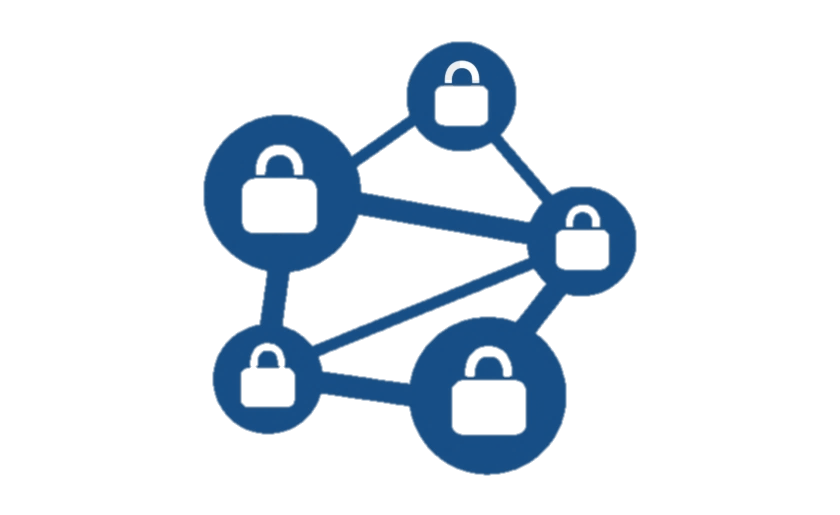
Εθνικό Μετσόβιο ΠολυτεχνείοΣχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών ΥπολογιστώνΕξάμηνο 9ο: Κατανεμημένα Συστήματα *(2021-2022)*  
Αναφορά Εξαμηνιαίας Εργασίας



*el17088 - Στυλιανός Κανδυλάκης*

*el17025 - Χρήστος (Κίτσος) Ορφανόπουλος*

*el17176 - Χρήστος Τσούφης*

**Εφαρμογή**

Υλοποιήσαμε ένα απλό σύστημα blockchain όπου καταγράφονται οι δοσοληψίες και το consensus επιτυγχάνεται με proof-of-work.

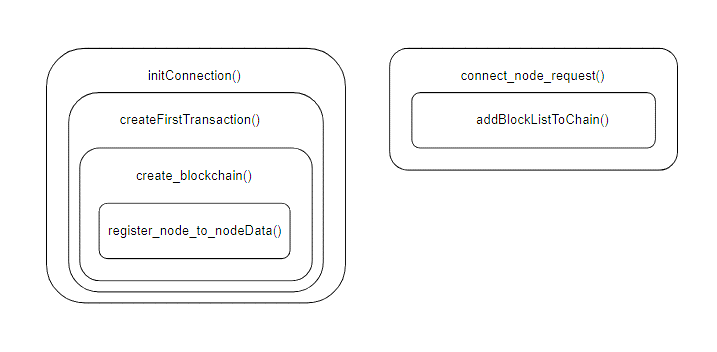
**Προ-απαιτούμενα**

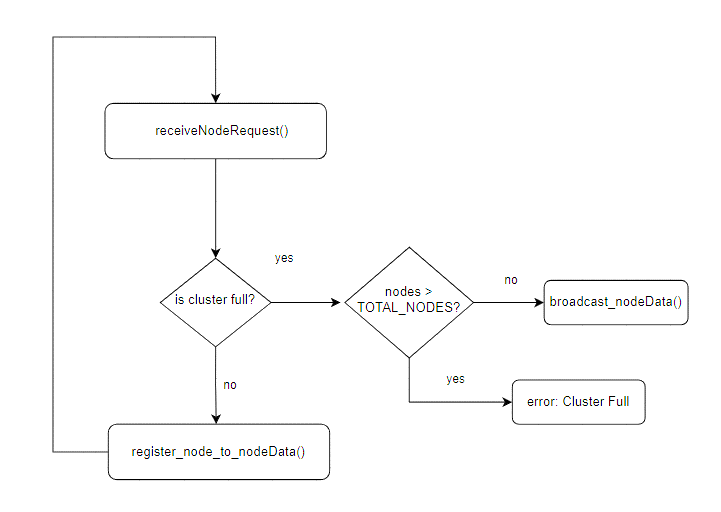
* Flask Rest API
* Python3
* Python libraries (requests, pycrypto, datetime, time, Flask, urllib)
* 5 VMs του Okeanos (2CPUs, 2GB RAM, 10GB per VM)
* 1 private network

