Εξαμηνιαία εργασία στα Κατανεμημένα Συστήματα

Καλλίτσης Γεώργιος AM: 03117051
Μυροπούλου Νεφέλη AM: 03117197
Ξύγκου Ηλιάνα-Μαρία AM: 03117059

Εισαγωγή – Σκοπός

Η παρούσα εργασία αποσκοπεί στην υλοποίηση του συστήματος "Noobcash", ενός απλού συστήματος blockchain, όπου καταγράφονται οι δοσοληψίες μεταξύ των συμμετεχόντων και εξασφαλίζεται consensus με χρήση Proof-of-Work.
Οι βασικές λειτουργίες της εφαρμογής, όπως αυτές περιγράφονται στην εκφώνηση, υλοποιούνται από το backend της εφαρμογής, ενώ στη συνέχεια υλοποιείται αντίστοιχος client, από όπου είναι δυνατή η εκτέλεση των ορισμένων εντολών από τον χρήστη. Τέλος, το αναπτυχθέν σύστημα αξιολογείται μέσω διεξαγωγής πειραμάτων, που πραγματοποιούνται για δύο τιμές πλήθους κόμβων.

Noobcash backend

Για αυτό το τμήμα της εργασίας χρησιμοποιήθηκε η γλώσσα Python 3.9.5, καθώς και το Python web framework Flask 2.0.3. Λοιπές βιβλιοθήκες ή πακέτα που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής αναφέρονται στο υποβαλλόμενο αρχείο 'requirements.txt', το οποίο μπορεί να χρησιμοποιηθεί και για πιο εύκολη εγκατάστασή της. Σημειώνουμε ότι για λόγους ευκολότερης ανάγνωσης, η ανάλυση του backend στην συνέχεια πραγματοποιείται ανά κλάση.

Κλάση `Transaction`:
Η κλάση αυτή υλοποιεί μια δοσοληψία στο Noobcash blockchain δίκτυο. Διαθέτει τα εξής πεδία:

πεδίο	τύπος μεταβλητής	περιγραφή
`sender_address`	`bytes`	public key του wallet από όπου προέρχονται τα χρήματα
`receiver_address`	`bytes`	public key του wallet όπου θα καταλήξουν τα χρήματα
`amount`	`int`	ποσό NBCs που μεταφέρεται
`timestamp`	`float`	timestamp δημιουργίας συναλλαγής
`transaction_id`	`Crypto.Hash.SHA256.SHA256Hash`	μοναδικό αναγνωριστικό συναλλαγής
`transaction_inputs`	`list`	δείχνει από ποιο/α προηγούμενο/α transaction(s) προήλθαν τα χρήματα που μεταφέρονται
`transaction_outputs`	`list`	αποτελείται από 2 ειδών outputs: ένα output που αντιστοιχεί στο `amount` της δοσοληψίας για τον παραλήπτη, και ένα output για τον αποστολέα σε περίπτωση που προκύψει υπόλοιπο από τη συναλλαγή

signature	str	υπογραφή που παράγεται με το private key του αποστολέα για την επαλήθευση της γνησιότητας της δοσοληψίας

Μέθοδοι που συμπεριλαμβάνονται στην κλάση:

- create_transaction_outputs: υπολογισμός του υπολοίπου της συναλλαγής με βάση τα πεδία transaction_inputs και amount και σε περίπτωση που αυτό είναι θετικό, συμπερίληψη του υπολοίπου στα transaction_outputs ως output με δέκτη τον αποστολέα. Διαφορετικά, το πεδίο αυτό περιλαμβάνει μόνο ως output αυτό που αντιστοιχεί στο amount της δοσοληψίας για τον παραλήπτη.
- sign_transaction: υπογραφή της δοσοληψίας με βάση το private key του αποστολέα. Η υπογραφή γίνεται με χρήση των συναρτήσεων new (δημιουργία του signature object) και sign του Crypto.Signature.pkcs1_15 module.
- __eq__: η μέθοδος αυτή καλείται όταν χρησιμοποιείται ο τελεστής '==' για την σύγκριση δύο δοσοληψιών. Εάν όλα τα πεδία των αντικειμένων διαθέτουν τις ίδιες τιμές επιστρέφεται True, διαφορετικά επιστρέφεται False.
- to_dict: δημιουργία OrderedDict αντικειμένου, με βάση το τρέχον Transaction αντικείμενο.
- __str__: αυτή η μέθοδος καλείται όταν χρησιμοποιείται η str() συνάρτηση σε αντικείμενα τύπου Transaction. Γίνεται μετατροπή του OrderedDict αντικειμένου που προκύπτει με χρήση της to_dict μεθόδου, σε string.
- __repr__: επιστροφή μιας printable εκδοχής του αντικειμένου. Καλείται εάν χρησιμοποιηθεί η repr() συνάρτηση σε αντικείμενα τύπου Transaction.

