate_hash`** και προκύπτει από τις παραπάνω μεταβλητές. Η κλάση περιέχει μια ακόμη μέθοδο, την **`get_transactions`**, με την βοήθεια της οποίας μπορούμε να ανακτήσουμε τα transactions του block.

2.2.1.3 Κλάση Blockchain

Η κλάση Blockchain βρίσκεται νοητικά ένα επίπεδο πιο πάνω από την κλάση Block, εφόσον περιέχει μια λίστα από αντικείμενα Block. Την λειτουργικότητα της κλάσης την παρέχουν κάποιες μέθοδοι που έχουν υλοποιηθεί. Πιο συγκεκριμένα, η μέθοδος **`add_block`** μας επιτρέπει να εισάγουμε ένα block στο blockchain, αρκεί αυτό να είναι έγκυρο. Για να ελεγχθεί η εγκυρότητα ενός block έχει οριστεί η μέθοδος **`validate_block`**, η οποία ελέγχει τόσο αν το current hash ενός block είναι αυτό που προκύπτει από την μέθοδο **`calculate_hash`** της κλάσης Block, όσο και το αν previous hash του block είναι ίδιο με το current hash του προηγούμενου block. Μια αντίστοιχη μέθοδος υπάρχει και για την εγκυρότητα της αλυσίδας. Συγκεκριμένα, η μέθοδος **`validate_chain`** χρησιμοποιώντας την **`validate_block`** ελέγχει αν όλα τα blocks του blockchain είναι έγκυρα, εκτός του Genesis. Τέλος, υπάρχουν 2 ακόμη μέθοδοι στην κλάση, η **`get_chain`** και η **`get_last_block`**, οι οποίες αντίστοιχα επιστρέφουν το blockchain και το τελευταίο block του blockchain, συναρτήσεις που είναι χρήσιμες για άλλες λειτουργίες του συστήματός μας.

2.2.1.4 Κλάση Wallet

Η κλάση Wallet καθιστά την ιδέα του πορτοφολιού στο σύστημά μας. Κάθε κόμβος κατά την αρχικοποίησή του έχει δημιουργήσει ένα ζεύγος από public/private keys. Το public key απαιτείται για να λάβει ο κάτοχος του wallet χρήματα από άλλον χρήστη, ενώ το private key είναι αυτό που πρέπει να κρατείται κρυφό και είναι αυτό που επιτρέπει να υπογράφονται οι συναλλαγές προς εκτέλεση για να ελεγχθεί η εγκυρότητά τους, δηλαδή πως ο κάτοχος του wallet είναι αυτός που όντως δημιούργησε την συναλλαγή. Το Wallet κάθε χρήστη περιέχει επίσης και μια ουρά από unspent transactions. Για την υλοποίηση της λειτουργικότητάς της, η κλάση αυτή υποστηρίζει την μέθοδο **`get_balance`**, η οποία επιστρέφει το άθροισμα όλων των τιμών των transactions της ουράς unspent transactions, και την μέθοδο **`update_wallet`**, που ενημερώνει την ουρά unspent transactions του wallet με κριτήριο το αν ο χρήστης είναι μέρος της συναλλαγής, είτε ως αποστολέας, είτε ως παραλήπτης.

2.2.1.5 Κλάση Transaction

Η κλάση Transaction περιέχει τις απαραίτητες πληροφορίες για την δημιουργία συναλλαγών και την μεταφορά χρημάτων από έναν χρήστη σε έναν άλλο. Συγκεκριμένα, περιέχει τις διευθύνσεις του αποστολέα και του παραλήπτη, που είναι public keys, το ποσό της συναλλαγής και το timestamp της που δείχνει τον χρόνο που πραγματοποιήθηκε. Επίσης, περιέχει ένα μοναδικό id για αυτήν, που παράγεται από την μέθοδο **`calculate_hash`** με

χρήση των πεδίων που περιέχει, την υπογραφή από τον αποστολέα, που παράγεται από την μέθοδο **sign_transaction** καθώς και δύο λίστες, την transaction inputs και την transaction outputs. Η transaction inputs περιέχει όλα τα UTXOs που χρησιμοποιούνται για να γίνει η συναλλαγή, ενώ η παραγωγή των transaction outputs γίνεται με την χρήση της μεθόδου **get_transaction_outputs**, η οποία υπολογίζει τα ρέστα για τον αποστολέα και το ποσό για τον παραλήπτη σε μορφή UTXO. Τέλος, η κλάση υποστηρίζει και την μέθοδο **verify_signature** που ελέγχει την εγκυρότητα της υπογραφής.