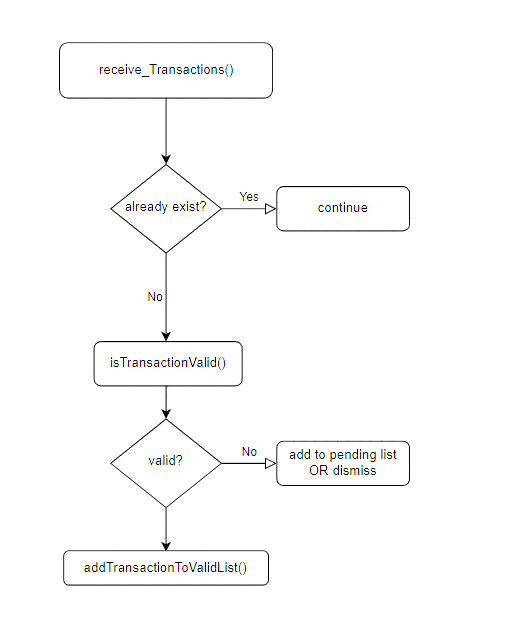
**Global Παράμετροι**  
- TOTAL\_NODES  
- CAPACITY  
- MINING\_DIFFICULTY

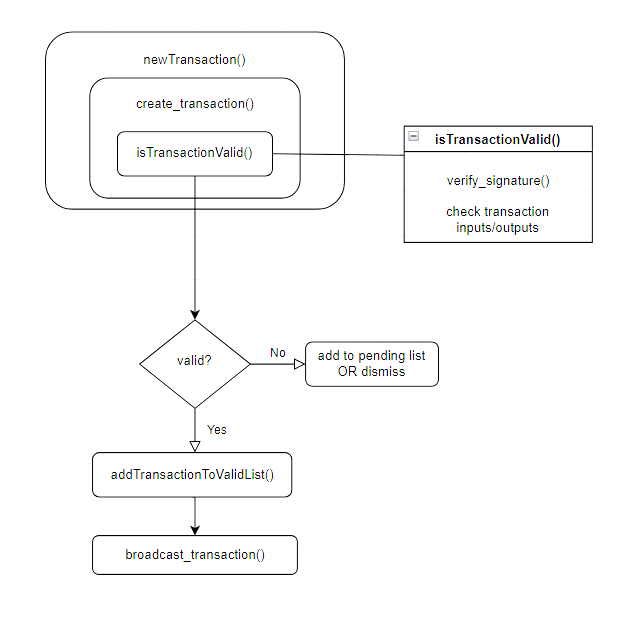
**Βασικές λειτουργίες**

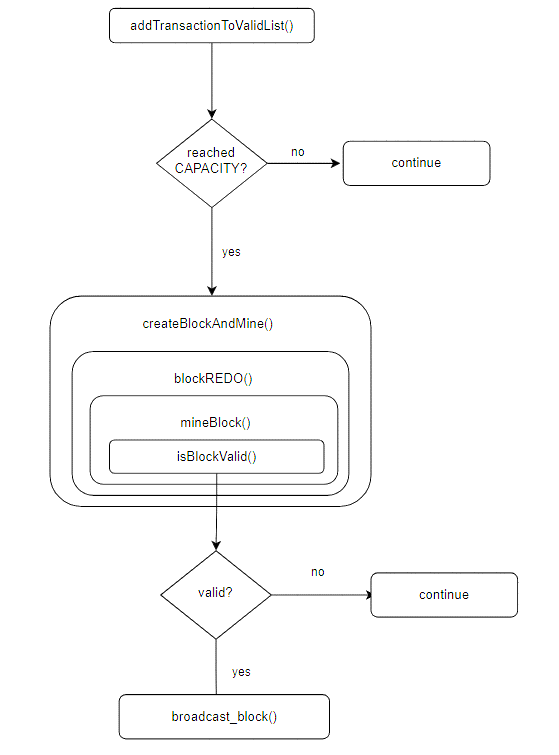
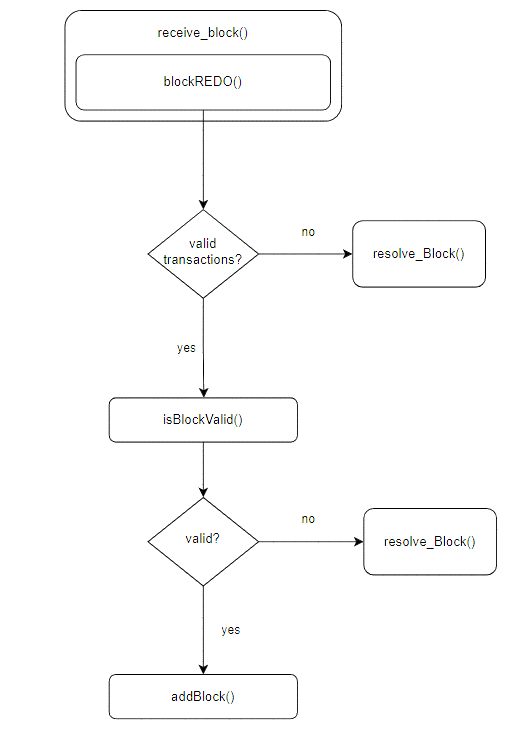
**Αρχικοποίηση του cluster**