Κλάση Wallet:
Η κλάση αυτή υλοποιεί το πορτοφόλι ενός κόμβου του δικτύου. Τα πεδία της είναι αυτά που παρουσιάζονται παρακάτω:

Πεδίο	τύπος μεταβλητής	περιγραφή
public_key	bytes	διεύθυνση που ο κόμβος μπορεί να μοιραστεί με οποιοδήποτε άλλον κόμβο για την διεκπεραίωση συναλλαγών μεταξύ τους
private_key	bytes	προσωπικό μυστικό κλειδί για την υπογραφή και την επαλήθευση γνησιότητας συναλλαγών του κόμβου
balance	int	τρέχον υπόλοιπο του wallet, αρχικοποίηση με 0

Παρακάτω παρουσιάζονται οι μέθοδοι της κλάσης Wallet:

- generateKeyPair: αρχικοποίηση των πεδίων public_key και private_key του αντικειμένου. Για την παραγωγή του private_key γίνεται χρήση του αλγορίθμου RSA και της μεθόδου generate() του αντίστοιχου RSA module. Με χρήση της μεθόδου public_key() γίνεται η παραγωγή του

αντίστοιχου public key, ενώ με χρήση της `export_key()` γίνεται εξαγωγή και των δύο κλειδιών σε μορφή pem.

- `wallet_balance`: επιστροφή του πεδίου `balance` του αντικειμένου.
- `to_dict`: δημιουργία `OrderedDict` αντικειμένου, με βάση το τρέχον `Wallet` αντικείμενο.

Κλάση `Block`:
Η κλάση αυτή αντιστοιχεί εννοιολογικά σε ένα block του blockchain. Περιλαμβάνει ως πεδία τα:

πεδίο	τύπος μεταβλητής	περιγραφή
`index`	`int`	ο αύξων αριθμός του block στο blockchain
`timestamp`	`float`	το timestamp δημιουργίας του block, αρχικοποίηση με την μέθοδο `time()` του time module
`nonce`	`bytes`	η λύση του proof-of-work
`hash`	`str`	το hash του block
`previousHash`	`str`	το hash του προηγούμενου block στο blockchain
`listOfTransactions`	`list`	λίστα με τα transactions που συμπεριλαμβάνονται στο block

Οι μέθοδοι της συγκεκριμένης κλάσης είναι αντίστοιχα οι:

- `myHash`: υπολογισμός του hash πεδίου με χρήση των συναρτήσεων `sha256()` και `hexdigest()` του python module hashlib. Η συνάρτηση `sha256()`, επιστρέφει την hash τιμή μήκους 256 bits μιας δεδομένης συμβολοσειράς s, ενώ η `hexdigest()` μετατρέπει την παραπάνω hash τιμή σε hex μορφή. Η hash τιμή του block υπολογίζεται με βάση τα υπόλοιπα πέντε πεδία της κλάσης, όπως αυτά παρουσιάζονται στον παραπάνω πίνακα.
- `add_transaction`: προσθήκη μιας δεδομένης συναλλαγής (αντικείμενο τύπου `Transaction`), στην λίστα συναλλαγών του block, `listOfTransactions`.
- `__eq__`: η μέθοδος αυτή καλείται όταν χρησιμοποιείται ο τελεστής '==' για την σύγκριση δύο αντικειμένων της κλάσης. Εάν όλα τα πεδία των αντικειμένων διαθέτουν τις ίδιες τιμές επιστρέφεται `True`, διαφορετικά επιστρέφεται `False`.
- `to_dict`: δημιουργία `OrderedDict` αντικειμένου, με βάση το τρέχον `Block` αντικείμενο.

Κλάση `Blockchain`:
Η κλάση `Blockchain` αντιπροσωπεύει το Blockchain του Noobcash δικτύου.

Πεδία κλάσης:

πεδίο	τύπος μεταβλητής	περιγραφή
`chain`	`list`	λίστα από blocks που αντιπροσωπεύει το blockchain

Οι μέθοδοι της κλάσης παρουσιάζονται στην συνέχεια:

- `build_genesis`: δημιουργία του πρώτου (genesis) block του Blockchain. Το genesis block δημιουργείται από τον πρώτο κόμβο του δικτύου (bootstrap node) και περιλαμβάνει λίστα από transactions που περιέχει μόνο ένα transaction που δίνει στον bootstrap κόμβο 100*n NBC coins (όπου n το μέγεθος του δικτύου).
- `__getitem__`: δεδομένης μιας θέσης, επιστροφή του block της αλυσίδας στην θέση αυτή.
- `__len__`: επιστροφή του μήκους της αλυσίδας `chain`.
- `add_block`: προσθήκη ενός νέου block στην αλυσίδα `chain`.
- `__eq__`: χρήση για την σύγκριση δύο αντικειμένων `Blockchain`. Επιστρέφεται `True` μόνο στην περίπτωση που τα πεδία `chain` των δύο αντικειμένων ταυτίζονται.