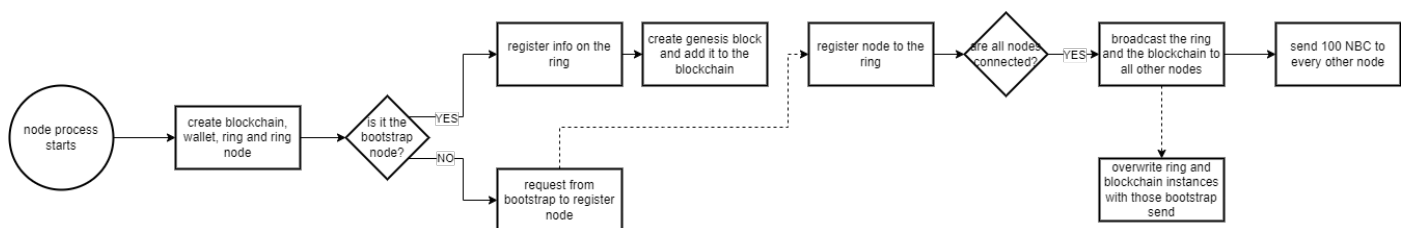
2.2.1.6 Κλάση Ring

Η κλάση Ring περιέχει το σύνολο των διεργασιών NoobCash Core που είναι συνδεδεμένες μεταξύ τους. Αυτές οι διεργασίες αναπαρίστανται ως RingNodes. Η RingNode είναι ένας τύπος κλάσης και περιέχει το id της διεργασίας, τα host, port για να επικοινωνούμε με αυτήν και το public key της. Επίσης περιέχει τα unspent transactions της σε μορφή ουράς και το balance. Με την κλάση Ring μπορούμε να παίρνουμε κάθε φορά τα nodes που θέλουμε με βάση το id τους, με χρήση της μεθόδου **get_node_by_id**, ή με βάση την public address τους με χρήση της μεθόδου **get_node_by_address**. Τέλος, δοσμένης μιας transaction μπορούμε να ανανεώσουμε την ουρά με τα unspent transactions και τα balances των nodes του ring με χρήση της μεθόδου **update_unspent_transactions**.

2.2.2 Αλληλεπιδράσεις Κλάσεων και Αλληλουχία Εντολών

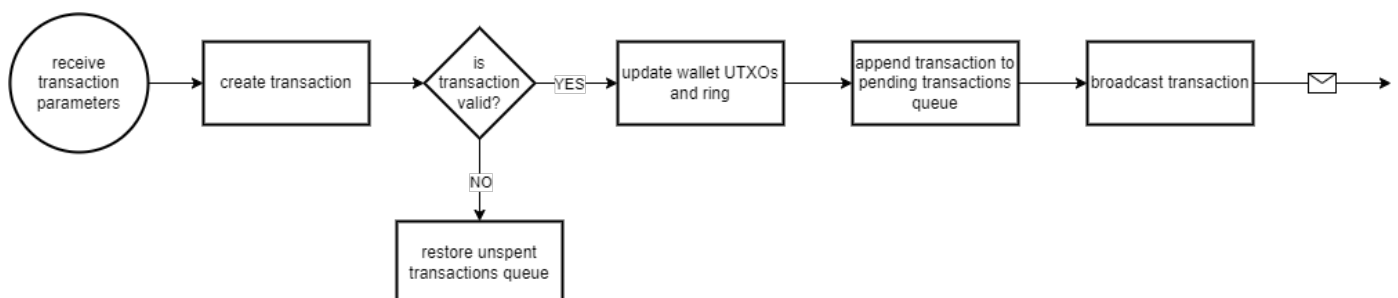
Για να λειτουργήσει σωστά το Noobcash Core πρέπει οι κλάσεις που αναφέρθηκαν παραπάνω να αλληλεπιδρούν μεταξύ τους. Δεδομένης της πολύπλοκης φύσης του project είναι αρκετά δύσκολη η περιγραφή όλων των αλληλεπιδράσεων που συμβαίνουν στο σύστημα κάθε χρονική στιγμή. Για αυτόν ακριβώς τον λόγο, και επειδή γνωρίζουμε πως η περιγραφή των κλάσεων και των μεθόδων δεν είναι αρκετή για να καταλάβει κάποιος το πώς λειτουργούν και αλληλεπιδρούν στο σύστημα μεταξύ τους, παρακάτω παρουσιάζονται αναλυτικά σε διαγράμματα οι βασικότερες λειτουργίες του συστήματος.

Αρχικά, στο διάγραμμα 2 παρουσιάζεται αναλυτικά η αλληλουχία των εντολών που ακολουθείται έτσι ώστε να αρχικοποιηθούν οι κόμβοι του συστήματος, και κατ' επέκταση το ίδιο το σύστημα.



