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

Ανάπτυξη μηχανισμού συναίνεσης για την επιλογή διακλαδώσεων σε δέντρα blockchain

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Τσούλιας

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου

Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβριος 2019

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

Ανάπτυξη μηχανισμού συναίνεσης για την επιλογή διακλαδώσεων σε δέντρα blockchain

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Τσούλιας

Επιβλέπουσα: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 31η Οκτωβρίου 2019.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Συμεών Παπαβασιλείου	Εμμανουήλ Βαρβαρίγος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2019

.....

Κωνσταντίνος Τσούλιας

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

Περίληψη

Η ψηφιοποίηση, η διαδικασία μετατροπής των φυσικών αντικειμένων σε ψηφιακή μορφή, συμβαίνει όλο και περισσότερο τα τελευταία χρόνια, αλλάζοντας πολλές πτυχές της καθημερινής μας ζωής. Το 2008 ο Shatoshi Nakamoto με τη δημοσίευση μιας ερευνητικής εργασίας με τίτλο «A Peer-to-Peer Electronic Cash System» εξηγούσε πως η ψηφιακή πληροφορία μπορεί να αποτελέσει ένα νόμισμα, το οποίο δεν θα εκδίδεται ούτε θα ελέγχεται από κάποια κυβέρνηση ή κεντρική τράπεζα. Το πρωτοποριακό αυτό νόμισμα το ονόμασε Bitcoin και βασίζεται στην τεχνολογία blockchain, ένα δημόσιο κατανεμημένο λογιστικό βιβλίο, στο οποίο καταγράφονται οι συναλλαγές και υποστηρίζεται από ένα δίκτυο ομότιμων κόμβων. Το Bitcoin αποτέλεσε την αφετηρία για ένα πλήθος νέων κρυπτονομισμάτων, όπως ονομάστηκαν, με τα οποία εξετάζονται σε βάθος τα πλεονεκτήματα του blockchain και αναπτύσσονται νέες ιδέες και τεχνολογίες σε αυτό, που θα μπορούσαν ενδεχομένως να μετατρέψουν τον τρόπο με τον οποίο οι άνθρωποι επιλέγουν να συναλλάσσονται παγκοσμίως..

Η παρούσα διπλωματική εργάζεται προς αυτήν την κατεύθυνση με την δημιουργία μίας πραγματικής blockchain εφαρμογής και την ανάπτυξη σύγχρονων μηχανισμών σε αυτήν, ώστε να επιτυγχάνεται η συναίνεση των χρηστών σε μία κανονική αλυσίδα συναλλαγών. Παράλληλα θα χρησιμοποιήσουμε τη βάση δεδομένων γράφου Neo4j τόσο για την ταχύτητα που προσφέρει στην εξυπηρέτηση αιτημάτων, όσο και για την οπτική αναπαράσταση του blockchain. Τέλος θα εξετάσουμε την αντοχή του συστήματος σε επιθέσεις κακόβουλων χρηστών και την αποτελεσματικότητα του μηχανισμού στην επίτευξη της συναίνεσης και την οριστικοποίηση των συναλλαγών.

Λέξεις κλειδιά

Blockchain, Casper, Βάση Δεδομένων Γράφου – Neo4j, Πρωτόκολλα συναίνεσης, Consensus, Ερωτήματα Cypher, Συναλλαγές, Python, Flask, Επίθεση 51%, Catastrophic Crashes

Abstract

Digitization, the process of converting physical objects into a digital format, is happening more and more in recent years, changing many aspects of our daily lives. In 2008, Shatoshi Nakamoto, published a research paper entitled "A Peer-to-Peer Electronic Cash System", in which he explained that digital information can be used as a currency, that will not be issued or controlled by any government or central bank. This pioneering currency, called Bitcoin, is based on blockchain technology, a publicly distributed ledger, for recording transactions, that is supported by a peer-to-peer network. Bitcoin has been the starting point for a multitude of new cryptocurrencies, as they are called, that examine in depth the benefits of blockchain technology and develop new ideas in it, that could potentially transform the way people choose to transact globally.

The present diploma thesis is working towards this, by creating a real blockchain application and developing modern mechanisms in it, in order to get users' to consent in a canonical chain of transactions. At the same time, we will use the Neo4j graph database, capitalizing on both the fast speeds at serving requests and the good visual representation of the blockchain database, that Neo4j provides. Finally, we will examine the system's resilience to malicious attacks and the mechanism's effectiveness in reaching consensus and finalizing transactions.

Key Words

Blockchain, Casper, Graph Database – Neo4j, Consensus protocols, Cypher Queries, Transactions, Python, Flask, 51% Attack, Catastrophic Crashes

Ευχαριστίες

Κατ’ αρχάς θα ήθελα να ευχαριστήσω την καθηγήτρια κυρία Θεοδώρα Βαρβαρίγου που με εμπιστεύθηκε με την εκπόνηση αυτής της διπλωματικής εργασίας. Θερμά ευχαριστώ στον δόκτορα Αντώνη Λίτκε και στον διδάκτορα Γιώργο Παλαιοκρασσά για τις εποικοδομητικές συζητήσεις και τις χρήσιμες κατευθύνσεις, οι όποιες με βοήθησαν σημαντικά, κατά την ενασχόλησή μου με αυτό το πολύ ενδιαφέρον και σύγχρονο αντικείμενο.

Ασφαλώς, τίποτα δε θα ήταν εφικτό χωρίς τη στήριξη και την αγάπη της οικογένειας και των φίλων μου.

Κωνσταντίνος Τσούλιας

Αθήνα, Οκτώβριος 2019

Περιεχόμενα

Κατάλογος Σχημάτων

1
Εισαγωγή

Η ραγδαία ανάπτυξη της τεχνολογίας τις τελευταίες δεκαετίες, γίνεται ολοένα και πιο αισθητή, αλλάζοντας την καθημερινότητα των ανθρώπων. Οι τεχνολογίες ψηφιοποίησης επέτρεψαν τη μετατροπή παραδοσιακών μορφών αποθήκευσης πληροφοριών, όπως το χαρτί, σε δυαδικό κώδικα αποθηκευμένο σε ηλεκτρονικούς υπολογιστές. Η πλήρης αξιοποίηση, όμως, της ψηφιακής αυτής πληροφορίας πραγματοποιήθηκε με τη δημιουργία του Ίντερνετ. Μέσω του διαδικτύου, τεράστιος όγκος πληροφοριών μπορεί να διανύσει σε ελάχιστο χρόνο τεράστιες αποστάσεις. Πλέον επιχειρήσεις, τράπεζες, οργανισμοί, σχολεία, κρατικές υπηρεσίες πραγματοποιούν "ψηφιακό μετασχηματισμό", μετατρέποντας την ψηφιοποίηση σε νέες διαδικασίες, δραστηριότητες και συναλλαγές. Η τάση αυτή για ψηφιοποίηση μπορεί να επηρεάσει ακόμα πιο άμεσα την οικονομία, μετατρέποντας το ίδιο το νόμισμα σε ηλεκτρονική πληροφορία, αλλάζοντας ταυτόχρονα την αντίληψη του κόσμου ως προς αυτό. Η πρώτη ρεαλιστική προσπάθεια σε αυτήν την κατεύθυνση πραγματοποιήθηκε το 2008 με τη δημοσίευση μιας ερευνητικής εργασίας με τίτλο «A Peer-to-Peer Electronic Cash System» από τον Shatoshi Nakamoto [1]. Στην εργασία αυτή ο Shatoshi – ο οποίος παραμένει άγνωστος μέχρι και σήμερα – μιλά για το Bitcoin, ένα πλήρως ψηφιακό νόμισμα που δεν εκδίδεται ούτε ελέγχεται από κάποια κυβέρνηση ή κεντρική τράπεζα. Αντίθετα, βασίζεται σε ένα peer-to-peer δίκτυο από ομότιμους κόμβους, οι οποίοι μπορούν να πραγματοποιούν συναλλαγές με Bitcoin. Κάθε συναλλαγή που πραγματοποιείται στο δίκτυο αποθηκεύεται στο blockchain, ένα δημόσιο λογιστικό βιβλίο, που χρησιμοποιείται για την παρακολούθηση και την επαλήθευση όλων των συναλλαγών. Συγκεκριμένα, οι συναλλαγές αποθηκεύονται σε blocks τα οποία συνδέονται σειριακά μεταξύ τους, με βάση την χρονική στιγμή δημιουργίας τους. Στο Bitcoin οι κόμβοι είναι ανώνυμοι, ωστόσο υπάρχει διαφάνεια στις συναλλαγές, αφού καθένας από αυτούς διατηρεί ένα αντίγραφο ολόκληρου του blockchain. Για την ασφαλή λειτουργία και την ακεραιότητα του νομίσματος, οι κόμβοι υπακούν σε ένα σύνολο κρυπτογραφικών πρωτοκόλλων. Λόγω των τελευταίων το Bitcoin ονομάζεται κρυπτονόμισμα. Σήμερα υπάρχουν πάνω από 1500 κρυπτονομίσματα και εφαρμογές που χρησιμοποιούν την τεχνολογία του blockchain, ώστε να λειτουργούν αποκεντρωμένα, χωρίς δηλαδή την παρουσία κάποιου μεσάζονται ή διαχειριστή.

1.1 Αντικείμενο της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία εστιάζει στην τεχνολογία του blockchain και συγκεκριμένα την εφαρμογή του Casper, ενός μηχανισμού για την επίτευξη συναίνεσης μεταξύ των κόμβων του δικτύου, που συνδυάζει τους αλγόριθμους proof of stake και «Byzantine Fault Tolerance» .

Σκοπός είναι η διερεύνηση και η αντιμετώπιση κάποιων εκ των μεγαλύτερων προβλημάτων του blockchain, όπως το Nothing-At-Stake problem και η επιλογή της κανονικής αλυσίδας συναλλαγών στα δέντρα της blockchain.

Η καινοτομία του Casper είναι πολλή πρόσφατη και ως εκ τούτου, δεν έχει εφαρμοστεί ακόμα σε κάποιο κρυπτονόμισμα. Για το λόγο αυτό, θα δημιουργήσουμε το δικό μας blockchain από την αρχή, πάνω στο οποίο θα εφαρμόσουμε και θα εξετάσουμε τη λειτουργία του. Η υλοποίηση της εφαρμογής για τους κόμβους (nodes) θα γίνει σε γλώσσα προγραμματισμού Python, ενώ για το blockchain θα χρησιμοποιηθεί η βάση δεδομένων γράφου Neo4j. Με τον τρόπο αυτό, τα ερωτήματα προς τη βάση θα γίνονται πολύ γρηγορότερα, σε σχέση με την σειριακή προσπέλαση του blockchain. Επίσης το γραφικό περιβάλλον που προσφέρει η Neo4j θα βοηθήσει στην καλύτερη αναπαράσταση και κατανόηση της λειτουργίας του Casper και του blockchain. Το Casper εισάγει νέες έννοιες στην τεχνολογία του blockchain (voting, slashing) και αλλάζει άλλες, προϋπάρχουσες (validators) .

Η διαδικασία της υλοποίησης, μπορεί να διακριθεί στην κατασκευή της αποκεντρωμένης εφαρμογής ακολουθώντας τις βασικές αρχές που διέπουν το blockchain (Bitcoin standards) και στην κατασκευή του Casper, όπως περιγράφεται στην αντίστοιχη εργασία των Buterin και Griffith [2] . Θα εξετάσουμε την λειτουργία του Casper με διαφορετικούς μηχανισμούς σύστασης block καθώς και την αντοχή του δικτύου σε επιθέσεις.

1.2 Οργάνωση Κειμένου

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

Το παρόν Κεφάλαιο αποτελεί την εισαγωγή.

Το Κεφάλαιο 2 περιγράφει το θεωρητικό υπόβαθρο της παρούσας διπλωματικής εργασίας. Στην αρχή, γίνεται μια σύντομη αναφορά στο αποκεντρωμένο διαδίκτυο, Web3, και στα δίκτυα ομότιμων κόμβων (peer-to-peer networks), πάνω στα οποία στηρίζεται το blockchain και κατά συνέπεια, η σύγχρονη αυτή μορφή του Ίντερνετ. Έπειτα παρουσιάζουμε τα βασικά στοιχεία του Bitcoin blockchain, καθώς και των πρωτοκόλλων σύστασης blocks που χρησιμοποιούν τα κρυπτονομίσματα. Στην συνέχεια, περιγράφουμε τα σημαντικά προβλήματα που συνδέονται με την τεχνολογία του blockchain, όπως το Consensus Problem, το Energy Problem και το Nothing-at-Stake problem. Τέλος, θα εστιάσουμε στη λειτουργία του Casper, στις αλλαγές που επιφέρει στην υπάρχουσα τεχνολογία και στον τρόπο με τον οποίο επιχειρεί να λύσει τα προαναφερθέντα προβλήματα.

Στο Κεφάλαιο 3 περιγράφουμε τα εργαλεία και τις τεχνολογίες που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής. Συγκεκριμένα, γίνεται αναφορά στις δικτυακές υπηρεσίες RESTful, μέσω των οποίων επιτεύχθηκε η επικοινωνία των κόμβων του δικτύου. Επίσης, μιλάμε για την γλώσσα προγραμματισμού Python με την οποία δημιουργήσαμε το λογισμικό των κόμβων του δικτύου. Τέλος, παρουσιάζεται και η Neo4j βάση δεδομένων γράφου, η οποία χρησιμοποιήθηκε για την απεικόνιση του blockchain, στο οποίο έχουν πρόσβαση όλοι οι κόμβοι.

Το Κεφάλαιο 4 αφορά τη διαδικασία σχεδίασης της εφαρμογής. Στην αρχή, αναφερόμαστε στη γενική δομή της εφαρμογής, στις κλάσεις που την αποτελούν και περιγράφουμε τις λειτουργίες τους. Έπειτα, περιγράφεται η διαδικασία υλοποίησης της εφαρμογής, ως δύο επιμέρους κομμάτια. Το πρώτο κομμάτι σχετίζεται με την σχεδίαση του blockchain, σύμφωνα με τα καθιερωμένα πρότυπα. Αυτό περιλαμβάνει τις έννοιες: mining, proof-of-work, proof-of-stake, hashing, branch. Το δεύτερο κομμάτι αφορά την υλοποίηση του Casper και τις καινοτομίες που επιφέρει στην προαναφερθείσα δομή. Οι καινοτομίες αυτές μπορεί αν είναι είτε καινούργιες έννοιες, όπως slashing, voting, staking, justification, finalization είτε αλλαγές σε προϋπάρχουσες δομές, όπως οι validators και η αποθηκευμένη πληροφορία των blocks.

Στο Κεφάλαιο 5 επιδεικνύεται ο τρόπος λειτουργίας της εφαρμογής, όσον αφορά τους χρήστες της. Συγκεκριμένα, δείχνουμε το ρόλο των διαφορετικών χρηστών (απλοί nodes, miners, validators) στο δίκτυο, τις ελευθερίες και τους περιορισμούς τους. Επιπλέον, παρουσιάζουμε την λειτουργία του Casper και κυρίως το πώς ανταποκρίνεται στα διάφορα σενάρια χρήσης, με την ύπαρξη ή μη κακόβουλων χρηστών.

Στο Κεφάλαιο 6 εξετάζουμε την αντοχή της εφαρμογής σε επιθέσεις κακόβουλων χρηστών. Παράλληλα εξετάζουμε την επιρροή διαφόρων παραμέτρων στην αποτελεσματικότητα του Casper.

Το Κεφάλαιο 7 αποτελεί τον επίλογο της διπλωματικής, όπου και συνοψίζονται οι παρατηρήσεις μας και προτείνονται ιδέες για την περαιτέρω εξέλιξη του αντικειμένου αυτής της εργασίας.

Το Κεφάλαιο 8 αποτελείται από την σχετική βιβλιογραφία που αξιοποιήθηκε κατά την εκπόνηση της διπλωματικής εργασίας, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο.

2
Θεωρητικό υπόβαθρο και σχετικές εργασίες

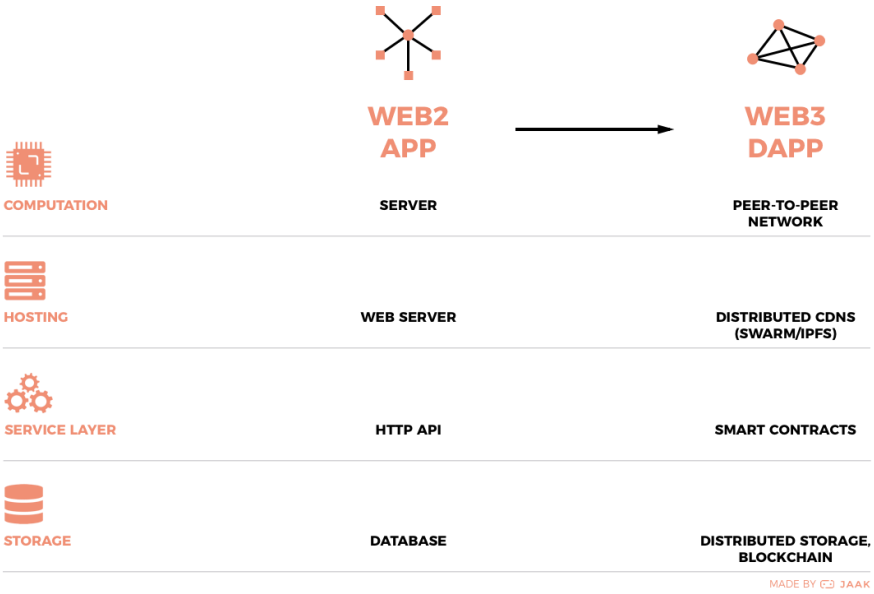
Στο κεφάλαιο αυτό διατυπώνεται το θεωρητικό υπόβαθρο που σχετίζεται με την τεχνολογία του blockchain και του Casper. Αρχικά γίνεται σύντομη παρουσίαση της εξέλιξης του διαδικτύου έως την πιο σύγχρονη μορφή του, Web3, που ξεκίνησε με την δημιουργία του Bitcoin και την ιδέα της αποκεντροποίησης. Έπειτα, αναφερόμαστε στην τεχνολογία των δικτύων ομότιμων κόμβων, που αποτελούν το βασικό συστατικό της blockchain τεχνολογίας και συνεπώς του αποκεντρωμένου ιστού. Στην συνέχεια, παρουσιάζονται οι βασικές αρχές του blockchain , δίνοντας έμφαση στη λειτουργία του κρυπτονομίσματος Bitcoin. Στην ίδια παράγραφο, γίνεται ειδική αναφορά στα πρωτόκολλα σύστασης blocks, που χρησιμοποιούνται σήμερα στα κρυπτονομίσματα. Εν συνεχεία, περιγράφουμε τα βασικά προβλήματα που συνδέονται με το blockchain και συγκεκριμένα: το Consensus Problem ή Byzantine Fault Tolerance, το πρόβλημα Ενέργειας και το πρόβλημα Nothing-at-Stake. Κατόπιν τούτου, γίνεται αναλυτική παρουσίαση του Casper και επιδεικνύεται ο τρόπος, με τον οποίο το τελευταίο επιχειρεί να δώσει λύση στα παραπάνω προβλήματα. Σε αυτό το πλαίσιο, θα δείξουμε πως το Casper επιτυγχάνει την συναίνεση, πως δημιουργεί θετικά κίνητρα στους χρήστες του δικτύου, τα είδη επιθέσεων που μπορεί αν δεχθεί και το πως τις αντιμετωπίζει. Τέλος, θα αναφερθούμε σε εργασίες σχετικές με το αντικείμενο της παρούσας διπλωματικής.

Εικόνα 2. 1 Το κρυπτονόμισμα Bitcoin

2.1 Το αποκεντρωμένο διαδίκτυο – Web3

Η εξέλιξη του παγκόσμιου ιστού WWW έχει εισέλθει στην τρίτη φάση, του αποκεντρωμένου ιστού «Decentralized Web» ή Web3 . Έχουν προηγηθεί η αρχική απλή μορφή του, που άλλαξε το τοπίο στη μετάδοση της πληροφορίας σε παγκόσμιο επίπεδο, και η δεύτερη φάση Web2, με κύρια χαρακτηριστικά την ανάπτυξη των κοινωνικών δικτύων και του ηλεκτρονικού εμπορίου, σχηματίζοντας έτσι μια ακόμα πιο εκτεταμένη και πιο καλά οργανωμένη μορφή της παγκόσμιας πληροφοριόσφαιρας. Και στις δύο φάσεις της μετεξέλιξης, το διαδίκτυο διατήρησε μία κεντρική δομή, με εταιρίες ή οργανισμούς να εκτελούν ρόλο μεσάζοντα σχεδόν σε κάθε εφαρμογή ή ιστότοπο. Είναι προφανές λοιπόν, ότι προϋπόθεση για τη σωστή λειτουργία του διαδικτύου, είναι η εμπιστοσύνη των χρηστών ως προς αυτές τις κεντρικές οντότητες [3].

Το blockchain, του αποκεντρωμένου ιστού «Decentralized Web» [4] επεκτείνει τη χρήση του WWW σε νέες δυνατότητες καταργώντας την ανάγκη για εμπιστοσύνη σε ενδιάμεσους και κεντρικούς οργανωτές, με τη διαμόρφωση μιας μεγαλύτερης αυτοτέλειας στους χρήστες. Με αυτό τον τρόπο γίνεται υπέρβαση των αδυναμιών του Web2, με πιο σημαντική καινοτομία την απελευθέρωση της ροής και της χρήσης των πληροφοριών από έναν κυρίαρχο ελεγκτικό μηχανισμό. Ωστόσο, η εκτεταμένη αυτή προσβασιμότητα των χρηστών σε πληροφορίες, που προσφέρει το αποκεντρωμένο διαδίκτυο, θέτει ορισμένους περιορισμούς στην λειτουργία του. Αυτοί οι περιορισμοί αφορούν στη μειωμένη δυνατότητα αποθήκευσης μεγάλου όγκου πληροφοριών και στην έλλειψη ιδιωτικότητας των χρηστών. Είναι όμως σαφές ότι οι τάσεις στο επόμενο διάστημα θα αφορούν την μετεξέλιξη της επικοινωνίας, της επεξεργασίας και της αποθήκευσης σε όλο και πιο αποκεντρωμένο επίπεδο, αφού παρέχεται νέα δυναμική στη ροή της πληροφορίας.

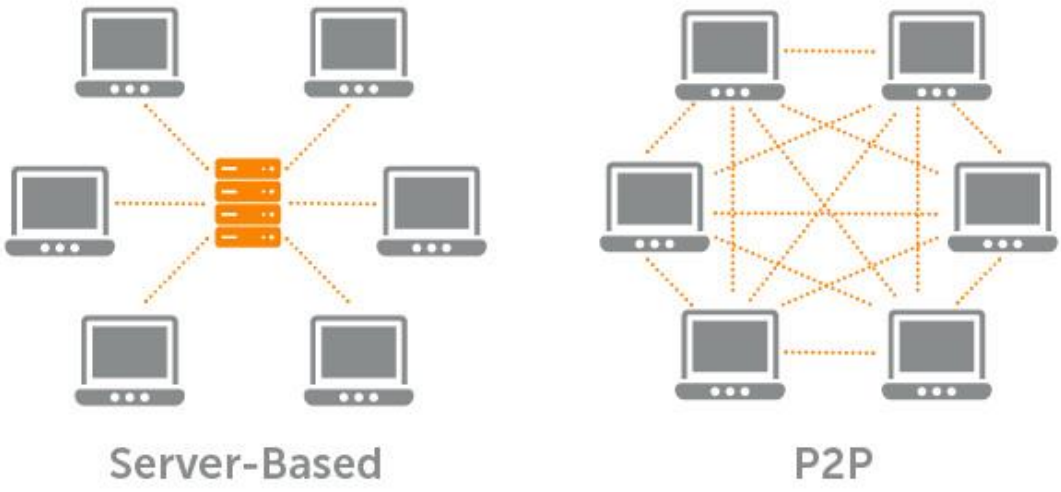


Εικόνα 2. 2 Μετάβαση από Web2 σε Web3

2.2 Δίκτυα ομότιμων κόμβων

Όπως γίνεται αντιληπτό τόσο το αποκεντρωμένο διαδίκτυο όσο και η τεχνολογία του blockchain βασίζονται στα δίκτυα ομότιμων κόμβων ή αλλιώς peer-to-peer (P2P) networks [5]. Στην παρούσα παράγραφο θα περιγράψουμε την αρχιτεκτονική αυτών των δικτύων, θα δούμε τα πλεονεκτήματα που προσφέρουν, θα εξετασουμε τις αδυναμίες του και θα τα συγκρίνουμε με τα δίκτυα πελάτη-εξυπηρετητή [6] που εδρεύουν στη σημερινή μορφή του διαδικτύου.

Τα δίκτυα peer-to-peer δεν συναντώνται συχνά στην τωρινή μορφή του διαδικτύου. Οι περισσότεροι ιστότοποι και εφαρμογές οργανώνονται με το μορφή πελάτη-εξυπηρετητή γνωστό και ως client-server model. Τα δίκτυα αυτά βασίζονται στη λειτουργία ενός κεντρικού υπολογιστή (server), ο οποίος λειτουργεί ακατάπαυστα και παρέχει τις απαιτούμενες πληροφορίες στους υπόλοιπους χρήστες του δικτύου (clients), χωρίς οι τελευταίοι να επικοινωνούν μεταξύ τους. Δηλαδή, υπάρχει ένα κεντρικό σημείο αποθήκευσης και αποστολής δεδομένων που συνεπάγεται σε ένα κεντρικό σημείο ελέγχου και ασφάλειας. Το μοντέλο πελάτη-εξυπηρετητή είναι εύκολα υλοποιήσιμο και προσφέρει μία σχετική σταθερότητα, αφού η σωστή λειτουργία του δικτύου δεν εξαρτάται από πολλούς ανεξάρτητους κόμβους. Ωστόσο, δεν παύει να έχει και μειονεκτήματα, όπως το υψηλό κόστος μίας τέτοιας υλοποίησης, ειδικά όταν πρόκειται για μεγάλα δίκτυα. Επιπλέον, λόγω της κεντρικής δομής του δικτύου υπάρχει Μοναδικό Σημείο Αποτυχίας (Single Point of Failure – SPOF) [7], αφού μία επίθεση ή βλάβη στον server μπορεί να προκαλέσει πρόβλημα στο δίκτυο και να χαθούν πολλά δεδομένα.



Εικόνα 2. 3 Αρχιτεκτονική δικτύων α) πελάτη-εξυπηρετητή β) ομότιμων κόμβων.

Στα δίκτυα peer-to-peer δεν υπάρχει κεντρικός server ή κόμβος. Αντίθετα, κάθε κόμβος του δικτύου λειτουργεί ταυτόχρονα και σαν πελάτης και σαν εξυπηρετητής, με τους χρήστες του δικτύου να μπορούν να στέλνουν απευθείας δεδομένα μεταξύ τους. Επιπλέον, λόγω της έλλειψης μιας κεντρικής αρχής υπεύθυνης για τον συντονισμό του δικτύου και την εκτέλεση υπολογισμών, οι κόμβοι προσφέρουν υπολογιστικούς πόρους για την σωστή λειτουργία του δικτύου. Η πληροφορία είναι κατανεμημένη στους κόμβους του δικτύου, οι οποίο έχουν τα ίδια προνόμια και το ίδιο ρόλο στο δίκτυο. Ουσιαστικά τα peer-to-peer δίκτυα επιτρέπουν την κοινή χρήση υπολογιστικών πόρων και την εκτέλεση υπολογισμών με κατανεμημένο τρόπο, σε αντίθεση με το μοντέλο πελάτη-εξυπηρετητή, όπου ο server είναι αρμόδιος για όλες τις παραπάνω λειτουργίες.

Η αφαίρεση του κεντρικού server, ισοδυναμεί και με μικρότερο κόστος σε σχέση με το client-server μοντέλο. Επιπλέον το δίκτυο είναι πιο ανεκτικό σε αποτυχία υπολογιστών του δικτύου, αφού εδώ οι κόμβοι δρουν σαν servers και δεν υπάρχει μοναδικό σημείο αστοχίας. Σημαντικό επίσης πλεονέκτημα των P2P δικτύων είναι και η αυτοκλιμακωσιμότητά τους. Δηλαδή, παρότι η ύπαρξη πολλών κόμβων σε οποιοδήποτε δίκτυο ισοδυναμεί με μεγαλύτερη κινητικότητα και αύξηση φόρτου εργασίας, τα δίκτυα peer-to-peer μπορούν να ανταπεξέλθουν αποτελεσματικά σε μια τέτοια περίπτωση, αφού οι clients δρουν ταυτόχρονα ως εξυπηρετητές, «απορροφώντας» ουσιαστικά την κίνηση, την οποία δημιουργούν..

Τα προβλήματα των peer-to-peer δικτύων εστιάζονται στον συντονισμό, την ασφάλεια και τη σταθερότητα. Ο συντονισμός του δικτύου και των κόμβων είναι πολύ δυσκολότερος σε σχέση με τα client-server δίκτυα, λόγω της έλλειψης μιας κεντρικής αρχής. Εξίσου δύσκολη είναι και η επίτευξη της ασφάλειας, αφού η ανοιχτή φύση του δικτύου απαιτεί ασφάλεια σε κάθε κόμβο, και όχι μόνο σε έναν κεντρικό server. Όσον αφορά τη σταθερότητα αυτή εξαρτάται άμεσα από τους δύο προηγούμενους παράγοντες. Σε κάθε περίπτωση οι χρήστες ενός P2P θα πρέπει να διακατέχονται από θετικά κίνητρα [8] και να δρουν προς όφελος του δικτύου. Τα κρυπτονομίσματα και το blockchain φροντίζουν ώστε να δημιουργούν τέτοιες ωφέλημες συμπεριφορές, επιβραβεύοντάς τες με κάποια χρηματική ανταμοιβή.

2.3 Blockchain και πρωτόκολλα

Με τον όρο blockchain αναφερόμαστε σε μια συνεχώς αναπτυσσόμενη, ψηφιακή λίστα από αρχεία, γνωστά ως blocks, τα οποία είναι ασφαλή και σειριακά συνδεδεμένα μεταξύ τους, με την χρήση κρυπτογραφίας. Στα blocks αποθηκεύονται τα στοιχεία των δοσοληψιών, μεταξύ των χρηστών ενός δικτύου. Λειτουργούν δηλαδή, σαν ένα δημόσιο και κατανεμημένο λογιστικό βιβλίο, με ρόλο την ασφαλή καταγραφή και επαλήθευση όλων των συναλλαγών ενός δικτύου.

2.3.1 Η λειτουργία του blockchain

Όπως προαναφέρθηκε, η τεχνολογίες του blockchain βρίσκουν εφαρμογή, σε δίκτυα ομότιμων κόμβων. Οι κόμβοι, δηλαδή οι χρήστες του δικτύου, μπορούν να πραγματοποιούν συναλλαγές μεταξύ τους, τις οποίες και καταχωρούν στο blockchain. Συγκεκριμένα, όταν ένας χρήστης πραγματοποιεί μία συναλλαγή, κοινοποιεί τα στοιχεία της στο δίκτυο. Τα στοιχεία αυτά ελέγχονται ως προς την εγκυρότητα τους από τους υπόλοιπους κόμβους του δικτύου (η διαδικασία μπορεί να διαφέρει από εφαρμογή σε εφαρμογή) και αν αυτά γίνουν αποδεκτά, η συναλλαγή καταχωρείται στο transaction pool των κόμβων, μαζί με πολλές άλλες σαν και αυτή. Από εκεί, οι συναλλαγές αποθηκεύονται στα νέα blocks, τα οποία με τη σειρά τους κοινοποιούνται στους χρήστες του δικτύου, για έλεγχο και αποδοχή. Ανατρέχοντας, λοιπόν, στο blockchain, οι κόμβοι μπορούν να διαπιστώσουν αν ένας χρήστης διαθέτει τα απαραίτητα χρήματα για μία δοσοληψία.

Όσον αφορά την ασφάλεια στις συναλλαγές, αυτή επιτυγχάνεται με τη δημιουργία μοναδικού αναγνωριστικού για κάθε block, που προκύπτει από τον κατακερματισμό των δεδομένων που περιέχει το block. Αυτό σημαίνει, ότι η παραμικρή αλλαγή σε οποιοδήποτε στοιχείο, οποιασδήποτε αποθηκευμένης συναλλαγής θα αλλάξει εντελώς την «υπογραφή» ολόκληρου του block. Ο αλγόριθμος κατακερματισμού, όμως, έχει σαν είσοδο και το αναγνωριστικό του προηγούμενου χρονικά block. Εν ολίγοις, τα αναγνωριστικά των blocks είναι συνδεδεμένα και αλληλοεξαρτώμενα, με αποτέλεσμα οι αλλαγές σε δεδομένα να αποτυπώνονται και σε όλα τα επόμενα blocks του blockchain.

Παρότι το blockchain είναι δημόσιο, τα στοιχεία των χρηστών του δικτύου, παραμένουν ιδιωτικά ως έναν βαθμό [9]. Οι χρήστες του δικτύου έχουν στην κατοχή δύο κρυπτογραφημένα κλειδιά, ένα ιδιωτικό και ένα δημόσιο, με τα οποία αποδεικνύουν την κυριότητα των συναλλαγών τους. Συγκεκριμένα, το δημόσιο κλειδί ενός χρήστη, αποτελεί μία σύντομη έκδοση του ιδιωτικού κλειδιού, κατασκευασμένο από μία σειρά πολύπλοκων αριθμητικών διαδικασιών, σχεδόν αδύνατον αν αναστραφούν. Για το λόγο αυτό η τεχνολογία blockchain θεωρείται εμπιστευτική.

2.3.2 Το κρυπτονόμισμα Bitcoin

Η σύλληψη της ιδέας του Blockchain χρονολογείται στις αρχές της δεκαετίας του 1990, όταν οι Stuart Haber και W. Scott Stornetta δημιούργησαν ένα σύστημα κρυπτογραφικής αλυσίδας blocks, όπου οι χρονικές σφραγίδες των εγγράφων είναι απαραβίαστες [10]. Ωστόσο, η χρησιμότητα αυτής της τεχνολογίας έγινε εμφανής το 2008, χάρη στην εργασία του (ή των) Satoshi Nakamoto και τη δημιουργία του Bitcoin, του πρώτου κρυπτονομίσματος.

Το Bitcoin αποτελεί την πρώτη χρονικά εφαρμογή της blockchain τεχνολογίας. Πρόκειται για ένα πλήρως ψηφιακό νόμισμα, που δεν ελέγχεται από καμία κεντρική τράπεζα ή κυβέρνηση και δεν βασίζεται στην εμπιστοσύνη των συναλλασσόμενων σε τρίτους, παρά μόνο στην τεχνολογία.

Όταν μία συναλλαγή κοινοποιείται στο δίκτυο του Bitcoin, οι κόμβοι του δικτύου την ελέγχουν ως προς την εγκυρότητά της [11]. Οι επιβεβαιωμένες συναλλαγές καταχωρούνται στα blocks του blockchain, απ' όπου και είναι σχεδόν αδύνατον να αλλαχθούν. Η δημιουργία των block πραγματοποιείται από τους υπολογιστές του δικτύου του Bitcoin, οι οποίοι ανταγωνίζονται ο ένας τον άλλον στην λύση ενός πολύπλοκου αλγοριθμικού προβλήματος. Η διαδικασία αυτή ονομάζεται mining (εξόρυξη) [12] και ο κόμβος που φτάνει πρώτος στη λύση δημοσιεύει το block στο δίκτυο και ανταμείβεται με κάποια Bitcoin για την δουλειά του. Περισσότερα για τη συγκεκριμένη λειτουργία θα ειπωθούν στην παράγραφο 2.3.3 .

Αυτή η ανταμοιβή, που λαμβάνουν οι miners, αποτελεί και τον τρόπο δημιουργίας νέων Bitcoins. Ωστόσο, χάρη στο halving process δεν παραμένει για πάντα η ίδια [13]. Συγκεκριμένα, οι ανταμοιβές μειώνονται στο μισό κάθε 4 χρόνια, με σκοπό τη σταδιακή μείωση της προσφοράς του κρυπτονομίσματος και την αποφυγή πληθωρισμού. Κατά συνέπεια, ο συνολικός αριθμός Bitcoin που μπορούν να παραχθούν περιορίζεται στα 21 εκατομμύρια, με το mining reward να μηδενίζεται το 2140.

Το Bitcoin εκμεταλλεύεται τα πλεονεκτήματα που προσφέρει το blockchain και που αναφέραμε προηγουμένως. Η διαφάνεια στις συναλλαγές είναι δεδομένη, αφού αυτές καταγράφονται με δημόσιο τρόπο στο blockchain. Όσον αφορά την ανωνυμία των χρηστών, επιτυγχάνεται σε μεγάλο βαθμό, μέσω της χρήσης ενός προγράμματος, ονόματι «wallet» (πορτοφόλι). Το ηλεκτρονικό αυτό πορτοφόλι είναι απαραίτητο στους χρήστες, καθώς τους παρέχει τα δύο κρυπτογραφικά κλειδιά (δημόσιο και ιδιωτικό) για την πραγματοποίηση συναλλαγών. Συγκεκριμένα, το δημόσιο κλειδί αντιπροσωπεύει τη διεύθυνση του χρήστη για καταθέσεις και αναλήψεις Bitcoin. Ωστόσο, μόνο μέσω του ιδιωτικού του κλειδιού [14] , μπορεί ένας χρήστης να επιβεβαιώσει τις συναλλαγές από και προς το δημόσιο κλειδί του. Όπως προείπαμε το δημόσιο κλειδί παράγεται από το ιδιωτικό, ωστόσο η διαδικασία είναι μη αναστρέψιμη, δεν μπορεί δηλαδή κάποιος να παράγει ιδιωτικά κλειδιά από δημόσια.

2.3.3 Το πρωτόκολλο Proof-Of-Work

Σύμφωνα με όσα αναφέραμε, γίνεται αντιληπτή ή σπουδαιότητα των miners, αφού αυτοί είναι που ελέγχουν τις συναλλαγές και δημιουργούν τα blocks που της οριστικοποιούν. Είναι απαραίτητος, λοιπόν, ένας μηχανισμός που να επιβραβεύει τους συμμετέχοντες, που συμβάλλουν στην επίτευξη των στόχων του δικτύου, κάνοντας ταυτοχρόνως δαπανηρή οποιαδήποτε επίθεση σε αυτό. Για τη διευθέτηση του ζητήματος εμπιστοσύνης, το Bitcoin δημιούργησε το πρωτόκολλο Proof of Work [15], σύμφωνα με το οποίο οι κόμβοι πρέπει να αποδείξουν ότι έχουν δουλέψει, λύνοντας ένα περίπλοκο αλγοριθμικό πρόβλημα. Ο miner θα λύσει πρώτος το πρόβλημα, θα είναι αυτός που θα πάρει το δικαίωμα δημιουργίας τους block, καθώς και την αντίστοιχη αμοιβή.

Ουσιαστικά, πρόκειται για ένα κρυπτογραφικό πάζλ, με την ακόλουθη διαδικασία: Οι miners αναζητούν έναν αριθμό/nonce και τον συνενώνουν με τις συναλλαγές του block προς δημοσίευση, ώστε να παραχθεί ένα λεκτικό. Το δημιουργηθέν λεκτικό δίνεται ως είσοδος σε έναν αλγόριθμο κατακερματισμού (SHA256), ο οποίος δίνει έξοδο μία συμβολοσειρά 64 χαρακτήρων. Η συμβολοσειρά αυτή θα πρέπει να ξεκινά με έναν συγκεκριμένο αριθμό μηδενικών, ώστε το πρόβλημα να θεωρηθεί λυμένο. Ο αριθμός των μηδενικών, αποτελεί και τη δυσκολία του προβλήματος, αφού όσο μεγαλύτερος είναι τόση πιο μικρή η πιθανότητα εύρεσης μίας τέτοιας συμβολοσειράς.

Η διαδικασία είναι τέτοια, ώστε η λύση του να είναι δύσκολο να βρεθεί, αλλά ταυτόχρονα εύκολο να επιβεβαιωθεί. Η λύση του προβλήματος διαφέρει πάντα, αφού εξαρτάται από τα δεδομένα του εκάστοτε block, ενώ για την εύρεσή της απαιτείται εξαντλητική αναζήτηση. Η υπολογιστική ισχύς που απαιτείται για την εύρεση του αριθμού είναι τέτοια, που προϋποθέτει τη χρήση ειδικού hardware και την μεγάλη κατανάλωση ενέργειας. Με τον τρόπο αυτό, το Proof-of-Work καθιστά ακριβή τη συμμετοχή στη διαδικασία εξόρυξης για τους miners, με την υπόσχεση της ανταμοιβής, αν λειτουργούν προς όφελος του δικτύου.

2.3.4 Το πρωτόκολλο Proof-Of-Stake

Ένα άλλο πρωτόκολλο για τη δημιουργία blocks, που χρησιμοποιείται σε κρυπτονομίσματα είναι το Proof-of-Stake [16]. Σύμφωνα με αυτό, οι εν δυνάμει δημιουργοί των blocks δεν ανταγωνίζονται για τη λύση ενός πάζλ, αντιθέτως εκλέγονται. Η εκλογή γίνεται από έναν αλγόριθμο τυχαίας επιλογής με βάρη, όπου το βάρος του κόμβου ισοδυναμεί με τον αριθμό κρυπτονομισμάτων που διαθέτει ή «ποντάρει». Έτσι αν ο χρήστης Α διαθέτει 10 φορές παραπάνω κρυπτονομίσματα από το χρήστη Β, θα έχει και 10 φορές μεγαλύτερη πιθανότητα εκλογής σε σχέση με αυτόν.

Ο εκλεγμένος κόμβος ελέγχει τις συναλλαγές και τις τοποθετεί στο block του, το οποίο ύστερα δημοσιεύει. Στο πρωτόκολλα Proof of Stake δεν αναφερόμαστε σε miners, αλλά σε stakeholders, που λαμβάνουν ένα μικρό φόρο από τις συναλλαγές που συμπεριλαμβάνουν στα blocks τους και όχι κάποια συγκεκριμένη ανταμοιβή όπως προηγουμένως. Μετά τη δημιουργία ενός block, τα χρήματα του stakeholder παγώνουν για μία χρονική περίοδο. Αυτό συμβαίνει για λόγους ασφαλείας, ώστε αν κατά την περίοδο αυτή διαπιστωθούν δόλιες συναλλαγές στο block του validator, μέρος των χρημάτων του να χαθεί.

Η λογική πίσω από το Proof of Stake, είναι παρόμοια με του Proof Of Work, δηλαδή η δημιουργία θετικών κινήτρων στους κατασκευαστές των blocks. Συγκεκριμένα, βασίζεται στην πεποίθηση, ότι ένας κάτοχος πολλών μονάδων ενός κρυπτονομίσματος δεν θα λειτουργούσε ενάντια στο δίκτυο, θέτοντας σε κίνδυνο τα χρήματά του και την τιμή του ίδιου του νομίσματος.

2.4 Προβλήματα στην Blockchain τεχνολογία

Η τεχνολογία του Blockchain περιλαμβάνει διάφορα προβλήματα, τα οποία προκύπτουν τόσο από τη φύση των κατανεμημένων δικτύων υπολογιστών, όσο και από τη λειτουργία των διάφορων πρωτοκόλλων που χρησιμοποιούνται στα κρυπτονομίσματα. Σε αυτήν την παράγραφο, παρουσιάζουμε κάποια από τα βασικά προβλήματα στο χώρο του Blockchain και τις λύσεις που έχουν προταθεί, με σκοπό να γίνει κατανοητή, αργότερα, η συμβολή και η καινοτομία του Casper στην επίλυσή τους.

2.4.1 Byzantine Generals Problem ή Consensus Problem

Η κλασσική διατύπωση του προβλήματος Byzantine Generals Problem έγινε από τον L. Lamport το 1982 [17] και αφορά την διαχείριση αντικρουόμενων μηνυμάτων σε ένα δίκτυο υπολογιστών, με σκοπό την λήψη ορθών αποφάσεων. Πρόκειται, ουσιαστικά, για μία παρομοίωση του δικτύου με μία ομάδα βυζαντινών στρατηγών, (εξού και η ονομασία Byzantine Generals Problem) οι οποίοι επικοινωνούν μεταξύ τους με μηνύματα για τη λήψη μιας κοινής απόφασης, σχετικά με την επίθεσή τους σε μία εχθρική πόλη. Το πρόβλημα εστιάζεται στην ύπαρξη προδοτών μεταξύ των στρατηγών, οι οποίοι στέλνουν δόλια μηνύματα, με σκοπό να εμποδίσουν τους πιστούς στρατηγούς από το να λάβουν μία απόφαση.

Το ίδιο πρόβλημα εμφανίζεται και στα κρυπτονομίσματα [18]. Συγκεκριμένα, κατά την υποβολή μίας συναλλαγής στο δίκτυο, θα πρέπει το τελευταίο να μπορεί επιβεβαιώσει, ότι ο αποστολέας δεν έχει ήδη ξοδέψει τα αντίστοιχα κρυπτονομίσματα σε παλαιότερες συναλλαγές. Όπως και στο Byzantine Generals Problem, έτσι και εδώ δεν υπάρχει κάποια κεντρική αρχή που να μπορεί να επιβεβαιώνει συναλλαγές (ή μηνύματα, στην περίπτωση του BGP). Για το λόγο αυτό, θα πρέπει να λυθεί αποκεντρωμένα, χωρίς δηλαδή κάποιο έμπιστο τρίτο πρόσωπο.