Κλάση `Node`:
Η κλάση `Node` αποτελεί την κυριότερη κλάση της εφαρμογής. Αναπαριστά έναν κόμβο του δικτύου και μέσω των μεθόδων της υλοποιεί την πλειοψηφία των λειτουργιών της εφαρμογής.

Πεδία κλάσης:

πεδίο	**τύπος μεταβλητής**	**περιγραφή**
`id`	`int`	μοναδικό αναγνωριστικό του κόμβου το οποίο ανατίθεται σε αυτόν από τον bootstrap κόμβο
`ip`	`str`	η IPv4 διεύθυνση του κόμβου
`port`	`int`	το port στο οποίο ακούει η Noobcash εφαρμογή
`NBCs`	`dict[Bytes,list[dict[]]]`	dictionary που περιέχει όλα τα unspent transaction outputs (UTXOs) για κάθε κόμβο του δικτύου key: public key κόμβου value: λίστα από UTXOs
`wallet`	`Wallet`	το wallet του κόμβου
`ring`	`list[(str, int, bytes)]`	λίστα από `(ip, port, public_key)` tuples. Η θέση στην λίστα αντιστοιχεί στο id του κόμβου για τον οποίο δίνονται οι πληροφορίες
`transactions`	`list[Transaction]`	λίστα που περιέχει όλα τα transactions που ο κόμβος έχει συλλέξει αλλά δεν έχει προσθέσει ακόμα στο blockchain
`block`	`Block`	τρέχον block στο οποίο θα προστεθούν τα συλλεγμένα `transactions`

`chain`	`Blockchain`	το blockchain του Noobcash δικτύου
`mining_flag`	`bool`	boolean μεταβλητή που φανερώνει εάν ο κόμβος βρίσκεται σε διαδικασία mining
`lock`	`threading.Lock`	lock που εξασφαλίζει την απομόνωση παράλληλων διεργασιών που επεξεργάζονται τα ίδια αντικείμενα
`mining_lock`	`threading.Lock`	lock που εξασφαλίζει την απομόνωση της διαδικασίας του mining ενός κόμβου

Μέθοδοι κλάσης `Node`:

- `create_wallet`: δημιουργία του `wallet` του κόμβου
- `register_node_to_ring`: με βάση το `public key`, την `ip` διεύθυνση και το `port` ενός νέου κόμβου, γίνεται προσθήκη των στοιχείων του κόμβου στο `ring`, ενώ στη συνέχεια του επιστρέφονται το `id` που του αντιστοιχεί και η τρέχουσα αλυσίδα `Blockchain`.
- `update`: ανανέωση των πεδίων `id` και `chain` του κόμβου, με βάση τις πληροφορίες που παρέχονται σε αυτόν από τον bootstrap κόμβο.
- `set_ring`: ανανέωση του `ring` του κόμβου, με βάση το ring που του παρέχει ο bootstrap κόμβος.
- `broadcast_ring`: μετάδοση του `ring` του δικτύου και μεταφορά 100 coins σε έκαστο εκ των υπολοίπων κόμβων (εκτελείται μόνο από τον bootstrap node).
- `broadcast_transaction`: μετάδοση μιας δοσοληψίας σε όλους τους υπόλοιπους κόμβους του δικτύου .
- `broadcast_block`: αποστολή ενός mined block σε όλους τους υπόλοιπους κόμβους του δικτύου.
- `create_transaction`: δημιουργία μιας νέας δοσοληψίας με `sender` τον τρέχοντα κόμβο και τιμές `amount` και `receiver` αυτά που προσδιορίζονται, εφόσον τα NBCs του επαρκούν.
- `validate_transaction`: έλεγχος εγκυρότητας μιας δοθείσας συναλλαγής με βάση την υπογραφή, το `id` και τα `transaction_inputs` και `transaction_outputs` πεδία της.
- `update_NBCs`: ανανέωση των NBCs του αποστολέα και του παραλήπτη με βάση μια δοθείσα συναλλαγή. Σε περίπτωση που ο τρέχων κόμβος είναι κάποιος από τους δύο, γίνεται ανανέωση του υπολοίπου του `wallet` του.
- `add_transaction_to_block`: προσθήκη μιας έγκυρης δοσοληψίας στο τρέχον `block` του κόμβου και σε περίπτωση που αυτό είναι πλήρες, έναρξη της διαδικασίας mining.
- `add_transactions_to_block`: προσθήκη των `transactions` που εκκρεμούν στο τρέχον `block` του κόμβου .