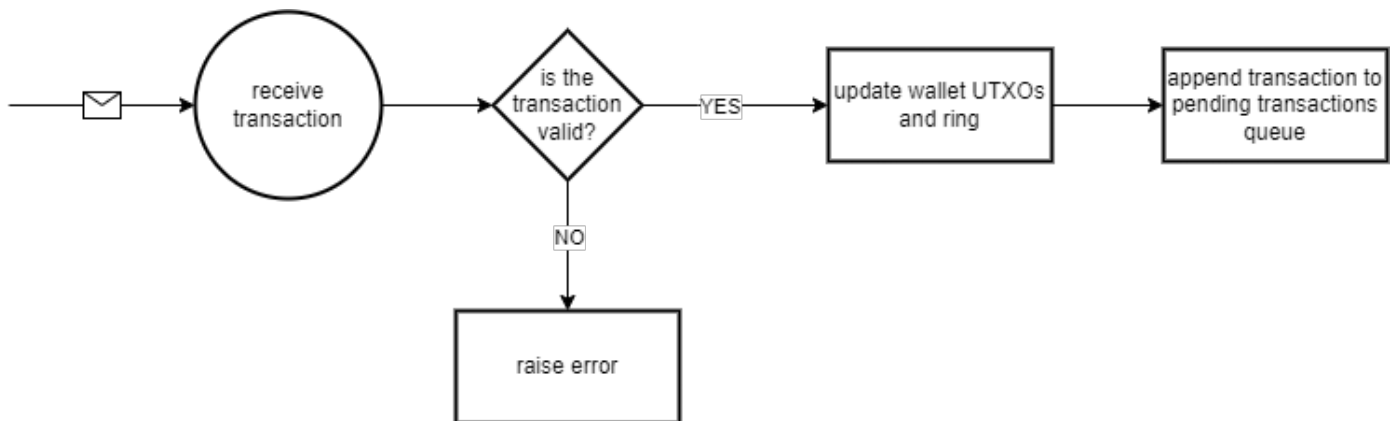
Διάγραμμα 2 Αρχικοποίηση κόμβου

Στην συνέχεια, στο διάγραμμα 3 φαίνεται αναλυτικά η διαδικασία που ακολουθείται από την δημιουργία ενός transaction έως και να γίνει broadcast στο υπόλοιπο δίκτυο.



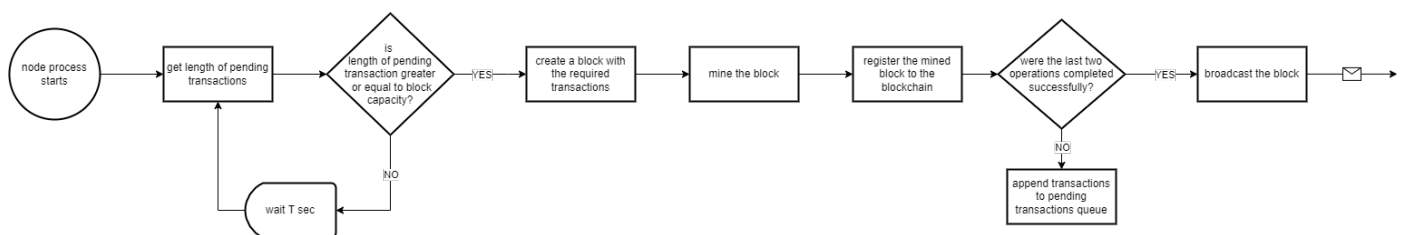
Διάγραμμα 3 Δημιουργία Transaction

Αυτή η λειτουργία της δημιουργίας και μετάδοσης του transaction ακολουθείται από την λειτουργία λήψης του transaction από τους κόμβους, η οποία φαίνεται αναλυτικά στο παρακάτω διάγραμμα 4.



