ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη κατανεμημένων πρωτοκόλλων συναίνεσης και Blockchain με χρήση Βάσης Δεδομένων Γράφου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Φράγκος

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη κατανεμημένων πρωτοκόλλων συναίνεσης και Blockchain με χρήση Βάσης Δεδομένων Γράφου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Φράγκος

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Σεπτεμβρίου 2018.

.....	.....	.....
Θεοδώρα Βαρβαρίγου Καθηγήτρια Ε.Μ.Π.	Δημήτριος Ασκούνης Καθηγητής Ε.Μ.Π.	Συμεών Παπαβασιλείου Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2018

.....

Γεώργιος Φράγκος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το διαδίκτυο αποτελεί το πλέον ισχυρό εργαλείο ψηφιακής τεχνολογίας και έχει φέρει επανάσταση στον τρόπο που οι άνθρωποι αντιλαμβάνονται τον κόσμο, ενώ παράλληλα είναι άρρηκτα συνδεδεμένο με την καθημερινότητα τους. Παρόλο που οι τεχνολογίες στις οποίες στηρίζεται αναπτύσσονται με πολύ γρήγορο ρυθμό, η κυρίαρχη αρχιτεκτονική που το διέπει είναι το μοντέλο αρχιτεκτονικής πελάτη-εξυπηρετητή (client-server model) η οποία τείνει να είναι πλήρως κεντρικοποιημένη. Πλέον όμως, εμφανίζονται νέες καινοτόμες τεχνολογίες, όπως το Blockchain, οι οποίες συμβάλλουν στην κατεύθυνση αποκεντρικοποίησης της διαχείρησης των διαδικτυακών εφαρμογών.

Η τεχνολογία Blockchain, η οποία είναι αδιαμφισβήτητα μια έξυπνη εφεύρεση, αποτελεί το πνευματικό τέκνο ενός ατόμου ή μιας ομάδας ατόμων που είναι γνωστοί με το ψευδώνυμο ‘Shatoshi Nakamoto’ και αρχικά επινοήθηκε το 2008 ως η βασική τεχνολογία για το ψηφιακό νόμισμα Bitcoin. Συγκεκριμένα το blockchain είναι ένα ψηφιακό κατανεμημένο δημόσιο καθολικό (public ledger) ‘βιβλίο’ στο οποίο καταγράφονται συναλλαγές και συμφωνίες με τρόπο αδιάβλητο και υποστηρίζεται από ένα δίκτυο ομότιμων κόμβων (peer-to-peer network).

Σκοπός της παρούσας διπλωματικής εργασίας είναι η διερεύνηση της τεχνολογίας blockchain και της πλέον σύγχρονης κατανεμημένης βάσης δεδομένων γράφου Neo4j, καθώς και η ανάπτυξη μιας αποκεντρωμένης εφαρμογής που θα συνδυάζει τα δύο ανωτέρα πεδία. Πιο συγκεκριμένα, θα υλοποιήσουμε ένα blockchain prototype σε γλώσσα προγραμματισμού Python το οποίο θα έχει ενσωματωμένη μια κατανεμημένη βάση δεδομένων γράφου Neo4j και θα αλληλεπιδρούν μεταξύ τους.

Κατά τη διάρκεια εκπόνησης της διπλωματικής, μελετήθηκαν σε βάθος οι προαναφερθείσες τεχνολογίες και αναπτύχθηκαν νέες τεχνικές για την πιο αποδοτική εξόρυξη δεδομένων και πληροφοριών από το blockchain χρησιμοποιώντας παράλληλα την Neo4j. Τέλος μελετήθηκαν και υλοποιήθηκαν ήδη υπάρχοντα καθώς και νέα καινοτόμα κατανεμημένα πρωτόκολλα συναίνεσης.

Λέξεις κλειδιά

Blockchain, Βάση Δεδομένων Γράφου – Neo4j, Πρωτόκολλα συναίνεσης, Consensus, Ερωτήματα Cypher – Queries, Συναλλαγές, Python, Flask

Abstract

Internet is the most powerful digital technology tool and has revolutionized the way people perceive the world while being indissolubly linked to their everyday life. Although the technologies on which it is based on are evolving at a very fast pace, the dominant architectural model that governs is the client-server architecture that tends to be fully centralized. However nowadays, new innovative technologies, such as Blockchain, are emerging and they are contributing to the decentralization of the Internet.

Blockchain, which is undoubtedly an extremely smart invention, is the spiritual child of a person or a group of people known by the nickname ‘Shatoshi Nakamoto’ and was originally devised in 2008 as the basic technology behind the Bitcoin cryptocurrency. Specifically, blockchain is a public, digitalized and distributed ledger in which transactions and agreements are recorded in a transparent manner and based on a peer to peer network.

The purpose of this diploma thesis is to research the Blockchain technology and the groundbreaking Neo4j Graph Database, as well as to develop a decentralized application which combines the two aforementioned fields. In particular, I will implement a blockchain prototype in Python programming language integrated with a distributed Neo4j Graph Database.

During the development of this diploma thesis, I thoroughly researched the aforementioned technologies and the way that decentralized systems operate. Furthermore, new techniques for the more efficient blockchain data extraction and analysis were developed, by using the Neo4j. Finally, already existing as well as new innovative consensus protocols have been studied and developed.

Keywords

Blockchain, Graph Database - Neo4j, Consensus Methods, Cypher Queries, Transactions, Python, Flask.

Ευχαριστίες

Η πτυχιακή εργασία εκπονήθηκε στον τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής στη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Η υπόδειξη του θέματος έγινε σε συνεργασία με την την καθήγητρια κυρία Θεοδώρα Βαρβαρίγου, καθώς και από τον δόκτορα Αντώνη Λίτκε και υποψήφιο διδάκτορα Γιώργο Παλαιοκρασσά.

Ένα τεράστιο και εγκάρδιο ευχαριστώ αξίζουν οι γονείς μου και ο αδερφός μου, που με στηρίζουν με όλους τους τρόπους όλα αυτά τα χρόνια, δίνοντας μου κουράγιο να προχωρώ και να υπερπηδώ κάθε εμπόδιο για να να υλοποιήσω τους στόχους μου. Επίσης θα ήθελα να ευχαριστήσω όλους μου τους φίλους.

Θα ήθελα επίσης να ευχαριστήσω την κυρία Θεοδώρα Βαρβαρίγου που με εμπιστεύτηκε στην περάτωση μιας τόσο ενδιαφέρουσας ερευνητικής διπλωματικής εργασίας. Επίσης, αξίζουν ένα μεγάλο ευχαριστώ ο δόκτορας Αντώνης Λίτκε και ο υποψήφιος διδάκτορας Γιώργος Παλαιοκρασσάς που με την πολύτιμη βοήθεια αλλά και την άψογη καθοδήγηση τους όλον αυτόν τον καιρό με καθοδήγησαν στην εκπόνηση της διπλωματικής εργασίας.

Τέλος, όχι κατά σειρά, θερμές ευχαριστίες απευθύνω σε όλο το διδακτικό προσωπικό του Εθνικού Μετσόβιου Πολυτεχνείου για τις γνώσεις που μου μετέδωσαν και με έκαναν καλύτερο άνθρωπο.

Γεώργιος Φράγκος
Αθήνα, Σεπτέμβριος 2018

Περιεχόμενα

Κατάλογος Σχημάτων

1

Εισαγωγή

Η σύγχρονη εποχή διακρίνεται από το γεγονός πως οι υπολογιστές και το Internet έχουν εισέλθει δυναμικά στην καθημερινή ζωή των ανθρώπων και γι'αυτόν τον λόγο ονομάζεται και εποχή της ψηφιοποίησης. Στην αρχή ψηφιοποιήθηκε η πληροφορία και έτσι έγινε δυνατή η ταχύτητα διάδοσης της σε πολύ μεγάλες αποστάσεις μέσω του διαδικτύου, πράγμα φυσικά που οδήγησε στην παγκοσμιοποίηση της γνώσης. Στην συνέχεια δημιουργήθηκαν τα ηλεκτρονικά καταστήματα (e-shops) τα οποία συνέβαλλαν αρκετά στην ανάπτυξη του διεθνούς εμπορίου. Έπειτα ακολούθησαν τα κοινωνικά δίκτυα, όπως για παράδειγμα το Facebook, Twitter, Instagram και άλλα, τα οποία τελικά κατάφεραν να συνδέσουν ανθρώπους από όλο τον κόσμο και έδωσαν δυνατότητα σε νέες μορφές επικοινωνίας και διαλόγου.

Η έννοια του κρυπτονομίσματος είχε πρωτοεμφανιστεί σαν ιδέα το 1998 από τον Wei Dai στη λίστα αλληλογραφίας cypherpunks [1]. Εκεί αναφερόταν στην ιδέα ενός νέου χρήματος όπου θα χρησιμοποιούσε κρυπτογραφία για να ελέγξει τις συναλλαγές. Έτσι μετά από δέκα χρόνια περίπου, το 2008, πρωτοεμφανίστηκε το Bitcoin όταν κάποιος Shatoshi Nakamoto (δεν έχει ταυτοποιηθεί ακόμα ποιος ακριβώς είναι) δημοσίευσε μια ερευνητική εργασία με τίτλο <<A Peer-to-Peer Electronic Cash System>> [2]. Από εκείνο το σημείο και μετά, υπήρξε τεράστια αλλαγή όσον αφορά την αντίληψη του κόσμου για τις χρηματοοικονομικές συναλλαγές καθώς και για το διαδίκτυο γενικότερα. Η ψηφιοποίηση του χρήματος ήταν πλέον γεγονός, και οι οποιεσδήποτε συναλλαγές δεν ελέγχονται από καμία κεντρική τράπεζα, κυβέρνηση ή μεσάζοντα (third party) γενικότερα. Αντίθετα η ασφαλής λειτουργία των κρυπτονομισμάτων βασίζεται σε ένα σύνολο από κρυπτογραφικά πρωτόκολλα και σίγουρα στην καινοτόμα τεχνολογία η οποία ονομάζεται Blockchain.

Το Blockchain είναι ένα κατανεμημένο δημόσιο ψηφιακό λογιστικό βιβλίο στο οποίο καταγράφονται συναλλαγές με τρόπο επαληθεύσιμο και αδιάβλητο. Παρόλο που χρησιμοποιείται κυρίως ως τρόπος παρακολούθησης και επαλήθευσης των νομισματικών συναλλαγών,

μπορεί επίσης να εντοπίζει και να ελέγχει οποιδήποτε είδος δεδομένων, καθιστώντας την μια απίστευτα ασφαλή πλατφόρμα που έχει τη δυνατότητα να αλλάξει ολόκληρο το διαδίκτυο. Κάθε νέα συστάδα καταχωρήσεων πληροφορίας ονομάζεται μπλοκ και συνδέεται με τα προηγούμενα ως το επόμενο κομμάτι της αλυσίδας. Λόγω του αποκεντρωμένου (decentralized) χαρακτήρα του, το blockchain βασίζει στην λειτουργία του σε ένα peer-to-peer δίκτυο, δηλαδή σε πολλούς υπολογιστές ανα τον κόσμο που καθένας από αυτούς αποτελεί έναν ισότιμο κόμβο (peer). Κάθε τέτοιος κόμβος έχει ακριβές αντίγραφο όλης της πληροφορίας που είναι καταγεγραμμένη στο blockchain από την αρχή της δημιουργίας της συγκεκριμένης δομής. Επιπροσθέτως, σημειώνεται πως ο καθένας έχει πρόσβαση στο blockchain αλλά χωρίς πραγματικά χαρακτηριστικά αναγνώρισης, διασφαλίζοντας έτσι την ανωνυμία του αλλά και την ασφάλεια του. Επίσης, ότι γράφεται στην δομή αυτή είναι αδύνατο να διαγραφεί ή να τροποποιηθεί λόγω συγκεκριμένων πρωτοκόλλων κρυπτογράφησης που διατηρούν την διαφάνεια του blockchain, αφού σε όλους παρουσιάζεται η ίδια πληροφορία. Από όλα τα παραπάνω αντιλαμβανόμαστε πως η νέα αυτή η τεχνολογία έχει την δυναμική να αποκεντρικοποιήσει την διαχείριση εφαρμογών και υπηρεσιών, κάνοντας μερικά βήματα προς τον εκδημοκρατισμό του διαδικτύου.

Εικόνα 1.1 <Κρυπτονομίσματα>

Συγχρόνως, παρατηρούμε πως στο διαδίκτυο αν και ο όγκος των δεδομένων είναι τεράστιος, πλέον μας απασχολεί περισσότερο η σχέση μεταξύ αυτών καθώς και τα μοτίβα τα οποία υπάρχουν μεταξύ αυτών. Για αυτόν τον λόγο εισάγω στην διπλωματική σε συνδυασμό με την διερεύνηση του blockchain, και την Βάση Δεδομένων Γράφου Neo4j [3] όπου είναι εξαιρετικά νέα και πολλά υποσχόμενη τεχνολογία και χαρακτηρίζεται από τα εξής γνωρίσματα: Τα δεδομένα και τα σχήματα αναπαριστώνται με τη μορφή γράφων, η διαχείριση των δεδομένων πραγματοποιείται μέσω των μετασχηματισμών των γράφων και τέλος η συνέπεια των δεδομένων επιτυγχάνεται μέσω της επιβολής των περιορισμών ακεραιότητας. Σκεπτόμενος πως το blockchain είναι δεδομένα τα οποία συνδέονται μεταξύ τους με έναν συγκεκριμένο τρόπο και μας νοιάζει ο τρόπος συσχέτισης τους, όπως για παράδειγμα τι συναλλαγές έχει κάνει ένας συγκεκριμένος χρήστης του δικτύου, μια κατανεμημένη βάση δεδομένων γράφου αποτελεί ένα ώριμο συμπλήρωμα του blockchain για την αποδοτικότερη ανάλυση των δεδομένων. Βλέπουμε λοιπόν ότι πολλές διαφορετικές τεχνολογίες αναπτύσσονται παράλληλα και υπόσχονται να βελτιώσουν και να αλλάξουν άρδην την μορφή και την ποιότητα των υπηρεσιών του διαδικτύου.

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Στην παρούσα διπλωματική εργασία πρωταγωνιστικό ρόλο θα καταλάβουν οι καινοτόμες τεχνολογίες του blockchain και της βάσης δεδομένων γράφου Neo4j.

Σκοπός της διπλωματικής αποτελεί η διερεύνηση και αξιοποίηση των πλεονεκτημάτων που μας δίνει η Neo4j DB για την αποδοτική εξόρυξη και ανάλυση δεδομένων των χρηστών από την δομή δεδομένων του blockchain, καθώς και για την υλοποίηση νέων καινοτόμων κατανεμημένων πρωτοκόλλων συναίνεσης.

Η καινοτομία έγκειται στο εξής γεγονός : Καταρχάς δεν θα χρησιμοποιηθεί μια ήδη υπάρχουσα πλατφόρμα, αλλά θα κατασκευάσουμε εξ'αρχής το δικό μας blockchain, υλοποιημένο σε γλώσσα προγραμματισμού Python έτσι ώστε να διερευνήσουμε σε βάθος όλα τα mechanics του blockchain. Επιπροσθέτως, λόγω της γραμμικής δομής που έχει το blockchain, ερωτήματα του τύπου "Σε πόσες και ποιες συναλλαγές συμμετέχουν ο User(x) & ο User(y)?" θα απαιτούσαν πολύ

χρόνο και μεγάλη υπολογιστική ισχύ για να απαντηθούν αφού θα έπρεπε να εξεταστεί σειριακά όλο το blockchain. Συγκεκριμένα, θα έπρεπε να γίνει σειριακή αναζήτηση από το πιο πρόσφατο block μέχρι και το genesis block, καθώς και σειρακή αναζήτηση μέσα στα δεδομένα του εκάστοτε block, άρα εκθετικός χρόνος συναρτήσει της αύξησης του αριθμού των blocks. Αυτό λοιπόν, καταφέρνουμε να το επιλύσουμε χρησιμοποιώντας βάση δεδομένων γράφου η οποία μας προσφέρει τεράστια απόδοση σε τέτοιου τύπου ερωτήματα που έχουν να κάνουν με συσχετισμούς και μοτίβα δεδομένων. Η συγκεκριμένη λοιπόν εφαρμογή μέσω της χρήσης βάσης δεδομένων γράφου προσφέρει ένα ευρύ φάσμα ανάλυσης δεδομένων των χρηστών, γεγονός που την διαφοροποιεί από τις κλασικές και ήδη υπάρχουσες blockchain εφαρμογές.

Επιπροσθέτως, μέσω της γρήγορης προσπέλασης που μας προσφέρει η Neo4j στα δεδομένα του blockchain υλοποίησα ένα νέο καινοτόμο κατανεμημένο πρωτόκολλο συναίνεσης το οποίο μπορεί να χρησιμοποιηθεί επιτυχώς σε συνδυασμό με τη δομή δεδομένων του blockchain. Συγκεκριμένα, παρατηρώντας κάποιες αδυναμίες των ήδη υπάρχοντων πρωτοκόλλων όπως το Proof Of Work ή το Proof Of Stake τα οποία τείνουν να κεντρικοποιήσουν τον χαρακτήρα του blockchain, υλοποίησα το Proof Of Motion το οποίο συνδυάζει τα καλύτερα χαρακτηριστικά από τα προαναφερθέντα. Φυσικά πρέπει να τονιστεί και το γεγονός πως η προαναφερθείσα εφαρμογή, δεν χρησιμοποιήθηκε το παραδοσιακό μοντέλο πελάτη – εξπηρετητή, αλλά διατηρήθηκε ο αποκεντρικοποιημένος χαρακτήρας που ορίζει το blockchain, ο οποίος βασίζεται στην αρχιτεκτονική ομότιμων κόμβων (P2P).

Για τον ανωτέρω σκοπό, όπως προαναφέρθηκε υλοποιούμε αρχικά τη δομή δεδομένων του blockchain δίνοντας προσοχή σε όλες τις σημαντικές αρχές που το διέπουν (ακολουθώντας τα βασικά standards του Bitcoin) και παράλληλα υλοποιούμε και την ενσωμάτωση της κατανεμημένης βάσης δεδομένων γράφου Neo4j στην παραπάνω δομή.

1.2 Οργάνωση κειμένου

Το κείμενο της διπλωματικής αποτελείται από 7 Κεφάλαια και 1 Παράρτημα.

Το παρόν Κεφάλαιο, το οποίο αποτελεί την εισαγωγή.

Το Κεφάλαιο 2 περιγράφει το θεωρητικό υπόβαθρο της παρούσας διπλωματικής εργασίας. Πιο συγκεκριμένα, γίνεται σύντομη παρουσίαση

του αποκεντρωμένου διαδικτύου (Web3), το οποίο είναι η πιο σύγχρονη μορφή διαδικτύου και επιτρέπει την ανάπτυξη των αποκεντρωμένων εφαρμογών. Εν συνεχεία παρουσιάζονται τα δίκτυα ομότιμων κόμβων (peer-to-peer networks) και οι κατηγορίες τους, τα οποία αποτελούν θεμέλιο λίθο του blockchain. Επιπροσθέτως παρουσιάζεται το πρόβλημα Διακριτού Λογαρίθμου καθώς και τα βασικά στοιχεία της κρυπτογραφίας των ελλειπτικών καμπυλών, πάνω στην οποία βασίζεται η ασφάλεια του blockchain. Τέλος παρουσιάζονται τα βασικά χαρακτηριστικά του Bitcoin blockchain, στα οποία στηριχτήκαμε ώστε να κατασκευάσουμε το Python Prototype blockchain της εφαρμογής μας, καθώς και οι βασικές αρχές των βάσεων δεδομένων γράφου (εφόσον το δεύτερο συστατικό της εφαρμογής μας είναι η Neo4j).

Στο Κεφάλαιο 3 γίνεται περιγραφή των σημαντικότερων εργαλείων και τεχνολογιών που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής. Αρχικά λοιπόν παρουσιάζεται το περιβάλλον RESTful υπηρεσίας ιστού, η γλώσσα προγραμματισμού Python και το REST framework το οποίο προσφέρει, το οποίο ονομάζεται Python Flask Microframework. Τέλος παρουσιάζεται και η Neo4j βάση δεδομένων γράφου, η οποία χρησιμοποιήθηκε για την απεικόνιση του blockchain σε τριεπίπεδη αρχιτεκτονική.

Στο Κεφάλαιο 4 γίνεται λεπτομερής παρουσίαση της διαδικασίας σχεδίασης και υλοποίησης των επιμέρους συστατικών της εφαρμογής. Αρχικά λοιπόν παρουσιάζεται η μακροσκοπική αρχιτεκτονική του συστήματος η οποία κατά κύριο λόγο περιγράφεται με την ενσωμάτωση του Python blockchain στην Neo4j. Ύστερα, περιγράφεται η βάσικη υλοποίηση της εφαρμογής όπως για παράδειγμα την αρχικοποίηση του δικτύου, την ένταξη των ομότιμων κόμβων σε αυτό, την σύνδεση μεταξύ τους καθώς και βασικές λειτουργίες τους στο δίκτυο blockchain, όπως την δημιουργία συναλλαγών. Ακολούθως, περιγράφεται η υλοποίηση των κατανεμημένων πρωτοκόλλων συναίνεσης με τη : Proof Of Work, Proof Of Stake & Proof Of Motion. Συγκεκριμένα, περιγράφεται η αλγοριθμική σκέψη πίσω από τα πρωτόκολλα καθώς και η υλοποίηση σε Python. Τέλος σε όλα τα παραπάνω παρουσίαζεται και ο κώδικας για την ενσωμάτωση της βάσης δεδομένων γράφου Neo4j.

Στο Κεφάλαιο 5 παρουσιάζεται ο τρόπος λειτουργίας της εφαρμογής όσον αφορά τους χρήστες της. Παρατίθενται εικόνες από διάφορα σενάρια χρήσης της εφαρμογής καθώς και διάφορα αποτελέσματα Cypher ερωτημάτων στην Neo4j.

Το Κεφάλαιο 6 αποτελεί τον επίλογο του κειμένου. Σε αυτό συνοψίζονται οι παρατηρήσεις και τα συμπεράσματα του συγγραφέα όσον αφορά την καταλληλότητα και αποδοτικότητα της εφαρμογής.

Επίσης, σημειώνονται ορισμένες μελλοντικές επεκτάσεις του συστήματος.

Το Κεφάλαιο 7 αποτελείται από την βιβλιογραφία που χρησιμοποιήθηκε για την σύνταξη του κειμένου της διπλωματικής εργασίας και την ανάπτυξη της εφαρμογής.

Τέλος, στο Παράρτημα Ι υπάρχει ο πηγαίος κώδικας της εφαρμογής. Περιλαμβάνεται επίσης και υπερσύνδεσμος για τον υπόλοιπο κώδικα της εφαρμογής, ο οποίος δεν θα ήταν δυνατόν να συμπεριληφθεί στο παρόν κείμενο.

2

Θεωρητικό υπόβαθρο και σχετικές εργασίες

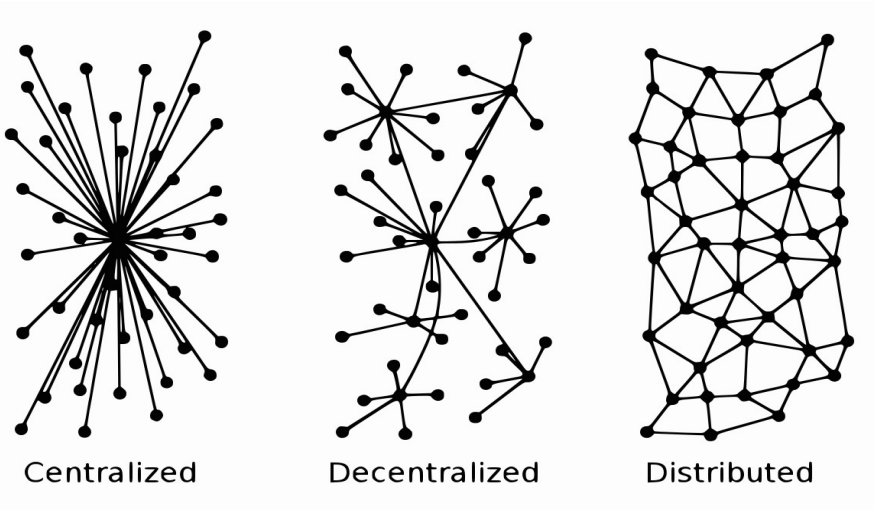
Στο κεφάλαιο αυτό διατυπώνεται το θεωρητικό υπόβαθρο σχετικά με την τεχνολογία του Bitcoin Blockchain καθώς και με την βάση δεδομένων γράφου Neo4j. Αρχικά, παρουσιάζεται η εξέλιξη του παγκόσμιου ιστού εώς την σύγχρονη εποχή καθώς και η τάση που αυτή την στιγμή υπάρχει, η οποία προστάζει την αποκεντρικοποίηση (decentralization) του παγκόσμιου ιστού, έτσι ώστε να υπογραμμστεί η σημασία της τεχνολογίας του Blockchain και γενικότερα των αποκεντρωμένων εφαρμογών. Εν συνεχεία, παρουσιάζονται και ερευνούνται οι βασικές αρχές των ομότιμων δικτύων (Peer-to-Peer networks) τα οποία αποτελούν και το κύριο συστατικό για τα Blockchain. Επιπροσθέτως, παρουσιάζεται συνοπτικά η θεωρία της κρυπτογραφίας των ελλειπτικών καμπυλών εφόσον είναι ο βασικός αλγόριθμος κρυπτογράφησης που χρησιμοποιείται για την ασφάλεια των συναλλαγών στο δίκτυο. Επίσης, γίνεται μια αναλυτική παρουσίαση του Bitcoin Blockchain και των βασικών αρχών του καθώς και των βάσεων δεδομένων γράφου που αποτελούν το σημαντικότερο κομμάτι της παρούσας διπλωματικής. Τέλος, γίνεται αναφορά σε εφαρμογές και εργασίες που είναι σχετικές με την παρούσα διπλωματική εργασία.

2.1 Το αποκεντρωμένο διαδίκτυο

Η αρχική αποστολή του διαδικτύου ήταν να οικοδομήσει ένα κοινό και ουδέτερο δίκτυο με ισότιμη συμμετοχή από τους χρήστες, με σκοπό την βελτίωση της καθημερινότητας των ανθρώπων. Μετά από την πρώτη φούσκα ‘dot com’, πολύ μεγάλες τεχνολογικές εταιρείες όπως η Google, Facebook και άλλες, συνηδειτοποιήσαν πως η μεγαλύτερη αξία που τους έδινε το δίκτυο αυτό ήταν άλλη. Συγκεκριμένα, το ουδέτερο και ‘ισότιμο’ αυτό δίκτυο εν τέλη χρησιμοποιούταν για τη συγκέντρωση, οργάνωση και δημιουργία εσόδων από την διακίνηση πληροφοριών μέσω κεντρικού διακομιστή. Για αυτόν τον λόγο οι τεχνολογικές εταιρείες οι οποίες

λειτουργούν σαν μεσάζοντες (third parties), δημιουργούν την αξία τους με την ανάπτυξη τεράστιων συγκεντρωτικών υπηρεσιών στο διαδίκτυο, ενώ απενεργοποιώντας ταυτόχρονα την δυνατότητα σύνδεσης με περιεχόμενο μέσω μιας διεύθυνσης URL καταφέρνουν και αποκωδικοποιούν αλγοριθμικά προσωπικές προτιμήσεις περιεχόμενου των τελικών χρηστών. Ουσιαστικά το βασικό χαρακτηριστικό του κλασικού διαδικτύου ήταν η κεντρικοποίηση, δηλαδή η ύπαρξη μιας κεντρικής οντότητας η οποία έχει τον πλήρη έλεγχο των δεδομένων των χρηστών μέσω κέντρου δεδομένων [4].

Τον παραπάνω περιορισμό έρχεται να λύσει το αποκεντρωμένο διαδίκτυο (Web3)[5] και κατά συνέπεια και το Blockchain. Διαπιστώνουμε πως διαμορφώνεται μια τάση, η οποία επιτάσσει την μείωση της χρήσης κεντρικών εξυπηρετητών, χάριν της υιοθέτησης ενός αποκεντρωμένου μοντέλου διαδικτύου. Συγκεκριμένα, η αποκέντρωση του Διαδικτύου δημιουργεί έναν κόσμο όπου βασικές καθημερινές υπηρεσίες όπως για παράδειγμα η επικοινωνία, οι τραπεζικές συναλλαγές, social networking, δεν παρέχονται από κεντρικές υπηρεσίες που ανήκουν σε μεμονωμένες οργανώσεις η ανθρώπους. Υπό αυτή την έννοια το διαδίκτυο γίνεται μια <<πραγματική δημοκρατία>> όπου οι ίδιοι οι χρήστες κατέχουν και ελέγχουν τον τρόπο με τον οποίο χρησιμοποιούνται και μοιράζονται τα ασφαλώς κρυπτογραφημένα δεδομένα τους. Τα εκάστοτε κρυπτογραφημένα δεδομένα προσπελάζονται μέσω κρυπτογραφίας δημοσίου κλειδού, διατηρώντας τα έτσι ασφαλή και ανώνυμα μέχρις ότου ο χρήστης αποφασίσει να τα "απελευθερώσει".



Εικόνα 2.1 Centralized Internet vs Decentralized vs Distributed

Το blockchain αποτελεί βασικό παράγοντα[6] στην ανάπτυξη του αποκεντρωμένου διαδικτύου. Παρόλα αυτά δεν ενδύκνειται για την αποθήκευση μεγάλων ποσοτήτων δεδομένων για δύο λόγους : Ο πρώτος λόγος σχετίζεται με την δυσκολία επεκτασιμότητας του καθώς και με το ότι είναι πάρα πολύ αργό όταν το διατρέχουμε. Ο δεύτερος λόγος είναι η ιδιωτικότητα καθώς όλη η πληροφορία είναι αποθηκευμένη στο blockchain και ορατή σε όλους τους χρήστες. Για αυτούς λοιπόν τους παραπάνω λόγους διερευνούμε και υλοποιούμε το integration του blockchain με μια κατανεμημένη βάση δεδομένων γράφου Neo4j.

Στα επόμενα χρόνια υπάρχει η πεποίθηση ότι τα τρία στοιχεία της υπολογιστικής τα οποία είναι: Επικοινωνία (Communication), Επεξεργασία (Processing), Αποθήκευση (Storage), θα είναι εντελώς αποκεντρωμένα.

2.2 Δίκτυα Ομότιμων Κόμβων

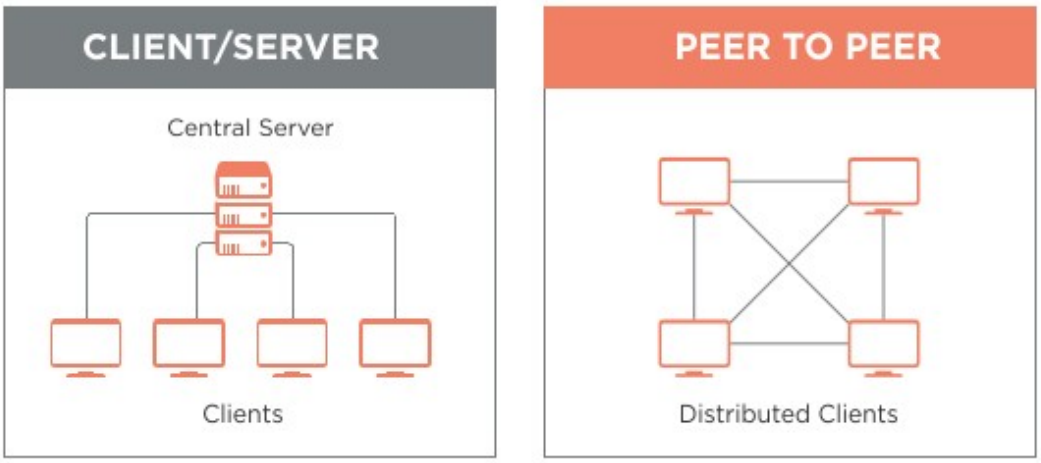
Στη συγκεκριμένη ενότητα θα παρουσιάσουμε την την αρχιτεκτονική των δικτύων ομότιμων κόμβων (peer to peer networks), πάνω στα οποία βασίζεται η τεχνολογία του blockchain.

Σήμερα, οι κόμβοι σε ένα κατανεμημένο σύστημα οργανώνονται σύμφωνα είτε με το μοντέλο πελάτη-εξυπηρετητή (client-server model) [7] ή με δίκτυα ομότιμων κόμβων[8], τα οποία μπορεί να είναι δομημένα ή αδόμητα. Στην περίπτωση του μοντέλου πελάτη εξυπηρετητή υπάρχει πάντα ένας ενεργός υπολογιστής ο οποίος είναι υπεύθυνος να παρέχει πόρους ή υπηρεσίες στους πελάτες. Συνεπαγωγικά, οι πελάτες δεν έχουν καμία επικοινωνία μεταξύ τους αλλά όλοι εξαρτώνται από τον κεντρικό εξυπηρετητή. Το παραπάνω μοντέλο έχει τα πλεονεκτήματα της εύκολης υλοποίησης και διαχείρισης αλλά και πληθώρα μειονεκτημάτων με κυριότερα από αυτά: Μοναδικό Σημείο Αποτυχίας (Single Point of Failure – SPOF) [9] καθώς και δύσκολη κλιμάκωση.

Παρόλα αυτά τα τελευταία χρόνια με την ραγδαία εξάπλωση του Internet δημιουργήθηκε μια ακόμη ανάγκη. Η ανάγκη για κοινή χρήση υπολογιστικών πόρων (αποθήκευση, υπολογιστική ισχύ, κλπ.). Για αυτόν τον λόγο συγκροτούνται δίκτυα από πολλούς υπολογιστές με σκοπό την ανταλλαγή αρχείων ή για κατανεμημένο υπολογισμό ή και για παροχή

καταναμεμημένων υπηρεσιών. Αυτά τα δίκτυα είναι γνωστά ως δίκτυα ομότιμων κόμβων (P2P networks).