- `create_new_block`: προσθήκη ενός νέου εισερχόμενου block στην αλυσίδα του κόμβου (εφόσον το block είναι έγκυρο). Γίνεται ακόμα ανανέωση του `NBCs` πεδίου του κόμβου με βάση τις δοσοληψίες που περιέχονται στο νέο block, καθώς και αφαίρεση των δοσοληψιών αυτών από τις εκκρεμείς δοσοληψίες του κόμβου (ανανέωση πεδίου `transactions`). Ακόμα δημιουργείται ένα νέο τρέχον block για τον κόμβο.
- `validate_block`: έλεγχος εγκυρότητας ενός block με βάση τη `hash` τιμή του, την επιλεγμένη τιμή mining difficulty, την τιμή `hash` του προηγούμενου block στην αλυσίδα, και τις δοσοληψίες που αυτό περιλαμβάνει.
- `mine_block`: mining ενός block. Σε περίπτωση που αυτό πετύχει, γίνεται μετάδοση του mined block σε όλους τους υπόλοιπους κόμβους, προσθήκη του στο blockchain και αφαίρεση των συμπεριλαμβανόμενων στο block δοσοληψιών από τις εκκρεμείς δοσοληψίες του κόμβου.
- `proof_of_work`: υπολογισμός του `nonce` ενός block. Γίνονται δοκιμές τυχαίων τιμών `nonce` ωσότου επιτευχθεί το ζητούμενο mining difficulty ή κάποιος άλλος κόμβος ολοκληρώσει πρώτος την διαδικασία mining.
- `validate_chain`: έλεγχος της εγκυρότητας μιας δοθείσας αλυσίδας και ανανέωση του πεδίου `NBCs` του κόμβου, με βάση τις συναλλαγές που αυτή περιλαμβάνει. Εκτελείται όταν ο κόμβος εισέρχεται στο δίκτυο για πρώτη φορά.
- `resolve_conflicts`: εύρεση της αλυσίδας μέγιστου μήκους του δικτύου και αντικατάσταση της αλυσίδας του κόμβου, εάν η αλυσίδα του είναι μικρότερου μήκους. Σε περίπτωση αντικατάστασης γίνεται ανανέωση των πεδίων `NBCs` και `transactions` του κόμβου.
- `recalculate_NBCs`: βοηθητική συνάρτηση για επανυπολογισμό του `NBCs` πεδίου του κόμβου έπειτα από αντικατάσταση της αλυσίδας του.

Τέλος, παρακάτω παρουσιάζονται οι διάφορες διαδρομές που χρησιμοποιούνται για την επικοινωνία και το συγχρονισμό των κόμβων του δικτύου. Οι διαδρομές αυτές ορίζονται και υλοποιούνται για κάθε κόμβο μέσω της κλάσης `app.py`.

Διαδρομή	τύπος request	περιγραφή
`/transactions/get`	`GET`	επιστροφή όλων των εκκρεμών `transactions` του κόμβου
`/setRing/`	`POST`	ανανέωση του `ring` του κόμβου με βάση αυτό που του παρέχεται από τον bootstrap node (εκτελείται σε όλους τους κόμβους εκτός από τον bootstrap)
`/registerNode/`	`POST`	προσθήκη ενός νέου κόμβου στο `ring` και επιστροφή του αποδιδόμενου `id` και του `chain` (εκτελείται μόνο στον bootstrap)
`/createTransaction/`	`POST`	δημιουργία μιας νέας συναλλαγής

/addTransaction/	POST	αποδοχή μιας δοσοληψίας που εκτελέστηκε από έναν άλλον κόμβο του δικτύου
/addBlock/	POST	προσθήκη ενός νέου mined block στην αλυσίδα του κόμβου
/chainLength/	GET	επιστροφή του μήκους της αλυσίδας του κόμβου
/getChain/	GET	επιστροφή της αλυσίδας του κόμβου
/balance/	GET	επιστροφή του υπολοίπου (balance) του wallet του κόμβου
/viewLast/	GET	επιστροφή των δοσοληψιών που περιέχονται στο τελευταίο block της αλυσίδας