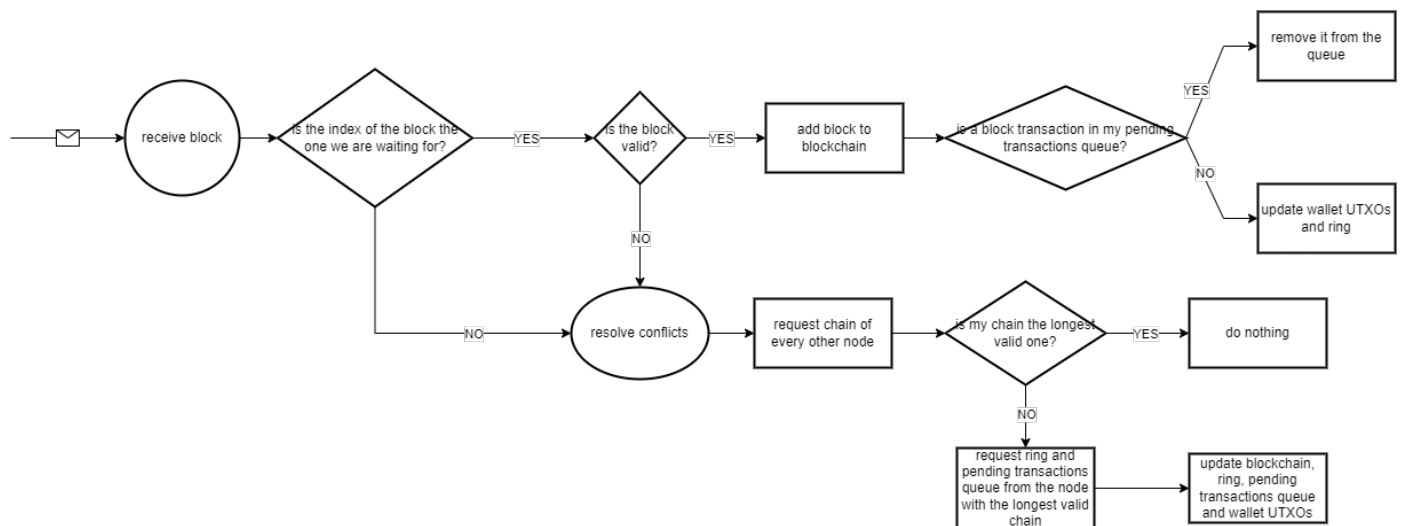
Διάγραμμα 4 Λήψη Transaction

Όταν συμπληρωθεί ο απαιτούμενος αριθμός από transactions σε ένα block (που καθορίζεται από την παράμετρο block capacity) τότε αρχίζει η διαδικασία δημιουργίας του block, η οποία «ολοκληρώνεται» με το broadcast του block στο υπόλοιπο δίκτυο. Η διαδικασία με την αλληλουχία των εντολών που εκτελούνται φαίνεται αναλυτικά στο παρακάτω διάγραμμα 5.



Διάγραμμα 5 Δημιουργία Block

Τέλος, η παραπάνω διαδικασία δημιουργίας και μετάδοσης του block ακολουθείται από την διαδικασία λήψης του block από τους κόμβους. Τα στάδια από τα οποία περνάει το block και οι έλεγχοι που γίνονται σε αυτό μαζί με τις απαιτούμενες ενέργειες σε κάθε περίπτωση που μπορεί να προκύψει από τους ελέγχους παρουσιάζονται αναλυτικά στο παρακάτω διάγραμμα 6, στο οποίο μάλιστα απεικονίζεται και η λειτουργία resolve conflicts, η οποία όπως αναφέρθηκε παραπάνω αποτελεί και εκείνη σημαντική λειτουργία για την ορθότητα του συστήματός μας.



Διάγραμμα 6 Λήψη Block

2.2.3 REST API και Σχεδιαστικό Μοτίβο Αντάπτορα

Για να λειτουργήσει η εφαρμογή πρέπει οι διεργασίες Noobcash Core να επικοινωνούν μεταξύ τους. Έτσι επιλέξαμε να υλοποιήσουμε ένα REST API που εκθέτει καταληκτικά σημεία (endpoints) που μπορούν να χρησιμοποιήσουν τόσο οι άλλες διεργασίες Noobcash Core όσο και τα Noobcash Client και Tester. Για να υλοποιήσουμε το API αυτό χρησιμοποιήσαμε το module flask που είναι ευρέως διαδεδομένο στην κοινότητα της Python, εφόσον είναι αρκετά «ελαφρύ», έχοντας λίγες εξαρτήσεις από άλλα modules, και είναι αναλυτικά τεκμηριωμένο. Στην συνέχεια, για να μετατρέψουμε τα δεδομένα που καταναλώνονται αλλά και διαμοιράζονται από και προς το REST API αντίστοιχα σε δεδομένα που μπορούν να διαχειριστούν οι κλάσεις που υλοποιήσαμε, χρησιμοποιήσαμε το σχεδιαστικό μοτίβο του αντάπτορα (adapter design pattern) (Adapter, n.d.). Όταν κάποιος καταναλώνει (χρησιμοποιεί) κάποιο endpoint, αυτό με χρήση των κλάσεων που βρίσκονται στον φάκελο usecases μετατρέπει τα δεδομένα που έχουν επισυναφθεί στο request σε δεδομένα που μπορούν να διαβαστούν από τις κλάσεις και ύστερα, όταν τελειώσει η επεξεργασία του αιτήματος, οι κλάσεις usecases μετατρέπουν και επισυνάπτουν τα δεδομένα, τα οποία θέλουμε να απαντήσουμε. Όλα τα endpoints που επιστρέφουν απαντήσεις στο Noobcash Client και στο Noobcash Tester επιστρέφουν JSON ή κενές απαντήσεις (204 http code) ενώ τα endpoints που αφορούν τις άλλες διεργασίες Noobcash Core επιστρέφουν κυρίως αντικείμενα κλάσης που είναι pickled (από το module της Python Pickle), δηλαδή τα αντικείμενα αυτά σειριοποιημένα.