Τη λύση σε αυτό το πρόβλημα αποτελεί η δημόσια ανακοίνωση των συναλλαγών στο δίκτυο και η συμφωνία πως η πρώτη συναλλαγή που καταφτάνει είναι η έγκυρη. Ωστόσο, εδώ προκύπτει το ζήτημα της συμφωνίας, για το ποια συναλλαγή κατέφθασε πρώτη, πράγμα που μπορεί να διαφέρει από κόμβο σε κόμβο, λόγω καθυστερήσεων του δικτύου. Για την επίτευξη της συναίνεσης στη χρονική σειρά των συναλλαγών, τοποθετείται χρονική σφραγίδα σε αυτές, κατά την εισαγωγή τους σε blocks.

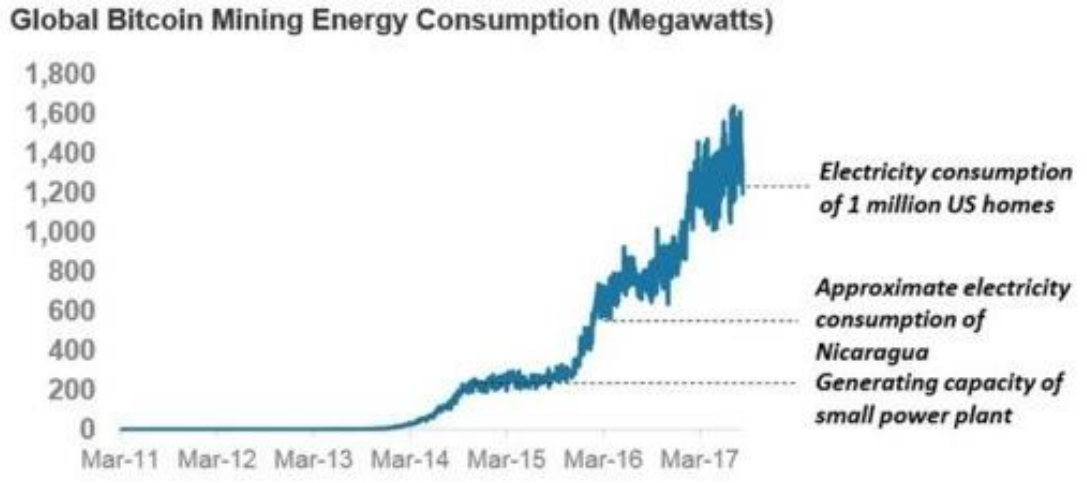
Το Bitcoin εφάρμοσε πρώτο αυτή τη διαδικασία χρονοσήμανσης δοσοληψιών και κατακερματισμού των δεδομένων των blocks, με το πρωτόκολλο Proof of Work που αναλύσαμε στην προηγούμενη παράγραφο. Όπως αναφέραμε, ένα block δεν μπορεί να αλλαχθεί, χωρίς την επανάληψη της δουλειάς που το παρήγαγε, που εν προκειμένω είναι το σύνολο της υπολογιστικής ισχύος, που χρησιμοποιήθηκε για τη λύση του κρυπτογραφικού πάζλ του block. Επειδή, μάλιστα, τα blocks είναι «αλυσοδεμένα» σειριακά, η επαναδημιουργία ενός block απαιτεί την επαναδημιουργία και όλων των blocks που ακολουθούν.

Τέλος, πρόβλημα της συναίνεσης προκύπτει και μεταξύ των υποαλυσίδων του Blockchain. Ένα block είναι πιθανό να αποκτήσει δύο παιδιά, δηλαδή το Blockchain ναι χωριστεί σε δύο ή περισσότερα παρακλάδια. Το consensus (αλγόριθμος συναίνεσης) του Bitcoin θεωρεί έγκυρη την αλυσίδα με τη μεγαλύτερη δυσκολία, όπου λόγω του Proof of Work θα είναι αυτή με το μεγαλύτερο μήκος. Η λειτουργία αυτή μπορεί να παρομοιαστεί με μια ψηφοφορία, όπου ψήφος θεωρείται το mining ενός κόμβου σε μία από τις αλυσίδες. Έτσι, η αλυσίδα με τις περισσότερες ψήφους, είναι αυτή με τους περισσότερους miners (μεγαλύτερο hash rate) και επομένως την μεγαλύτερη πιθανότητα να αναπτυχθεί γρηγορότερα.

2.4.2 Proof of Work - Energy Problem

Είδαμε, ότι το πρωτόκολλο Proof-of-Work και η διαδικασία της εξόρυξης είναι απαραίτητα στοιχεία στο Bitcoin, τόσο για την ασφάλεια των συναλλαγών, όσο για τη συναίνεση τον κόμβων στην κανονική αλυσίδα του blockchain. Τα προνόμια αυτά, όμως έρχονται με ένα κόστος πολύ υψηλό, σε χρήματα και σε ενέργεια. Σήμερα στο χώρο του mining κυριαρχούν οι μεγάλες φάρμες εξόρυξης (mining farms) με χιλιάδες υπολογιστές να λειτουργούν ανελλιπώς, με σκοπό την λύση του κρυπτογραφικού γρίφου και την είσπραξη της ανταμοιβής. Οι μεγάλες ποσότητες ενέργειας που καταναλώνονται καθημερινά για τη δημιουργία block, σε συνδυασμό με το ειδικό hardware που απαιτείται για την εξόρυξη (GPUs, ψυκτικά συστήματα), ξεπερνούν σε κόστος τα 3,2 δισεκατομμύρια ετησίως, μόνο όσον αφορά το δίκτυο του Bitcoin [19]. Μάλιστα, η δαπανώμενη ηλεκτρική ενέργεια για τη λειτουργία του Bitcoin ισοδυναμεί με αυτήν της Τσεχίας, ενώ υπολογίζεται πως 17 νοικοκυριά περίπου, θα μπορούσαν να λειτουργήσουν για μία μέρα, με την ενέργεια που καταναλώνεται για μία μόνο συναλλαγή.

Για το λόγο αυτό, δημιουργείται η ανάγκη δημιουργίας πρωτοκόλλων, που θα παρέχουν τα αντίστοιχα προνόμια με το Proof-of-Work, περιορίζοντας ταυτόχρονα την κατανάλωση ενέργειας που απαιτείται. Μία λύση προς αυτήν την κατεύθυνση είναι και το Proof-of-Stake, που δεν απαιτεί την λύση κάποιου πολύπλοκου πάζλ. Το Proof-of-Work απαιτεί οι miners να καταναλώνουν υπολογιστική δύναμη για να προφυλάσσουν το δίκτυο, αντίθετα το Proof-of-Stake προσομοιάζει την κατανάλωση αυτή, ώστε να μην δαπανούνται πραγματικά χρήματα και ενέργεια.



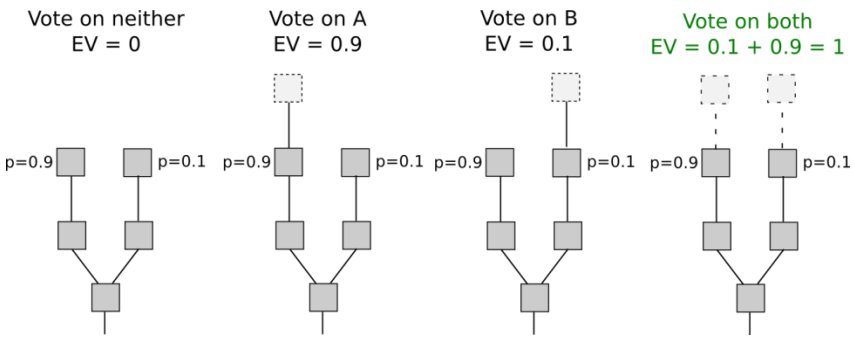
Εικόνα 2. 4 Γράφημα κατανάλωσης ενέργειας λόγω Bitcoin mining.

2.4.3 Proof of Stake – Nothing at Stake Problem

Όπως αναφέραμε, το Proof of Stake δεν απαιτεί κατανάλωση υπολογιστικής ισχύος για την κατασκευή των blocks, αλλά τιμωρεί τους παραβάτες validators, με την παρακράτηση των κρυπτονομισμάτων τους (stake). Ωστόσο, αν στο blockchain προκύψουν διακλαδώσεις (branches), τίποτα δεν εμποδίζει ένα χρήστη να υπογράψει σε όλες. Και αυτό, γιατί εδώ δεν του κοστίζει τίποτα να διεκδικήσει την δημιουργία ενός block και την είσπραξη των φόρων, ανεξαρτήτως της αλυσίδας στην οποία βρίσκεται. Αντίθετα, στα πρωτόκολλα Proof of Work το mining σε μη κανονική αλυσίδα, ισοδυναμεί με χαμένη a priori επένδυση τόσο σε hardware, όσο και σε ηλεκτρική ενέργεια. Εδώ είναι που εντοπίζεται και το πρόβλημα Nothing-at-Stake.

Ας υποθέσουμε την ύπαρξη δύο αλυσίδων σε ένα blockchain: την κανονική αλυσίδα και μία κατασκευασμένη από ένα κακόβουλο χρήστη. Αν το πρωτόκολλο είναι PoW, οι χρήστες που κάνουν mining στην κακόβουλη αλυσίδα (που προφανώς είναι λιγότερο δημοφιλής από την κανονική), θα καταλήξουν να χάσουν τα χρήματά τους, αφού η γρηγορότερα αναπτυσσόμενη αλυσίδα θα καταλήξει να είναι και η έγκυρη. Αντίθετα, στην περίπτωση του PoS οι βέλτιστη στρατηγική για τους χρήστες είναι να υπογράφουν όλα τα blocks και στις δύο αλυσίδες, κάνοντάς τες να μοιάζουν ισοδύναμες. Επιπροσθέτως, αν ο κακόβουλος χρήστης τοποθετήσει τα χρήματά του (όσα και αν είναι αυτά) μόνο στην κατασκευασμένη από αυτόν αλυσίδα, τότε αυτή είναι, που κατά μεγαλύτερη πιθανότητα θα προχωρήσει ταχύτερα και θα καταλήξει να αποτελεί την κανονική αλυσίδα συναλλαγών.

Το πρόβλημα αυτό του PoS δημιουργείται για δύο λόγους. Ο πρώτος είναι τα λανθασμένα κίνητρα, αφού η στρατηγική μεγαλύτερου κέρδους δεν είναι και η πιο ωφέλιμη για το δίκτυο. Συγκεκριμένα η έλλειψη μίας τιμωρίας ή δαπάνης, επιτρέπει στους κόμβους να χρησιμοποιούν το stake τους προς όφελος τους, χωρίς να φοβούνται για τις επιπτώσεις. Ο δεύτερος λόγος εστιάζεται στο κριτήριο της κύριας αλυσίδας, αφού προφανώς ο κανόνας της μεγαλύτερης σε μήκος αλυσίδας του PoW δεν μπορεί να εφαρμοστεί αυτούσια στο PoS, το consensus πρέπει να είναι τέτοιο ώστε να εξασφαλίζει την ασφάλεια στο δίκτυο, επιλέγοντας την πιο «δύσκολη» αλυσίδα. Στην παρούσα διπλωματική θα εξετάσουμε αυτά τα ζητήματα και θα επιχειρήσουμε να δώσουμε μία λύση μέσω της υλοποίησης του Casper.



Εικόνα 2. 5 Η βέλτιστη στρατηγική της υπογραφής όλων των branches.

2.5 Casper – Μηχανισμός συναινετικής οριστικοποίησης της κύριας αλυσίδας

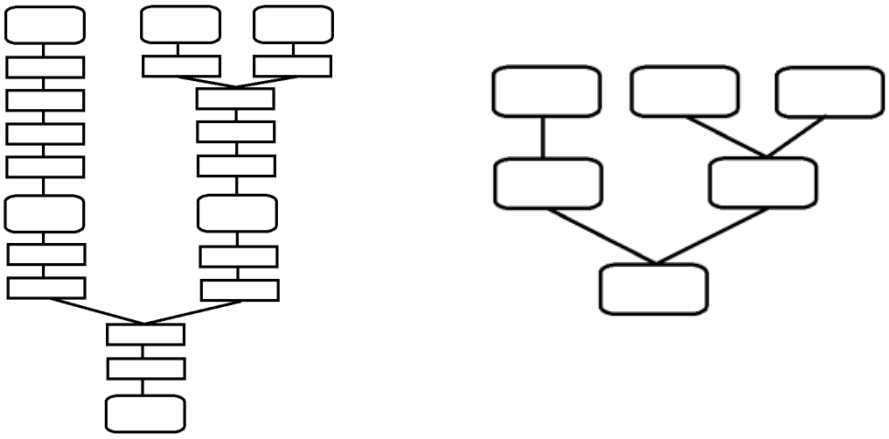
Από τα όσα λέχθηκαν, γίνεται εμφανής η ανάγκη για ένα μηχανισμός, που να επιτυγχάνει την συναίνεση των χρηστών στο blockchain, να εξασφαλίζει την ασφάλεια των συναλλαγών και να μην απαιτεί τη δαπάνη μεγάλων ποσών ενέργειας. Σε αυτήν την κατεύθυνση εργάστηκαν οι Vitalik Buterin και Virgil Griffith, δημιουργώντας το Casper, έναν μηχανισμό μερικής συναίνεσης που συνδυάζει την Proof-of-Stake λογική με τη θεωρία συναίνεσης (consensus theory). Συγκεκριμένα πρόκειται για ένα σύστημα, υπεύθυνο για την οριστικοποίηση των blocks που δημιουργούνται, επιλέγοντας ουσιαστικά μία μοναδική, κανονική αλυσίδα. Το Casper μπορεί να εφαρμοστεί σε οποιοδήποτε υπάρχον blockchain δίκτυο χωρίς να επηρεάσει τις υπόλοιπες βασικές του λειτουργίες, που έχουμε αναφέρει έως τώρα.

Όσον αφορά το Proof-of-Stake κομμάτι, το Casper προσομοιώνει τη λειτουργία των Proof-of-Work πρωτοκόλλων, χρησιμοποιώντας το κρυπτονόμισμα ως διακύβευμα, αντί της ηλεκτρικής ισχύος. Το δεύτερο μέρος, βασίζεται στη θεωρία της συναίνεσης τύπου Byzantine Fault Tolerance (BFT), σύμφωνα με την οποία δεν είναι δυνατή η οριστικοποίηση αντικρουόμενων αλυσίδων, εφόσον $> \frac{2}{3}$ των συμμετεχόντων ακολουθούν πιστά το πρωτόκολλο.

2.5.1 Η λειτουργία του Casper

Η λειτουργία του Casper, βασίζεται σε μία ομάδα χρηστών, τους validators [21]. Οι validators, στην ύπαρξη πολλών αλυσίδων, ψηφίζουν checkpoints, με σκοπό την οριστικοποίηση μίας κανονικής αλυσίδας. Τα checkpoints είναι κανονικά blocks, που βρίσκονται σε συγκεκριμένα ύψη στο blockchain tree. Τα ύψη αυτά είναι τα πολλαπλάσια ενός αριθμού epoch_size, που ορίζεται από το εκάστοτε δίκτυο. Αν για παράδειγμα epoch_size = 100 , τα checkpoint blocks θα είναι στα ύψη 0, 100, 200, 300, 400 κ.ο.κ. Σε κάθε περίπτωση το Genesis Block [22] αποτελεί το πρώτο checkpoint του blockchain.

Για να γίνει ένας κόμβος validator, πρέπει να κάνει μία κατάθεση που να ξεπερνά ένα προκαθορισμένο αριθμό κρυπτονομισμάτων. Όταν γίνει validator , η αρχική του κατάθεση αυξομειώνεται με ανταμοιβές και κυρώσεις, ανάλογα με τη συμπεριφορά του. Η δύναμη της ψήφου ενός validator, απορρέει από το μέγεθος της κατάθεσης του, σε σύγκριση με αυτές των υπολοίπων. Συνεπώς, όταν μιλάμε για τα $\frac{2}{3}$ των validators, αναφερόμαστε σε ποσοστό καταθέσεων και όχι χρηστών.



(α) Δέντρο blockchain με epoch size = 5 (β) Το δέντρο checkpoint του διπλανού blockchain

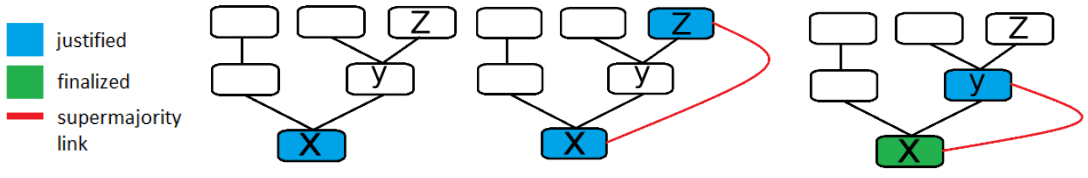
Εικόνα 2. 6 Σχεδιασμός checkpoint tree, από το blockchain tree.

Η ψήφος, που αποστέλλουν οι validators, αποτελείται από τέσσερις πληροφορίες: δύο checkpoints s και t (πηγή και δέκτη), μαζί με τα ύψη τους h(s) και h(t) αντίστοιχα. Για να θεωρηθεί η ψήφος έγκυρη, θα πρέπει το t να είναι «απόγωνος» του s στο δέντρο της blockchain. Στην ψήφο συμπεριλαμβάνεται και η υπογραφή του αποστολέα, η οποία προκύπτει από το ιδιωτικό του κλειδί και ταυτοποιεί, ότι ο τελευταίος είναι validator.

Αν τουλάχιστον τα $\frac{2}{3}$ των validators δημοσιεύσουν την ίδια ψήφο με πηγή x και δέκτη y, τότε μιλάμε για ψήφο «υπερπλειοψηφίας», supermajority link και συμβολίζεται με $x \rightarrow y$.

Ένα checkpoint y θα λέγεται justified (δικαιολογημένο), αν είναι το Genesis Block ή υπάρχει supermajority link $x \rightarrow y$, όπου το x είναι justified checkpoint.

Ένα checkpoint x θα λέγεται finalized (οριστικοποιημένο), αν είναι το Genesis Block ή υπάρχει supermajority link $x \rightarrow y$, όπου το x είναι justified checkpoint και το y είναι το αμέσως επόμενο checkpoint μετά το x (δηλαδή το παιδί του x στο checkpoint tree).



(α) x justified (b) $x \rightarrow z$, z justified (c) $x \rightarrow y$, y justified και x finalized

Εικόνα 2. 7 Παραδείγματα justification και finalization στο checkpoint tree.

Η παραπάνω λογική γίνεται κατανοητή με την παρουσίαση των παρακάτω κανόνων για τους validators:

Ένας validator απαγορεύεται να αποστείλει δύο ξεχωριστές ψήφους,
$\{ \nu, s_1, t_1, h(s_1), h(t_1) \}$ και $\{ \nu, s_2, t_2, h(s_2), h(t_2) \}$

Για τις οποίες ισχύουν τα παρακάτω:

$h(t_1) = h(t_2)$.
Δηλαδή, ένας validator απαγορεύεται να αποστείλει διαφορετικές ψήφους με targets στο ίδιο ύψος.

$h(s_2) < h(s_1) < h(t_1) < h(t_2)$.
Δηλαδή, ένας validator απαγορεύεται να αποστείλει ψήφο, όπου το διάστημα των υψών περιέχεται σε (ή περιέχει) διάστημα υψών προηγούμενης ψήφου .

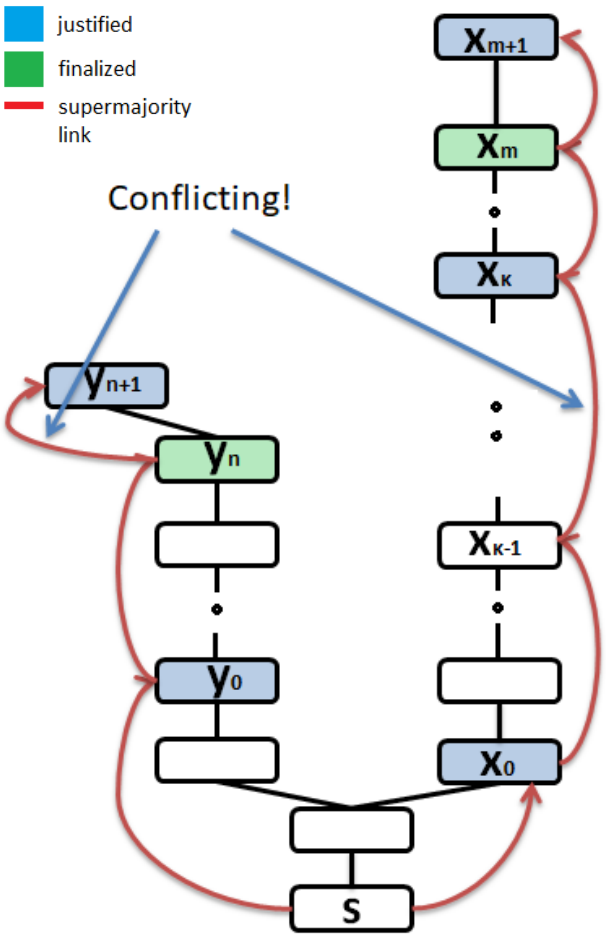
Στην περίπτωση που κάποιος validator παραβεί κάποιον από τους παραπάνω κανόνες, όλα τα απαραίτητα στοιχεία για τον εντοπισμό του περιέχονται στο blockchain, οπότε χάνει ολόκληρη την κατάθεσή του και βγαίνει από το set των validators (slashing). Εφόσον, τουλάχιστον τα $\frac{2}{3}$ των validators ακολουθούν αυτούς τους κανόνες είναι αδύνατη η οριστικοποίηση δύο blocks σε αντικρουόμενες αλυσίδες.

Για τη μαθηματική απόδειξη της παραπάνω πρότασης θα εργαστούμε στο checkpoint tree: Έστω δύο finalized checkpoints x_m και y_n σε διαφορετικές αλυσίδες. Αυτό προϋποθέτει την ύπαρξη δύο διαφορετικών «αλυσίδων» από supermajority links με κάποια κοινή αρχή (είτε αυτή είναι το Genesis Block, είτε όχι). Δηλαδή, θα υπάρχουν τα παρακάτω links από checkpoints:

$s \rightarrow y_0 \rightarrow y_1 \rightarrow \ldots \rightarrow y_n \rightarrow y_{n+1}$ και $s \rightarrow x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x_m \rightarrow x_{m+1}$

Όπου y_{n+1} , x_{m+1} τα παιδιά των y_n και x_m , αντίστοιχα, στο checkpoint tree, αφού θεωρήσαμε τα y_n και x_m finalized (βλ. κανόνα finalization). Τα ύψη όλων των x και y στα παραπάνω supermajority links πρέπει να είναι διαφορετικά μεταξύ τους. Αν υπάρχουν x_j και y_i τέτοια ώστε $h(x_j) = h(y_i)$, η πρώτη συνθήκη του κανόνα που θέσαμε παραβιάζεται, αφού τουλάχιστον τα $\frac{2}{3}$ των validators απέστειλαν ψήφο με target το x_j και τουλάχιστον τα $\frac{2}{3}$ των validators απέστειλαν ψήφο με target το y_i στο ίδιο ύψος. Άρα τουλάχιστον το $\frac{1}{3}$ των validators παραβίασε τον κανόνα 1. Γνωρίζουμε επίσης ότι $h(x_m) + 1 = h(x_{m+1})$ και $h(y_n) + 1 = h(y_{n+1})$ στο checkpoint tree. Χωρίς βλάβη της γενικότητας, θεωρούμε ότι $h(x_m) > h(y_n)$ και επομένως $h(x_m) > h(y_{n+1})$, αφού όπως είπαμε $h(x_j) \neq h(y_i)$. Υποθέτουμε κ το μικρότερο ακέραιο τέτοιο ώστε $h(x_\kappa) > h(y_{n+1})$. Αυτό φανερώνει την ύπαρξη $x_{\kappa-1}$, όπου $x_{\kappa-1} \rightarrow x_\kappa$ με $h(x_\kappa) < h(y_n)$. Δηλαδή τουλάχιστον το $\frac{1}{3}$ των validators απέστειλε ψήφους $x_{\kappa-1} \rightarrow x_\kappa$ και $y_n \rightarrow y_{n+1}$ με $h(x_{\kappa-1}) < h(y_n) < h(y_{n+1}) < h(x_\kappa)$, παραβιάζοντας το κανόνα 2.

Η μαθηματική απόδειξη γίνεται κατανοητή στην εικόνα 2.8



Εικόνα 2. 8 Απόδειξη ασφάλειας Casper στο checkpoint tree.

Τέλος, ο κανόνας επιλογής αλυσίδας ή αλλιώς «Fork Choice Rule» [23] από τους χρήστες δεν είναι αυτός της μεγαλύτερης αλυσίδας, όπως στα PoW πρωτόκολλα. Εδώ οι χρήστες πάντα ακολουθούν την αλυσίδα, στην οποία βρίσκεται το μεγαλύτερο σε ύψος justified checkpoint. Ο παραπάνω κανόνας είναι correct-by-construction και όλοι οι χρήστες οφείλουν να τον ακολουθούν, για την αποφυγή παθολογικών σεναρίων, όπου η μεγαλύτερη σε μήκος αλυσίδα δεν μπορεί να γίνει finalized χωρίς την εσκεμμένη διαγραφή κάποιων validators.

2.5.2 Τα δυναμικά validator sets

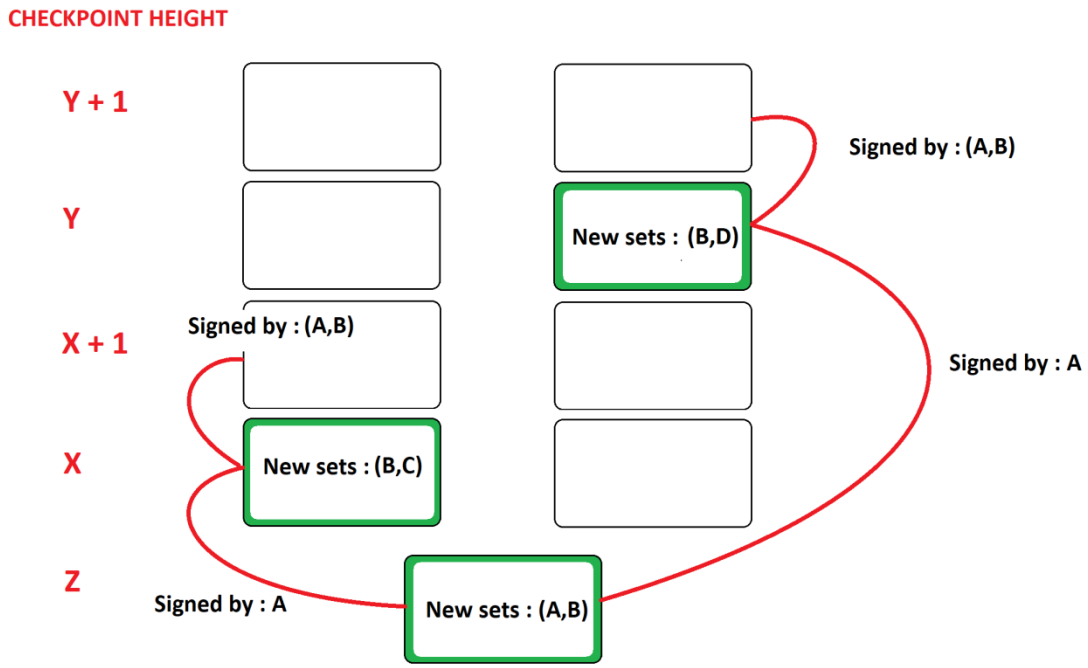
Σύμφωνα με τα όσα αναφέρθηκαν για το Casper, αντιλαμβανόμαστε πως η ασφάλεια του blockchain επιτυγχάνεται από τους περιορισμούς που έχουν τεθεί στις ψήφους των validators. Τεκμήριο για την επιβολή μιας τιμωρίας, δηλαδή του slashing, αποτελεί το ιστορικό των ψήφων του validator. Ωστόσο, το set των validator θα πρέπει να μπορεί να αλλάζει, χωρίς όμως αυτό να δημιουργεί ασυνέπειες στον τρόπο λειτουργίας του πρωτοκόλλου και της επιβολής των ποινών.

Υποθέτουμε, ένα δίκτυο blockchain με Casper, στο οποίο οι κόμβοι μπορούν να εισέλθουν στο validator set, αποστέλλοντας ένα μήνυμα κατάθεσης deposit και οι υπάρχοντες validators μπορούν να εξέλθουν από αυτό, με ένα μήνυμα «ανάληψης» «withdraw». Οι αλλαγές στο validator set πραγματοποιούνται με το finalization κάποιου checkpoint. Κάθε checkpoint περιέχει το ήδη υπάρχον validator set καθώς και εκείνο που θα αποτελέσει το καινούργιο validator set, σε περίπτωση οριστικοποίησής του. Στο σύστημα αυτό μπορούμε εύκολα να δείξουμε, πως τα υπάρχοντα κριτήρια δεν λειτουργούν αποτελεσματικά, όπως για ένα αμετάβλητο validator set.

Αν για παράδειγμα, οι validators κάνουν finalize ένα checkpoint σε ύψος λ, δεν υπάρχει κανένα μέτρο που να εμποδίζει μελλοντικούς validators με «καθαρό ιστορικό», να κάνουν επίσης finalize ένα αντικρουόμενο checkpoint στο ίδιο ύψος λ. Σε αυτό το πρόβλημα, μία πιθανή λύση είναι το finalization ενός block να συμβαίνει αμέσως μετά τη δημιουργία του, δηλαδή να το consensus model να έχει τη μορφή «create block 1, finalize block 1, create block 2, finalize block 2 …». Ωστόσο, ένα τέτοιο σύστημα θα προκαλούσε καθυστερήσεις στο δίκτυο, αφού θα έπρεπε το consensus να συμβαίνει σε κάθε block ή checkpoint, για τη δημιουργία νέων blocks.

Ουσιαστικά, χρειαζόμαστε ένα σύστημα «συνυφασμένης συναίνεσης» [24], όπου η δημιουργία και το finalization των blocks συμβαίνουν παράλληλα, χωρίς μάλιστα να είναι απαραίτητη η διαρκής επίτευξη της συναίνεσης για τη βιωσιμότητα του δικτύου. Από όσα αναφέρθηκαν, φαίνεται ότι το βασικό ζήτημα είναι η απώλεια ιστορικού ψήφων των νέων validators. Για το λόγο αυτό, θα χρησιμοποιούμε δύο validator set αντί του ενός, το front και το rear. Το front validator set, είναι το validator set που μπήκε και αντίστοιχα το rear εκείνο που έφυγε, δηλαδή το προηγούμενο front. Τα δύο set αυτά δεν είναι ξένα μεταξύ τους, αντίθετα πολλές φορές είναι σε μεγάλο ποσοστό ίδια, αφού πρακτικά διαφέρουν μόνο στα πρόσφατα deposits και withdraws. Για παράδειγμα αν έχουμε Front = {1,2,3,4,5} , Rear = X και ακολουθήσουν τα deposits των 6, 7 και τα withdraws των 2, 4, τα νέα sets που προκύπτουν είναι τα Front_New = {1,3,5,6,7} , Rear_New = {1,2,3,4,5} .

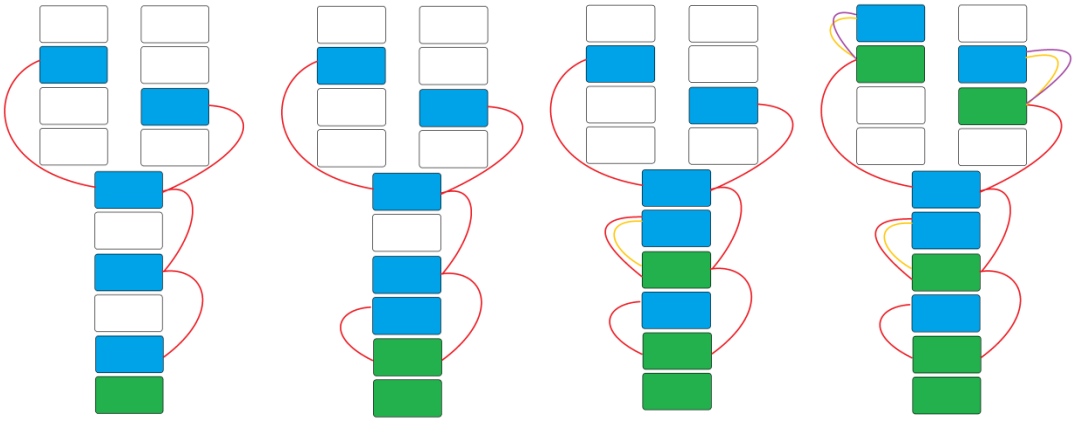
Τώρα, αλλάζουμε ελαφρώς τους υπάρχοντες κανόνες για justification και finalization και απαιτούμε τα $\frac{2}{3}$ των validators του front και τα $\frac{2}{3}$ των validators του rear να έχουν αποστείλει την ίδια ψήφο, για τη δημιουργία supermajority link. Με τον τρόπο αυτό το rear validator set κάθε checkpoint αποκτά ιστορικό, αφού θα έχει σίγουρα αποστείλει ψήφο $x \rightarrow x + 1$, ως πρώην front validator set, για την οριστικοποίηση του προηγούμενου finalized checkpoint. Με τον τρόπο αυτό το finalization ενός checkpoint σε ένα ύψος $y > x$ σε αντικρουόμενη αλυσίδα, θα απαιτούσε το link $z \rightarrow y$ με $z < x$. Σε αυτήν την περίπτωση το rear validator set θα γινόταν slashed αφού υπάγεται στον κανόνα 2. $z < x < x + 1 < y$.Το παραπάνω παράδειγμα παρουσιάζεται στην εικόνα 2.9



Αρχικό validator set A με ψήφους $z \rightarrow y$ (y justified) και $z \rightarrow x$ (z finalized – x justified) ($z = x - 1$ εδώ)
Νέο validator set (A,B) με ψήφους $x \rightarrow x + 1$ (x finalized) και $y \rightarrow y + 1$ (y finalized)

Εικόνα 2. 9 Παραβίαση κανόνα 2 από το validator set A

Ωστόσο τα justification και finalization των checkpoints, μπορούν να γίνουν με τέτοιο τρόπο που να επιτρέψουν σε μελλοντικούς validators να οριστικοποιήσουν conflicting blocks. Στην εικόνα 2.10 ακολουθεί παράδειγμα μίας τέτοιας περίπτωσης, όπου το σύστημα αποτυγχάνει.



a) Το Validator set A πραγματοποιεί τις ψήφους για justification (κόκκινο χρώμα)

b) Το A κάνει finalize το validator set B (κίτρινο χρώμα)

c) Τα A, B κάνουν finalize το validator set C (μωβ χρώμα)

d) Τα B,C κάνουν finalize δύο conflicting checkpoints

Εικόνα 2. 10 Οριστικοποίηση αντικρουόμενων checkpoints από δυναμικά validator sets

Το πρόβλημα στο παραπάνω παράδειγμα εστιάζεται στο γεγονός ότι το finalization ενός παλιού block μπορεί να συμβεί ανά πάσα στιγμή προκαλώντας αλλαγή στα validator sets. Εδώ σημειώνεται ότι αν ένα checkpoint οριστικοποιηθεί, τότε τα απαραίτητα στοιχεία που επικυρώνουν το finalization μπορούν να καταχωρηθούν στο blockchain, μέχρι τη δημιουργία του επόμενου checkpoint. Αντίθετα, αν ένα block δεν γίνει finalized, τότε σίγουρα δεν μπορούν να βρεθούν τα στοιχεία αυτά. Χρειαζόμαστε δηλαδή ένα σύστημα «ναι ή ίσως», όπου αν το blockchain δείξει «ίσως», τότε δεν προχωράμε σε οριστικοποίηση και αλλαγή validator set, ενώ αν πει «ναι», το block γίνεται finalized. Εν ολίγοις, κάθε checkpoint που γίνεται finalize, μπορεί να φτιάξει ένα παιδί που να φανερώνει το finalization του και αντίστοιχα για την αντίθετη περίπτωση.

Έτσι στο παραπάνω παράδειγμα το finalization των block από το validator set A, δεν θα προκαλέσει αλλαγή στα validator set, αφού όλα τα δημιουργηθέντα checkpoints θα βλέπουν μονίμως το A ως validator set.

Τελικά οι κανόνες για το justification και το finalization των blocks είναι οι εξής:

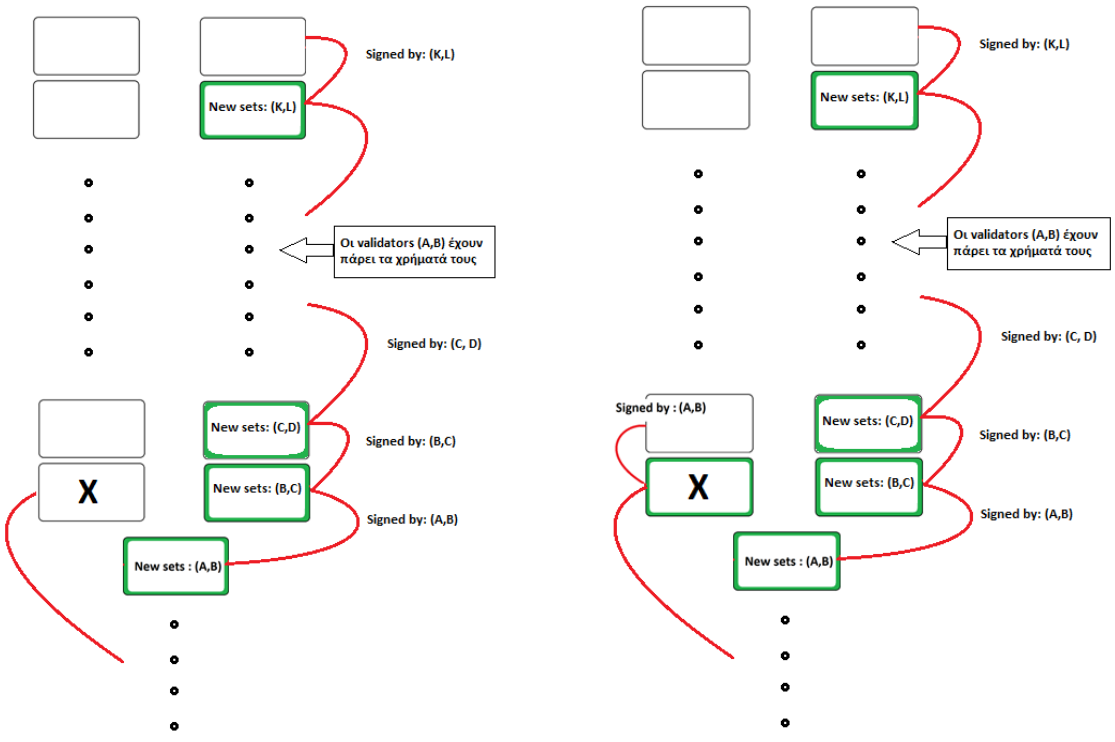
Ένα checkpoint y θα λέγεται justified (δικαιολογημένο), αν είναι το Genesis Block ή υπάρχει supermajority link x → y, δηλαδή τα 2/3 του front validator set και τα 2/3 του rear validator set έχουν αποστείλει ψήφο x → y, όπου το x είναι justified checkpoint.

Ένα checkpoint x θα λέγεται finalized (οριστικοποιημένο), αν είναι το Genesis Block ή υπάρχει supermajority link x → y, δηλαδή τα 2/3 του front validator set και τα 2/3 του rear validator set έχουν αποστείλει ψήφο x → y, όπου το x είναι justified checkpoint και το y είναι το αμέσως επόμενο block μετά το x. Για την οριστικοποίηση του x, όλες οι ψήφοι για finalization και justification του x, πρέπει να περιέχονται στο Blockchain του x, πριν τη δημιουργία του επόμενου checkpoint.

2.5.3 Αναθεώρηση μεγάλου εύρους

Μετά την έξοδο ενός validator από τα validator sets, τα χρήματά του δεν είναι άμεσα διαθέσιμα. Αντίθετα, ακολουθεί μία δίμηνη περίοδος κατά την οποία αν διαπιστωθεί οποιαδήποτε παραβίαση κανόνων από τον validator, τότε ο τελευταίος χάνει τα χρήματά του. Η ύπαρξη αυτής της καθυστέρησης μετά την αναχώρηση ενός validator, εισάγει το ζήτημα του συγχρονισμού μεταξύ των validators και των χρηστών.

Μόλις μία υποομάδα validators αποσυρθεί, αν αυτή η υποομάδα είχε πάνω από τα $\frac{2}{3}$ των καταθέσεων κάποια στιγμή στο παρελθόν, μπορούν να χρησιμοποιήσουν την ιστορική υπερπλειοψηφία τους, για να οριστικοποιήσουν αντικρουόμενα checkpoints, χωρίς τον φόβο να χάσουν τα χρήματά τους, μιας και θα τα έχουν ήδη αποσύρει. Μία τέτοια επίθεση ονομάζεται αναθεώρηση μεγάλου εύρους «Long range revision» και παρουσιάζεται στην Εικόνα 2.11



a) Τα Validator sets A,B έχουν αποχωρήσει στα ενδιάμεσα blocks Το checkpoint X περιμένει ψήφο από τα set A,B .

b) Τα set A,B χρησιμοποιούν την ιστορική τους πλειοψηφία και κάνουν finalize το checkpoint X, έχοντας αποχωρήσει

Εικόνα 2. 11 Οριστικοποίηση αντικρουόμενων checkpoints από validator sets που έχουν αποχωρήσει

Η ασφάλεια σε αυτή την περίπτωση επιτυγχάνεται μέσω του μηχανισμού επιλογής διακλάδωσης, όπου απαγορεύει την αναθεώρηση ενός ήδη finalized block. Επιπλέον προσδοκούμε ότι οι κόμβοι θα εισέρχονται στο δίκτυο σε κάποια τακτική συχνότητα, ώστε να λαμβάνουν την πλήρη εικόνα του blockchain ελαχιστοποιώντας έτσι τον κίνδυνο δημιουργίας ασυμφωνιών μεταξύ τους. Έτσι οποιαδήποτε οριστικοποίηση checkpoint σε χαμηλότερο ύψος θα αγνοούνταν, καθώς οι κόμβοι θα αρνηθούν να αλλάξουν ένα ήδη υπάρχον σε μεγαλύτερο ύψος.

Εικόνα 2. 12 Το λογότυπο του Casper

2.6 Σχετικές Εργασίες

Σε αυτήν την ενότητα παρουσιάζονται εργασίες με αντικείμενο σχετικό με αυτό της παρούσας διπλωματικής. Συγκεκριμένα, θα αναφερθούμε σε εργασίες και εφαρμογές στο χώρο του blockchain και των αποκεντρωμένων εφαρμογών, καθώς και σε αυτές που καταπιάνονται με τις νέες τεχνολογίες του Casper.

Εργασία [25]

Η παραπάνω εργασία εξετάζει τις τεχνολογίες του Casper και του Proof of Stake μέσω μίας προσομοίωσης ενός δικτύου blockchain σε γλώσσα προγραμματισμού Python. Η προσομοίωση λειτουργεί αυτόματα, χωρίς δηλαδή την παρέμβαση χρηστών όπως σε ένα πραγματικό blockchain δίκτυο. Η απλοποιημένη αυτή μορφή δικτύου έχει σκοπό να εξοικειώσει τον αναγνώστη με τη βασική λειτουργία του Casper και τον τρόπο που το τελευταίο επιτυγχάνει την συναίνεση. Επιπλέον με τη ρύθμιση των παραμέτρων του δικτύου Latency και αποσυνδεδεμένων validators, καταγράφεται επιρροή αυτών στο finalization νέων checkpoints στο blockchain. Τα αποτελέσματα των παραπάνω μετρήσεων αποτυπώνονται και γραφικά στο τελευταίο μέρος της εργασίας, δίνοντας έτσι μία εικόνα για το φάσμα των τιμών των μεταβλητών αυτών, για τις οποίες το δίκτυο παραμένει λειτουργικό (είναι εφικτή η οριστικοποίηση νέων checkpoints).