Περιγραφή της λειτουργίας του συστήματος

Για την καλύτερη κατανόηση του τρόπου λειτουργίας του συστήματος προχωρούμε σε περαιτέρω ανάλυση της αλληλεπίδρασης μεταξύ των παραπάνω κλάσεων, καθώς και της γενικής ροής που ακολουθεί η εκτέλεση της εφαρμογής. Για την πραγματοποίηση των ζητούμενων συναλλαγών είναι αρχικά απαραίτητη η προσθήκη όλων των κόμβων του δικτύου, το πλήθος των οποίων καθορίζεται από το αντίστοιχο μέγεθος N. Η ενεργοποίηση της εφαρμογής σε κάθε κόμβο πραγματοποιείται μέσω της εντολής: `python3 app.py [--port port] [--id 0] [--test]`, εκτελεσμένης στο κατάλληλο directory. Οι παράμετροι `port,id` και `test` είναι προαιρετικές (και για αυτό αναφέρονται εντός αγκυλών). Σημειώνουμε ότι η default τιμή του `port` είναι 5000 και για παράμετρο `id (default = None)` ίση με 0 υποδηλώνεται ότι ο προστιθέμενος κόμβος θα λειτουργήσει ως ο bootstrap, ενώ τέλος η παράμετρος `test (default = False)` χρησιμοποιείται για την εκτέλεση των απαιτούμενων πειραμάτων. Το πρώτο block της αλυσίδας (genesis block) κατασκευάζεται αποκλειστικά από τον bootstrap κόμβο και περιλαμβάνει στη λίστα δοσοληψιών του ως μοναδική συναλλαγή αυτή που προσφέρει στον bootstrap node 100*N NBC coins. Ακόμη, κάθε νέος κόμβος, κατά την προσθήκη του, «παρουσιάζει» τον εαυτό του στον bootstrap, ο οποίος στη συνέχεια προσθέτει το νέο κόμβο στο `ring` και του επιστρέφει το αποδιδόμενο `id` και το τρέχον `blockchain`. Αφού έχει ολοκληρωθεί η διαδικασία πλήρωσης του δικτύου, ο πρώτος κόμβος (bootstrap) μεταδίδει σε όλους τους υπόλοιπους το τελικό `ring`, καθώς και μεταφέρει 100 coins σε έκαστο για τις μετέπειτα συναλλαγές τους. Στη συνέχεια, κάθε κόμβος είναι έτοιμος να πραγματοποιήσει τις συναλλαγές που περιλαμβάνονται στο .txt αρχείο που του αναλογεί. Για κάθε γραμμή του αρχείου δημιουργείται μια νέα συναλλαγή (εάν είναι επαρκές το υπόλοιπο του wallet του κόμβου), η οποία γίνεται broadcast σε όλους τους υπόλοιπους κόμβους, έτσι ώστε να την προσθέσουν στις δικές τους εκκρεμείς συναλλαγές (`transactions`) και στα δικά τους blocks, ενώ προστίθεται και στις εκκρεμείς συναλλαγές (`transactions`) και στο τρέχον block του κόμβου (αν αυτό δεν είναι πλήρες). Εάν το τρέχον block ενός κόμβου γεμίσει, τότε αυτός εισέρχεται σε κατάσταση mining. Το mining ενός block ολοκληρώνεται από τον έγκαιρο υπολογισμό του `nonce` του block ή διακόπτεται στην περίπτωση που κάποιος άλλος κόμβος ολοκληρώσει πρώτος το mining. Ένα επιτυχώς mined block προστίθεται στην αλυσίδα

του κόμβου και στη συνέχεια μεταδίδεται σε όλο το υπόλοιπο δίκτυο, έτσι ώστε και οι υπόλοιποι κόμβοι να ανανεώσουν το `chain` τους, έπειτα από επιβεβαίωση της εγκυρότητας του εισερχόμενου block. Επισημαίνουμε ότι έπειτα από την επιβεβαίωση κάθε νέας δοσοληψίας ή block από έναν κόμβο γίνεται κατάλληλη ανανέωση των αντίστοιχων `NBCs` και `transactions` πεδίων του. Επίσης, στην περίπτωση που προκύψει κάποια διακλάδωση στο blockchain γίνεται επίλυση συγκρούσεων, μέσω εύρεσης της αλυσίδας του δικτύου με το μέγιστο μήκος και αντικατάσταση του chain του κόμβου με αυτή (αν φυσικά έχει μεγαλύτερο μήκος από τη δική του τρέχουσα αλυσίδα). Τέλος, αναφέρουμε ότι γίνεται χρήση δύο locks, με το πρώτο να εξασφαλίζει την απομόνωση παράλληλων διεργασιών που επεξεργάζονται τα ίδια αντικείμενα και το δεύτερο να εξασφαλίζει την απομόνωση της διαδικασίας του mining ενός κόμβου. Βέβαια, αξίζει να σημειωθεί ότι όσο ο κόμβος βρίσκεται σε διαδικασία mining, δέχεται νέες συναλλαγές (δηλαδή δεν «μπλοκάρει») με χρήση πολλαπλών threads, οι οποίες συναλλαγές υφίστανται επεξεργασία για την αποδοχή ή απόρριψή τους και ανάλογα τοποθετούνται ή όχι στη λίστα των εκκρεμών συναλλαγών, η οποία θα τροφοδοτήσει τα επόμενα blocks.