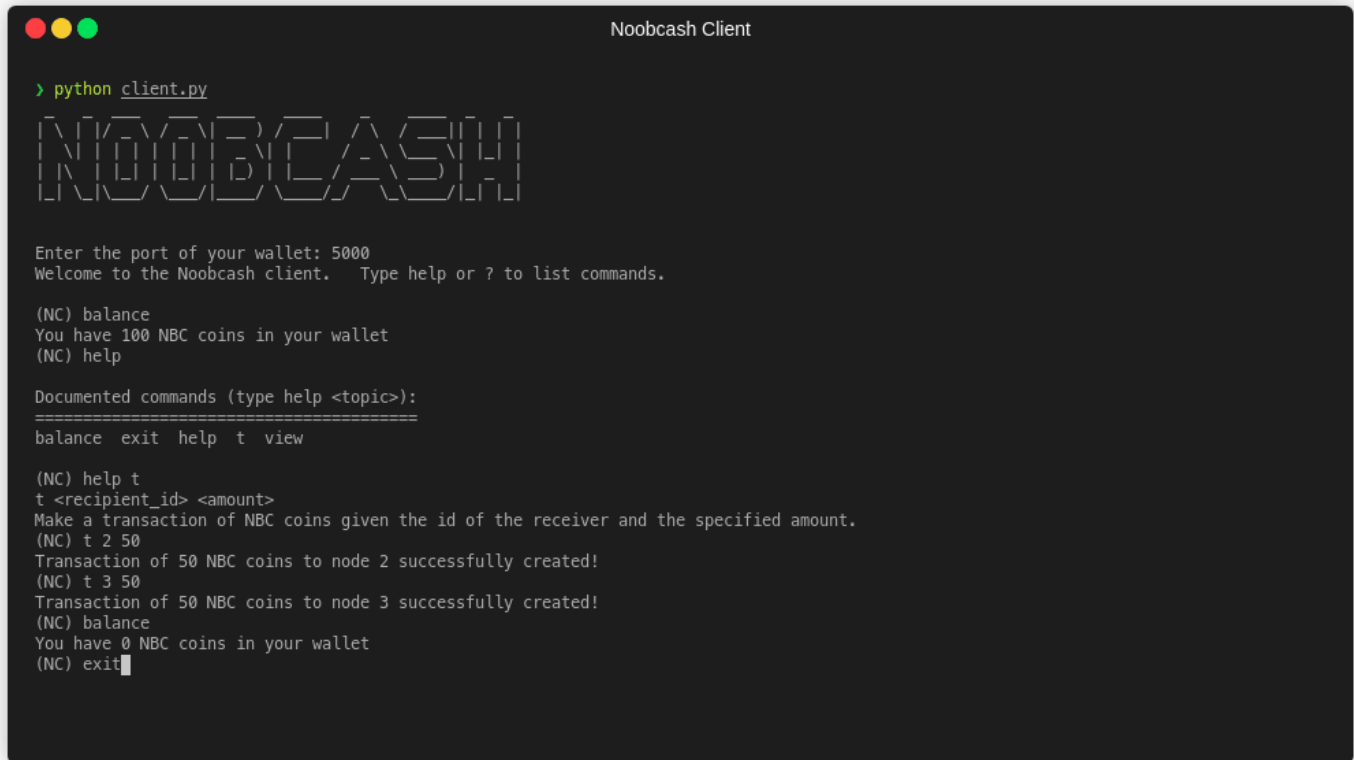
2.3 NOOBCASH CLIENT

Το σύστημα Noobcash Client αποτελεί ένα Command Line Interface (CLI) που επιτρέπει στον χρήστη να συνδεθεί στον κόμβο (Noobcash Core) που του αντιστοιχεί και να εκτελέσει τις εξής εντολές:

1. *t <recipient id> <amount>*
Δίνει στον χρήστη την δυνατότητα να δημιουργεί μια συναλλαγή δίνοντας το id του παραλήπτη και το ποσό από NBC που θέλει να μεταφέρει σε αυτόν.
2. *view*
Δίνει στον χρήστη την δυνατότητα να δει τις συναλλαγές που βρίσκονται στο τελευταίο επικυρωμένο block του Noobcash blockchain.
3. *balance*
Δίνει στον χρήστη την δυνατότητα να δει το υπόλοιπο που έχει στο wallet του.
4. *exit*
Δίνει στον χρήστη την δυνατότητα να εξέλθει από το Noobcash Client.
5. *help*
Δίνει στον χρήστη την δυνατότητα να λάβει επεξήγηση για την χρήση των παραπάνω εντολών.

Για να υλοποιήσουμε το CLI ως ένα read-eval-print loop (REPL) περιβάλλον χρησιμοποιήσαμε το module της python *cli*, ενώ για να κάνουμε χρήση του API που προσφέρει το Noobcash backend χρησιμοποιήσαμε το module *requests*. Και τα δύο αυτά modules αποτελούν built-in modules της Python.

Παρακάτω φαίνεται μια ενδεικτική χρήση του Noobcash Client



```
Noobcash Client

> python client.py

NOOBCASH

Enter the port of your wallet: 5000
Welcome to the Noobcash client. Type help or ? to list commands.

(NC) balance
You have 100 NBC coins in your wallet
(NC) help

Documented commands (type help <topic>):
=====
balance  exit  help  t  view

(NC) help t
t <recipient_id> <amount>
Make a transaction of NBC coins given the id of the receiver and the specified amount.
(NC) t 2 50
Transaction of 50 NBC coins to node 2 successfully created!
(NC) t 3 50
Transaction of 50 NBC coins to node 3 successfully created!
(NC) balance
You have 0 NBC coins in your wallet
(NC) exit
```

Εικόνα 1 Χρήση Noobcash Client

2.4 NOOBCASH TESTER

Το σύστημα Noobcash Tester μας επιτρέπει να δημιουργούμε ένα σύνολο από προκαθορισμένες συναλλαγές σε κάθε κόμβο (Noobcash Core) για 5 ή 10 κόμβους και να λαμβάνουμε στατιστικά για αυτούς. Συγκεκριμένα, κάναμε την σχεδιαστική επιλογή ο κάθε κόμβος να στέλνει την συναλλαγή του παράλληλα με άλλους και για προσομοιώσουμε ένα πιο ρεαλιστικό σύστημα η χρονική διάρκεια που δημιουργούνται οι συναλλαγές ακολουθεί μια ομοιόμορφη κατανομή μεταξύ των 0 και 10 δευτερολέπτων.