Εργασία [26]

Η συγκεκριμένη εργασία καταπιάνεται με τα πρωτόκολλα συναίνεσης στο blockchain. Αρχικά αναλύεται η δυσκολία τόσο στην ανάπτυξη όσο και στην αξιολόγηση των μηχανισμών συναίνεσης σε ασύγχρονα συστήματα. Αναφέρονται επίσης τα βασικά μοντέλα που χρησιμοποιούνται στην blockchain τεχνολογία για την επίτευξη της συναίνεσης μεταξύ των κόμβων του δικτύου (Crash tolerant, PBFT). Εν συνεχεία μέσω των βασικών κριτηρίων Plausible Liveness και Accountable Safety, στοιχειοθετείται η αδυναμία του BFT πρωτοκόλλου Tangaroa να αντιμετωπίσει κακόβουλους χρήστες. Στις δύο τελευταίες ενότητες παρουσιάζονται τα μοντέλα συναίνεσης που χρησιμοποιούνται σε permissioned και permisionless blockchains αντίστοιχα. Τα permissioned blockchains συστήματα που αναλύονται χρησιμοποιούν, κατά κύριο λόγο, τα BFT και PBFT consensus models ή παραλλαγές αυτών. Στην περίπτωση των permissionless blockchains, τα πρωτόκολλα PoW και PoS είναι αυτά που χρησιμοποιούνται σε μεγαλύτερο βαθμό. Ωστόσο, η συγκεκριμένη εργασία επικεντρώνεται σε μοντέλα συναίνεσης που ξεφεύγουν από τις παραδοσιακές αντιλήψεις τύπου BFT, Crash Tolerant και Proof of Work και αναλύει μεταξύ άλλων, τους μηχανισμούς συναίνεσης στα κρυπτονομίσματα Ripple, Stellar και IOTA

Εργασία [27]

Το πρωτόκολλα Proof of Activity, επιχειρεί να συνδυάσει τα πρωτόκολλα Proof of Work και Proof of Stake με σκοπό την παροχή μεγαλύτερης ασφάλειας στο δίκτυο. Η λειτουργία του PoA προϋποθέτει κατασκευή blocks από miners μέσω PoW, τα οποία ελέγχουν και «υπογράφουν» οι ενεργοί stakeholders του δικτύου. Το hash κάθε καινούργιου block header (όχι transactions) που επιλύει ένας miner κοινοποιείται στο δίκτυο. Το hash αυτό αντιστοιχίζεται με ντετερμινιστικό τρόπο σε ένα από τα shatoshi του δικτύου. Ο αλγόριθμος follow-the-shatoshi «ακολουθεί» την πορεία αυτού του κέρματος από την δημιουργία του μέχρι και τον τωρινό κάτοχό του. Μέσω του αλγορίθμου αυτού, οι online stakeholders κοιτούν αν το hash που κοινοποιήθηκε αντιστοιχεί σε shatoshi που έχουν στην κατοχή τους. Στην περίπτωση αυτή, ο stakeholder υπογράφει μέσω του ιδιωτικού του κλειδιού το hash και κοινοποιεί το αποτέλεσμα στους υπόλοιπους κόμβους. Η διαδικασία επαναλαμβάνεται N φορές (N ορισμένο από το εκάστοτε δίκτυο) έως ότου ο N-οστός stakeholder να επεκτείνει το block header τοποθετώντας συναλλαγές στο block. Το block κοινοποιείται και ελέγχεται για την ορθότητά του από τους κόμβους του δικτύου, ενώ την επιβράβευση λαμβάνουν οι N stakeholders και ο miner του block.

Το πρωτόκολλο ονομάζεται Proof of Activity γιατί προϋποθέτει από τους N stakeholder να είναι ενεργοί, καθώς σε αντίθετη περίπτωση κάποιο άλλο block header (με διαφορετικούς N stakeholder) θα είναι αυτό που θα υπογραφεί πρώτο. Το Proof of Stake μέρος του πρωτοκόλλου προκύπτει από τη λειτουργία του "follow-the-shatoshi" αλγορίθμου, ο οποίος δίνει περισσότερες πιθανότητες εκλογής σε stakeholders με πολλά shatoshi στην κατοχή τους. Το Proof of Work κομμάτι αφορά την επίλυση του block header που απαιτεί δαπάνη υπολογιστικής ισχύος από miners. Έτσι, παρότι το κίνητρο των stakeholders είναι να υπογράφουν όλες τις αλυσίδες, οι miners θα πρέπει να ακολουθούν τον κανόνα της μακρύτερης αλυσίδας.

Εργασία [28]

Οι συγγραφείς της συγκεκριμένης εργασίας προτείνουν μία εναλλακτική blockchain εφαρμογή χωρίς χρηματοοικονομική πλευρά, αλλά που θα βασίζεται στην γνώση. Ουσιαστικά, πρόκειται για μία δημόσια κατανεμημένη βάση δεδομένων όπου χρήστες ή πανεπιστήμια μπορούν να αποθηκεύουν-κοινοποιούν τις ιδέες και τις εργασίες τους. Με τον τρόπο αυτό, δημιουργείται ένα ημερολόγιο επιτευγμάτων στο χώρο των επιστημών, το οποίο δεν μπορεί να αλλαχθεί, παρά μόνο να ενημερωθεί με νέο υλικό. Μάλιστα επειδή κάθε καταχώρηση είναι υπογεγραμμένη θα μπορούσε η παραπάνω εφαρμογή να λειτουργήσει ως τρόπος διαφύλαξης πνευματικών δικαιωμάτων. Το συνάλλαγμα σε ένα τέτοιο δίκτυο θα είναι η φήμη (Reputation) και πάνω στην οποία θα βασίζεται η «εκπαιδευτική οικονομία». Μία τέτοια εφαρμογή, λοιπόν θα μπορούσε να ξεκινήσει με την κοινοπραξία εκπαιδευτικών ιδρυμάτων και εταιρειών, με τους πρώτους κόμβους να λαμβάνουν ένα αρχικό ποσό συναλλάγματος με βάση μιας αξιολόγησης του μέχρι τώρα επιστημονικού τους έργου (H-index για ακαδημαϊκούς, Amazon author rank για συγγραφείς κλπ). Έπειτα κάθε άτομο ή οργανισμός θα μπορούσε να ανταλλάξει την φήμη αυτό με άλλους χρήστες του δικτύου. Για παράδειγμα, ένα πανεπιστήμιο μπορεί να βραβεύσει έναν επιστήμονα αποστέλλοντας του ένα ποσό συναλλάγματος και κοινοποιώντας στην blockchain το αντίστοιχο πτυχίο ή βραβείο. Αντίστοιχα κάποιο άτομο μπορεί να ανταμείψει κάποιο άλλο για τη συμβολή του (credits) σε μία ιδέα ή εργασία την οποία κοινοποιεί στο δίκτυο. Τέλος, το συνάλλαγμα θα μπορούσε να «εξορύσσεται» από ιδρύματα, τα οποία ποντάρουν τη φήμη τους στην προσθήκη έγκυρων blocks στην αλυσίδα (μέσω αλγορίθμου Proof of Stake) για τον οποίο επιβραβεύονται. Μία τέτοια εφαρμογή αποτελεί ένα παράδειγμα, για τις πολλές και διαφορετικές εφαρμογές που μπορεί να βρει η blockchain τεχνολογία στο μέλλον, αλλάζοντας την καθημερινή ζωή των ανθρώπων.

3
Εργαλεία και Τεχνολογίες

Στο παρών κεφάλαια παρουσιάζουμε τα βασικά εργαλεία και τεχνολογίες που χρησιμοποιήθηκαν κατά την ανάπτυξης της blockchain εφαρμογής της παρούσας διπλωματικής. Πιο συγκεκριμένα, γίνεται αναφορά στη γλώσσα Python με την οποία υλοποιήθηκε η blockchain εφαρμογή των κόμβων. Επιπλέον παρουσιάζουμε την αρχιτεκτονική REST (Representational State Transfer) την οποία αξιοποιήσαμε μέσω της βιβλιοθήκη Python Flask Microframework, για την ανάπτυξη της εφαρμογής μας ως web application. Τέλος, αναφερόμαστε στη βάση δεδομένων γράφου Neo4j. η χρήση της οποίας με προσφέρει την απτή αναπαράσταση του blockchain και την γρήγορη προσπέλαση πληροφοριών μέσα σε αυτό.

3.1 Η γλώσσα προγραμματισμού Python

Η Python [29] είναι μια διερμηνευόμενη, υψηλού επιπέδου, γενικού σκοπού γλώσσα προγραμματισμού. Ανήκει στις γλώσσες προστακτικού προγραμματισμού και υποστηρίζει τόσο το διαδικαστικό (procedural programming) όσο και το αντικειμενοστραφές (object-oriented programming) προγραμματιστικό υπόδειγμα. Είναι δυναμική γλώσσα προγραμματισμού (dynamically typed) και υποστηρίζει συλλογή απορριμμάτων (garbage collection). Δημιουργήθηκε από τον Ολλανδό Guido van Rossum και κυκλοφόρησε για πρώτη φορά το 1991. Η φιλοσοφία σχεδίασης της Python τονίζει την αναγνωσιμότητα του κώδικα, με τη χρήση εσοχών (identation) κατά τη σύνταξή του, ξεχωρίζοντας έτσι τα διαφορετικά block ομαδοποίησης του κώδικα χωρίς να είναι απαραίτητη η χρήση αγκυλών. Επιπλέον, το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα από ότι θα ήταν δυνατόν σε γλώσσες προγραμματισμού.

Η Python διακρίνεται επίσης για την πληθώρα έτοιμων βιβλιοθηκών διαθέτει και που μπορούν να χρησιμοποιηθούν εύκολα και άμεσα. Οι βιβλιοθήκες μπορούν να επεκταθούν με νέα τμήματα γραμμένα σε C ή C++. Επιπλέον, η Python διαθέτει αρκετές δομές δεδομένων και παρέχει επαρκή υποστήριξη αυτών, με πιο σημαντικές δομές τις λίστες, τις πλειάδες και τα λεξικά.

3.1.1 Βασικά πλεονεκτήματα της γλώσσας Python

Η γλώσσα Python είναι μια γλώσσα απλή και εύκολη στην εκμάθηση, ισχυρή, δυναμική, αποδοτική, παραγωγική και επεκτάσιμη. Είναι κατάλληλη και για αρχάριους και για έμπειρους προγραμματιστές. Η απλότητα και η αναγνωσιμότητα του κώδικα που προσφέρουν την καθιστούν χρήσιμη τόσο για εκπαιδευτικούς σκοπούς όσο και για την ανάπτυξη ολοκληρωμένων εφαρμογών.

Η Python διαθέτει αποδοτικές δομές δεδομένων υψηλού επιπέδου και υποστηρίζει, χωρίς όμως να επιβάλλει, μια απλή αλλά συνάμα αρκετά αποτελεσματική προσέγγιση στον αντικειμενοστραφή προγραμματισμό. Υποστηρίζει και άλλες γνωστές προγραμματιστικές προσεγγίσεις, όπως είναι ο διαδικαστικός και ο συναρτησιακός προγραμματισμός. Η σύνταξή της είναι κομψή και οι τύποι της δυναμικοί. Άλλα θετικά χαρακτηριστικά της γλώσσας είναι η υποστήριξη εξαιρέσεων, οι δυναμικοί τύποι κωδικοποίησης, η αυτόματη διαχείριση μνήμης καθώς επίσης και μια εκτενής βιβλιοθήκη, που καθιστούν την Python μια από τις πιο διαδεδομένες γλώσσες προγραμματισμού.

Όπως αναφέραμε προηγουμένως, η Python για μια διερμηνευόμενη γλώσσα προγραμματισμού και μπορεί να χρησιμοποιηθεί τόσο για τη δημιουργία σεναρίων εντολών όσο και για τη γρήγορη ανάπτυξη ολοκληρωμένων εφαρμογών σε διάφορες περιοχές ενδιαφέροντος και στις περισσότερες πλατφόρμες υλικού υπολογιστών και Λειτουργικών Συστημάτων (Windows, Unix, Linux, Mac OS X, κ.λπ.). Τα προγράμματα σε Python είναι συμπαγή, ευανάγνωστα, και γράφονται και συντηρούνται γρηγορότερα σε σχέση με άλλες δημοφιλείς γλώσσες προγραμματισμού όπως οι C, C++ και Java. Ο κώδικας μπορεί να ομαδοποιηθεί σε αρθρώματα (modules) και πακέτα (packages).

3.1.2 Μειονεκτήματα της γλώσσας Python

Πέρα όμως από την πληθώρα των θετικών χαρακτηριστικών της Python, οφείλουμε να αναφέρουμε και μειονεκτήματα. Ο χρόνος εκτέλεσης των προγραμμάτων της Python μπορεί να μην είναι πάντα τόσο γρήγορος όσο είναι στις μεταγλωττιζόμενες (compiled) γλώσσες όπως η C και η C++. Αυτό οφείλεται στο ότι ένα πρόγραμμα σε Python δεν μεταγλωττίζεται σε δυαδικό κώδικα μηχανής που εκτελείται άμεσα από τον επεξεργαστή του υπολογιστή. Το μειονέκτημα αυτό αντισταθμίζεται από το ότι πολλές φορές είναι σημαντικότερη η εξοικονόμηση χρόνου που έχουμε κατά την ανάπτυξη μιας εφαρμογής σε Python. Επιπλέον, η γλώσσα Python δεν ενδείκνυται για την ανάπτυξη mobile applications, γι' αυτό και δεν συναντάται συχνά σε προγράμματα και εφαρμογές αυτού του χώρου. Τέλος, η Python δεν συνίσταται για εργασίες που απαιτούν μεγάλα ποσά μνήμης καθώς λόγω της ευελιξίας των τύπων δεδομένων η κατανάλωση μνήμης είναι επίσης υψηλή.

Εικόνα 3. 1 Το λογότυπο της γλώσσας προγραμματισμού Python

3.2 Η Αρχιτεκτονική REST

Η REST (Representational State Transfer) είναι ένα αρχιτεκτονικό στυλ, για την παροχή προτύπων μεταξύ υπολογιστικών συστημάτων στο διαδίκτυο, διευκολύνοντας έτσι της επικοινωνία τους. Η μεταβολή της κατάστασης των πόρων (π.χ. δεδομένα) του συστήματος περιγράφεται και μεταφέρεται στο σύστημα μέσω του πρωτοκόλλου HTTP από διάφορους clients (ανεξαρτήτως της γλώσσας στην οποία έχουν υλοποιηθεί). Τα βασικά χαρακτηριστικά του REST είναι τα εξής:

1. Αποκλειστική χρήση HTTP αιτημάτων/μεθόδων για την επικοινωνία του χρήστη με τον παροχές της δικτυακής υπηρεσίας.

Η βασική αρχή σχεδίασης του REST είναι η ένα-προς-ένα αντιστοίχιση μεταξύ λειτουργιών CRUD (create, read, update, delete) και HTTP μεθόδων. Σύμφωνα με αυτή την αντιστοίχιση:

- Για τη δημιουργία ενός πόρου στον server, χρησιμοποιούμε την μέθοδο POST.
- Για την ανάσυρση ενός πόρου, χρησιμοποιούμε την GET.
- Για την αλλαγή της κατάστασης ενός πόρου ή την ενημέρωσή του, χρησιμοποιούμε την PUT.
- Για την απομάκρυνση ή διαγραφή ενός πόρου, χρησιμοποιούμε την DELETE.

2. Είναι stateless

Η ουσία της «έλλειψης κατάστασης» είναι ότι οποιαδήποτε κλήση σε μια υπηρεσία REST δε θα πρέπει να αναφέρεται σε άλλη προγενέστερη κλήση. Όλες οι κλήσεις πρέπει να είναι ανεξάρτητες. Ο server δε γνωρίζει το τι έγινε με κάποια προηγούμενη κλήση, τη στιγμή που επεξεργάζεται την τρέχουσα κλήση, πράγμα που σημαίνει ότι κάθε φορά που χρησιμοποιήσουμε κάποια υπηρεσία, θα χρειαστεί να υπενθυμίσουμε τα δεδομένα μας, ανεξάρτητα από το αν πρόκειται για διαπιστευτήρια χρήστη ή για οποιαδήποτε άλλη πληροφορία. Αυτό που, από τη μία πλευρά, μπορεί να φαίνεται μειονέκτημα - το κουραστικό καθήκον της επανάληψης δεδομένων - είναι στην πραγματικότητα ένα από τα δυνατά του χαρακτηριστικά: δεδομένου ότι δεν τα απομνημονεύει, επιτρέπει μεγαλύτερη επεκτασιμότητα. Αυτό συμβαίνει καθώς σε αντίθετη περίπτωση, θα απαιτούνται πολύ ισχυροί εξυπηρετητές, που να μπορούν να αποθηκεύουν όλες τις καταστάσεις των πελατών τους.

3. Υποστηρίζει JSON και XML

Ένα άλλο πλεονέκτημα του REST API [31] είναι ότι ικανοποιεί τις προσδοκίες εκείνων που χρησιμοποιούν τη γλώσσα JSON [32] καθώς και αυτών που βασίζονται στην XML. Με τον τρόπο αυτό ικανοποιεί τις προτιμήσεις ενός μεγάλου συνόλου προγραμματιστών.

3.2.1 Flask Microframework

Το Flask αποτελεί μία REST βιβλιοθήκη της Python, που έχει σχεδιαστεί για να υποστηρίξει την ανάπτυξη των διαδικτυακών εφαρμογών, συμπεριλαμβανομένων των υπηρεσιών web, των δικτυακών πόρων και APIs διαδίκτυο. Ουσιαστικά παρέχει τον πρότυπο τρόπο σχεδίασης και υλοποίησης για εφαρμογές διαδικτύου στον παγκόσμιο ιστό WWW.

Τα βασικά χαρακτηριστικά του Python Flask web microframework είναι τα εξής:

- Περιέχει development server και λειτουργία αποσφαλμάτωσης (debugger) , όπου βοηθά κατά τη διαδικασία ανάπτυξης της εφαρμογής
- Εξυπηρετεί RESTful requests
- Υποστηρίζει Sessions
- Είναι Unicode-Based
- Έχει συμβατότητα με Google App Engine και WSGI 1.0
- Διαθέτει εκτεταμένο documentation
- Διαθέτει πληθώρα επεκτάσεων για τη βελτίωση των επιθυμητών λειτουργιών

Εικόνα 3. 2 Το λογότυπο του Python Flask

3.3 Η βάση δεδομένων γράφου Neo4j

Η Neo4j [34] πρόκειται για ένα σύστημα διαχείρισης βάσης δεδομένων γράφου και αποτελεί την πιο δημοφιλή εφαρμογή στο πεδίο αυτό. Η Neo4j είναι προγραμματισμένη σε γλώσσα Java, ωστόσο είναι προσιτή και σε εφαρμογές γραμμένες σε άλλες γλώσσες προγραμματισμού, χρησιμοποιώντας την Cypher Query Language μέσω ενός τελικού σημείου συναλλαγών HTTP ή μέσω του δυαδικού Bolt Protocol. Το Python API, το οποίο χρησιμοποιήθηκε στην εφαρμογή μας, προσφέρει έναν αντικειμενοστραφή τρόπο χειρισμού των οντοτήτων (κόμβοι και ακμές του γράφου) της βάσης δεδομένων. Η Neo4j προτιμήθηκε, κατά κύριο λόγο, για την γρήγορη προσπέλαση σε δεδομένα που προσφέρει, μέσω των αλγορίθμων γράφων που εκτελεί. Ένας δεύτερος λόγος είναι και η σαφής οπτικοποίηση των δεδομένων και των σχέσεων μεταξύ αυτών, που πραγματοποιείται μέσω της πλατφόρμας Neo4j Desktop.

3.3.1 Χαρακτηριστικά της Neo4j

- Χρησιμοποιεί SQL-Like γλώσσα ερωτημάτων, την Neo4j CQL
- Ακολουθεί το μοντέλο Property Graph Model για την αποθήκευση και τον χειρισμό των δεδομένων.
- Υποστηρίζει Ευρετήρια (Indexes) χρησιμοποιώντας το Apache Lucence
- Υποστηρίζει UNIQUE περιορισμούς
- Περιέχει ένα UI για την εκτέλεση εντολών CQL: Neo4j Data Browser
- Υποστηρίζει πλήρως τους κανόνες του ACID (Ατομικότητα, Συνέπεια, Απομόνωση και Ανθεκτικότητα)
- Χρησιμοποιεί την εγγενή αποθήκευση γραφημάτων με Native GPE (Graph Processing Engine)
- Υποστηρίζει την εξαγωγή δεδομένων ερωτήματος σε μορφή JSON και XLS
- Παρέχει ένα REST API, προσβάσιμο από οποιαδήποτε γλώσσα προγραμματισμού, όπως Java, Spring, Scala κλπ.
- Παρέχει πρόσβαση σε Java Script από οποιοδήποτε περιβάλλον εργασίας UI MVC όπως το Node JS [36].
- Υποστηρίζει δύο είδη API Java: Cypher API και Native Java API [37] για την ανάπτυξη εφαρμογών Java.

Εικόνα 3. 3 Το λογότυπο της Neo4j

3.3.2 Μοντέλο δεδομένων Neo4j

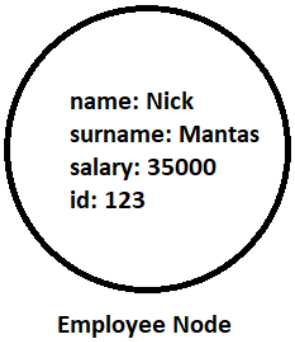
Η βάση δεδομένων γράφου Neo4j, ακολουθεί το μοντέλο δεδομένων Property Graph Data Model για την αποθήκευση και την διαχείρηση των δεδομένων. Τα βασικά χαρακτηριστικά του μοντέλου αυτού είναι:

- Τα δεδομένα παρουσιάζονται ως κόμβοι (Nodes), σχέσεις (Relationships) και ιδιότητες (Properties)
- Οι ιδιότητες είναι ζεύγη κλειδιών-τιμών
- Οι κόμβοι αναπαρίστανται με κύκλο και οι σχέσεις μεταξύ αυτών αναπαρίστανται με βέλη
- Οι σχέσεις έχουν κατευθύνσεις και είναι μονοκατευθυντικές
- Κάθε σχέση περιέχει τον "κόμβο εκκίνησης" ή "από κόμβο" και τον "προς κόμβο" ή "κόμβο τερματισμού"
- Τόσο οι κόμβοι όσο και οι σχέσεις περιέχουν ιδιότητες
- Οι σχέσεις συνδέουν κόμβους

Σύμφωνα και με το παραπάνω μοντέλο, τα δομικά χαρακτηριστικά της Βάση Δεδομένων Neo4j Graph είναι τα ακόλουθα:

1. Κόμβοι

Ο κόμβος είναι μια θεμελιώδης μονάδα ενός γραφήματος. Περιέχει ιδιότητες με ζεύγη κλειδιών-τιμών όπως φαίνεται στην εικόνα 3.5.



Εικόνα 3. 4 Αναπαράσταση κόμβου στη Neo4j

Εδώ, Node = "Employee" και περιέχει ένα σύνολο ιδιοτήτων ως ζεύγη κλειδιών-τιμών.

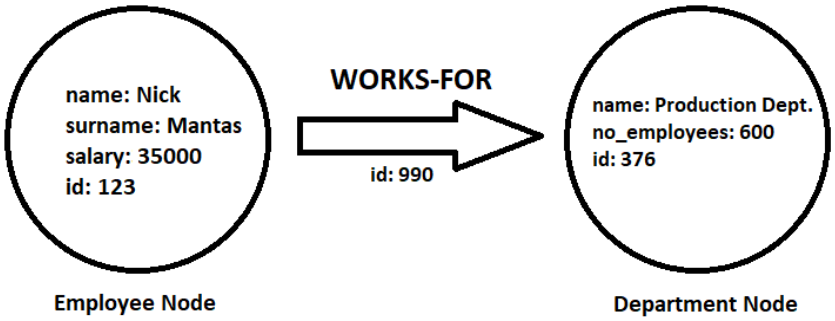
2. Ιδιότητες

Η ιδιότητα είναι ένα ζεύγος κλειδιού-τιμής για την περιγραφή κόμβων γραφήματος και σχέσεων. Ουσιαστικά Key = Value , όπου το Key είναι μια συμβολοσειρά (String) και το Value μπορεί να εκπροσωπηθεί χρησιμοποιώντας οποιονδήποτε τύπο δεδομένων της Neo4j.

Στο παράδειγμα της εικόνας 3.5, μία τέτοια σχέση είναι η “surname” : “Mantas”, όπου Key το “surname” με Value “Mantas”.

3. Σχέσεις

Οι σχέσεις είναι ένα άλλο σημαντικό δομικό στοιχείο μιας βάσης δεδομένων γράφου. Συνδέει δύο κόμβους όπως απεικονίζεται στην εικόνα 3.6.



Εικόνα 3. 5 Αναπαράσταση σχέσης στη Neo4j

Όπως υποδηλώνει, το σημάδι βέλους από το Employee στο Department, έτσι και η σχέση περιγράφει Employee WORKS_FOR Department. Κάθε σχέση περιέχει έναν κόμβο έναρξης και έναν τελικό κόμβο. Εδώ, το Employee είναι ένας κόμβος εκκίνησης και το Department είναι ένας τελικός κόμβος. Κατά συνέπεια, αυτή η σχέση θεωρείται "Εισερχόμενη σχέση" για τον κόμβο Department και "Εξερχόμενη σχέση" για τον κόμβο Employee.

Επιπλέον, όπως οι κόμβοι, έτσι και οι σχέσεις μπορούν να περιέχουν ιδιότητες ως ζεύγη κλειδιών-τιμών. Στην εικόνα 3.6, βλέπουμε ένα τέτοιο ζεύγος “id” : 990

4.Ετικέτες

Η ετικέτα συσχετίζει ένα κοινό όνομα σε ένα σύνολο κόμβων ή σχέσεων. Ένας κόμβος ή μια σχέση μπορεί να περιέχει μία ή περισσότερες ετικέτες. Μπορούμε να δημιουργήσουμε νέες ετικέτες σε υπάρχοντες κόμβους ή σχέσεις. Μπορούμε, επίσης να αφαιρέσουμε τις υπάρχουσες ετικέτες από τους υπάρχοντες κόμβους ή σχέσεις. Στην εικόνα 3.6 μπορούμε να παρατηρήσουμε ότι υπάρχουν δύο κόμβοι. Ο κόμβος της αριστερής πλευράς έχει μια ετικέτα: Employee και ο δεξιός κόμβος έχει μια ετικέτα: Department. Η σχέση μεταξύ αυτών των δύο κόμβων έχει επίσης μια ετικέτα "WORKS_FOR".

4
Σχεδιασμός και υλοποίηση Συστήματος

Στο παρών κεφάλαιο αναλύεται η διαδικασία υλοποίησης της εφαρμογής. Αρχικά, γίνεται περιγραφή της μακροσκοπικής αρχιτεκτονικής της εφαρμογής και των δομικών της στοιχείων. Εν συνεχεία, παρουσιάζουμε τις λεπτομέρειες σχεδίασης που αφορούν το δύο βασικά κομμάτια της εφαρμογής μας: το μοντέλο δικτύου blockchain που κατασκευάσαμε και το σύστημα Casper που εφαρμόσαμε πάνω σε αυτό. Σκοπός, είναι να δειχθούν οι καινοτομίες του Casper στην υπάρχουσα blockchain τεχνολογία, καθώς και ο τρόπος με τον οποίο ξεπεράσαμε τις σχεδιαστικές δυσκολίες και μετουσιώσαμε την υπάρχουσα θεωρία σε ένα λειτουργικό σύστημα.

4.1 Μακροσκοπική Αρχιτεκτονική Συστήματος

Σε αυτήν την παράγραφο θα αναφερθούμε στη μακροσκοπική αρχιτεκτονική της εφαρμογής και περιγράψουμε τα δομικά στοιχεία του προγράμματος και το ρόλο που το καθένα εκτελεί.

Για να προσομοιώσουμε ένα δίκτυο blockchain χρειαζόμαστε την εφαρμογή blockchain που τρέχει κάθε κόμβος (τοπικά) και το peer-to-peer δίκτυο στο όποιο αυτοί συνδέονται. Το πρώτο κομμάτι υλοποιήθηκε με τη γλώσσα προγραμματισμού Python, ενώ για την προσομοίωση του δικτύου χρησιμοποιήθηκε το Python Flask, ένα REST framework που μας επέτρεψε την επικοινωνία μεταξύ των κόμβων με http requests. Κάθε κόμβος του δικτύου, αρχικοποιεί και χρησιμοποιεί μια βάση δεδομένων Neo4j (http://localhost:7474), η οποία αποτελεί ένα αντίγραφο του blockchain και μας διευκολύνει τόσο στην οπτικοποίηση των όσων λαμβάνουν χώρα στο δίκτυο, καθώς και στην γρήγορη επίλυση ερωτημάτων, που διαφορετικά θα απαιτούσαν τη σειριακή εξέταση ολόκληρου του blockchain.

Το blockchain της εφαρμογής μας έχει διπλή υπόσταση: ως δομές δεδομένων (dictionaries) στα τοπικά αντίγραφα που διατηρούν οι κόμβοι και ως βάσης δεδομένων γράφου στη Neo4j. Αυτό σημαίνει ότι οι λειτουργίες όπως η δημιουργία ενός block, πραγματοποιούνται δύο φορές. Μία για το block ως δομή δεδομένων που αναμεταδίδεται στους υπόλοιπους κόμβους του δικτύου και μία για το block ως οντότητα στη βάση δεδομένων γράφου Neo4j. Αυτή η διπλή παρουσία του blockchain μας επιτρέπει, εκτός των άλλων, να καταχωρούμε και να λαμβάνουμε τα απαραίτητα δεδομένα με το βέλτιστο δυνατό τρόπο.

Σύμφωνα με τα όσα είπαμε η εφαρμογή μας αποτελείται από τα παρακάτω στοιχεία, σύμφωνα με τις λειτουργίες που εκτελούν:

- main : Αρχικοποιεί τον server στον οποίο λειτουργεί ο κάθε κόμβος, καθώς και τη βάση δεδομένων γράφου Neo4j.
- network : Εκτελεί όλες τις ενέργειες και ερωτήματα που σχετίζονται με τη βάση Neo4j.
- node : Αποθηκεύει τις δομές δεδομένων που περιγράφουν το blockchain και εκτελεί όλες τις ενέργειες ενός node, που σχετίζονται με αυτές.
- block : Αποθηκεύει και χειρίζεται τα δεδομένα ενός block.
- validator : Αποθηκεύει τις δομές δεδομένων που αφορούν το Casper και εκτελεί όλες τις ενέργειες ενός validator που σχετίζονται με αυτές.
- Flask Resources : Είναι υπεύθυνο για τη εξυπηρέτηση των http αιτημάτων που ανταλλάσσουν οι κόμβοι του δικτύου.
- parameters : Περιέχει τις παραμέτρους του προγράμματος, με τις οποίες μπορούμε να ρυθμίσουμε και να προσαρμόσουμε περαιτέρω την εφαρμογή.

Η λειτουργία της εφαρμογής μας περιγράφεται επιγραμματικά ως εξής : Όταν ένας κόμβος εισέρχεται στο δίκτυο συνδέεται με την Neo4j όπου και κοιτά για την ύπαρξη άλλων κόμβων στο δίκτυο. Στην περίπτωση που είναι ο πρώτος, κατασκευάζει το Genesis Block ως αντικείμενο τύπου Block (το οποίο αποθηκεύει τοπικά) και στη συνέχεια ως οντότητα στη βάση δεδομένων γράφου Neo4j. Στην περίπτωση που δεν είναι ο πρώτος, αλλά υπάρχουν κόμβοι στο δίκτυο, στέλνει σε αυτούς ένα http get request «/getblockchain». Οι υπόλοιποι κόμβοι απαντούν στο νέο χρήστη, αποστέλλοντάς του τα αποθηκευμένα δεδομένα τους, που περιγράφουν το υπάρχον blockchain. Από εκεί και μετά, κάθε κόμβος μπορεί να εκτελεί διάφορες ενέργειες (mining, transactions, voting κλπ), αποστέλλοντας τα κατάλληλα http αιτήματα τα οποία μπορούν να εξυπηρετηθούν από το τοπικό Flask Server. Όλα αυτά τα αιτήματα κοινοποιούνται σε όλους του κόμβους του δικτύου, ούτως ώστε το τελευταίο να διατηρεί τον peer-to-peer χαρακτήρα του.

Η πρόσβαση που οι κόμβοι στη Neo4j είναι περιορισμένη και πλήρως ελεγχόμενη. Όπως θα δούμε στη συνέχεια, μόνο ο εκάστοτε miner μπορεί να πραγματοποιεί πολύπλοκα ερωτήματα προς τη βάση, ενώ για την άλλες ενέργειες των nodes προτιμούνται τα τοπικά δεδομένα. Σε περίπτωση που η ανάγνωση και η εγγραφή από και προς τη βάση ήταν ελεύθερες, θα μπορούσαν να προκύψουν καθυστερήσεις και ασυμφωνίες.

4.2 Σχεδιασμός και Υλοποίηση της Blockchain εφαρμογής

Στην παράγραφο αυτή περιγράφεται πιο αναλυτικά, η διαδικασία μοντελοποίησης της blockchain εφαρμογής. Συγκεκριμένα, θα περιγραφεί ο τρόπος ενσωμάτωσης της βάσης δεδομένων γράφου στη Neo4j, ο τρόπος επικοινωνίας των κόμβων με χρήση του Python Flask Microframework, οι βασικές λειτουργίες των χρηστών του δικτύου, οι δομές δεδομένων που περιγράφουν το blockchain και ο χειρισμός αυτών για τη καλύτερη λειτουργία του προγράμματος. Εδώ, πρέπει να υπογραμμίσουμε ότι το δίκτυο bblockchain που κατασκευάσαμε δεν πληροί όλες τις προδιαγραφές (κυρίως σε θέμα ασφάλειας), που απαιτούνται από ένα σύγχρονο και λειτουργικό δίκτυο blockchain. Ωστόσο, αποτελεί ικανό μοντέλο για την επίδειξη των βασικών λειτουργιών μίας blockchain εφαρμογής, καθώς και για την εφαρμογή και την κατανόηση της λειτουργίας του Casper, το οποίο αποτελεί και το βασικό αντικείμενο της παρούσας διπλωματικής.

Η λειτουργία του προγράμματος ξεκινά με την κλάση Main, όταν ο εκάστοτε χρήστης συνδέεται στο δίκτυο. Κάθε κόμβος του δικτύου, "ακούει" σε διαφορετική Python Flask πόρτα, καθώς θέλουμε να προσομοιώσουμε διαφορετικούς χρήστες, οι οποίοι θα "τρέχουν" στο ίδιο μηχάνημα (σε αντίθεση με ένα κατανεμημένο Blockchain δίκτυο). Μέσω του Python API [38], συνδέουμε την εφαρμογή με την τοπική βάση, που έχουμε δημιουργήσει με το Neo4j Desktop Application [39] , η οποία και ακούει στην πόρτα 7474. Στην συνέχεια, ακολουθούν εντολές δημιουργίας των ετικετών «labels» που χρησιμοποιούνται για την κατηγοριοποίηση των διαφορετικών οντοτήτων της βάσης Neo4j. Ακολουθεί η δημιουργία δύο αντικειμένων: του Node, υπεύθυνο για το χειρισμό των τοπικών δεδομένων που περιγράφουν το υπάρχον blockchain και του Validator, υπεύθυνο για το χειρισμό των τοπικών δεδομένων που σχετίζονται με το Casper. Τα αντικείμενα αυτά, καθώς και οι ετικέτες της βάσης δίνονται ως όρισμα στον κατασκευαστή του αντικειμένου Network, το οποίο και είναι υπεύθυνο για όλες τις ενέργειες που σχετίζονται με τη βάση Neo4j. Εν ολίγοις το αντικείμενο τύπου Network μπορεί και χρησιμοποιεί τα τοπικά δεδομένα των Node και Validator για τη Neo4j, όχι όμως το αντίστροφο.

Η παραπάνω λειτουργία που εκτελεί η κλάση main, συνοψίζεται στις παρακάτω γραμμές κώδικα:

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

    #database initialization
    db = GraphDatabase("http://localhost:7474", username="neo4j",
                       password="123456789")

    #Neo4j labels
    db_blocks = db.labels.create("Blocks")
    db_transactions = db.labels.create("Transactions")
    db_validator = db.labels.create("Validator")
    db_nodes = db.labels.create("Nodes")
    db_checkpoints = db.labels.create("Checkpoints")
    db_messages = db.labels.create("Messages")
    db_votes = db.labels.create("Votes")

    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()

    my_node = Node(int(args.port))
    my_validator = Validator(int(args.port))
    my_blockchain = Network(my_node, my_validator, db, db_blocks,
                        db_transactions, db_nodes,db_checkpoints,
                        db_validator, db_messages, db_votes)

    app.run(host=args.host, port=args.port, debug=True,
            use_reloader=False)
```

Στον κατασκευαστή του αντικειμένου Network, εκτός από τις προαναφερθείσες μεταβλητές, αρχικοποιούνται και οι node_transaction, όπου αποθηκεύονται οι συναλλαγές προς δημοσίευση και node_ports, όπου αποθηκεύονται οι κόμβοι (Flask ports) που υπάρχουν στο δίκτυο. Όπως υλοποιήσαμε το δίκτυο, κάθε χρήστης πρέπει να γνωρίζει τις πόρτες των υπόλοιπων κόμβων του δικτύου, ώστε να μπορεί να δημοσιεύει τις διάφορες συναλλαγές, blocks και λοιπά, μέσω Http μηνυμάτων. Για το λόγω αυτό κάθε νεοεισαχθείς κόμβος αποθηκεύει το port number του στη Neo4j, για εκείνους που θα ακολουθήσουν. Με ένα απλό ερώτημα στη βάση λοιπόν, ο χρήστης αποθηκεύει τους κόμβους που υπάρχουν στο δικτύου και ελέγχει τον αριθμό τους. Αν αυτός είναι ο μοναδικός, τότε προχωρά στην κατασκευή του Genesis Block, αν όχι, τότε ζητά από τους υπάρχοντες χρήστες τις δομές δεδομένων που περιγράφουν το blockchain.

```
#if you are the only node in the network
if len(self.node_ports) == 1:
            #Create Genesis Block
            self.createGenesisBlock()
else:
            #Get existing Blockchain
            self.get_valid_chain()
```

Η δημιουργία του Genesis Block (όπως και κάθε άλλου block) γίνεται σε δύο φάσεις. Πρώτα δημιουργείται ένα αντικείμενο τύπου Block με τις κατάλληλες τιμές το οποίο κοινοποιείται σε όλους τους κόμβους του δικτύου. Στη συνέχεια δημιουργείται μία καταχώρηση του στην βάση Neo4j η οποία και είναι ορατή από τους χρήστες. Τα δεδομένα που αποθηκεύονται στο block σε κάθε περίπτωση δεν είναι τα ίδια, για την αποφυγή επανάληψης της πληροφορίας. Πληροφορίες όπως είναι οι συναλλαγές αποθηκεύονται στη Neo4j καθώς χρησιμοποιούνται μόνο από τον εκάστοτε miner (περιορισμένη πρόσβαση – αποφυγή συμφόρησης) και απαιτούν αναζήτηση σε ολόκληρο το blockchain (μικρότερος χρόνος απ' ότι σειριακά). Αντίθετα οι πληροφορίες όπως τα validator set του εκάστοτε block, έχουν σημασία μόνο για το block που αφορούν και απαιτούν συχνές αλλαγές, όποτε και είναι προτιμότερο να αποθηκεύονται σαν τοπικές μεταβλητές στα αντικείμενα blocks των κόμβων. Οι δύο συναρτήσεις κατασκευής του Genesis Block για κάθε περίπτωση φαίνονται παρακάτω.

Network - Database

```
#creates the database entry for the Genesis block and its transactions
def createGenesisBlock(self):
    #the node creation of genesis block should be first
    genesis_block = self.my_node.create_genesis_block()
    self.my_validator.new_validation_block(genesis_block)
    genesis_block_data = vars(genesis_block)

    #put genesis block transactions in node_transactions
    for data in GENESIS_DATA:
        data['transactionHash'] = hex_dig
        self.node_transactions.append(data)

    #create Neo4j instance of Genesis Block (both block and checkpoint)
    appended_block = self.database.nodes.create(name="Genesis Block",
                            height=0,hash=genesis_block_data['hash'],
                            timestamp=genesis_block_data['timestamp'],
                            justified="yes", finalized="yes")
    self.db_blocks.add(appended_block)
    self.db_checkpoints.add(appended_block)

    #Genesis Block is the first justified and finalized block
    self.my_validator.check_highest_justified(genesis_block)
    self.my_validator.check_highest_finalized(genesis_block)

    #create Neo4j instance of Genesis Block's transactions
    for d in self.node_transactions:

        appended_transaction=self.create_transaction(d['from'], d['to'],
                                            d['amount'],d['description'])
        appended_transaction.relationships.create(d['transactionHash'],
                                            appended_block)
    # Reset the current block transactions --> Waiting for new
    self.node_transactions = []
```

Node - Local

```
#creates local block object of the genesis block
def create_genesis_block(self):

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

    #genesis block is justified by definition
    genesis_block = Block(0, timestamp,"0","0",
         {hashlib.sha256((self.name).encode('utf-8')).hexdigest():1},
         {hashlib.sha256((self.name).encode('utf-8')).hexdigest() : 1},
         {hashlib.sha256((self.name).encode('utf-8')).hexdigest() : 1}, [0])

    genesis_block_hash = genesis_block.get_hash()

    #genesis block has None parent
    self.parent[genesis_block_hash] = None

    #genesis block has currently no children
    self.tail[genesis_block_hash] = genesis_block

    #genesis block is our selected branch
    self.selected_branch = genesis_block

    self.cold_start = False
    #we return the nessecary data for the database
    return genesis_block
```

4.2.1 Δομές δεδομένων blockchain

Σύμφωνα με τα όσα ειπώθηκαν αντιλαμβανόμαστε ότι δεν είναι απαραίτητη η αποθήκευση ολόκληρου του blockchain από τους κόμβους, παρά μόνο εκείνων των blocks που χρησιμοποιούνται στις διάφορες ενέργειές τους. Συγκεκριμένα, όλες οι εργασίες των κόμβων στο δίκτυο μας μπορούν να εκπονηθούν με τη χρήση των παρακάτω δομών, στις οποίες αποθηκεύουμε αυτά τα «χρήσιμα» blocks:

parent: dictionary αποθηκευμένο στο αντικείμενο node στο οποίο αποθηκεύονται τα hashes των checkpoints του blockchain με τη μορφή parent[child_checkpoint_hash] = parent_checkpoint_hash, ούτως ώστε από οποιοδήποτε checkpoint να μπορούμε να φτάσουμε αναδρομικά μέχρι στο genesis block (parent[genesis_block_hash] = None).

tail: dictionary αποθηκευμένο στο αντικείμενο node στο οποίο αποθηκεύονται τα blocks όπου δεν έχουν παιδία, δηλαδή οι «ουρές» του blockchain. με τη μορφή tail[block_hash] = Block .

validation_blocks: dictionary αποθηκευμένο στο αντικείμενο validator στο οποίο αποθηκεύονται τα blocks όπου μπορούν να δεχτούν ψήφους από τους validators, με τη μορφή validation_blocks [block_hash] = Block

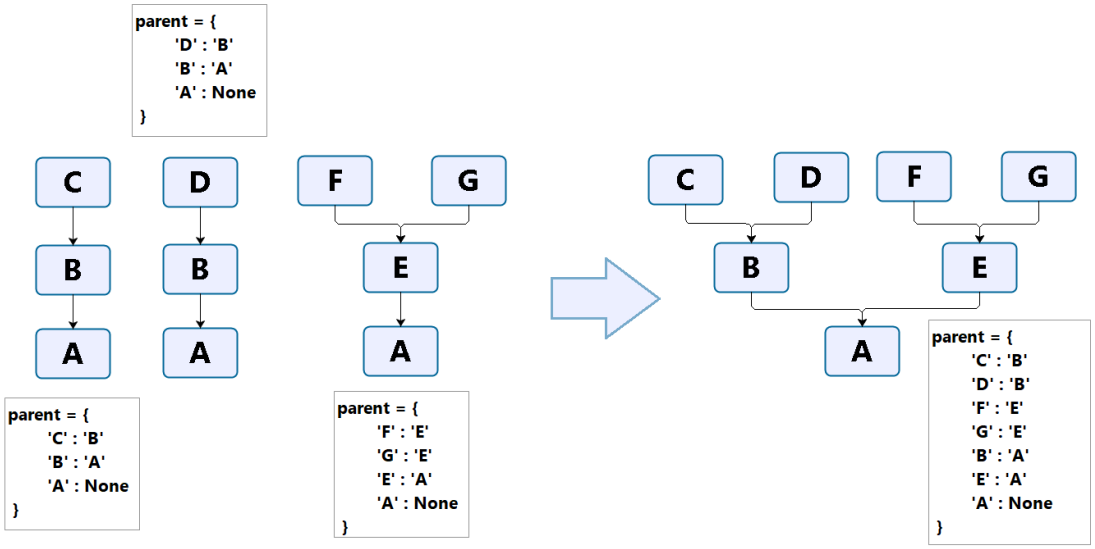
highest_justified: dictionary αποθηκευμένο στο αντικείμενο validator στο οποίο το hash και το ύψος του justified checkpoint στο μεγαλύτερο ύψος, με τη μορφή highest_justified = Block

highest_finalized: dictionary αποθηκευμένο στο αντικείμενο validator στο οποίο το hash και το ύψος του finalized checkpoint στο μεγαλύτερο ύψος, με τη μορφή highest_finalized = Block

Με τις παραπάνω πληροφορίες μπορούμε να περιγράψουμε πλήρως το checkpoint tree του εκάστοτε blockchain. Μάλιστα, αυτά είναι και τα δεδομένα που αποστέλλονται σε κάθε νέο χρήστη όταν εισέρχεται στο δίκτυο. Πρέπει να τονίσουμε ότι χρησιμοποιήθηκαν οι συγκεκριμένες δομές, καθώς με αυτές μπορούμε εύκολα να καταλήγουμε στο σωστό checkpoint tree, ακόμα και με την ύπαρξη ασυμφωνιών στο δίκτυο. Αυτό γίνεται εμφανές από την παρακάτω συνάρτηση, υπεύθυνη για την ενημέρωση κάθε νέου κόμβου από τους υπόλοιπους χρήστες του δικτύου:

```
#send a getblockchain request to the rest of the nodes and updates the
local values accordingly
def get_valid_chain(self):
   for port in self.node_ports:
      if port!=self.my_node.port:
.        try:
            response = requests.get('http://localhost:' + str(port) +
               '/getblockchain', headers =
               {'port':str(self.my_node.port)}).json()
            parent = response['parent']
            self.my_node.concat_parent(parent)
            tail = response['tail']
            self.my_node.concat_tail(tail)
            validation_blocks = response['validationblocks']

            self.my_validator.concat_validation_blocks(validation_blocks)
            highest_finalized = response['highestfinalized']

            self.my_validator.check_highest_finalized(highest_finalized)
            highest_justified = response['highestjustified']

            self.my_validator.check_highest_justified(highest_justified)
         except ConnectionError :
            print (str(port) + " is offline")

self.my_node.select_branch(self.my_validator.get_highest_justified())
```

Η λειτουργία της παραπάνω συνάρτησης είναι η ακόλουθη: Ο χρήστης στέλνει ένα Http get request «/getblockchain» στους υπόλοιπους κόμβους του δικτύου. Εκείνοι απαντούν αποστέλλοντας τις παραπάνω αποθηκευμένες δομές που περιγράφουν το blockchain. Ο χρήστης λαμβάνει τα δεδομένα αυτά, που μπορεί να διαφέρουν ανά κόμβο (μη ενημερωμένοι κόμβοι) και τα αποθηκεύει με τέτοιο τρόπο, ώστε να καταλήγει πάντα στο σωστό blockchain. Όσον αφορά τα highest justified και highest finalized checkpoints, η συμφωνία επιτυγχάνεται εύκολα, αφού και να υπάρχουν ανενημέρωτοι κόμβοι, στο τέλος θα επιλεγούν τα checkpoints στο μεγαλύτερο ύψος. Η δομή των parent και validation blocks είναι διευκολύνει και αυτή, την συμφωνία μεταξύ των κόμβων, αφού η σωστή δομή είναι αυτή που περιέχει το υπερσύνολο όλων των αποθηκευμένων blocks. Ένα τέτοιο παράδειγμα φαίνεται και στην εικόνα 4.1 .



Εικόνα 4. 1 Συμφωνία τριών κόμβων στο τελικό checkpoint tree.

Αν επιλέξουμε το υπερσύνολο των δομών και για την tails, το τελικό dictionary θα περιέχει σίγουρα όλες τις «ουρές» του blockchain. Εντούτοις, είναι πιθανόν να υπάρχουν κόμβοι που να διατηρούν blocks στο tails λανθασμένα, γιατί ενώ τα τελευταία απέκτησαν παιδιά, εκείνοι δεν ενημερώθηκαν γι' αυτό. Η δομή tails χρησιμοποιείται για την επιλογή του block – branch προς επέκταση. Όπως έχουμε αναφέρει στην παράγραφο 2.5, το fork choice rule του Casper επιβάλλει την επιλογή μόνο εκείνων των block τα οποία είναι απόγονοι του highest justified checkpoint. Έτσι με τη δημιουργία ενός νέου justified checkpoint σε μέγιστο ύψος, αφαιρούμε από τη δομή tail εκείνα τα blocks σε μικρότερο ύψος, πράγμα το οποίο περιορίζει το μέγεθος του dictionary και εξασφαλίζει ότι σε βάθος χρόνου η δομή θα περιέχει τα σωστά στοιχεία. Η παραπάνω διαδικασία αποτελεί κομμάτι της συνάρτησης επιλογής branch η οποία φαίνεται παρακάτω

```
def select_branch(self,justified):
    deleted = []
    hash = justified.get_hash()
    height = justified.get_height()
    if self.selected_branch!=None:

    #if our selected branch didn't get any children and it is still a
    descendant of the highest justified , we don't change it
    if self.selected_branch.get_hash() in self.tail and
    self.is_ancestor(self.selected_branch,hash):
        print(self.selected_branch.get_hash())
        return
    previous_branch = self.selected_branch
    #else we search for another tail that is a descendant of the highest
    justified checkpoint
    parent_block = None
    for t in self.tail:
    #remove tails in height lower that the highest justified checkpoint
        if self.tail[t].get_height() < height:
            deleted.append(t)
            continue
```

```
        #if our block is a descendant of/is the highest justified block,
        then it is a candidate branch
        if t==hash or self.is_ancestor(self.tail[t],hash):
           parent_block = self.tail[t]
           #with a greater preference to the child of our previous
           selected branch (if that exists)
           if parent_block.get_previous_hash()== previous_branch:
               break

        #if no tail block was found to be an ancestor of the highest
        justified checkpoint we throw KeyError to force a client update
        if parent_block == None:
               raise KeyError

        for t in deleted:
               del self.tail[t]

        self.selected_branch = parent_block
        print(self.selected_branch.get_hash())
```

Η πρώτη συνθήκη αφορά την προτίμηση του κόμβου στο υπάρχον branch, αν εξακολουθεί να είναι «ουρά» και απόγονος του highest justified checkpoint. Η συνάρτηση is_ansestor χρησιμοποιεί τη δομή parent ώστε να συμπεράνει αν ένα tail είναι απόγονος του highest_justified checkpoint. Τέλος, αν χωρίς να υπάρξει αλλαγή στο highest justified checkpoint, το επιλεγμένο από τον κόμβο branch αποκτήσει παιδί, η προτίμησή του κόμβου στρέφεται στο παιδί αυτό.

Εν κατακλείδι, οι χρήσιμες πληροφορίες, για τις πιο συχνές λειτουργίες των nodes αποθηκεύονται ως μεταβλητές των αντικειμένων Block. Οι δομές δεδομένων που αναλύθηκαν προσφέρουν γρήγορη πρόσβαση στα χρήσιμα blocks του blockchain και διευκολύνουν τις διάφορες εργασίες των node, όπως η επιλογή branch και η συναίνεση στο σωστό checkpoint tree.

4.2.2 Αναπαράσταση blockchain στη Neo4j

Η Neo4j και το Neo4j Desktop Application μας προσφέρουν μία οπτική αναπαράσταση του blockchain, που βοηθά στην καλύτερη κατανόηση της λειτουργίας του Casper, αλλά και της blockchain τεχνολογίας γενικότερα. Όπως αναφέρθηκε στις προηγούμενες παραγράφους, λόγω τις γρήγορης ταχύτητας στην αναζήτηση που προσφέρει η βάση δεδομένων γράφου, επιλέγουμε να αποθηκεύουμε και να χρησιμοποιούμε δεδομένα που απαιτούν αναζήτηση σε ολόκληρο το blockchain. Χαρακτηριστικό παράδειγμα είναι η εύρεση του ποσού νομισμάτων που διαθέτει ένας χρήστης, κοιτώντας όλες τις συναλλαγές στις οποίες συμμετέχει ως αποστολέας ή ως παραλήπτης. Αντίστοιχο παράδειγμα στο Casper είναι η αναζήτηση των ψήφων που έχει αποστείλει ένας validator, ώστε να διαπιστώσουμε αν ο τελευταίος έχει παραβιάσει τους κανόνες που έχουν τεθεί.

Ωστόσο, οι πολλές και ταυτόχρονες κλήσεις προς τη βάση δεδομένων, είτε για ανάγνωση είτε για εγγραφή, μπορούν να προκαλέσουν μεγάλες καθυστερήσεις και προβλήματα συγχρονισμού. Για το λόγο, πρόσβαση στη βάση έχει μόνο ο εκάστοτε miner, ο οποίος δημιουργεί τις καινούργιες οντότητες στη βάση και μπορεί να πραγματοποιεί ελέγχους στις αποθηκευμένες πληροφορίες του blockchain. Οι διαφορετικές οντότητες (τα αντίστοιχα «labels» της συνάρτησης main) που έχουμε στη βάση Neo4j είναι οι εξής:

Blocks : Όλα τα blocks στο υπάρχον blockchain. Κάθε block συνδέεται με το block-πατέρα του με ακμή «CHILD_OF»

Checkpoints : Τα blocks του blockchain σε ύψος πολλαπλάσιο του epoch_size. Τα block αυτά έχουν διπλή ετικέτα ώστε να συμμετέχουν στην αναπαράσταση τόσο του blockchain tree (Blocks) όσο και του Checkpoint tree (Checkpoints). Κάθε checkpoint συνδέεται με το checkpoint-πατέρα του με ακμή «PREVIOUS_CHECKPOINT»

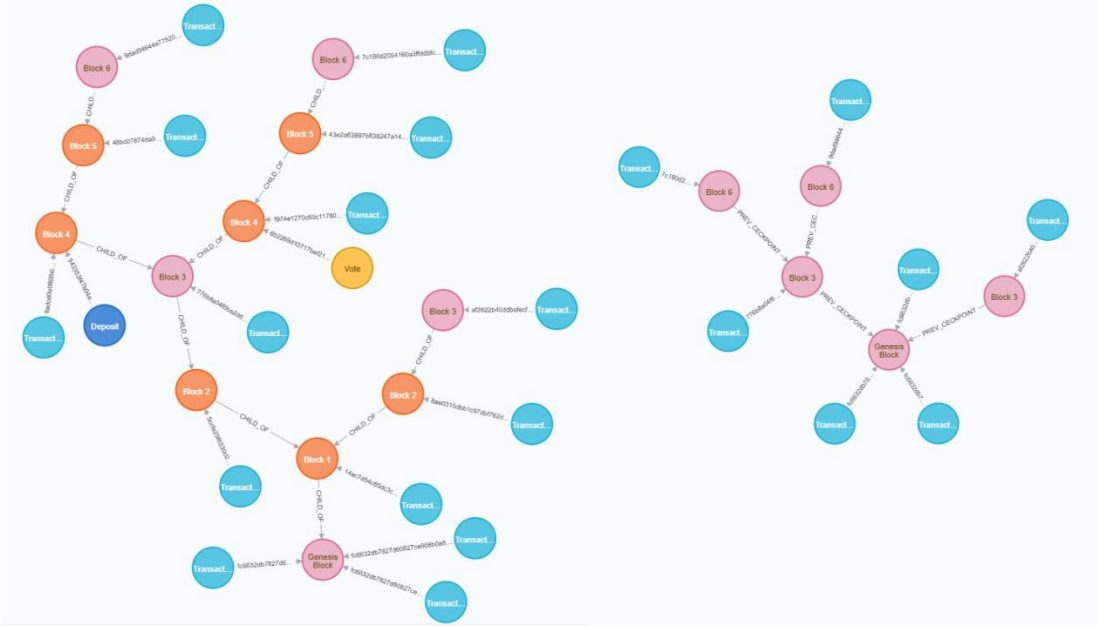
Transactions : Οι συναλλαγές του blockchain. Κάθε συναλλαγή συνδέεται με ακμή στο block, στο οποίο περιέχεται

Messages : Αντίστοιχα με τις συναλλαγές των nodes, τα messages αφορούν τους validators και χωρίζονται σε Deposit, Withdraw και Slashing messages. Κάθε message συνδέεται με ακμή στο block, στο οποίο περιέχεται

Votes : Αποτελεί την ψήφο ενός validator. Τα votes συνδέονται με ακμή στο block, στο οποίο περιέχονται.

Nodes : Οι κόμβοι του δικτύου.

Για την καλύτερη κατανόηση των παραπάνω, παρουσιάζεται στην εικόνα 4.2 η μορφή ενός blockchain tree με epoch_size=3 στην Neo4j καθώς και το αντίστοιχο Checkpoint tree.



Εικόνα 4. 2 Αναπαράσταση Blockchain tree και Checkpoint tree στη Neo4j.

Πέραν της οπτικής αναπαράστασης, η βάση δεδομένων γράφου μας διευκολύνει και στην γρήγορη προσπέλαση των αποθηκευμένων δεδομένων. Στο παρακάτω κομμάτι κώδικα βλέπουμε την συνάρτηση «find_user_balance», η οποία αποτελεί την πιο ουσιαστική αξιοποίηση των προαναφερθέντων πλεονεκτημάτων.

```
#calculates the balance of this node from the highest finalized block to the Genesis
block
def find_user_balance(self,user,hash):
    this_block_recieved = "MATCH (b:Blocks{hash: '" + hash+ "'})-[k]-
    (t:Transactions) WHERE t.to_address='" + user + "' RETURN SUM(t.amount)"
    results = self.database.query(this_block_recieved, returns=(int))
    total_balance = 0
    for r in results:
        total_balance += r[0]
    this_block_sent = "MATCH (b:Blocks{hash: '" + hash + "'})-[k]-
    (t:Transactions) WHERE t.from_address='" + user + "' RETURN
    SUM(t.amount)"
    results1 = self.database.query(this_block_sent, returns=(int))
    for r1 in results1:
        total_balance -= r1[0]
    this_block_deposit = "MATCH (c:Blocks{hash: '" + hash + "'})-[k]-(m:Messages)
    WHERE  m.name = 'Deposit' and m.user='" + user + "' RETURN SUM(m.amount)"
    results2 = self.database.query(this_block_deposit, returns=(int))
    for r2 in results2:
        total_balance -= r2[0]
    previous_blocks_recieved = "MATCH (b:Blocks{hash: '" + hash +
    "'})-[r:CHILD_OF*]->(b1:Blocks) OPTIONAL MATCH (b1)-[k]-
    (t:Transactions) WHERE  t.to_address='" + user + "' RETURN
    SUM(t.amount)"
    results = self.database.query(previous_blocks_recieved, returns=(int))
    for r in results:
        total_balance += r[0]
    previous_blocks_sent = "MATCH (c:Blocks{hash: '" + hash+ "'})-
    [r:CHILD_OF*]->(b:Blocks) OPTIONAL MATCH (b)-[k]-(t:Transactions) WHERE
    t.from_address='" + user + "' RETURN SUM(t.amount)"
    results1 = self.database.query(previous_blocks_sent, returns=(int))
    for r1 in results1:
        total_balance -= r1[0]
    #Deposit tokens are considered sent tokens
    previous_blocks_deposit = "MATCH (c:Blocks{hash: '" + hash+ "'})-
    [r:CHILD_OF*]->(b:Blocks) OPTIONAL MATCH (b)-[k]-(m:Messages) WHERE
    m.name = 'Deposit' and m.user='" + user + "' RETURN SUM(m.amount)"
    results2 = self.database.query(previous_blocks_deposit, returns=(int))
    for r2 in results2:
        total_balance -= r2[0]
    return total_balance
```

Η συνάρτηση αυτή υπολογίζει το ποσό των tokens που διαθέτει ένας χρήστης. Αυτό πραγματοποιείται με μία σειρά ερωτημάτων προς τη βάση (Cypher Queries), με τα οποία ζητά όλες τις συναλλαγές μίας αλυσίδας, στις οποίες συμμετέχει ο χρήστης «user». Συγκεκριμένα, ο miner ενός νέου block σε κάποιο branch του blockchain, λαμβάνει μία συναλλαγή με αποστολέα το χρήστη «user». Τότε, ο miner πρέπει να ελέγξει όλες της συναλλαγές του χρήστη αυτού, οι οποίες περιέχονται στη συγκεκριμένη αλυσίδα του blockchain, συμπεριλαμβανομένων και των transactions του παρόντος block. Τα ερωτήματα this_block_recieved και this_block_sent, αφορούν τις αυτές συναλλαγές στο νέο block με το χρήστη να είναι παραλήπτης και αποστολέας αντίστοιχα. Τα ερωτήματα previous_blocks_recieved και previous_blocks_sent λειτουργούν με τον ίδιο τρόπο, αυτή το φορά για κάθε προηγούμενο block τις παρούσας αλυσίδας. Τέλος πρέπει να συμπεριλάβουμε και tokens που καταθέτει ένας χρήστης προκειμένου να γίνει validator. Για το λόγο αυτό με τα ερωτήματα this_block_deposit και previous_blocks_deposit, αναζητούμε στο blockchain μήνυμα κατάθεσης από το χρήστη και αν αυτό υπάρχει συμπεριλαμβάνουμε το ποσό της κατάθεσης στα απεσταλμένα από το χρήστη tokens.

4.3 Κατανεμημένα πρωτόκολλα συναίνεσης

Σε αυτή την παράγραφο παρουσιάζονται τα πρωτοκόλλα σύστασης blocks, Proof-of-Work και Proof-of-Stake που υλοποιήσαμε και ενσωματώσαμε στο blockchain. Παράλληλα, θα εξηγήσουμε τις σχεδιαστικές μας επιλογές σχετικά με τη λειτουργία τους και πιο σκοπό τελικά εξυπηρετούν στο blockchain και στο Casper

4.3.1 Υλοποίηση Proof of Work

Το PoW πρόκειται για τη διαδικασία κατά την οποία οι κόμβοι του δικτύου συναγωνίζονται στην ταχύτερη εύρεση ενός αριθμού nonce, ο οποίος αποτελεί τη λύση ενός κρυπτογραφικού πάζλ. Ο αριθμός nonce είναι απόλυτα εξαρτώμενος από τα δεδομένα του εκάστοτε block, ενώ δεν υπάρχει κάποια συνάρτηση ή αλγόριθμος που να μπορεί να τον υπολογίσει. Αντίθετα, οι χρήστες πρέπει να δοκιμάσουν τυχαίους αριθμούς (Brute-force search) μέχρι κάποιος εξ' αυτών να δώσει το επιθυμητό αποτέλεσμα. Όπως είναι προφανές, οι κόμβοι που διαθέτουν μεγαλύτερη υπολογιστική δύναμη μπορούν να δοκιμάζουν αριθμούς σε μικρότερο χρόνο και άρα να έχουν μεγαλύτερες πιθανότητες εύρεσης της λύσης. Ο πρώτος που θα βρει έναν τέτοιο αριθμό, είναι και εκείνος που θα κατασκευάσει το καινούργιο block, λαμβάνοντας παράλληλα ένα ποσό tokens ως επιβράβευση. Το σύνολο αυτών των ενεργειών είναι γνωστό ως «mining».

Για την προσομοίωση μίας τέτοιας διαδικασίας στο πρόγραμμά μας χρειαζόμαστε ουσιαστικά τρία πράγματα: έναν αλγόριθμο κατακερματισμού, έναν αλγόριθμο παραγωγής αριθμών και μία μέθοδο επαλήθευσης του νικητή. Για τα δύο πρώτα χρησιμοποιούμε τις έτοιμες συναρτήσεις της Python «sha256» [40] της βιβλιοθήκης «hashlib» [41] και «random.choice» αντίστοιχα, ενώ για το τρίτο αξιοποιούμε την βάση Neo4j. Η διαδικασία mining ξεκινά με την αποστολή ενός http αιτήματος από τον κόμβο που θέλει να συμμετάσχει σε αυτήν, προς τον τοπικό Python Flask Server. Η κλάση η οποία εξυπηρετεί το αίτημα αυτό είναι η «powmine» και παρουσιάζεται παρακάτω:

```
class powmine(Resource):
    def get(self):
        if my_node.cold_start:
            print("You can't mine until the next epoch")
            return
        print("Mining...")
        prev_blocks = my_blockchain.get_blocks_number()

        parent_block = my_node.selected_branch
        newblock = my_node.prepare_block(parent_block)
        my_node.mine_pow(DIFFICULTY,newblock)
        current_blocks = my_blockchain.get_blocks_number()
        if prev_blocks == current_blocks:
            print("You won!")
            final_block = my_blockchain.create_block(vars(newblock))

            my_blockchain.broadcast_send_nodes(final_block,'/accept_block')
        else:
            print("Better luck next time.")
```

Ο επίδοξος miner ζητά τον αριθμό των blocks από τη βάση δεδομένων Neo4j προτού ξεκινήσει τη διαδικασία εξόρυξης. Εν συνεχεία κατασκευάζει το νέο block για την αλυσίδα που έχει επιλέξει, ως αντικείμενο της κλάσης Block μέσω της συνάρτησης «prepare_block». Το block αυτό δεν αποθηκεύεται στις δομές δεδομένων, επειδή πρώτα πρέπει να επιλυθεί το hash του. Το σκοπό αυτό εξυπηρετεί η συνάρτηση «mine_pow» της κλάσης Node, η οποία δοκιμάζει πέντε εκατομμύρια τυχαία nonce και επιστρέφει True αν κάποιο εξ αυτών επιλύει το κρυπτογραφικό πάζλ ή False στην αντίθετη περίπτωση. Σε περίπτωση False ο κόμβος ελέγχει αν ο αριθμός των blocks στη βάση έχει αυξηθεί σε σχέση με προηγουμένως. Αν ναι, προκύπτει ότι κάποιος άλλος κόμβος έφτασε γρηγορότερα στη λύση και δημιούργησε πρώτος το block του στη Neo4j, οπότε και η διαδικασία τερματίζει. Αν όχι, εκτελεί πάλι την «mine_pow» δοκιμάζοντας άλλα πέντε εκατομμύρια nonce. Σε περίπτωση που η «mine_pow» βρει ένα κατάλληλο nonce και επιστρέψει True, ο κόμβος πρέπει και πάλι να ελέγξει αν ήταν και ο γρηγορότερος, ώστε να μπορεί να δημοσιεύσει το block που έχει επιλύσει. Αν ο αριθμός δεν έχει αυξηθεί, τότε ο χρήστης είναι ο πρώτος που έλυσε το κρυπτογραφικό πάζλ, οπότε δημιουργεί την οντότητα του block στη βάση δεδομένων Neo4j και δημοσιεύει το block του στους υπόλοιπους κόμβους του δικτύου προς αποδοχή και αποθήκευση. Αντίθετα, αν ο αριθμός είναι αυξημένος, η διαδικασία τερματίζει.

Ο επαναληπτικός βρόχος while και η λειτουργία της «mine_pow» είναι τέτοιοι, ώστε ο έλεγχος για νικητές να γίνεται πιο τακτικά από τους επίδοξους miners. Σε περίπτωση που δεν υπήρχαν, ο κάθε miner θα έπρεπε να φτάσει μέχρι τη λύση του κρυπτογραφικού πάζλ ώστε να ελέγξει αν κάποιος άλλος τα κατάφερε πρώτος, με αποτέλεσμα την άσκοπη σπατάλη υπολογιστικών πόρων. Ο αριθμός των προσπαθειών επιλέχθηκε στα πέντε εκατομμύρια, έτσι ώστε να περιορίσουμε την μεγάλη σπατάλη πόρων, αλλά ταυτόχρονα να αποφύγουμε τα πολύ συχνά ερωτήματα προς τη βάση. Στο υπάρχον σύστημα οι κόμβοι ελέγχουν περίπου μία φορά το λεπτό για πιθανούς νικητές. Ας εστιάσουμε τώρα στη διαδικασία επίλυσης του block, που πραγματοποιείται με τη συνάρτηση «mine_pow», όπως φαίνεται παρακάτω:

```
#solve pow puzzle by brute force
def mine_pow(self,newblock):
    nonce = random_dig("")
    hash = newblock.calculate_hash(str(nonce))
    zero_string = '0'*DIFFICULTY
    #while (str(hash[:DIFFICULTY]) != zero_string):
    for x in range(0,5000000):
        nonce = self.random_dig (nonce)
        hash = newblock.calculate_hash(str(nonce))
        if (str(hash[:DIFFICULTY]) == zero_string):
            return True
    time.sleep(2)
    return False
```

Όπου random_dig είναι συνάρτηση παραγωγής τυχαίων συμβολοσειρών, τη βοήθεια της Python βιβλιοθήκης random και της συνάρτησης random.choice

Αρχικά υπολογίζουμε το hash που προκύπτει από τα δεδομένα του block και του αριθμού nonce με αρχική τιμή 0, με τη συνάρτηση «calculate_hash» του αντικειμένου Block. Στη συνέχεια ελέγχουμε αν το hash που προκύπτει ξεκινά με έναν προκαθορισμένο αριθμό μηδενικών που ορίζει η σταθερά «DIFFICULTY». Αν ναι το block έχει επιλυθεί και η διαδικασία τερματίζει, σε διαφορετική περίπτωση επαναλαμβάνουμε τον αλγόριθμο κατακερματισμού για διαφορετικά nonce, μέχρι να καταλήξουμε σε string που να ξεκινά με τον επιθυμητό αριθμό μηδενικών. Η σταθερά «DIFFICULTY» ορίζεται στις παραμέτρους του προγράμματος, έτσι ώστε ο χρόνος δημιουργίας ενός νέου block να είναι ελεγχόμενος. Όσο μεγαλώνει το «DIFFICULTY», η πιθανότητα εύρεσης hash μικραίνει, άρα ο μέσος χρόνος που απαιτείται για την επίλυση ενός νέου block αυξάνεται. Παράλληλα η εισαγωγή νέων χρηστών στη διαδικασία mining, μειώνει το μέσο χρόνο επίλυσης block, όποτε σε αυτήν την περίπτωση η αύξηση του «difficulty» διατηρεί την ισορροπία.

Όσον αφορά την συνάρτηση «calculate_hash» του αντικειμένου Block, λαμβάνει ως όρισμα έναν αριθμό ή string nonce και επιστρέφει τον κατακερματισμένο αριθμό hash. Το hash προκύπτει από τον κατακερματισμό της συνένωσης των nonce, height (ύψος block), timestamp (χρονική στιγμή δημιουργίας του block), data (συναλλαγές και μηνύματα του block), previous_hash (κατακερματισμένος αριθμός του block-πατέρα), μέσω του αλγόριθμου κατακερματισμού SHA-256. Οι παρακάτω γραμμές κώδικα εκτελούν τη λειτουργία που περιγράψαμε:

```
def calculate_hash(self,nonce):
    my_string = (str(self.height) + str(self.previousHash) +
    str(self.timestamp) + str(nonce)).encode('utf-8')
    hash_object = hashlib.sha256(my_string)
    hex_dig = hash_object.hexdigest()
    return (hex_dig)
```

Κατά την παραπάνω υλοποίηση του PoW πάρθηκαν ορισμένες σχεδιαστικές ελευθερίες. Πρώτον, όλοι οι κόμβοι ανταγωνίζονται μεταξύ τους, στην επίλυση ενός block ανεξάρτητα με το branch – αλυσίδα στην οποία ανήκουν. Το γεγονός αυτό δεν επηρεάζει το κριτήριο «longest difficulty-weighted chain» , της επικράτησης δηλαδή της αλυσίδας με τη μεγαλύτερη δυσκολία, αφού και εδώ η αλυσίδα με τη μεγαλύτερη υπολογιστική ισχύ έχει μεγαλύτερη πιθανότητα να προχωρήσει. Ο σχεδιασμός είναι τέτοιος ώστε να αποφεύγονται τα ταυτόχρονα πολύπλοκα queries προς τη βάση Neo4j, που μπορούν να προκαλέσουν καθυστερήσεις (δηλαδή εύρεση αριθμού blocks στο κάθε branch για όλους του miners, αντί για εύρεση αριθμού block στη Neo4j). Επιπλέον, τα μικρά branches που απαρτίζονται από λίγους ανενημέρωτους κόμβους, είναι σχεδόν αδύνατον να κατασκευάσουν νέα blocks αφού θα ανταγωνίζονται την πλειοψηφία των κόμβων του δικτύου. Έτσι αποτρέπεται η μετακίνηση περιττής πληροφορίας στο δίκτυο, όπως δηλαδή των blocks αυτών, που σε αντίθετη περίπτωση θα δημιουργούνταν, αλλά δεν θα είχαν ελπίδες οριστικοποίησης.

4.3.2 Υλοποίηση Proof of Stake

Η λειτουργία και η λογική του PoS διαφέρει αρκετά σε σχέση με αυτήν του PoW, κάτι που αποτυπώνεται και στο σχεδιασμό των δυο πρωτοκόλλων. Ουσιαστικά η υλοποίηση πρέπει να είναι τέτοια που να προσομοιώνει τα προτερήματα που προσφέρει το PoW και παράλληλα να διευκολύνει τη λειτουργία του Casper, την οποία θα αναλύσουμε στη συνέχεια. Εδώ, δεν έχουμε miners αλλά stakeholders οι οποίοι χρησιμοποιούν το stake τους, για συμμετάσχουν στην κλήρωση για το δικαίωμα κατασκευής του επόμενου block. Κάθε token αποτελεί και μία συμμετοχή, δηλαδή εκείνοι με τα περισσότερα, θα έχουν τις πιο πολλές συμμετοχές και άρα τις περισσότερες πιθανότητες να κερδίσουν. Ο νικητής της διαδικασίας ανακοινώνεται στο δίκτυο, και αναλαμβάνει την δημιουργία του block.

Στο blockchain της εφαρμογής μας η διαδικασία ξεκινά με τους κόμβους που επιθυμούν να συμμετάσχουν στη διαδικασία PoS να στέλνουν ένα http αίτημα στον τοπικό Python Flask Server. Η κλάση εξυπηρέτησης αυτού του αιτήματος είναι η «applypos» της οποίας και παραθέτουμε τον κώδικα:

```
# If all requirements are met (stake > 0, cold start = False)
stakeholder send his application to the rest of the nodes
class applypos(Resource):
    def get(self):
        if my_node.has_applied_stakeholder():
            print("You have already applied")
        else:
            parent_block = my_node.selected_branch.get_hash()
            stake = my_blockchain.find_user_balance(my_node.hex_name,
            parent_block)
            application = {
                'stakeholder': my_node.port,
                'branch': parent_block,
                'stake': stake
            }
            my_blockchain.broadcast_send_nodes(application,
           '/getposstakeholder')
```

Οι πρώτες γραμμές κώδικα διασφαλίζουν ότι ο κόμβος είναι αρκετή ώρα στο δίκτυο ώστε να μπορεί να συμμετάσχει στη διαδικασία. Με την επόμενη συνθήκη αποφεύγονται τα πολλαπλά αιτήματα για PoS από τον ίδιο κόμβο. Αν ο χρήστης ικανοποιεί τις απαραίτητες αυτές συνθήκες, προχωρά στην σύνταξη της αίτησης συμμετοχής στη διαδικασία. Η αίτηση αποτελείται από τρία στοιχεία: το χαρακτηριστικό του κόμβου, το branch στο οποίο βρίσκεται και το stake του, δηλαδή το συνολικό αριθμό tokens που διαθέτει. Τα δύο πρώτα στοιχεία είναι αποθηκευμένα ως μεταβλητές στην κλάση Node του προγράμματος. Το stake υπολογίζεται από την συνάρτηση «find_user_balance», που αναλύσαμε σε προηγούμενη παράγραφο και αφορά τις συναλλαγές του χρήστη που περιέχονται αποκλειστικά σε blocks του επιλεγμένου branch. Η αίτηση αποστέλλεται σε όλους τους ενεργούς κόμβους του δικτύου και εξυπηρετείται από την κλάση «getposstakeholder», όπως φαίνεται παρακάτω:

```
class getposstakeholder(Resource):
    def post(self):
        application = request.get_json()
        branch = application.get('branch')
        stakeholder = application.get('stakeholder')
        stake = application.get('stake')
        if branch == my_node.selected_branch.get_hash():
            my_node.add_pos_stakeholder(stakeholder,stake)
```

Όταν ένας κόμβος λάβει μία αίτηση συμμετοχής στο PoS, ελέγχει αν ο αποστολέας βρίσκεται στο ίδιο branch με εκείνον. Αν αυτό ισχύει, τότε ο κόμβος αποθηκεύει το stake και το όνομα του αποστολέα σε πίνακες της κλάσης Node. Αν όχι, τότε αγνοεί την αίτηση. Η εκλογή του νικήτη πραγματοποιείται και αυτή από τους κόμβους του εκάστοτε branch, με προγραμματισμένα http αιτήματα. Η κλάση εξυπηρέτησης αυτών είναι η «startpos». Η εκλογή του νικητή πραγματοποιείται με τη συνάρτηση «find_PoS_winner» της κλάσης Node. Το όνομα του νικητή, το branch που αφορά η κλήρωση καθώς και το συνολικό stake των stakeholders του συγκεκριμένου branch κοινοποιούνται στο δίκτυο με μορφή http αιτήματος.

```
class startpos(Resource):
    def get(self):
        if my_node.miners != []:
            winner = my_node.find_PoS_winner()
            stake = my_node.pos_miners_stake
            branch = my_node.selected_branch.get_hash()
            my_blockchain.broadcast_send_nodes({'winner': winner,
            'stake': stake, 'branch':branch}, '/poswinner')
        else:
            print("Not enough applies yet")
```

Η κλάση «poswinner» εξυπηρετεί αυτό το αίτημα και εκτελεί τις εξής ενέργειες: Αν ο παραλήπτης δεν βρίσκεται στο branch που επεκτάθηκε, δεν πραγματοποιεί καμία ενέργεια. Αντίθετα, αν βρίσκεται στο ίδιο branch, τότε διαγράφει τα αποθηκευμενα δεδομένα στην κλάση Node που σχετίζονται με την ολοκληρωμένη κλήρωση (stakeholders, stakes κλπ). Αν επιπλέον ο παραλήπτης είναι ο νικητής, κατασκευάζει το καινούργιο block, ως αντικείμενο τύπου block που κοινοποιεί στο δίκτυο, και ως οντότητα στη βάση δεδομένων γράφου Neo4j.

```
class poswinner(Resource):
    def post(self):
        data = request.get_json()
        winner = data.get('winner')
        stake = data.get('stake')
        branch = data.get('branch')
        if branch == my_node.selected_branch.get_hash():
            my_node.clear_pos_miners()
        if winner == my_node.port:
            print("You won")
            parent_block = my_node.selected_branch
            newblock = my_node.prepare_block(parent_block,stake)
            my_blockchain.create_block(vars(newblock))
            my_blockchain.broadcast_send_nodes(vars(newblock),
       '/accept_block')
        else:
            print("Better luck next time")
            my_blockchain.node_transactions = []
```

Για την προσομοίωση της διαδικασίας εκλογής του νικητή stakeholder, που πραγματοποιεί η συνάρτηση «find_PoS_winner», χρησιμοποιείται μόνο ένας αλγόριθμος παραγωγής τυχαίων αριθμών και η λίστα των αποθηκευμένων stakeholders. Αρχικά παράγουμε έναν τυχαίο ακέραιο «lucky_number» στο διάστημα [1, total_stake] και αρχικοποιούμε μία μεταβλητή sum = 0. Στη συνέχεια ξεκινάμε να διατρέχουμε σειριακά την λίστα των αποθηκευμένων stakeholders. Για κάθε stakeholder που συναντάμε, προσθέτουμε το stake του στην μεταβλητή sum. Όταν ένα stake προστεθεί στο μερικό άθροισμα sum και το εκείνο φτάσει ή ξεπεράσει τον αριθμό «lucky_number» ο αντίστοιχος stakeholder ανακηρύσσεται νικητής. Με τον τρόπο αυτό ο κάθε stakeholder καταλαμβάνει ένα διάστημα ευνοϊκών αριθμών ανάλογα με το stake του. Αν ο «lucky_number» βρεθεί εντός αυτού του διαστήματος ο stakeholder θα είναι και ο νικητής. Στο παρακάτω παράδειγμα γίνονται ξεκάθαρα τα όσα αναφέρθηκαν.

Έστω ότι η λίστα pos_stakeholders έχει αποθηκευμένα τα παρακάτω στοιχεία:
- stakeholder: User1, stake: 100 tokens
- stakeholder: User2, stake: 300 tokens
- stakeholder: User3, stake: 200 tokens
- stakeholder: User4, stake: 50 tokens

Το συνολικό stake των stakeholders είναι total_stake = 100 + 300 + 200 + 50 = 650
Σύμφωνα με όσα είπαμε το «lucky_number» είναι ένας ακέραιος αριθμός στο διάστημα [1, total_stake] , δηλαδή στο διάστημα [1, 650]
Διατρέχοντας τη λίστα σειριακά υπολογίζουμε το sum για κάθε stakeholder. Υπενθυμίζουμε ότι ο νικητής ανακηρύσσεται ς, όταν ικανοποιηθεί η συνθήκη:

$$\text{lucky\_number} \leq \text{sum}$$

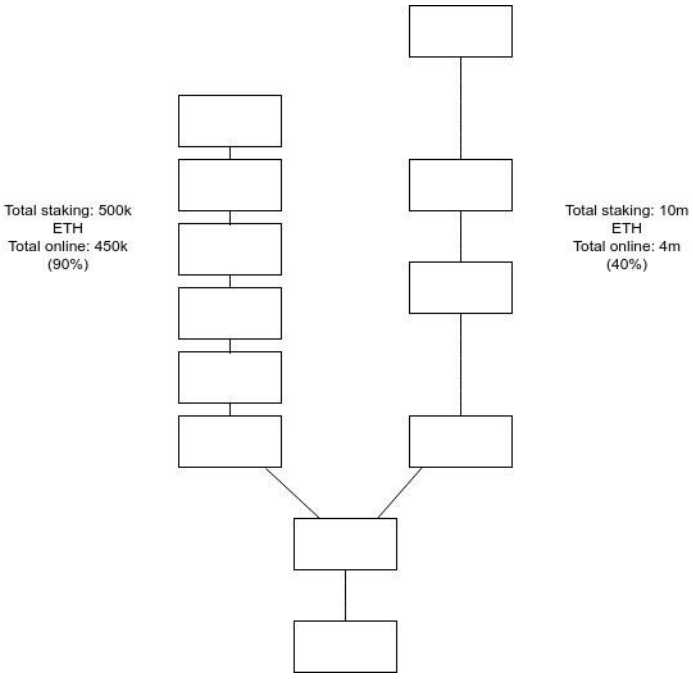
Έτσι έχουμε:
User1 : sum = 0 + 100 = 100 => Αν «lucky_number» στο διάστημα [1, 100]
νικητής ο User1
User2 : sum = 100 + 300 = 400 => Αν «lucky_number» στο διάστημα [101, 400]
νικητής ο User2
User3 : sum = 400 + 200 = 600 => Αν «lucky_number» στο διάστημα [401, 600]
νικητής ο User3
User4 : sum = 600 + 50 = 650 => Αν «lucky_number» στο διάστημα [601, 650]
νικητής ο User4
Βλέπουμε ότι το ευνοϊκό διάστημα τιμών για κάποιον stakeholder είναι ανάλογο με το stake του. Δηλαδή αν ο User2 έχει καταθέσει τριπλάσιο stake από τον User1, τότε θα έχει και τριπλάσιες πιθανότητες για εκλογή σε σχέση με αυτόν.

Ο κώδικας της συνάρτησης «find_PoS_winner» που εκτελεί την παραπάνω διαδικασία του νικητή stakeholder είναι ο εξής:

```
def find_PoS_winner(self):
    lucky_number=randit(1,self.pos_stakeholders_stake)
    sum = 0
    for d in self.pos_stakeholders:
        sum = sum + d['stake']
        if lucky_number <= sum:
            winner = d['miner']
            break;
    print("The winner is User"+str(winner)+" with the lucky number "
    + str(lucky_number))
    return winner
```

Η συναίνεση στο PoW, επιτυγχάνεται με τους κόμβους να εμπιστεύονται την πιο ασφαλή αλυσίδα, Η ασφάλεια αυτή είναι άρρηκτα συνδεδεμένη με την δυσκολία κατασκευής της αλυσίδας, η οποία στο PoW μεταφράζεται σε υπολογιστική ισχύ. Έτσι μπορούμε εύκολα να συμπεράνουμε ότι το branch για το οποίο δαπανήθηκαν περισσότεροι πόροι, είναι το μεγαλύτερο σε μήκος, αφού αυτό περιλαμβάνει τα περισσότερα blocks. Στο PoS χρειαζόμαστε μία αντίστοιχη μονάδα μέτρησης της δυσκολίας για να καταλήξουμε στην πιο ισχυρή αλυσίδα, ενώ θα δείξουμε ότι η τακτική της «μεγαλύτερης σε μήκος αλυσίδας» δεν λειτουργεί αποτελεσματικά. Η δυσκολία στο PoS μεταφράζεται σε staking, το συνολικό δηλαδή «ποντάρισμα» που έχει λάβει ένα block σε κάποιο ύψος. Αυτός είναι και ο λόγος για τον όποιο στην παραπάνω υλοποίηση διαχωρίσαμε τη διαδικασία εκλογής ανά branch. Ώστε να είναι ξεκάθαρο ποίοι stakeholders «ποντάρουν» σε κάθε branch του Blockchain tree. Η παράμετρος staking αποθηκεύεται στα Blocks αντικείμενα, αφού όπως θα δούμε στη συνέχεια, αποτελεί βασικό κριτήριο ψήφου των Casper validators.

Στο παράδειγμα της εικόνας 4.3 βλέπουμε δύο branches A και B του Blockchain tree. Η αλυσίδα A καταλήγει να είναι μεγαλύτερη σε μήκος, αφού οι stakeholders σε αυτήν εμφανίζονται κατά 90% για τη δημιουργία νέων block, σε αντίθεση με την B που μόλις το 40% είναι κάθε φορά online. Αν θεωρήσουμε τώρα ότι η αλυσίδα A κατασκευάστηκε από έναν κακόβουλο κόμβο με stake πολύ μικρότερο από αυτό της αλυσίδας B, μπορούμε να καταλάβουμε ότι το κριτήριο της μεγαλύτερης σε μήκος αλυσίδας δεν λειτουργεί.



Εικόνα 4. 3 Παράδειγμα ανεπάρκειας κριτηρίου «longest-chain» στα πρωτόκολλα PoS.

Όπως γίνεται αντιληπτό, το PoS, στην προσπάθεια του να προσομοιώσει την καταβολή ενέργειας του PoW, αλλάζει τα κίνητρα των stakeholders από την αλυσίδα της μεγαλύτερης δυσκολίας σε όλες τις υπάρχουσες αλυσίδες, αφού αυτή είναι η στρατηγική με το μέγιστο δυνατό κέρδος για τους stakeholders.

Τέλος θα αναφερθούμε στην κλάση εξυπηρέτησης «/accept_block» οποία λαμβάνει τα Block που κοινοποιούν οι miners και τα αποθηκεύει στις κατάλληλες δομές. Το Block αποστέλλεται ως json object, έχει δηλαδή τη μορφή ενός dictionary με keys τις μεταβλητές του αντικειμένου Block και values τις τιμές αυτών. Από τα στοιχεία αυτά μπορούμε να δημιουργήσουμε σε κάθε κόμβο το αντίστοιχο αντικείμενο Block, το οποίο είναι και αυτό που τελικά αποθηκεύουμε.

Η κλάση «/accept_block» είναι όμως χρήσιμη και για έναν επιπλέον λόγο: μας βοηθά στην εσκεμμένη δημιουργία διακλαδώσεων στο Blockchain tree, ώστε πάνω σε αυτές να επιδειχθεί η σωστή λειτουργία του μηχανισμού Casper θα δούμε στη συνέχεια. Η διαδικασία είναι απλή: στο αρχείο με τις παραμέτρους του προγράμματος αποθηκεύουμε έναν αριθμό DROP_MESSAGE_PRB. Η τιμή αυτού ρυθμίζει την πιθανότητα ένας κόμβος να μη λάβει ένα νέο Block. Συγκεκριμένα η πιθανότητα μη αποδοχής ενός νέου Block είναι 1/DROP_MESSAGE_PRB, με αποτέλεσμα αυτή να μεγαλώνει όσο το DROP_MESSAGE_PRB μικραίνει.

Η μη αποδοχή ενός Block από έναν κόμβο έχει ως αποτέλεσμα, εκείνος να παραμείνει σε προηγούμενο Block που πιθανώς να μην αποτελεί πλέον ουρά του Block. Έτσι σε περίπτωση που στο μέλλον κληθεί να κατασκευάσει ένα Block, θα το τοποθετήσει λανθασμένα μετά από κάποιο Block, το οποίο έχει ήδη αποκτήσει παιδί, δημιουργώντας έτσι μία δεύτερη παράλληλη αλυσίδα. Με τον μηχανισμό αυτόν και τις κατάλληλες ρυθμίσεις στην παράμετρο DROP_MESSAGE_PRB μπορούμε με σιγουριά να δημιουργήσουμε branches, χωρίς να βασιζόμαστε στην ελάχιστη πιθανότητα ταυτόχρονης επίλυσης block από τους miners και παράλληλα να δείξουμε πως το δίκτυο μας διατηρεί τη συνοχή του παρά την ύπαρξη ανενημέρωτων κόμβων.

Ο κώδικας της accept_block είναι ο παρακάτω:

```
#When a block is created it is sent to the nodes to accept it and update
their local copies of the blockchain
class accept_block(Resource):
    def post(self):
        block = request.get_json()
        newblock = Block(block.get('height'), block.get('timestamp'),
        block.get('previousHash'),block.get('previousCheckpoint'),
        block.get('new_front'), block.get('current_front'),
        block.get('current_rear'),  block.get('justified_checkpoints'),
        block.get('front_votes'), block.get('rear_votes'),
        block.get('stake'), block.get('hash'))
        if block['height']%EPOCH_SIZE==0 or block['height']%EPOCH_SIZE==1:
            my_validator.new_validation_block(newblock)
            my_validator.has_voted=[]
            my_node.cold_start = False
        #We add a DROP_MESSAGE probability factor to purposely create
branches
        x = randint(1, DROP_MESSAGE_PRB)
        if x != 1 :
            my_node.register_block(newblock)
            highest_justified = my_validator.get_highest_justified()
            try:
                my_node.select_branch(highest_justified)
            except KeyError:
                print ("Something went wrong. Updating client...")
                my_blockchain.get_valid_chain()
        my_blockchain.node_transactions = []
```

4.4 Υλοποίηση Casper Blockchain

Στην παρούσα παράγραφο παρουσιάζονται η υλοποίηση του μηχανισμού συναινετικής επιλογής κανονικής αλυσίδας Casper και η εφαρμογή του στο μοντέλο εφαρμογής blockchain που κατασκευάσαμε. Ταυτόχρονα, θα δείξουμε πως η υπάρχουσα θεωρία και οι νέες έννοιες που εισάγει το Casper μετουσιώνονται σε γραμμές κώδικα στο πρόγραμμά μας. Τέλος, θα αναλύσουμε τη λειτουργία και τις αρμοδιότητες των κόμβων που μετέχουν στις σχετικές διαδικασίες.

4.4.1 Υλοποίηση δυναμικών validator sets

Κάθε κόμβος έχει τη δυνατότητα να γίνει validator αποστέλλοντας ένα http αίτημα στον τοπικό Python Flask Server. Το αίτημα αυτό πρόκειται για ένα μήνυμα κατάθεσης και η κλάση που το εξυπηρετεί είναι η «deposit». Ο κόμβος είναι σε θέση να ορίσει το stake του τα tokens, δηλαδή που θα δεσμεύσει για όσο καιρό είναι validator. Το stake κάθε validator είναι πολύ σημαντικό, από αυτό καθορίζει τις επιβραβεύσεις και οι τιμωρίες που θα λαμβάνει καθώς το ποσοστό το επί τις εκατό θα αντιπροσωπεύει η ψήφος του. Η κλάση «deposit» τις οποίας ο κώδικας παρουσιάζεται παρακάτω, κατασκευάζει ένα transaction με περιγραφή «Validator Deposit Message» η όποια κοινοποιείται στο δίκτυο και αντιπροσωπεύει την κατάθεση του validator. Οι κόμβοι λαμβάνουν την κατάθεση και την αποθηκεύουν στο node_transactions, όπως κάνουν και για μία κανονική συναλλαγή.