Σε ένα τέτοιο δίκτυο κάθε κόμβος ο οποίος συμμετέχει είναι ισότιμος με κάθε άλλο και μπορεί να ενεργήσει είτε σαν πελάτης (client) ή σαν εξυπηρετητής (server). Συγκεκριμένα, οι μετέχοντες κόμβοι προσαρμόζονται και αυτοδιαργανώνονται καθώς εισέρχονται ή αποχωρούν από το σύστημα, ικανοποιώντας την ιδιότητα της κλιμάκωσης και της ανοχής στις αποτυχίες (όπως στην προαναφερθείσα αποτυχία Single Point Of Failure). Οι λειτουργίες του είναι κατανεμημένες στους κόμβους που μετέχουν σε ένα τέτοιο σύστημα, όπου εκατομμύρια διαφορετικοί χρήστες μπορούν να είναι συνδεδεμένοι ταυτόχρονα.



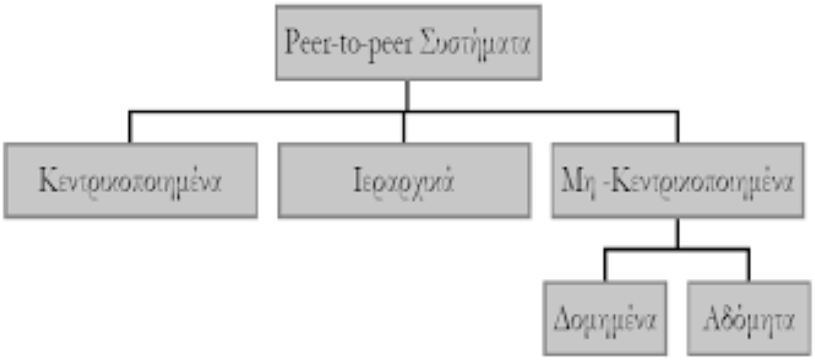
Εικόνα 2.2 Αρχιτεκτονική πελάτη εξυπηρετητή & Αρχιτεκτονική ομότιμων κόμβων

Peer-to-Peer συστήματα είναι κατανεμημένα συστήματα που αποτελούνται από διασυνδεδεμένους κόμβους, ικανούς να αυτοδιαργανώνται σε τοπολογίες δικτύου με σκοπό την κοινή χρήση πόρων όπως περιεχόμενα, κύκλους μηχανής, χώρο αποθήκευσης και εύρος, ικανά να προσαρμόζονται στις αποτυχίες και στις παροδικές μετακινήσεις κόμβων ενώ διατηρούν προσβάσιμη συνδετικότητα και εκτελούνται χωρίς την απαίτηση για μεσολάβηση ή υποστήριξη ενός καθολικού κεντρικού εξυπηρετητή.

Παραδείγματα εφαρμογών που βασίζονται σε αρχιτεκτονικές ομότιμων κόμβων είναι περιλαμβάνουν μεταξύ άλλων κατηγορίες όπως διανομή αρχείων (π.χ BitTorrent [10]), επιτάχυνση κατεβάσματος

αρχείων μέσω βοήθειας ομότιμων κόμβων (π.χ Xunlei) και τηλεφωνία διαδικτύου (π.χ Skype). Το blockchain αποτελεί και αυτό μια τεχνολογία η οποία βασίζεται στην αρχιτεκτονική ομότιμων κόμβων.

Υπάρχουν διάφορες αρχιτεκτονικές για τον σχηματισμό του δικτύου ομότιμων κόμβων (Εικόνα 2.3):



Εικόνα 2.3 Peer-to-Peer Αρχιτεκτονικές

2.2.1 Πλεονεκτήματα P2P δικτύων

Τα βασικότερα πλεονεκτήματα της αρχιτεκτονικής των ομότιμων κόμβων είναι τα εξής:

- *Μείωση Κόστους,* καθώς δεν απαιτείται σημαντική υποδομή και εύρος ζώνης εξυπηρετητή.
- *Μη ύπαρξη μοναδικού σημείου αστοχίας του δικτύου(No SPOF),* αφού η βλάβη σε έναν μεμωνομένο κόμβο δεν επηρεάζει την ομαλή λειτουργία του υπόλοιπου δικτύου όπως γίνεται στο μοντέλο πελάτη-εξυπηρετητή.

- *Αυτοκλιμακωσιμότητα του δικτύου,* εφόσον όσο περισσότεροι κόμβοι προστίθενται στο δίκτυο, τόσο αυξάνεται ο φόρτος εργασίας, η απαίτηση σε πόρους δηλαδή, όμως τόσο αυξάνονται και οι διαθέσιμοι πόροι λόγω της διττής φύσης του κάθε κόμβου.

2.3 Κρυπτογραφία ελλειπτικών καμπυλών

Η κρυπτογραφία ελλειπτικών καμπυλών [11] είναι μια σχετικά νέα επιστήμη η οποία συμβάλλει στην προσπάθεια για παγκόσμια ασφάλεια και είναι η βάση της ασφάλειας στην τεχνολογία του blockchain. Η συγκεκριμένη θεωρία προτάθηκε πρώτη φορά το 1985 από τους Victor Miller (IBM) και Neil Koblitz (University of Washington) ως εναλλακτικός μηχανισμός για την υλοποίηση της κρυπτογραφίας δημοσίου κλειδιού [12]. Τα κρυπτοσυστήματα που είναι βασισμένα στις ελλειπτικές καμπύλες βασίζονται στο πρόβλημα του διακριτού λογαρίθμου και η βασική αρχή τους είναι η εξής: Χρησιμοποιούν κάποια πεπερασμένα σώματα που έχουν τάξη κάποιον πρώτο αριθμό και βρίσκονται κάτω από μια ελλειπτική καμπύλη.

Οι ελλειπτικές καμπύλες μας δίνουν την δυνατότητα να ασφαλίσουμε την τεχνολογία μας αφού δεν υπάρχει καμία γνωστή και αποδοτική μέθοδος επίθεσης στο συγκεκριμένο κρυπτοσύστημα. Αυτό έχει ως αποτέλεσμα να εξασφαλίζεται με τη χρήση τους το ίδιο επίπεδο ασφάλειας χρησιμοποιώντας σώματα αρκετά μικρότερα σε σχέση με τα σώματα που χρησιμοποιούμε στα συμβατικά κρυπτοσυστήματα.

Η επικοινωνία μέσω δημοσίων καναλιών κάνει την ανάγκη για χρήση της κρυπτογραφίας επιτακτική. Η ασφάλεια των ψηφιακών υπογραφών, των αλγορίθμων ασφαλούς ανταλαγής κλειδιών και τα κρυπτοσυστήματα δημοσίου κλειδιού μέσω ελλειπτικών καμπυλών βασίζονται στη δυσκολία εύρεσης του διακριτού λογαρίθμου, που είναι εκτελέσιμος σε υποεκθετικό χρόνο.

Σήμερα θεωρείται ότι ο αλγόριθμος ψηφιακής υπογραφής ελλειπτικών καμπυλών (Elliptic Curve Digital Signature Algorithm – ECDSA) [13], που χρησιμοποιούν τα περισσότερα κρυπτονομίσματα για κρυπτογράφηση όπως τα Bitcoin & Ethereum, είναι απαραβίαστος. Με αυτόν τον τρόπο η ασφάλεια των συναλλαγών που εκτελούνται στο blockchain θεωρούνται ασφαλείς και γενικότερα η συνολική πλατφόρμα είναι προστατευμένη απέναντι σε επιθέσεις. Παρόλα αυτά σε περίπτωση που υπάρξει μεγάλη πρόοδος στην κβαντική υπολογιστική (Quantum

Computing) τότε ο αλγόριθμος αυτός θα σταματήσει να είναι ασφαλής αφού η υπολογιστική ισχύς που θα παρέχεται θα είναι τεράστια.

```
// *** ECDSA Algorithm Pseudocode *** //

/* Entity A has domain parameters
* D = (q,a,b,G,n,h) and public key QA and
* private key dA. Entity B has authentic
* copies of D and QA */

To sign a message m, A does the following:
Step1: Select a random integer k from [1,n-1]
Step2: Compute kG=(x1,y1) and r=x1(modn). If r=0 then
       go to Step1.
Step3: Compute  k^-1 mod n. Compute e=SHA-1(m)
Step4: Compute s= k^-1 {e + dA * r} mod n
If s=0 then go to Step1.

//A's signature for the message m is (r,s)
```

2.3.1 Ο Διακριτός Λογάριθμος

Έστω p πρώτος αριθμός και α,b ακέραιοι αριθμοί οι οποίοι δεν διαιρούνται με το p. Έστω ότι ∃ k με $k \in Z$ τέτοιος ώστε $a^k \equiv b(modp)$. Το πρόβλημα του διακριτού λογαρίθμου είναι να βρούμε το k. Πιο γενικά έστω G μια ομάδα και έστω ότι $a,b \in G$ και ταυτόχρονα κάνουμε την παραδοχή ότι υπάρχει $k \in Z$ τέτοιο ώστε $a^k = b$. Σημειώνεται πως και σε αυτή την περίπτωση το πρόβλημα του διακριτού λογαρίθμου είναι να βρούμε το k. Οπότε η G θα μπορούσε να είναι και η $E(F_q)$ για κάποια ελλειπτική καμπύλη. Σε αυτή λοιπόν την περίπτωση τα α και b επαληθεύουν την εξίσωση της ελλειπτικής καμπύλης αφού είναι σημεία πάνω της και εν συνεχεία προσπαθώ να βρω έναν ακέραιο k τέτοιο ώστε kα = b. Όπως προαναφέρθηκε η ασφάλεια πολλών κρυπτοσυστημάτων (βλέπε Bitcoin, Ethereum κα.) βασίζονται στη δυσκολία του προβλήματος του διακριτού λογαρίθμου.

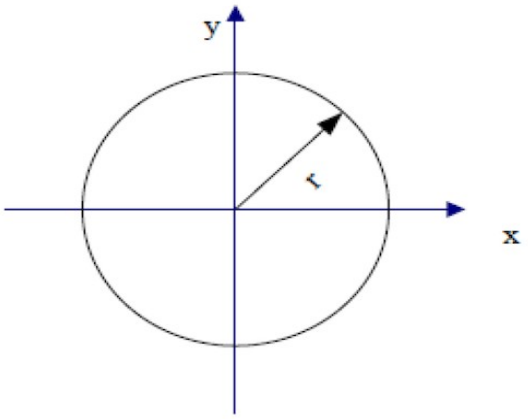
2.3.2 Ελλειπτικές Καμπύλες στο σώμα των πραγματικών αριθμών

Αντίθετα από την αίσθηση που μας δημιουργεί ο όρος καμπύλη, μια ελλειπτική καμπύλη μπορεί να αποτελείται στην πραγματικότητα από δύο καμπύλες και ένα σημείο που βρίσκεται εκτός καμπυλών.

Ξεκινώντας με το βασικό θεωρητικό υπόβαθρο έχω ότι ένας κύκλος με κέντρο το O(0,0) ορίζεται από την εξίσωση:

$x^2 + y^2 = r^2$, όπου ρ είναι η ακτίνα του κύκλου.

Απεικονίζοντας λοιπόν όλα τα σημεία (x,y) ενός επιπέδου που ικανοποιούν την παραπάνω εξίσωση προκύπτει ο παρακάτω κυκλικός γεωμετρικός τόπος:



Εικόνα 2.6 Γεωμετρικός τόπος Εξίσωσης Κύκλου

Όμως γνωρίζω ότι ο κύκλος είναι ειδική περίπτωση της έλλειψης, όπου η εξίσωση είναι:

$ax^2 + by^2 = c$

Παρατηρώ ότι τόσο στην έλλειψης όσο και στον κύκλου οι μεταβλητές x,y συνδέονται με δευτεροβάθμιες εξισώσεις, με αποτέλεσμα για μια συγκεκριμένη τιμή του x να αντστοιχίζονται δύο τιμές του y και αντίστροφα.

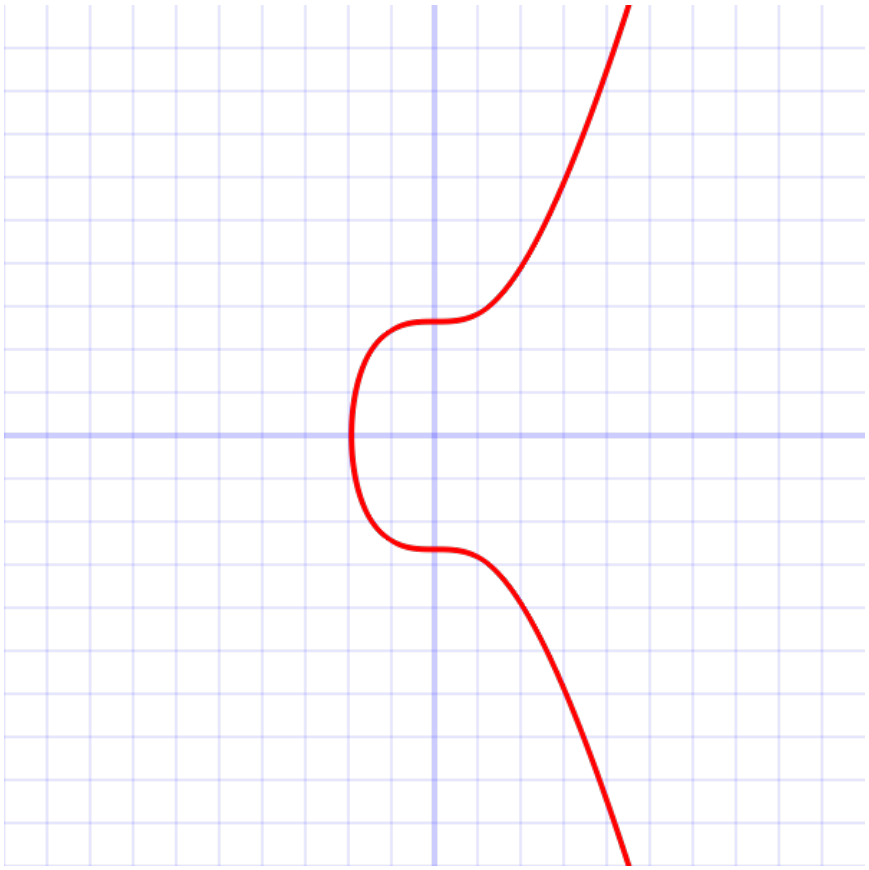
Στην περίπτωση όμως της ελλειπτικής καμπύλης, η εξίσωση της καμπύλης είναι δευτεροβάθμια ως προς y, αλλά τριτοβάθμια ως προς x. Η εξίσωση της ελλειπτικής καμπύλης δίνεται λοιπόν από την εξίσωση:

$y^2 = x^3 + ax + b$, για σταθερές α και b.

Σημειώνεται πως η ελλειπτική καμπύλη η οποία χρησιμοποιείται στο Bitcoin είναι η *Secp256k1 [14]* και συγκεκριμένα η εξίσωση της είναι:

$y^2 = x^3 + 7$, δηλαδή α=0 και b=7.

Παρακάτω παρουσιάζεται η γραφική παράσταση της συγκεκριμένης καμπύλης:



Εικόνα 2.7 Γραφική Παράσταση της ελλειπτικής καμπύλης Secp256k1

2.4 Bitcoin Blockchain

Ερμηνεία: Το **blockchain** είναι ένα ψηφιακό, κατανεμημένο, δημόσιο καθολικό (ή λογιστικό βιβλίο – ledger), μέσω του οποίου καταγράφονται με ασφάλεια συναλλαγές, συμφωνίες, συμβόλαια και γενικώς οτιδήποτε χρειάζεται να έχει καταγραφεί και να είναι διαθέσιμο προς επαλήθευση.

Ερμηνεία: **Κατανεμημένο σύστημα** είναι μια συλλογή από ανεξάρτητους υπολογιστές, οι οποίοι εμφανίζονται στους χρήστες τους ως ένα ενιαίο συνεκτικό σύστημα.

Ερμηνεία: **Κρυπτονόμισμα** είναι ένα ψηφιακό νόμισμα το οποίο ασφαλίζει τις συναλλαγές με κρυπτογραφικό κώδικα, που βασίζεται στην υπολογιστική ισχύ του hardware για την εκτέλεση του (proof of work) ή λιγότερο ενεργειακά απαιτητικούς τρόπους, όπως το proof of stake.

2.4.1 Το κρυπτονόμισμα Bitcoin

Το ψηφιακό νόμισμα bitcoin είναι μια συλλογή τεχνολογιών η οποία αποτελεί την βάση για ένα ψηφιακό χρηματικό οικοσύστημα. Μονάδες του νομίσματος που ονομάζονται Bitcoins χρησιμοποιούνται για την αποθήκευση και τη μετάδοση αξίας μεταξύ των συμμετεχόντων του δικτύου Bitcoin. Σημειώνεται πως η στοίβα πρωτοκόλλου Bitcoin, το διαθέσιμο λογισμικό ανοιχτού κώδικα μπορεί να τρέξει σε ένα ευρύ φάσμα των υπολογιστικών συσκευών, συμπεριλαμβανομένων των φορητών υπολογιστών και smartphones καθιστώντας έτσι την συγκεκριμένη τεχνολογία εύκολα προσβάσιμη από το ευρύ κοινό. Οι χρήστες οι οποίοι συμμετέχουν στο δίκτυο μπορούν να πραγματοποιούν συναλλαγές με τα συγκεκριμένα κρυπτονομίσματα, όπως ακριβώς κάνουν με το συμβατικό χρήμα, καθώς τα Bitcoin μπορούν να αγοραστούν, να πωληθούν και να αναταλλαχθούν με άλλα νομίσματα σε εξειδικευμένες συναλλαγματικές ισοτιμίες.

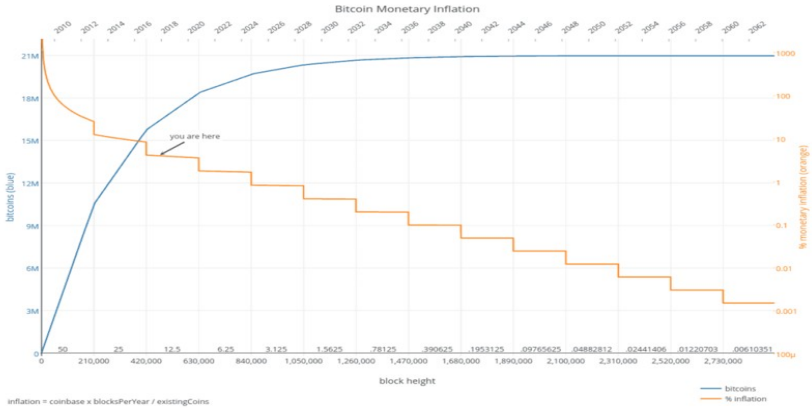
Το Bitcoin λοιπόν είναι ένα νόμισμα σε ένα σύστημα ηλεκτρονικών συναλλαγών που βασίζεται στα πρωτόκολλα της κρυπτογράφησης και εμπεριέχει ψηφιακές υπογραφές για να δίνει τον πλήρη έλεγχο της ιδιοκτησίας. Οι συναλλαγές γίνονται μεταξύ των δύο μερών σε ένα δίκτυα ομότιμων κόμβων, χωρίς να εμπλέκονται τρίτα χρηματοπιστωτικά ιδρύματα για τον έλεγχο και πιστοποίηση της συναλλαγής. Ο ρόλος των μεσαζόντων αντικαθίσταται από τους χρήστες του δικτύου που μοιράζονται τους πόρους των υπολογιστών τους, απ’όπου παράγονται τα bitcoins (εξόρυξη/mining)[15] και μπορούν να αποθηκευτούν ψηφιακά. Εφόσον επαληθευτούν και καταγραφούν οι συναλλαγές στην αλυσίδα των μπλοκ (blockchain) που περιέχουν όλες τις συναλλαγές που έχουν προηγηθεί και ακολουθούν το κάθε νόμισμα, γίνονται αποδεκτές από το εκάστοτε μέλος του δικτύου.

Οι χρήστες του Bitcoin έχουν στην κατοχή τους δικά τους κρυπτογραφημένα κλειδιά με τα οποία αποδυκνείουν την κυριότητα των συναλλαγών και τις υπογράφουν με τις επονομαζόμενες ψηφιακές υπογραφές (digital signatures). Συγκεκριμένα, οι χρήστες έχουν ένα δημόσιο κλειδί (public key) το οποίο είναι ορατό και γνωστό σε όλους

τους συμμετέχοντες στο δίκτυο Blockchain καθώς και ένα ιδιωτικό κλειδί (private key) το οποίο είναι κρυφό και αποθηκεύεται στο ψηφιακό πορτοφόλι του κάθε χρήστη (χρησιμοποιείται για την ψηφιακή υπογραφή).

Τα Bitcoin δημιουργούνται και αποκτούνται μέσω μιας διαδικασίας που ονομάζεται εξόρυξη (Mining) η οποία ορίζει την εξής λειτουργία: Οποιοσδήποτε χρήστης ο οποίος είναι διατεθειμένος να θυσιάσει ένα μέρος από την υπολογιστική του ισχύ, μπαίνει στην διαδικασία ανταγωνισμού λύσης ενός μαθηματικού παζλ έτσι ώστε να επαληθεύσει τις συναλλαγές οι οποίες είναι αποθηκευμένες σε ένα μέρος της μνήμης του που ονομάζεται transaction pool. Περισσότερα και πιο συγκεκριμένα θα αναλυθεί η διαδικασία της εξόρυξης παρακάτω.

Τέλος, το πρωτόκολλο διαιρεί στα δύο (halving process)[16] τον ρυθμό με τον οποίο νέα bitcoins δημιουργούνται κάθε τέσσερα χρόνια και περιορίζει τον συνολικό αριθμό των bitcoins που θα δημιουργηθούν σε ένα σταθερό σύνολο των 21 εκατομμυρίων "κερμάτων". Συνέπεια αυτού, είναι ότι ο αριθμός των bitcoins σε κυκλοφορία ακολουθεί πιστά μια εύκολα προβλέψιμη καμπύλη που φτάνει τα 21 εκατομμύρια μέχρι το έτος 2140. Φυσικά, δεν είναι δυνατό να εκτυπωθεί νέο χρήμα πέρα και πάνω από τον αναμενόμενο ρυθμό έκδοσης.

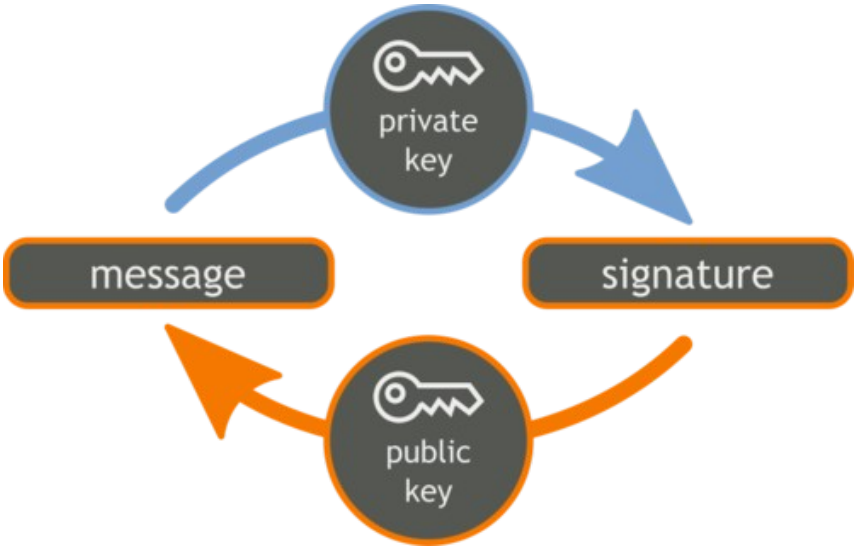


Εικόνα 2.8 Γράφημα Bitcoin Halving Production (Πηγή: https://rados.io)

2.4.2 Ψηφιακή Υπογραφή

Η ψηφιακή υπογραφή χρησιμοποιείται στο σύστημα Bitcoin για την αποφυγή κακόβουλων συναλλαγών, όπου το προανεφερθέν ζεύγος δημοσίου και ιδιωτικού κλειδιού βοηθάει με την κρυπτογράφηση και αποκρυπτογράφηση του απεσταλμένου μηνύματος, δημιουργώντας ψηφιακή υπογραφή κάποιου που έστειλε το μήνυμα.

Συγκεκριμένα έστω ότι θέλουμε να στείλουμε ένα υπογεγραμμένο μήνυμα M. Τότε αυτό κατακερματίζεται σε HM και εν συνεχεία κρυπτογραφείται με το ιδιωτικό κλειδί Kprivate ώστε να δημιουργηθεί η υπογραφή S=encrypt(H,Kprivate), η οποία στέλνεται με το HM στον παραλήπτη. Εκείνος, την στιγμή που λαμβάνει το HM με τη χρήση του δημοσίου κλειδιού Kpublic το αποκρυπτογραφεί H'=decrypt(S,Kpublic) και έτσι συγκρίνοντας τα αποτελέσματα H'=H διαπιστώνει την εγκυρότητα του. Ουσιαστικά αυτός ο τρόπος είναι σε πιο υψηλό επίπεδο ο Elliptic Curve Digital Signature Algorithm που περιγράφηκε στο Κεφ. 2.3.



Εικόνα 2.9 Δημιουργία και Έλεγχος Ψηφιακής Υπογραφής

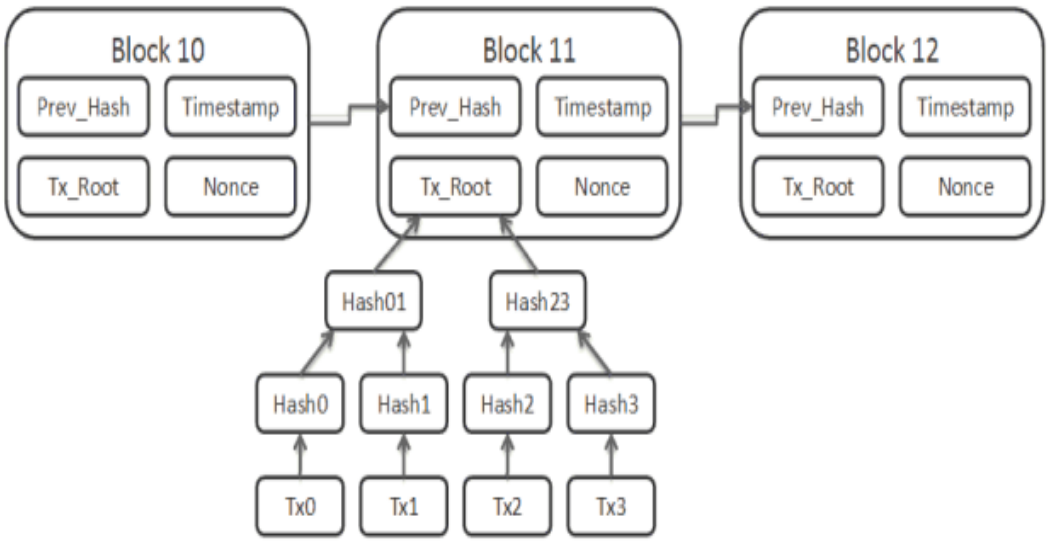
2.4.3 Η τεχνολογία του Bitcoin Blockchain

Μια αλυσίδα μπλοκ (blockchain) είναι μια βάση δεδομένων συναλλαγών η οποία είναι κοινόχρηστη σε όλους τους κόμβους που συμμετέχουν σε ένα σύστημα με βάση το πρωτόκολλο του Bitcoin. Το blockchain λειτουργεί στο διαδίκτυο, πάνω σε ένα δίκτυο ομότιμων κόμβων που τρέχουν το πρωτόκολλο και διατηρούν ένα πιστό αντίγραφο όλων των συναλλαγών/δεδομένων, επιτρέποντας την πραγματοποίηση συναλλαγών χωρίς την παρουσία κάποιας ενδιάμεσης αρχής (Trusted Third Party TTP) παρά μόνο με την βοήθεια των υπολοίπων κόμβων του δικτύου.

Κάθε μπλοκ έχει έναν κατακερματισμένο δείκτη στο προηγούμενο μπλοκ (hash pointer), έχοντας σαν αποτέλεσμα την ασφαλή δημιουργία μιας αλυσίδας από μπλοκ από το εναρκτήριο μπλοκ (genesis block)[17] μέχρι το τρέχον. Επιπροσθέτως, μέσω των hash pointers διασφαλίζεται πως κάθε μπλοκ θα έρθει χρονολογικά μετά από προηγούμενο διότι

διαφορετικά ο κατακερματισμός των δεδομένων δεν θα γίνει σωστά. Με το blockchain παρέχεται ένα δημόσιο καθολικό ημερολόγιο συναλλαγών το οποίο διαθέτει ταξινομημένη και χρονικά χωρισμένη καταγραφή των συναλλαγών, όπου με τις δικλείδες ασφαλείας που παρέχει δεν επιτρέπει την τροποποίηση και διαγραφή οποιασδήποτε καταχωρημένης πληροφορίας.

Φυσικά σε κάθε μπλοκ αποθηκεύονται συναλλαγές στο τμήμα δεδομένων του, οι οποίες είναι επικυρωμένες από το δίκτυο P2P. Η παρακάτω εικόνα δείχνει μια απλοποιημένη εκδοχή του blockchain:

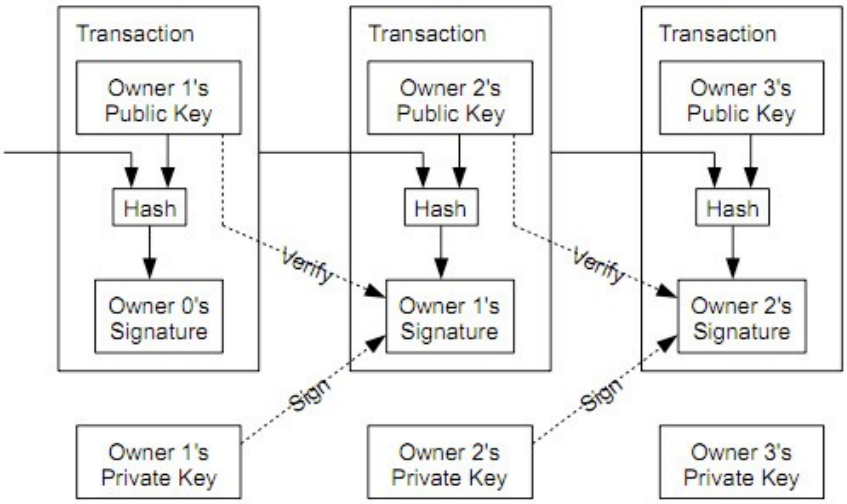


Εικόνα 2.10 Απλοποιημένη Εκδοχή του blockchain

2.4.4 Σύντομη Περιγραφή του Proof Of Work πρωτοκόλλου

Όπως έχουμε ήδη αναφέρει το blockchain είναι ο θεμέλιος λίθος του Bitcoin και η συναλλαγή είναι η βασική μονάδα του. Κάθε νέο μπλοκ που δημιουργείται περιέχει τις συναλλαγές οι οποίες ελέγχθησαν για την ορθότητα τους και βρέθηκαν σωστές. Το μέγεθος κάθε μπλοκ στο πρωτόκολλο του Bitcoin έχει οριστεί στο 1MB και το πλήθος των συναλλαγών που μπορούν να μπουν μέσα σε αυτόν τον χώρο εξαρτάται από το μέγεθος της κάθε συναλλαγής.

Όσον αφορά τις συναλλαγές έχουμε τα εξής: Κάθε συναλλαγή έχει τουλάχιστον μια είσοδο και μια έξοδο. Κάθε είσοδος καταναλώνει τα bitcoins τα οποία παράγει η προηγούμενη έξοδος. Κάθε έξοδος λειτουργεί ως μια έξοδος μη χρησιμοποιούμενων δαπανών (Unspent Transaction Output – UTXO) [18] εώς ότου τα δαπανήσει μια μεταγενέστερη είσοδος. Η παρακάτω εικόνα παρουσιάζει αναλυτικά τον τρόπο με τον οποίο πραγματοποιείται μια συναλλαγή στο Bitcoin δίκτυο:



Εικόνα 2.11 Διαδικασία πραγματοποίησης συναλλαγής στο δίκτυο Bitcoin

Σε κάθε συναλλαγή η οποία πραγματοποιείται πρέπει να συμπεριληφθεί και ένα μικρό τέλος/φόρος (transaction fee) [19]. Οι ανωτέρω φόροι εισπράττονται από τους κόμβους που επικυρώνουν το δίκτυο, τους miners. Οι miners είναι κόμβοι του δικτύου που λαμβάνουν, διαδίδουν, επικυρώνουν και εκτελούν συναλλαγές. Συγκεντρώνουν ορισμένες συναλλαγές κάθε φορά σε μπλοκ και στη συνέχεια ανταγωνίζονται τους υπόλοιπους miners ώστε να είναι αυτοί που θα δημοσιεύσουν και θα εισάγουν το μπλοκ τους στο blockchain. Κάθε φορά που ένας miner καταφέρνει και εισάγει το δικό του μπλοκ στο blockchain πέρα από τα transction fees λαμβάνει και 12.5 bitcoins σαν reward. Αυτό το ποσό λειτουργεί σαν κίνητρο στους miners ώστε να εκτελούν την εργασία αυτή. Σημειώνεται πως το block mining reward υποδιπλασιάζεται κάθε 210.000 blocks σύμφωνα με το πρωτόκολλο του Bitcoin.