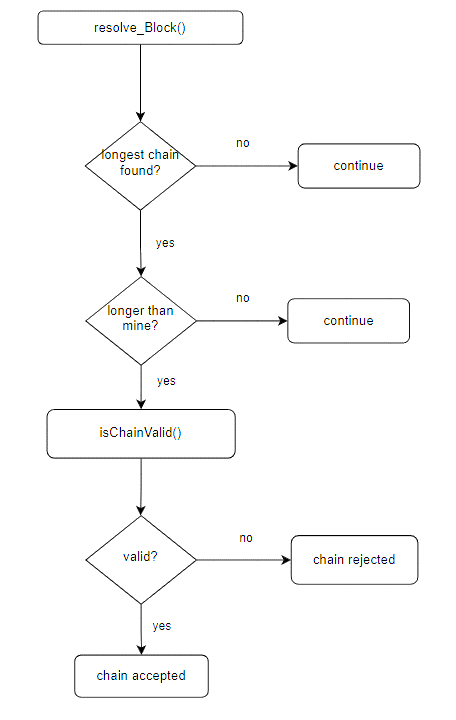


**Δημιουργία συναλλαγής**

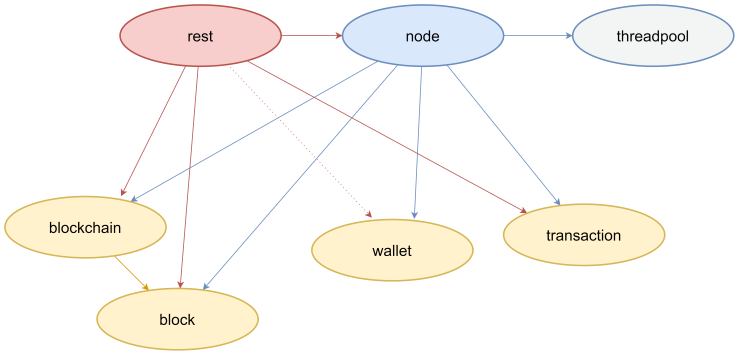


**Mining**

**Conflict resolution**



**Κλάσεις**



**Rest**

**initConnection**()

καλείται από τον bootstrap ώστε να αρχικοποιηθεί το cluster

**connect\_node\_request**()

αίτημα σύνδεσης συγκεκριμένου node στο cluster

**receiveNodeRequest**()  
καλείται από τον bootstrap όταν πρόκειται να αποδεχθεί κάποιο node, αποστέλλοντας μήνυμα αποδοχής

**receiveTransactions**()  
οι κόμβοι λαμβάνουν την validated transaction που έγινε broadcast

**receiveBlock**()  
λαμβάνει block που έγινε broadcast

**getBlockchain**()  
αποστέλλει το blockchain ως json

**getChainLength**()  
επιστρέφει το μήκος συγκεκριμένου blockchain ως json

**newTransaction**()  
Κόμβος ζητάει τη δημιουργία νέου transaction

**showBalance**()  
Επιστρέφει το balance του wallet

**viewBlockchain**()Επιστρέφει το blockchain

**getTransactions**()  
Επιστρέφει τα transactions

**viewTransactions**()  
Επιστρέφει τα transactions

**transactionsTime**()

επιστρέφει χρονικό διάστημα ενός transaction

**getBlockTime**()

επιστρέφει μέσο χρόνο δημιουργίας block

**Node**

- το ID του  
- το wallet του   
- το chain από blocks για το τρέχον κόμβο  
- το ring με πληροφορίες για όλο το cluster  
- valid\_trans: λίστα με όσα transactions έχουν γίνει validated  
- pending\_trans: λίστα με transactions που δεν έχουν γίνει validated ακόμα   
- unreceived\_trans: λίστα με transactions που λήφθηκαν μέσω block που έγινε broadcast

**broadcast**()  
Βασική συνάρτηση που καλείται σε οποιοδήποτε broadcast

**broadcast\_transaction**()  
Το transaction αποστέλλεται με broadcast σε όλους τους κόμβους

**broadcast\_block**()  
Μόλις βρεθεί ο κατάλληλος nonce, ο κόμβος κάνει broadcast το επαληθευμένο block σε όλους τους υπόλοιπους κόμβους.