```
class deposit(Resource):
    def post(self):
        transaction = request.get_json()
        transaction['from'] = my_node.hex_name
        transaction['description'] = "Validator Deposit Message"
        my_blockchain.broadcast_send_nodes(transaction,
       '/gettransaction')
```

Αντίστοιχα, για την αποχώρηση ενός validator από το validator set, πρέπει ο τελευταίος να αποστείλει http αίτημα ανάληψης. Η κλάση εξυπηρέτησης του αιτήματος «withdraw», λειτουργεί παρόμοια με την «deposit» δημιουργώντας ένα transaction με περιγραφή «Validator Withdraw Message», με τη διαφορά ότι το ποσό ανάληψης καθορίζεται από το αρχικό stake και τις επιβραβεύσεις ή τιμωρίες που έλαβε ο validator κατά την «θητεία» του. Εδώ πρέπει να σημειωθεί ότι για την αποφυγή χειρισμού πολλαπλών αιτημάτων από τον ίδιο κόμβο, το Casper απαγορεύει την επανένταξη κόμβων στα validator sets, που έχουν υπάρξει validators στο παρελθόν.

```
class withdraw(Resource):
    def get(self):
        transaction={}
        transaction['from'] = my_node.hex_name
        transaction['description'] = "Validator Withdraw Message"
        transaction['amount'] = 0
        my_blockchain.broadcast_send_nodes(transaction,
       '/gettransaction')
```

Όπως αναφέρθηκε στην παράγραφο 2.5, οι αλλαγές στα validator sets λαμβάνουν χώρα κατά την οριστικοποίηση των checkpoints. Δηλαδή, αν ένα μήνυμα κατάθεσης ενταχθεί στο Block X, ο αντίστοιχος κόμβος θα εισέλθει στο front_validator_set, όταν είτε το Block X (αν είναι ταυτόχρονα και checkpoint) είτε κάποιο checkpoint-απόγονός του γίνει finalized. Με την οριστικοποίηση κάθε checkpoint το rear_validator_set αποχωρεί και το front_validator_set παίρνει τη θέση του. Αν τώρα το μήνυμα που είναι αποθηκευμένο στο Block X είναι μήνυμα ανάληψης, η οριστικοποίηση των προαναφερθέντων checkpoints σηματοδοτεί την αποχώρηση του κόμβου μόνο από το front_validator_set, ενώ εξακολουθεί να βρίσκεται στο rear_validator_set. Για την ολοκληρωτική έξοδό του, απαιτείται η οριστικοποίηση ενός ακόμα checkpoint-απογόνου του Block X.

Η παραπάνω διαδικασία γίνεται εύκολα κατανοητή με τη χρήση τριών sets για κάθε block:

New_front: Αποτελεί το νέο front_validator_set που θα ισχύσει με την οριστικοποίηση του παρόντος block

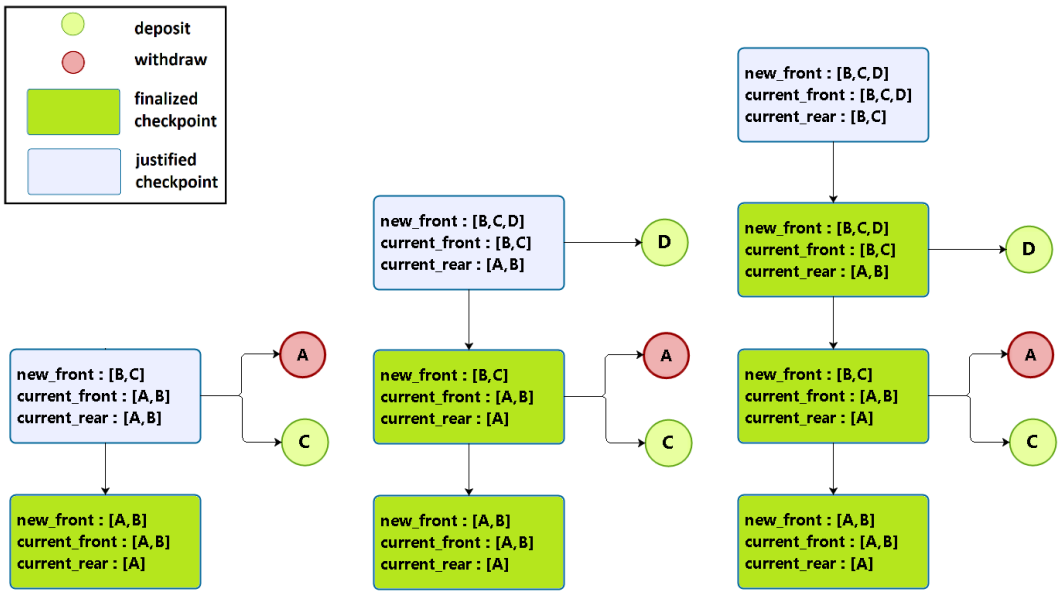
Current_front: Αποτελεί το υπάρχον front_validator_set που περιέχει τους validators από τους οποίους μπορεί να δεχθεί ψήφο το συγκεκριμένο block.

Current_rear: Αποτελεί το υπάρχον rear_validator_set που περιέχει τους validators από τους οποίους μπορεί να δεχθεί ψήφο το συγκεκριμένο block.

Έτσι τα deposits και withdraws προκαλούν προσθέσεις και αφαιρέσεις validators (αντίστοιχα) μόνο στο New_front set. Ενώ η οριστικοποίηση του προηγούμενου checkpoint block επιφέρει την μετακύλιση των set ως εξής:

New_front (checkpoint-πατέρα) → Current_front (checkpoint-παιδιού) → Current_rear (checkpoint-παιδιού) → Exits

Στην εικόνα 4.4 παρουσιάζεται ένα παράδειγμα αυτής της διαδικασίας.



Εικόνα 4. 4 Υλοποίηση δυναμικών validator sets με τρεις αποθηκευμένες δομές.

Αποθηκεύοντας, λοιπόν τις τρεις αυτές δομές σε κάθε block, μπορούμε να περιγράψουμε την τωρινή και την επομένη κατάσταση των validator sets σε κάθε ύψος του blockchain.

Σύμφωνα με τα παραπάνω η διαδικασία κατασκευής νέων block είναι η ακόλουθη: Κάθε νέο Block κατά τη δημιουργία του, κληρονομεί αυτούσιες τις δομές του Block πατέρα του. Αυτό γίνεται μέσω της συνάρτησης prepare_block της κλάσης Node. Το δημιουργηθέν αντικείμενο δίνεται στην create_block της κλάσης Network, η οποία αρχικά δημιουργεί την καταχώρησή του στη βάση δεδομένων γράφου Neo4j. Στη συνέχεια διατρέχοντας την λίστα node_transactions με της αποθηκευμένες συναλλαγές, ελέγχουμε για μηνύματα κατάθεσης ή ανάληψης από την περιγραφή τους. Σε περίπτωση κατάθεσης καταχωρούμε τον νέο κόμβο και το stake του στο new_front. Αντίστοιχα, σε περίπτωση ανάληψης αφαιρούμε τον validator από το new_front. Σημειώνεται ότι κατά την παραπάνω διαδικασία, ελέγχουμε ότι ο κόμβος που αποστέλλει μήνυμα κατάθεσης δεν έχει υπάρξει validator στο παρελθόν (δεν υπάρχει προηγούμενο μήνυμα ανάθεσης από τον κόμβο) και διαθέτει το αντίστοιχο stake. Οι παρακάτω γραμμές κώδικα της συνάρτησης «create_block» είναι υπεύθυνες για τον έλεγχο των αποθηκευμένων μηνυμάτων και την κατάλληλη τροποποίηση του new_front set σε κάθε περίπτωση

```
#Validator Deposits => Add him to the front validator set of the new block
if d['description'] == "Validator Deposit Message":
   #Validator is not in the new front and hasn't sent a withdraw message yet
   if d['from'] not in new_front and self.get_withdraw(d['from'])==0:
        new_front[d['from']] = d['amount']
        appended_message = self.create_message(d['from'],d['amount'])
        appended_message.relationships.create(d['from'], appended_block)
   # Validator Withdraws => Remove him from the front validator set of the
   new block (He remains in the rear validator set until the next epoch)
   elif d['description'] == "Validator Withdraw Message" :
        if d['from'] in new_front:
            del new_front[d['from']]
            appended_message = self.create_message(d['from'])
            appended_message.relationships.create(d['from'], appended_block)
```

Η μετακύλιση των set πραγματοποιείται στην περίπτωση που το προηγούμενο checkpoint οριστικοποιηθεί. Τότε το new_front του checkpoint-πατέρα γίνεται το current_front του checkpoint-παιδιού, ενώ το current_front θα αποτελέσει το current_rear του νέου block. Αφού ολοκληρωθεί η διαδικασία το ενημερωμένο πλέον Block αποστέλλεται στους υπόλοιπους κόμβους για αποδοχή και αποθήκευση.

```
newblock['current_rear'] = dict(newblock['current_front'])
newblock['current_front'] =
dict(self.my_validator.get_new_front(newblock['previousCheckpoint']))
```

Όπου η «get_new_front» βρίσκει ουσιαστικά το previousCheckpoint Block από τη δομή validation_blocks της κλάσης Validator, και ζητά το new_front set του.

Τελικά γίνεται εμφανές, ότι με τις παραπάνω δομές μπορούμε και υλοποιούμε τα δυναμικά validator sets για κάθε νέο block, αξιοποιώντας την αποθηκευμένη πληροφορία σε δύο μόνο block. Το ένα είναι το block-πατέρας που αποτελεί το επιλεγμένο branch και αποθηκεύεται στη κλάση Node. Το δεύτερο είναι το προηγούμενο checkpoint block, το οποίο αποθηκεύεται μαζί με τα υπόλοιπα checkpoints στη δομή validation_blocks της κλάσης Validator και χρησιμοποιείται σε περίπτωση οριστικοποίησης του.

4.4.2 Υλοποίηση διαδικασίας ψηφοφορίας

Η κατασκευή των validator set ως αποθηκευμένες δομές στα Blocks, φανερώνει ότι ένα block έχει τη δυνατότητα να αποθηκεύει πληροφορίες που απεικονίζουν την αποθηκευμένη πληροφορία σε όλη την αλυσίδα του. Συγκεκριμένα, κάθε block γνωρίζει τους validators του, χωρίς να είναι απαραίτητη η σειριακή προσπέλαση της αλυσίδας για την εύρεση αποθηκευμένων μηνυμάτων κατάθεσης και ανάληψης.

Χρησιμοποιώντας πάλι το αντικείμενο του block, μπορούμε να υλοποιήσουμε και τη διαδικασία ψηφοφορίας των validators στο blockchain. Κάθε validators έχει τη δυνατότητα να γίνει ψηφίσει αποστέλλοντας ένα http αίτημα στον τοπικό Python Flask Server. Το αίτημα εξυπηρετείται από την κλάση «submit_vote». Ο κόμβος αποστέλλει τις παρακάτω τέσσερις πληροφορίες: source_hash, target_hash, source_height, target_height. Η κλάση εξυπηρέτησης, της οποίας ο κώδικας φαίνεται παρακάτω, επεξεργάζεται ελέγχει την ψήφο ως προς την εγκυρότητά της και την μετατρέπει σε συναλλαγή με περιγραφή «Validator Vote» και κοινοποιείται σαν τέτοια στο δίκτυο. Οι κόμβοι λαμβάνουν την ψήφο και την αποθηκεύουν στο node_transactions, όπως κάνουν και για μία κανονική συναλλαγή.

```
#A validator broadcasts a vote to the rest of the nodes, after the vote is checked to be correct
class submit_vote(Resource):
   def post(self):
      vote = request.get_json()
      my_blockchain.check_vote(vote)
      vote['from'] = my_node.hex_name
      vote['description'] = "Validator Vote"
      my_blockchain.broadcast_send_nodes(vote, '/gettransaction')
```

Η ψήφος κάθε validator αντιπροσωπεύει ένα link {source → target}. Αν το link έχει ψηφιστεί από ένα σύνολο front validators, οι οποίο κατέχουν τουλάχιστον τα 2/3 του συνολικού stake του current_front set, μιλάμε για front supermajority link. Αντίστοιχα αν οι validators ανήκουν στο current_rear και κατέχουν τουλάχιστον τα 2/3 του συνολικού stake αυτού, αναφερόμαστε σε rear supermajority link. Αν ένα link έχει ταυτόχρονα front supermajority και rear supermajority θα λέμε ότι πρόκειται για supermajority link. Όπως αναλύθηκε στην παράγραφο 2.5, το justification ενός checkpoint X πραγματοποιείται αν υπάρχει supermajority link {Y → X}, όπου Y checkpoint πρόγονος του X. Η διαδικασία οριστικοποίησης ενός checkpoint είναι λίγο πιο σύνθετη, ώστε να αποτρέπονται παθολογικά σενάρια οριστικοποίησης δύο blocks σε διαφορετικές αλυσίδες. Συγκεκριμένα, για να γίνει το checkpoint X finalized, θα πρέπει:

1.Το X να είναι justified
2. Να υπάρχει supermajority link {X → Y}, όπου Y block απόγονος του X
3. Οι ψήφοι για finalization και justification του X να είναι αποθηκευμένες στα block της αλυσίδας του X, πριν τη δημιουργία του παιδιού checkpoint του X

Για λόγους απλότητας, επιλέξαμε οι ψήφοι οριστικοποίησης ενός checkpoint να δίνονται με target το block-παιδί του checkpoint, δηλαδή το block σε ύψος block_height(X) + 1. Αυτό είναι χρήσιμο για δύο λόγους. Πρώτον, τα χρονικά περιθώρια της ψηφοφορίας είναι μεγαλύτερα, αφού οι validators από τη δημιουργία μόλις του πρώτου παιδιού ενός checkpoint (μέχρι τη δημιουργία του επόμενου checkpoint) μπορούν να ψηφίζουν για την οριστικοποίησή του. Δεύτερον περιορίζει το πιθανά targets των ψήφων οριστικοποίησης ενός checkpoint, κάτι που διευκολύνει τον έλεγχο εγκυρότητας των ψήφων, αλλά και μετέπειτα τον έλεγχο για finalization.

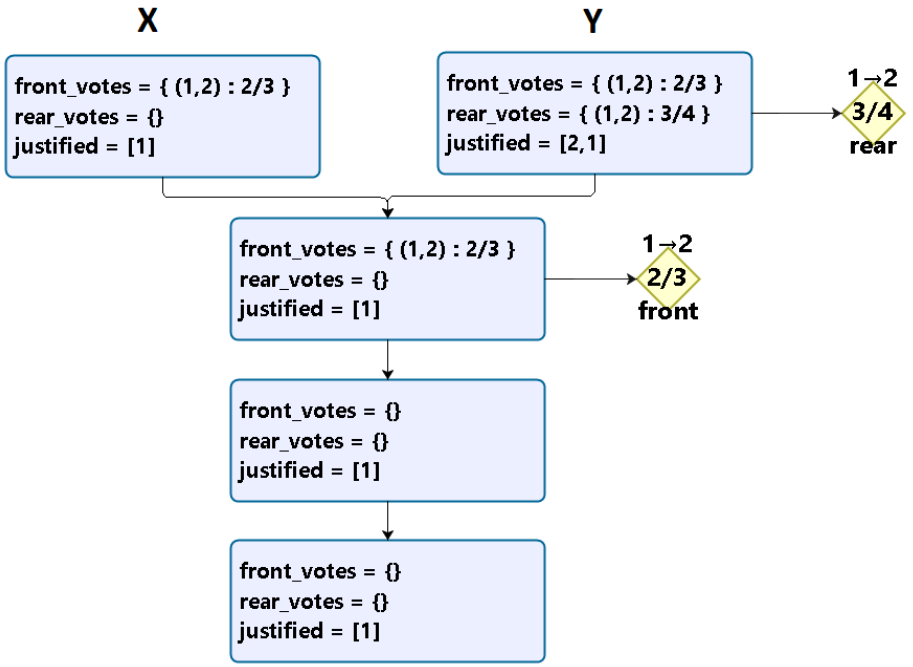
Όσον αφορά την εγκυρότητα των ψήφων αυτός γίνεται με την συνάρτηση «check_vote». Για να θεωρηθεί έγκυρη μία ψήφος θα πρέπει να ικανοποιεί τα εξής κριτήρια: Τα δοσμένα hashes πρέπει να αντιστοιχούν σε blocks αποθηκευμένα στη δομή validation_blocks. Στη validation_blocks αποθηκεύονται ουσιαστικά τα checkpoints και τα άμεσα παιδία αυτών, αφού αυτά είναι τα blocks που μπορούν να δεχθούν ψήφο. Τέλος αν πρόκειται για ψήφο justification (το target block είναι checkpoint)διασφαλίζουμε μέσω της δομής parent της κλάσης Node, ότι το target checkpoint είναι απόγονος του source checkpoint. Αντίθετα αν είναι ψήφος finalization ελέγχουμε το target block να έχει previous_hash = source_hash. Ο κώδικας της «check_vote» είναι ο παρακάτω:

```
#checks the validity of a given vote
    def check_vote(self,vote):
        #source should be in validation_blocks
        assert self.my_validator.confirm_validation_block(vote['source_hash'], vote['source_height']), "source
hash not found"
        #target should be in validation blocks
         assert self.my_validator.confirm_validation_block(vote['target_hash'], vote['target_height']), "target
hash not found"
         #Source and target must be on the same subchain
         #if target is a checkpoint (justification vote) we check this through the «parent» stucture
         if vote['target_height']%EPOCH_SIZE == 0:
                assert self.my_node.is_ancestor(vote['target_hash'],vote['source_hash']), "source not an
ansestor of target"
        #if target is a block (finalization vote) source checkpoint should be its parent
          else:
                assert self.my_validator.validation_blocks[vote['target_hash']].get_previous_hash() ==
vote['source_hash'], "target not a child of source"
```

Κατά την κατασκευή ενός νέου μπλοκ και την προσπέλαση της λίστας συναλλαγών από την «create_block», εντοπίζουμε τις αποσταλμένες ψήφους των validators. Για να θεωρείται η ψήφος έγκυρη θα πρέπει ο validator να βρίσκεται είτε στο current_front είτε στο current_rear set του target checkpoint της ψήφου του. Επιπλέον από τα κριτήρια finalization και justification, προκύπτει ότι το κάθε block αποδέχεται ψήφους με target-blocks (άρα και source) της αλυσίδας του. Αν η ψήφος ικανοποιεί τα παραπάνω, αποθηκεύουμε το αντίστοιχο link-ψήφο {source → target} με ποσοστό ψήφου ίση με το ποσοστό του validator στο εκάστοτε set. Το ποσοστό αυτό υπολογίζεται ως validator_stake/total_stake.

Αν κάποιος άλλος validator ψηφίσει ένα υπάρχον link, το ποσοστό της ψήφου του προστίθεται σε αυτά των υπόλοιπων. Προφανώς πρέπει ξεχωρίζουμε τις ψήφους των front validators από αυτές των rear validators σε δύο διαφορετικές δομές. Αν τα ποσοστά και των δύο set φτάσουν τα 2/3, τότε το block γίνεται justified ή finalized ανάλογα με το είδος των ψήφων. Πρέπει να γίνει αντιληπτό ότι αυτές οι έννοιες δεν είναι καθολικές για όλο το blockchain, αλλά αποτελούν την εικόνα που έχει ένα block για την αλυσίδα στην οποία ανήκει. Δηλαδή, αν σε μία αλυσίδα φτάσουν οι ψήφοι για το justification του Block X, τότε για την αλυσίδα αυτή το X θεωρείται οριστικοποιημένο. Αντίθετα για μία διαφορετική αλυσίδα, που δεν περιλαμβάνει τις σχετικές ψήφους, το μπλοκ δεν έχει οριστικοποιηθεί.

Όπως και για τα δυναμικά validator sets, έτσι και εδώ θα χρησιμοποιήσουμε τρεις δομές αποθηκευμένες στα Block. Οι δύο από αυτές είναι τα dictionaries front_votes και rear_votes, στις οποίες θα καταχωρούμε τις ψήφους-link των validators που βρίσκονται αποθηκευμένες στην παρούσα αλυσίδα, με τα αντίστοιχα ποσοστά. Θα έχουν δηλαδή τη μορφή front_votes = { (source1,target1) : stake1 , (source2,target2) : stake2, … }. Η άλλη είναι η λίστα justified που αποθηκεύει τα justified checkpoints της αλυσίδας, δηλαδή justified = [checkpoint1, checkpoint2, ...]. Στις δομές αυτές, για τις καταχωρήσεις των checkpoints ή blocks μπορούν να χρησιμοποιηθούν τα ύψη αυτών αντί των hashes, καθώς η αλυσίδα που «βλέπει» το κάθε block έχει μοναδικό block σε κάθε ύψος. Στο παράδειγμα της εικόνας 4.5, βλέπουμε πως διαμορφώνονται οι τρεις δομές ανάλογα με τις αποθηκευμένες ψήφους.



Το block X δεν έλαβε την ψήφο του rear validator set γι' αυτό και σε αντίθεση με το block Y, δεν θεωρεί το checkpoint σε ύψος 2 justified

Εικόνα 4. 5 Υλοποίηση διαδικασίας ψηφοφορίας με τρεις αποθηκευμένες δομές.

Με τις παρακάτω γραμμές κώδικα, η συνάρτηση «create_block» ενημερώνει τις δομές front_votes, rear_votes και justified, ανάλογα με το είδος των ψήφων. Συγκεκριμένα, αν το transaction έχει περιγραφή «Validator Vote», ξεκινάμε βρίσκοντας το ποσοστό συμμετοχής του stake του validator σε κάθε set. Έπειτα, αν το target block της ψήφου ανήκει στην παρούσα αλυσίδα, αποδεχόμαστε την ψήφο-link και την εισάγουμε κατάλληλα στα dictionaries.Εν συνεχεία ελέγχουμε αν το απεσταλμένο link συμπλήρωσε front και rear supermajority, ώστε το target block να γίνει justified. Τέλος, δημιουργούμε την οντότητα της ψήφου στη βάση δεδομένων γράφου Neo4j.

```
elif d['description'] == "Validator Vote":
    (front_prc,rear_prc) =
     self.my_validator.get_validator_percentages(d['target_hash'], d['from'])
    #The vote's target-block needs to be in the new block's blockchain
    if self.same_branch(newblock['hash'], d['target_hash']) and (not front_prc == 0 or
    not rear_prc==0):
        vote_link = str((d['source_height'],d['target_height']))
        if vote_link in front_votes:
            front_votes[vote_link] += front_prc
        else:
            front_votes[vote_link] = front_prc
        if vote_link in rear_votes:
            rear_votes[vote_link] += rear_prc
        else:
            rear_votes[vote_link] = rear_prc
        #if the vote link has supermajority in both sets, and the target block is a
        checkpoint block => justify it
        if front_votes[vote_link]>2/3 and rear_votes[vote_link]>2/3 and
        d['target_height']%EPOCH_SIZE==0:
            justified_checkpoints.append(d['target_height'])
            self.broadcast_send_nodes({'hash': d['target_hash']},
            '/newhighestjustified')
            timestamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-
            %d %H:%M:%S')
            hash_object =
            hashlib.sha256((str(timestamp)+str(randint(1,1000000))).encode('utf-8'))
            vote_hash = hash_object.hexdigest()
            #create db object vote
            appended_vote = self.database.nodes.create(name="Vote", r_from=d['from'],
            source=d['source_hash'], source_height = d['source_height'],
            target=d['target_hash'], target_height=d['target_height'],
            front=front_prc, rear=rear_prc, hash=vote_hash)
            self.db_votes.add(appended_vote)
            appended_vote.relationships.create(d['from'], appended_block)
```

Η όλη διαδικασία της ψηφοφορίας που είδαμε, αποσκοπεί στην αποτελεσματική οριστικοποίηση checkpoints και στην επίτευξη της ασφάλειας μέσω της ικανοποίησης των κριτηρίων που έχουν τεθεί. Συγκεκριμένα, μπορούμε να διασφαλίσουμε το κριτήριο 1 από την λίστα justified και το κριτήριο 2 από τα dictionaries front_votes, rear_votes. Λόγω της φύσης των δομών αυτών, εξασφαλίζεται ταυτόχρονα, ότι οι καταχωρημένες ψήφοι βρίσκονται στην αλυσίδα του προς οριστικοποίηση checkpoint. Τέλος, επειδή υπάρχει η επιπλέον απαίτηση όλες οι ψήφοι να έχουν αποσταλεί πριν τη δημιουργία του επόμενου checkpoint, πρέπει κατά τη δημιουργία ενός νέου checkpoint να ελέγχεται αν υπάρχει ο απαιτούμενος αριθμός ψήφων για το finalization του προηγούμενου checkpoint. Αν οι ψήφοι υπάρχουν το προηγούμενο checkpoint γίνεται finalized, ενημερώνουμε τους κόμβους για την οριστικοποίηση και πραγματοποιούμε την μετακύλιση των validator sets στο παρόν checkpoint. Το κομμάτι κώδικα της «create_block» υπεύθυνο για τη λειτουργία αυτή είναι το παρακάτω:

```
#if newblock is a checkpoint block
if newblock['height']%EPOCH_SIZE == 0:

    prev_checkpoint_height = newblock['height'] - EPOCH_SIZE
    vote_link = str((prev_checkpoint_height, prev_checkpoint_height + 1))

    # check if the previous checkpoint became finalized ( => by this time all votes
    finalizing the previous Checkpoint should have arrived)
    if vote_link in front_votes and vote_link in rear_votes:
        if front_votes[vote_link] > 2 / 3 and rear_votes[vote_link] > 2 / 3 and
        prev_checkpoint_height in justified_checkpoints and prev_checkpoint_height!=0:
            self.get_block(newblock['previousCheckpoint'])['finalized'] = 'yes'
            #rotation of validator sets
            newblock['current_rear'] = dict(newblock['current_front'])
            newblock['current_front'] =
            dict(self.my_validator.get_new_front(newblock['previousCheckpoint']))
            #inform nodes about the finalization
            self.broadcast_send_nodes({'hash':
            newblock['previousCheckpoint']},'/newhighestfinalized')

        #label newblock as a checkpoint block in Neo4j
        self.create_block_relationship(appended_block, newblock['previousCheckpoint'],
        'PREV_CECKPOINT')
        self.db_checkpoints.add(appended_block)
```

Στην περίπτωση που το παρών block είναι checkpoint-block, δηλαδή block_height%EPOCH_SIZE = 0, ελέγχουμε την ύπαρξη ψήφων οριστικοποίησης στις δομές front_votes κα rear_votes. Η ψήφος οριστικοποίησης θα έχει τη μορφή (source,target) = (previous_checkpoint_height, previous_checkpoint_height + 1). Αν επιπλέον το παραπάνω link έχει λάβει front και rear supermajority προχωράμε στην οριστικοποίηση του previous_checkpoint, στην ενημέρωση των κόμβων μέσω http αιτήματος και στην μετακύλιση των validator sets.

Όταν οι υπόλοιποι κόμβοι του δικτύου λάβουν το αίτημα για νέο finalized checkpoint, ελέγχουν αν είναι σε μεγαλύτερο ύψος από το υπάρχον highest_finalized_checkpoint, ώστε να το αντικαταστήσουν. Κάθε Blockchain ξεκινάει με το Genesis Block ως highest_finalized_checkpoint. Ακριβώς ίδια είναι και η διαδικασία αποδοχής νέων justified checkpoints, με την επιπλέον λειτουργία της επιλογής branch. Αυτό συμβαίνει καθώς το fork choice rule επιβάλλει στους κόμβους να χτίζουν στην αλυσίδα που ανήκει το highest_justifed_checkpoint, επομένως μία αλλαγή στο τελευταίο μπορεί να επιφέρει και αλλαγή στο επιλεγμένο branch. Παρουσιάζουμε ενδεικτικά τον κώδικα υπεύθυνο για τη διαδικασία ελέγχου και αποδοχής finalized checkpoint σε μέγιστο ύψος, με την αντίστοιχη διαδικασία για τα justified checkpoints να είναι ακριβώς η ίδια:

```
#given either a block object or a block hash check it for being the highest
justified
def check_highest_finalized(self, block):
    if isinstance(block,str):
        if block in self.validation_blocks:
            block = self.validation_blocks[block]
        else:
            return False
    if self.highest_finalized == None :
        self.highest_finalized = block
        return True
    elif self.highest_finalized.get_height() < block.get_height() :
        self.highest_finalized = block
        return True
    return False
```

4.4.3 Ποινές και επιβραβεύσεις των Validators

Η σωστή λειτουργία του Casper, απαιτεί από τους validator να δρουν προς όφελος του δικτύου. Αυτό επιτυγχάνεται με την χρηματική επιβράβευση εκείνων που συμβάλλουν θετικά, τηρώντας τους θεσπισμένους κανόνες και την επιβολή χρηματικών ποινών στους παραβάτες. Η επιβολή της ποινής είναι και αυτή που αναγκάζει τους validators να επιλέγουν μία μοναδική κανονική αλυσίδα, έτσι ώστε να επιτυγχάνεται η συναίνεση.

Η ποινή αυτή ονομάζεται slashing και πρόκειται για την συνολική διαγραφή του stake ενός validator και συνεπώς την έξοδό του από τα validator sets. Το slashing επιβάλλεται στους validators που θα παραβούν κάποιον απ' τους κανόνες που ισχύουν για τις ψήφους που αποστέλλουν. Δηλαδή αν κάποιος validator αποστείλει δύο διαφορετικές ψήφους με target στο ίδιο ύψος, η ψήφους, όπου το διάστημα των υψών της μίας περιέχεται στο (ή περιέχει το) διάστημα υψών της άλλης. Με τους παραπάνω κανόνες αποτρέπουμε την ύπαρξη οριστικοποιημένων blocks σε διαφορετικές αλυσίδες. Επιπλέον, σε περίπτωση που προκύψουν conflicting finalized blocks στο δέντρο του blockchain, μπορούμε να αποδείξουμε την παραβίαση των κανόνων, να εντοπίσουμε και να τιμωρήσουμε τους παραβάτες.

Για τον εντοπισμό των παραβατών, εισάγουμε ένα σύστημα validator reporting στο δίκτυο, στο οποίο οι κόμβοι μπορούν να αναφέρουν παραβάτες validators. Αν η αναφορά επιφέρει την διαγραφή ενός validator ο αποστολέας κόμβος θα λάβει χρηματική επιβράβευση για την συνεισφορά του. Η διαδικασία αναφοράς ξεκινά με τον κόμβο να αποστέλλει μέσω http αιτήματος τα hashes δύο παράνομων ψήφων ενός validator, προς τον τοπικό Python Flask Server. Η κλάση η οποία εξυπηρετεί το αίτημα αυτό είναι η «submit_report» και παρουσιάζεται παρακάτω:

```
class submit_report(Resource):
    def post(self):
        report = request.get_json()
        #target and source doesn't have a literal meaning here
        if report['source_hash']!=report['target_hash']:
            report['from'] = my_node.hex_name
            report['description'] = "Validator Report"
            my_blockchain.broadcast_send_nodes(report,
            '/gettransaction')
```

Αφού ελεγχθεί, ότι τα δύο απεσταλμένα hashes είναι διαφορετικά, η «submit_report» δημιουργεί ένα μήνυμα συναλλαγής με αυτά και την ειδική περιγραφή «Validator Report». Μετά την κατασκευή της, η συναλλαγή-report κοινοποιείται στο δίκτυο και αποθηκεύεται από τους κόμβους στη λίστα node_transactions της κλάσης Network.

Διατρέχοντας τη αυτή λίστα, ο miner του επόμενου block θα εντοπίσει τις απεσταλμένες αναφορές από την μοναδική περιγραφή τους. Στη συνέχεια, από τα δοσμένα hashes θα εντοπίσει τις ψήφους στην βάση δεδομένων γράφου Neo4j και θα ζητήσει τα source height και target height και τον αντίστοιχο validator για κάθε μία από αυτές. Έχοντας τη διάταξη των παραπάνω υψών, μπορεί εύκολα να ελέγξει αν υπάρχει παράβαση των κανόνων. Απαραίτητη προϋπόθεση είναι ο ίδιος validator να έχει αποστείλει και τις δύο ψήφους και βρίσκεται σε κάποιο current validator set. Στην περίπτωση αυτή ο miner δημιουργεί μήνυμα «Slash» στην Neo4j και αφαιρεί τον validator από τα current_front και current_rear validator sets. Η επιβράβευση του αποστολέα κόμβου γίνεται με τη δημιουργία ενός reward transaction, το οποίο θα περιέχεται στο παρόν block.

Ωστόσο πρέπει να λάβουμε υπόψιν μας και την περίπτωση όπου ένας validator παραβίασε τους κανόνες και δεν εντοπίστηκε μέχρι που αποχώρησε και από τα δύο current sets. Για το σκοπό αυτό χρησιμοποιούμε την λίστα slashed_validators την οποία αποθηκεύουμε σε κάθε block. Σε αυτήν καταχωρούμε τους validators που έχουν διαγραφεί στην παρούσα αλυσίδα είτε κατά τη διάρκεια, είτε μετά το τέλος της θητείας τους. Έτσι μετά την λήξη της περιόδου withdrawal delay, μπορούμε να ακυρώσουμε την ανάληψη αν ο validator ανήκει στη λίστα slashed_validators του παρόντος block. Επιπλέον σε περίπτωση πολλαπλών reports για τον ίδιο validator, η ανταμοιβή πρέπει να δίνεται μόνο σε έναν από τους κόμβους που απέστειλαν αναφορά. Τώρα η πρώτη αναφορά που συμπεριλαμβάνεται σε μία αλυσίδα τοποθετεί τον παραβάτη στην λίστα slashed_validators, οπότε κάθε επόμενη αναφορά για αυτόν τον validator στη συγκεκριμένη αλυσίδα αγνοείται. Να σημειωθεί ότι η αφαίρεση των validators από τα new_front, current_front, current_rear πραγματοποιείται στο τέλος της create_block, ώστε να λάβει χώρα μετά την πιθανή μετακύλιση των set σε περίπτωση οριστικοποίησης του προηγούμενου checkpoint.

Η όλη διαδικασία συνοψίζεται στις παρακάτω γραμμές κώδικα:

```
#Validator Report given 2 vote hashes , confirm or disconfirm that the voter
should be slashed
elif d['description'] == "Validator Report":
    (source1,target1,valid1) = self.get_vote(d['source_hash'])
    (source2,target2,valid2) = self.get_vote(d['target_hash'])
    if valid1 not in slashed_validators and valid1==valid2:
        #if two separate votes have the same target or one is within the
        span of the other, then the validator gets slashed
        if target1==target2 or source1<source2<target2<target1 or
        source2<source1<target1<target2:
            slashed_validators.append(valid1)
            #slashing message in Neo4j
            appended_message = self.database.nodes.create(name="Slash",
            user=valid1,vote_hash1=d['source_hash'],
            vote_hash2=['target_hash'])
            self.db_messages.add(appended_message)
            appended_message.relationships.create(d['from'], appended_block)
            #reward transaction for the sender
            appended_transaction = self.create_transaction('reward_tx',
            d['from'], 10, "Validator Reporting Reward")
            appended_transaction.relationships.create(d['from'],
            appended_block)
```

Με το σύστημα αυτό παρέχουμε χρηματικά κίνητρα στους κόμβους για την τιμωρία των κακόβουλων validators. Έτσι ακόμη και ένας κόμβος να είναι ειλικρινής στο δίκτυο, αυτός αρκεί ώστε να τιμωρήσει τους παραβάτες. Επιπλέον, στην περίπτωση ύπαρξης πολλαπλών branches, περιμένουμε ότι οι κόμβοι θα αποστείλουν αναφορές σε κάθε υποαλυσίδα με την προοπτική της επιβράβευσης σε περίπτωση που το συγκεκριμένο branch οριστικοποιηθεί.

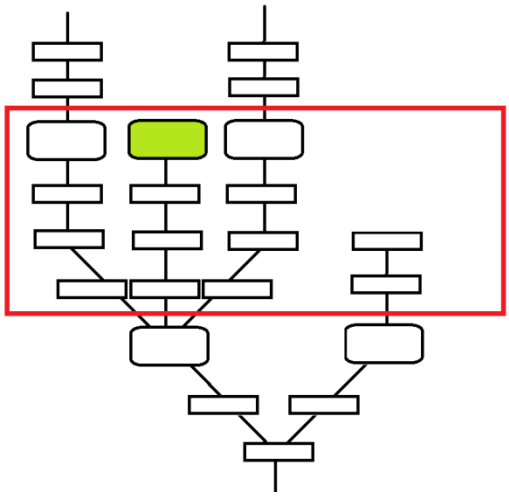
Τέλος, το Casper πρέπει να επιβραβεύει τους validators που ενεργούν θετικά στο δίκτυο. Εμείς στο πρόγραμμά μας επιλέξαμε το reward να δίνεται στους validators με κάθε καινούργιο checkpoint. Το κριτήριο είναι αν ο validator ψήφισε οποιοδήποτε block από τη στιγμή δημιουργίας του προηγούμενου checkpoint μέχρι τώρα. Δηλαδή επιβραβεύουμε την δραστηριότητα των validators και όχι την επίτευξη συναίνεσης απαραίτητα. Καθώς αν το κίνητρο ήταν η συναίνεση, στην περίπτωση που μία αλυσίδα έχει ψηφιστεί από το 50% των validators ενός set (οι οποίοι μπορεί να είναι και κακόβουλοι), η προοπτική επιβράβευσης δελεάζει τους ειλικρινής validators να ψηφίσουν την πιο δημοφιλή αλυσίδα. Η επιβράβευση στη δική μας εφαρμογή είναι ανάλογη με το stake του κάθε validator καθώς και η σημαντικότητα κάθε ψήφου είναι ανάλογη με αυτό.

Αντίστοιχα, θέλουμε να τιμωρούμε τους ανενεργούς validators καθώς η απουσία τους δυσκολεύει τη συγκέντρωση του απαραίτητου ποσοστού ψήφων και επομένως τη συναίνεση. Επιπλέον στην ακραία περίπτωση όπου πάνω από το 1/3 των validators είναι ανενεργοί το consensus καθίσταται αδύνατο. Η ποινή λοιπόν για αυτούς τους validators ονομάζεται Inactivity Leak και σκοπό έχει τη σταδιακή μείωση των stake των ανενεργών validators μέχρι και την ολοκληρωτική έξοδό τους από τα validator sets. Όπως και η επιβράβευση έτσι και το Inactivity Leak είναι ανάλογο του stake του validator, αφού όσο μεγαλύτερο το stake που απουσιάζει τόσο δυσκολότερη η επίτευξη συναίνεσης στο δίκτυο, άρα και πιο επιτακτική η ανάγκη επαναλειτουργίας των validator sets.

Για την εύρεση όσον ψήφισαν ο miner του νέου checkpoint, κοιτά τις καταχωρημένες ψήφους στα blocks όλων των αλυσίδων με ύψος στο διάστημα (New_Checkpoints_height – EPOCH_SIZE , New_Checkpoints_height] και αυτό καθώς θέλουμε να επιβραβεύουμε όλους του ενεργούς validators ανεξάρτητα από την αλυσίδα την οποία ψηφίζουν. Η εύρεση των validators αυτών γίνεται με την παρακάτω συνάρτηση :

```
def get_voters(self,height):
    min = height - EPOCH_SIZE
    max = height
    has_voted=[]
    this_epoch_voted = "MATCH (n:Blocks) WHERE n.height > " + str(min) +
    " and n.height <= " + str(max) + "OPTIONAL MATCH(n:Blocks)-[x]-(v:Votes)
    RETURN v.r_from"
    results = self.database.query(this_epoch_voted, returns=(str))
    for r in results:
        has_voted.append(r[0])
        print(r[0])
    return has_voted
```

Όπου λαμβάνει το ύψος «height» του τωρινού checkpoint και αναζητά τις ψήφους που έχουν καταχωρηθεί σε Blocks με ύψος: previous_checkpoint_height < ύψος ≤ height. Για τις ψήφους αυτές επιστρέφει του αποστολείς validators σε μία λίστα has_voted. Έτσι στο παράδειγμα τις εικόνας 4.6 ο miner του καινούργιου checkpoint (πράσινο χρώμα) αναζητά ψήφους σε όλη την σημειωμένη (με κόκκινο χρώμα) ζώνη.



Εικόνα 4. 6 Παράδειγμα αναζήτησης ψηφοφόρων στο Blockchain Tree.

Αντίστοιχα, η ποινή αποδίδεται σε όσους δεν έχουν καταθέσει ψήφο στην παραπάνω ζώνη οπότε και θεωρούνται inactives. Να σημειωθεί ότι σε ένα πραγματικό δίκτυο Blockchain το EPOCH_SIZE είναι πολύ μεγαλύτερο, επομένως και δίνεται ο απαραίτητος χρόνος στους validators να καταθέσουν κάποια ψήφο. Το διάστημα αυτό επιλέχθηκε από εμάς, με τη λογική ότι οι ψήφοι οριστικοποίησης για οποιοδήποτε checkpoint στο ίδιο ύψος , θα βρίσκονται σίγουρα στην παραπάνω ζώνη. Δηλαδή επιβραβεύουμε την προσπάθεια για συναίνεση και όχι απαραίτητα την επίτευξή της Τέλος τονίζουμε ότι οι επιβραβεύσεις και οι ποινές εφαρμόζονται στα stake των sets του παρόντος Block, δηλαδή ανά αλυσίδα.

Η όλη λειτουργία του Casper συνοψίζεται στη λειτουργία της «create_block», η οποία πραγματοποιεί:

1. Τις προσθαφαιρέσεις validators στα sets ανάλογα με τις καταθέσεις και τις αναλήψεις.
2. Την αποθήκευση και καταμέτρηση των απεσταλμένων ψήφων
3. Το justification και το finalization των checkpoints ανάλογα με τις ψήφους
4. Την μετακύλιση των validators sets σε περίπτωση finalization
5. Τον έλεγχο των reports και το slashing των validators, όποτε κρίνεται απαραίτητο
6. Την επιβράβευση και την τιμωρία των ενεργών και ανενεργών validators αντίστοιχα

5
Επίδειξη λειτουργικότητας εφαρμογής

Σε αυτό το κεφάλαιο γίνεται παρουσίαση του τρόπου λειτουργίας της εφαρμογής blockchain που κατασκευάσαμε και ιδιαίτερα του μηχανισμού Casper, που αποτελεί το κεντρικό αντικείμενο της παρούσας διπλωματικής. Κατά την παρουσίαση, θα δείξουμε τις διάφορες ενέργειες που εκτελούν οι κόμβοι του δικτύου και το αποτέλεσμα αυτών, που αποτυπώνεται τόσο στις τοπικά αποθηκευμένες δομές και blocks, όσο και στην βάση δεδομένων γράφου Neo4j.

5.1 Αρχικοποίηση κόμβων στο blockchain δίκτυο

Η εκκίνηση της εφαρμογής, σηματοδοτείται από την αρχικοποίηση του πρώτου κόμβου στο δίκτυο. Όπως έχουμε αναφέρει, στο δίκτυο μας προσομοιώνουμε τους διαφορετικούς χρήστες, σαν Flask Servers που τρέχουν σε διαφορετικές πόρτες στο ίδιο «μηχάνημα».

Η παρακάτω εντολή της cmd (γραμμή εντολών) αρχικοποιεί έναν κόμβο στο δίκτυό μας, ο οποίος ακούει στη Python Flask πόρτα 5000:

```
py start.py -p 5000
```

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

```
C:\Users\Konstantinos\Desktop\Casperfin>py start.py -p 5000
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

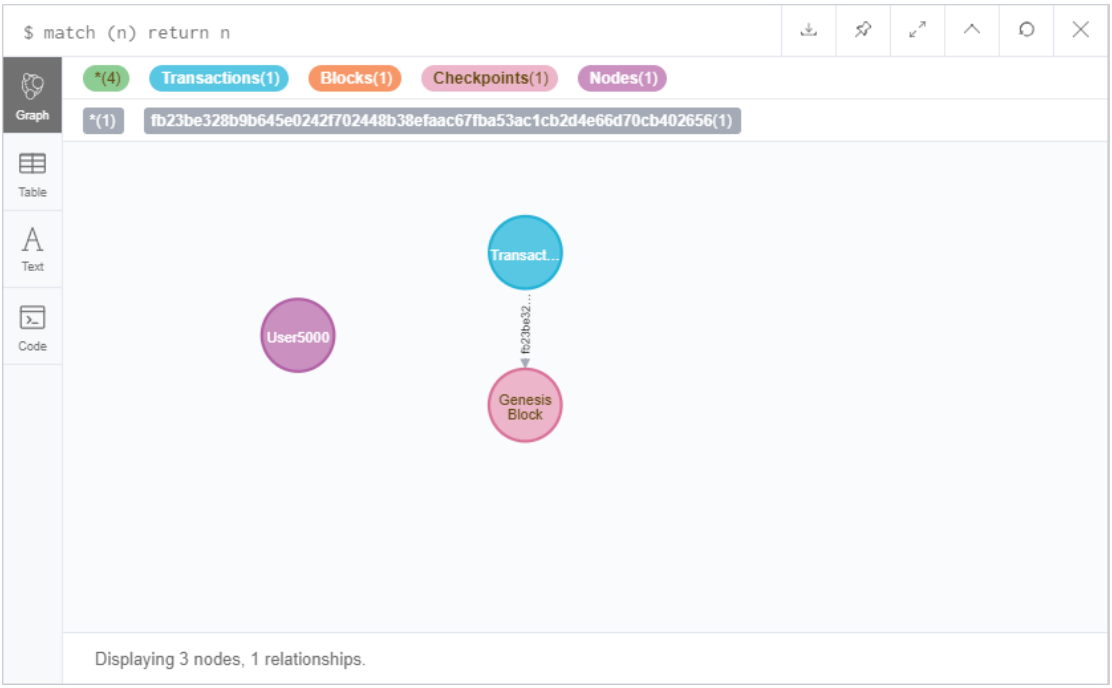
Εικόνα 5. 1 Αρχικοποίηση κόμβου από τη γραμμή εντολών

Όπως γνωρίζουμε, η αρχικοποίηση του πρώτου κόμβου στο δίκτυο, σημαίνει και την κατασκευή του Genesis Block από αυτόν. Η συναλλαγή που περιέχεται σε αυτό το Block, είναι μία μεγάλη ανταμοιβή για τον κατασκευαστή κόμβο, που στο δίκτυο μας είναι ορισμένη στα 2000 tokens.