Εικόνα 2.12 Χρονομετρητής Υποδιπλασιασμού Επιβράβευσης των Miners

(Πηγή: https://www.bitcoinblockhalf.com)

Για να εισαχθεί ένα νέο μπλοκ στο blockchain θα πρέπει να συνοδεύεται από την απόδειξη εργασίας (nonce). Αυτό ουσιαστικά αποτελεί την απόδειξη λύσης ενός μαθηματικού κρυπτογραφικού παζλ το οποίο είναι το εξής:

`hash(tx1|tx2|tx3|...|txn|nonce) < target`

Ουσιαστικά, ο κάθε miner ψάχνει τον αριθμό/nonce που όταν συνενωθεί με όλες τις συναλλαγές που έχει στο μπλοκ προς δημοσίευση και εν συνεχεία όλο αυτό το δημιουργηθέν λεκτικό κατακερματιστεί θα δώσει έναν αριθμό ο οποίος είναι μικρότερος από έναν αριθμό στόχο που έχει οριστεί από το πρωτόκολλο. Ο πρώτος miner που θα καταφέρει να βρει τέτοιο nonce δημοσιεύει το νέο μπλοκ στο blockchain και λαμβάνει και το αντίστοιχο reward μαζί με τα transaction fees.

Αυτή η εργασία απαιτεί μεγάλη υπολογιστική ισχύ και συγκεκριμένα απαιτεί ειδικού hardware, όπως για παράδειγμα εξελιγμένων καρτών γραφικών. Η εργασία αυτή αποτελεί και το κατανεμημένο πρωτόκολλο συναίνεσης που χρησιμοποιεί το πρωτόκολλο του bitcoin και ονομάζεται ***Proof Of Work – PoW [20]***.

Εναλλακτικό πρωτόκολλο συναίνεσης αποτελεί το Proof Of Stake – PoS το οποίο επιλέγει τον νικήτη κόμβο με βάση τον αριθμό των tokens που έχει στην κατοχή του. Θα παρουσιαστεί αναλυτικά πιο κάτω.

2.5 Εισαγωγή στις Βάσεις Δεδομένων Γράφου

2.5.1 Η προσέγγιση NoSQL

Η προσέγγιση NoSQL (Not Only SQL) [21] αναφέρεται σε μια τάξη συστημάτων διαχείρισης βάσεων δεδομένων τα οποία δεν ακολουθούν τους κανόνες της σχεσιακής σχεδίασης και δεν χρησιμοποιούν την SQL ως γλώσσα ερωτημάτων. Τα συγκεκριμένα συστήματα χρησιμοποιούνται σε περιπτώσεις όπου ο όγκος των δεδομένων είναι πολύ μεγάλος (όπως για παράδειγμα στην περίπτωση μας, δηλαδή στην τεχνολογία blockchain) και παρουσιάζονται προβλήματα απόδοσης λόγω της SQL και της σχεσιακού μοντέλου σχεδίασης. Η κύρια διαφοροποίηση μεταξύ σχεσιακών και NoSQL βάσεων είναι η εξής σημαντική: Ενώ οι σχεσιακές βάσεις εφαρμόζουν lock σε δεδομένα για την εξασφάλιση της συνέπειας τους, με τις αντίστοιχες φυσικά καθυστερήσεις στην απόδοση τους, οι NoSQL βάσεις προσπαθούν να διασφαλίσουν μόνο τη συνέπεια μεταξύ των γεγονότων εφαρμόζοντας τις αλλαγές που προκύπτουν περιοδικά ή σε στιγμές που το σύστημα δεν δέχεται ερωτήματα.

Χαρακτηριστικό των κλασικών σχεσιακών βάσεων δεδομένων είναι η συμμόρφωση στους κανόνες ACID: Atomicity, Consistency, Integrity, Durability) [22]. Οι βάσεις NoSQL θυσιάζουν την συμμόρφωση στα παραπάνω κριτήρια, με στόχο την επίτευξη των υψηλών επιδόσεων. Τα δεδομένα τα οποία αποθηκεύονται σε αυτού του τύπου βάσεις ποικίλλουν από απλά ζευγάρια κλειδιού-τιμής σε έγγραφα και επεκτάσιμες εγγραφές. Συνήθως όλες οι βάσεις NoSQL χαλαρώνουν την απαίτηση για ένα ή περισσότερα κριτήρια ACID και για αυτόν τον λόγο ισχύουν πλέον τα κριτήρια BASE τα οποία είναι τα εξής:

1. Basically Available
2. Soft State
3. Eventually Consistent

Η προσέγγιση BASE δέχεται ότι η συνέπεια της βάσης δεδομένων θα είναι σε ρευστή κατάσταση. Η διαθεσιμότητα της βάσης επιτυγχάνεται με την υποστήριξη επιμέρους αποτυχιών και χωρίς πλήρη αποτυχία του συστήματος.

Επιπροσθέτως ένα άλλο πολύ σημαντικό χαρακτηριστικό της NoSQL προσέγγισης είναι η κατανεμημένη αρχιτεκτονική των συστημάτων βάσεων δεδομένων. Τα δεδομένα αποθηκεύονται με πλεοναστικό τρόπο σε πολλαπλούς servers με σκοπό την αντιμετώπιση της διανομής του φορτίου δεδομένων ή τη βλάβη σε κάποιο μηχάνημα.

Αντίθετα στις σχεσιακές βάσεις δεδομένων η διανομή του φορτίου αντιμετωπίζεται με τη χρήση πιο ισχυρών μηχανημάτων.

Συνοψίζοντας λοιπόν τα παραπάνω, τα πλεονεκτήματα των βάσεων NoSQL είναι εμφανή και είναι τα παρακάτω:

1. *Οικονομικότερη Κλιμάκωση.* Στις σύγχρονες συνθήκες του cloud computing και του virtualization, η κλιμάκωση με τη χρήση περισσότερων servers είναι πιο συμφέρουσα από εκείνη των ισχυρότερων μηχανημάτων καθώς οι βάσεις NoSQL έχουν σχεδιαστεί με χαμηλές απαιτήσεις στο υλικό.
2. *Όγκος δεδομένων.* Οι συγκεκριμένες βάσεις ανταποκρίνονται πολύ πιο αποδοτικά στην διαχείριση μεγάλου όγκου δεδομένων σε σχέση με τις σχεσιακές βάσεις.
3. *Ευκολία διαχείρισης.*
4. *Ευελιξία.* Οφείλεται στην ελαστικότητα των δομών.

Οι κατηγορίες των βάσεων NoSQL είναι οι: βάσεις κλειδιού-τιμής, βάσεις οικογενειών από στήλες, βάσεις αρχείων και τέλος βάσεις γράφων οι οποίες και θα αναλυθούν παρακάτω διεξοδικά.

2.5.2 Αρχές Βάσεων Δεδομένων Γράφου

Οι βάσεις δεδομένων γράφου είναι βάσεις που ακολουθούν μια απλή δομή, η οποία μοντελοποιείται με τη βοήθεια κόμβων και σχέσεων μεταξύ τους. Όπως και η δομή των σχεσιακών βάσεων δεδομένων, η δομή των βάσεων γράφου είναι γενική αφού παρέχουν μια απλή και ταυτόχρονα ισχυρή δόμηση δεδομένων. Σημειώνεται πως τα συνδεδεμένα δεδομένα – σε γράφο – υποβοηθούνται με τοπικά ευρετήρια και με αυτόν τον τρόπο η εκτέλεση queries είναι απλή και αποδοτική.

Συνοπτικά μια βάση δεδομένων γράφου έχει τα εξής χαρακτηριστικά:

- Τα δεδομένα αναπαριστώνται με τη δομή γράφων.
- Η διαχείριση των δεδομένων πραγματοποιείται μέσω των μετασχηματισμών του γράφου.
- Η συνέπεια των δεδομένων επιτυγχάνεται μέσω της επιβολής των περιορισμών ακεραιότητας.

Σε αντίθεση με άλλα συστήματα διαχείρισης βάσεων δεδομένων (DBMS) [23] οι σχέσεις μεταξύ των δεδομένων έχουν την βασική προτεραιότητα σε βάσεις δεδομένων γράφου. Αυτή η προσέγγιση (connections-first approach) που ακολουθούν οι βάσεις δεδομένων γράφου καθιστά τις συνδέσεις θεμέλιο λίθο σε κάθε τμήμα του κύκλου ζωής των δεδομένων: Από την ιδέα, μέχρι τον σχεδιασμό του λογικού

μοντέλου, την υλοποίηση του φυσικού μοντέλου και τη λειτουργία του χρησιμοποιώντας μια γλώσσα ερωτημάτων (Query Language). Επιπροσθέτως, τα παραπάνω συνεπάγονται το γεγονός ότι οι εφαρμογές οι οποίες στηρίζονται σε βάσεις δεδομένων γράφου δεν χρειάζονται να συνάγουν συσχετίσεις δεδομένων χρησιμοποιώντας τεχνολογίες όπως ξένα κλειδιά (foreign keys) ή επεξεργασία εκτός ζώνης (out-of-band processing), όπως το MapReduce [24].

Πολλά συστήματα βάσεων δεδομένων έχουν παρόμοια χαρακτηριστικά, αλλά οι βάσεις δεδομένων γράφου έχουν σημαντικά γνωρίσματα που τις καθιστούν μοναδικά. Οι δύο πιο σημαντικές ιδιότητες της συγκεκριμένης τεχνολογίας είναι οι εξής:

1. Αποθήκευση Γράφων

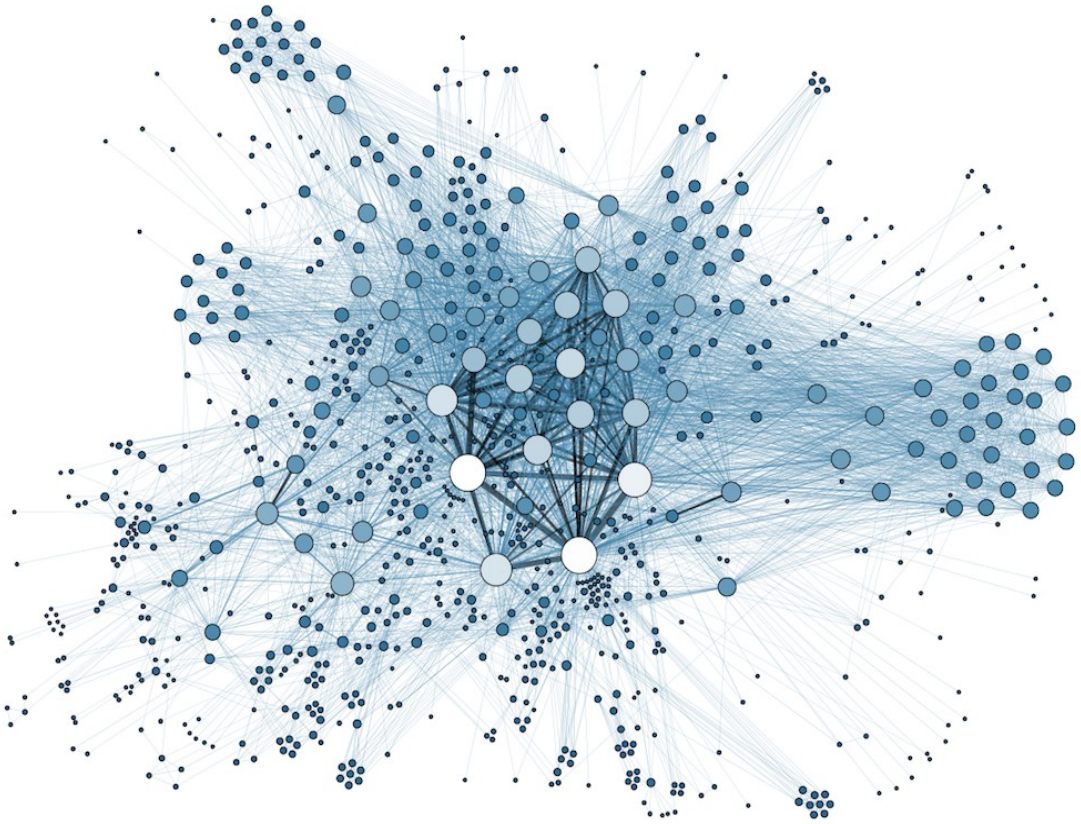
Ορισμένες βάσεις δεδομένων γράφου χρησιμοποιούν εγγενή αποθήκευση γραφημάτων η οποία έχει σχεδιαστεί ειδικά για την αποθήκευση και την διαχείριση τους. Άλλες τεχνολογίες γράφων χρησιμοποιούν σχεσιακές βάσεις, βάσεις οικογενειών από στήλες , αντικειμενοστραφείς βάσεις δεδομένων ως βασικό στρώμα αποθήκευσης. Η τελευταία προσέγγιση προφανώς είναι πιο αργή από την εγγενή αποθήκευση, επειδή όλες οι συνδέσεις των δεδομένων στα γραφήματα πρέπει να μεταφραστούν σε ένα διφορετικό μοντέλο δεδομένων.

2. Επεξεργασία Γραφημάτων

Η εγγενής επεξεργασία των γράφων (index-free adjacency) είναι το πιο αποδοτικό μέσο επεξεργασίας δεδομένων σε ένα γράφημα, διότι οι συνδεδεμένοι κόμβοι δείχνουν εκ φύσεως το ένα στο άλλο στη βάση δεδομένων. Οι μη εγγενείς μηχανές επεξεργασίας γράφων (Non-native graph processing engines) χρησιμοποιούν άλλα μέσα για να επεξεργαστούν λειτουργίες του τύπου Δημιουργίας, Ανάγνωσης, Ενημέρωσης, Διαγραφής (CRUD Functions) [25], οι οποίες όμως καταλήγουν να μην είναι βελτιστοποιημένες για τον χειρισμό των συνδεδεμένων δεδομένων.

Σημειώνεται πως η Neo4j Graph Database αποτελεί μια από τις πιο σύγχρονες και πρωτοπόρες τεχνολογίες όσον αφορά την αποθήκευση και επεξεργασία γράφων.

Δύο μειονεκτήματα των βάσεων δεδομένων γράφου μπορούν να θεωρηθούν τα εξής: Πρώτον, είναι η αδυναμία partitioning η οποία όμως αντισταθμίζεται από τη δυνατότητα scaling out και δεύτερον, η ανάγκη μοντελοποίησης δεδομένων με δομές άγνωστες προς τις κλασικές σχεσιακές, πράγμα που απαιτεί αλλαγή της νοοτροπίας των προγραμματιστών. Παραδείγματα βάσεων δεδομένων γράφου αποτελούν η Neo4j -που χρησιμοποιούμε και στην παρούσα διπλωματική και θα αναφερθούμε εκτενώς σε αυτή-, OrientDB, InfiniteGraph και άλλες.



Εικόνα 2.13 Οπτικοποίηση δομής βάσης δεδομένων γράφου.

2.6 Σχετικές εργασίες

Στην παρούσα ενότητα θα παρουσιαστούν εργασίες σχετικές με το αντικείμενο της συγκεκριμένης διπλωματικής και γενικότερα με τις τεχνολογίες που χρησιμοποίησα. Συγκεκριμένα, θα παρουσιαστούν εργασίες και εφαρμογές που βασίζονται στην τεχνολογία του blockchain για την δημιουργία διάφορων αποκεντρωμένων εφαρμογών καθώς και εφαρμογές που βασίζονται και σε βάσεις δεδομένων γράφου. Σημειώνεται πως οι περισσότερες σχετικές εργασίες οι οποίες θα παρουσιαστούν αφορούν εφαρμογές που σχετίζονται καθαρά με την χρήση blockchain και όχι τόσο με το integration του με βάση δεδομένων γράφου, αφού το εγχείρημα που παρουσιάζεται στην παρούσα διπλωματική είναι από τα πρώτα στον συγκεκριμένο τομέα.

Bloomen [26]

Το ερευνητικό έργο Bloomen υλοποιείται υπό την αιγίδα της Ευρωπαικής Ένωσης (H2020 funded EU Program) και πραγματοποιείται από την Ομάδα εργαστηρίου Κατανεμημένης Γνώσης και Συστημάτων Πολυμέσων της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, με συμμετοχή και του συγγραφέα της παρούσας εργασίας. Το συγκεκριμένο έργο πραγματεύεται την χρήση της πλέον σύγχρονης τεχνολογίας του Blockchain για την δημιουργία μιας αποκεντρωμένης εφαρμογής (Decentralized Application), στους τομείς της μουσικής βιομηχανίας, του πολυμεσικού περιεχομένου και της διαδικτυακής τηλεόρασης. Στόχος της εφαρμογής είναι να φέρει σε άμεση επαφή τους δημιουργούς με τους καταναλωτές του περιεχομένου και να προσφέρει λύσεις που θα δίνουν τη δυνατότητα στους δημιουργούς να εμπορεύονται τα αρχεία των οποίων κατέχουν τα πνευματικά δικαιώματα χωρίς τη βοήθεια μεσαζόντων. Τέλος, μελετώνται τρόποι διαχείρισης πνευματικών δικαιωμάτων αλλά και δίκαιης και ασφαλούς αμοιβής των δημιουργών.

Deploying blockchains for a new paradigm of media experience [27]

Στη σύγχρονη εποχή παρατηρείται το φαινόμενο οι χρήστες να δημοσιεύουν στο διαδίκτυο πολυμεσικό περιεχόμενο, χωρίς όμως να έχουν τον απόλυτο έλεγχο για το ποιος και πως μπορεί να επαναχρησιμοποιήσει το περιεχόμενο αυτό. Προς αυτή την κατεύθυνση προτείνεται μια υπηρεσία βασισμένη στη τεχνολογία του blockchain, η οποία συνδυάζει με επιτυχίες διάφορες πλατφόρμες για την παροχή μεγαλύτερης διαφάνειας και άμεσης πληρωμής όσον αφορά την διανομή του περιεχομένου. Χρησιμοποιήθηκαν δύο πλατφόρμες Blockchain και συγκεκριμένα το Ethereum Platform αλλά και το Hyperledger Fabric v1.0 σε συνδυασμό με μια document-oriented βάση δεδομένων, την MongoDB. Η υλοποίηση αυτή αναδυκνείει τα προτερήματα της τεχνολογίας blockchain ως βάση δεδομένων για διαχείριση αρχείων.

Η συγκεκριμένη δημοσίευση έχει συνταχθεί από μέλη της εργαστηριακής ομάδας Κατανεμημένης Γνώσης και Συστημάτων Πολυμέσων της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου με συμμετοχή και του συγγραφέα της παρούσας εργασίας. Η δημοσίευση έγινε δεκτή στο συνέδριο: 15th International Conference on the Economics of Grids, Clouds, Systems and Services – GECON 2018.

How to import the Bitcoin Blockchain into Neo4j [28]

Η συγκεκριμένη εργασία παρουσιάζει τα βασικά βήματα για την στατική εξαγωγή του Bitcoin Blockchain σε βάση δεδομένων γράφου Neo4j. Η όλη διαδικασία αφορά την λήψη του Bitcoin Blockchain μέχρι και το τελευταίο block (κατά την στιγμή της λήψης) και την μετατροπή του σε γράφο. Εν συνεχεία, αναπαραστώντας το σύνολο της πληροφορίας σε βάση δεδομένων γράφου Neo4j μπορεί κανείς να εκτελέσει ανάλυση δεδομένων η οποία δεν θα ήταν δυνατή με τις παραδοσιακές SQL βάσεις. Για παράδειγμα δίνει τη δυνατότητα εκτέλεσης πολύπλοκων ερωτημάτων ώστε να εξαχθεί το συμπέρασμα όσον αφορά το πως σχετίζονται δύο διαφορετικές διεθύνσεις χρηστών. Σημειώνεται πως η συγκεκριμένη εργασία αποτελεί community post στην επίσημο ιστότοπο της Neo4j και ο αντίστοιχος πηγαίος κώδικας μπορεί να βρεθεί στο Github στο εξής link: https://github.com/in3rsha/bitcoin-to-neo4j .

Thing-to-thing electricity micro payments using blockchain technology [29]

Η ομάδα χρησιμοποίησε το Bitcoin ως την πιο εξέχουσα εφαρμογή blockchain για να παρουσιάσει μια απόδειξη (Proof Of Concept – PoC) της εφαρμογής ενός έξυπνου καλωδίου που συνδέεται με μια έξυπνη πρίζα και χωρίς καμία ανθρώπινη παρέμβαση πληρώνει την κατανάλωση ηλεκτρικής ενέργειας. Το κόστος της ηλεκτρικής ενέργειας για οτιδήποτε συνδέεται στο καλώδιο μπορεί να καταβληθεί σε Bitcoins από το έξυπνο καλώδιο. Κάθε καλώδιο διαθέτει το δικό του λογαριασμό Bitcoin και ο χρήστης δεν θα χρειάζεται να γνωρίζει τις πληρωμές εκτός και αν ο λογαριασμός εξαντληθεί. Τότε χρειάζεται να μεταφέρει χρήματα για να συνεχιστεί η λειτουργία του συστήματος. Τα πλεονεκτήματα της χρήσης του blockchain για αυτή την εφαρμογή είναι η δυνατότητα υποστήριξης πολυάριθμων αυτόνομων συναλλαγών , η εύκολη δημιουργία λογαριασμού για κάθε καλώδιο και το γεγονός ότι δεν υπάρχει κεντρική αρχή που να ελέγχει τους λογαριασμούς. Για να μειωθεί ο αντίκτυπος των αθροιστικά υψηλών χρεώσεων συναλλαγής (transaction fee) κατά την πραγματοποίηση μικρο-συναλλαγών στο δίκτυο Bitcoin, η ομάδα παρουσίασε ένα πρωτόκολλο μικροπληρωμών που συγκεντρώνει πολλαπλές μικρότερες πληρωμές σε μια μεγαλύτερη συναλλαγή που χρειάζεται μόνο μια χρέωση συναλλαγής.

An empirical analysis of linkability in the Monero blockchain[30]

Η συγκεκριμένη δημοσίευση περιγράφει μια εφαρμογή, η οποία χρησιμοποιεί βάση δεδομένων γράφου Neo4j και το δημόσιο Monero blockchain. Ουσιαστικά, περιγράφεται ο τρόπος με τον οποίο όλη η πληροφορία η οποία είναι αποθηκευμένη στο blockchain μεταφέρεται ως γράφος στην Neo4j, έτσι ώστε η ασφάλεια των μελλοντικών συναλλαγών να βελτιωθεί. Σημειώνεται πως η συγκεκριμένη δημοσίευση αποτελεί work-in-progress paper.

Decentralizing Privacy: Using Blockchain to Protect Personal Data [31]

Στη δημοσίευση αυτή παρουσιάζεται ένα κατανεμημένο σύστημα διαχείρισης προσωπικών δεδομένων. Η εφαρμογή αυτή χρησιμοποιεί την τεχνολογία του blockchain για να δημιουργήσει έναν αυτοματοποιημένο διαχειριστή ελέγχου πρόσβασης που παρακάμπτει τις ενδιάμεσες έμπιστες αρχές (TTP) και διασφαλίζει ότι οι χρήστες κατέχουν και ελέγχουν τα προσωπικά τους δεδομένα. Το σύστημα αυτό έχει τρεις οντότητες: τους χρήστες (χρήστες κινητών τηλεφώνων), τις υπηρεσίες και τους κόμβους οι οποίοι είναι φορείς επιφορτισμένοι με την συντήρηση του blockchain και ενός κατανεμημένου χώρου αποθήκευσης ιδιωτικών δεδομένων τύπου κλειδιού-τιμής. Το blockchain δέχεται τη διαχείριση ελέγχου πρόσβασης (Taccess) και τις συναλλαγές αποθήκευσης και διαχείρισης δεδομένων. Ο χρήστης και η υπηρεσία διερευνούν τα δεδομένα από το blockchain (Tdata), το οποίο επαληθεύει αν έχουν πρόσβαση σε αυτό. Οι χρήστες μπορούν ανά πάσα στιγμή να αλλάξουν τα δικαιώματα που έχουν χορηγηθεί σε μια υπηρεσία (Taccess).

3

Εργαλεία και τεχνολογίες

Στο κεφάλαιο αυτό παρουσίαζονται οι κυριότερες τεχνολογίες οι οποίες χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής της παρούσας διπλωματικής εργασίας. Συγκεκριμένα, παρουσιάζονται εργαλεία όπως η αρχιτεκτονική REST (Representational State Transfer) η οποία υιοθετήθηκε στην ανάπτυξη της εφαρμογής μας ως υπηρεσία διαδικτύου. Επιπροσθέτως παρουσιάζεται η σύγχρονη γλώσσα προγραμματισμού Python καθώς και διάφορα εργαλεία τα οποία αυτή προσφέρει (βιβλιοθήκες, Flask Framework κτλ.), τα οποία χρησιμοποιήσα στην παρούσα διπλωματική έτσι ώστε να κατασκευάσω εξ'αρχής ένα δικό μου blockchain βασισμένο στα Bitcoin Standards με τις κύριες λειτουργίες του. Τέλος παρουσιάζεται αναλυτικά η βάση δεδομένων γράφου Neo4j, η οποία είναι το επόμενο κυρίαρχο συστατικό της εφαρμογής αφού ενσωματώθηκε με το προαναφερθέν blockchain.

3.1 Αρχιτεκτονική REST

REST – Representational State Transfer [32] αποτελεί ένα αρχιτεκτονικό στύλ που καθορίζει ένα σύνολο περιορισμών που θα χρησιμοποιηθούν για τη δημιουργία υπηρεσιών ιστού. Οι υπηρεσίες Web που συμμορφώνονται με την REST αρχιτεκτονική ή τις υπηρεσίες RESTful Web ουσιαστικά παρέχουν διαλειτουργικότητα μεταξύ συστημάτων υπολογιστών στο διαδίκτυο. Επιπροσθέτως οι συγκεκριμένες υπηρεσίες ιστού επιτρέπουν στα αιτούμενα συστήματα να έχουν πρόσβαση και να χειρίζονται κειμενικές αναπαραστάσεις των πόρων του διαδικτύου χρησιμοποιώντας όμως έναν ομοιόμορφο και προκαθορισμένο σύνολο stateless λειτουργιών. Σημειώνεται πως άλλα είδη υπηρεσιών ιστού, όπως για παράδειγμα υπηρεσίες ιστού τύπου SOAP (Simple Object Access Protocol) [33], εκθέτουν στο διαδίκτυο τα δικά τους αυθαίρετα σύνολα λειτουργιών.

Οι περιορισμοί του αρχιτεκτονικού στυλ REST ορίζουν τις εξής ιδιότητες οι οποίες έχουν να κάνουν με την γενικότερη αρχιτεκτονική του εκάστοτε συστήματος:

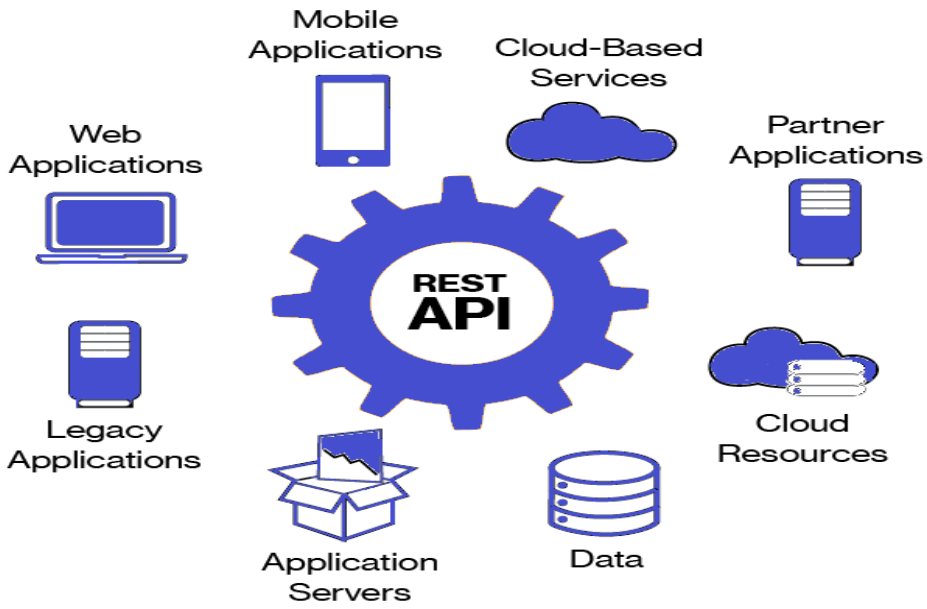
- Την επίδοση στις αλληλεπιδράσεις στα διάφορα components του συστήματος, οι οποίες όμως μπορούν να είναι ο κυρίαρχος παράγοντας στην επίδοση και στην απόδοση του δικτύου.
- Την δυνατότητα κλιμάκωσης (Scalability Potential).
- Την απλότητα μιας ομοιόμορφης διεπαφής.
- Παρέχει την δυνατότητα της εύκολης και γρήγορης τροποποίησης των διαφόρων components για την κάλυψη των μεταβαλλόμενων αναγκών.
- Την ορατότητα μεταξύ των διαφόρων components από service agents.
- Την φορητότητα του πηγαίου κώδικα του συστήματος.
- Την αξιοπιστία σε περίπτωση συνολικής αστοχίας του συστήματος.

Ένα βασικό πλεονέκτημα της χρήσης της τεχνολογίας αυτής τόσο από την πλευρά του πελάτη όσο και από την πλευρά του διακομιστή είναι οι REST αλληλεπιδράσεις που συμβαίνουν μεταξύ των τμημάτων του συστήματος χρησιμοποιώντας όμως δομές οι οποίες είναι γνωστές σε όσους είναι εξοικειωμένοι με τη χρήση του HTTP (Hypertext Transfer Protocol). Παρόμοια, τεχνικά χαρακτηριστικά όπως η κρυπτογράφηση αλλά και η ακεραιότητα μεταφοράς των δεδομένων επιλύονται όχι με την προσθήκη νέων τεχνολογιών, αλλά με την αξιοποίηση της γνωστής μεθόδου κρυπτογράφησης SSL (Secure Sockets Layer) και TLS (Transport Layer Security).

Επίσης το αρχιτεκτονικό στυλ REST είναι ανεξάρτητο από την γλώσσα προγραμματισμού. Οι εφαρμογές που βασίζονται σε αυτή τη τεχνολογία μπορούν να γραφτούν σε οποιαδήποτε γλώσσα προγραμματισμού όπως για παράδειγμα Python, Java, .NET, AngularJS, JavaScript και άλλες. Ουσιαστικά αν μια γλώσσα είναι ‘ικανή’ να κάνει αιτήσεις στο διαδίκτυο χρησιμοποιώντας το HTTP πρωτόκολλο, τότε είναι πιθανό αυτή η γλώσσα να μπορεί να χρησιμοποιηθεί για την κλήση ενός RESTful API [34] ή μιας υπηρεσίας ιστού. Αυτό έχει ως αποτέλεσμα οι προγραμματιστές να υλοποιούν μια REST υπηρεσία ιστού χρησιμοποιώντας ως βάση την γλώσσα προγραμματισμού την οποία εκείνοι ξέρουν να μεταχειρίζονται καλύτερα.

Το άλλο πλεονέκτημα που προσφέρεται είναι η διαπερατότητα του. Από την πλευρά του διακομιστή (server side), υπάρχουν ποικίλα πλαίσια που βασίζονται σε REST για να βοηθήσουν τους προγραμματιστές να δημιουργήσουν RESTful υπηρεσίες ιστού, συμπεριλαμβανομένων των RESTlet, Python Flask και ApacheCXF. Από την πλευρά του πελάτη (client side) όλα τα νέα πλαίσια που έχουν προταθεί (και ιδιαίτερα της JavaScript όπως το JQuery, Node.js, Angular & EmberJS) διαθέτουν

όλες τις τυπομοιημένες βιβλιοθήκες ενσωματωμένες στο API τους για εύκολη και γρήγορη κλήση RESTful υπηρεσιών ιστού.



Εικόνα 3.1 Περιβάλλον RESTful υπηρεσίας ιστού

3.2 Η γλώσσα προγραμματισμού Python

Η Python [35] είναι μια αντικειμενοστραφής διερμηνευόμενη γλώσσα γενικού σκοπού με δυναμική σημασιολογία (semantics) και δημιουργήθηκε από τον Guido van Rossum και κυκλοφόρησε δημοσίως το 1991. Αποτελεί γλώσσα υψηλού επιπέδου και και η δημιουργία της βασίστηκε στην εύκολη αναγνωσιμότητα του κώδικα ενώ ταυτόχρονα διακρίνεται για την πλούσια εκφραστικότητα της.

Έχει μια μεγάλη κύρια βιβλιοθήκη η οποία καλύπτει ένα τεράστιο εύρος πεδιών, κάνοντας την ιδανική για χρήση σε οποιαδήποτε σχεδόν εφαρμογή. Συμβάλλει στην εξαιρετικά γρήγορη ανάπτυξη εφαρμογών αλλά σε καμία περίπτωση δεν υστερεί σε λειτουργίες που προσφέρουν οι υπόλοιπες εξίσου διαδεδομένες γλώσσες προγραμματισμού όπως η C ή η Java. Επιπροσθέτως, υποστηρίζει πολύ υψηλού επιπέδου δομές δεδομένων, παρέχει αυτόματη διαχείριση μνήμης και είναι φιλική σε οποιοδήποτε λειτουργικό σύστημα.

Εικόνα 3.2 Το λογότυπο της Python (Πηγή: https://www.python.org)

3.2.1 Κύρια Χαρακτηριστικά της Python

Όπως προαναφέρθηκε η Python αποτελεί αντικειμενοστραφή γλώσσα προγραμματισμού. Επιπροσθέτως ένα άλλο πολύ σημαντικό χαρακτηριστικό της είναι ότι συνδυάζει πολλαπλά πρότυπα προγραμματισμού, συμπεριλαμβανομένων του προστακτικού και του συναρτησιακού προγραμματισμού. Εν συνεχεία οι υψηλού επιπέδου δομές δεδομένων που προσφέρει, οι δυναμικοί τύποι κωδικοποίησης, η αυτόματη διαχείριση μνήμης καθώς επίσης και μια μεγάλη πρότυπη βιβλιοθήκη την καθιστούν μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού.