Noobcash client

Ο Noobcash client της εφαρμογής υλοποιείται μέσω των μεθόδων που περιλαμβάνονται στο αντίστοιχο `client.py` αρχείο και ενεργοποιείται με χρήση της εντολής `python3 client.py` εκτελεσμένης στο κατάλληλο directory. Μέσω του `noobcash cli` ο χρήστης μπορεί να πραγματοποιήσει συναλλαγές, να δει τις δοσοληψίες που συμπεριλαμβάνονται στο τελευταίο block της αλυσίδας και να ενημερωθεί για το υπόλοιπο του wallet που του αντιστοιχεί. Ακόμα διατίθεται ως δευτερεύουσα λειτουργία η παροχή βοήθειας σε περίπτωση που ο χρήστης δε γνωρίζει τις διαθέσιμες εντολές του cli (`help`). Σημειώνεται ότι στην περίπτωση της εντολής `t <recipient_address> <amount>` έχει προτιμηθεί η χρήση της παραμέτρου `recipient_id` αντί της παραμέτρου `recipient_address` με στόχο τη διευκόλυνση του χρήστη.

Παρακάτω παρατίθεται μέσω στιγμιότυπων ένα session χρήσης:

```
(venv) user@snf-25403:~/Noobcash_Blockchain$ python3 client.py

Available commands:
===================
balance  bye  help  t  view

Noobcash client! You can spend your money now!
noobcash>>
```

```
noobcash>>help

Available commands:
===================
balance  bye  help  t  view

noobcash>>help t
t <recipient_id> <amount>
        Send to recipient with id <recipient_id> <amount> coins
noobcash>>
```

```
noobcash>>balance
189
noobcash>>t 4 3
Done!
noobcash>>balance
186
noobcash>>view
Node 3 sent to Node 4 1 coins on Tue, 29 Mar 2022 16:46:51.
Node 3 sent to Node 1 2 coins on Tue, 29 Mar 2022 16:46:52.
Node 0 sent to Node 1 3 coins on Tue, 29 Mar 2022 17:06:09.
Node 0 sent to Node 1 3 coins on Tue, 29 Mar 2022 17:06:48.
Node 0 sent to Node 4 3 coins on Tue, 29 Mar 2022 17:06:55.
Node 0 sent to Node 3 3 coins on Tue, 29 Mar 2022 17:07:02.
Node 0 sent to Node 2 3 coins on Tue, 29 Mar 2022 17:07:10.
Node 0 sent to Node 1 3 coins on Tue, 29 Mar 2022 17:07:19.
Node 0 sent to Node 1 3 coins on Tue, 29 Mar 2022 17:07:28.
Node 0 sent to Node 4 3 coins on Tue, 29 Mar 2022 17:07:30.
noobcash>>t 3 1000
Failed!
noobcash>>bye
(venv) user@snf-25403:~/Noobcash_Blockchain$
```

Αξιολόγηση Συστήματος – Πειράματα

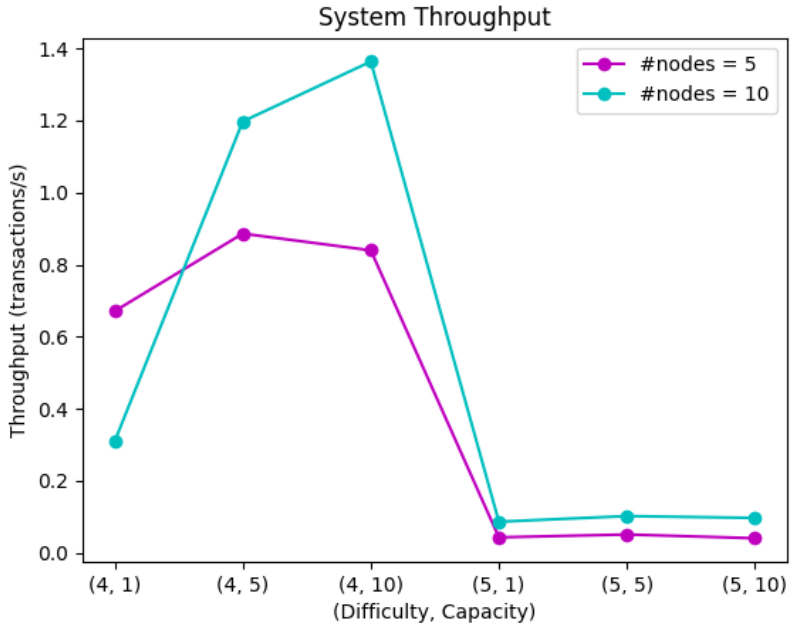
Στο σκέλος αυτό το σύστημα μας αξιολογείται ως προς την απόδοση και την κλιμακωσιμότητά του, μέσω της διεξαγωγής κάποιων πειραμάτων. Όλα τα πειράματα εκτελέστηκαν με χρήση Virtual Machines της υπηρεσίας Okeanos.