Ας δούμε τώρα την αποτύπωση αυτών των ενεργειών στη βάση δεδομένων Neo4j. Μέσω του monitoring tool που προσφέρει η Nep4j, το Neo4j Browser εκτελούμε το παρακάτω Chyper Query με το οποίο ζητάμε όλη την αποθηκευμένη πληροφορία στη βάση:

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

Η εικόνα που μας επιστρέφει το Neo4j Browser Είναι η ακόλουθη:



Εικόνα 5. 2 Αναπαράσταση αρχικής κατάστασης Blockchain στη Neo4j

Στην Εικόνα 5.2 παρατηρούμε τρεις κόμβους:

- Ο κόμβος User5000 φανερώνει την ύπαρξη κόμβου στο δίκτυο, που ακούει στην πόρτα 5000, κάτι που είναι χρήσιμο για την επικοινωνία με τους επόμενους κόμβους του δικτύου.
- Ο κόμβος Genesis Block που αναπαριστά το πρώτο block του blockchain. Το Genesis Block έχει διπλή ετικέτα, αφού είναι ταυτόχρονα block και checkpoint.
- Ο κόμβος Transaction αποτελεί την πρώτη συναλλαγή του blockchain και συγκεκριμένα την συναλλαγή ανταμοιβής του User5000, που κατασκεύασε το Genesis Block.

Επιλέγοντας έναν κόμβο στο γραφικό περιβάλλον της Neo4j, ο χρήστης μπορεί να δει τις ιδιότητες αυτού ως ζεύγος key-value. Οι ιδιότητες καθενός από τους παραπάνω κόμβους είναι οι ακόλουθες:

Εικόνα 5. 3 Ιδιότητες των κόμβων του Blockchain στη Neo4j

Αντίστοιχα με το κόμβο της πόρτας 5000, μπορούμε να αρχικοποιήσουμε και άλλους χρήστες σε διαφορετική Flask Server πόρτα. Με το παρακάτω batch script, αρχικοποιούμε έξι ακόμα κόμβους στις πόρτες 5001-5006 του Python Flask Server:

```
start /min cmd /c py start.py -p 5001
start /min cmd /c py start.py -p 5002
start /min cmd /c py start.py -p 5003
start /min cmd /c py start.py -p 5004
start /min cmd /c py start.py -p 5005
start /min cmd /c py start.py -p 5006
```

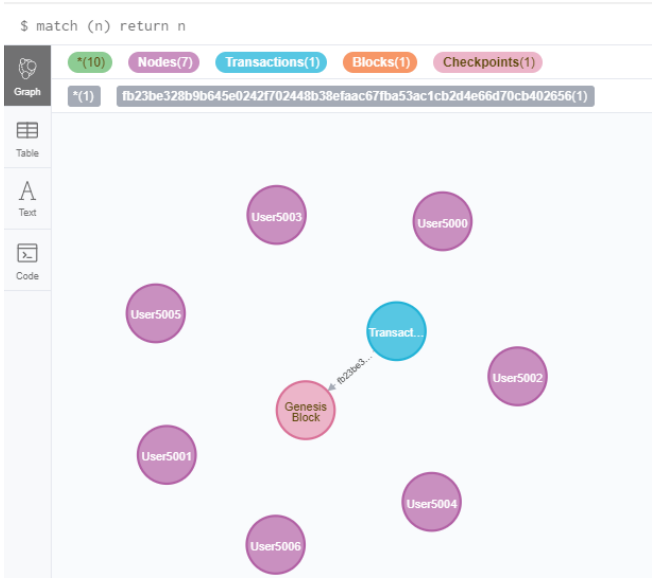
Εικόνα 5. 4 Αρχικοποίηση κόμβων μέσω batch script file

Όπως έχουμε πει, κάθε επόμενος κόμβος που εισέρχεται στο δίκτυο, εκτελεί τρεις ενέργειες:

1. Δημιουργεί την οντότητά του στη βάση δεδομένων γράφου Neo4j, ως κόμβο Node
2. Αναζητά του υπόλοιπους Nodes του δικτύου μέσω της βάσης Neo4j
3. Αποστέλλει σε αυτούς http αίτημα /getblockchain με το οποίο ζητά από αυτούς τις αποθηκευμένες δομές δεδομένων που περιγράφουν το υπάρχον blockchain και τους ενημερώνει (μέσω του header) για την εισαγωγή του στο δίκτυο (ώστε να μην απαιτείται κάθε φορά έλεγχος της βάσης δεδομένων γράφου, για την εύρεση των νεοεισαχθέντων κόμβων)

Ας δούμε πως αποτυπώνονται τώρα αυτές οι λειτουργίες στην εφαρμογή μας για τους έξι νεοεισαχθείς κόμβους.

Η πρώτη ενέργεια μπορεί να δειχθεί εκτελώντας το Chyper Query με το οποίο ζητάμε όλη την αποθηκευμένη πληροφορία στη βάση:



Εικόνα 5. 5 Εισαγωγή νέων κόμβων στη Neo4j

Η δεύτερη και η τρίτη εργασία αποτυπώνεται στα μηνύματα /getblockchain που λαμβάνει ο κάθε κόμβος. Συγκεκριμένα ο κόμβος User5000 θα λάβει αίτημα και από τους έξι κόμβους αφού προϋπάρχει στο δίκτυο. Αντίστοιχα ο κόμβος User5001 θα λάβει /getblockchain από τους κόμβους που εισέρχονται στο δίκτυο μετά από αυτόν, δηλαδή στην προκειμένη των User5002 – User5006 κ.ο.κ. Στην παρακάτω εικόνα φαίνονται τα μηνύματα που λαμβάνουν οι χρήστες στη γραμμή εντολών:

```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:19:20] "GET /getblockchain HTTP/1.1" 200 -
```

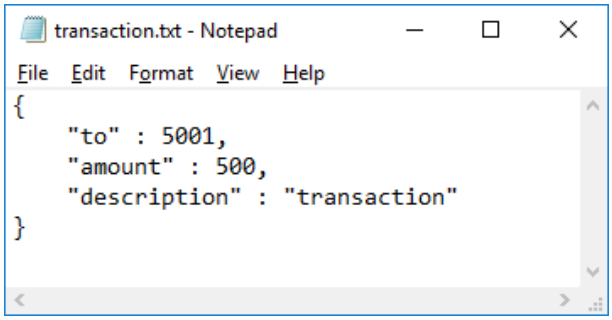
```
 * Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 20:44:50] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:44:51] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:44:51] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:44:53] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 20:44:53] "GET /getblockchain HTTP/1.1" 200 -
```

Εικόνα 5. 6 Αιτήματα /getblockchain που λαμβάνονται κατά την εισαγωγή νέων κόμβων στο δίκτυο

5.2 Δημιουργία Συναλλαγής στο δίκτυο blockchain

Τώρα θα δούμε τη διαδικασία αποστολής συναλλαγών από έναν κόμβο του δικτύου. Επειδή στο δίκτυο μας χρήματα διαθέτει μόνον ο χρήστης User5000 θα αποστείλουμε από αυτόν χρήματα προς τους υπόλοιπους κόμβους. Αυτό γίνεται με την ένα post HTTP request με τα απαραίτητα στοιχεία να αποστέλλονται σε JSON format προς τη κλάση εξυπηρέτησης /submittransaction του αποστολέα κόμβου. Από εκεί επεξεργάζονται και κοινοποιούνται με παρόμοιο τρόπο και στους υπόλοιπους κόμβους του δικτύου.

Συγκεκριμένα στοιχεία της συναλλαγής αποθηκεύονται σε ένα .txt αρχείο της παρακάτω μορφής:



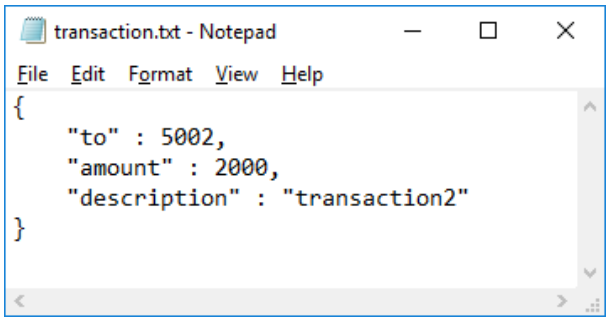
Εικόνα 5. 7 Στοιχεία της συναλλαγής προς κοινοποίηση στο δίκτυο

Τα οποία και στέλνουμε τελικά με το παρακάτω http αίτημα:

```
Curl "http://localhost:5000/submittransaction" -H "Content-
Type:application/json" -X POST -d @transaction.txt
```

Η παραπάνω συναλλαγή σημαίνει την αποστολή 500 tokens από τον User5000 προς τον User5001. Αν ξαναδούμε όμως τα περιεχόμενα στη βάση δεδομένων Neo4j θα μας επιστραφεί πάλι η Εικόνα 5.4 . Αυτό είναι λογικό καθώς η παραπάνω συναλλαγή βρίσκεται ακόμα στα transaction pool των κόμβων και περιμένει τη δημιουργία ενός νέου block για να συμπεριληφθεί στο blockchain.

Επιπλέον υπενθυμίζουμε ότι ο έλεγχος για την επάρκεια tokens του αποστολέα συμβαίνει κατά την εισαγωγή του κάθε transaction στο νέο block. Αυτό σημαίνει, ότι μπορούμε με τον ίδιο τρόπο, να αποστείλουμε μία συναλλαγή από τον User5000 με τα παρακάτω στοιχεία, χωρίς να λάβουμε άμεσα κάποιο μήνυμα σφάλματος:



Εικόνα 5. 8 Στοιχεία συναλλαγής με λανθασμένο amount

Η παραπάνω συναλλαγή ξεπερνά τις οικονομικές δυνατότητες του User5000, ο οποίος έχει ήδη στείλει 500 από τα 2000 tokens με τα οποία ξεκίνησε, στον User5001. Επομένως, περιμένουμε η παραπάνω συναλλαγή να μη συμπεριληφθεί στο επόμενο block.

5.3 Δημοσίευση νέου block με βάση το PoW πρωτόκολλο

Για τη δημιουργία ενός νέου block στο δίκτυο, χρειάζεται να γίνει η επιλογή του κόμβου που θα το κατασκευάσει. Το PoW αποτελεί ένα κατανεμημένο πρωτόκολλο, που επιλέγει αυτόν τον κατάλληλο κόμβο με βάση της ταχύτητάς του στην επίλυση ενός κρυπτογραφικού πάζλ. Η διαδικασία αυτή είναι γνωστή ως mining και οι συμμετέχοντες λέγονται miners.

Για να συμμετέχει ένας κόμβος στη διαδικασία αυτή, αρκεί να αποστείλει το παρακάτω http αίτημα:

```
curl "http://localhost:5000/powmine"
```

Έτσι ο κόμβος που ακούει στην πόρτα 5000 ξεκινά αμέσως τις προσπάθειες επίλυσης του κρυπτογραφικού πάζλ.

Κάθε άλλος κόμβος του δικτύου μπορεί να συμμετάσχει επίσης στην διαδικασία με τον ίδιο ακριβώς τρόπο, ανταγωνιζόμενος τους υπόλοιπος miners στην επίλυση του πάζλ. Στο παράδειγμά μας βάζουμε τους χρήστες User5000, User5001 και User5002 να προσπαθούν να επιλύσουν το επόμενο block, μέσω του παρακάτω batch script:

```
start /min cmd /c curl "http://localhost:5000/powmine"
start /min cmd /c curl "http://localhost:5001/powmine"
start /min cmd /c curl "http://localhost:5002/powmine"
```

Εικόνα 5. 9 Εκκίνηση mining κόμβων μέσω batch script file

Με την εκτέλεση του script, μήνυμα «mining» εμφανίζεται στους συμμετέχοντες miners, που επιβεβαιώνει τη εκκίνηση της διαδικασίας εξόρυξης:

```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:00] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:01] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:01] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:03] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:03] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:03] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:05] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:09] "POST /submittransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:17] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:21] "POST /submittransaction HTTP/1.1" 200 -
Mining...

 * Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:02] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:02] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:04] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:04] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:04] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:04] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:16] "POST /gettransaction HTTP/1.1" 200 -
Mining...

 * Running on http://0.0.0.0:5002/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:03] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:05] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:05] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:05] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:06] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:18] "POST /gettransaction HTTP/1.1" 200 -
Mining...
```

Εικόνα 5. 10 Εκκίνηση mining κόμβων μέσω batch script file

Ένας από αυτούς ανακηρύσσεται νικητής και λαμβάνει το αντίστοιχο μήνυμα, ενώ οι υπόλοιποι κόμβοι ενημερώνονται για την «ήττα» τους. Στο παράδειγμα μας, ο User5000 ήταν ο νικητής της διαδικασίας και επομένως ο κατασκευαστής του νέου block. Τα μηνύματα που έλαβε ο κάθε miner μετά τη λήξη της διαδικασίας, είναι τα παρακάτω:

User5000

```
Mining...
You won!
999074315ff63a109836a4cfad6bd5b2f8a966634300ccb1f01e186183d89c9d
127.0.0.1 - - [15/Jul/2019 16:54:26] "POST /accept_block HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:54:30] "GET /powmine HTTP/1.1" 200 -
```

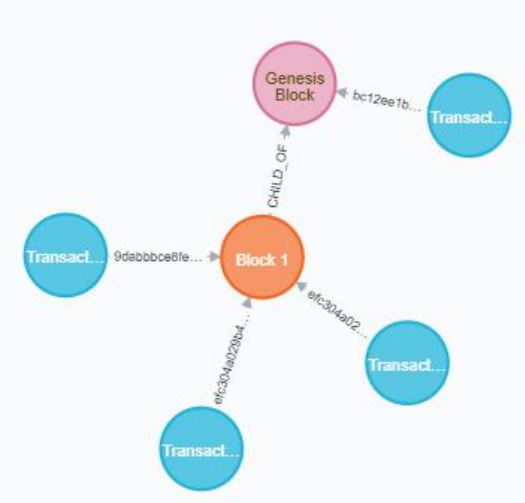
User5001 & User5002

```
Mining...
999074315ff63a109836a4cfad6bd5b2f8a966634300ccb1f01e186183d89c9d
127.0.0.1 - - [15/Jul/2019 16:54:25] "POST /accept_block HTTP/1.1" 200 -
Better luck next time.
127.0.0.1 - - [15/Jul/2019 16:54:27] "GET /powmine HTTP/1.1" 200 -
```

Εικόνα 5. 11 Απόκριση κόμβων μετά την ολοκλήρωση της εξόρυξης

Εδώ τονίζουμε ότι ο κατακερματισμένος αριθμός που εμφανίζεται στα τερματικά των κόμβων αποτελεί το hash του block-branch στο οποίο βρίσκεται ο συγκεκριμένος κόμβος. Ο αριθμός αυτός επαναϋπολογίζεται και τυπώνεται από όλους τους κόμβους (miners και μη) κατά την αποδοχή ενός νέου block.

Ας ελέγξουμε τώρα το blockchain όπως παρουσιάζεται στη βάση δεδομένων Neo4j, μετά την ολοκλήρωση της διαδικασίας εξόρυξης block:



Εικόνα 5. 12 Δημιουργία νέου block με PoW στη Neo4j

Όσον αφορά την καταχώρηση του νέου block βλέπουμε ότι διαθέτει μόνο την ετικέτα Blocks αφού δεν αποτελεί checkpoint (EPOCH_SIZE = 3). Επίσης συνδέεται με το Genesis Block με ακμή CHILD_OF ως block-παιδί αυτού.

Οι συναλλαγές που περιέχονται στο block, είναι τρεις:

1. Η συναλλαγή επιβράβευσης του κατασκευαστή του block με 10 tokens
2. Η συναλλαγή «transaction1» με την οποία ο User5000 απέστειλε 500 tokens στον User5001
3. Η συναλλαγή φόρου, όπου ο miner του block λαμβάνει 1 επιπλέον token από τους κόμβους, που απέστειλαν τις συναλλαγές που συμπεριέλαβε ο πρώτος στο block του. Εδώ η συναλλαγή φόρου είναι μοναδική και αφορά την συναλλαγή «transaction1»

Οι ιδιότητες του Block1 καθώς και αυτές των παραπάνω συναλλαγών φαίνονται στην Εικόνα 5.13, με αντίστροφή σειρά από την οποία αναλύθηκαν.

Αντιλαμβανόμαστε, ότι κατά την κατασκευή του Block1, πραγματοποιήθηκε έλεγχος εγκυρότητας της απεσταλμένης συναλλαγής «transaction2», κατά τον οποίον η τελευταία απορρίφθηκε, οπότε και δεν συμπεριλήφθηκε στο block. Επιπλέον οι διευθύνσεις των χρηστών που εμφανίζονται στη Neo4j είναι κατακερματισμένες για λόγους ανωνυμίας. Ωστόσο εδώ μπορούμε να αντιληφθούμε ότι ο κατασκευαστής του Block1 που έλαβε τα rewards (to_address) ταυτίζεται με τον αποστολέα του «transaction1» (from_address), ο οποίος γνωρίζουμε πως είναι ο User5000.

Εικόνα 5. 13 Ιδιότητες του νέου block και των συναλλαγών αυτού στη Neo4j

Παρατηρούμε ότι ο κατακερματισμένος αριθμός (hash) του Block1 ταυτίζεται με αυτόν της εικόνας 5.11, πράγμα που σημαίνει ότι οι συγκεκριμένοι κόμβοι αποδέχθηκαν το Block1, το οποίο και θεωρούν το υπάρχον branch, δηλαδή το επόμενο block προς επέκταση.

5.4 Δημοσίευση νέου block με βάση το PoS πρωτόκολλο

Σε αυτήν την παράγραφο θα δούμε την δημιουργία ενός νέου Block μέσω του πρωτοκόλλου Proof-of-Stake. Όπως έχουμε αναφέρει και στην παράγραφο 4.3.2. , ο σχεδιασμός των πρωτοκόλλων είναι τέτοιος, που η διαδικασία σύστασης blocks με PoS διαφέρει σε μεγάλο βαθμό απ' ότι με PoW. Η μεγαλύτερη διαφορά είναι ότι στο PoS ανταγωνίζονται μόνο οι κόμβοι που βρίσκονται στο παρόν branch για την επέκτασή του. Αντίθετα στο PoW ο πιο γρήγορος κόμβος στην επίλυση του κρυπτογραφικού πάζλ επεκτείνει το branch του, χωρίς να ενδιαφερόμαστε για το ποιο είναι αυτό.

5.4.1 Δημιουργία διακλαδώσεων στο Blockchain Tree

Για το λόγο αυτό, θα εφαρμόσουμε το PoS σε blockchain tree με διακλαδώσεις. Με τον τρόπο αυτό θα γίνουν περισσότερο κατανοητά τα όσα αναφέρθηκαν και θα δειχθεί η σημασία του staking. Αρχικά όμως θα δούμε τον τρόπο με τον οποίο δημιουργούνται διακλαδώσεις στο δίκτυο μας. Όπως είδαμε στην προηγούμενη παράγραφο και την εικόνα 5.11 οι κόμβοι στις πόρτες 5000, 5001 και 5002 αποδέχθηκαν το Block1 και το επέλεξαν ως το επικρατέστερο branch. Ωστόσο αυτό δεν ισχύει για όλους του κόμβους του δικτύου.

Αυτό συμβαίνει γιατί έχουμε ορίσει την παράμετρο DROP_MESSAGE_PRB = 4 , που σημαίνει ότι υπάρχει ¼ πιθανότητα ένας κόμβος να μη λάβει/αποδεχθεί το νέο block. Στο παρόν δίκτυο, μπορούμε να εντοπίσουμε αυτούς τους κόμβους κοιτώντας τα τερματικά τους.

```
 * Running on http://0.0.0.0:5003/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:08] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:20] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:54:29] "POST /accept_block HTTP/1.1" 200 -
```

```
 * Running on http://0.0.0.0:5006/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:09] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:07] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:19] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:54:28] "POST /accept_block HTTP/1.1" 200 -
```

Εικόνα 5. 14 Τερματικά κόμβων User5003 και User5006

Όπως φαίνεται στην εικόνα 5.14 οι κόμβοι User5003 και User5006 δεν αποδέχθηκαν το Block1 ως το υπάρχον branch, παρέμειναν δηλαδή στον αρχικό Genesis Block. Για να συμμετέχουν όμως στη διαδικασία του PoS πρέπει να λάβουν κάποια tokens. Θα αποστείλουμε λοιπόν από τον User5000 τις δύο παρακάτω συναλλαγές:

transaction.txt - Notepad

File Edit Format View Help

```
{
    "to" : 5003,
    "amount" : 1000,
    "description" : "transaction3"
}
```

transaction.txt - Notepad

File Edit Format View Help

```
{
    "to" : 5006,
    "amount" : 800,
    "description" : "transaction4"
}
```

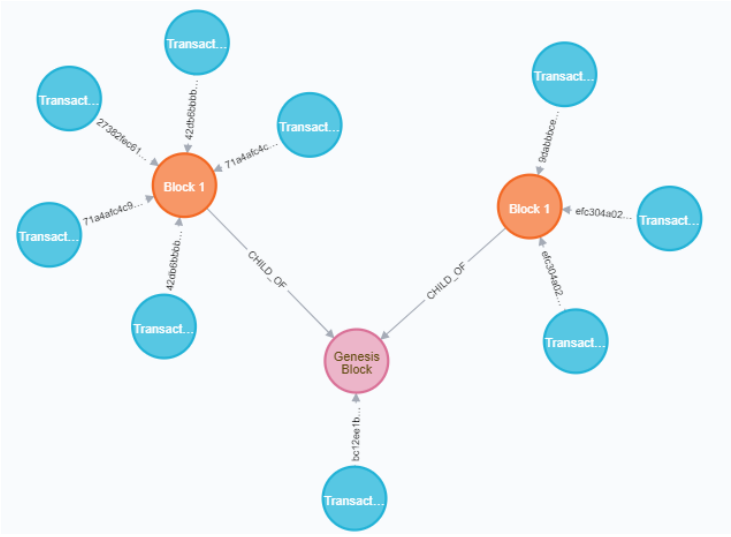
Εικόνα 5. 15 Συναλλαγές δεύτερης αλυσίδας

Αν επιχειρήσουμε να συμπεριλάβουμε τις παραπάνω συναλλαγές, σε block της πρώτης αλυσίδας η δεύτερη δεν θα γίνει αποδεκτή, καθώς εκεί ο User5000 έχει ήδη αποστείλει 500 tokens στον User5001 από τα 2000 που είχε αρχικά.

Ας δούμε όμως τι θα γίνει στην περίπτωση που ο User5003 επιχειρήσει να της εντάξει στο block που θα κατασκευάσει. Βάζουμε τον User5003 να προτείνει block μέσω του PoS πρωτοκόλλου με αίτημα:

```
curl "http://localhost:5003/powmine"
```

Προφανώς χωρίς ανταγωνισμό ο User5003 κερδίζει και επεκτείνει το branch του. Στη συνέχεια ελέγχουμε την κατάσταση του blockchain στη Neo4j:



Εικόνα 5. 16 Δημιουργία διακλάδωσης στο Blockchain Tree

Βλέπουμε ότι ο User5003 έφτιαξε block παράλληλο στο ήδη κατασκευασμένο από τον User5000, Block1. Αυτό ήταν αναμενόμενο αφού ο συγκεκριμένος κόμβος δεν είχε λάβει το προηγούμενο Block1, επομένως συνέχισε να «βλέπει» το Genesis Block ως το block προς επέκταση. Επιπλέον βλέπουμε ότι το καινούργιο block περιλαμβάνει 5 συναλλαγές:

1. Η συναλλαγή επιβράβευσης του κατασκευαστή του block με 10 tokens
2. Η συναλλαγή «transaction3» με την οποία ο User5000 απέστειλε 1000 tokens στον User5003
3. Η συναλλαγή φόρου, για την 2.
4. Η συναλλαγή «transaction4» με την οποία ο User5000 απέστειλε 800 tokens στον User5006
5. Η συναλλαγή φόρου, για την 4.

Δηλαδή και οι δύο συναλλαγές που απέστειλε ο User5000 εγκρίθηκαν, αφού για την παρούσα αλυσίδα ο User5000 εξακολουθούσε να διαθέτει τα αρχικά 2000 tokens του Genesis Block. Οι ιδιότητες των συναλλαγών 2,4 και 1 φαίνονται στην παρακάτω εικόνα:

Εικόνα 5. 17 Ιδιότητες των συναλλαγών της δεύτερης αλυσίδας στη Neo4j

Τέλος, αν κοιτάξουμε τα τερματικά των κόμβων User5003 και User5006 θα δούμε ότι και οι δύο κόμβοι αποδέχθηκαν το νέο Block1 ως το block προς επέκταση:

```
 * Running on http://0.0.0.0:5003/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:46:06] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:08] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:20] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:54:29] "POST /accept_block HTTP/1.1" 200 -
127.0.0.1 - - [16/Jul/2019 00:24:58] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [16/Jul/2019 00:26:44] "POST /gettransaction HTTP/1.1" 200 -
Mining...
You won!
fc7364ddb63baa23928ce6fcce2d45a3fbe63a1858cee76a58d821076a11a0c6
127.0.0.1 - - [16/Jul/2019 00:36:48] "POST /accept_block HTTP/1.1" 200 -
127.0.0.1 - - [16/Jul/2019 00:36:50] "GET /powmine HTTP/1.1" 200 -
```

```
 * Running on http://0.0.0.0:5006/ (Press CTRL+C to quit)
127.0.0.1 - - [15/Jul/2019 16:46:09] "GET /getblockchain HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:07] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:47:19] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2019 16:54:28] "POST /accept_block HTTP/1.1" 200 -
127.0.0.1 - - [16/Jul/2019 00:23:20] "POST /gettransaction HTTP/1.1" 200 -
127.0.0.1 - - [16/Jul/2019 00:26:43] "POST /gettransaction HTTP/1.1" 200 -
fc7364ddb63baa23928ce6fcce2d45a3fbe63a1858cee76a58d821076a11a0c6
127.0.0.1 - - [16/Jul/2019 00:36:49] "POST /accept_block HTTP/1.1" 200 -
```

Εικόνα 5. 18 Τερματικά κόμβων User5003 και User5006

5.4.2 Δημοσίευση νέων blocks σε διακλαδώσεις με το PoS πρωτόκολλο

Οι συνθήκες που έχουν δημιουργηθεί στο δίκτυό μας είναι ιδανικές για την επίδειξη της λειτουργίας του PoS πρωτόκολλου. Συγκεκριμένα, θα ασχοληθούμε με τους κόμβους User5000 και User5001, που βρίσκονται στην πρώτη αλυσίδα και διαθέτουν σε αυτήν 1510 (1500 + 10 Block1 reward) και 500 tokens αντίστοιχα. Με την ίδια λογική, στη δεύτερη αλυσίδα θα μας απασχολήσουν οι User5003 και User5006, στην οποία διαθέτουν 1012 (1000 + 10 Block1 reward + 2 Block1 fees) και 800 tokens αντίστοιχα.

Με τα παρακάτω http get requests οι τέσσερις αυτοί κόμβοι θέτουν υποψηφιότητα για την εκλογή του κατασκευαστή του επόμενου block:

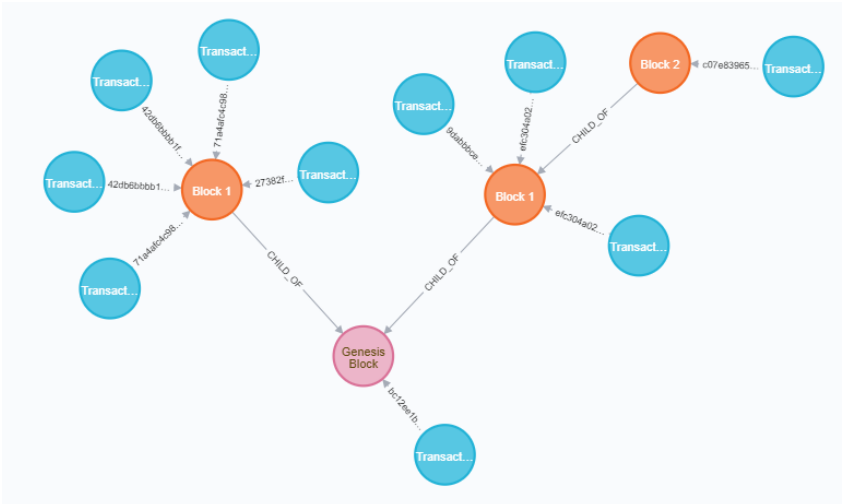
```
curl "http://localhost:5000/applypos"
curl "http://localhost:5001/applypos"
curl "http://localhost:5003/applypos"
curl "http://localhost:5006/applypos"
```

Για την εκκίνηση της διαδικασίας εκλογής, πρέπει να αποστείλουμε αίτημα http στον κόμβο που θα την επιτελέσει. Με το get request :

```
curl "http://localhost:5000/startpos"
```

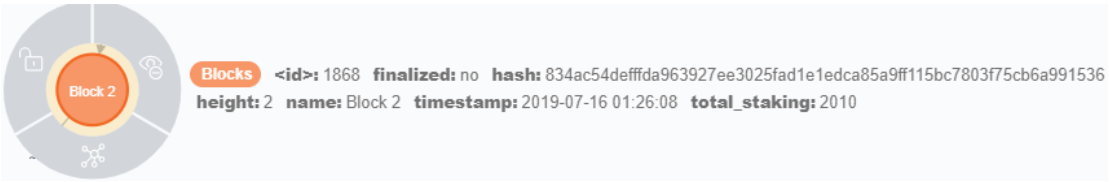
ο κόμβος User5000 εκλέγει τον νικητή stakeholder για το branch στο οποίο ανήκει. Εδώ συγκεκριμένα, οι υποψήφιοι κατασκευαστές είναι οι User5000 και User5001, καθώς οι άλλοι δύο κόμβοι βρίσκονται σε block άλλης αλυσίδας. Η διαδικασία ολοκληρώνεται με τους συμμετέχοντες κόμβους να ενημερώνονται για την έκβαση της ψηφοφορίας. Εδώ νικητής είναι ο User5000 όπου λαμβάνει αντίστοιχο μήνυμα «You Won» με αυτό στο PoW πρωτόκολλο.

Ζητάμε τώρα το blockchain από τη βάση δεδομένων Neo4j και μας επιστρέφεται η εικόνα:



Εικόνα 5. 19 Δημιουργία νέου block με PoS στη Neo4j

Βλέπουμε ότι δημιουργήθηκε νέο block στην αλυσίδα που βρίσκονται οι User5000 και User5001. Το Block2 περιέχει τη συναλλαγή επιβράβευσης προς τον User5000 που το κατασκεύασε. Ας δούμε τώρα τις ιδιότητες του Block2:



Εικόνα 5. 20 Ιδιότητες του δημιουργηθέντος με PoS block στη Neo4j

Η τιμή που αξίζει να παρατηρήσουμε εδώ είναι αυτή του staking. Το staking του block αποτελεί ουσιαστικά το συνολικό stake των κόμβων που συμμετείχαν στη διαδικασία εκλογής για την κατασκευή του. Το staking είναι σημαντικό γιατί αποτελεί το μέτρο της δυσκολίας κατασκευής ενός block ή μίας αλυσίδας στο πρωτόκολλο PoS. Εδώ η τιμή του είναι 2010 που προκύπτει από τα 1510 tokens του User5000 και τα 500 tokens του User5001.

Τώρα θα επαναλάβουμε τη διαδικασία εκλογής από τον κόμβο User5006 που βρίσκεται στην άλλη αλυσίδα:

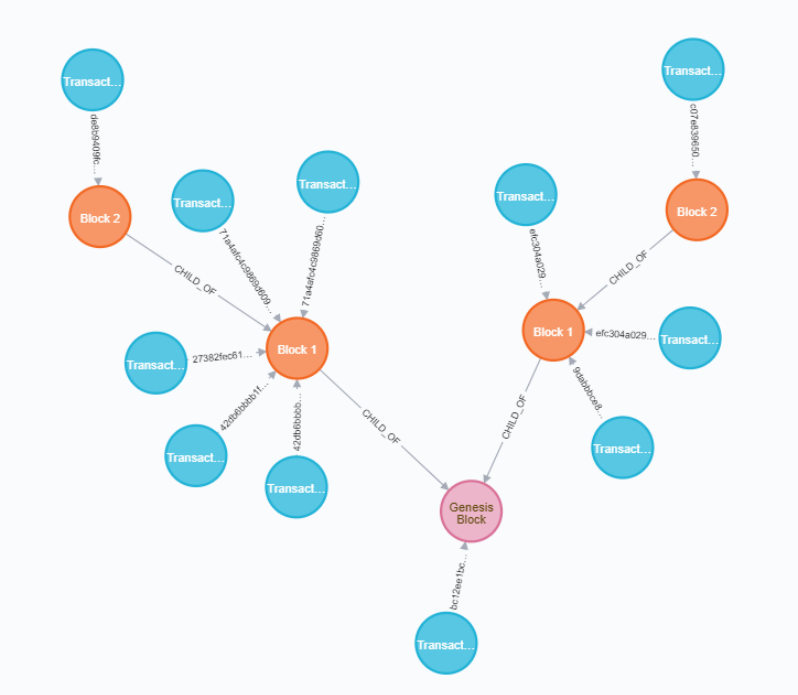
```
curl "http://localhost:5006/startpos"
```

Ο κόμβος στην πόρτα 5006 τυπώνει μήνυμα :

```
The winner is User5003 with the lucky number 769
```

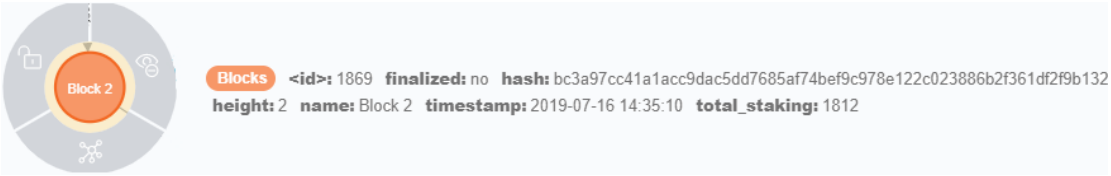
Ενώ οι συμμετέχοντες κόμβοι ενημερώνονται για την έκβαση, όπως προηγουμένως.

Το blockchain στη Neo4j τώρα την εξής εικόνα:



Εικόνα 5. 21 Δημιουργία νέου block με PoS στη Neo4j

Εδώ, όπως ήταν αναμενόμενο επεκτάθηκε το Block1 στο οποίον βρίσκονταν οι κόμβοι User5003 και User5006. Το νέο Block2 περιέχει την συναλλαγή επιβράβευσης προς τον User5003 που το κατασκεύασε και έχει τις παρακάτω ιδιότητες:



Εικόνα 5. 22 Ιδιότητες του δημιουργηθέντος με PoS block στη Neo4j

Εδώ η τιμή του staking είναι 1812 το οποίο προκύπτει από το άθροισμα των 1012 tokens του User5003 και των 800 tokens του User5006.

Γίνεται λοιπόν αντιληπτό, το πώς αυτός ο σχεδιασμός που προτείναμε για το PoS, εφαρμόζεται στην πράξη και μας δίνει μία καλή μετρική δυσκολίας για κάθε αλυσίδα του blockchain Tree.

5.5 Διαμόρφωση Validator sets - Καταθέσεις και Αναλήψεις

Τώρα, θα δούμε τη λειτουργία του Casper ξεκινώντας από την διαμόρφωση των validator sets. Για να μπορεί το Casper να είναι λειτουργικό, θα πρέπει να προϋπάρχει ένας validator ο οποίος θα κάνει finalize τα επόμενα blocks ώστε να μπορούν να εισέλθουν νέοι validators στα validator sets. Στο δίκτυο μας, αυτό επιτυγχάνεται με τον κατασκευαστή του Genesis Block να βρίσκεται στα front και rear validator sets αυτού με μόλις 1 token. Έτσι όταν καινούργιοι validators εισέλθουν σε αυτά η συμμετοχή του πρώτου κόμβου θα είναι ελάχιστη.

Για το λόγο αυτό θα κατασκευάσουμε εξ' αρχής ένα blockchain και θα παρακολουθήσουμε τα validator sets μέσω των αποθηκευμένων δομών στους κόμβους. Με το Chyper Query:

```
match (n) detach delete n
```

σβήνουμε όλες τις αποθηκευμένες πληροφορίες από την Neo4j ενώ με την εντολή:

```
py start.py -p 5000
```

αρχικοποιούμε τον User5000, ο οποίος και κατασκευάζει το Genesis Block.

Στη συνέχεια με το http get αίτημα:

```
curl "http://localhost:5000/getblockchain"
```

ζητάμε από τον User5000 να μας επιστρέψει τις αποθηκευμένες δομές του, με τις οποίες περιγράφεται το blockchain. Το http response που μας επιστρέφει είναι αυτό της εικόνας 5.23:

Command Prompt

```
C:\Users\Konstantinos\Desktop\Casperfin>curl "http://localhost:5000/getblockchain"
{
  "highestfinalized": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
  "highestjustified": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
  "parent": {
    "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30": null
  },
  "tail": [
    {
      "current_front": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "current_rear": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "front_validators_stake": 1,
      "front_votes": {},
      "hash": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
      "height": 0,
      "justified_checkpoints": [
        0
      ],
      "new_front": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "previousCheckpoint": "0",
      "previousHash": "0",
      "rear_validators_stake": 1,
      "rear_votes": {},
      "slashed_validators": [],
      "stake": 0,
      "timestamp": "2019-07-16 20:09:44"
    }
  ],
  "validationblocks": [
    {
      "current_front": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "current_rear": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "front_validators_stake": 1,
      "front_votes": {},
      "hash": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
      "height": 0,
      "justified_checkpoints": [
        0
      ],
      "new_front": {
        "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1
      },
      "previousCheckpoint": "0",
      "previousHash": "0",
      "rear_validators_stake": 1,
      "rear_votes": {},
      "slashed_validators": [],
      "stake": 0,
      "timestamp": "2019-07-16 20:09:44"
    }
  ]
}
```

Εικόνα 5. 23 Απόκριση κόμβου στο http get request /getblockchain

Ουσιαστικά πρόκειται για μία λίστα σε json format με τις δομές highest_finalized, highest_finalized, parent, tail και validationblocks. Αυτές είναι και οι πληροφορίες που στέλνονται από τους χρήστες προς του νεοεισαχθείς κόμβους για αποθήκευση.

Εδώ τα highest_finalized και highest_finalized δίνουν το hash του Genesis Block, το οποίο όπως έχουμε πει, είναι το πρώτο justified και finalized checkpoint κάθε blockchain. Η δομή parent προσφέρει ένα mapping του hash κάθε checkpoint του δικτύου με το hash του checkpoint-πατέρα του. Εδώ δεν υπάρχει checkpoint να προηγείται του Genesis Block, οπότε λαμβάνουμε { hash : null }. Η δομή tails περιέχει τις «ουρές» του blockchain (τα blocks που δεν έχουν αποκτήσει παιδιά) και οι δομή validationblocks τα blocks μπορούν αν δεχθούν ψήφο. Και οι δύο αυτές δομές περιέχουν το μοναδικό block του blockchain το Genesis Block, με τις μεταβλητές του να δίνονται σε μορφή dictionary (‘key’ : ‘value’) .

Από αυτές τις μεταβλητές μας ενδιαφέρουν προς στιγμήν:

Τα current_front και current_rear, που περιέχουν τον User5000 με το stake του, που ισούται με 1 token. Σε κάθε περίπτωση αναγράφεται η συμβολοσειρά που προκύπτει από τον κατακερματισμό του «User5000» μέσω του αλγορίθμου SHA_256, για λόγους ανωνυμίας. Σε ένα πραγματικό δίκτυο blockchain θα μπορούσε να είναι το δημόσιο κλειδί του κόμβου.

Τα front_validator_stake και rear_validator_stakem, που δίνουν το άθροισμα των stakes των front και rear validator sets, αντίστοιχα και που εδώ ισούνται και τα δύο με 1.

Το new_front, που περιέχει τους validators του αμέσως επόμενου checkpoint, σε περίπτωση οριστικοποίησης του παρόντος block. Εφόσον δεν έχουν προηγηθεί μηνύματα κατάθεσης και ανάληψης το new_front είναι το ίδιο με το current_front.