Επίσης, ένα άλλο αξιοσημείωτο χαρακτηριστικό της Python είναι ότι προάγει την εύκολη συντήρηση και επαναχρησιμοποίηση του πηγαίου κώδικα, αφού δίνει την δυνατότητα της ομαδοποίησης του σε μονάδες και πακέτα. Τέλος, σημαντικά γνωρίσματα της είναι και τα παρακάτω:

- Υποστηρίζει τις εξαιρέσεις.
- Δυνατότητα ενσωμάτωσης σε μια εφαρμογή ώστε να λειτουργεί σαν RESTful υπηρεσία διαδικτύου.
- Συμβατότητα με όλες τις κύριες πλατφόρμες υλικού και λογισμικού.
- Χρησιμοποιεί διερμηνέα και είναι scripting language.
- Δημιουργία μικρότερων σε μέγεθος προγραμμάτων σε σχέση με άλλες γλώσσες προγραμματισμού.
- Πληθώρα IDEs : IDLE, Ipython, PythonAnywhere (online), ...

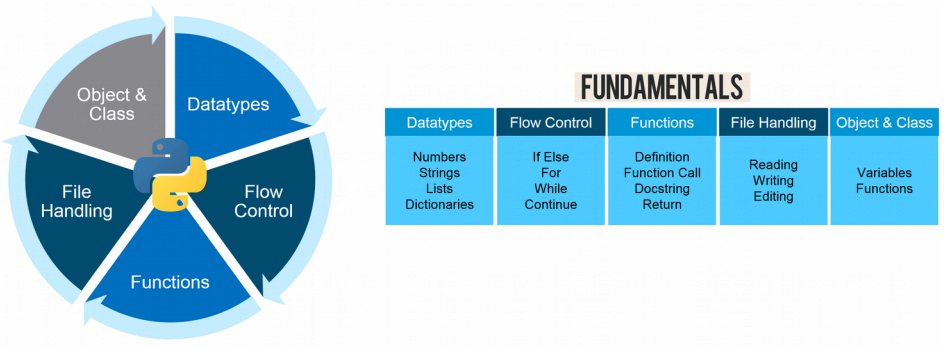
3.2.2 Υλοποιήσεις της Python

Η γλώσσα προγραμματισμού Python αποτελείται από το πρότυπο και την υλοποίηση αναφοράς CPython[36], αλλά υπάρχουν και διαφορετικές υλοποιήσεις που διαφέρουν κατά κάποιο τρόπο από την υλοποίηση αναφοράς. Ως υλοποίηση της Python νοείται έν πρόγραμμα η περιβάλλον που παρέχει υποστήριξη για την εκτέλεση προγραμμάτων γραμμένα στην γλώσσα προγραμματισμού Python, όπως παριστάνεται από την υλοποίηση αναφοράς της CPython.

Η υλοποίηση αναφοράς CPython, είναι η κύρια και αρχική υλοποίηση της Python, γραμμένη με τη γλώσσα προγραμματισμού C πληρώντας το πρότυπο προδιαγραφών C89. Η CPython και οι περισσότερες υλοποιήσεις της Python, μεταφράζουν προγράμ- ματα γραμμένα σε κώδικα Python σε μορφή κώδικα byte, ο οποίος εκτελείται από την εικονική μηχανή παρέχοντας φορητότητα, καθώς είναι σε μια μορφή ανεξάρτητη από τη πλατφόρμα.

Κάποιες εναλλακτικές υλοποιήσεις της Python βασίζονται στον πυρήνα εκτέλεσης της υλοποίησης αναφοράς CPython, αλλά με εκτεταμένη συμπεριφορά ή χαρακτηριστικά σε ορισμένες πτυχές όπως η Stackless Python, που δίνει έμφαση στη παραλληλία.

Άλλες υλοποιήσεις δεν εξαρτώνται ή αλληλοεπιδρούν με τον πυρήνα εκτέλεσης αλλά επαναχρησιμοποιούν ένα μεγάλο μέρος της πρότυπης βιβλιοθήκης και είναι συμβατές με την προτυποποίηση της γλώσσας. Χαρακτηριστικά παραδείγματα η IronPython και η Jython, υλοποιήσεις για τις πλατφόρμες Common Language Runtime (CLR/.NET) και Java αντίστοιχα.



Εικόνα 3.3 Βασικά γνωρίσματα της Python

3.2.3 Python Flask Microframework

Το Python Flask [37] αποτελεί μια μικρή αλλά πανίσχυρη REST βιβλιοθήκη της Python η οποία συμβάλλει στην υλοποίηση Python εφαρμογών για το διαδίκτυο. Σημειώνεται πως βασίζεται στα εργαλεία Werkzeug και Jinja2 και διατίθεται με την άδεια BSD.

Τα βασικά χαρακτηριστικά του Flask είναι τα παρακάτω:

- Εξυπηρετεί RESTful requests
- Περιέχει λειτουργία αποσφαλμάτωσης (Flask Debugger)
- Παρέχει εξυπηρετητή για την διαδικασία της ανάπτυξης
- Ενσωματωμένη λειτουργία για σενάρια δοκιμών της εφαρμογής
- Υποστηρίζει Sessions
- Πλήρης συμβατότητα με WSGI 1.0
- Unicode-based
- Δυνατότητα εισαγωγής προσθέτων

Εικόνα 3.4 Το λογότυπο του Flask (Πηγή: http://flask.pocoo.org)

3.3 Η βάση δεδομένων γράφου Neo4j

Η Neo4j[38] αποτελεί τη γνωστότερη εμπορική εφαρμογή βάσης δεδομένων γράφου. Είναι γραμμένη κυρίως σε Java και διακρίνεται για την προσαρμοσμένη μορφή αποθήκευσης και τις εγκαταστάσεις της Java Transaction Architecture (JTA) [39]. Το Python API, το οποίο χρησιμοποιούμε εμείς στην παρούσα διπλωματική, (καθώς και το Java API) προσφέρει έναν αντικειμενοστραφή τρόπο εργασίας με τους κόμβους και τις ακμές του γράφου και οπτικοποιεί με σαφήνεια τις εργασίες διάσχισης μονοπατιών. Η συγκεκριμένη βάση υλοποιεί διάφορους αλγορίθμους γράφων, όπως για παράδειγμα ο αλγόριθμος Dijkstra [40] για το υπολογισμό της συντομότερης διαδρομής. Σημειώνεται ότι παρέχεται η εφαρμογή/πλατφόρμα της Neo4j (Neo4j Graph Database Platform) δωρεάν για λήψη στην επίσημη ιστοσελίδα.

Εικόνα 3.5 Το λογότυπο της Neo4j

3.3.1 Χαρακτηριστικά της Neo4j

Οι θεμέλιοι λίθοι της Neo4j είναι οι κόμβοι οι οποίοι αναπαριστούν οντότητες και οι ακμές οι οποίες αναπαριστούν τις μεταξύ τους σχέσεις. Και τα δύο αυτά στοιχεία διαθέτουν χαρακτηριστικά.

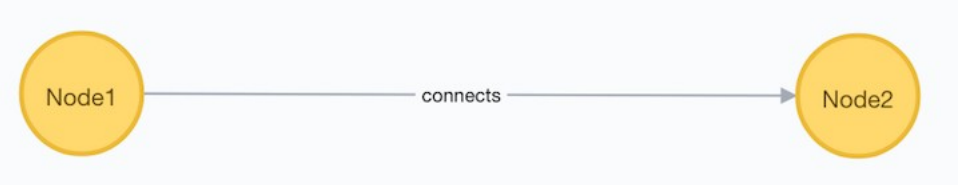
Κάθε κόμβική δομή του γράφου περιέχει ένα μοναδικό κλειδί και έναν πίνακα που περιγράφει τις σχέσεις του με άλλους κόμβους του γράφου. Ο πίνακας που περιγράφει τις σχέσεις αποτελείται από τους

κωδικούς των κόμβων από τους οποίους αποτελείται η σχέση με την παρακάτω κωδικοποίηση : out_node → in_node. Στην φάση δημιουργίας του κόμβου στην βάση δεδομένων γράφου Neo4j αρχικοποιείται ένα μοναδικό κλειδί (unique key) και εν συνεχεία εγγράφεται ο κόμβος με το κλειδί του και έναν άδειο πίνακα σχέσεων. Επιπροσθέτως, στην φάση δημιουργίας μιας σχέσης μεταξύ δύο κόμβων, ενημερώνονται αρχικά οι κόμβοι που συμμετέχουν στη σχέση και ύστερα αρχικοποιείται ένα κλειδί για τη σχέση η οποία στο τέλος εγγράφεται ως αυτόνομη οντότητα.

Για να γίνει πιο κατανοητή η σύνδεση σε μια βάση δεδομένων γράφου Neo4j μέσω Python API [41], καθώς και η δημιουργία κόμβων και σχέσεων παρατίθεται το παρακάτω παράδειγμα (τμήμα πηγαίου κώδικα):

```
# Import the appropriate Python libraries
from neo4jrestclient.client import GraphDatabase
from neo4jrestclient import client

#Connect with the corresponding Neo4j db
neo4j_db = GraphDatabase(DB_PATH,UNAME,PASS)

#Create Nodes Category and afterwards a node
db_nodes = neo4j_db.labels.create("Node")
node1 = neo4j_db.nodes.create(name="Node1")
db_nodes.add(node1)
node2 = neo4j_db.nodes.create(name="Node2")
db_nodes.add(node2)

#Create a Relationship between Node1 and Node2
#Node1-connects->Node2
node1.relationships.create("connects")
```

Εικόνα 3.6 Δημιουργία κόμβων και σχέσεων στην Neo4j μέσω Python API

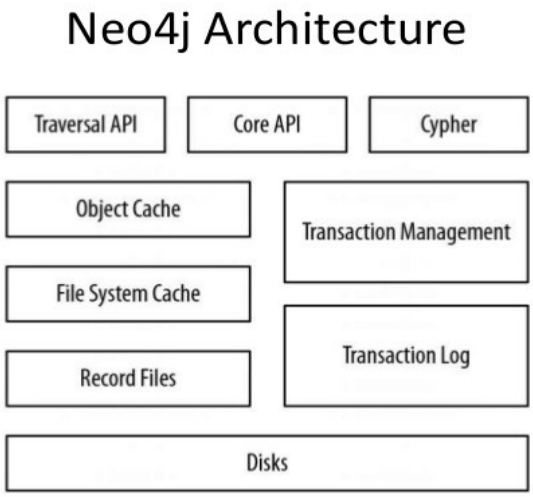


Εικόνα 3.7 Οπτικοποίηση του παραπάνω κώδικα στην Neo4j Graph Platform

Η Neo4j χρησιμοποιεί τη δηλωτική γλώσσα επερώτησης (Query Language) Cypher [42], η οποία είναι εύκολα αντιληπτή για τους χρήστες που γνωρίζουν την SQL των σχεσιακών βάσεων δεδομένων. Οι χρήστες μπορούν να τρέξουν ad-hoc ερωτήματα σχετικά με το προκύπτων γράφημα για μια ποικιλία από use cases. Σημειώνεται πως η Cypher υποστηρίζει συναρτήσεις φιλτραρίσματος, ομαδοποίησης, σελιδοποίησης και επιτρέπει την εύκολη δημιουργία, διαγραφή, ενημέρωση και κατασκευή γράφου. Τέλος υπάρχει δυνατότητα binding με μια πληθώρα γλωσσών μέσω εξειδικευμένων API, όπως Jython, CPython, JRuby, Clojure, Scala.

Η συγκεκριμένη βάση δεδομένων γράφου έχει υλοποιήσει μια πληθώρα API σχεδίασης τα οποία λαμβάνουν υπόψιν παραμέτρους που εισάγει ο χρήστης και αυτό έχει ως αποτέλεσμα να υποστηρίζονται πολύπλοκα εξειδικευμένα ερωτήματα χωρίς την ανάγκη προγραμματιστικών γνώσεων.

Η Neo4j διακρίνεται για την απόδοση της. Συγκεκριμένα, η ταχύτητα ανάγνωσης που προσφέρει είναι εντυπωσιακή καθώς διασχίζει 1.000.000 σχέσεις ανά 1 δευτερόλεπτο. Επίσης, ένα άλλο πλεονέκτημα είναι ότι μπορεί να χειριστεί σχήματα με μεγάλο συντελεστή εξέλιξης ή χαμηλό δείκτη δόμησης, όπως είναι για παράδειγμα τα κοινωνικά δίκτυα. Τέλος προσφέρει δυνατότητες αποτελεσματικού scaling out μέσω αντιγραφής (replication) – σύγχρονο write μεταξύ master και όλων των slaves, ασύγχρονο update μεταξύ slaves και partitioning.



Εικόνα 3.8 Αρχιτεκτονική της βάσης δεδομένων γράφου Neo4j

4

Σχεδιασμός και υλοποίηση Συστήματος

Στο κεφάλαιο αυτό παρουσιάζονται λεπτομέρειες που αφορούν τον σχεδιασμό και την υλοποίηση της εφαρμογής που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας. Επιπροσθέτως θα περιγραφούν τα κυριότερα προβλήματα που αντιμετωπίστηκαν κατά την ανάπτυξη της εφαρμογής, ώστε να προβληθεί ακόμη περισσότερο η εκπαιδευτική πτυχή της όλης υλοποίησης.

4.1 Μακροσκοπική Αρχιτεκτονική Συστήματος

Στο παρών κεφάλαιο θα περιγραφεί η γενικότερη αρχιτεκτονική του συστήματος όσον αφορά την δομή δεδομένων blockchain και την ενσωμάτωση της με την βάση δεδομένων γράφου Neo4j, καθώς και η εσωτερική οργάνωση των δεδομένων στην βάση.

Σημειώνεται πως πριν ξεκινήσουμε την περιγραφή, παραθέτουμε τεχνική ορολογία η οποία είναι αναγκαία για την πλήρη κατανόηση της εφαρμογής:

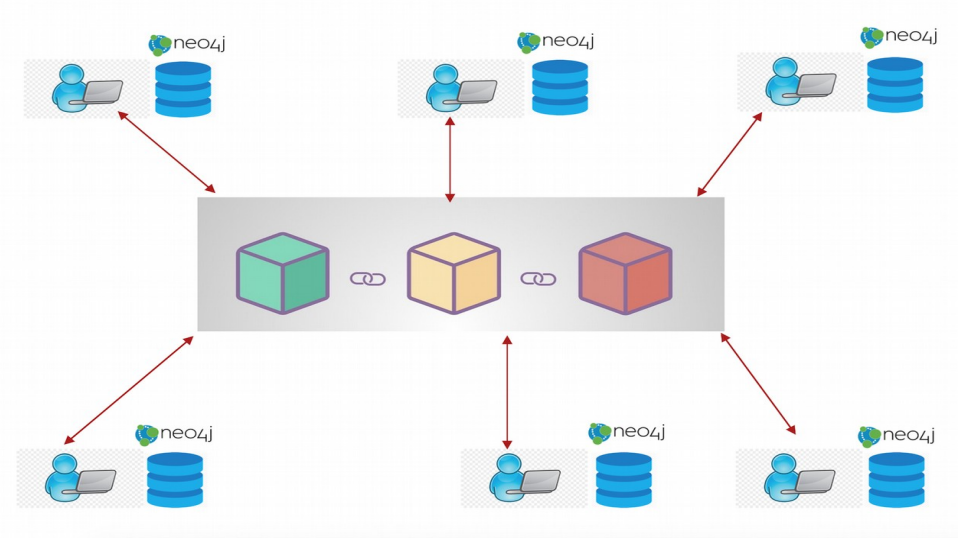
- *Block:* Ένα μπλοκ είναι μια μοναδική εγγραφή στο blockchain που περιέχει συναλλαγές, timestamps, index, hash και οποιαδήποτε άλλη πληροφορία κρίνεται αναγκαία. Κατά κύριο λόγο υπάρχουν 3 είδη Blocks και συγκεκριμένα το Genesis Block (πρώτο block στο blockchain), το Current Block (δηλαδή το τελευταίο block στο blockchain) και τέλος το Orphan Block (δεν ανήκει στην κύρια αλυσίδα).
- *Node:* Ένας διακομιστής αποτελεί έναν ξεχωριστό κόμβο σε ένα peer-to-peer δίκτυο blockchain.
- *Consensus Algorithms:* Όταν στο δίκτυο έχουμε παραπάνω από έναν node, για να σιγουρευτούμε ότι όλοι οι κόμβοι έχουν το ίδιο ακριβώς blockchain και να συμφωνούν σε αυτό, χρησιμοποιούμε τέτοιου είδους αλγορίθμους.

Ο γενικότερος σχεδιασμός του συστήματος είναι ο εξής: Το πρωτότυπο blockchain υλοποιείται εξ'αρχής στη γλώσσα προγραμματισμού Python με στόχο να αντιπροσωπεύσουμε αποτελεσματικά τον κατανεμημένο χαρακτήρα του καθώς και να φανούν τα βασικά δομικά χαρακτηριστικά του blockchain τα οποία έχουν ήδη προαναφερθεί. Επιπλέον, χρησιμοποιώντας το Python Flask μπορέσαμε να μοντελοποιήσουμε και να προσομοιώσουμε πραγματικές συνθήκες peer-to-peer δικτύων αφού κάθε διαφορετικός κόμβος διατηρεί την ίδια ακριβώς δομή δεδομένων.

Ταυτόχρονα, κάθε ομότιμος κόμβος της εφαρμογής αρχικοποιεί και χρησιμοποιεί μια βάση δεδομένων Neo4j (*http://localhost:7474*), η οποία λειτουργεί ως ένα πολύ πιο ευέλικτο αντίγραφο/γράφο του blockchain και συγχρονίζεται δυναμικά με αυτό, προσφέροντας στον χρήστη μεγάλες δυνατότητες (όπως για παράδειγμα complex queries). Η βάση δεδομένων γράφου Neo4j η οποία τρέχει σε κάθε κόμβο, περιγράφει το blockchain μέσω μιας τριεπίπεδης αρχιτεκτονικής:

- *1ο Επίπεδο*: Blocks που έχουν δημοσιευθεί στο blockchain.
- *2ο Επίπεδο*: Συναλλαγές που αντιστοιχούν στο εκάστοτε block του πρώτου επιπέδου.
- *3ο Επίπεδο*: Οι χρήστες οι οποίοι συμμετέχουν στις εκάστοτε συναλλαγές του δεύτερου επιπέδου.

Η μακροσκοπική αρχιτεκτονική του συστήματος μας φαίνεται στην παρακάτω εικόνα:



Εικόνα 4.1 Μακροσκοπική Αρχιτεκτονική της Εφαρμογής

4.2 Υλοποίηση Συστήματος

Προφανώς η δομή του Blockchain είναι η βάση της παρούσας εφαρμογής και στο συγκεκριμένο κεφάλαιο θα περιγραφεί η υλοποίηση ενός πρωτότυπου στη γλώσσα προγραμματισμού Python, χρησιμοποιώντας επίσης και το Python Flask Microframework. Επιπροσθέτως, στο παρών κεφάλαιο θα περιγράψουμε και την πραγματική ενσωμάτωση της βάσης δεδομένων γράφου Neo4j.

Ο συνολικός αντικειμενοστραφής πηγαίος κώδικας χωρίζεται σε τέσσερα βασικά μέρη τα οποία προφανώς αλληλεπιδρούν μεταξύ τους, όπου κάθε μέρος υλοποιεί και ένα βασικό συστατικό της εφαρμογής. Συγκεκριμένα, τα μέρη είναι τα εξής:

- *Κλάση Main*, η οποία είναι υπεύθυνη για την αρχικοποίηση της Neo4j βάσης και του Server στον οποίο λειτουργεί ο κάθε κόμβος.
- *Flask Resources.* Είναι το τμήμα του κώδικα το οποίο είναι υπεύθυνο για την εξυπηρέτηση των http αιτημάτων (get & post requests) του δικτύου.
- *Κλάση Block,* η οποία είναι υπεύθυνη για τον χειρισμό της δομής του κάθε block.
- *Κλάση Blockchain,* η οποία είναι υπεύθυνη για τον χειρισμό ολόκληρης της δομής του blockchain.

Παρακάτω ακολουθεί ο πηγαίος κώδικας του Main τμήματος της εφαρμογής μας το οποίο εκτελείται πρώτο όταν ξεκινάει η εφαρμογή σε κάθε κόμβο.

```
if __name__ == '__main__':


 # Neo4j Database Initialization
 db = GraphDatabase("http://localhost:7474" , username =
      "neo4j", password = "123456789")
 db_blocks = db.labels.create("Blocks")
 db_transactions = db.labels.create("Transactions")
 db_users = db.labels.create("Users")

#  Blockchain Data Structure Initialization
my_blockchain                          =
 Blockchain(db,db_blocks,db_transactions,db_users)
```

```
    # Flask Server Initialization
    parser = ArgumentParser()
    parser.add_argument('-H', '--host', default='0.0.0.0')
    parser.add_argument('-p', '--port', default=5000, type =
int)
    args = parser.parse_args()
    app.run(host = args.host, port =
args.port ,debug=True,use_reloader=False)
```

Παρατηρούμε λοιπόν παραπάνω πως την στιγμή που ξεκινάει η εφαρμογή στον εκάστοτε κόμβο, το Python API μας δίνει τη δυνατότητα να συνδεθούμε στην τοπική βάση που έχουμε δημιουργήσει μέσω Neo4j Desktop Application [43] και αυτό επιτυγχάνεται με την πρώτη εντολή (βάζοντας φυσικά τα σωστά credentials σαν παραμέτρους). Σημειώνεται πως η εκάστοτε τοπική Neo4j του κάθε κόμβου ακούει στην πόρτα 7474. Οι επόμενες τρεις εντολές ουσιαστικά ορίζουν τους κόμβους της βάσης γράφου και περιγράφουν την τριεπίπεδη αρχιτεκτονική της (Blocks, Transactions, Users). Επιπροσθέτως, η αρχικοποίηση της δομής του blockchain πρέπει με κάποιο τρόπο προφανώς να πραγματοποιείται αυτοματοποιημένα σε κάθε κόμβο. Οπότε, αρχικοποιώ την συγκεκριμένη δομή του blockchain καλώντας τους κατάλληλους κατασκευαστές (constructors) και τον εκάστοτε τοπικό Server πάνω στον οποίο τρέχει ο κάθε κόμβος μέσω Python Flask Microframework, χρησιμοποιώντας την βιβλιοθήκη ArgumentParser που προσφέρει η Python.

Ύστερα από την σύνδεση στη βάση δεδομένων γράφου μέσω Python API, πρέπει να ξεκινήσουμε να υλοποιούμε το πρωτότυπο blockchain εκμεταλλευόμενοι φυσικά και την ήδη συνδεδεμένη Neo4j. Συγκεκριμένα μέσω της κλάσης Block, σε κάθε block αποθηκεύονται οι σφραγίδες χρόνου (timestamps), ένα index το οποίο προσδιορίζει σε ποιο μήκος βρίσκεται το blockchain μέχρι την δεδομένη χρονική στιγμή καθώς και το κρυπτογραφικό κατακερματισμένο αριθμό (hash) το οποίο εγγυάται την ακεραιότητα του. Σημειώνεται πως ο συγκεκριμένος αριθμός υπολογίζεται μέσω της συνάρτησης *calculateHash,* η οποία χρησιμοποιεί την βιβλιοθήκη της Python "*hashlib*" [44] και συγκεκριμένα τον αλγόριθμο SHA256 [45]. Τέλος, κάθε block αποθηκεύει και τον κρυπτογραφικό κατακερματισμένο αριθμό του προηγούμενου block σε μορφή κατακερματισμένου δείκτη (hash pointer). Παρακάτω δίνεται το τμήμα του κώδικα το οποίο περιέχει τον

κατασκευαστή του block, ο οποίος ορίζει τα παραπάνω βασικά πεδία καθώς και κάποια ακόμα.

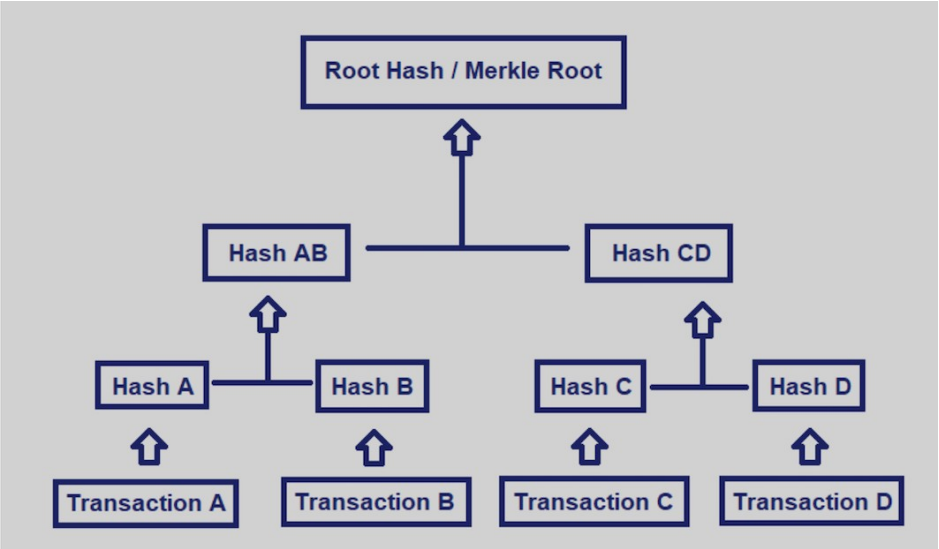
```
     # Constructor of the Block
            def __init__(self, index, timestamp, data,
            previousHash = '' ):
              self.index = index
              self.previousHash = previousHash
              self.timestamp = timestamp
              self.data = data
              self.nonce = 0  #Only for PoW
               self.hash_merkle_block = ""
              self.hash = self.calculateHash()

     # Calculate the current unique block hash number
            def calculateHash(self) :
             my_string = (str(self.index) + str(self.previousHash)
+str(self.timestamp)+str(self.data + str(self.nonce)).encode('utf-
8')
             hash_object = hashlib.sha256(my_string)
             hex_dig = hash_object.hexdigest()
             return(hex_dig)
```

Παρατηρούμε ότι στον παραπάνω κατασκευαστή (constructor) του block υπάρχουν όλα τα πεδία τα οποία προαναφέρθηκαν καθώς και άλλο ένα πεδίο, το nonce. Αυτό το πεδίο ουσιαστικά περιγράφει τον αριθμό ο οποίος λύνει το κρυπτογραφικό puzzle (Ενότητα 2.4.3) μέσω του αλγορίθμου Proof of Work του οποίου η υλοποίηση θα αναλυθεί ύστερα εκτενέστερα. Επιπροσθέτως, υπάρχει και το πεδίο των data μέσα στο οποίο αποθηκεύονται όλες οι πληροφορίες για τις συναλλαγές.

Επιπλέον παρατηρούμε άλλο ένα πεδίο που ονομάζεται hash_merkle_block, το οποίο ορίζεται μέσω της χρήσης της δομής του Merkle Tree [46]. Συγκεκριμένα, ένα δέντρο Merkle συνοψίζει όλες τις συναλλαγές σε ένα μπλοκ δημιουργώντας έμα ψηφιακό δακτυλικό αποτύπωμα του συνόλου των συναλλαγών, επιτρέποντας έτσι σε έναν χρήστη να επαληθεύσει εάν μια συναλλαγή περιλαμβάνεται ή όχι σε ένα μπλοκ. Η συγκεκριμένη δομή κατασκευάζεται μέσω του επαναλαμβανόμενου κατακερματισμού (hashing) ζευγών κόμβων συναλλαγών μέχρι να μείνει μόνο ένα hash, το οποίο και ονομάζεται ρίζα Merkle ή αλλιώς Root Hash. Κάθε φύλλο του δέντρου είναι δεδομένα από συναλλαγές και κάθε ενδιάμεσος κόμβος (όχι φύλλο) είναι ο κατακερματισμένος αριθμός από τα προηγούμενα hashes. Οπότε το πεδίο

hash_merkle_block περιέχει την ρίζα Merkle παρέχοντας ασφάλεια όσον αφορά τις συναλλαγές.

Παρακάτω παρατίθεται ένα γράφημα το οποίο δείχνει την διαδικασία παραγωγής της ρίζας Merkle από τις συναλλαγές οι οποίες υπάρχουν στο εκάστοτε block.



Εικόνα 4.2 Δέντρο Merkle από τις συναλλαγές A,B,C,D

Στην παρούσα εφαρμογή υλοποιήσαμε τον αλγόριθμο ο οποίος δημιουργεί το δέντρο Merkle με επαναλαμβανόμενους κατακερματισμούς από τα αριστερά μέχρις ότου καταλήξει στην ρίζα Merkle. Σημειώνεται πως το παρακάτω τμήμα κώδικα χρησιμοποιείται σαν βιβλιοθήκη στην κύρια εφαρμογή μας. Ο Python κώδικας για τον προσδιόρισμο της ζητούμενης ρίζας είναι ο εξής:

```
# 0. Import the needed library
import hashlib,json
from collections import OrderedDict


# 1. Declare the class trees
class MerkTree:

    # 2. Initiate the class object
    def __init__(self,listoftransaction=None):
        self.listoftransaction = listoftransaction
        self.past_transaction = OrderedDict()

    # 3. Create the Merkle Tree
    def create_tree(self):
```

```
            # 3.0 Continue on the declaration
            listoftransaction = self.listoftransaction
            past_transaction = self.past_transaction
temp_transaction = []

        # 3.1 Loop until the list finishes
            for index in range(0,len(listoftransaction),2):

                # 3.2 Get the most left element
                current = listoftransaction[index]

                # 3.3 If there is still index left get the
right of the left most element
                if index+1 != len(listoftransaction):
                    current_right = listoftransaction[index+1]

                # 3.4 If we reached the limit of the list then
make a empty string
                else:
                    current_right = ''

                # 3.5 Apply the Hash 256 function to the
current values
                current_hash = hashlib.sha256(current)

                # 3.6 If the current right hash is not a '' <-
empty string
                if current_right != '':
                    current_right_hash                          =
hashlib.sha256(current_right)

                # 3.7 Add the Transaction to the dictionary
                past_transaction[listoftransaction[index]]    =
current_hash.hexdigest()

                # 3.8 If the next right is not empty
                if current_right != '':
                    past_transaction[listoftransaction[index+1
]] = current_right_hash.hexdigest()

                # 3.9 Create the new list of transaction
                if current_right != '':
                    temp_transaction.append(current_hash.hexdi
gest() + current_right_hash.hexdigest())

                # 3.01 If the left most is an empty string
then only add the current value
                else:
                    temp_transaction.append(current_hash.hexdi
gest())

            # 3.02 Update the variables and rerun the function
again
            if len(listoftransaction) != 1:
                self.listoftransaction = temp_transaction
```

```
                self.past_transaction = past_transaction



            # 3.03 Call the function repeatly again and again
#until we get the root
                self.create_tree()


        # 4. Return the past Transaction
        def Get_past_transacion(self):
            return self.past_transaction

        # 5. Get the root of the transaction
        def Get_Root_leaf(self):
            last_key = self.past_transaction.keys()[-1]
            return self.past_transaction[last_key]
```

Εφόσον τώρα ορίσαμε την δομή του block, πρέπει να δημιουργήσουμε τελικά το blockchain το οποίο αποτελείται από τα παραπάνω blocks. Συγκεκριμένα, ο κάθε κόμβος του δικτύου θα πρέπει να διατηρεί έναν πίνακα `chain[]` όπου εκεί θα διατηρεί όλα τα επικυρωμένα blocks για τα οποία έχει ενημερωθεί πως έχουν εισαχθεί στο public blockchain. Επιπροσθέτως, ο κάθε κόμβος είναι υποχρεωμένος να διατηρεί μια προσωρινή μνήμη η οποία θα περιέχει τις συναλλαγές τις οποίες έχει "ακούσει" από το ομότιμο δίκτυο και είναι υποψήφιες να δημοσιευτούν στο επόμενο block – transaction pool - , αν ο εκάστοτε κόμβος καταφέρει να δημοσιεύσει αυτός το block. Ο κώδικας ο οποίος αρχικοποιεί αυτές τις δομές είναι ο παρακάτω:

```
       class Blockchain(object):


    'Common base for the instance of blockchain'
    def
__init__(self,database,db_blocks,db_transactions,db_users):
    self.chain = []
    self.node_transactions = [] # Storing the transactions
    self.nodes = set() # Here we keep all the connected nodes
                       #in our blockchain network

    # --- Check if you are the first node in the network --- #
    # --- If not add the standard seed localhost:5000 and
synchronize with him --- #
    number_of_blocks = "MATCH (b:Blocks) return COUNT(b)"
    results = db.query(number_of_blocks,returns=(int))
    for r in results:
    blocks_number = r[0]
    if blocks_number == 0:
```

```
    self.chain.append(self.createGenesisBlock(database,db_bloc
ks,db_transactions,db_users))
    else:
    self.create_node('http://localhost:5000')
    peer_chains = self.get_peers_blockchain(self)
    longest_chain = max(peer_chains, key=len)

    self.chain = [self.get_data_blockchain(block1) for block1
in longest_chain]
    self.difficulty = 5
```

Στον παραπάνω κώδικα παρατηρούμε ότι αφου αρχικοποιηθούν οι προαναφερθείσες δομές του blockchain (chain[], node_transactions[]), πραγματοποιείται ένα query στην εκάστοτε τοπική Neo4j κατανεμημένη βάση του κόμβου. Συγκεκριμένα, ο κόμβος ζητάει από την βάση να μάθει πόσα blocks έχουν δημοσιευτεί/αποθηκευτεί και συγκεκριμένα το Cypher query είναι το εξής: "`MATCH (b:Blocks) RETURN COUNT(b)`". Ύστερα, χρησιμοποιώντας το Python API το οποίο προσφέρεται από την Neo4j εκτελώ στην βάση το παραπάνω query μέσω της εξής εντολής: "`db.query(number_of_blocks, returns(int))`", η οποία μας επιστρέφει σε μορφή πίνακα ακεραίων τον αριθμό των κόμβων που έχουν οριστεί σαν blocks. Αυτό συμβαίνει, διότι κάθε κόμβος ο οποίος αρχικοποιεί τις παραπάνω δομές πρέπει να γνωρίζει αν είναι ο πρώτος κόμβος του δικτύου ώστε να δημιουργήσει και το Genesis Block. Αν υπάρχουν ήδη και άλλα blocks σημαίνει ότι όταν ο νέος κόμβος συγχρονίστηκε με το δίκτυο υπήρχαν ήδη άλλα blocks, και έτσι συνδέεται με άλλους peers ώστε να λάβει και την πιο τελευταία έκδοση του blockchain.