Ταυτόχρονα, με χρήση του εργαλείου Prometheus, που χρησιμοποιούμε στο Noobcash Core, λαμβάνουμε διάφορες μετρικές από τον κάθε κόμβο και προσαρμόζοντας αυτές υπολογίζουμε την ρυθμαπόδοση (throughput) του συστήματος, δηλαδή πόσα transactions εξυπηρετούνται στην μονάδα του χρόνου, και το block time, δηλαδή τον μέσο χρόνο που απαιτείται για να γίνει mine ένα block (από διευκρίνιση που δόθηκε σε ερώτηση στο forum του μαθήματος στο HELIOS).

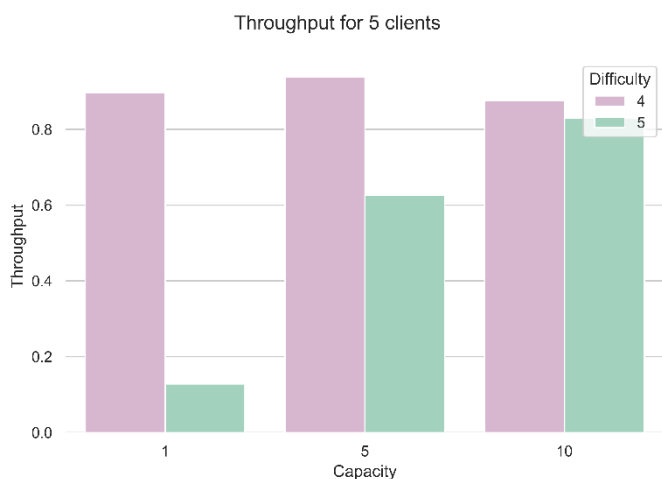
3 ΑΞΙΟΛΟΓΗΣΗ ΣΥΣΤΗΜΑΤΟΣ

3.1 ΠΕΡΙΒΑΛΛΟΝ ΠΕΙΡΑΜΑΤΩΝ

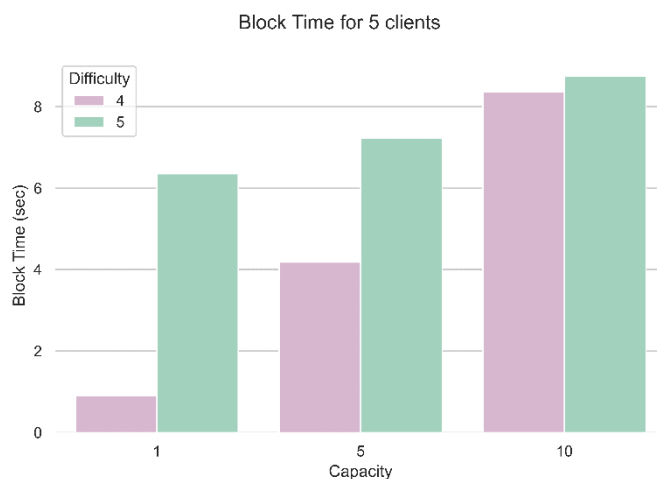
Για την διεξαγωγή των πειραμάτων χρησιμοποιήσαμε πόρους από την υπηρεσία [~okeanos-knossos](#) του GRNET. Συγκεκριμένα, για να προσομοιώσουμε ένα σύστημα blockchain όσο πιο κοντά στην πραγματικότητα γίνεται, μας δόθηκαν 5 virtual machines (καθένα εκ των οποίων έχει τα εξής χαρακτηριστικά: 2 cores, 2GB μνήμης και 30GB

δίσκου), μια public IPV4 IP και ένα private network. Η επικοινωνία μεταξύ των κόμβων γινόταν μέσω του private IPV4 network και του πρωτοκόλλου http. Στα πειράματα που υλοποιήσαμε μετρήσαμε την απόδοση του συστήματος με την χρήση δύο μετρικών. Αυτές είναι το block time, δηλαδή ο μέσος χρόνος που απαιτείται για να γίνει mine ένα block (σύμφωνα με διευκρίνιση που δόθηκε σε ερώτηση στο forum του μαθήματος στο HELIOS) και το throughput (ρυθμαπόδοση), δηλαδή το πλήθος των transactions που εξυπηρετούνται στην μονάδα του χρόνου, το οποίο το υπολογίσαμε ως τον λόγο του αθροίσματος όλων των validated transactions προς τον συνολικό χρόνο από τη στιγμή που εστάλη το πρώτο transaction (από τα transactions που μας δίνονται στα txt αρχεία, δηλαδή αφότου «αρχικοποιηθεί» το σύστημα) έως ότου να γίνει mine και το τελευταίο block.