**broadcast\_nodeData**()  
Γίνονται broadcast στο cluster όλες οι απαραίτητες πληροφορίες των κόμβων

**addBlockListToChain**()  
Μετατρέπει τη json λίστα σε dictionary από blocks επιστρέφοντας list από block objects

**register\_node\_to\_nodeData**()  
Ο bootstrap node προσθέτει το νέο κόμβο στο cluster ενημερώνοντας τους υπολοίπους για τη προσθήκη του

**public\_key\_to\_nodeData\_id**()  
Επιστρέφει το id του node, δοσμένου του public key

**createFirstTransaction**()  
Δημιουργεί το πηγαίο transaction

**createTransaction**()   
Δημιουργείται ένα νέο transaction που περιέχει όλα τα στοιχεία που απαιτούνται. Εδώ το πεδίο transaction\_inputs γεμίζει με τα Transaction Input που περιέχουν τα ids των UTXOs που απαιτούνται για να συμπληρωθεί το ποσό που θέλουμε να ξοδέψουμε.

**isTransactionValid**()  
Eπαληθεύεται η ορθότητα του transaction που έχει ληφθεί.

Η επαλήθευση περιλαμβάνει

(α) την επαλήθευση της υπογραφής (verify\_signature)

(β) τον έλεγχο των transaction inputs/outputs για να εξασφαλίσουμε ότι το wallet αποστολέας έχει το ποσό amount που μεταφέρει στον παραλήπτη.   
Για να επιτευχθεί το (β) ελέγχεται αν τα transaction inputs είναι πράγματι unspent transactions, αν είναι αφαιρούνται από τη λίστα των UTXO του κόμβου. Δημιουργούνται τα δύο Transaction Outputs και προστίθενται στη λίστα στη λίστα UTXO του node μας.

**validate\_pending**()  
Ελέγχει αν μπορεί να επικυρωθεί κάποια pending transaction

**add\_transaction\_to\_pending**()  
προσθήκη transactions ως pending

**removeFromOldTransactions** ()

διαγραφή transaction από τη λίστα με τις παλιότερες

**addTransactionToValidList**()  
Προσθέτει validated transaction στη λίστα με τις validated

**receive\_block**()  
Λαμβάνει block που έγινε broadcasted

**createNewBlock**()  
Δημιουργεί νέο block

**mineBlock**()  
Η συνάρτηση αυτή καλείται μόλις capacity transactions έχουν ληφθεί και επαληθευτεί από κάποιον κόμβο και υλοποιεί το proof of work δοκιμάζοντας διαφορετικές τιμές της μεταβλητής nonce και hashάροντας το block μέχρι το hash που θα προκύψει να αρχίζει από έναν συγκεκριμένο αριθμό από μηδενικά. Ο αριθμός αυτός καθορίζεται από τη σταθερά difficulty.

**isBlockValid**()  
Αυτή η συνάρτηση καλείται από τους nodes κατά τη λήψη ενός νέου block (εκτός του genesis block). Επαληθεύεται ότι

(a) το πεδίο current\_hash είναι πράγματι σωστό

(b) το πεδίο previous\_hash ισούται πράγματι με το hash του προηγούμενου block.

**createBlockAndMine**()  
Δημιουργεί block και εκκινεί το mining

**blockREDO**()  
Εκτελεί πάλι τις συναλλαγές του block για να βεβαιωθεί ότι είναι έγκυρες, ελέγχει αν τα transactions έχουν προστεθεί ήδη στο chain και στη συνέχεια μέσα σε lock κάνει validate το block και το προσθέτει αν είναι έγκυρο. Ελέγχει τέλος εκτός του lock αν μπορεί να κάνει validate συναλλαγές των pending, χρησιμοποιώντας τα inputs του νέου block.

Σημειώνουμε ότι η συνάρτηση block\_REDO χρησιμοποιείται και σε άλλες περιπτώσεις όπου χρειάζεται έλεγχος των transactions ενός block.