Αν όμως στην συγχρονισμένη κατανεμημένη βάση δεν υπάρχουν άλλα blocks, αυτό σημαίνει ότι πρέπει να δημιουργήσει το πρώτο block του blockchain και αυτό το επιτυγχάνει μέσω της γραμμής: "`self.chain.append(self.createGenesisBlock(database,db_bl
ocks,db_transactions,db_users))`".
Έτσι καλείται η παρακάτω συνάρτηση της κλάσης Blockchain – createGenesisBlock- , η οποία δημιουργεί το πρώτο block (genesis block) στο Python Blockchain Prototype καθώς και στην Neo4j βάση:

```
def
createGenesisBlock(self,database,db_blocks,db_transactions,db_
users) :

initial_amount = 20000     #Initial Amount of tokens ever
           #available in the p2p network
```

```
    timestamp                                                    =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-
%d %H:%M:%S')

    hash_object = hashlib.sha256(str(timestamp)) #Creating a
unique Transaction Hash from the unique timestamp

    hex_dig = hash_object.hexdigest()

    self.node_transactions.append({
    'from': 'reward_tx',
    'to':hashlib.sha256("User1").hexdigest(), #The node that
creates/mines the Genesis Block is User1.
    'amount':initial_amount,

    'transactionHash':hex_dig,
    'description':'genesis_block'
    })



    genesis_block        =        Block(0,        timestamp,
self.node_transactions,'0')


    # ------- Update the Neo4j Graph Database with the Genesis
                #Block --------------------
    appended_block = database.nodes.create(name = "Genesis
Block", id = 0, nonce = 0,timestamp = timestamp)
    db_blocks.add(appended_block)

    appended_transaction   =   database.nodes.create(name   =
"Transaction",      id      =      hex_dig,      amount      =
initial_amount,description="genesis_block")
    db_transactions.add(appended_transaction)

    appended_from  =  database.nodes.create(name  =  "From",
address = 'reward_tx')
    appended_to = database.nodes.create(name = "To", address
=hashlib.sha256("User1").hexdigest())
    db_users.add(appended_from, appended_to)

    appended_transaction.relationships.create(hex_dig        ,
appended_block)     #Genesis_Block     <---(transactionHash)---
Transaction
    appended_from.relationships.create('reward_tx',appended_tr
ansaction)
    appended_to.relationships.create(hashlib.sha256("User1").h
exdigest(),appended_transaction)

    self.node_transactions = [] # Reset the current block
transactions --> Waiting for new transactions


    return genesis_block
```

Στον παραπάνω κώδικα αρχικά ορίζουμε τα δεδομένα που θα έχει το Genesis Block, όπως για παράδειγμα τον αριθμό των tokens τα οποία θα είναι διαθέσιμα στο ευρύτερο δίκτυο ομότιμων κόμβων (τα οποία μάλιστα τα λαμβάνει και αυτός που δημιουργεί το Genesis Block για testing σκοπούς), ένα μοναδικό αριθμό κατακερματισμού ο οποίος δημιουργείται από το μοναδικό timestamp και τα εκάστοτε transactions μαζί με όλες τις ανάλογες πληροφορίες (δηλαδή τα initial tokens, source address, destination address κτλ.). Αφού οριστούν όλα αυτά, το πρώτο block μπαίνει στην Python δομή και εν συνεχεία ενημερώνεται και η κατανεμημένη Neo4j βάση. Η συγκεκριμένη διαδικασία ενημέρωσης γίνεται σε 3 φάσεις αφού όπως προαναφέρθηκε και η βάση είναι τριεπίπεδη (Blocks – Transactions – Users). Αρχικά, εισάγεται το Genesis Block με την αντίστοιχη πληροφορία, ύστερα η συναλλαγή με την οποία ορίζεται το συνολικό πλήθος των διαθέσιμων tokens και τέλος σαν τελευταίο επίπεδο εισάγονται οι Users οι οποίοι εμπλέκονται στην συναλλαγή αυτή (στην περίπτωση μας ο χρήστης ο οποίος δημιούργησε το πρώτο block). Όλοι αυτοί οι κόμβοι, συνδέονται με σχέσεις: οι συναλλαγές συνδέονται με τα blocks μέσω των transaction hashes και οι χρήστες συνδέονται με τις συναλλαγές μέσω των δημοσίων κλειδιών τους.

Προφανώς για να λειτουργήσει η παραπάνω δομή, υλοποιήθηκε και ο θεμέλιος λίθος του blockchain το οποίο είναι ο αλγόριθμος consensus. Όπως προαναφέρθηκε στην αντίστοιχη ενότητα της θεωρίας ο συγκεκριμένος αλγόριθμος ο οποίος χρησιμοποιείται ευρέως στα κατανεμημένα συστήματα, εφαρμόζει τέλεια και σε ένα δίκτυο blockchain. Συγκεκριμένα, αυτός ο αλγόριθμος χρησιμοποιείται στο δίκτυο blockchain με σκοπό όλοι οι συνδεδεμένοι κόμβοι να έχουν ομοφωνία όσον αφορά την δομή του blockchain. Δηλαδή με απλά λόγια, χρησιμοποιείται με σκοπό όλοι οι κόμβοι να έχουν το ίδιο ακριβώς blockchain, ώστε να βλέπουν και την ίδια πληροφορία.

Σύμφωνα με τα παραπάνω λοιπόν ένα εύλογο ερώτημα το οποίο ίσως δημιουργηθεί είναι το εξής: Αν υπάρχουν για παράδειγμα 1000 συνδεδεμένοι κόμβοι στο δίκτυο και ο καθένας για κάποιον λόγο έχει διαφορετική δομή blockchain, ποιο απόλα είναι σωστό; Για χάρη της παρούσας διπλωματικής εργασίας σημειώνεται πως σωστό blockchain θεωρείται αυτό που έχει τον μεγαλύτερο αριθμό blocks. Η συγκεκριμένη παραδοχή είναι εμπνευσμένη από το κριτήριο longest-chain του Bitcoin, το οποίο χρησιμοποιείται κυρίως για αποφυγή κακόβουλων επιθέσεων. Οπότε αυτή η παραδοχή χρησιμοποιείται σε συνδυασμό με τα

κατανεμημένα πρωτόκολλα συναίνεσης τα οποία κατασκεύασα και θα αναλυθούν εκτενώς παρακάτω. Παρακάτω παρουσιάζεται ο αντίστοιχος κώδικας για αυτή την διαδικασία ελέγχου ορθότητας:

```
    class Consensus(Resource):

    def get(self):
    peer_chains                                                    =
my_blockchain.get_peers_blockchain(my_blockchain)

    if not peer_chains :
    response = {
    'message': 'No available peer at the moment! Please add
some peers...'
    }
    return jsonify(response)

    # Consensus Algorithm

    longest_chain = max(peer_chains, key=len)
    if len(my_blockchain.chain) >= len(longest_chain):
    response = {

    'message':   'We   have   the   latest   version   of   the
blockchain...',
    'chain': my_blockchain.get_serialized_chain()
    }

    else:
    my_blockchain.chain                                             =
[my_blockchain.get_data_blockchain(block)    for    block    in
longest_chain]
    response = {
    'message': 'The blockchain is synced. You have the latest
version...',
    'chain': my_blockchain.get_serialized_chain()
    }
    return jsonify(response)
```

Ο παραπάνω κώδικας (κλάση Python) ο οποίος εκτελείται σε κάθε κόμβο κάνει ακριβώς τα παραπάνω τα οποία περιγράφησαν και συγκεκριμένα μέσω αυτής της γραμμής: `if len(my_blockchain.chain) >= len(longest_chain)`.
Δηλαδή, ο κόμβος που τρέχει τον κώδικα ρωτάει όλους τους peers τι blockchain έχουν μέσω της συνάρτησης `get_peers_blockchain`. Οπότε, παίρνοντας όλες τις απαντήσεις ελέγχει τα μήκη των δομών μέσω της παραπάνω συνθήκης και κρατάει το μεγαλύτερο σε μήκος blockchain. Εν

συνεχεία αυτό το κάνουν όλοι οι κόμβοι και έτσι επιτυγχάνεται η ικανοποίηση της συνθήκης όλοι οι κόμβοι να έχουν ακριβώς το ίδιο αντίγραφο του blockchain.

Τέλος, το τελευταίο κομμάτι του πηγαίου κώδικα είναι ο ορισμός των πόρων και των αντίστοιχων κλάσεων (*Flask Resources*), όπου εκεί εξυπηρετούνται τα http αιτήματα (get & post requests) των κόμβων.

Θα αναφερθούν απλώς ποια αιτήματα (Flask URLs) μπορούν να εξυπηρετηθούν (με τη βοήθεια της κατανεμημένης βάσης δεδομένων Neo4j) καθώς ο αναλυτικός κώδικας τους θα παρατεθεί στον παράρτημα.

- */getblockchain* : Ο κόμβος μπορεί να ζητήσει να δει όλη την δομή του blockchain και την πληροφορία η οποία είναι αποθηκευμένη μέσα σε αυτό.
- */create_transaction* : Αίτημα δημιουργίας συναλλαγής προς δημοσίευση στο επόμενο block.
- */addnode* : Αίτημα ώστε ο κόμβος να προσθέσει σαν peer κάποιον άλλον κόμβο.
- */getnodes* : Αίτημα ώστε ο κόμβος να λάβει σαν απάντηση όλους τους ομότιμους κόμβους με τους οποίους είναι συνδεδεμένος.
- */getblock* : Αίτημα ώστε ο κόμβος να δει την πληροφορία ενός συγκεκριμένου block.
- */getlatestblock* : Αίτημα ώστε ο κόμβος να δει το πιο πρόσφατα δημοσιευμένο block.
- */consensus* : Έλεγχος για μεγαλύτερο μήκος blockchain σε σχέση με τους άλλους ομότιμους κόμβους.
- */findbalance* : Αίτημα ώστε ο κόμβος να μάθει πόσα tokens έχει στην κατοχή του.
- */mine* : Συμμετοχή στον Proof Of Work Consensus Algorithm
- */pos* : Συμμετοχή στον Proof Of Stake Consensus Algorithm
- */pom* : Συμμετοχή στον Proof Of Motion Consensus Algorithm
- */traverse* : Ένας κόμβος με την βοήθεια της Neo4j μπορεί να δει σε τι transactions έχουν συμμετάσχει 2 κόμβοι, λαμβάνοντας όλα τα αντίστοιχα transaction hashes.

4.2.1 Υλοποίηση κατανεμημένων πρωτοκόλλων συναίνεσης με χρήση της Neo4j

Στο παρών κεφάλαιο θα αναλυθούν οι υλοποιήσεις των κατανεμημένων πρωτοκόλλων συναίνεσης, τα οποία συνδυάζουν τις δύο τεχνολογίες της παρούσας διπλωματικής εργασίας δηλαδή την Neo4j Graph Database και το blockchain. Παρακάτω λοιπόν παρουσιάζονται οι εξής αλγόριθμοι : Proof Of Work, Proof Of Stake [47] καθώς και ένα νέο πρωτόκολλο το οποίο το ονομάσαμε Proof Of Motion.

A) Proof Of Work

Καταρχάς, στο συγκεκριμένο πρωτόκολλο οι ομότιμοι κόμβοι του blockchain δικτύου μπορούν να συμμετάσχουν στον "αγώνα" για τη λύση ενός μαθηματικού κρυπτογραφικού παζλ. Συγκεκριμένα ο πρώτος κόμβος ο οποίος θα βρει σωστά έναν συγκεκριμένο αριθμό -ονομάζεται nonce-, θεωρείται ο νικητής του διαγωνισμού και είναι και αυτός ο οποίος τελικά θα δημοσιεύσει το νέο block στο blockchain. Για όλη αυτή τη διαδικασία, η οποία ονομάζεται εξόρυξη – mining, όπως είναι λογικό σπαταλούνται υπολογιστικοί πόροι και πολλή ενέργεια, αφού ο τρόπος με τον οποίο κάθε κόμβος βρίσκει το εκάστοτε nonce, είναι δοκιμάζοντας τυχαίους αριθμούς μέσω αλγορίθμου "ωμής βίας" (bruteforce). 'Ετσι λοιπόν προκύπτει εύλογα το ερώτημα: Γιατί να σπαταλήσει τους υπολογιστικούς του πόρους κάποιος κόμβος; Η απάντηση είναι η εξής: Διότι ο νικητής αυτού του μαθηματικού διαγωνισμού κερδίζει κάθε φορά κάποια tokens σαν επιβράβευση από το σύστημα και αυτό έχει ως συνέπεια η όλη διαδικασία να λειτουργεί ως κίνητρο για τους χρήστες.

Αρχικά λοιπόν ο κόμβος ο οποίος θέλει να συμμετάσχει στην διαδικασία της εξόρυξης/mining , πρέπει να κάνει ένα http αίτημα στον τοπικό Python Flask Server από τον οποίο και θα εξυπηρετηθεί. Η κλάση η οποία εξυπηρετεί αυτό το αίτημα είναι η εξής:

```
class mineBlock(Resource):

def post(self):
 info = request.get_json()
 block =  my_blockchain.mine_block(info.get('miner'))
```

```
    response = {
      'message':'Successfully Mined'
     }
     return jsonify(response)
```

Ουσιαστικά η παραπάνω κλάση η οποία εξυπηρετεί το αίτημα του κόμβου για εξόρυξη, καλεί την συνάρτηση *mine_block* της κλάσης *my_blockchain,* παιρνώντας για παράμετρο την διεύθυνση του εκάστοτε κόμβου.

Οπότε λόγω του αιτήματος καλείται σειριακά η προαναφερθείσα συνάρτηση, της οποίας ο κώδικας είναι ο εξής:

```
    def mine_block(self, miner):

        self.create_transaction(from1 ='reward_tx', to =
hashlib.sha256(str(miner)).hexdigest()    ,    amount    =
10,description='coinbase_transaction')

    block                                                =
self.create_block(hashlib.sha256(str(miner)).hexdigest())

    return vars(block)
```

Παραπάνω παρατηρούμε λοιπόν ότι αρχικά δημιουργείται δυναμικά μια συναλλαγή η οποία παίρνει τις εξής παραμέτρους:

- *Αποστολέας* : reward_tx , δηλαδή το δίκτυο
- *Παραλήπτης*: Ο κόμβος ο οποίος ζήτησε να συμμετέχει στην διαδικασία της εξόρυξης.
- *Ποσόν:* Είναι πάντα σταθερό και ίσο με 10 tokens
- *Περιγραφή :* coinbase_transaction

Προσοχή! Αυτή η συναλλαγή ΔΕΝ πραγματοποιείται ακόμα, αφού το μόνο που κάνει η παραπάνω συνάρτηση είναι να την αρχικοποιήσει. Αυτό συμβαίνει διότι όπως είπαμε παραπάνω αν ο κόμβος ο οποίος κάνει εξόρρυξη καταφέρει να λύσει πρώτος το κρυπτογραφικό παζλ και να βρει τον σωστό αριθμό, τότε και μόνο τότε θα πρέπει να λάβει το ποσό της παραπάνω συναλλαγής (αφού λειτουργεί σαν κίνητρο). Οπότε μέσω της

συνάρτησης create_transaction, η συναλλαγή αυτή μπαίνει στο μπλοκ το οποίο, αν ο κόμβος καταφέρει να βρει το nonce, θα δημοσιεύσει.
Ύστερα, ο κόμβος καλώντας την συνάρτηση create_block, δημιουργεί το block προς δημοσίευση σε περίπτωση που κερδίσει και ξεκινάει να ψάχνει τον μοναδικό αριθμό του κρυπτογραφικού puzzle. Ο κώδικας της create_block είναι ο παρακάτω:

```
def create_block(self,miner) :

        ts = time.time()
    block = Block(index = len(self.chain), timestamp =
datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:
%S'), data = self.node_transactions)
        block.previousHash = self.getLatestBlock().hash

        merkletree = MerkleTrees.MerkTree()
        merkletree.listoftransaction =
str(self.node_transactions)
        merkletree.create_tree()
        block.hash_merkle_block=merkletree.Get_Root_leaf()

        print("Mining Block...")
        block.mineBlock(self.difficulty,miner)

        time.sleep(2) # Thread is sleeping for 2 seconds, for
good synchronization !

        number_of_blocks = "MATCH (b:Blocks) return COUNT(b)"
        results = db.query(number_of_blocks,returns=(int))

        for r in results:
    blocks_number = r[0]
    print("The Blocks Number currently in the Neo4j DB are :
"+str(blocks_number))

if(len(my_blockchain.chain)<blocks_number):

    print("You didn't solved first the cryptographic puzzle!
          Sorry! Next time...")
    peer_chains =
     my_blockchain.get_peers_blockchain(my_blockchain)
    longest_chain = max(peer_chains, key=len)
    my_blockchain.chain =
[my_blockchain.get_data_blockchain(block1) for block1 in
longest_chain]
    return block

else:
    print("You are the winner!!!")
    self.node_transactions = []
    self.chain.append(block)
    update_neo4j =
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users)
    return block
```

Παραπάνω λοιπόν παρατηρούμε ότι ο εκάστοτε κόμβος δημιουργεί το μπλοκ ορίζοντας όλα τα απαραίτητα πεδία όπως: index, timestamp, transactions data, Merkle Tree root, hash, previous_hash. Αυτό το μπλοκ θα δημοσιευτεί μόνο αν ο κόμβος καταφέρει να βρει το nonce πρώτος στο δίκτυο και γι'αυτόν τον λόγο καλείται η συνάρτηση *block.mineBlock(self.difficulty, miner),* όπου και αποτελεί την ουσία του αλγορίθμου Proof Of Work, αφού έτσι ξεκινάει να ψάχνει την λύση του κρυπτογραφικού παζλ. Ο αντίστοιχος κώδικας για την εύρεση του κατάλληλου nonce φαίνεται παρακάτω:

```
  def mineBlock(self, difficulty,miner) :

     zero_array = ['0'] * (difficulty)
     while       (list(str(self.hash[:difficulty]))       !=
zero_array):
         self.nonce = self.nonce + 1
         self.hash = self.calculateHash()
        print("Block mined by the Miner : "+ str(miner)+
          " ,and the block hash is : " + self.hash +"\n")
         time.sleep(2)

      for transaction in my_blockchain.node_transactions:

         transaction_to = transaction['to']
         if transaction_to == "transaction_fee_miner":
              transaction['to'] = miner

def calculateHash(self) :

     my_string = (str(self.index) + str(self.previousHash)
+      str(self.timestamp)      +      str(self.data)      +
str(self.nonce)).encode('utf-8')
     hash_object = hashlib.sha256(my_string)
     hex_dig = hash_object.hexdigest()
     return(hex_dig)
```

Παραπάνω λοιπόν γίνεται η εξής διαδικασία εξόρυξης : Αρχικά λοιπόν μέσω της μεταβλητής "difficulty", η οποία για τις ανάγκες της παρούσας διπλωματικής παραμένει σταθερή και ίση με 5, ουσιαστικά ορίζει το πόσο "δύσκολο" είναι να βρεθεί ο αριθμός ο οποίος επιλύει το παζλ. Συγκεκριμένα τα βήματα του PoW αλγορίθμου μας είναι τα εξής:
1. Δημιουργία πίνακα #difficulty θέσεων γεμάτο με μηδενικά.
2. Στο self.hash τοποθετώ τον κατακερματισμένο αριθμό που προκύπτει από την κατακερματισμένη συνένωση των δεδομένων του block προς

δημοσίευση (συνάρτηση calculateHash), δηλαδή των : index, previousHash, timestamp, data καθώς και το nonce το οποίο αρχικοποιείται με 0. Δηλαδή με συμβολικό τρόπο έχουμε το εξής:
block_hash = hash_sha256(index|previousHash|timestamp|data|nonce).

Οπότε λοιπόν η έννοια της εξόρυξης ειπεισέρχεται ως εξής: Το hash το οποίο προκύπτει από την συνένωση αυτή είναι ένας δεκαεξαδικός αριθμός ο οποίος για να είναι αποδεκτός θέλουμε στην αρχή του να έχει όσα μηδενικά ορίζει το difficulty, δηλαδή 5. Ένα παράδειγμα αποδεκτού hash είναι το 00000a6f... Την αλλαγή στο hash παρατηρούμε από τις παραπάνω συναρτήσεις την προκαλεί μόνο η αλλαγή του nonce, αφού όλα τα άλλα προαναφερθέντα πεδία παραμένουν σταθερά. Οπότε ο κόμβος δοκιμάζει συνεχώς διάφορα nonce και κατακερματίζει την παραπάνω συμβολοσειρά κάθε φορά με το διαφορετικό nonce μέχρι να βρεθεί το σωστό το οποίο να δίνει τον κατάλληλο αριθμό μηδενικών. Η παραπάνω διαδικασία φαίνεται και από τις εξής τρείς γραμμές:

```
while (list(str(self.hash[:difficulty])) != zero_array):
     self.nonce = self.nonce + 1
     self.hash = self.calculateHash()
```

Όλη αυτή η διαδικασία των συνεχόμενων δοκιμών και κατακερματισμών απαιτεί υπολογιστική ισχύ από τους κόμβους, οι οποίοι και λόγω της υλοποίησης του PoW αλγορίθμου ονομάζονται miners. Ο πρώτος κόμβος ο οποίος θα καταφέρει να λύσει το παζλ είναι ο νικητής, οπότε ενημερώνει τους άλλους κόμβους για το νέο block το οποίο θα προστεθεί στο blockchain και ενημερώνεται και η κατανεμημένη Neo4j για την εισαγωγή του νέου block! Αυτό λαμβάνει χώρα στην create_block συνάρτηση του κόμβου μέσω του κώδικα:

```
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users).
```

Η συνάρτηση αυτή είναι ζωτικής σημασίας για την εφαρμογή και σημειώνεται πως είναι ακριβώς η ίδια και για τα 3 κατανεμημένα πρωτόκολλα συναίνεσης αφού ο τρόπος και η αρχιτεκτονική με βάση την οποία ενημερώνεται η Neo4j δεν αλλάζει. Το μόνο το οποίο διαφοροποιείται είναι η αλγοριθμική σκέψη στον τρόπο που κάποιος κόμβος δημοσιεύει το νέο block στην δομή του blockchain. Για λόγους εξοικονόμησης χώρου η συνάρτηση αυτή θα παρουσιαστεί μόνο σε αυτό το πρωτόκολλο συναίνεσης και όχι στα άλλα δύο, αφού ισχύουν ακριβώς τα ίδια: Αρχικά λοιπόν προστίθεται το block σαν οντότητα στην Neo4j και πρέπει να συνδεθεί με το προηγούμενο δημοσιευμένο block του blockchain μέσω του previousHash πεδίου. Οπότε μέσω των Cypher Queries και του API που προσφέρει η Neo4j, δημιουργώ ερώτημα στην βάση να παρέχει το hash του τελευταίου δημοσιευμένου block:

```
q = 'MATCH (u:Blocks) WHERE u.id='+str(previous_block_id)+'
RETURN u'
results = db.query(q, returns=(client.Node, str, client.Node))
```

Οπότε, μόλις το επιστρέψει η βάση το αποτέλεσμα δημιουργώ σχέση μεταξύ του block το οποίο μόλις δημοσίευσε ο κόμβος και του προηγούμενου του δυναμικά, χωρίς να πρέπει να ασχοληθεί ο εκάστοτε χρήστης!

```
appended_block.relationships.create(new_block.previousHash,r[0
]).
```

Ακριβώς με τον ίδιο τρόπο δημιουργούνται οι οντότητες των συναλλαγών και των χρηστών καθώς και των αντιστοίχων σχέσεων μεταξύ τους, δημιουργώντας κατ'αυτόν τον τρόπο την τριεπίπεδη αρχιτεκτονική της Neo4j η οποία αναπαριστά το blockchain. Ο κώδικας της συνάρτησης της updatedb φαίνεται παρακάτω:

```
def updatedb(self,database,db_blocks, db_transactions, db_users):
         new_block = self.chain[-1]
        block_index = new_block.index
         block_hash = new_block.hash
         block_nonce = new_block.nonce
         block_merkle = new_block.hash_merkle_block
         block_timestamp = new_block.timestamp
         appended_block    =    database.nodes.create(name    =    "Block    "
+str(block_index), id =      block_index,  hash  =  block_hash  ,  nonce  =
block_nonce, merkle =                   block_merkle,timestamp             =
block_timestamp)
         db_blocks.add(appended_block)


             '''**** Connecting Blocks with previous Hashes *********  '''
         previous_block = self.chain[-2]
         previous_block_id = previous_block.index
         print("Previous Block ID is : "+str(previous_block_id))
         q = 'MATCH (u:Blocks) WHERE u.id='+str(previous_block_id)+' RETURN
u'
         results = db.query(q, returns=(client.Node, str, client.Node))
         for r in results:
       appended_block.relationships.create(new_block.previousHash,r[0])


# -- Create the Transactions & Users of the Block in Neo4j (2nd Layer) -- #
         for transaction in new_block.data:
     transaction_hash = transaction['transactionHash']
     transaction_amount = transaction['amount']
     transaction_description = transaction['description']
     appended_transaction = database.nodes.create(name = "Transaction", id =
transaction_hash,     amount     =     transaction_amount,description     =
transaction_description)
     db_transactions.add(appended_transaction)

     # *************** Users of Each Transaction (3rd Layer) ***********
     transaction_from = transaction['from']
     transaction_to = transaction['to']
     appended_from   =   database.nodes.create(name   =   "From",   address   =
transaction_from)
     appended_to   =   database.nodes.create(name   =   "To",   address   =
transaction_to)
     db_users.add(appended_from, appended_to)
```

```
      # Connecting the transactions with the block.
        #(Appended_Transaction)-r->(Appended_block)

      appended_transaction.relationships.create(transaction_hash            ,
appended_block)

      #  Connecting  the  Users  with  the  corresponding  transactions.
        #(Transaction from/to)-r->(Appended_Transaction)

      appended_from.relationships.create(transaction_from,appended_transactio
n)
      appended_to.relationships.create(transaction_to,appended_transaction)

       return True
```

B) Proof Of Stake

Το Proof Of Work πρωτόκολλο, αν και ήταν το πρωτόκολλο συναίνεσης το οποίο προτάθηκε από το Bitcoin, φαίνεται πως αρχίζει και χάνει την αποκεντρωμένη φύση του και τείνει να λάβει μια ιδιαίτερα κεντρικοποιημένη. Συγκεκριμένα, όπως αναφέραμε πιο πάνω ο κόμβος-νικητής ορίζεται ουσιαστικά από το ποιος διαθέτει την μεγαλύτερη υπολογιστική ισχύ ώστε να κάνει περισσότερους κατακερματισμούς σε μικρότερο χρονικό διάστημα. Αυτό έχει ως συνέπεια, ο κόμβος που έχει τον καλύτερο υπολογιστή να νικάει συνεχώς και οι άλλοι κόμβοι να μην έχουν πλέον τόσες πιθανότητες. Αυτό το παρατηρήσαμε και στην εφαρμογή μας καθώς συνήθως ο κόμβος νικητής ήταν αυτός ο οποίος "έτρεχε" και στο καλύτερο μηχάνημα (με καλύτερη CPU, GPU κτλ.).

Για αυτόν τον λόγο λοιπόν προτάθηκαν νέα κατανεμημένα πρωτόκολλα συναίνεσης τα όποια ορίζουν με άλλα κριτήρια τον κόμβο τον οποίο θα δημοσιεύσει το επόμενο block. Ένα από αυτά είναι και το παρών το οποίο ονομάζεται Proof Of Stake. Το συγκεκριμένο πρωτόκολλο συναίνεσης, με βάση το οποίο οι κόμβοι σε ένα κατανεμημένο δίκτυο blockchain έρχονται σε ομοφωνία, λειτουργεί ως εξής: Ο δημιουργός του επόμενου block επιλέγεται με βάση τον "πλούτο"/αριθμό tokens που έχει στην κατοχή του και μια είδους τυχαιότητα. Πέρα από τον κλασικό ορισμό του PoS πρωτοκόλλου, μετά προτάθηκαν και διάφορες άλλες παραλλαγές όπως: Συνδυασμός πλούτου με ωριμότητα του πλούτου και τυχαιότητα, Casper-like PoS[48] και άλλα.

Στην παρούσα διπλωματική υλοποιήθηκε το κλασικό PoS το οποίο αποτελείται από τον πλούτο του κάθε κόμβου καθώς και από έναν βαθμό πιθανοτικής τυχαιότητας.

Αρχικά λοιπόν ο κόμβος ο οποίος θέλει να συμμετάσχει στην διαδικασία του PoS , πρέπει να κάνει ένα http αίτημα στον τοπικό Python Flask Server από τον οποίο και θα εξυπηρετηθεί. Η κλάση η οποία εξυπηρετεί αυτό το αίτημα είναι η εξής:

```
class choose_pos(Resource):

def get(self):
winner = my_blockchain.mine_pos()

#*** Update Neo4j Database ***
update_neo4j                                                                    =
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users)
if update_neo4j:
        print("Neo4j Graph Database is successfully updated!!!")

response = {
        'message':'The Proof Of Stake winner of PoS Protocol is : ',
        'user':winner
}
return jsonify(response)
```

Παρατηρούμε λοιπόν ότι μόλις γίνει το αίτημα από τον κόμβο, η παραπάνω κλάση που εξυπηρετεί το αίτημα (choose_pos) καλεί κατευθείαν την συνάρτηση mine_pos() η οποία και υλοποιεί όλη την λογική του συγκεκριμένου κατανεμημένου πρωτοκόλλου συναίνεσης. Αρχικά θα παρατεθεί ο πηγαίος κώδικς του πρωτοκόλλου και ύστερα θα αναλυθεί η αλγοριθμική σκέψη:

```
def mine_pos(self):
 # Step1 : Find all Users that have ever make a transaction - Cypher
  #Neo4j Query

    pos_query ='match (u:Users)-[r]-(t:Transactions) with type(r) as
posusers, collect(r) as nodes where size(nodes)>0 return [n in nodes |
type(n)][0] order by size(nodes) desc'
    pos_addresses = db.query(pos_query, returns=(str))
    print('The amount of Users in the Network is : '+
str(len(pos_addresses)-1))
    number_of_users = len(pos_addresses)-1
    temp = number_of_users #
    temp_tokens = 0

  #Step2 : Create 2d array --> We create a list that contains
  #len(pos_addresses)-1 lists and each of them contains 2 elements
  #(address,balance_temp)
    pos_table = [['' for x in range(2)] for y in
range(len(pos_addresses)-1)]
```

```
      for address in pos_addresses:
              if(address[0]!='reward_tx'):
                     print address[0]

                     pos_table[number_of_users-temp][0] = address[0]
                     user_tokens = self.find_current_balance(str(address[0]))

                     pos_table[number_of_users-temp][1]                    =
user_tokens+temp_tokens
                     temp_tokens = temp_tokens + user_tokens
                     temp -= 1

       #Step3 : Produce a random number between 1 -
       #pos_table[number_of_users][1] to opoio taytizetai me temp_tokens -->
       #Number that defines the winner!

          lucky_number = randint(1,temp_tokens)
          print('The PoS Lucky Number is : '+str(lucky_number))

          temp1 = number_of_users
      while(temp1!=0):
       if(lucky_number <= pos_table[number_of_users-temp1][1]):
              pos_winner = str(pos_table[number_of_users-temp1][0])
              print('The PoS winner is : ' + str(pos_table[number_of_users-
temp1][0]))
              break
       temp1 -= 1

      for transaction in self.node_transactions:
       transaction_to = transaction['to']
       if transaction_to == "transaction_fee_miner":
       transaction['to'] = pos_winner
```

Η αλγοριθμική λογική του παραπάνω κώδικα είναι η εξής:

1. Εκτελώντας ένα Cypher Query στην Neo4j, σύμφωνα πάντα με την παρούσα δομή του blockchain την οποία περιγράφει, αρχικά βρίσκω όλους τους Users (τις διευθύνσεις τους) οι οποίοι έχουν κάνει τουλάχιστον μια συναλλαγή. Το Cypher Query όπως φαίνεται και από τον παραπάνω κώδικα είναι το εξής:

```
match (u:Users)-[r]-(t:Transactions) with type(r) as
posusers, collect(r) as nodes where size(nodes)>0 return
[n in nodes | type(n)][0] order by size(nodes) desc
```

2. Με βάση τα αποτελέσματα τα οποία θα μας επιστρέψει το συγκεκριμένο query, δημιουργώ έναν πίνακα δύο διαστάσεων, όπου η μια διάσταση (γραμμές) εκφράζεται από το πλήθος των διαφορετικών διευθύνσεων /χρηστών και η άλλη (στήλες) είναι 2. Οπότε στην πρώτη στήλη συμπληρώνω αρχικά τις διευθύνσεις των διαφορετικών χρηστών και στην δεύτερη στήλη το αντίστοιχο πόσο tokens που έχει μέχρι εκείνη

την στιγμή ο καθένας στην κατοχή του. Προσοχή όμως, διότι η δημιουργία του δισδιάστατου πίνακα είναι ιδιαίτερη.
Συγκεκριμένα, έστω ότι ισχύουν τα εξής, προερχόμενα από το query:
- Ο User1 (0xa....) έχει 1968 tokens.
- Ο User2 (0xb...) έχει 35 tokens.
- Ο User3 (0xc...) έχει 78 tokens.
Οπότε ο δισδιάστατος πίνακας θα είναι ως εξής:

User1 (0xa...)	1968
User2 (0xb...)	2004 (=1969+35)
User3 (0xc...)	2083 (=2005+78)

3. Το συγκεκριμένο βήμα αποτελεί την ουσία του συγκεκριμένου κατανεμημένου πρωτοκόλλου συναίνεσης. Ουσιαστικά ο παραπάνω πίνακας περιέχει εύρη τιμών για κάθε User. Δηλαδή ο User1, ο οποίος έχει και τα περισσότερα tokens έχει το μεγαλύτερο range από όλους τους άλλους [1-1968]. Αντίστοιχα ο User2 [1969-2004] και ο User3 [2005-2083]. Οπότε αυτό που κάνουμε μετά στον αλγόριθμο είναι το εξής:
Παράγουμε έναν τυχαίο ακέραιο ο οποίος είναι μεταξύ των αριθμών 1 και 2083 δηλαδή μεταξύ των άκρων του πίνακα και αυτό επιτυγχάνεται μέσω της Python εντολής : `lucky_number = randint(1,temp_tokens).` Αυτό έχει ως συνέπεια ότι ο ακέραιος ο οποίος θα παραχθεί τυχαία από το Λειτουργικό Σύστημα να είναι πιο πιθανόν να ανήκει στο μεγαλύτερο εύρος, δηλαδή στον χρήστη με τα περισσότερα tokens (δηλαδή στο παράδειγμα μας στον User1). Με αυτόν τον τρόπο λοιπόν υλοποιούμε κατά γράμμα το Proof Of Stake συνδυάζοντας πλούτο με τυχαιότητα.