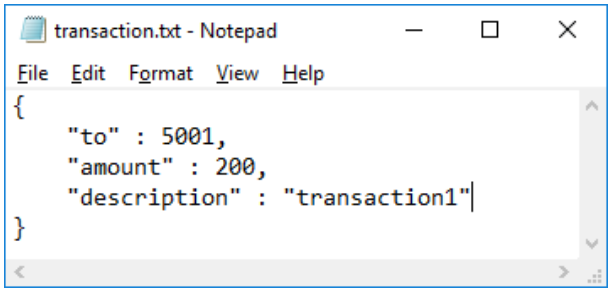
Τώρα ας δούμε, πως οι παραπάνω μεταβλητές και δομές αλλάζουν με τις καταθέσεις νέων validators. Για το σκοπό αυτό θα εντάξουμε τον κόμβο User5001 στο δίκτυο μας εκτελώντας:

```
py start.py -p 5001
```

Στη συνέχεια θα πρέπει να του αποστείλουμε tokens από τον User5000 ώστε να μπορεί να πραγματοποιήσει μήνυμα κατάθεσης. Με το http post αίτημα:

```
Curl "http://localhost:5000/submittransaction" -H "Content-
Type:application/json" -X POST -d @transaction.txt
```

Κοινοποιούμε την συναλλαγή με τα παρακάτω στοιχεία στο δίκτυο:

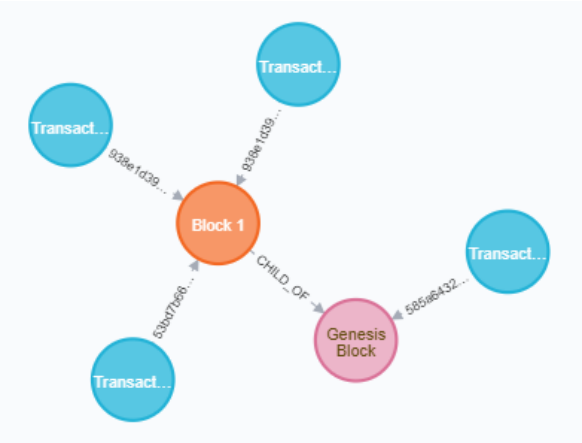


Εικόνα 5. 24 Στοιχεία της απεσταλμένης από τον User5000 συναλλαγής

Για την πραγματοποίηση της συναλλαγής απαιτείται κατασκευή νέου block. Με http αίτημα:

```
curl "http://localhost:5000/powmine"
```

ο User5000 ξεκινά την επίλυση του νέου block. Μετά ολοκλήρωση της διαδικασίας, που ασφαλώς βρίσκει νικητή τον User5000, η εικόνα του blockchain είναι η παρακάτω:

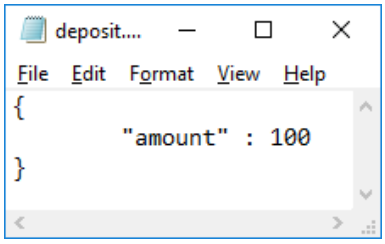


Εικόνα 5. 25 Αναπαράσταση blockchain μετά την κατασκευή του πρώτου block

Τώρα ο χρήστης User5001 διαθέτει 200 tokens και μπορεί να καταθέσει μέρος αυτών ώστε να μπει στο validator set. Αυτό πραγματοποιείται με το http post αίτημα:

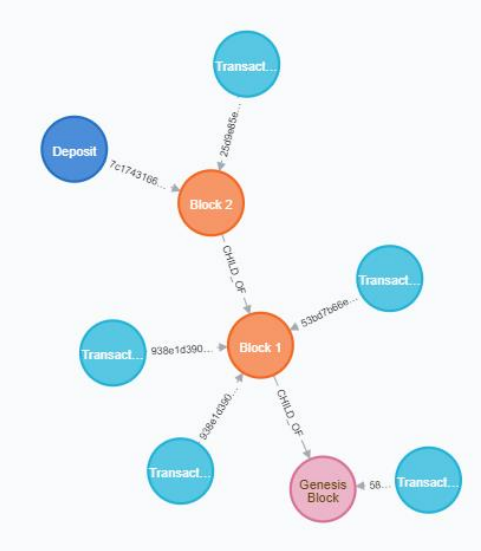
```
Curl “http://localhost:5001/deposit” –H “Content-
Type:application/json” –X POST –d @deposit.txt
```

με το οποίο κοινοποιεί το stake της κατάθεσης του στο δίκτυο:



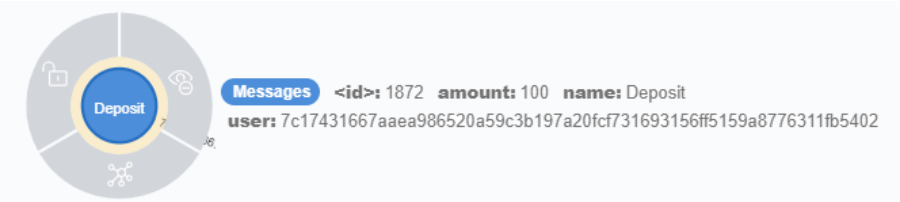
Εικόνα 5. 26 Stake μηνύματος κατάθεσης

Το μήνυμα κατάθεσης αντιμετωπίζεται από τους κόμβους σαν μία συναλλαγή, με την έννοια ότι ελέγχεται αν ο αποστολέας διαθέτει το ποσό των tokens και αποθηκεύεται στο transaction pool. Συνεπώς για να τεθεί σε ισχύ πρέπει να συμπεριληφθεί σε ένα νέο block. Ξεκινάμε με τον γνωστό τρόπο τη διαδικασία mining από τον User5001, ο οποίος κατασκευάζει το επόμενο block. Τώρα το αποθηκευμένο blockchain στη βάση Neo4j έχει τη μορφή της εικόνας 5.27.



Εικόνα 5. 27 Οπτικοποίηση μηνύματος κατάθεσης στην Neo4j

Το μήνυμα deposit λοιπόν εμπεριέχεται στο Block2 και οι ιδιότητες του είναι οι ακόλουθες:



Εικόνα 5. 28 Ιδιότητες μηνύματος κατάθεσης στην Neo4j

Μεγαλύτερο ενδιαφέρον όμως έχει η διαμόρφωση των validator sets στο καινούργιο block. Με το http get αίτημα:

```
curl "http://localhost:5001/getblockchain"
```

ο User5001 μας επιστρέφει τις αποθηκευμένες δομές του. Το Block2 μπορούμε να το εντοπίσουμε μέσω της δομής tail, αφού αποτελεί ουρά του blockchain:

```
{
  "current_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "current_rear": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "front_validators_stake": 1,
  "front_votes": {},
  "hash": "5f94b99df5cd42cee523e1c0df5498fe45e11284ad1f1dae515ff885da30f0bf",
  "height": 2,
  "justified_checkpoints": [
    0
  ],
  "new_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1,
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
  "previousHash": "97ebcaae280778296a36d52a5b240cbc885c60e5d665e2f2ccbd90424d6d6f5e",
  "rear_validators_stake": 1,
  "rear_votes": {},
  "slashed_validators": [],
  "stake": 0,
  "timestamp": "2019-07-17 20:04:48"
}
```

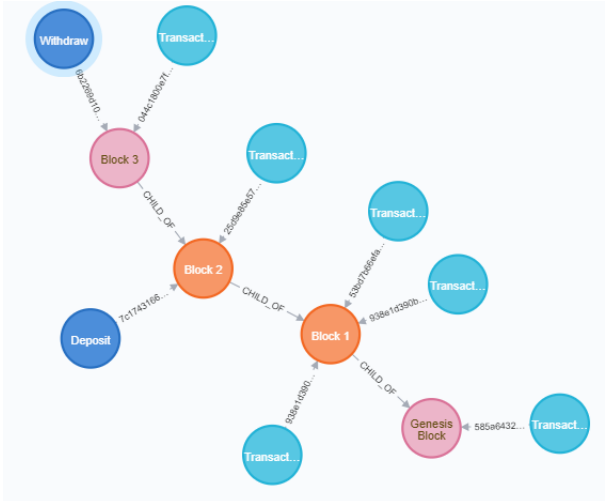

Εικόνα 5. 29 Αποθηκευμένες δομές στο Block2

Από τις δομές που αφορούν τα validator sets μόνο η new front έχει αλλάξει σε σχέση με πριν. Αυτό συμβαίνει γιατί η αλλαγή των validator set πραγματοποιείται με την οριστικοποίηση ενός checkpoint. Μέχρι τότε, όλες οι αλλαγές εφαρμόζονται στο new front, το οποίο αποτελεί το επόμενο front validator set για την παρούσα αλυσίδα.

Εν συνεχεία, θα αποστείλουμε ένα μήνυμα ανάληψης. Η ανάληψη γίνεται μόνο από τους validators που βρίσκονται στο current front validator set. Αυτό σημαίνει, ότι στην προκειμένη περίπτωση μόνον ο User5000 μπορεί να πραγματοποιήσει μία ανάληψη, ενώ μία τέτοια προσπάθεια από άλλο κόμβο θα απορριπτόταν. Επιπλέον επειδή η ανάληψη αφορά το συνολικό stake ενός validator, δεν χρειάζεται να αποστείλουμε κάποια μεταβλητή, οπότε ικανοποιείται με το παρακάτω http get αίτημα:

```
curl "http://localhost:5001/withdraw"
```

Με το γνωστό τρόπο εκκινούμε τη διαδικασία εξόρυξης από τον κόμβο User5001 και ελέγχουμε το blockchain στη Neo4j:



Εικόνα 5. 30 Οπτικοποίηση μηνύματος ανάληψης στην Neo4j

Εδώ, πέραν του μηνύματος withdraw, που συμπεριλαμβάνεται στο καινούργιο block, παρατηρούμε ότι το Block3 διαθέτει και το label του Checkpoint (ροζ χρώμα). Αυτό συμβαίνει αφού στο συγκεκριμένο blockchain έχουμε ορίσει EPOCH_SIZE=3, άρα τα blocks σε ύψος πολλαπλάσιο του 3 είναι checkpoints. Όσον αφορά την ανάληψη, στις ιδιότητες του μηνύματος withdraw της εικόνας 5.30, βλέπουμε το amount = 0. Αυτό συμβαίνει, καθώς το ποσό ανάληψης κατοχυρώνεται μετά την withdraw delay περίοδο του validator. Κατά την περίοδο αυτή το stake, μπορεί ακόμα και να μηδενιστεί αν επιβεβαιωθεί ότι ο validator παρέβη κάποιον από τους θεσπισμένους κανόνες.

Εικόνα 5. 31 Ιδιότητες μηνύματος ανάληψης στην Neo4j

Στη συνέχεια, ελέγχουμε την κατάσταση των validator sets στο καινούργιο block με το http get αίτημα:

```
curl "http://localhost:5001/getblockchain"
```

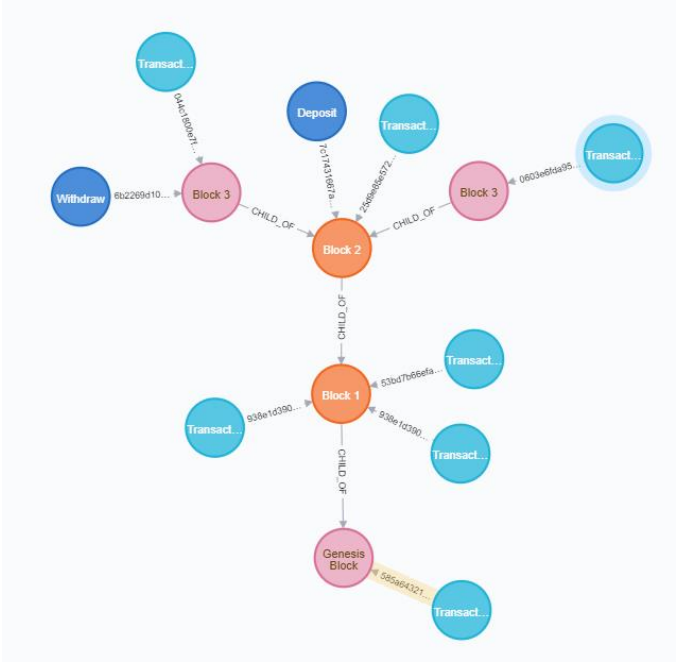
Το Block3 μπορούμε να το εντοπίσουμε τόσο μέσω της δομής tail, αφού αποτελεί ουρά του blockchain, όσο και μέσω της validation blocks αφού αποτελεί checkpoint block:


```
{
  "current_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "current_rear": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "front_validators_stake": 0.99,
  "front_votes": {},
  "hash": "df02fd15f82e7ba7190b39ba036247b74ac7d6df1f334aaf41baf59ea84d528a",
  "height": 3,
  "justified_checkpoints": [
    0
  ],
  "new_front": {
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "d64f8d77d06732afe897bedc95bdcb390c48b16658eca691326ef213c7e07c30",
  "previousHash": "5f94b99df5cd42cee523e1c0df5498fe45e11284ad1f1dae515ff885da30f0bf",
  "rear_validators_stake": 0.99,
  "rear_votes": {},
  "slashed_validators": [],
  "stake": 0,
  "timestamp": "2019-07-17 20:06:32"
},
```


Εικόνα 5. 32 Αποθηκευμένες δομές στο Block3

Εδώ παρατηρούμε, ουσιαστικά δύο αλλαγές. Η πρώτη είναι η αφαίρεση του User5000 από το new_front set, λόγω του μηνύματος ανάληψης που απέστειλε. Η δεύτερη είναι η αλλαγή στα stakes των validator sets από 1 σε 0.99. Αυτό πρόκειται ουσιαστικά για το Inactivity Leak, το οποίο εφαρμόζεται σε κάθε νέο checkpoint, στα stakes των εν ενεργεία validators, που δεν απέστειλαν ψήφο την παρούσα εποχή. Με EPOCH_SIZE = 3 τα χρονικά περιθώρια μοιάζουν μικρά για την αποστολή ψήφο από κάποιον validator, όμως σε ένα πραγματικό blockchain δίκτυο ο αριθμός αυτός θα ξεπερνούσε το 100.

Τέλος, σημειώνουμε ότι οι δομές διαμορφώνονται τελικά ανάλογα με τα μηνύματα καταθέσεων και αναλήψεων που συμπεριλαμβάνονται σε blocks της παρούσας αλυσίδας. Αυτό σημαίνει, ότι αν στο παρόν blockchain ένας κόμβος επεκτείνει το Block2, όπως φαίνεται στην εικόνα 5.32, το καινούργιο Block3 θα συνεχίσει να έχει τον User5000 στο new_front set του, καθώς το μήνυμα Withdraw δεν είναι αποθηκευμένο στην αλυσίδα του.



Εικόνα 5. 33 Το blockchain μετά την επέκταση του Block2

Για να δούμε το πώς πραγματοποιούνται οι αλλαγές στα validator sets πρέπει να δούμε πρώτα την διαδικασία ψηφοφορίας των validators.

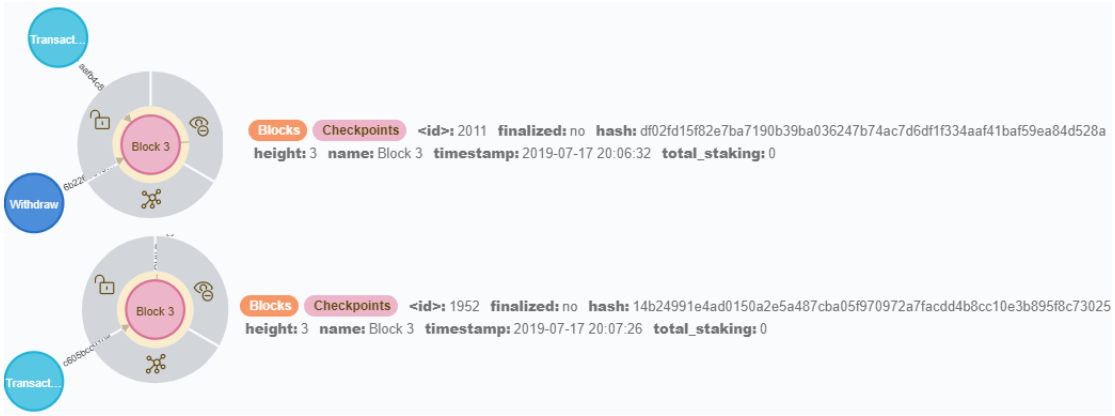
5.6 Αποστολή ψήφου από Validator

Στην παράγραφο αυτή θα δούμε τον τρόπο με τον οποίο ψηφίζουν οι υπάρχοντες validators, τα validation blocks. Επιπλέον θα δούμε το αντίκτυπο των απεσταλμένων ψήφων, τόσο σε επίπεδο αποθηκευμένων δομών, όσο και στη γενικότερη λειτουργία του blockchain και των κόμβων. Οι ψήφοι που αποστέλλονται χωρίζονται σε δύο κατηγορίες σύμφωνα με το σχεδιασμό που έχουμε πραγματοποιήσει:

1. Justification Vote ή Ψήφος Δικαιολόγησης ονομάζεται η ψήφος με source και target δύο checkpoints του blockchain Tree. Αν η ψήφος πάρει την απαραίτητη πλειοψηφία το **target checkpoint γίνεται justified**.
2. Finalization Vote ή Ψήφος Οριστικοποίησης ονομάζεται η ψήφος με source ένα checkpoint και target ένα άμεσο block-παιδί του source (target height = source height +1). Σε περίπτωση ικανοποίησης των κριτηρίων πλειοψηφίας και χρονικών περιορισμών το **source checkpoint γίνεται finalized.**

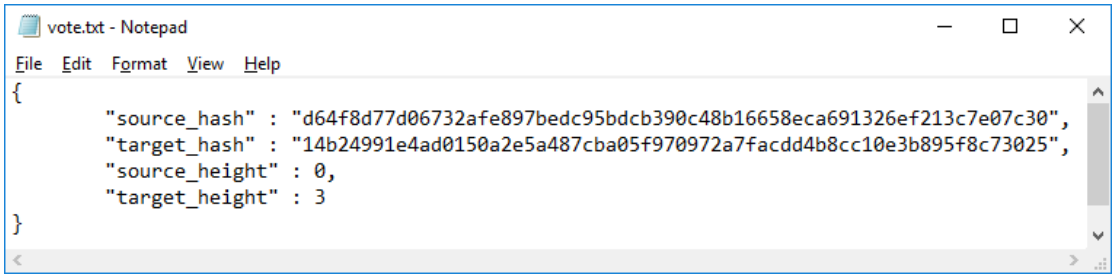
5.6.1. Block Justification

Για το justification χρησιμοποιήσουμε το blockchain που έχουμε κατασκευάσει και που φαίνεται στην Εικόνα 5.32. Εδώ υπάρχουν δύο υποψήφια blocks προς justification, των οποίων οι ιδιότητες είναι οι ακόλουθες:



Εικόνα 5. 34 Ιδιότητες των blcoks προς justification

Ο User5000 είναι ο μοναδικός validator και στα δύο αυτά blocks, οπότε μόνον αυτός μπορεί να αποστείλει ψήφο σε αυτή τη φάση. Τα στοιχεία της ψήφου βρίσκονται στο παρακάτω .txt αρχείο σε JSON μορφή:



Εικόνα 5. 35 Τα στοιχεία της ψήφου του validator

και αποστέλλονται με το παρακάτω http get αίτημα:

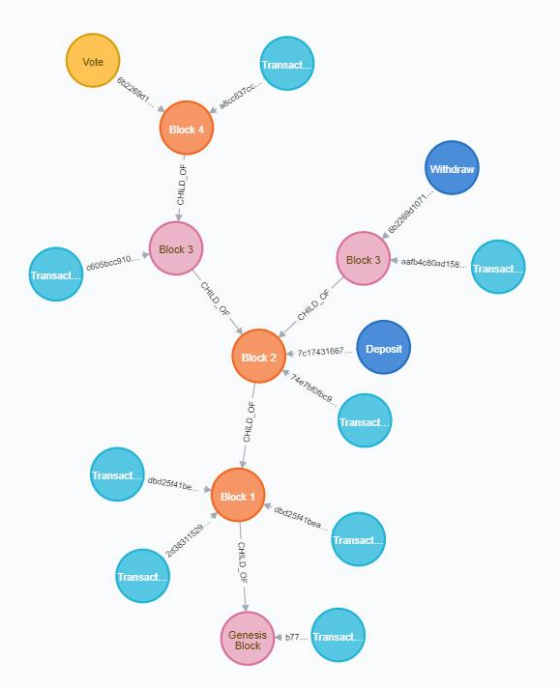
```
Curl "http://localhost:5000/submitvote" -H "Content-
Type:application/json" -X POST -d @vote.txt
```

Όπως είναι εμφανές από τα παραπάνω στοιχεία, το source είναι το Genesis Block και το target είναι το Block3, που δεν περιέχει Withdraw message. Αν όμως κοιτάξουμε τα τερματικά των κόμβων (Εικόνα 5.35), θα δούμε ότι ο User5000 και ο User5001 βρίσκονται στην άλλη αλυσίδα, ενώ μόνον ο User5002 βρίσκεται στο checkpoint το οποίο ψηφίσαμε. Αυτό συνέβη εσκεμμένα, ώστε να επιδειχθεί το fork choice rule του Casper, δηλαδή η αλλαγή στην αλυσίδα με το highest justified checkpoint. Εδώ highest justified checkpoint είναι ακόμα το Genesis Block, οπότε και τα δύο Block3 είναι εξίσου αποδεκτά. Μόλις όμως, η ψήφος που αποστείλαμε συμπεριληφθεί σε block, το target block αυτής θα αποτελεί το νέο highest justified checkpoint.

Εικόνα 5. 36 Επιλεγμένες αλυσίδες των κόμβων

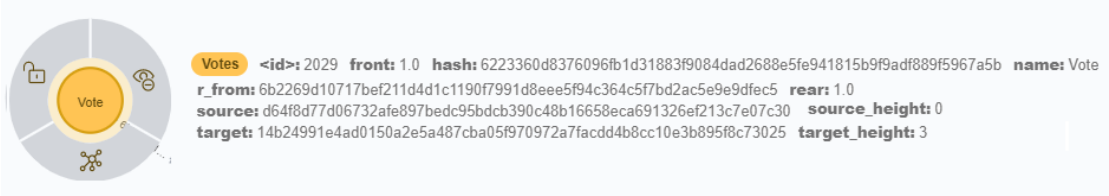
Μένει λοιπόν να κατασκευάσουμε ένα καινούργιο block, ώστε η ψήφος να τεθεί σε ισχύ. Όπως έχουμε πει για να συμπεριληφθεί μία ψήφος σε ένα καινούργιο block πρέπει η πρώτη να αφορά blocks της αλυσίδας του καινούργιου block (κανόνες justification/finalization) . Για το λόγο αυτό ο κόμβος που θα κατασκευάσει το νέο block θα είναι ο User5002, που βρίσκεται στην σωστή αλυσίδα. Θα μπορούσαμε εναλλακτικά να αποστέλλουμε τα votes σε συγκεκριμένα branches, δηλαδή να λαμβάνονται μόνο από κόμβους/miners της ίδιας αλυσίδας, ωστόσο κάτι τέτοιο δεν είναι απαραίτητο για τις ανάγκες τις παρούσας διπλωματικής.

Μετά την κατασκευή του νέου block από τον User5002 το blockchain έχει την παρακάτω εικόνα:



Εικόνα 5. 37 Αναπαράσταση ψήφου validator στη Neo4j

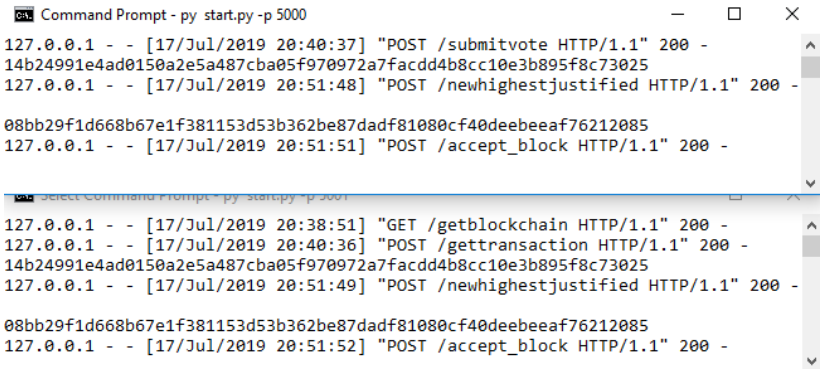
Η ψήφος καταχωρήθηκε επιτυχημένα στο Block4 ως οντότητα με ετικέτα «Votes». Οι ιδιότητες της παρουσιάζονται στην παρακάτω εικόνα:



Εικόνα 5. 38 Ιδιότητες της οντότητας Vote στη Neo4j

Από τις ιδιότητες αυτές, άξιες αναφοράς είναι οι front και rear που περιέχουν το ποσοστό συμμετοχής της ψήφου του validator στο συνολικό stake κάθε validator set. Οι τιμές που λαμβάνουν αυτές είναι στο διάστημα (0,1]. Εδώ, και οι δύο έχουν τιμή 1, αφού ο validator που απέστειλε την ψήφο, ελέγχει το 100% του συνολικού stake και των δύο αυτών set.

Ας δούμε τώρα τις αλλαγές που προκύπτουν στους κόμβους και τις δομές τους με την οριστικοποίηση του Block3. Αρχικά, στα τερματικά των User5000 και User5001 παρατηρούμε μετά την λήψη αιτήματος /newhighestjustified, την αλλαγή αυτών από την άλλη αλυσίδα προς το νεοδημιουργηθέν Block4, ως ο απόγονος του highest justified checkpoint που δεν έχει αποκτήσει ακόμα παιδιά:



Εικόνα 5. 39 Ιδιότητες της οντότητας Vote στη Neo4j

Η άλλη αλλαγή εντοπίζεται στις αποθηκευμένες δομές των blocks. Στη διαδικασία ψηφοφορίας μας ενδιαφέρουν:

Τα rear_votes και front_votes με τα απεσταλμένα vote links στην παρούσα αλυσίδα, αποθηκευμένα στη μορφή {(source,target) = percentage}. Στα blocks πριν του Block4 οι δομές αυτές ήταν άδειες αφού δεν είχε αποσταλεί καμία ψήφος στο δίκτυο.

Το justified_checkpoints, μία λίστα με τα justified checkpoints της παρούσας αλυσίδας. Στα blocks πριν του Block4 η λίστα αυτή ήταν justified = [0], δηλαδή περιείχε μόνον το Genesis Block.

Οι νέες τιμές των δομών φαίνονται στο αντικείμενο του Block4, το οποίο βρίσκεται αποθηκευμένο τόσο στη δομή tails, όσο και στη δομή validation blocks. Με ένα http get αίτημα /getblockchain σε οποιονδήποτε κόμβο του δίκτυο λαμβάνουμε τις παρακάτω πληροφορίες:

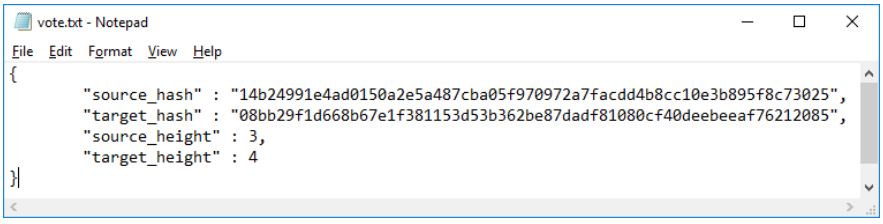
```
{
  "current_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "current_rear": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "front_validators_stake": 0.99,
  "front_votes": {
    "(0, 3)": 1.0
  },
  "hash": "08bb29f1d668b67e1f381153d53b362be87dadf81080cf40deebeeaf76212085",
  "height": 4,
  "justified_checkpoints": [
    0,
    3
  ],
  "new_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1,
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "14b24991e4ad0150a2e5a487cba05f970972a7facdd4b8cc10e3b895f8c73025",
  "previousHash": "14b24991e4ad0150a2e5a487cba05f970972a7facdd4b8cc10e3b895f8c73025",
  "rear_validators_stake": 0.99,
  "rear_votes": {
    "(0, 3)": 1.0
  },
  "slashed_validators": [],
  "stake": 0,
  "timestamp": "2019-07-17 20:51:39"
}
```


Εικόνα 5. 40 Αποθηκευμένη πληροφορία στο Block 4

Εδώ βλέπουμε τα front_votes και rear_votes να έχουν τις τιμές {(0,3) : 1.0}, αυτό σημαίνει ότι το 100% και του current_front και του current_rear απέστειλε ψήφο από το checkpoint σε ύψος 0 προς το checkpoint σε ύψος 3. Αντίστοιχα η λίστα justified_checkpoints περιλαμβάνει πλέον και το checkpoint σε ύψος 3, αφού αυτό έγινε μόλις justified. Βλέπουμε ότι στις δομές αυτές, δεν είναι ανάγκη να χρησιμοποιήσουμε τα hashes των αντίστοιχων checkpoints, γιατί εδώ μόνο από το ύψος τους μπορούμε να τα προσδιορίσουμε μοναδικά. Αυτό συμβαίνει γιατί η δομές αναφέρονται σε blocks/checkpoints της παρούσας αλυσίδας και επιπλέον κάθε μπλοκ μπορεί να έχει μόνο έναν πατέρα. Εν ολίγοις η αλυσίδα από οποιοδήποτε block μέχρι το Genesis Block είναι μοναδική, άρα σε αυτή υπάρχει μόνο ένα block σε κάθε ύψος.

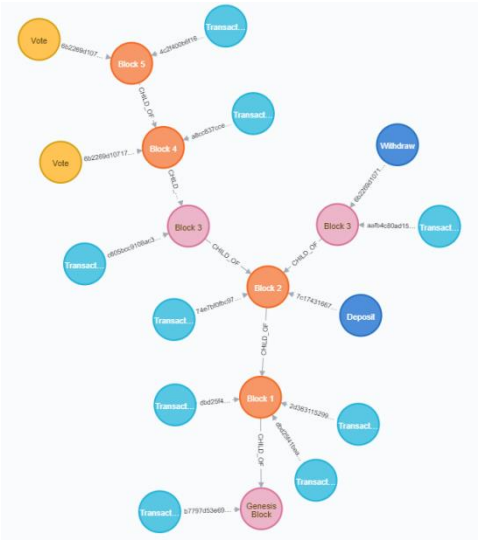
5.6.2. Block Finalization

Στη συνέχεια θα προχωρήσουμε στην οριστικοποίηση του justified Block3, από τον User5000. Η ψήφος αποστέλλεται με τον ίδιο τρόπο με προηγουμένως, με τα στοιχεία της αυτή τη φορά να είναι τα ακόλουθα:



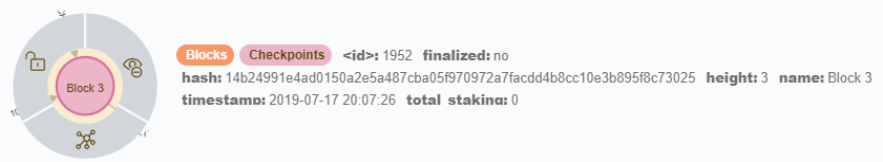
Εικόνα 5. 41 Ψήφος οριστικοποίησης Block3

Όπως έχουμε αναφέρει για να οριστικοποιηθεί ένα justified block θα πρέπει να έχει αποσταλεί ο απαραίτητος αριθμός ψήφων, με target ένα άμεσο παιδί του, πριν τη δημιουργία του επόμενου checkpoint και με όλες αυτές τις ψήφους να συμπεριλαμβάνονται σε blocks της αλυσίδας του. Έτσι επεκτείνοντας το Block4 το blockchain στη Neo4j λαμβάνει την παρακάτω εικόνα:



Εικόνα 5. 42 Το blockchain στη Neo4j μετά τη δημιουργία του Block5

Βλέπουμε ότι η ψήφος συμπεριλαμβάνεται στο επόμενο block, αφού αφορά blocks της παρούσας αλυσίδας. Ωστόσο, παρότι όλες οι προϋποθέσεις έχουν ικανοποιηθεί, το Block3 δεν έχει γίνει ακόμα finalized:



Εικόνα 5. 43 Η οριστικοποίηση του Block3 δεν πραγματοποιήθηκε

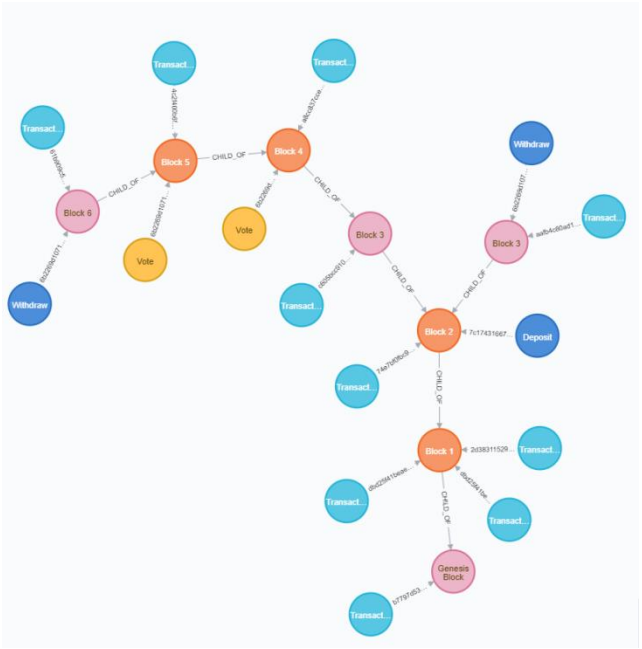
Αυτό συμβαίνει γιατί ο έλεγχος για την οριστικοποίηση ενός checkpoint πραγματοποιείται μόνο κατά τη δημιουργία του αμέσως επόμενου checkpoint. Αυτό για εδώ σημαίνει ότι κατά τη δημιουργία του επόμενου checkpoint (της παρούσας αλυσίδας), δηλαδή του Block6, θα γίνει ο έλεγχος για την οριστικοποίηση του Block3. Όσον αφορά της πληροφορίες του Block5 που κατασκευάσαμε αυτές είναι οι ακόλουθες:


```
{
  "current_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "current_rear": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 0.99
  },
  "front_validators_stake": 0.99,
  "front_votes": {
    "(0, 3)": 1.0,
    "(3, 4)": 1.0
  },
  "hash": "a1be3adf1b3ec3b91fb7d208bda4065da4b802f91dabd73432bcededb67148cc",
  "height": 5,
  "justified_checkpoints": [
    0,
    3
  ],
  "new_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1,
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "14b24991e4ad0150a2e5a487cba05f970972a7facdd4b8cc10e3b895f8c73025",
  "previousHash": "08bb29f1d668b67e1f381153d53b362be87dadf81080cf40deebeeaf76212085",
  "rear_validators_stake": 0.99,
  "rear_votes": {
    "(0, 3)": 1.0,
    "(3, 4)": 1.0
  },
  "slashed_validators": [],
  "stake": 0,
  "timestamp": "2019-07-17 22:13:31"
}
```


Εικόνα 5. 44 Αποθηκευμένη πληροφορία στο Block 5.

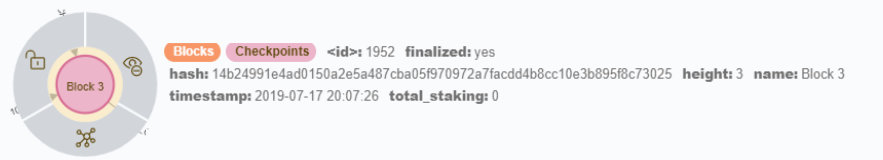
Βλέπουμε ότι η ψήφος οριστικοποίησης έχει ενταχθεί κανονικά στις δομές front_votes και rear_votes. Προτού προχωρήσουμε στην κατασκευή του επόμενου block της αλυσίδας, με την οποία οριστικοποιείται το Block3 θα αποστείλουμε μήνυμα ανάληψης από τον User5000. Αυτό γίνεται για να είναι εμφανέστερος ο τρόπος με τον οποίο τα validator sets αλλάζουν κατά την οριστικοποίηση ενός checkpoint.

Έτσι μετά και την κατασκευή του επόμενου block το blockchain παίρνει την παρακάτω μορφή:



Εικόνα 5. 45 Το blockchain στη Neo4j μετά τη δημιουργία του Block5

Ενώ αν κοιτάξουμε τις ιδιότητες του Block3 αντιλαμβανόμαστε ότι το finalization τέθηκε σε ισχύ.



Εικόνα 5. 46 Η οριστικοποίηση του Block3 πραγματοποιήθηκε

Το finalization του Block3 σημαίνει και αλλαγές στα validator sets. Το πώς διαμορφώνονται τα τελευταία γίνεται ορατό, μέσα από τις αποθηκευμένες πληροφορίες του Block6. Πάλι με αίτημα /getblockchain προς κάποιον κόμβο του δικτύου, λαμβάνουμε τις δομές περιγραφής του blockchain. Το Block6 μας δίνεται στην μορφή της εικόνας 5.46.

```
{
  "current_front": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1.0296,
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "current_rear": {
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5": 1.0296
  },
  "front_validators_stake": 101.0296,
  "front_votes": {
    "(0, 3)": 1.0,
    "(3, 4)": 1.0
  },
  "hash": "a78ca86ca3b8c81ef7c0961f3597bc8b6d9356080759c6f4e80ed0c26eb1ffe4",
  "height": 6,
  "justified_checkpoints": [
    0,
    3
  ],
  "new_front": {
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "14b24991e4ad0150a2e5a487cba05f970972a7facdd4b8cc10e3b895f8c73025",
  "previousHash": "a1be3adf1b3ec3b91fb7d208bda4065da4b802f91dabd73432bcededb67148cc",
  "rear_validators_stake": 1.0296,
  "rear_votes": {
    "(0, 3)": 1.0,
    "(3, 4)": 1.0
  },
  "slashed_validators": [],
  "stake": 0,
  "timestamp": "2019-07-17 22:20:40"
}
```


Εικόνα 5. 47 Αποθηκευμένη πληροφορία στο Block 6.

Ας δούμε λοιπόν, πως διαμορφώνονται τα τρία sets:

Το current_front λαμβάνει τους validators του new_front του προηγούμενου checkpoint, δηλαδή του Block3. Ωστόσο το stake του User5000 είναι αυξημένο σε σχέση με πριν, επειδή ανταμείβεται για τις ψήφους που απέστειλε την περασμένη εποχή. Αντίθετα το stake του User5001 είναι το ίδιο με το deposit του, αφού μόλις εισήλθε στο validator set.

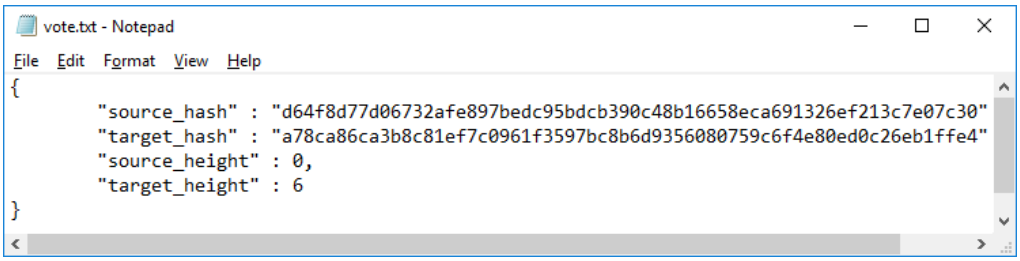
Το current_rear παίρνει τους validators από το current_front του προηγούμενου block. Μέχρι τώρα ο validator User5000 ήταν ο μοναδικός και στα δύο validator sets.

Το new_front ξεκινά πάντα το ίδιο με το new_front του προηγούμενου block και διαμορφώνεται ανάλογα με τις καταθέσεις και τις αναλήψεις του νέου block. Στην προκειμένη περίπτωση το new_front του Block5 περιείχε τους User5000 και User5001, με τον πρώτο να στέλνει μήνυμα ανάληψης και να αποχωρεί από αυτό.

5.7 Αναφορά κακόβουλων Validators – Slashing

Το τελευταίο κομμάτι της επίδειξης σχετίζεται με την αναφορά των κακόβουλων validators. Κάθε validator που αποστέλλει ψήφο που παραβαίνει τους κανόνες που έχουν αναλυθεί στο Κεφάλαιο 2 και στο Κεφάλαιο 4, μπορεί να εντοπιστεί και να τιμωρηθεί. Την δουλεία αυτή την αναλαμβάνουν οι ίδιοι οι κόμβοι του δικτύου, μέσω της αποστολής reports για τους παραβάτες και με κίνητρο τη χρηματική επιβράβευση.

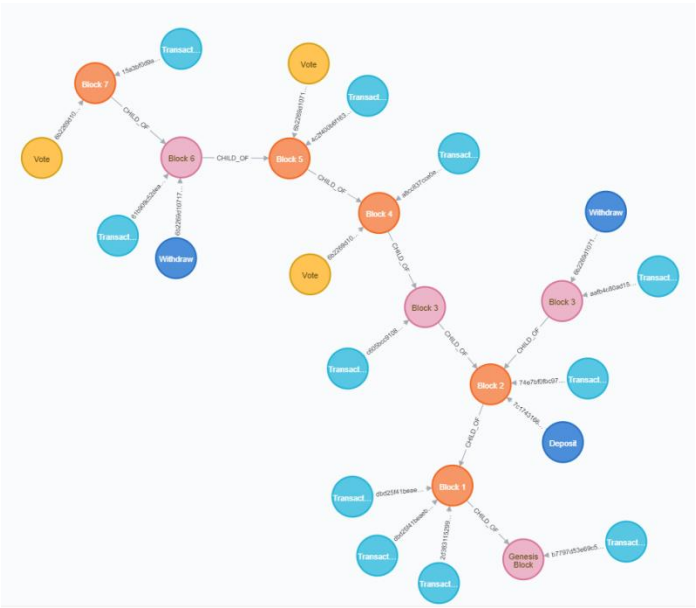
Για να δείξουμε αυτήν τη λειτουργία εδώ θα πρέπει πρώτα κάποιος κόμβος να παραβεί τους κανόνες. Στο παρόν blockchain μία τέτοια ψήφος από τον User5000 είναι η ακόλουθη:



Εικόνα 5. 48 Αντικρουόμενη ψήφος validator

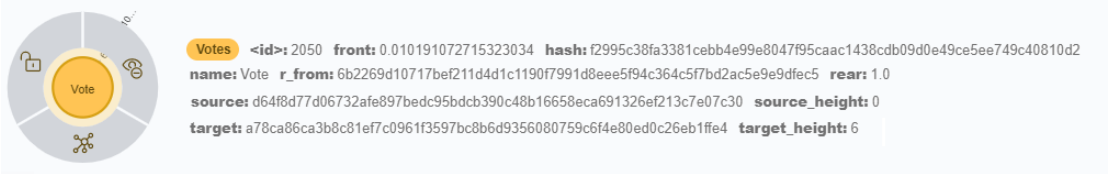
Και αυτό γιατί αντικρούει την ψήφο finalization που έχει αποστείλει. Όπως έχουμε πει για τα διαστήματα των υψών, που περιλαμβάνονται σε δύο ψήφους ενός validator, απαγορεύεται να περιέχεται το ένα στο άλλο. Εδώ όμως το διάστημα υψών (3,4) της προηγούμενης ψήφου περιέχεται στο διάστημα (0,6) αυτής της ψήφου του User5000, κάτι που τις καθιστά αντικρουόμενες.

Μετά την αποστολή της παραπάνω ψήφου και τη δημιουργία του επόμενου block το blockchain παίρνει την παρακάτω μορφή:



Εικόνα 5. 49 Το blockchain μετά τη δημιουργία του Block7

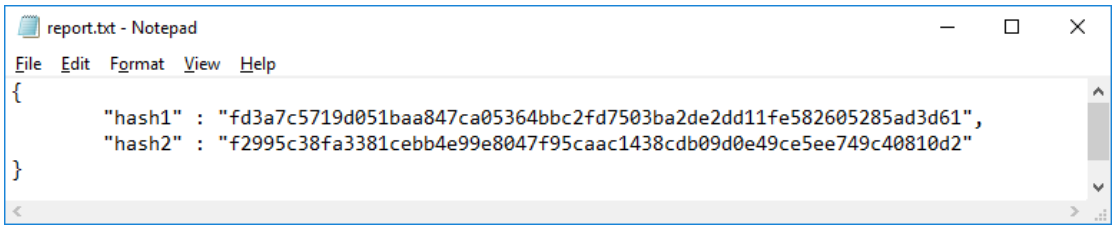
Ενώ η νέα ψήφος έχει τις παρακάτω ιδιότητες



Εικόνα 5. 50 Ιδιότητες αντικρουόμενης ψήφου

Παρατηρούμε επίσης, ότι το ποσοστό συμμετοχής του User5000 στο current_front validator set, έχε σχεδόν εκμηδενιστεί μετά την είσοδο του User5001 σε αυτό με 100 tokens.