3.2 ΑΠΟΔΟΣΗ



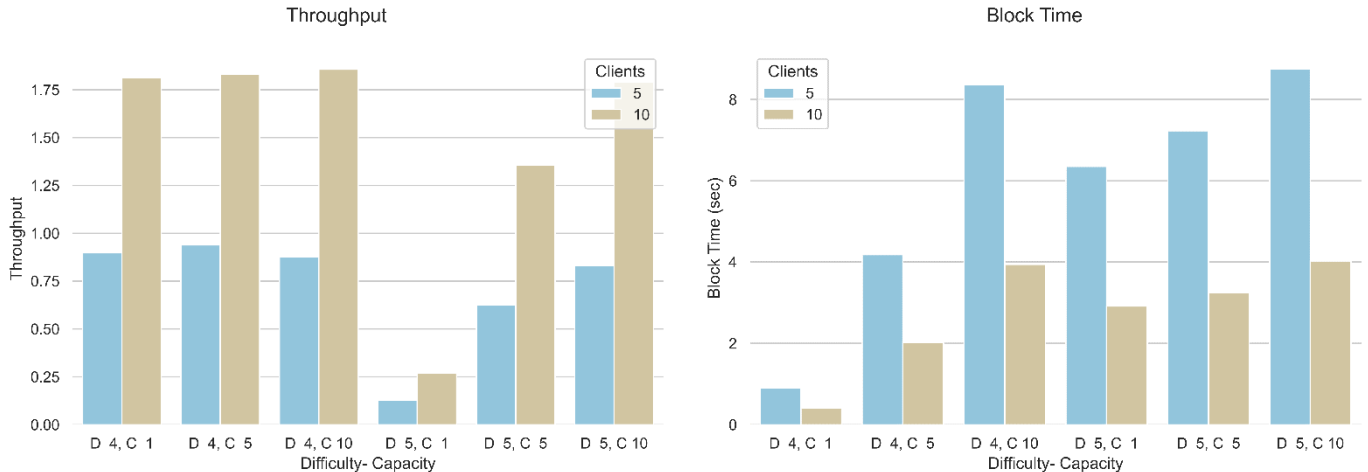
Διάγραμμα 7 Throughput για 5 clients



Διάγραμμα 8 Block Time για 5 Clients

Στα αρχικά πειράματα προσπαθήσαμε να προσομοιώσουμε ένα καταναμημένο σύστημα με 5 κόμβους και έτσι είχαμε έναν client για κάθε virtual machine, όπου ένα από αυτά μοιραζόταν την διεργασία του και με την διεργασία Noobcash Tester. Έτσι, κάθε κόμβος είχε την δική του private IPV4 address και ένα συγκεκριμένο port για το peer-to-peer network. Τα πειράματα διεξήχθησαν με διαφορετικούς συνδυασμούς των παραμέτρων block capacity και mining difficulty. Πιο συγκεκριμένα, το block capacity αναφέρεται στο μέγιστο πλήθος από transactions που περιέχει το κάθε block, και κατά την διάρκεια των πειραμάτων πήρε τις τιμές 1, 5 και 10, ενώ το mining difficulty αναφέρεται στο πλήθος των μηδενικών που πρέπει να έχει στην αρχή του το hash ενός block για να θεωρηθεί το block mined και πήρε τις τιμές 4 και 5. Άρα, συνολικά διεξήχθησαν 6 πειράματα, τα αποτελέσματα των οποίων φαίνονται στα διαγράμματα 7 και 8 παραπάνω.

3.3 ΚΛΙΜΑΚΩΣΙΜΟΤΗΤΑ



Διάγραμμα 9 Throughput για 5 και 10 Clients

Διάγραμμα 10 Block Time για 5 και 10 clients

Στο επόμενο βήμα των πειραμάτων που χρειάστηκε να προσομοιώσουμε σύστημα με 10 χρήστες, ανοίξαμε δύο συνδέσεις στο κάθε virtual machine, προσομοιώνοντας δύο κόμβους στο κάθε ένα. Σε αυτή την μορφή ανά δύο οι κόμβοι του συστήματος είχαν την ίδια private IPV4 address αλλά διαφορετικά ports. Στο σύστημα ορίσαμε πως κάθε client μπορεί να είναι και miner, συμπεριλαμβανομένου και του bootstrap κόμβου. Έτσι, έχοντας «στήσει» κατάλληλα τις υποδομές για το σύστημα με τους 10 κόμβους, τρέξαμε τα πειράματα με τους ίδιους συνδυασμούς παραμέτρων που είχαμε χρησιμοποιήσει και για τα πειράματα με τους 5 κόμβους. Τα αποτελέσματα των πειραμάτων με τους 10 κόμβους φαίνονται στα διαγράμματα 9 και 10, συγκριτικά με τα αντίστοιχα αποτελέσματα των πειραμάτων με τους 5 κόμβους, έτσι ώστε να είναι ευκολότερα αντιληπτή η διαφορά των αποτελεσμάτων των πειραμάτων αυτών.