4. Αφού λοιπόν επιλεχθεί με βάση τον πλούτο και την τυχαιότητα ο νικητής, δημιουργεί το μπλοκ το οποίο θα δημοσιεύσει περιέχοντας μέσα όλες τις εκάστοτε συναλλαγές που περιέχονται στο transaction pool του. Σε αυτό το συγκεκριμένο πρωτόκολλο ο νικητής του PoS δεν λαμβάνει κάποιο reward σαν το PoW, παρά μόνο παίρνει τους φόρους από τις συναλλαγές. Συγκεκριμένα θεωρούμε ότι για να γίνει μια συναλλαγή ένα token θα πρέπει να πληρωθεί σαν φόρος ο οποίος θα ληφθεί από τον PoS Winner. Προφανώς εδώ δεν έχουμε έννοιες όπως κρυπτογραφικό παζλ ή nonce. Τέλος, με τον πηγαίο κώδικα ο οποίος παρουσιάστηκε στο Proof Of Work πρωτόκολλο ενημερώνεται δυναμικά και η κατανεμημένη Neo4j βάση δεδομένων για το νέο block, μέσω δηλαδή της συνάρτησης updatedb().

C) Proof Of Motion

Το παραπάνω κατανεμημένο πρωτόκολλο συναίνεσης (Proof Of Stake) παρατηρήσαμε ότι τείνει και αυτό να γίνει κεντρικοποιημένο εφόσον τις περισσότερες φορές, νικητής είναι αυτός που έχει τα πιο πολλά tokens παρά τον παράγοντα της τυχαιότητας που υπάρχει. Παρόλα αυτά, κάποιες τροποποιήσεις στο PoS, όπως για παράδειγμα η προσθήκη του παράγοντα της ωρίμανσης των tokens – δηλαδή για πόσο καιρό τα έχει κάποιος στην κατοχή του- ξαναδείνει στο πρωτόκολλο αυτό τον αποκεντρωμένο χαρακτήρα του.

Τώρα, λοιπόν θα παρουσιάσω ένα νέο κατανεμημένο πρωτόκολλο συναίνεσης το οποίο προσπαθεί να συμβαδίσει με τον αποκεντρωμένο χαρακτήρα του blockchain και μάλιστα συνδυάζει και την τεχνολογία της Neo4j βάσης δεδομένων γράφου. Συγκεκριμένα η φιλοσοφία του πρωτοκόλλου είναι η εξής : Καταρχάς, ένας χρήστης ο οποίος έχει κάνει πολλές συναλλαγές στο δίκτυο blockchain δεν τον συμφέρει να λειτουργήσει κακόβουλα και όλες αυτές οι συναλλαγές του να χαθούν ή να "αλλάξουν". Οπότε, εφόσον δεν υπάρχει μεγάλη πιθανότητα να λειτουργήσει κακόβουλα είναι ένας καλός υποψήφιος χρήστης ώστε να δημοσιεύσει το νέο block με ασφάλεια και να ενημερώσει τους υπόλοιπους χρήστες. Άρα, το Proof Of Motion (PoM) βασίζεται στην "κινητικότητα" του χρήστη, δηλαδή στο πόσο ενεργός είναι στο δίκτυο. Φυσικά, σε αυτό το ερώτημα – πόσο ενεργός είναι – μας βοηθάει με την γρήγορη προσπέλαση στα δεδομένα του blockchain που μας προσφέρει η Neo4j Graph Database.

Αρχικά λοιπόν ο κόμβος ο οποίος θέλει να συμμετάσχει στην διαδικασία του PoM , πρέπει να κάνει ένα http αίτημα στον τοπικό Python Flask Server από τον οποίο και θα εξυπηρετηθεί. Η κλάση η οποία εξυπηρετεί αυτό το αίτημα είναι η εξής:

```
class choose_pom(Resource):
     def get(self):
     winner = my_blockchain.mine_pom() # Call function to implement PoA
Protocol and get the winner

     #*** Update Neo4j Database - http://localhost:7474 *****
     update_neo4j                                                    =
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users)
     if update_neo4j:
             print("Neo4j Graph Database is successfully updated!!!")
     #*************************************************************
```

```
response = {
        'message':'The Proof of Motion winner is : ',
        'user':winner
}
return jsonify(response)
```

Παρατηρούμε λοιπόν ότι μόλις γίνει το αίτημα από τον κόμβο, η παραπάνω κλάση που εξυπηρετεί το αίτημα (choose_pom) καλεί κατευθείαν την συνάρτηση mine_pom() η οποία και υλοποιεί όλη την λογική του συγκεκριμένου κατανεμημένου πρωτοκόλλου συναίνεσης. Αρχικά θα παρατεθεί ο πηγαίος κώδικς του πρωτοκόλλου και ύστερα θα αναλυθεί η αλγοριθμική σκέψη:

```
def mine_pom(self):
     # Step 1 : Find how many blocks to examine --> 60% of
     #the whole blockchain
     chain_length = len(my_blockchain.chain)
     last_blocks_float = float((chain_length*60)/100.0)
     examine_blocks = math.ceil(last_blocks_float)
     print examine_blocks

     # Step 2 : In the last examine_blocks [60%] we should
     #find how many transactions did each user
     pom_query    ='MATCH    (e1:Blocks)-[rels*1..2]-(e2:Users)
where e1.id>='+str(chain_length-examine_blocks)+' WITH e1, e2,
rels, extract(rel IN rels | startNode(rel)) AS startNodes,
extract(rel IN rels | endNode(rel)) AS endNodes, range(1,
size(rels)-1)  AS  indexes WITH  e1,  e2,  rels,  startNodes,
endNodes, indexes, startNodes[0] AS start UNWIND indexes AS i
WITH e1, e2, rels, e1 = start as isOutFirst, (endNodes[i-1] =
startNodes[i] OR startNodes[i-1] = startNodes[i]) AS isOut
WITH e1, e2, rels, isOutFirst, collect(isOut) AS isOuts WITH
e1,  e2,  rels,  [isOutFirst]  +  isOuts  AS  isOuts,  range(0,
size(rels)-1)  AS  indexes2  UNWIND  indexes2  AS  i  RETURN
e2.address,count(e2.address) order by count(e2.address) desc'

     # Step3 : Find a randomized algorithm, to choose the
     #winner of Proof Of Motion
     pom_users = db.query(pom_query, returns=(str,int))
         temp = len(pom_users)
         temp_transactions = 0

         print len(pom_users)
          pom_table  =  [['' for  x  in  range(2)]  for  y  in
range(len(pom_users))]
         for user in pom_users:
           if(user[0]!='reward_tx'):
                pom_table[len(pom_users)-temp][0] = user[0]
                pom_table[len(pom_users)-temp][1]            =
user[1]+temp_transactions
```

```
      temp_transactions = temp_transactions + user[1]
                  temp -= 1

            print user[0]
            print user[1]


# Step4 : Produce a random number & find the PoA winner
      lucky_number = randint(1,temp_transactions)
      print('The PoM Lucky Number is : '+str(lucky_number))

      temp1 = len(pom_users)
      while(temp1!=0):
            if(lucky_number  <=  pom_table[len(pom_users)-temp1]
[1]):
                  pom_winner   =   str(pom_table[len(pom_users)-
temp1][0])
                  print('The   PoM   winner   is   :   '   +
str(pom_table[len(pom_users)-temp1][0]))
                  break
            temp1 -= 1


    for transaction in self.node_transactions:
            transaction_to = transaction['to']
            if transaction_to == "transaction_fee_miner":
                  transaction['to'] = pom_winner


    block = self.pom_create_block(pom_winner)

    return pom_winner
```

Η αλγοριθμική λογική του παραπάνω κώδικα είναι η εξής:
1. Γενικότερα έχει νόημα να εξετάσω ένα ‘παράθυρο’ του blockchain ώστε να δω ποιος χρήστης είναι ενεργός, και όχι ολόκληρη την δομή του blockchain. Αυτό η σκέψη βασίζεται καταρχάς στην αποδοτικότητα της εφαρμογής – δεν είναι αποδοτικό να ελέγχουμε για κάθε νέο μπλοκ ολόκληρο το blockchain – και επίσης μας ενδιαφέρει να δούμε ποιος χρήστης είναι ενεργός στα τελευταία blocks και όχι στα πρώτα. Η έννοια του “ενεργός” βασίζεται στο πόσες συναλλαγές έχει κάνει. Γι’αυτόν τον λόγο οι πρώτες γραμμές υπολογίζουν το 60% του συνολικού πλήθους των blocks (με στρογγυλοποίηση προς τα πάνω) και ελέγχονται τα αντίστοιχα blocks από το τελευταίο προς το Genesis Block.

2. Εκτελώντας ένα Cypher Query στην Neo4j, σύμφωνα πάντα με την παρούσα δομή του blockchain την οποία περιγράφει, βρίσκω για τον συγκεκριμένο αριθμό blocks που υπολόγισα στο βήμα 1 ποιοι Users

έχουν κάνει συναλλαγές και μάλιστα πόσες έχει κάνει ο καθένας επιστρέφοντας τα αποτελέσματα του query σε φθίνουσα σειρά. Το Cypher query το οποίο κάνει αυτή την δουλειά στον παραπάνω πηγαίο κώδικα είναι το εξής:

```
MATCH (e1:Blocks)-[rels*1..2]-(e2:Users) where
e1.id>='+str(chain_length-examine_blocks)+' WITH e1, e2, rels,
extract(rel IN rels | startNode(rel)) AS startNodes,
extract(rel IN rels | endNode(rel)) AS endNodes, range(1,
size(rels)-1) AS indexes WITH e1, e2, rels, startNodes,
endNodes, indexes, startNodes[0] AS start UNWIND indexes AS i
WITH e1, e2, rels, e1 = start as isOutFirst, (endNodes[i-1] =
startNodes[i] OR startNodes[i-1] = startNodes[i]) AS isOut
WITH e1, e2, rels, isOutFirst, collect(isOut) AS isOuts WITH
e1, e2, rels, [isOutFirst] + isOuts AS isOuts, range(0,
size(rels)-1) AS indexes2 UNWIND indexes2 AS i RETURN
e2.address,count(e2.address) order by count(e2.address) desc
```

3. Με βάση τα αποτελέσματα τα οποία θα μας επιστρέψει το συγκεκριμένο query, δημιουργώ έναν πίνακα δύο διαστάσεων, όπου η μια διάσταση (γραμμές) εκφράζεται από το πλήθος των διαφορετικών διευθύνσεων /χρηστών και η άλλη (στήλες) είναι 2. Οπότε στην πρώτη στήλη συμπληρώνω αρχικά τις διευθύνσεις των διαφορετικών χρηστών και στην δεύτερη στήλη το πλήθος των συναλλαγών που έχει κάνει ο εκάστοτε χρήστης. Προσοχή όμως, διότι η δημιουργία του δισδιάστατου πίνακα είναι ιδιαίτερη.
Συγκεκριμένα, έστω ότι ισχύουν τα εξής, προερχόμενα από το query:
- Ο User1 (0xa....) έχει 115 transactions.
- Ο User2 (0xb...) έχει 35 transactions.
- Ο User3 (0xc...) έχει 15 transactions.

Οπότε ο δισδιάστατος πίνακας θα είναι ως εξής:

User1 (0xa....)	115
User2 (0xb...)	151 (= 116+35)
User3 (0xc...)	167 (= 152+15)

4. Το συγκεκριμένο βήμα αποτελεί την ουσία του συγκεκριμένου κατανεμημένου πρωτοκόλλου συναίνεσης. Ουσιαστικά ο παραπάνω πίνακας περιέχει εύρη τιμών για κάθε User. Δηλαδή ο User1, ο οποίος έχει και τις περισσότερες συναλλαγές, δηλαδή είναι πιο ενεργός στο δίκτυο, έχει το μεγαλύτερο range από όλους τους άλλους [1-115]. Αντίστοιχα ο User2 [116-151] και ο User3 [152-167]. Οπότε αυτό που κάνουμε μετά στον αλγόριθμο είναι το εξής:
Παράγουμε έναν τυχαίο ακέραιο ο οποίος είναι μεταξύ των αριθμών 1 και 167 δηλαδή μεταξύ των άκρων του πίνακα και αυτό επιτυγχάνεται

μέσω της Python εντολής : `lucky_number = randint(1,temp_transactions)`.
Αυτό έχει ως συνέπεια ότι ο ακέραιος ο οποίος θα παραχθεί τυχαία από το Λειτουργικό Σύστημα να είναι πιο πιθανόν να ανήκει στο μεγαλύτερο εύρος, δηλαδή στον πιο ενεργό χρήστη με τις περισσότερες συναλλαγές (δηλαδή στο παράδειγμα μας στον User1). Το συγκεκριμένο πρωτόκολλο δεν τείνει να γίνει κεντρικοποιημένο αφού κάθε φορά ο χρήστης ο οποίος έχει κάνει τις περισσότερες συναλλαγές ίσως είναι διαφορετικός, εφόσον και το "παράθυρο" του blockchain το οποίο εξετάζουμε αλλάζει κάθε φορά. Η τυχαιότητα στο συγκεκριμένο πρωτόκολλο παίζει σημαντικό ρόλο.

5. Αφού λοιπόν επιλεχθεί ο νικητής με βάση την ενεργητικότητα του στο δίκτυο ομότιμων κόμβων, δημιουργεί το μπλοκ το οποίο θα δημοσιεύσει περιέχοντας μέσα όλες τις εκάστοτε συναλλαγές που περιέχονται στο transaction pool του. Σε αυτό το συγκεκριμένο πρωτόκολλο ο νικητής του PoM δεν λαμβάνει κάποιο reward σαν το PoW, παρά μόνο παίρνει τους φόρους από τις συναλλαγές. Συγκεκριμένα θεωρούμε ότι για να γίνει μια συναλλαγή ένα token θα πρέπει να πληρωθεί σαν φόρος ο οποίος θα ληφθεί από τον PoM Winner. Προφανώς εδώ δεν έχουμε έννοιες όπως κρυπτογραφικό παζλ ή nonce. Τέλος, με τον πηγαίο κώδικα ο οποίος παρουσιάστηκε στο Proof Of Work πρωτόκολλο ενημερώνεται δυναμικά και η κατανεμημένη Neo4j βάση δεδομένων για το νέο block, μέσω δηλαδή της συνάρτησης updatedb().

5

Επίδειξη λειτουργικότητας εφαρμογής

Στο κεφάλαιο αυτό θα γίνει παρουσίαση της εφαρμογής και του τρόπου λειτουργίας της. Η παρουσίαση του κεφαλαίου αυτού αφορά το κομμάτι της εφαρμογής που αφορά τον χρήστη, ενώ στο προηγούμενο κεφάλαιο γίνεται παρυσίαση του τρόπου υλοποίησης της καθώς και της αλγοριθμικής σκέψης πίσω από αυτή.

5.1 Αρχικοποίηση κόμβων στο blockchain δίκτυο

Ξεκινώντας λοιπόν την περιγραφή της λειτουργικότητας της εφαρμογής, πρέπει να αρχικοποιήσουμε κάποιους κόμβους στο δίκτυο blockchain. Για τους σκοπούς της παρούσας διπλωματικής θα χρησιμοποιήσουμε 3 κόμβους με τους οποίους θα πειραματιστούμε.

Οπότε με τις εξής εντολές φλοιού (bash commands) θα αρχικοποιήσουμε τους παραπάνω κόμβους, οι οποίοι θα "τρέχουν" στο ίδιο μηχάνημα αλλά θα "ακούνε" σε διαφορετική Python Flask πόρτα.

```
python blockchain-cli.py -p 5000
python blockchain-cli.py -p 5001
python blockchain-cli.py -p 5002
```

Η απάντηση η οποία πρέπει να προέλθει από τον Flask Server κατά την αρχικοποίηση του κάθε κόμβου είναι η εξής:

```
Last login: Sat Sep  1 18:30:30 on ttys000
[GiorgosFragkos:Source Code$ python blockchain-PoW-cli.py -p 5001
 * Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
```

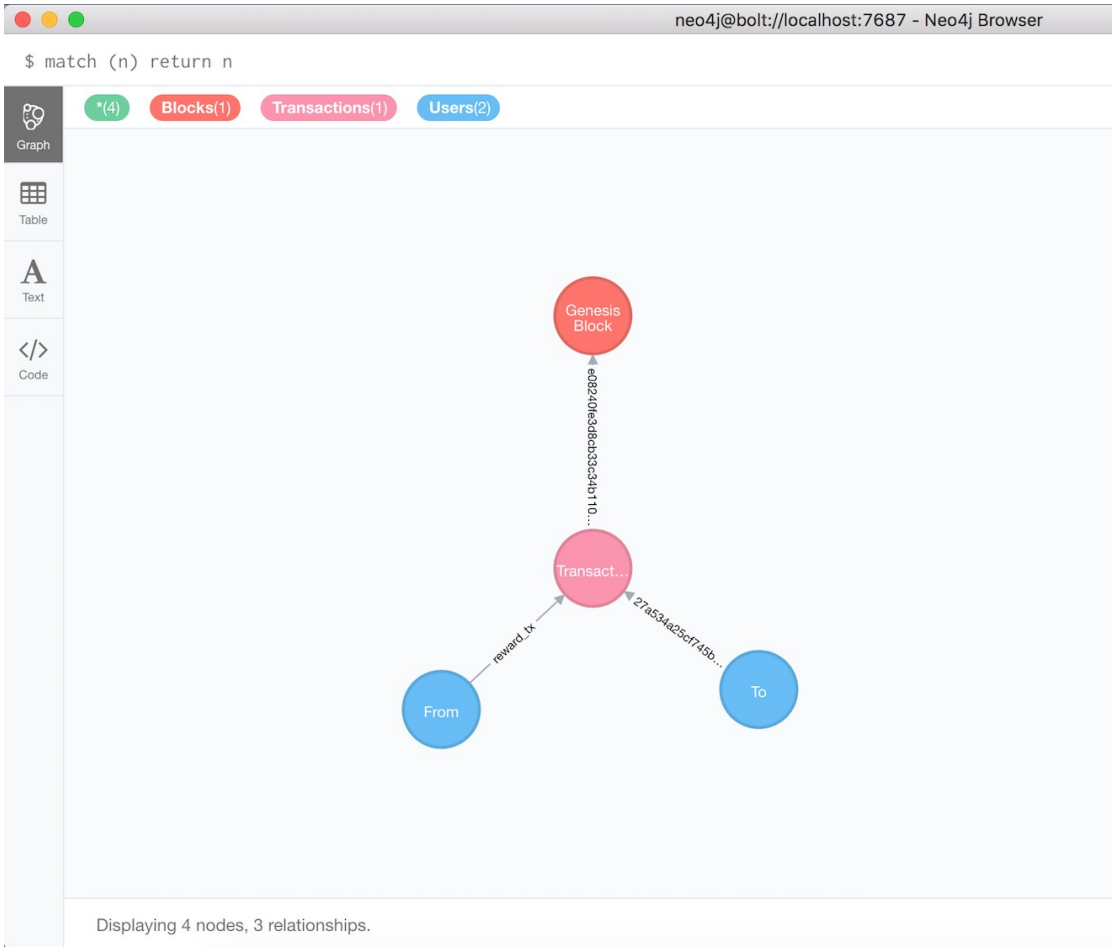
Εικόνα 5.1 Αρχικοποίηση πρώτου κόμβου User1

Εφόσον έχουν αρχικοποιηθεί οι κόμβοι αυτόματα, έχει αρχικοποιηθεί και η κατανεμημένη βάση δεδομένων γράφου Neo4j σε καθέναν από αυτούς τους κόμβους. Προφανώς ο πρώτος κόμβος που δημιουργείται στο δίκτυο, στην περίπτωση μας αυτός που ακούει στην

πόρτα 5000 (User1), είναι υπεύθυνος και για την δημιουργία του πρώτου block του blockchain. Οπότε στην τοπική Neo4j του κόμβου πρέπει να φαίνεται η ζητούμενη τριεπίπεδη αρχιτεκτονική η οποία περιγράφει το blockchain.
Μέσω του monitoring tool που προσφέρει η Neo4j, το Neo4j Browser[49], υλοποιούμε το παρακάτω Cypher Query έτσι ώστε να λάβουμε όλη την πληροφορία που υπάρχει αποθηκευμένη στην τοπική βάση του Neo4j:

```
match (n) return n
```

και το αποτέλεσμα που παίρνουμε είναι το εξής! **Σημειώνεται πως η δημιουργία και η αναπαράσταση του blockchain στην Neo4j γίνεται αυτόματα! Ο χρήστης απλώς παρακολουθεί τις διαθέσιμες πληροφορίες.**



Εικόνα 5.2 Οπτικοποίηση περιεχομένου Neo4j του User1

Στην παραπάνω εικόνα παρατηρούμε τα εξής (top-down approach):
1. Παρατηρούμε πως στο πρώτο επίπεδο της αρχιτεκτονικής έχει δημιουργηθεί το πρώτο block, το Genesis Block ως κόμβος του γράφου.
2. Στο δεύτερο επίπεδο έχει δημιουργηθεί μια συναλλαγή/Transaction node, η οποία ουσιαστικά δίνει ολόκληρο το διαθέσιμο amount στον User1 ο οποίος δημοσίευσε το πρώτο block στο Blockchain.
3. Στο τρίτο επίπεδο έχουν δημιουργηθεί οι χρήστες οι οποίοι συνδέονται με το παραπάνω transaction σε μορφή From/To.
4. Τέλος, οι παραπάνω κόμβοι συνδέονται μεταξύ τους μέσω σχέσεων σχηματίζοντας έτσι την τριεπίπεδη αρχιτεκτονική. Συγκεκριμένα ο κόμβος Transaction συνδέεται με το Block node μέσω του transaction hash (e0824...). Τα From/To nodes συνδέονται με το Transaction node μέσω των δημοσιών κλειδιών τους (εκτός και αν είναι reward transaction του δικτύου, που η σχέση τότε ονομάζεται reward_tx.).

Επιπροσθέτως, μέσω του Neo4j Browser, ο χρήστης μπορεί να περιηγηθεί σε πιο στοχευμένες πληροφορίες οι οποίες μπορούν να σχετίζονται με δεδομένα του Block και άλλων. Ειδικότερα, αν κάνει "κλικ" πάνω στα αντίστοιχα nodes μπορεί να πάρει τις εξής πληροφορίες:

Εικόνα 5.3 Πληροφορίες από Neo4j για το blockchain

5.2 Διασύνδεση κόμβων στο δίκτυο Blockchain

Παραπάνω λοιπόν δημιουργήσαμε τρεις κόμβους/Users και μάλιστα δημιουργήθηκε και το Genesis Block στην κατανεμημένη βάση Neo4j του User1. Εκτελώντας το ίδιο Cypher Query και στους υπόλοιπους χρήστες παρατηρούμε ότι και η δικιά τους τοπική Neo4j έχει ενημερωθεί αυτόματα για το καινούργιο block το οποίο δημοσίευσε ο User1. Φυσικά, μέσω του Flask API, αυτό μπορούμε να το ελέγξουμε και με ένα http get αίτημα σε κάθε κόμβο όπου ζητάμε την δομή του blockchain.
Ακολουθούν οι απαντήσεις του Flask Server καθενός από τους τρεις κόμβους:
User1:

- Http Get Request

```
GiorgosFragkos:Source Code$ curl "http://localhost:5000/getblockchain"
```

Εικόνα 5.4 User1 GetBlockchain request

- Http Response


```
{
  "chain": [
    {
      "data": [
        {
          "amount": 20000,
          "description": "genesis_block",
          "from": "reward_tx",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "7ce1f57b3d57d5ef7ed5db1dbee22198519c27402dec7d5c73144090a64ab076"
        }
      ],
      "hash": "273876760355b579529e2afc0820bdc5e079da23da873544568a579051f9717f",
      "hash_merkle_block": "",
      "index": 0,
      "nonce": 0,
      "previousHash": "0",
      "timestamp": "2018-09-01 19:35:25"
    }
  ]
}
```


Εικόνα 5.5 User1 GetBlockchain response

Όπως ήταν αναμενόμενο ο Flask Server για τον User1, μας επέστρεψε το blockchain όπου μπορούμε να παρατηρήσουμε τα εξής στοιχεία:
Κάθε στοιχείο της δομής chain είναι ένα block με τη μορφή JSON. Μέσα στο κάθε block παρατηρούμε όλα τα transactions που υπάρχουν (πεδίο data) καθώς και πληροφορίες του block όπως: hash, nonce, previous Hash, timestamp και άλλα.

Προφανώς η απάντηση, την οποία θα λαμβάνουν ο User2 και User3, από τον Flask Server θα είναι η ίδια με παραπάνω αφού το δίκτυο blockchain είναι κατανεμημένο!
Για παράδειγμα η απάντηση που παίρνει ο User2 είναι η εξής:

```
{
  "chain": [
    {
      "data": [
        {
          "amount": 20000,
          "description": "genesis_block",
          "from": "reward_tx",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "7ce1f57b3d57d5ef7ed5db1dbee22198519c27402dec7d5c73144090a64ab076"
        }
      ],
      "hash": "273876760355b579529e2afc0820bdc5e079da23da873544568a579051f9717f",
      "hash_merkle_block": "",
      "index": 0,
      "nonce": 0,
      "previousHash": "0",
      "timestamp": "2018-09-01 19:35:25"
    }
  ]
}
```


Εικόνα 5.6 User2 GetBlockchain response

Εισερχόμενοι οι User2 και User3 στο δίκτυο Blockchain, συγχρονίζονται αυτόματα και δυναμικά από τον User1 ο οποίος δημιούργησε το Genesis Block. Δηλαδή, ο User1 αποτελεί για τους άλλους χρήστες DNS Seed στην γλώσσα του Bitcoin πρωτοκόλλου.
Μέσω του Python Flask API, έχω δημιουργήσει κατάλληλα Resources τα οποία χειρίζονται ερωτήματα των κόμβων τα οποία έχουν να κάνουν με την διασύνδεση τους με άλλους κόμβους.
Συγκεκριμένα:

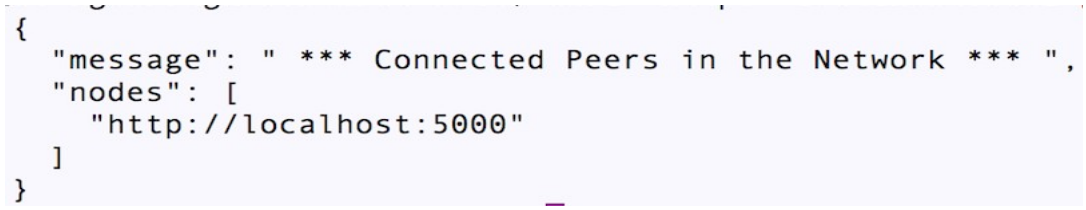
- Ο User2 μπορεί να μάθει με ποιους άλλους κόμβους είναι συνδεδεμένος ως εξής:

http get request

```
GiorgosFragkos:Source Code$ curl "http://localhost:5002/getnodes"
```

Εικόνα 5.7 User2 Getnodes Request

http response



```
{
  "message": " *** Connected Peers in the Network *** ",
  "nodes": [
    "http://localhost:5000"
  ]
}
```


Εικόνα 5.8 User2 Getnodes response

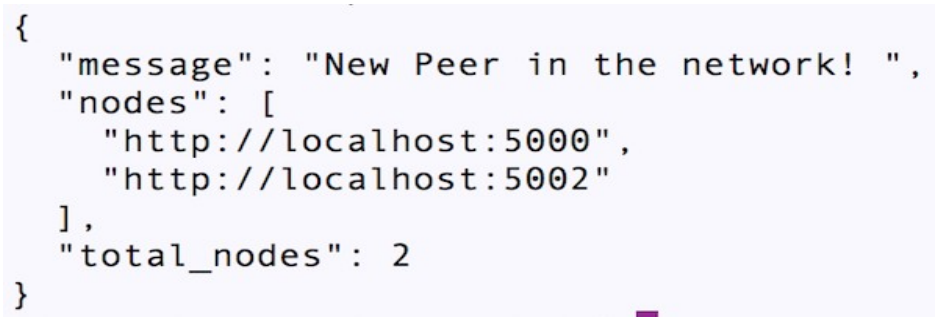
Παρατηρούμε λοιπόν ότι ο User2 είναι αυτόματα συνδεδεμένος με τον User1 (http://localhost:5000), αλλά δεν είναι ενήμερος για την παρουσία του User3 στο δίκτυο. Οπότε εκμεταλλευόμενοι ένα κατάλληλο Flask Resource θα τον προσθέσει στους γνωστούς του peers. Ακολουθεί η εικόνα που υποδυκνείει την αντίστοιχη ενέργεια του User2:

http post request

```
GiorgosFragkos:Source Code$ curl -X POST -H "Content-Type: application/json" -d '{"address": "http://localho
st:5002"}' "http://localhost:5001/addnode"
```

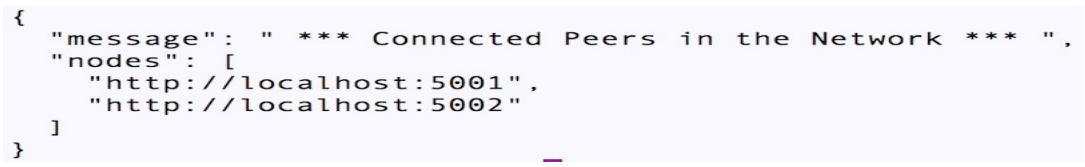
Εικόνα 5.9 User2 Addnode post request

και παρακάτω ακολουθεί η απάντηση του Server:



Εικόνα 5.10 User2 Addnode Response

Παρατηρούμε λοιπόν από την παραπάνω απάντηση ότι στην λίστα των συνδεδεμένων peers του User2 προστέθηκε και ο User3 (http://localhost:5002).
Αντίστοιχα, ο User3 πρέπει να κάνει ακριβώς την ίδια διαδικασία που ακολουθήθηκε για τον User2 έτσι ώστε να τον προσθέσει ως peer του. Τέλος, το ίδιο πρέπει να κάνει ο User1 για τον User2 και User3 έτσι ώστε να τους έχει peers του. Η συγκεκριμένη διαδικασία είναι αναγκαία να γίνεται κατά την είσοδο νέων peers έτσι ώστε να επιτυγχάνεται καλύτερος συγχρονισμός και επικοινωνία μεταξύ των ομότιμων κόμβων.
Οπότε όταν ολοκληρωθεί η συγκεκριμένη διαδικασία αν ο User1 ρωτήσει τον Server ποιοι είναι η συνδεδεμένοι κόμβοι με αυτόν, η απάντηση που θα λάβει θα είναι η εξής:



Εικόνα 5.11 User1 Getnode Response

5.3 Δημιουργία Συναλλαγής στο δίκτυο Blockchain

Κάθε ομότιμος κόμβος στο δίκτυο blockchain έχει την δυνατότητα να δημιουργεί συναλλαγές. Για παράδειγμα ο UserX μπορεί να στείλει στον UserY Z tokens, αν φυσικά υπάρχει η αντίστοιχη οικονομική επάρκεια. Οπότε έχουμε δημιουργήσει το αντίστοιχο Flask resource το οποίο δημιουργεί ένα transaction και το δημοσιεύει στο blockchain δίκτυο.
Παρακάτω ακολουθούν οι αντίστοιχες εικόνες:
http post request

```
GiorgosFragkos:Source Code$ curl -X POST -H "Content-Type: application/json" -d '{"from1": "User1", "to": "U
ser2", "amount": 357,"description":"1st Transaction"}' "http://127.0.0.1:5000/create_transaction"
```

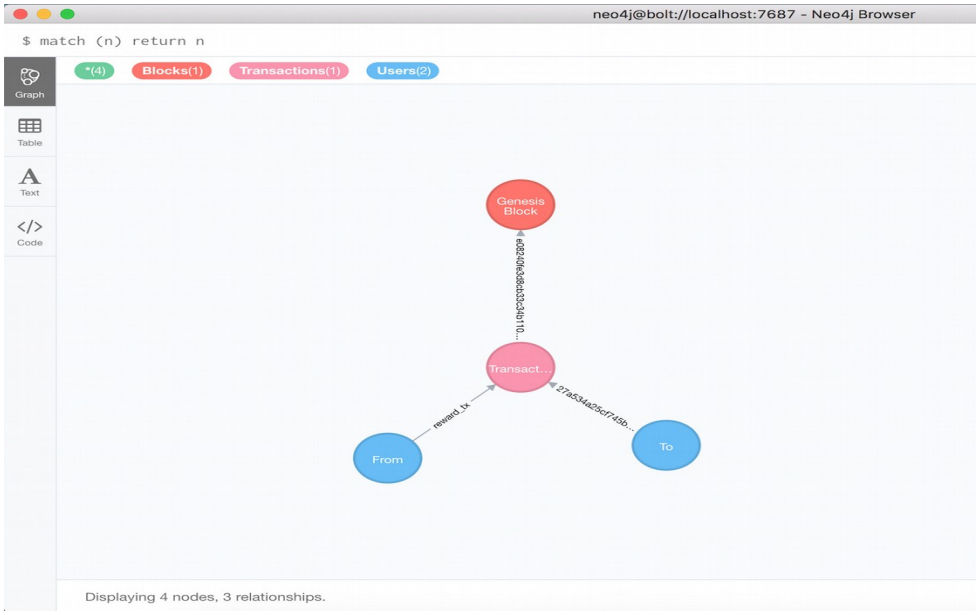
Εικόνα 5.12 User1 Create Transaction Request

http Response


```
{
  "block_index": 1,
  "message": "Transaction submitted to the network"
}
```


Εικόνα 5.13 Create Transaction Response

Παρατηρούμε λοιπόν εδώ ότι ο χρήστης User1 στέλνει στον User2 357 tokens, και αφού γίνει ο κατάλληλος έλεγχος στον πηγαίο κώδικα ότι το υπόλοιπο του User1 είναι επαρκές για αυτή τη συναλλαγή, δημοσιεύεται στο δίκτυο σαν http response η συγκεκριμένη συναλλαγή. Το πεδίο block index εκφράζει σε ποιο block του blockchain θα συμπεριληφθεί αυτή η συναλλαγή, οπότε στην περίπτωση μας στο 1 (στο επόμενο από το Genesis Block).
Εκτελώντας ένα Cypher Query στην τοπική βάση όμως του User1 (ή του User2) θα παρατηρήσουμε ότι αυτή η συναλλαγή δεν έχει δημοσιευτεί στην Neo4j.