Για την αξιολόγηση της απόδοσης και της κλιμακωσιμότητας του συστήματος χρησιμοποιείται δίκτυο 5 και 10 κόμβων, όπου ο κάθε κόμβος εκτελεί ένα σύνολο από 100 transactions. Τα transactions αυτά ορίζονται για κάθε κόμβο στα δοθέντα .txt αρχεία, ενώ οι δοσοληψίες εκτελούνται για χωρητικότητα block 1, 5 και 10 και τιμές difficulty 4 και 5. Οι μετρικές που καταγράφονται είναι το throughput (ρυθμαπόδοση) και το block time του συστήματος, τα οποία ορίζονται όπως φαίνεται παρακάτω:

$$throughput = \frac{\#transactions}{t_{last\_mining} - t_{first\_trans}},$$

όπου t_{last_mining} ο χρόνος που ολοκληρώνεται το mining του τελευταίου block του blockchain και t_{first_trans} ο χρόνος δημιουργίας της πρώτης δοσοληψίας του δικτύου.
Αντίστοιχα, το block time ορίζεται ως ο μέσος χρόνος που απαιτείται για το mining ενός block (την εύρεση της κατάλληλης τιμής του `nonce`). Επίσης, σημειώνεται ότι στον υπολογισμό της μετρικής της ρυθμαπόδοσης δε λαμβάνονται υπόψη η συναλλαγή εντός του genesis block και οι πρώτες προκαθορισμένες συναλλαγές που αφορούν τη μεταφορά των αρχικών 100 NBCs από τον bootstrap κόμβο σε καθένα εκ των υπολοίπων. Ομοίως, και στο μέσο χρόνο mining ενός block δε λαμβάνονται υπόψη το genesis block αλλά και όσα επόμενα block περιέχουν αποκλειστικά μόνο τέτοιες αρχικές συναλλαγές.

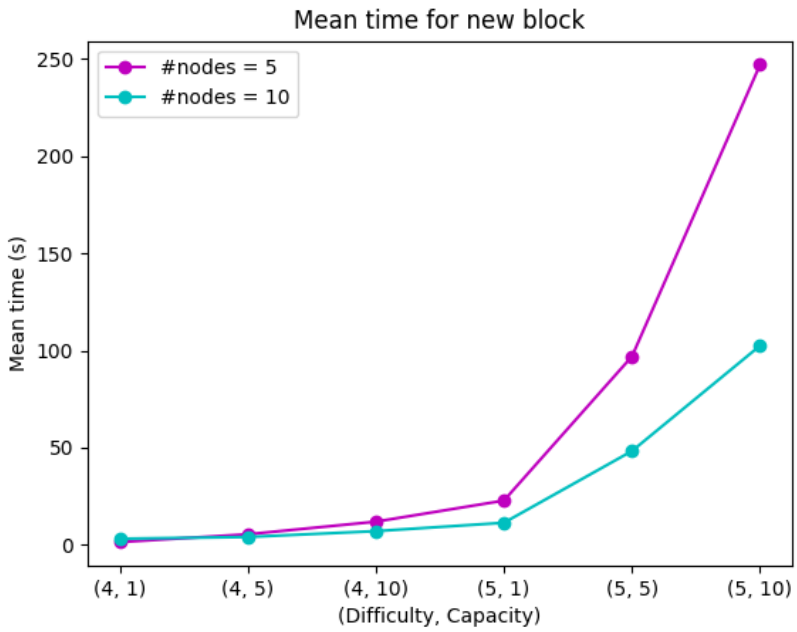
Γραφική παράσταση Ρυθμαπόδοσης – (Mining difficulty, Capacity) που προέκυψε έπειτα από εκτέλεση των πειραμάτων:



Σχόλια/Παρατηρήσεις:
Με βάση το παραπάνω διάγραμμα αρχικά παρατηρούμε ότι η αύξηση του mining difficulty προκαλεί σημαντική μείωση της ρυθμαπόδοσης και για τις δύο τιμές πλήθους κόμβων, κάτι το οποίο μπορεί να εξηγηθεί από την αναμενόμενη αύξηση του μέσου χρόνου που απαιτείται για το mining του κάθε block. Αυτή η αύξηση αναπόφευκτα οδηγεί σε υψηλότερο χρόνο t_{last_mining} και στις δύο περιπτώσεις, κι επομένως πτώση της τιμής throughput. Όσον αφορά τις αλλαγές στην παράμετρο της χωρητικότητας, παρατηρούμε ότι μεταβάλλοντας το capacity από την τιμή 1 σε 5, παρατηρείται αύξηση του throughput για όλους τους συνδυασμούς πλήθους κόμβων και mining difficulty. Αυτό μπορεί να δικαιολογηθεί από τη μαζικότερη συμπερίληψη των δοσοληψιών σε blocks και τελικά τον περιορισμό του συνολικού αριθμού blocks που χρειάζονται για την εξυπηρέτηση όλων των transactions. Καταλήγουμε στο ότι τα προαναφερθέντα οδηγούν σε μείωση του συνολικού χρόνου mining που απαιτείται, κι επομένως σε αύξηση της ρυθμαπόδοσης. Παρ' όλα αυτά, κατά την περίπτωση διπλασιασμού του capacity (αλλαγή τιμής από 5 σε 10), σε ορισμένες περιπτώσεις φαίνεται τελικά να έχουμε περιορισμένη μείωση του throughput, αντί για αύξηση. Μπορούμε να συμπεράνουμε ότι μεγάλο πλήθος transactions ανά block μπορεί τελικά να προκαλέσει άνοδο του μέσου και συνολικού χρόνου mining, λόγω πιο χρονοβόρας διαδικασίας hashing κατά τον υπολογισμό του nonce. Τέλος, σημειώνουμε ότι για όλα τα ζεύγη (difficulty, capacity) εκτός του (4,1), το αυξημένο πλήθος κόμβων φαίνεται να ευνοεί την ρυθμαπόδοση, λόγω του μεγαλύτερου συνολικού αριθμού transactions που πραγματοποιούνται (1000 για n=10, έναντι 500 για n=5), καθώς και λόγω του συντομότερου μέσου χρόνου mining που προκύπτει, λόγω των περισσότερων κόμβων που διεξάγουν παράλληλα mining για κάθε block. Αξίζει να επισημανθεί ότι για n = 10 και (difficulty, capacity) = (4, 1), είναι αναμενόμενο τα blocks να εκδίδονται με τη μέγιστη παρατηρούμενη ταχύτητα, λόγω μεγαλύτερου αριθμού

κόμβων στο δίκτυο, μειωμένης mining δυσκολίας και περιορισμένης χωρητικότητας. Έτσι εμφανίζονται πολλές διαφορετικές διακλαδώσεις στην αλυσίδα και ανακύπτει η ανάγκη επίλυσης των αντίστοιχων συγκρούσεων, με αποτέλεσμα την αφιέρωση περισσότερου χρόνου σε αυτό και την πτώση του throughput.

Γραφική παράσταση Block time – (Mining difficulty, Capacity):



Σχόλια/Παρατηρήσεις:
Έπειτα από παρατήρηση της παραπάνω γραφικής παράστασης, μπορούμε αρχικά να συμπεράνουμε ότι η αύξηση του mining difficulty οδηγεί σε άνοδο του block time για όλες τις τιμές πλήθους κόμβων και χωρητικότητας. Αυτό μπορεί να δικαιολογηθεί από τον αναμενόμενα δυσκολότερο υπολογισμό του nonce και κατά επέκταση τον υψηλότερο χρόνο που απαιτείται για το mining κάθε block. Αντίστοιχα, παρατηρούμε ότι αύξηση του capacity γενικά φαίνεται να προκαλεί άνοδο του block time, κάτι που μπορεί να εξηγηθεί από πιο χρονοβόρο hashing κατά τον υπολογισμό του nonce, λόγω του υψηλότερου πλήθους transactions ανά block. Τέλος, όπως και πριν, μέσω της γραφικής μπορούμε να συμπεράνουμε ότι διπλασιασμός του αριθμού των κόμβων του δικτύου συνεπάγεται μείωση του mean time. Η πτώση αυτή οφείλεται στην παράλληλη εργασία 10 κόμβων αντί για 5 με στόχο την εύρεση του `nonce` που έχει ως συνέπεια την αύξηση της πιθανότητας της επιτυχίας του mining σε μικρότερο χρονικό διάστημα.