3.4 ΣΧΟΛΙΑΣΜΟΣ ΠΕΙΡΑΜΑΤΩΝ

Όσον αφορά στο throughput του συστήματος, εκτός κάποιων outliers, παρατηρούμε πως για σταθερό difficulty έχουμε καλύτερες τιμές όσο αυξάνεται το capacity του block και για τα δύο πειράματα (5 και 10 nodes). Αυτό είναι λογικό, γιατί για τον ίδιο αριθμό transactions χρειάζεται να γίνουν mine λιγότερα blocks, πράγμα που συμβάλλει στην αποσυμφόρησή του συστήματος και στην αύξηση της απόδοσης. Ακόμη, παρατηρούμε πως η ρυθμαπόδοση του συστήματος για mining difficulty 5 είναι μικρότερη σε σχέση με αυτή για mining difficulty 4. Αυτό είναι αναμενόμενο, καθώς ο απαιτούμενος χρόνος για mining αυξάνεται εκθετικά σε σχέση με τη δυσκολία. Επίσης, με την αύξηση των κόμβων του συστήματος σε 10 παρατηρούμε και σημαντική αύξηση του throughput. Αυτό, επίσης, είναι αναμενόμενο αφού έχουν αυξηθεί οι miners και άρα οι απαιτούμενοι χρόνοι για να καταχωρηθεί ένα block στο σύστημα έχουν μειωθεί, συμβάλλοντας και αυτό αντίστοιχα στην αύξηση της απόδοσης και του throughput. Επίσης, μην ξεχνάμε πως με την είσοδο ακόμα 5 κόμβων στο σύστημα, διπλασιάζεται και ο αριθμός των συναλλαγών που διεξάγονται σε αυτό, κάτι το οποίο επίσης μπορεί να δικαιολογήσει την αύξηση της ρυθμαπόδοσης με την αύξηση των κόμβων στο σύστημα (κάθε αρχείο txt, δηλαδή κάθε κόμβος, περιέχει 100 συναλλαγές).

Όσον αφορά τώρα στο block time του συστήματος, η σημαντικότερη μεταβολή που βλέπουμε στο σύστημα και για τα δύο πειράματα (5 και 10 nodes) είναι στην αλλαγή του difficulty. Προφανώς, όσο αυξάνεται το difficulty τόσο πιο δύσκολος είναι ο υπολογισμός του hash με τόσα μηδενικά στην αρχή όσα η παράμετρος mining difficulty και άρα αυξάνεται και ο μέσος χρόνος του mine, πράγμα που φαίνεται και από τα πειράματα. Επίσης, θα περιμέναμε με σταθερό difficulty και μεγαλύτερες τιμές capacity να αυξάνεται και το block time, καθώς το hash μεγαλώνει σαν string, πράγμα που βλέπουμε και πειραματικά στα παραπάνω αποτελέσματα. Αυτό που είναι αξιοσημείωτο είναι το γεγονός πως το block time για 10 κόμβους όπως φαίνεται στα παραπάνω διαγράμματα είναι περίπου το μισό κάθε φορά σε σχέση με αυτό του αντίστοιχου πειράματος για 5 κόμβους. Αυτό είναι αναμενόμενο εφόσον όπως εξηγήσαμε παραπάνω πλέον υπάρχουν 10 κόμβοι στο σύστημα, και άρα 10 miners. Επίσης, η προσθήκη 5

παραπάνω κόμβων στο σύστημα επέφερε και τον διπλασιασμό του πλήθους των συναλλαγών, και άρα την μείωση του χρόνου αναμονής συμπλήρωσης του επόμενου block, κάτι το οποίο σε συνδυασμό με τα παραπάνω είχε ως αποτέλεσμα το mining να γίνει περίπου στον μισό χρόνο.

Από τα παραπάνω, μπορούμε να καταλήξουμε στο ότι το σύστημα μας είναι κλιμακώσιμο, αφού λειτουργεί αποδοτικά και σε αριθμό κόμβων μεγαλύτερο του 5. Μάλιστα, με την είσοδο περισσότερων κόμβων-miners, το σύστημά μας αποδεδειγμένα λειτουργεί ταχύτερα και αποδοτικότερα. Ωστόσο, πρέπει να λάβουμε υπόψη ότι ο αριθμός και το μέγεθος των μηνυμάτων είναι μεγάλο και είναι πολύ πιθανό να προκύψουν bottlenecks. Για αυτό εάν μπορούσαμε να αφιερώσουμε παραπάνω χρόνο στην εφαρμογή θα μειώναμε το μέγεθος των μηνυμάτων (όπως για παράδειγμα να μην στέλνουμε όλη την αλυσίδα όταν γίνεται κάποιο conflict) και θα στέλναμε τα μηνύματα που είναι ασύγχρονα (τα περισσότερα broadcast) με τη χρήση ενός μοντέλου producer (ο δημιουργός των μηνυμάτων ή requests) - consumer (στέλνει requests) με χρήση ουρών.

4 ΑΝΑΦΟΡΕΣ

Adapter. (χ.χ.). Ανάκτηση από Refactoring.Guru: <https://refactoring.guru/design-patterns/adapter>

bitcoin/bitcoin: Bitcoin Core integration/staging tree. (χ.χ.). Ανάκτηση από GitHub :
<https://github.com/bitcoin/bitcoin/>

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*.