Συνεχίζουμε να παρατηρούμε πως υπάρχει μόνο το Genesis Block.

Αυτό σημαίνει πως η συναλλαγή δεν εκτελέστηκε καθόλου ή δεν εκτελέστηκε σωστά; Φυσικά και όχι! Συγκεκριμένα η συναλλαγή για να επικυρωθεί πρέπει να συμπεριληφθεί σε ένα block και μάλιστα αυτό να δημοσιευθεί τελικά από κάποιον χρήστη. Αυτός είναι ο λόγος που δεν παρατηρούμε την συναλλαγή να έχει καταγραφή στη Neo4j όταν εκτελούμε το query: match (n) return n.

Η εργασία αυτή πραγματοποιείται με τα κατανεμημένα πρωτόκολλα συναίνεσης τα οποία και κατασκευάσαμε, δηλαδή με το PoW, PoS και το PoM. Εν συνεχεία παρουσιάζονται οι λειτουργικότητες αυτών των πρωτοκόλλων.

5.4 Δημοσίευση νέου block με βάση το PoW πρωτόκολλο

Όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, στο Proof Of Work πρωτόκολλο οι κόμβοι οι οποίοι συμμετέχουν στη διαδικασία της εξόρυξης είναι υποχρεωμένοι να σπαταλήσουν υπολογιστικούς πόρους έτσι ώστε να λύσουν ένα κρυπτογραφικό παζλ. Ο πρώτος χρήστης ο οποίος θα βρει το nonce είναι και αυτός ο οποίος θα δημοσιεύσει το νέο block μαζί με όλες τις συναλλαγές, συμπεριλαμβανομένου και του reward από το δίκτυο.

Για τους σκοπούς της παρούσας διπλωματικής λοιπόν θεωρούμε ότι και οι 3 Users συμμετέχουν στην διαδικασία της εξόρυξης, οπότε και είναι υποψήφιοι για να δημοσιεύσουν το νέο block. Για αυτόν τον σκοπό

έχουμε δημιουργήσει ένα Sript File όπου ξεκινάμε και τους τρεις κόμβους να κάνουν mine ανά 2 λεπτά. Ο πηγαίος κώδικας για αυτό είναι ο εξής:

```
#!/bin/bash


    while :; do clear; curl -X POST -H "Content-Type:
application/json" -d '{"miner": "User1"}'
"http://127.0.0.1:5000/mine" & curl -X POST -H "Content-Type:
application/json" -d '{"miner": "User2"}'
"http://127.0.0.1:5001/mine" & curl -X POST -H "Content-Type:
application/json" -d '{"miner": "User2"}'
"http://127.0.0.1:5002/mine"
    ; sleep 120; done
```

Παρατηρούμε λοιπόν πως καλούμε το αντίστοιχο Flask Resource και για τους 3 Users ανά 120 δευτερόλεπτα, δηλαδή ανά δύο λεπτά. Η εντολή για να τρέξει το αντίστοι script είναι:
./PoW_competition.sh

Εκτελώντας λοιπόν την εντολή οι 3 Users ξεκινούν την προαναφερθείσα διαδικασία και μάλιστα αφού τελειώσει η εξόρυξη για το 1ο πρώτο block οι απαντήσεις σε κάθε κόμβο από τον Flask Server είναι οι εξής:

- User1

```
Mining Block...
Block mined by the Miner : 27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a ,and the block h
ash is : 00000109415b5357df08b41e52df6862583f7c6a030ca83627f829bf79b69a2b

The Blocks Number currently in the Neo4j DB are : 1
You are the winner!!!
Previous Block ID is : 0
```

Εικόνα 5.14 User1 Mining Winning Response

- User2 & 3

```
Mining Block...
Block mined by the Miner : 0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285 ,and the block h
ash is : 000008c0bee388894c5935d71333884a7bbcfcccbed61d250bdd0f5e71e4bb52

The Blocks Number currently in the Neo4j DB are : 2
You didn't solved first the cryptographic puzzle! Sorry! Next time...
```

Εικόνα 5.15 User2 & User3 Mining Response

Παρατηρούμε λοιπόν από τις παραπάνω εικόνες ότι νικητής του διαγωνισμού εξόρυξης είναι ο User1, ενώ οι User2 και User2 ενημερώνονται από το δίκτυο ότι ο νικητής είναι ο User1 και να

σταματήσουν την διαδικασία της εξόρυξης. Αυτό έχει ως συνέπεια ο νικητής να λάβει το αντίστοιχο reward και να ενημερώσει την δομή του blockchain και την Neo4j με το νέο block.
Πραγματικά, κάνοντας http get request στον Flask Server ο User1 θα λάβει το ανανεωμένο blockchain ως απάντηση και αυτό φαίνεται στην παραπάνω εικόνα:


```
{
  "chain": [
    {
      "data": [
        {
          "amount": 20000,
          "description": "genesis_block",
          "from": "reward_tx",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "7ce1f57b3d57d5ef7ed5db1dbee22198519c27402dec7d5c73144090a64ab076"
        }
      ],
      "hash": "273876760355b579529e2afc0820bdc5e079da23da873544568a579051f9717f",
      "hash_merkle_block": "",
      "index": 0,
      "nonce": 0,
      "previousHash": "0",
      "timestamp": "2018-09-01 19:35:25"
    },
    {
      "data": [
        {
          "amount": 357,
          "description": "1st Transaction",
          "from": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "to": "0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285",
          "transactionHash": "411427ddab810e7c0526a1e64d10b4505324de5edcb3d070e633bb75e9398fc2"
        },
        {
          "amount": 1,
          "description": "tr_transaction_fee :411427ddab810e7c0526a1e64d10b4505324de5edcb3d070e633bb75e9398fc2",
          "from": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "9bae6ac887753954a3c220fbe73102ee9b31682b6281e8df7a42ecbd88b462da"
        },
        {
          "amount": 10,
          "description": "coinbase_transaction",
          "from": "reward_tx",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "c617b1f9c949fdcf9ad41519a95a7bd43f8aa24864f6a925d7d2e437935f9a59"
        }
      ],
      "hash": "00000109415b5357df08b41e52df6862583f7c6a030ca83627f829bf79b69a2b",
      "hash_merkle_block": "05c2a53b73d03cc7dfa3ddaa5b141c43b58f180e836a2ecd6073dde30b20f5c3",
      "index": 1,
      "nonce": 1223233,
      "previousHash": "273876760355b579529e2afc0820bdc5e079da23da873544568a579051f9717f",
      "timestamp": "2018-09-01 21:22:22"
    }
  ]
```

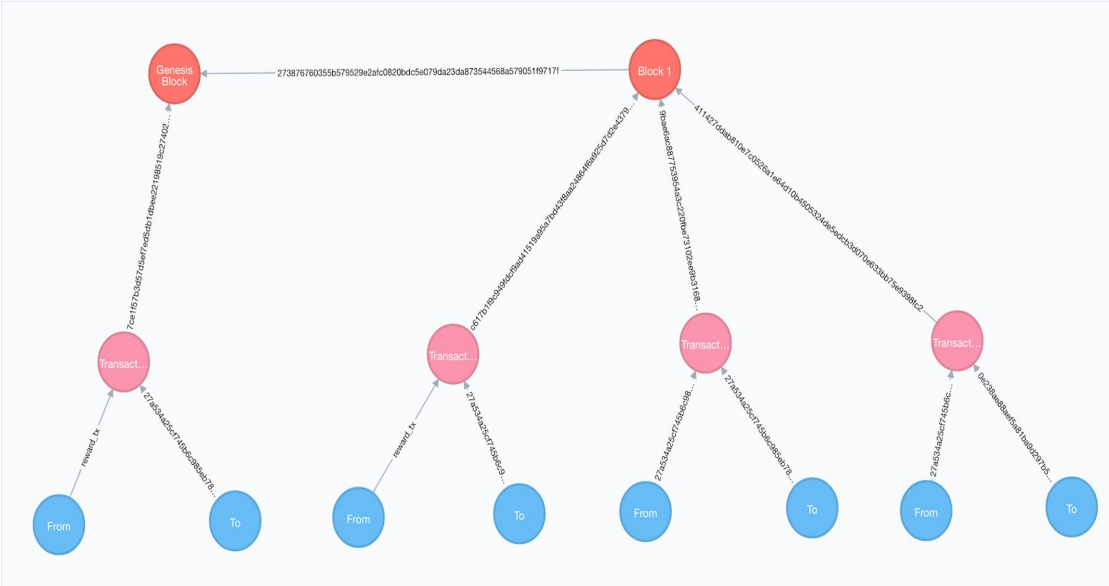

Εικόνα 5.16 User1 GetBlockchain Response

Στην παραπάνω εικόνα λοιπόν όπου φαίνεται το blockchain, το οποίο αποτελείται από 2 blocks πλέον, παρατηρούμε τις εξής πληροφορίες:

- Μέσα στη δομή chain[] παρατηρούμε δύο δομές JSON[50], δηλαδή δύο διαφορετικά blocks.
- Το πρώτο JSON είναι το Genesis Block το οποίο περιγράφηκε πιο πάνω.
- Το δεύτερο JSON είναι το block το οποίο έκανε mine ο User1 και εν συνεπεία το δημοσίευσε.
- Μέσα στο δεύτερο block (JSON), στο πεδίο data, παρατηρούμε το transaction στο οποίο έστειλε ο User1 στον User2 357 tokens. Επιπροσθέτως παρατηρούμε ότι ο User1, με public adress *27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a*, έλαβε 10 tokens αυτόματα ως reward από το δίκτυο επειδή ήταν ο νικητής της mining διαδικασίας. Τέλος, λόγω της συναλλαγής έλαβε 1 token πίσω ως transaction fee το οποίο βέβαια τα είχε δώσει ο ίδιος ώστε να τα πάρει ο πιθανός miner. Φυσικά για κάθε συναλλαγή εμφανίζεται και το αντίστοιχο transaction Hash το οποίο λειτουργεί ως απόδειξη της συναλλαγής.

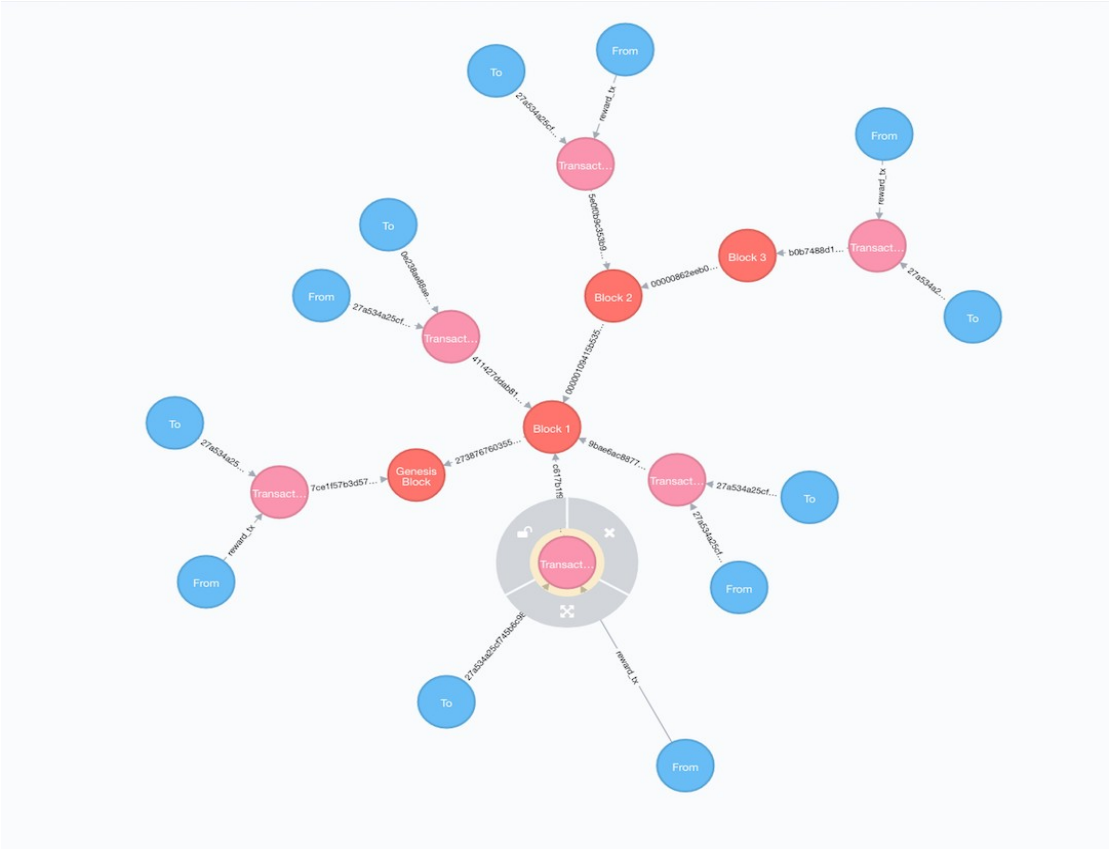
- Τέλος μέσα στη δομή chain[] , παρατηρούμε τις γενικότερες πληροφορίες του 2ου block όπως: hash -το οποίο έχει 5 μηδενικά στην αρχή αφού το difficulty του δικτύου είναι 5- , hash merkle block, index, nonce, previousHash και timestamp.

Ύστερα αν οποισδήποτε από τους Users κάνει ένα Cypher Query (match (n) return n) στην Neo4j θα λάβει το ανανεωμένο blockchain:



Εικόνα 5.17(α) Neo4j Blockchain Graph Representation - PoW

Σημειώνεται πως καθώς η εφαρμογή συνεχίζει να λειτουργεί, συνεχίζουν οι χρήστες να κάνουν διάφορες συναλλαγές και να ανταγωνίζονται ώστε να λύσουν το κρυπτογραφικό παζλ και να δημοσιεύσουν το νέο block. Οπότε ως συνέπεια, το blockchain μεγαλώνει σε μέγεθος και ταυτόχρονα δυναμικά ενημερώνεται η Neo4j διατηρώντας ταυτόχρονα τον αποκεντρωμένο χαρακτήρα της εφαρμογής τον οποίο τον επιβάλλει το blockchain. Ακολουθεί μια εικόνα η οποία δείχνει τον γράφο της Neo4j αφού προστεθούν κάποια blocks ακόμα:



Εικόνα 5.17(β) Neo4j Blockchain Graph Representation - PoW

5.5 Δημοσίευση νέου block με βάση το PoS πρωτόκολλο

Όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, στο Proof Of Stake πρωτόκολλο οι κόμβοι είναι πιθανοί νικητές της διαδικασίας της δημοσίευσης ενός νέου block με βάση τον πλούτο και έναν δείκτη τυχαιότητας. Σημειώνεται πως η διαδικασία εισαγωγής κόμβων στο peer to peer δίκτυο καθώς και αυτή της επικοινωνίας μεταξύ τους παραμένει ίδια όπως και στο PoW, αφού ανήκουν σε χαμηλότερα επίπεδα της στοίβας πρωτοκόλλων. Για αυτόν τον λόγο δεν παρατίθενται ξανά οι εντολές εισαγωγής, επικοινωνίας και συγχρονισμού κόμβων.

Οπότε έχουμε ξανά το ίδιο σενάριο, δηλαδή 3 Users οι οποίοι επικοινωνούν στο δίκτυο Blockchain. Επειδή λοιπόν ο User1 δημιούργησε και δημοσίευσε το Genesis Block έλαβε και όλα τα αρχικά tokens (20000) του δικτύου. Αυτό φυσικά, λόγω της φύσης του πρωτοκόλλου έχει ως συνέπεια να έχει τις περισσότερες πιθανότητες να

δημοσιεύσει το νέο μπλοκ, εφόσον οι υπόλοιποι χρήστες θα έχουν πολύ λιγότερα tokens.

Έτσι θα ακολουθήσουμε το ίδιο σενάριο με το PoW: (Σημειώνεται πως δεν παρατίθενται οι αντίστοιχες εικόνες -πχ για την δημιουργία συναλλαγής- αφού χρησιμοποιούνται τα ίδια Flask Resources)

- Εισάγουμε τους User1, User2 & User3.
- Διασφαλίζουμε την επικοινωνία μεταξύ τους.
- Εφόσον ο User1 εισάγεται πρώτος δημιουργεί το Genesis Block.
- Ο User1 στέλνει στον User2 357 tokens.

Ομοίως με πριν αν οποιασδήποτε από τους Users πάει να δει τον γράφο του blockchain στην Neo4j θα παρατηρήσει πως υπάρχει μόνο το Genesis Block. Αυτό συμβαίνει διότι για να φανεί η συγκεκριμένη συναλλαγή πρέπει να συμπεριληφθεί σε ένα μπλοκ και στη συνέχεια να δημοσιευτεί. Εδώ λοιπόν ειπεισέρχεται το πρωτόκολλο του Proof Of Stake το οποίο τρέχει ανεξάρτητα και ουσιαστικά φτιάχνει τα αντίστοιχα εύρη τιμών με βάση τον αριθμό των αντίστοιχων tokens, δίνοντας μεγαλυτερη πιθανότητα νίκης στον User1 που έχει τον μεγαλύτερο πλούτο.

Σε έναν οποιοδήποτε κόμβο λοιπόν κάνω http post request στον Python Flask Server έτσι ώστε να εφαρμοστεί το πρωτόκολλο Proof Of Stake:

http post request

```
GiorgosFragkos:Source Code$ curl "http://localhost:5000/pos"
```

Εικόνα 5.18 Http Post request – PoS Protocol

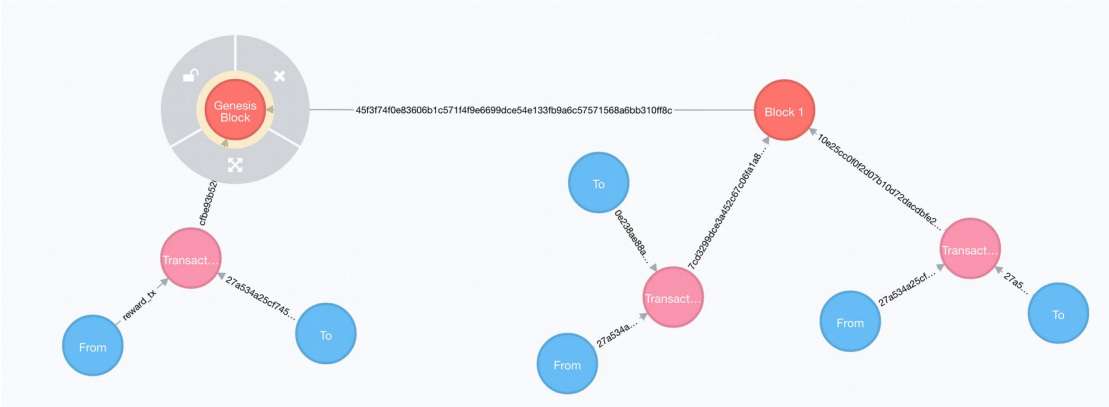
http response


```
{
  "message": "The Proof Of Stake winner of PoS Protocol is : ",
  "user": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a"
}
```


Εικόνα 5.19 Http Post response – PoS Protocol

Όπως παρατηρούμε στην παραπάνω επικοινωνία με τον Server, όπως ήταν αναμενόμενο ο χρήστης ο οποίος επιλέχθηκε από το κατανεμημένο πρωτόκολλο συναίνεσης PoS είναι ο User1 (public address:

27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a), ο οποίος έχει στην κατοχή του τα περισσότερα tokens. Αν τώρα γίνει στην κατανεμημένη Neo4j βάση το query `match(n) return n` θα λάβουμε το παρακάτω αποτέλεσμα στον Neo4j Browser :



Εικόνα 5.20 Neo4j Blockchain - PoS Protocol

Παρατηρούμε λοιπόν από την γραφική αναπαράσταση του blockchain που μας προσφέρει η Neo4j δυναμικά και αυτόματα τα εξής:

- Παρατηρούμε 2 blocks, το Genesis Block καθώς και το block το οποίο δημοσίευσε ο User1.
- Στο δεύτερο Block (Block 1), το οποίο συνδέεται με το Genesis μέσω του previousHash πεδίου σε μορφή graph relationship (previousHash: 27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a), συνδέονται 2 διαφορετικές συναλλαγές.
- Η πρώτη συναλλαγή η οποία συνδέεται μέσω του Transaction Hash με το Block1 μέσω graph relation (27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a), ουσιαστικά απεικονίζει τα 357 tokens τα οποία έστειλε ο User1 στον User2.
- Η δεύτερη συναλλαγή απεικονίζει την επιστροφή του transaction fee (1 token) στον User1 ο οποίος προκάλεσε την συναλλαγή.

Επιπροσθέτως, χρησιμοποιώντας το αντίστοιχο Python Flask Resource σε οποιονδήποτε συγχρονισμένο ομότιμο κόμβο μπορούμε να δούμε ακριβώς τις ίδιες πληροφορίες για το blockchain δωσμένες από τον αντίστοιχο τοπικό server:

http get request

```
GiorgosFragkos:Source Code$ curl "http://localhost:5000/getblockchain"
```

http response


```
{
  "chain": [
    {
      "data": [
        {
          "amount": 20000,
          "description": "genesis_block",
          "from": "reward_tx",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "cfbe93b526f0ac3d14a5944c584c4894eb400167d9e35e342471af4e38c78a1e"
        }
      ],
      "hash": "45f3f74f0e83606b1c571f4f9e6699dce54e133fb9a6c57571568a6bb310ff8c",
      "hash_merkle_block": "",
      "index": 0,
      "nonce": 0,
      "previousHash": "0",
      "timestamp": "2018-09-01 22:43:21"
    },
    {
      "data": [
        {
          "amount": 357,
          "description": "1st Transaction",
          "from": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "to": "0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285",
          "transactionHash": "7cd3299dce3a452c67c06fa1a828e648100cb5b7eecf6e410f3703242ff53a82"
        },
        {
          "amount": 1,
          "description": "tr_transaction_fee :7cd3299dce3a452c67c06fa1a828e648100cb5b7eecf6e410f3703242ff53a82",
          "from": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "to": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
          "transactionHash": "10e25cc0f0f2d07b10d72dacdbfe29aff04913e5a78eee35e26328dc1d4e0d26"
        }
      ],
      "hash": "56ce745d93a61f68a8e312e15059b34179f005ba99406346aadce616f06febfe",
      "hash_merkle_block": "3a0df5dddba31a27bfd6f7e673c85ced820903037e36ccb8fbf741fa559ec792",
      "index": 1,
      "nonce": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a",
      "previousHash": "45f3f74f0e83606b1c571f4f9e6699dce54e133fb9a6c57571568a6bb310ff8c",
      "timestamp": "2018-09-01 23:09:24"
    }
```


Εικόνα 5.21 Http Server Response – PoS Protocol

Εν κατακλείδι όπως παρατηρήσαμε, το συγκεκριμένο κατανεμημένο πρωτόκολλο συναίνεσης, στην συντριπτική πλειοψηφία των φορών παρά την τυχαιότητα, επιλέγει τον User με τον μεγαλύτερο πλούτο συνεχόμενες φορές και για αυτόν τον λόγο τείνει να γίνει κεντρικοποιημένο.

5.6 Δημοσίευση νέου block με βάση το PoM πρωτόκολλο

Όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο, στο Proof Of Motion πρωτόκολλο οι κόμβοι είναι πιθανοί νικητές της διαδικασίας της δημοσίευσης ενός νέου block με βάση το πόσο ενεργοί είναι μέσα στο δίκτυο (σε ένα παράθυρο του blockchain) και έναν δείκτη τυχαιότητας. Σημειώνεται πως η διαδικασία εισαγωγής κόμβων στο peer to peer δίκτυο καθώς και αυτή της επικοινωνίας μεταξύ τους παραμένει ίδια όπως και στο PoW, αφού ανήκουν σε χαμηλότερα επίπεδα της στοίβας πρωτοκόλλων. Για αυτόν τον λόγο δεν παρατίθενται ξανά οι εντολές εισαγωγής, επικοινωνίας και συγχρονισμού κόμβων. Υπενθυμίζεται επίσης πως το πόσο ενεργός είναι ένας χρήστης εξαρτάται από τον αριθμό των συναλλαγών που έχει κάνει.

Οπότε έχουμε ξανά το ίδιο σενάριο, δηλαδή 3 Users οι οποίοι επικοινωνούν στο δίκτυο Blockchain. Επειδή λοιπόν ο User1 δημιούργησε και δημοσίευσε το Genesis Block έλαβε και όλα τα αρχικά tokens (20000) του δικτύου. Εν συνεχεία θα παραλλαχθεί λίγο το σενάριο ώστε να φανεί η φύση του συγκεκριμένου πρωτοκόλλου. Το σενάριο λοιπόν το οποίο θα ακολουθήσουμε για αρχή είναι το εξής:

- Εισάγουμε τους User1, User2 & User3.
- Διασφαλίζουμε την επικοινωνία μεταξύ τους.
- Εφόσον ο User1 εισάγεται πρώτος δημιουργεί το Genesis Block.
- Ο User1 στέλνει στον User2 357 tokens.
- Ο User1 στέλνει στον User3 67 tokens.
- Εφαρμογή κατανεμημένου πρωτοκόλλου PoM

Ομοίως με πριν αν οποιασδήποτε από τους Users πάει να δει τον γράφο του blockchain στην Neo4j θα παρατηρήσει πως υπάρχει μόνο το Genesis Block. Αυτό συμβαίνει διότι για να φανεί η συγκεκριμένη συναλλαγή πρέπει να συμπεριληφθεί σε ένα μπλοκ και στη συνέχεια να δημοσιευτεί. Εδώ λοιπόν ειπεισέρχεται το πρωτόκολλο του Proof Of Motion το οποίο τρέχει ανεξάρτητα, ελέγχει ένα συγκεκριμένο παράθυρο του blockchain (το 60% των blocks), και ύστερα δίνει την μεγαλύτερη πιθανότητα νίκης στον χρήστη με τις πιο πολλές συναλλαγές, δηλαδή στον πιο ενεργό.

Στο συγκεκριμένο σενάριο αφού ο μόνος χρήστης ο οποίος έχει κάνει συναλλαγές είναι ο User1 και έχουμε μόνο 1 block προς εξέταση είναι και ο πιθανός νικητής, όσον αφορά τη δημοσίευση του νέου block

στο blockchain. Σε έναν οποιοδήποτε κόμβο λοιπόν κάνω http post request στον Python Flask Server έτσι ώστε να εφαρμοστεί το πρωτόκολλο Proof Of Motion:

http post request

```
GiorgosFragkos:Source Code$ curl "http://localhost:5000/pom"
```

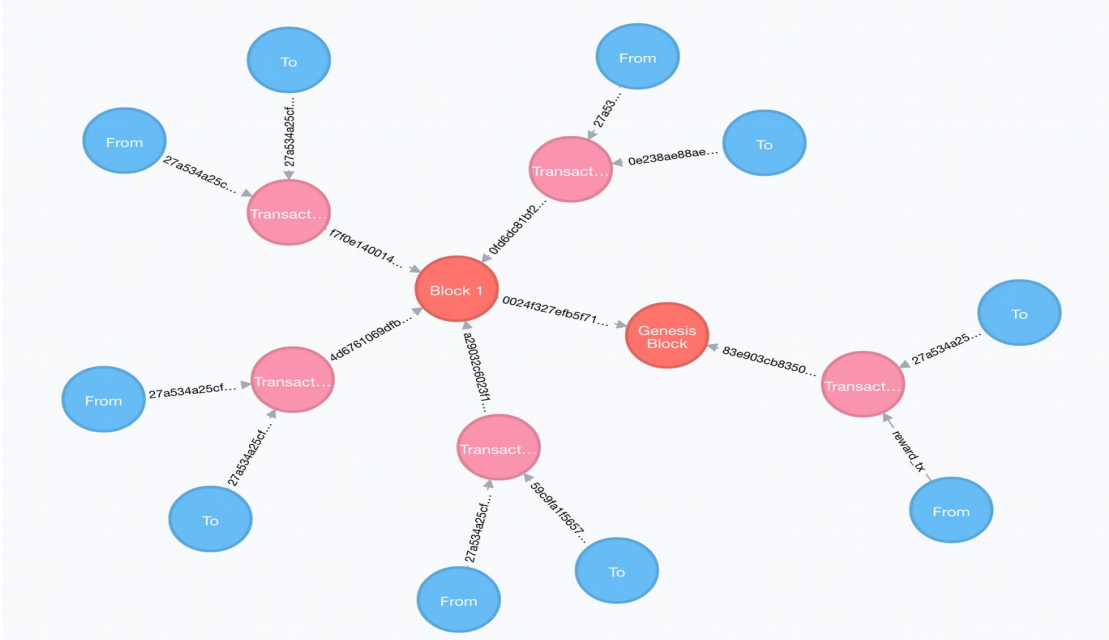
Εικόνα 5.22 Http Post request – PoM Protocol

http response

```
{
  "message": "The Proof of Motion winner is : ",
  "user": "27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a"
}
```

Εικόνα 5.23 Http Response – PoM Protocol

Όπως παρατηρούμε στην παραπάνω επικοινωνία με τον Server, όπως ήταν αναμενόμενο ο χρήστης ο οποίος επιλέχθηκε από το κατανεμημένο πρωτόκολλο συναίνεσης PoM είναι ο User1 (public address: 27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a), ο οποίος ήταν ο πιο ενεργός στο δίκτυο κάνοντας τις περισσότερες συναλλαγές. Αν τώρα γίνει στην κατανεμημένη Neo4j βάση το query `match(n) return n` θα λάβουμε το παρακάτω αποτέλεσμα στον Neo4j Browser :

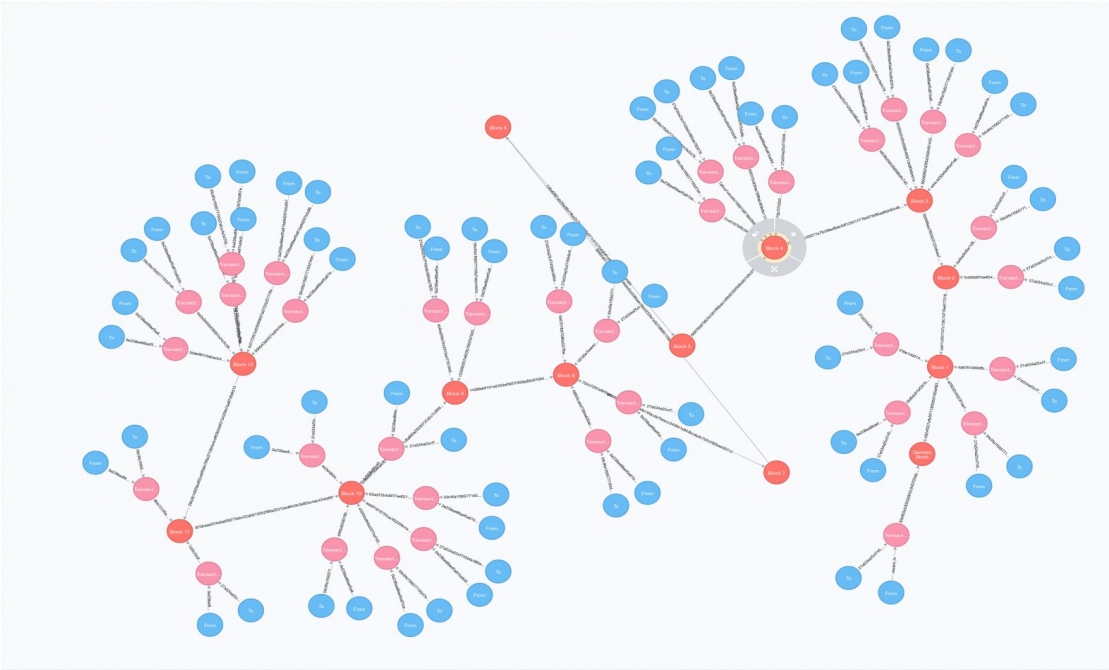


Εικόνα 5.24 Neo4j Blockchain - PoM Protocol

Παρατηρούμε λοιπόν από την γραφική αναπαράσταση του blockchain που μας προσφέρει η Neo4j δυναμικά και αυτόματα τα εξής:

- Παρατηρούμε 2 blocks, το Genesis Block καθώς και το block το οποίο δημοσίευσε ο User1 μέσω του PoM.
- Στο δεύτερο Block (Block 1), το οποίο συνδέεται με το Genesis μέσω του previousHash πεδίου σε μορφή graph relationship (previousHash: 27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a), συνδέονται 3 διαφορετικές συναλλαγές.
- Η πρώτη συναλλαγή η οποία συνδέεται μέσω του Transaction Hash με το Block1 μέσω graph relation , ουσιαστικά απεικονίζει τα 357 tokens τα οποία έστειλε ο User1 στον User2.
- Η δεύτερη συναλλαγή η οποία συνδέεται μέσω του Transaction Hash με το Block1 μέσω graph relation , ουσιαστικά απεικονίζει τα 67 tokens τα οποία έστειλε ο User1 στον User3.
- Η τρίτη συναλλαγή απεικονίζει την λήψη των transaction fees των συναλλαγών από τον User1 ο οποίος δημοσίευσε το block.