**isChainHashesValid**()  
Κάνει validate τα hashes του blockchain

**isChainValid**()  
Αυτή η συνάρτηση καλείται από τους νεοεισερχόμενους κόμβους, οι οποίοι επαληθεύουν την ορθότητα του blockchain που λαμβάνουν από τον bootstrap κόμβο. Στην πραγματικότητα καλείται η validate\_block για όλα τα blocks εκτός του genesis.

**resolveBlock**()  
Αυτή 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. Ο κόμβος ρωτάει τους υπόλοιπους για το μήκος του blockchain και επιλέγει να υιοθετήσει αυτό με το μεγαλύτερο μήκος.

**Wallet** (subclass)

- private key  
- public key  
- utxos

**balance()**  
Επιστρέφει το υπόλοιπο οποιουδήποτε wallet προσθέτοντας όλα τα UTXOs που έχουν παραλήπτη το συγκεκριμένο wallet.

**Block**

- index: αρίθμηση του block μέσα στο blockchain  
- hash: hash του εαυτού του (SHA256)  
- previous\_hash: hash του προηγούμενου block  
- transactions: λίστα με transactions του μπλοκ

- timestamp: χρονοσφραγίδα για τη στιγμή που δημιουργήθηκε  
- nonce: το νούμερο για το proof-of-work

**myHash()**   
Υπολογίζει το hash value ολόκληρου του block

**listToSerializable()**

**printΒlock()**Τυπώνει τις πληροφορίες του block

**Blockchain**

- block\_list: λίστα με τα blocks του τρέχοντος node

**createBlockchain()**   
Καλείται από το bootstrap node και για τη δημιουργία του genesis block.

**printChain**()  
Εκτυπώνει το chain με πληροφορίες που περιέχει το κάθε block

**addBlock**()  
Προσθήκη block στο chain

**Transaction**

- sender: public key του αποστολέα  
- receiver: public key του αποδέκτη  
- senderID: το ID του κόμβου-αποστολέα στο cluster  
- receiverID: το ID του κόμβου-αποδέκτη στο cluster  
- amount: το ποσό που μεταφέρεται  
- id: hash του transaction (hash(public key + amount + transaction\_inputs))  
- transaction\_inputs:  
- transaction\_outputs:  
- signature:

∙ **hash**()

Αρχικοποιημένη συνάρτηση για το hashing ( SHA384 )

∙ **signTransaction**()  
Υπογράφεται το transaction με το private key του wallet.

∙ **verify\_signature**()  
Επαληθεύεται η υπογραφή του transaction αμέσως μετά τη λήψη του

**Πειράματα**

Μετρήσαμε την απόδοση για 5 και 10 nodes αντίστοιχα. Στη δεύτερη περίπτωση 2 nodes ανά virtual machine.  
  
Μετρικές απόδοσης

* throughput (transactions per sec )
* block time (creation and mining of block)

Παράμετροι

* CAPACITY (= 1, 5 , 10)
* DIFFICULTY (= 4, 5 )

Για 5 nodes:

Για 10 nodes:

**Σχολιασμός Αποτελεσμάτων**

**Ως προς capacity/difficulty:**

* Throughput

Όσο αυξάνεται το capacity του block, αυξάνεται και η απόδοση, καθώς μειώνεται η συχνότητα του mining.

Αύξηση του difficulty οδηγεί σε συνολική μείωση της απόδοσης, αφού το mining χρειάζεται περισσότερο χρόνο για να βρει το nonce.

* Block time

To capacity δεν επηρεάζει ιδιαίτερα το το Block time.  
  
Αύξηση του difficulty οδηγεί σε αύξηση του Block time, διότι χρειάζεται επιπλέον χρόνος για εύρεση του nonce.

**Ως προς τους nodes:**

* Throughput

αύξηση των nodes οδηγεί σε αύξηση της απόδοσης, αφού περισσότεροι nodes κάνουν ταυτόχρονα mining το εκάστοτε block

* Block time

αύξηση των nodes οδηγεί σε μείωση του block time, αφού περισσότεροι nodes κάνουν ταυτόχρονα mining το εκάστοτε block, οπότε αυξάνεται η πιθανότητα να βρεθεί το nonce από κάποιον miner

**Πηγές**

<https://ravikantagrawal.medium.com/digital-signature-from-blockchain-context-cedcd563eee5>

<https://www.youtube.com/watch?v=Lx9zgZCMqXE&ab_channel=CuriousInventor>

<https://www.youtube.com/watch?v=xIDL_akeras&ab_channel=AndersBrownworth>

<https://flask-restful.readthedocs.io/en/latest/api.html>

Github εργασίας: [**Source code**](https://github.com/stylkand/Distributed-Systems)