Εν συνεχεία για να φανούν καλύτερα τα λειτουργικά χαρακτηριστικά του Proof Of Motion, θεωρούμε ότι το blockchain αποτελείται από 13 blocks τα οποία όλα έχουν δημιουργηθεί χρησιμοποιώντας προφανώς το συγκεκριμένο κατανεμημένο πρωτόκολλο συναίνεσης. Συγκεκριμένα από Neo4j Browser έχω:



Εικόνα 5.25 Neo4j Blockchain με 12 Blocks - PoM Protocol

Οπότε για την δημοσίευση του 14ου block θα ακουληθηθεί η εξής διαδικασία:

- Έλεγχος του 60% του συνολικού πλήθους των blocks στο blockchain με στρογγυλοποίηση προς τα πάνω. Οπότε αφού έχουμε 13 blocks, θα εξεταστούν (60/100)*13 = 7.8 = 8 blocks.
- Οπότε στα τελευταία 8 blocks θα εξεταστεί το ποιοι είναι οι πιο ενεργοί χρήστες ανάλογα με το πόσες συναλλαγές έχουν κάνει. Αυτό πραγματοποιείται μέσω του πηγαίου κώδικα αυτοματοποιημένα χρησιμοποιώντας την Neo4j. Έτσι κάνουμε το εξής Cypher query :

```
MATCH (e1:Blocks)-[rels*1..2]-(e2:Users) where
e1.id>='+str(chain_length-examine_blocks)+' WITH e1, e2,
rels, extract(rel IN rels | startNode(rel)) AS
startNodes, extract(rel IN rels | endNode(rel)) AS
endNodes, range(1, size(rels)-1) AS indexes WITH e1, e2,
rels, startNodes, endNodes, indexes, startNodes[0] AS
start UNWIND indexes AS i WITH e1, e2, rels, e1 = start
as isOutFirst, (endNodes[i-1] = startNodes[i] OR
startNodes[i-1] = startNodes[i]) AS isOut WITH e1, e2,
rels, isOutFirst, collect(isOut) AS isOuts WITH e1, e2,
rels, [isOutFirst] + isOuts AS isOuts, range(0,
size(rels)-1) AS indexes2 UNWIND indexes2 AS i RETURN
e2.address,count(e2.address) order by count(e2.address)
desc
```

, όπου str(chain_length-examine_blocks)=8.

- Το παραπάνω Cypher Query μας επιστρέφει τον εξής πίνακα σαν αποτέλεσμα εξετάζοντας το παράθυρο των 8 blocks:

0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285 (User2)	42 Transactions
59c9fa1f56577193d7ebccfe34793a521c5b3b5ebc01f7075ebc90546ef3c6be (User3)	20 Transactions
27a534a25cf745b6c985eb782079a6fe8641b00003dada14f392a2d01b9c790a (User1)	18 Transactions

Παρατηρούμε λοιπόν ότι ο πιο ενεργός χρήστης του δικτύου την συγκεκριμένη χρονική στιγμή είναι ο User2 έχοντας κάνει 42 συναλλαγές. Οπότε, λόγω της αλγοριθμικής σκέψης πίσω από το συγκεκριμένο κατανεμημένο πρωτόκολλο συναίνεσης ο πιο πιθανός χρήστης για να δημοσιεύσει το block είναι ο User2.

- Έτσι σε έναν οποιοδήποτε κόμβο κάνω http post request στον Python Flask Server έτσι ώστε να εφαρμοστεί το πρωτόκολλο Proof Of Motion και η απάντηση που λαμβάνω από τον Flask Server είναι η εξής:

http response


```
{
  "message": "The Proof of Motion winner is : ",
  "user": "0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285"
}
```


Εικόνα 5.26 PoM Winner – User2 αφού είναι ο πιο ενεργός

- Τελικά όπως ήταν αναμενόμενο, ο χρήστης User2 με δημόσια διεύθυνση 0e238ae88aef5a81ba9d297b5df67e74af15d168e5b765db22227c91b8672285 είναι ο νικητής του Proof Of Motion και είναι λογικό/αναμενόμενο αφού στο εξεταζόμενο παράθυρο του blockchain ήταν ο πιο ενεργός. Ως αποτέλεσμα λοιπόν ο User2 δημοσίευσε το 14ο block και συνέλλεξε τα transactions fees των αντίστοιχων συναλλαγών.

5.7 Δημιουργία και εκτέλεση Cypher Queries στην Neo4j

Χρησιμοποιώντας την βάση δεδομένων γράφου στην εφαρμογή μας, παρέχουμε τη δυνατότητα στον χρήστη να έχει πρόσβαση σε πληροφορίες, στις οποίες σε κλασικές πλατφόρμες blockchain (τύπου Bitcoin ή Ethereum Platform[51]) δεν θα μπορούσε να έχει.

Συγκεκριμένα, όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο ο χρήστης μέσω του Neo4j Browser έχει οπτική επαφή με τη δομή του blockchain η οποία ανανεώνεται δυναμικά και αυτόματα μέσω του πηγαίου κώδικα. Οπότε μπορεί να κάνει περίπλοκα ερωτήματα στο User Interface που παρέχεται από τον Neo4j Browser όπως :

- Στα τελευταία X Blocks, ποιοι χρήστες συμμετείχαν σε συναλλαγές και πόσες έκανε ο καθένας από αυτούς ; Τα αποτελέσματα να επιστραφούν σε φθίνουσα σειρά ώστε να αποφανθεί ποιος είναι ο πιο ενεργός χρήστης του δικτύου.

```
 MATCH (e1:Blocks)-[rels*1..2]-(e2:Users) where e1.id>=X
WITH e1, e2, rels, extract(rel IN rels | startNode(rel))
AS startNodes, extract(rel IN rels | endNode(rel)) AS
endNodes, range(1, size(rels)-1) AS indexes WITH e1, e2,
rels, startNodes, endNodes, indexes, startNodes[0] AS
start UNWIND indexes AS i WITH e1, e2, rels, e1 = start
as isOutFirst, (endNodes[i-1] = startNodes[i] OR
startNodes[i-1] = startNodes[i]) AS isOut WITH e1, e2,
rels, isOutFirst, collect(isOut) AS isOuts WITH e1, e2,
rels, [isOutFirst] + isOuts AS isOuts, range(0,
size(rels)-1) AS indexes2 UNWIND indexes2 AS i RETURN
e2.address,count(e2.address) order by count(e2.address)
desc
```

- Σε ποιες συναλλαγές έχουν συμμετάσχει ο UserX και ο UserY; Για τις συναλλαγές αυτές να επιστραφούν τα αντίστοιχα Transaction Hashes. Για αυτό το ερώτημα μπορεί να χρησιμοποιηθεί το Flask Resource το οποίο ουσιαστικά χρησιμοποιεί Cypher Queries τα οποία τα συνδυάζει και τελικά απαντάει στο ζητούμενο ερώτημα.
  Το προαναφερθέν Flask Resource στον κώδικα είναι το */traverse.*

Τα παραπάνω είναι κάποια ενδεικτικά Cypher Queries τα οποία μπορούμε να κάνουμε στην Neo4j. Φυσικά, οποιοδήποτε query το οποίο είναι συντακτικά σωστό είναι ταυτόχρονα και αποδεκτό.

6

Επίλογος

6.1 Σύνοψη και συμπεράσματα

Σκοπός της παρούσας διπλωματικής εργασίας ήταν η εξέταση της νέας αποκεντρωμένης τεχνολογίας blockchain σε συνδυασμό με την ενσωμάτωση της με την βάση δεδομένων γράφου Neo4j. Έτσι δημιουργήσαμε μια εφαρμογή αποκεντρωμένου χαρακτήρα, η οποία αποτελείται από ένα πρωτότυπο Python blockchain και μια κατανεμημένη βάση Neo4j, η οποία ουσιαστικά περιγράφει σε τριεπίπεδη αρχιτεκτονική το προαναφερθέν blockchain.

Κατά την διάρκεια ενασχόλησης με το Blockchain από αλγοριθμικής πλευράς, παρατήρησα ότι σε περίπτωση που ένας χρήστης θέλει να αναζητήσει κάποια συγκεκριμένη πληροφορία τότε πρέπει να διατρέξει όλη τη δομή. Χαρακτηριστικό παράδειγμα είναι το εξής : Έστω ότι ένας χρήστης θέλει να μάθει σε ποιες συναλλαγές συμμετείχε ο UserK. Στις κλασικές blockchain πλατφόρμες, θα έπρεπε η αναζήτηση να ξεκινήσει σειριακά από το τελευταίο block μέχρι και το Genesis block και μάλιστα να γίνεται μια επιπλέον αναζήτηση στα εσωτερικά δεδομένα του κάθε μπλοκ. Αυτό φυσικά έχει ως αποτέλεσμα όσο αυξάνεται ο αριθμός των block και της πληροφορίας που βρίσκεται μέσα σε αυτά, ο αντίστοιχος χρόνος αναζήτησης να αυξάνεται εκθετικά. Επιπροσθέτως, ένα άλλο πρόβλημα το οποίο παρατήρησα ήταν το γεγονός ότι τα ήδη υπάρχοντα και προτεινόμενα κατανεμημένα πρωτόκολλα συναίνεσης ίσως έχαναν τον αποκεντρωμένο χαρακτήρα τους.

Για αυτόν τον λόγο, στράφηκα στον συνδυασμό αυτής της δομής δεδομένων με την Neo4j βάση δεδομένων γράφου. Συγκεκριμένα, στο blockchain δεν μας ενδιαφέρουν τόσο τα δεδομένα αυτά καθε αυτά, αλλά μας ενδιαφέρουν οι σχέσεις που δημιουργούν μεταξύ τους, όπως για παράδειγμα συμβαίνει και στα κοινωνικά δίκτυα. Δηλαδή μας ενδιαφέρει ποιες συναλλαγές συνδέονται με ποια blocks, ποιοι χρήστες συνδέονται με ποια blocks κτλ.

Επιπροσθέτως, η Neo4j μου έδωσε τη δυνατότητα πέρα από την πιο εύκολη υλοποίηση των υπαρχόντων κατανεμημένων πρωτοκόλλων συναίνεσης (Proof Of Work, Proof Of Stake) στο Python πρωτότυπο blockchain το οποίο κατασκεύασα, να υλοποιήσω και ένα ακόμα καινοτόμο πρωτόκολλο συναίνεσης. Συγκεκριμένα, στην παρούσα διπλωματική εργασία προτείνεται σαν νέο κατανεμημένο πρωτόκολλο συναίνεσης το Proof Of Motion. Σε αυτό, η συμφωνία μεταξύ των ομότιμων κόμβων στο peer-to-peer δίκτυο όσον αφορά την δημοσίευση του επόμενου block στο blockchain, εξαρτάται από το πόσο ενεργός είναι ο κάθε χρήστης στο δίκτυο, δηλαδή από πόσες συναλλαγές έχει κάνει.

Φυσικά, η εξέταση και υλοποίηση των παραπάνω κατανεμημένων πρωτοκόλλων συναίνεσης μου με θεμέλιο λίθο τη Neo4j, μου έδωσε τη δυνατότητα να τα συγκρίνω και να βγάλω ένα συνολικό συμπέρασμα. Συγκεκριμένα, το Proof Of Work τείνει να λάβει μια κεντρικοποιημένη φύση, καθώς οι χρήστες/κόμβοι χρησιμοποιούν την υπολογιστική ισχύ τους για να λύσουν το κρυπτογραφικό πάζλ. Οπότε, θεωρητικά αυτός που έχει το "δυνατότερο" μηχάνημα έχει και τις πιο πολλές πιθανότητες να κερδίσει. Ανάλογα λειτουργεί και το κλασικό Proof Of Stake αφού ο πλούσιος γίνεται πλουσιότερος. Οπότε σε αυτό πρέπει να συμπεριληφθούν και άλλοι παράμετροι όπως το coin maturity. Τέλος προτείνεται και το Proof Of Motion το οποίο προσπαθεί να δώσει έναν αξιοκρατικό χαρακτήρα, δίνοντας την μεγαλύτερη πιθανότητα να κερδίσει στον πιο ενεργό χρήστη του δικτύου.

Εν κατακλείδι, στην παρούσα διπλωματική είδαμε τους θεμέλιους λίθους της δομής δεδομένων blockchain λόγω της εξ'ολοκλήρου υλοποίησης της. Επιπροσθέτως, η κατανεμημένη Neo4j την οποία είχε ο κάθε κόμβος, δίνει την δυνατότητα στους χρήστες για δημιουργία ερωτημάτων (Cypher Queries) και εξαιρετικά γρήγορες απαντήσεις σε αυτά μέσω του γράφου του blockchain. Τέλος, ο συνδυασμός αυτών των δύο τεχνολογιών είχε ως αποτέλεσμα την υλοποίηση του καινοτόμου, κατανεμημένου πρωτοκόλου συναίνεσης Proof Of Motion.

6.2 Μελλοντικές επεκτάσεις

Οι τεχνολογίες πάνω στις οποίες βασίζεται η εφαρμογή είναι σχετικά νέες, όμως η κοινότητα τους μεγαλώνει και αυτές αναπτύσσονται ταχύτατα. Είναι βέβαια σίγουρο ότι οι τεχνολογίες του blockchain και

των βάσεων δεδομένων γράφου, όπως η Neo4j, θα διαδραματίσουν καθοριστικό ρόλο στο εγγύς μέλλον.

Παρακάτω προτείνονται ορισμένες μελλοντικές επεκτάσεις της εφαρμογής, με την σημείωση ότι η υλοποίηση και επιτυχία μερικών εξαρτάται σε μεγάλο βαθμό από την εξέλιξη των παραπάνω προαναφερθέντων τεχνολογιών:

- **Βελτίωση κατανεμημένου πρωτοκόλλου συναίνεσης Proof Of Motion.** Κατά την σχεδίαση και υλοποίηση του αλγορίθμου επιδιώχθηκε η εύρεση του πιο ενεργού χρήστη σε ένα "παράθυρο" του blockchain, και συγκεκριμένα στα τελευταία 60% blocks. Παρόλα αυτά το συγκεκριμένο ποσοστό είναι στατικό, και για αυτόν τον λόγο θα μπορούσε να μετατραπεί σε δυναμικό. Δηλαδή, το ποσοστό αυτό να δημιουργείται μέσω μιας γραμμικής συνάρτησης 2 μεταβλητών, όπου η ανεξάρτητη μεταβλητή να συμβολίζει τον συνολικό αριθμό των συναλλαγών που έχουν γίνει. Η μορφή δηλαδή της συνάρτησης να είναι y=ax+b.
- **Υλοποίηση Caspe-like Proof Of Stake.** Όπως παρατηρήσαμε στην υλοποίηση του κλασικού Proof Of Stake, το οποίο στηρίζεται καθαρά στον πλούτο του κάθε κόμβου και σε έναν παράγοντα τυχαιότητας, τείνει να έχει έναν κεντρικοποιημένο χαρακτήρα. Αυτό συμβαίνει αφού ο πλούσιος γίνεται πλουσιότερος χρησιμοποιώντας μόνο αυτές τις δύο παραμέτρους. Οπότε μια βελτίωση του συγκεκριμένου αλγορίθμου θα ήταν η υλοποίηση του Casper το οποίο προτάθηκε από το Ethereum platform το οποίο χρησιμοποιεί validators και άλλες παραμέτρους.
- **Υλοποίηση υβριδικού κατανεμημένου πρωτοκόλλου συναίνεσης.** Συγκεκριμένα, θα μπορούσα να συνδυάσω το Proof Of Work με το Proof Of Stake ως εξής : Ένα σύνολο υποψηφίων χρηστών για την δημοσίευση του νέου block θα επιλέγονται με βάση την υπολογιστική τους ισχύ, δηλαδή μέσω της εφαρμογής του Proof Of Work. Ύστερα, από αυτό το σύνολο χρηστών ο νικητής θα προέλθει από την υλοποίηση του Proof Of Stake.
- **Μελλοντική έρευνα όσον αφορά το scaling out της εφαρμογής.** Συγκεκριμένα, πρέπει να διερευνηθούν θέματα χώρου της εφαρμογής καθώς και κατά πόσο μπορεί να γίνει scale out σε τεράστιο όγκο δεδομένων.

7

Βιβλιογραφία

[1] Bitcoin Wiki, <<Wei Dai>> [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Wei_Dai . [Πρόσβαση 13 06 2018]

[2] Satoshi Nakamoto, <<A Peer-to-Peer Electronic Cash System>> [Ηλεκτρονικό]. Available: https://bitcoin.org/bitcoin.pdf . [Πρόσβαση 13 06 2018]

[3] «White Paper: Graph Databases in Network and Data Center management» [Ηλεκτρονικό]. Available: https://neo4j.com/whitepapers/network-datacenter-management-graph-databases/ . [Πρόσβαση 15 6 2018].

[4] J. Kurose και K. Ross, Δικτύωση Υπολογιστών, Αθήνα: Μ. Γκιούρδας.

[5] «What is the Web3? The Decentralized Web - Blockchain,» [Ηλεκτρονικό]. Available: https://blockchainhub.net/web3-decentralized-web/. [Πρόσβαση 15 6 2018].

[6] B. Pon, «Blockchain will usher the era of decentralized computing,» 15 4 2016. [Ηλεκτρονικό]. Available: https://blog.bigchaindb.com/blockchain-will-usher-in-the-era-of-decentralised-computing-7f35e94af0b6. [Πρόσβαση 15 6 2018].

[7] Benatallah, B.; Casati, F.; Toumani, F. (2004). "Web service conversation modeling: A cornerstone for e-business automation". *IEEE Internet Computing*.**8**: 46.doi:10.1109/MIC.2004.1260703. [Πρόσβαση 16 6 2018].

[8] Foundation of Peer-to-Peer Computing, Special Issue, Elsevier Journal of Computer Communication, (Ed) Javed I. Khan and Adam Wierzbicki, Volume 31, Issue 2, February 2008 .[Ηλεκτρονικό] [Πρόσβαση 16 6 2018].

[9] 1: Designing Large-scale LANs – Page 31, K. Dooley, O'Reilly, 2002. [Πρόσβαση 16 6 2018].

[10] "BitTorrent: The "one third of all Internet traffic" Myth". TorrentFreak. 24 January 2007.[Ηλεκτρονικό]. [Πρόσβαση 18 6 2018].

[11] J. Ray, «Ethereum introduction,» [H]. Available: https://github.com/ethereum/wiki/wiki/Ethereum-introduction. [19 6

2018].

[12] «Elliptic Curve Cryptography (ECC),» [Ηλεκτρονικό]. Available: https://www.certicom.com/content/certicom/en/ecc.html. [Πρόσβαση 20 6 2018].

[13] Accredited Standards Committee X9, *American National Standard X9.62-2005, Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 16, 2005. [Πρόσβαση 20 6 2018].

[14] Bitcoin Wiki, <<Secp256k1>> [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Secp256k1. [Πρόσβαση 21 06 2018]

[15] Bitcoin Wiki, «Mining» [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Mining. [Πρόσβαση 21 06 2018]

[16] «Bitcoin Block Reward Halving Countdown» [Ηλεκτρονικό]. Available: https://www.bitcoinblockhalf.com . [Πρόσβαση 21 06 2018]

[17] Bitcoin Wiki, «Genesis Block» [Ηλεκτρονικό]. Available: https://en.bitcoin.it/wiki/Genesis_block . [Πρόσβαση 21 06 2018]

[18] Bitcoin Glossary, «Unspent Transaction Output, UTXO» [Ηλεκτρονικό]. Available: https://bitcoin.org/en/glossary/unspent-transaction-output . [Πρόσβαση 23 06 2018]

[19] «Proof of work, bitcoinwiki,» [H]. Available: https://en.bitcoin.it/wiki/Proof_of_work. [Πρόσβαση 23 6 2018].

[20] Jakobsson, Markus; Juels, Ari (1999). "Proofs of Work and Bread Pudding Protocols". *Communications and Multimedia Security*. Kluwer Academic Publishers: 258–272. [Ηλεκτρονικό]. Available: https://www.rsa.com/en-us [Πρόσβαση 23 6 2018].

[21] "NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable" . [Πρόσβαση 26 6 2018].

[22] Gray, Jim (September 1981). "The Transaction Concept: Virtues and Limitations" (PDF). *Proceedings of the 7th International Conference on Very Large Databases*. Cupertino, CA: Tandem Computers. pp. 144–154. Retrieved March 27, 2015. [Ηλεκτρονικό]. Available: http://jimgray.azurewebsites.net/papers/thetransactionconcept.pdf. [Πρόσβαση 26 6 2018].

[23] Codd, Edgar F. (1970). "A Relational Model of Data for Large Shared Data Banks"(PDF). *Communications of the ACM*. **13** (6): 377–387 [Ηλεκτρονικό]. Available: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf . [Πρόσβαση 26 6 2018].

[24] "MapReduce: Simplified Data Processing on Large Clusters" (PDF). [Ηλεκτρονικό]. Available: http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf . [Πρόσβαση 29 6 2018].

[25] Martin, James (1983). *Managing the Data-base Environment*. Englewood Cliffs, New Jersey: Prentice-Hall. p. 381 . [Πρόσβαση 29 6 2018].

[26] «Bloomen,» [Ηλεκτρονικό]. Available: www.bloomen.io. [Πρόσβαση 30 6 2018].

[27] A. L. G. F. V. P. T. V. Georgios Palaiokrassas, «Deploying blockchains for a new paradigm of media experience,» σε *15th International Conference on the Economics of Grids, Clouds, Systems, and Services (GECON 2018)*, Pisa, Italy, 2018.

[28] Greg Walker, «How to import the Bitcoin Blockchain into Neo4j» . [Ηλεκτρονικό]. Available: https://neo4j.com/blog/import-bitcoin-blockchain-neo4j/. [Πρόσβαση 30 6 2018].

[29] A. d. B. H. R. H. A. Thomas Lundqvist, «Thing-to-thing electricity micro payments using blockchain technology,» 2017. [H]. Available: https://ieeexplore.ieee.org/document/8016254/. [30 6 2018].

[30] M. M. K. L. A. N. Andrew Miller, «An Empirical Analysis of Linkability in the Monero Blockchain» 2017. [Ηλεκτρονικό]. Available: https://maltemoeser.de/paper/monerolink.pdf. [Πρόσβαση 30 6 2018].

[31] O. N. A. '. P. Guy Zyskind, «Decentralizing Privacy: Using Blockchain to Protect Personal Data,» 2015. [Ηλεκτρονικό]. Available: https://ieeexplore.ieee.org/document/7163223/. [Πρόσβαση 30 6 2018].

[32] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine. This chapter introduced the Representational State Transfer (REST) architectural style for distributed hypermedia systems. REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems. [Ηλεκτρονικό]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm . [Πρόσβαση 02 7 2018].

[33] "Web Services Addressing (WS-Addressing)" [Ηλεκτρονικό]. Available: https://www.w3.org/Submission/ws-addressing/ . [Πρόσβαση 02 7 2018].

[34] «RESTful API» [Ηλεκτρονικό]. Available: https://searchmicroservices.techtarget.com/definition/RESTful-API. [Πρόσβαση 03 7 2018].

[35] Guttag, John V. (2016-08-12). *Introduction to Computation and Programming Using Python: With Application to Understanding Data*. MIT

Press . [Πρόσβαση 03 7 2018].

[36] Github, «cpython» [Ηλεκτρονικό]. Available: https://github.com/python/cpython. [Πρόσβαση 12 7 2018].

[37] «Flask web development, one drop at a time» [Ηλεκτρονικό]. Available: http://flask.pocoo.org . [Πρόσβαση 15 7 2018]

[38] «Neo4j» [Ηλεκτρονικό]. Available: https://neo4j.com . [Πρόσβαση 15 7 2018].

[39] JSR 220: Enterprise JavaBeans,Version 3.0, EJB 3.0 Expert Group, Sun Microsystems, 2006 . [Πρόσβαση 29 7 2018].

[40] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". *Introduction to Algorithms* (Second ed.). MIT Press and McGraw–Hill. pp. 595–601. [Πρόσβαση 30 7 2018].

[41] «Using Neo4j from Python» [Ηλεκτρονικό]. Available: https://neo4j.com/developer/python/. [Πρόσβαση 01 8 2018].

[42] «Intro to Cypher» [Ηλεκτρονικό]. Available: https://neo4j.com/developer/cypher-query-language/. [Πρόσβαση 04 8 2018].

[43] «Neo4j Desktop User Interface Guide» [Ηλεκτρονικό]. Available: https://neo4j.com/developer/guide-neo4j-desktop/ . [Πρόσβαση 04 8 2018].

[44] «hashlib – Secure hashes and message digests» [Ηλεκτρονικό]. Available: https://docs.python.org/2/library/hashlib.html. [Πρόσβαση 21 8 2018].

[45] Dmitry Khovratovich, Christian Rechberger & Alexandra Savelieva (2011). "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family" (PDF). *IACR Cryptology ePrint Archive*. 2011:286.[Ηλεκτρονικό]. Available: https://eprint.iacr.org/2011/286.pdf . [Πρόσβαση 22 8 2018].

[46] Becker, Georg (2008-07-18). "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis" (PDF). Ruhr-Universität Bochum. p. 16. Retrieved 2013-11-20.

[Ηλεκτρονικό]. Available: http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf . [Πρόσβαση 22 8 2018].

[47] King, Sunny. "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake". [Ηλεκτρονικό]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf [Πρόσβαση 27 8 2018].

[48] Github, «Proof Of Stake FAQs» [Ηλεκτρονικό]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#what-is-proof-of-stake . [Πρόσβαση 29 8 2018].

[49] «Neo4j Browser User Interface Guide» [Ηλεκτρονικό]. Available: https://neo4j.com/developer/guide-neo4j-browser/ . [Πρόσβαση 30 8 2018].

["Douglas Crockford — The JSON Saga". YouTube. 28 August 2011.

50] Retrieved 23 September 2016. [Πρόσβαση 01 9 2018].

[«ethereum Blockchain App Platform» [Ηλεκτρονικό]. Available:
51] https://www.ethereum.org [Πρόσβαση 02 9 2018].

Παράρτημα Ι: Κώδικες

Στο παράρτημα αυτό βρίσκεται ο κώδικας που ορίζουν τους Flask πόρους, οι οποίοι εξυπηρετούν τα http αιτήματα των κόμβων, που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας. ***Ο συνολικός πηγαίος κώδικας (συμπεριλαμβανομένων των κλάσεων Block και Blockchain, Merkle Tree library) βρίσκεται στην σελίδα :*** https://gitlab.com/george_fr/PoW_neo4j_implem.git
Σημειώνεται πως για την πρόσβαση στο παραπάνω Gitlab Project, πρέπει να ζητηθεί access από τον συγγραφέα της παρούσας διπλωματικής εργασίας.

1 Flask Resources Source Code

Ο πηγαίος κώδικας των Flask Resources:

```
class getBlock(Resource):

     def post(self):
       info = request.get_json()
       index = int(info.get('index'))
       response = {
         'block':my_blockchain.get_serialized_block(index)
       }
       return jsonify(response)

class createTransaction(Resource):
     def post(self):
       trans_data = request.get_json()
       index = my_blockchain.create_transaction(**trans_data)
       if index!= -1 :
           response = {
           'message' : 'Transaction submitted to the network',
           'block_index': index
          }

       else:
           response = {
           'message' : 'Not submitted to the network. Not enough
tokens for this transaction',
        }
       return jsonify(response)


class getblockchain(Resource):
     def get(self):
```

```
        response = {
           'chain': my_blockchain.get_serialized_chain()
          }
        return jsonify(response)



class choose_pos(Resource):
     def get(self):
       winner = my_blockchain.mine_pos()
       update_neo4j =
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users)
       if update_neo4j:
             print("Neo4j Graph Database is successfully updated!!!")

       response = {
             'message':'The Proof Of Stake winner of PoS Protocol
is : ',
             'user':winner
         }
       return jsonify(response)



class choose_poa(Resource):
     def get(self):
       winner = my_blockchain.mine_poa()


       update_neo4j =
my_blockchain.updatedb(db,db_blocks,db_transactions,db_users)
       if update_neo4j:
             print("Neo4j Graph Database is successfully updated!!!")


       response = {
             'message':'The Proof of Motion winner is : ',
             'user':winner
       }
       return jsonify(response)


class mineBlock(Resource):
     def post(self):
       info = request.get_json()
       block = my_blockchain.mine_block(info.get('miner'))
        response = {
             'message':'Successfully Mined'

       }

       return jsonify(response)



class addingNode(Resource):
     def post(self):
         info = request.get_json()
         my_blockchain.create_node(info.get('address'))
         print info.get('address')
```

```
         response = {
             'message': "New Peer in the network! ",
             'total_nodes' : len(my_blockchain.nodes),
             'nodes' : list(my_blockchain.nodes)
         }
         return jsonify(response)



class getConnectedNodes(Resource):
     def get(self):
     response = {
     'message':" *** Connected Peers in the Network *** ",
     'nodes' : list(my_blockchain.nodes)
     }
     return jsonify(response)


class traverseBlockchain(Resource):
     def get(self):
       examined_chain = my_blockchain.chain
       all_traversed =
my_blockchain.traverse_blockchain(examined_chain)
       if all_traversed:
         response = {
         'message' : 'The traversal all the way back to Genesis Block
is done!'
        }
       else:
          response = {
         'message' : 'Something went wrong... Please retry...'
       }
       return jsonify(response)



class getLatest(Resource):
     def get(self):
       response = {
        'Latest Block': str(my_blockchain.getLatestBlock())
       }
        return jsonify(response)



class findCurrentBalance(Resource):
     def post(self):
       info = request.get_json()
       total_balance =
my_blockchain.find_current_balance(hashlib.sha256(str(info.get('addre
ss'))).hexdigest())
       response = {
         'message': "Here is the total Balance of this address! ",
         'total_balance' : total_balance
         }
       return jsonify(response)
```

```
        def get(self):
           peer_chains =
my_blockchain.get_peers_blockchain(my_blockchain)
           if not peer_chains :
               response = {
                  'message': 'No available peer at the moment!
Please add some peers...'
                }
               return jsonify(response)

           longest_chain = max(peer_chains, key=len)
           if len(my_blockchain.chain) >= len(longest_chain):
              response = {
                 'message': 'We have the latest version of the
blockchain...',
                 'chain': my_blockchain.get_serialized_chain()
              }
           else:
                  my_blockchain.chain =
[my_blockchain.get_data_blockchain(block) for block in longest_chain]
                  response = {
                    'message': 'The blockchain is synced. You have
the latest version...',
                    'chain': my_blockchain.get_serialized_chain()
    }
           return jsonify(response)
```

2 Flask Resources URLs

```
api.add_resource(getblockchain, '/getblockchain')
api.add_resource(createTransaction,'/create_transaction')
api.add_resource(mineBlock,'/mine')
api.add_resource(addingNode,'/addnode')
api.add_resource(getConnectedNodes,'/getnodes')
api.add_resource(getBlock, '/getblock')
api.add_resource(Consensus, '/consensus')
api.add_resource(getLatest,'/getlatestblock')
api.add_resource(findCurrentBalance,'/find_balance')
api.add_resource(choose_pos,'/pos')
api.add_resource(choose_poa,'/pom')
api.add_resource(traverseBlockchain,'/traverse')
```

3 Main Class

```
    if __name__ == '__main__':
        from argparse import ArgumentParser

        # Connect to the Neo4j Graph Database through the Neo4j Rest
Client
        # And then Create the 3 Layers in our Graph Database :
        # 1st Layer : Blocks
        # 2nd Layer : Transactions
        # 3rd Layer : Users
       db = GraphDatabase("http://localhost:7474" , username =
"neo4j", password = "123456789")
        db_blocks = db.labels.create("Blocks")
        db_transactions = db.labels.create("Transactions")
        db_users = db.labels.create("Users")

        my_blockchain =
Blockchain(db,db_blocks,db_transactions,db_users)


        parser = ArgumentParser()
        parser.add_argument('-H', '--host', default='0.0.0.0')
        parser.add_argument('-p', '--port', default=5000, type = int)
        args = parser.parse_args()
         app.run(host = args.host, port =
args.port ,debug=True,use_reloader=False)
```

Σημείωση: Οι οδηγίες για την εγκατάσταση της εφαρμογής καθώς και οι εντολές για να "τρέξει" (αν και έχουν περιγραφεί παραπάνω) παρουσιάζονται εκτενώς στο αρχείο README.md file, στο Gitlab!