Τώρα μπορούμε από οποιονδήποτε κόμβο, να αποστείλουμε μήνυμα αναφοράς με τα αναγνωριστικά (hashes) των αντικρουόμενων ψήφων:

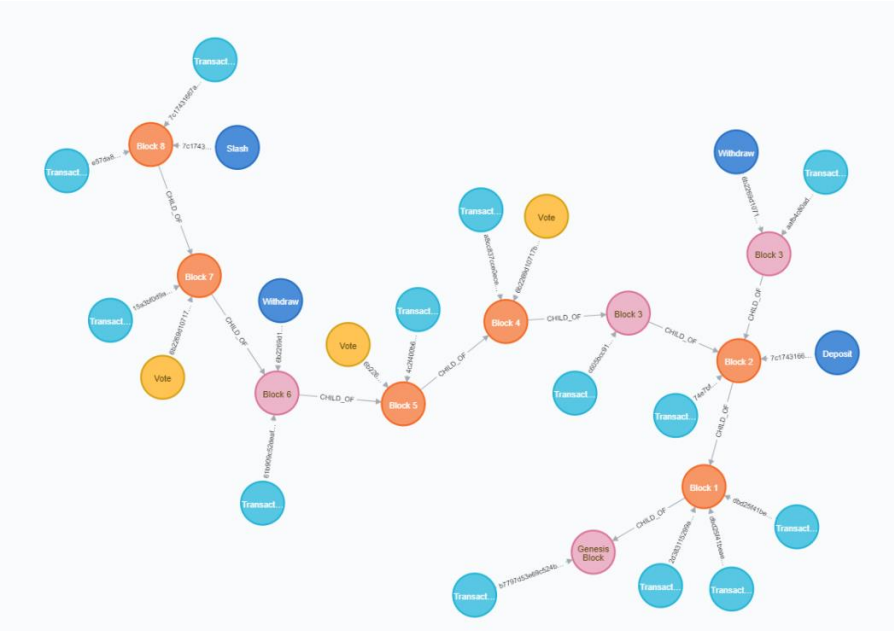


Εικόνα 5. 51 Στοιχεία μηνύματος αναφοράς

μέσω του http post αιτήματος:

```
"http://localhost:5001/submitreport" -H "Content-
Type:application/json" -X POST -d @report.txt
```

Το report εδώ πραγματοποιείται από τον User5001 και εξετάζεται κατά την δημιουργία του επόμενου block. Έτσι μετά την επέκταση του Block7 το blockchain είναι το ακόλουθο:



Εικόνα 5. 52 Blockchain με μήνυμα Slash

Στην εικόνα 5.48 βλέπουμε το μήνυμα Slash για τον validator User5000 και μία επιπλέον συναλλαγή (πέραν της καθιερωμένης επιβράβευσης για την κατασκευή του block), η οποία πρόκειται για την συναλλαγή επιβράβευσης του User5001, που απέστειλε την αναφορά. Οι ιδιότητες των δύο αυτών οντοτήτων παρουσιάζονται στην παρακάτω εικόνα:

Όσον αφορά τις δομές του νέου block μετά το slashing του User5000 αυτές διαμορφώνονται ως εξής:


```
{
  "current_front": {
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "current_rear": {},
  "front_validators_stake": 100,
  "front_votes": {
    "(0, 3)": 1.0,
    "(0, 6)": 0.010191072715323034,
    "(3, 4)": 1.0
  },
  "hash": "7f365b0cd9f2b4db9167870cca742a02cccbbba9d83b4ddc1421eba79f3e1d3c",
  "height": 8,
  "justified_checkpoints": [
    0,
    3
  ],
  "new_front": {
    "7c17431667aaea986520a59c3b197a20fcf731693156ff5159a8776311fb5402": 100
  },
  "previousCheckpoint": "a78ca86ca3b8c81ef7c0961f3597bc8b6d9356080759c6f4e80ed0c26eb1ffe4",
  "previousHash": "51117bcb2cb14e2dec3f49c0d7265b6961eb06b936d0f6c1263e70ebf50682f9",
  "rear_validators_stake": 0,
  "rear_votes": {
    "(0, 3)": 1.0,
    "(0, 6)": 1.0,
    "(3, 4)": 1.0
  },
  "slashed_validators": [
    "6b2269d10717bef211d4d1c1190f7991d8eee5f94c364c5f7bd2ac5e9e9dfec5"
  ],
  "stake": 0,
  "timestamp": "2019-07-18 00:58:33"
}
```


Εικόνα 5. 53 Αποθηκευμένη πληροφορία στο Block8 μετά το slashing

Παρατηρούμε ότι ο validator User5000 έχει αφαιρεθεί και από τα δύο validator sets στα οποία βρισκόταν. Επιπλέον έχει προστεθεί στη λίστα slashed_validators, με την οποία διασφαλίζουμε ότι όταν η ανάληψη που έχει καταχωρηθεί πραγματοποιηθεί, εκείνος δεν θα λάβει κανένα token.

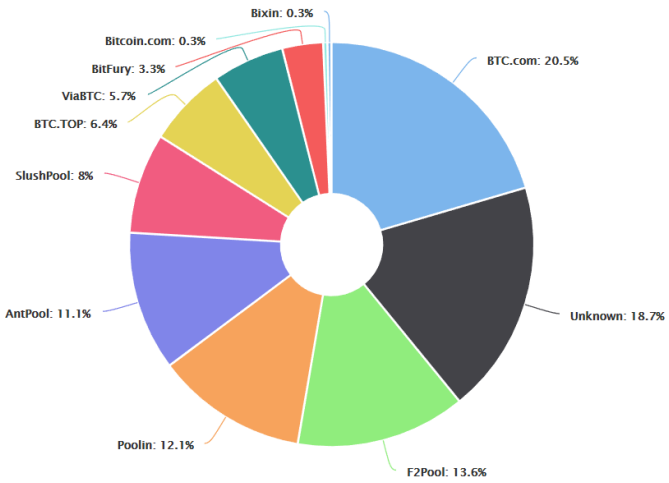
6
Επιθέσεις στο Casper Blockchain

Στο κεφάλαιο αυτό θα εξετάσουμε πιθανές επιθέσεις που μπορούν να πραγματοποιηθούν στο Casper blockchain και στο κατά πόσο οι μηχανισμοί άμυνας που διαθέτει το δίκτυό μας είναι ικανοί στο να τις αντιμετωπίσουν. Για τις μετρήσεις που έγιναν πάνω στο δίκτυο που κατασκευάσαμε χρησιμοποιήθηκαν .bat scripts που αυτοματοποίησαν την λειτουργία των κόμβων του δικτύου.

6.1 51% Attack

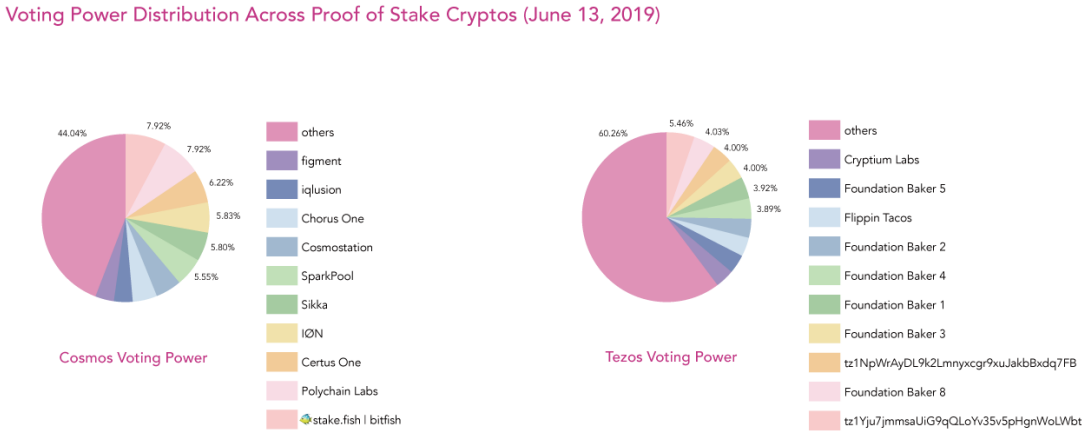
Η συγκεκριμένη επίθεση μπορεί να πραγματοποιηθεί από έναν ή περισσότερους miners οι οποίοι ελέγχουν τουλάχιστον το 51% του hashrate ενός δικτύου blockchain. Συγκεκριμένα, αν σε ένα PoW κρυπτονόμισμα ομάδα από miners συγκεντρώνει υπολογιστική δύναμη μεγαλύτερη από αυτήν του υπόλοιπου δικτύου, θα μπορεί να καθυστερήσει μελλοντικές συναλλαγές ή να αλλάξει άλλες ήδη καταχωρημένες (double-spending). Αυτό συμβαίνει, καθώς οι συγκεκριμένοι miners θα μπορούσαν να κατασκευάσουν μία αλυσίδα ελέγχοντας όλες τις συναλλαγές σε αυτήν. Έτσι αφού διαθέτουν μεγαλύτερη υπολογιστική δύναμη από το υπόλοιπο δίκτυο, επιλύουν τα blocks γρηγορότερα και η αλυσίδα τους καταλήγει να είναι η μεγαλύτερη σε μήκος και επομένως η «κανονική». Μία τέτοια οργανωμένη επίθεση θα είχε καταστροφικές συνέπειες για το δίκτυο και το ίδιο το κρυπτονόμισμα. Αντίστοιχη επίθεση μπορεί να πραγματοποιηθεί και σε ένα PoS κρυπτονόμισμα με τους επιτιθέμενους να διαθέτουν την πλειοψηφία των tokens του δικτύου, που ισοδυναμεί με την μεγαλύτερη πιθανότητα εκλογής.

Το πόσο πιθανή είναι μία τέτοια επίθεση εξαρτάται από το πώς διανέμεται το hashrate στο εκάστοτε κρυπτονόμισμα. Στα PoW πρωτόκολλα έχει διατυπωθεί το πρόβλημα της συγκέντρωσης του hashrate σε λίγους, κάτι που κυρίως αποδίδεται στην φύση του πρωτοκόλλου (ο πλούσιος γίνεται πλουσιότερος). Συγκεκριμένα στο bitcoin τρεις μόλις φάρμες (mining farms) συγκεντρώνουν πάνω απ’ το 51% του συνολικού hashrate.



Εικόνα 6. 1 Η κατανομή του hashrate στο Bitcoin

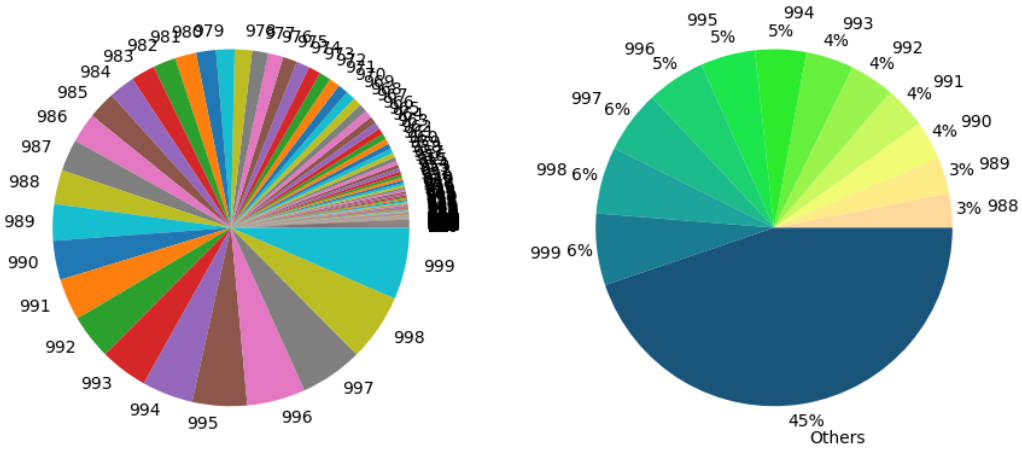
Αντίθετα τα PoS κρυπτονομίσματα δεν δείχνουν τόσο κεντροποιημένα με την πλειοψηφία των tokens να βρίσκονται κατανεμημένα σε πολλές διευθύνσεις και σε μικρότερες ποσότητες.



Εικόνα 6. 2 Κατανομές Voting-Power σε PoS κρυπτονομίσματα

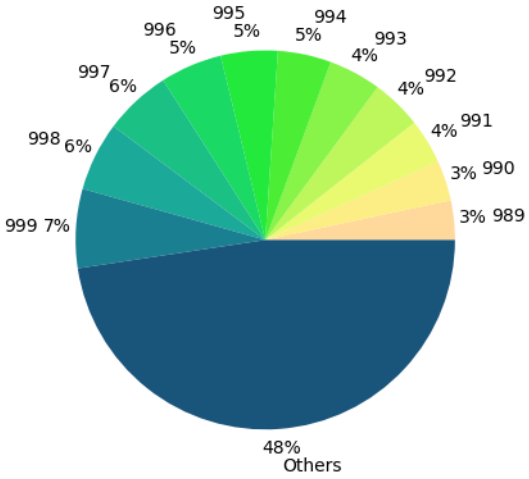
Το κατά πόσο το PoS ευνοεί την κεντρικοποίηση του Voting Power στους «ισχυρότερους» stakeholders μπορούμε να το ελέγξουμε χρησιμοποιώντας το σύστημα για PoS που εφαρμόσαμε στο δίκτυο μας. Σύμφωνα με αυτό οι stakeholders έχουν πιθανότητα εκλογής ανάλογη με το ποσοστό κεφαλαίου που διαθέτουν. Ο νικητής κόμβος θα ανταμείβεται με ένα σταθερό ποσό κρυπτονομισμάτων.

Αρχικοποιήσαμε 1000 κόμβους με την κατανομή voting power να είναι παρόμοια με αυτήν των πραγματικών PoS δικτύων. Η κατανομή του πλούτου παρουσιάζεται στην εικόνα 6.3 σε αναλυτική και απλοποιημένη μορφή με τη βοήθεια του εργαλείου της Python pyplot [].



Εικόνα 6. 3 Αρχική κατανομή Voting-Power

Εν συνεχεία ξεκινούμε τη διαδικασία εκλογής – επιβράβευσης για 1,000,000 blocks και ελέγχουμε την καινούργια κατανομή των tokens στους χρήστες. Η εικόνα που λαμβάνουμε είναι η παρακάτω:



Εικόνα 6. 4 Κατανομή Voting-Power μετά από 1.000.000 blocks

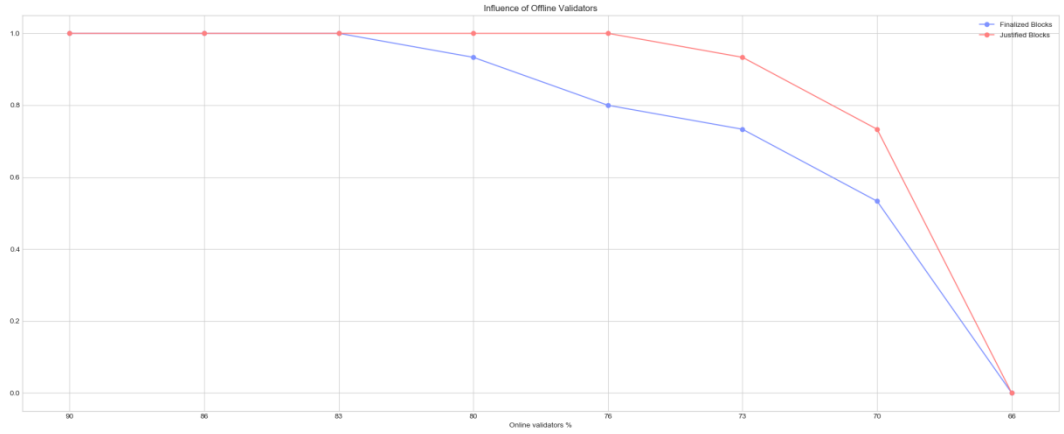
Όπως βλέπουμε οι διαφορές με την αρχική κατανομή είναι ελάχιστες, πράγμα που σημαίνει ότι το PoS που χρησιμοποιήσαμε διατηρεί τις οικονομικές διαφορές μεταξύ των κόμβων του δικτύου χωρίς να τις διευρύνει ποσοστιαία.

Όσον αφορά το blockchain ακόμη και στην περίπτωση που κάποιος ή κάποιοι ελέγχουν αποκλειστικά την πιο ισχυρή αλυσίδα συναλλαγών, η οριστικοποίηση της τελευταίας εξαρτάται αποκλειστικά από τους validators. Ουσιαστικά, μία επίθεση 51% στο Casper πραγματοποιείται από τους validators και όχι από του miners/stakeholders. Ωστόσο ούτε το 51% των validators δεν είναι ικανό να οριστικοποιήσει συναλλαγές. Όπως έχουμε εξηγήσει το finalization ενός block προϋποθέτει την συμφωνία τουλάχιστον του 66,66% των validators και των δύο sets. Έτσι ενώ το 51% των validators είναι ικανό να εμποδίσει την οριστικοποίηση νέων blocks, δεν αρκεί για την οριστικοποίηση μίας πλαστής «αλυσίδας».

6.2 Catastrophic Crashes

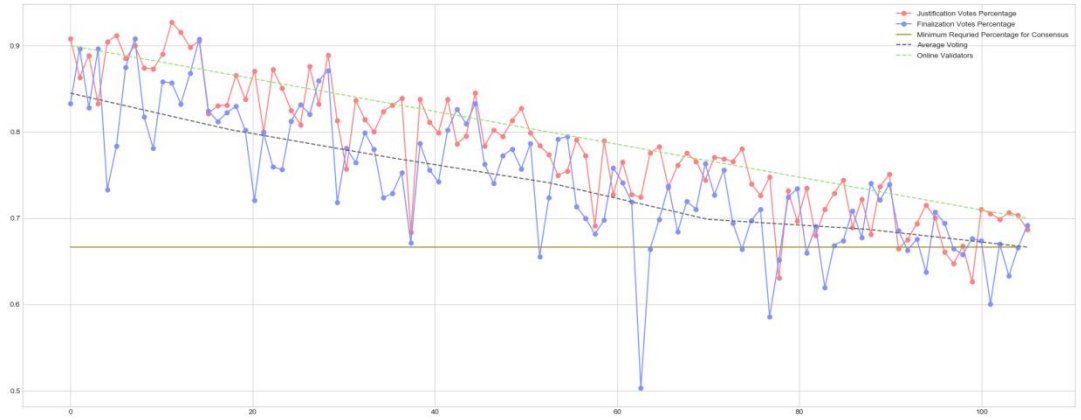
Στην περίπτωση που >1/3 των ενεργών validators αποσυνδεθεί από το δίκτυο λόγω αποτυχίας υπολογιστών, network partitioning ή επικίνδυνης συμπεριφοράς των τελευταίων, η οριστικοποίηση νέων blocks είναι πρακτικά αδύνατη. Όπως αναφέραμε σε προηγούμενο κεφάλαιο, το inactivity leak βοηθά στο να μπορέσει το δίκτυο να επανέλθει από μία τέτοια κατάσταση αυτή σε βάθος μερικών «εποχών». Με τον τρόπο αυτό οι ανενεργοί validator χάνουν μέρος των χρημάτων τους, ώστε η ισχύς των ψήφων τους να εξασθενεί με την πάροδο του χρόνου. Το Inactivity Leak μπορεί είτε να είναι σταθερό είτε κυμαινόμενο (π.χ. να αυξάνεται όσο δεν γίνονται finalized νέα blocks) και τα χρήματα που αφαιρούνται από τους ανενεργούς validators μπορεί είτε να χάνονται είτε να τους επιστρέφονται μετά από κάποιο χρονικό διάστημα αφού οι validators επιστρέψουν στο δίκτυο.

Στο δίκτυό μας εφαρμόσαμε ένα σταθερό Inactivity Leak, με τα tokens να χάνονται από τους ανενεργούς validators με την πάροδο του χρόνου. Μπορούμε τώρα να εξετάσουμε την επιρροή των ανενεργών validators στην οριστικοποίηση νέων blocks, καθώς και το χρόνο επαναφοράς του δικτύου από ένα catastrophic crash για το inactivity leak που χρησιμοποιήσαμε. Για τον παραπάνω σκοπό θα ορίσουμε δύο set validators και μειώνοντας σταδιακά του ενεργούς validators θα μετρήσουμε το ποσοστό των δικαιολογημένων και οριστικοποιημένων checkpoints επί των συνολικών checkpoints. Προκειμένου τα αποτελέσματα να προσεγγίζουν καλύτερα αυτά ενός πραγματικού δικτύου blockchain, θεωρήσαμε μία πιθανότητα 2% ένα απεσταλμένο μήνυμα ψήφου να αποτύχει (λανθασμένη ψήφος, αποτυχία δικτύου κλπ.) και 7% πιθανότητα μία ψήφος οριστικοποίησης να αποτύχει, λόγω εκπρόθεσμης αποστολής. Για τη συγκεκριμένη μέτρηση θέσαμε τα INACTIVITY_LEAK και VOTE_PROFITE στη μονάδα ώστε να μελετηθεί αποκλειστικά η επίδραση των απενεργοποιημένων κόμβων στην επίτευξη της συναίνεσης. Εκτελώντας την επαναληπτική διαδικασία με τα παραπάνω δεδομένα στο δίκτυο μας, λαμβάνουμε το παρακάτω γράφημα.



Εικόνα 6. 5 Επιρροή των αποσυνδεδεμένων validators στην επίτευξη συναίνεσης του δικτύου μας

Βλέπουμε ότι η συναίνεση είναι αδύνατη με το 1/3 των validators να είναι αποσυνδεδεμένοι. Τα αναλυτικά ποσόστα ψήφων που έλαβε το κάθε ckeckpoint στην παραπάνω εκτέλεση παρουσίαζονται παρακάτω.

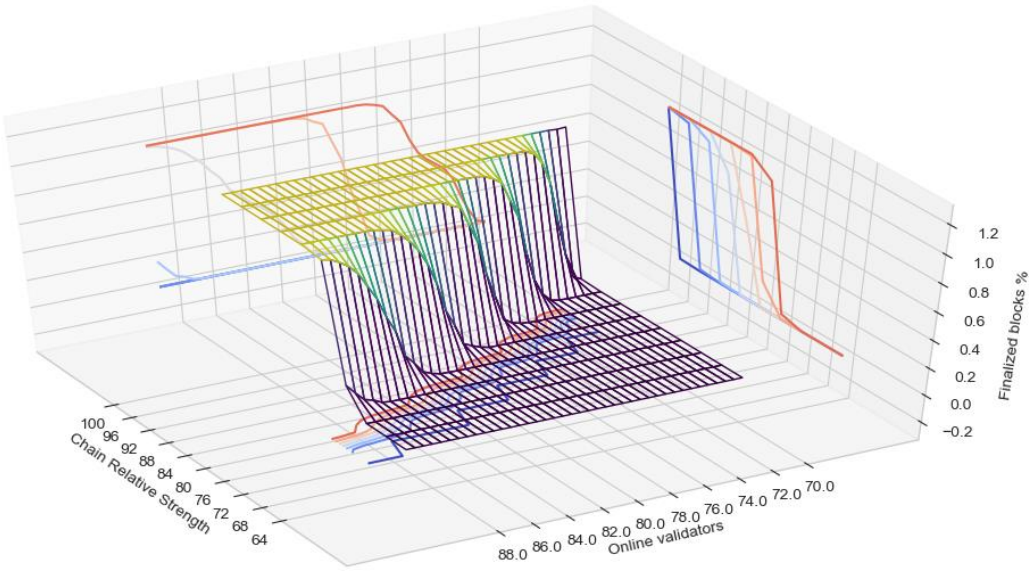


Εικόνα 6. 6 Ποσοστό συναίνεσης σε justification και finalization ανά checkpoint

Όπως παρατηρούμε ο τυχαίος παράγοντας που αφορά την επιτυχία ή όχι μίας απεσταλμένης ψήφου μπορεί να διαφοροποιήσει ελαφρώς τα ποσοστά που λαμβάνουν τα checkpoints. Ωστόσο, παρατηρώντας τον μέσο όρο συναίνεσης ανά 15 checkpoints (όπου πραγματοποιείται η μείωση του αριθμού των ενεργών validators), αντιλαμβανόμαστε ότι ο πρώτος ακολουθεί παράλληλη πορεία με αυτή του ποσοστού των ενεργών validators στο δίκτυο.

Εάν τώρα λάβουμε υπόψη μας την ύπαρξη επιπλέον αλυσίδων στο δίκτυο, μπορούμε να θεωρήσουμε την πιθανότητα να ψηφιστεί ένα checkpoint σε ύψος X ανάλογη με τη δυναμική του, σε σύγκριση πάντα με αυτή των υπόλοιπων checkpoints στο ίδιο ύψος X. Η δυναμική ενός checkpoint (και μίας αλυσίδας) εξαρτάται σε μεγάλο βαθμό από το πρωτόκολλο και συγκεκριμένα αποτελεί το μέτρο δυσκολίας κατασκευής του πρώτου. Έτσι για ένα PoS δίκτυο με τρία υποψήφια checkpoints στο ίδιο ύψος και block staking 50 tokens, 20 tokens , 30 tokens μπορούμε να θεωρήσουμε πιθανότητες 50%, 20% και 30% τα αντίστοιχα blocks να λάβουν ψήφο.

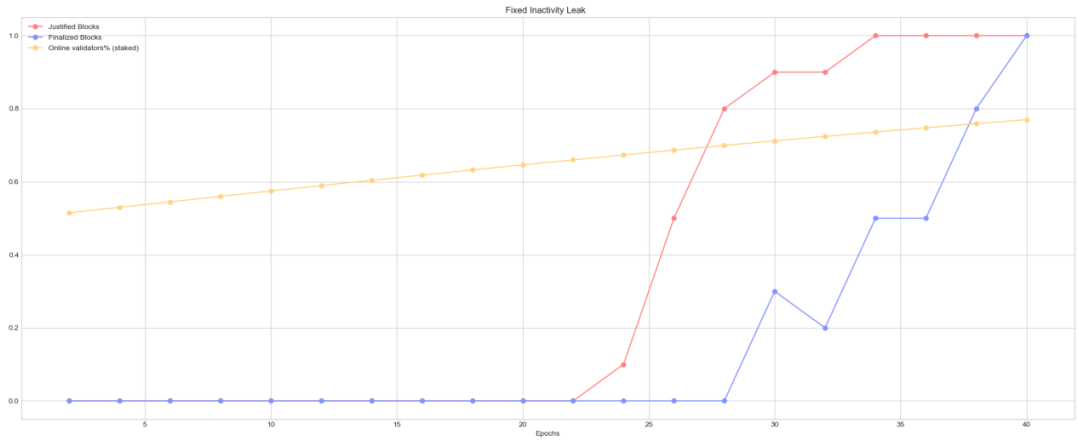
Για να ελέγξουμε την επιρροή του επιπλέον αυτού παράγοντα στη συναίνεση των Validators κατασκευάσαμε μία εφαρμογή σε Python που προσομοιώνει τη διαδικασία ψηφοφορίας του Casper και μας επιτρέπει να ελέγξουμε απόλυτα τη δυναμική της κύριας αλυσίδας. Με τον τρόπο αυτό λαμβάνουμε το 3D γράφημα της εικόνας 6.7, στο οποίο μπορούμε να δούμε το ποσοστό οριστικοποιημένων checkpoints συναρτήσει τόσο των online validators όσο και της σχετικής δυναμικής του block.



Εικόνα 6. 7 Ποσοστό οριστικοποιημένων checkpoints συναρτήσει δύο παραγόντων

Όπως ήταν αναμενόμενο βλέπουμε πως η δυναμική της κύριας αλυσίδας επηρεάζει τον ελάχιστο αριθμό validators που απαιτείται για την επίτευξη συναίνεσης και το αντίστροφο.

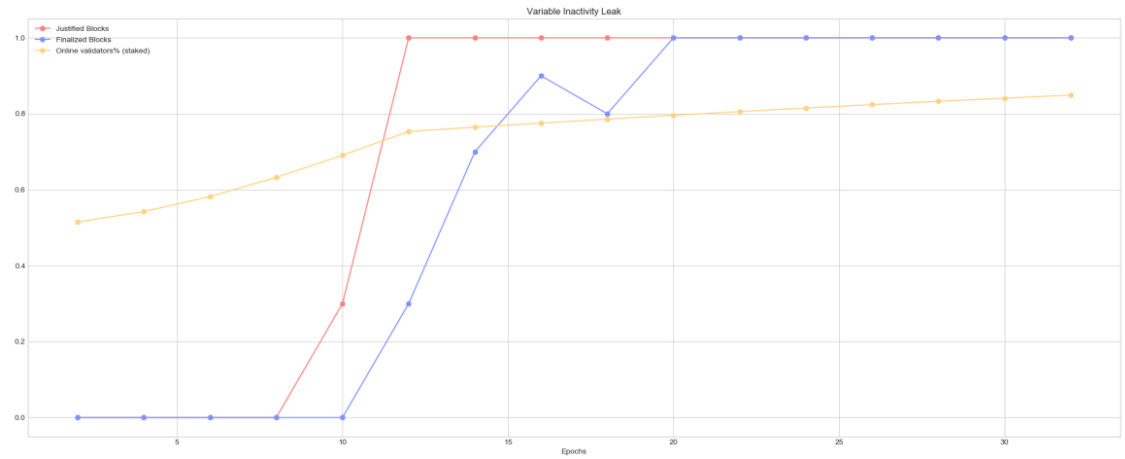
Τέλος εξετάσαμε την αποτελεσματικότητα που έχουν τα Inactivity Leak και Vote Reward στο δίκτυό μας. Συγκεκριμένα εξετάζουμε το χρόνο επαναφοράς του δικτύου όσον αφορά την επίτευξη της συναίνεσης έπειτα από ένα Catastrophic Crash. Στο γράφημα 6.7 παρουσιάζονται αυτές οι μετρήσεις για σταθερό INACTIVITY_LEAK = -1% και VOTE_PROFIT = +4% που εφαρμόζεται ανά 10 checkpoints. Στην αρχική κατάσταση το 50% των validators είναι ενεργοί, ενώ πιθανότητες αποτυχίας για τις απεσταλμένες ψήφους είναι οι ίδιες με προηγουμένως.



Εικόνα 6. 8 Συναίνεση σε δίκτυο με σταθερό Inactivity Leak

Στο παραπάνω διάγραμμα παρατηρούμε ότι η επαναφορά του δικτύου από την αρχική κατάσταση απαιτεί 300 checkpoints. Μάλιστα η ύπαρξη μίας μόνο κανονικής αλυσίδας καθιστά σχεδόν άμεσο το justification του πρώτου checkpoint, από τη στιγμή που οι online validators αποκτήσουν το 66% του συνολικού stake. Ωστόσο, το Inactivity Leak έχει σκοπό τόσο την παρακίνηση των validators για συμμετοχή στην διαδικασία συναίνεσης, όσο και την επαναφορά του δικτύου από μία ένα Catastrophic Crash. Για τον πρώτο σκοπό προτιμάται μία μικρή ποινή, ώστε να μην είναι ιδιαίτερα δαπανηρή μία σύντομη αποχή ενός validator από την διαδικασία ψηφοφορίας. Αντίθετα, στην δεύτερη περίπτωση θέλουμε η επαναφορά του δικτύου και η οριστικοποίηση νέων συναλλαγών να γίνεται όσο το δυνατόν γρηγορότερα, κάτι που προϋποθέτει την μείωση των stakes των ανενεργών validators με μεγαλύτερο ρυθμό. Για να ικανοποιήσουμε και τις δύο περιπτώσεις εφαρμόζουμε ένα κυμαινόμενο Inactivity Leak ανάλογα με το κατά πόσο είναι εφικτή η συναίνεση στο δίκτυο μας.

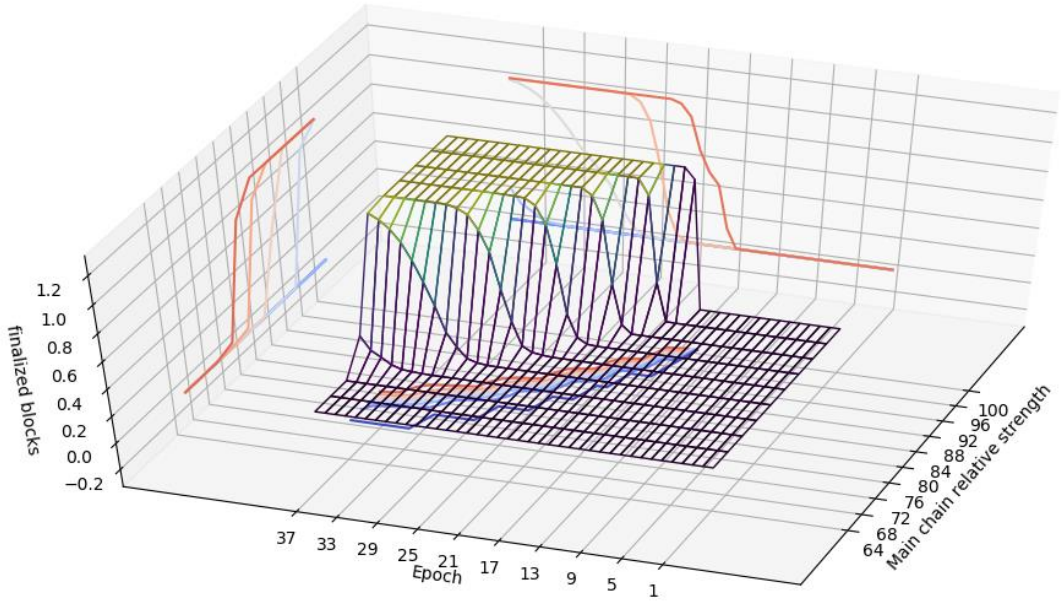
Στο παρακάτω γράφημα επαναλαμβάνουμε τις μετρήσεις του 6.8 για την ίδια αρχική κατάσταση και αρχικές τιμές VOTE_PROFIT και INACTIVITY_LEAK με το τελευταίο όμως να μειώνεται με ρυθμό 5% ανά 10 checkpoints, στα οποία δεν επιτεύχθηκε η συναίνεση. Αντίστοιχα με την οριστικοποίηση του πρώτου checkpoint επαναφέρουμε το INACTIVITY_LEAK στην αρχική του τιμή. Οι μετρήσεις που λάβαμε σε αυτήν την περίπτωση είναι οι ακόλουθες.



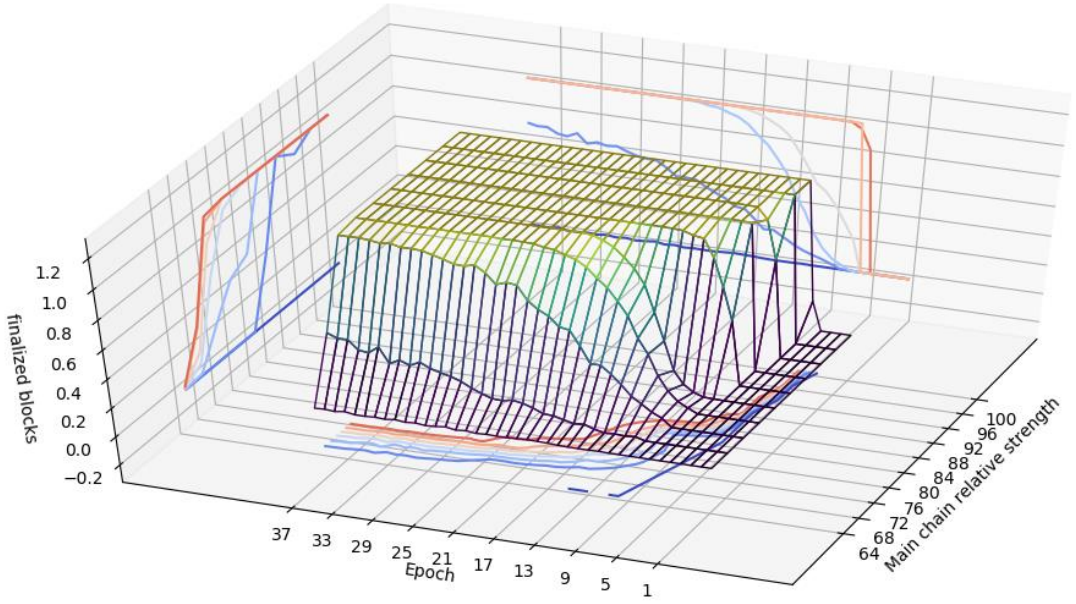
Εικόνα 6. 9 Συναίνεση σε δίκτυο με κυμαινόμενο Inactivity Leak

Όπως ήταν αναμενόμενο, η οριστικοποίηση του πρώτου checkpoint συνέβη συντομότερα σχέση με πριν, σε μόλις 120 checkpoints. Ωστόσο από την στιγμή της επίτευξης συναίνεσης, ο ρυθμός αύξησης του ποσοστού των online validators μειώθηκε, αποτρέποντας έτσι την εκτεταμένη μείωση των καταθέσεων των ανενεργών χρηστών.

Επιπλέον, ενώ η ύπαρξη πολλών αλυσίδων δυσκολεύει την συναίνεση σε μία κανονική αλυσίδα, στην τελευταία περίπτωση έχει μικρότερη επιρροή στην ταχύτητα επαναφοράς του δικτύου. Αυτό συμβαίνει, αφού η μη επίτευξη της συναίνεσης σημαίνει την περαιτέρω μείωση της επιρροής των ανενεργών validators στη διαδικασία ψηφοφορίας. Εκτελώντας μία προσομοίωση της διαδικασίας ψηφοφορίας του Casper, τόσο για το σταθερό όσο και για το κυμαινόμενο Inactivity Leak των προηγούμενων γραφημάτων, λαμβάνουμε τις παρακάτω τρισδιάστατες απεικονίσεις της ταχύτητας επίτευξης συναίνεσης σε αλυσίδες οποιασδήποτε δυναμικής, για κάθε περίπτωση.



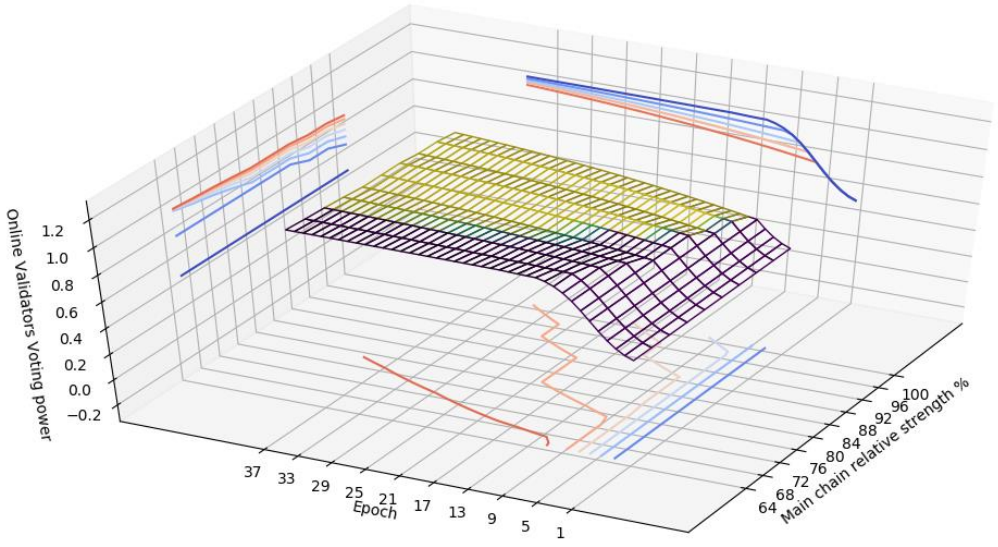
Εικόνα 6. 10 Συναίνεση σε δίκτυο με σταθερό Inactivity Leak για διαφορετικής δυναμικής κύρια αλυσίδα



Εικόνα 6. 11 Συναίνεση σε δίκτυο με κυμαινόμενο Inactivity Leak για διαφορετικής δυναμικής κύρια αλυσίδα

Παρατηρώντας τα παραπάνω αποτελέσματα, βλέπουμε ότι με σταθερό Inactivity Leak η δυναμική της αλυσίδας καθορίζει σε μεγάλο βαθμό την ταχύτητα επαναφοράς του δικτύου σε κατάσταση συναίνεσης. Αντίθετα, το δίκτυο με κυμαινόμενο Inactivity Leak φαίνεται να μην επηρεάζεται ιδιαίτερα από την ύπαρξη άλλων διακλαδώσεων με το πρώτο checkpoint να οριστικοποιείται κοντά στην 7η εποχή σε κάθε περίπτωση.

Ουσιαστικά μπορούμε να πούμε ότι το κυμαινόμενο Inactivity Leak μετατρέπει την εξάρτηση Chain Relative Strength – Epoch of Consensus σε εξάρτηση Chain Relative Strenght – Online Validators per Epoch. Αυτό μπορεί να φανεί και στο παρακάτω γράφημα, όπου παρουσιάζεται το ποσοστό των ενεργών validators ανά εποχή για κύριες αλυσίδες διαφορετικής δυναμικής.



Εικόνα 6. 12 Εξάρτηση μεταβολής του ποσοστού ενεργών validators από την δυναμική της κύριας αλυσίδας σε δίκτυα με κυμαινόμενο Inactivity Leak.

7
Επίλογος

7.1 Σύνοψη και συμπεράσματα

Η παρούσα διπλωματική εστίασε στην ανάπτυξη και την αξιολόγηση σύγχρονων μηχανισμών συναίνεσης που εφαρμόζονται σήμερα, στην blockchain τεχνολογία. Για το σκοπό αυτό, δημιουργήσαμε μία αποκεντρωμένη εφαρμογή σε Python, στην οποία και ενσωματώσαμε την βάση δεδομένων γράφου Neo4j. Οι υψηλές ταχύτητες που προσφέρει η Neo4j στην εξυπηρέτηση αιτημάτων, κατέστησε πιο εύκολη την διεκπεραίωση εργασιών που απαιτούν την σειριακή προσπέλαση ολόκληρου του blockchain. Επιπλέον, η καλή οπτική αναπαράσταση των δεδομένων της βάσης Neo4j, βοήθησαν στην καλύτερη κατανόηση της δομής του blockchain και της λειτουργίας του μηχανισμού συναίνεσης Casper.

Το Casper αποτελεί ένα μηχανισμό συναίνεσης, που συνδυάζει στοιχεία BFT και Proof of Stake, με σκοπό η συναίνεση των χρηστών να επιτυγχάνεται ανέξοδα και με ασφάλεια. Στην blockchain εφαρμογή που κατασκευάσαμε, εφαρμόσαμε το Casper και διατηρώντας ανέπαφα τα θεμελιώδη χαρακτηριστικά του, πειραματιστήκαμε με κάποιες από τις λειτουργίες του. Συγκεκριμένα, επεκταθήκαμε στα κριτήρια voting των validators τα οποία διαφοροποιούνται σημαντικά, ανάλογα με το πρωτόκολλο σύστασης block που χρησιμοποιείται. Επιπλέον, προτείναμε έναν εναλλακτικό τρόπο αξιολόγησης της συνεισφοράς ενός validator στην διαδικασία ψηφοφορίας, για την πιο δίκαιη απόδοση ανταμοιβών και επιβολή τιμωριών στους validators.

Τέλος, εξετάσαμε την αντοχή της εφαρμογής μας σε διάφορα είδη επιθέσεων. Με τη χρήση προσομοιώσεων διαπιστώσαμε ότι το hashrate τόσο των Proof of Stake stakeholders όσο και των Casper validators της εφαρμογής μας δεν τείνει κεντροποιηθεί, πράγμα που καθιστά μία 51% επίθεση σχεδόν αδύνατη. Στη συνέχεια εξετάσαμε την επιρροή των offline validators στη λειτουργία του Casper και την αποτελεσματικότητα των μηχανισμών του Casper για την επαναφορά του δικτύου σε κατάσταση συναίνεσης. Συγκεκριμένα, μελετήσαμε το χρόνο που απαιτείται, μετά από ένα catastrophic crash για την οριστικοποίηση συναλλαγών, για διαφορετικούς τρόπους τιμωρίας των ανενεργών validators και αλυσίδες διαφορετικής δυναμικής.

7.2 Μελλοντικές επεκτάσεις

Οι τεχνολογίες της εφαρμογής που αναπτύξαμε είναι πολύ πρόσφατες και διαθέτουν πολλά περιθώρια για έρευνα και εξέλιξη. Ιδιαίτερα χρήσιμη είναι η μελέτη του Casper με διαφορετικά πρωτόκολλα σύστασης blocks, όπως και με συνδυασμό αυτών, για την επίτευξη καλύτερης αποδοτικότητας στην επίτευξη της συναίνεσης. Εξίσου σημαντική είναι η επίλυση καταστάσεων ασυγχρονισμού που μπορούν να προκύψουν σε συγκεκριμένες περιπτώσεις κατά την ταυτόχρονη διαγραφή μεγάλου αριθμού παραβατών-validators. Ασφαλώς η ανάπτυξη public blockchains που χρησιμοποιούν το Casper είναι απαραίτητη, για την καλύτερη αξιολόγηση του τελευταίου σε πραγματικές συνθήκες.

Η περαιτέρω χρήση της Neo4j σε εφαρμογές blockchain, μπορεί να αποτελέσει λύση σε πολλά σε προβλήματα ταχύτητας που παρουσιάζει η τεχνολογία ή ακόμα να βοηθήσει στην μελέτη των blockchain εφαρμογών με την εξυπηρέτηση σύνθετων αιτημάτων σε αυτές. Όσον αφορά την εφαρμογή που κατασκευάσαμε, μπορούμε να επεκταθούμε στην υλοποίηση και την μελέτη νέων τεχνολογιών στο blockchain. Ένα τέτοιο παράδειγμα είναι η κατασκευή του Lighting Network, είναι πρωτόκολλο πληρωμής που λειτουργεί σαν ένα δεύτερο επίπεδο στο blockchain και επιτρέπει την off-chain πραγματοποίηση πολλών συναλλαγών μικρής αξίας μεταξύ δύο χρηστών, αντιμετωπίζοντας έτσι τα προβλήματα scalability της τεχνολογίας. Ιδιαίτερο ενδιαφέρον παρουσιάζει η ενσωμάτωση της Neo4j σε αυτό, για την γρήγορη αναζήτηση πληροφοριών σε μεγάλο όγκο δεδομένων.

8
Βιβλιογραφία

[1] Satoshi Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System» Available: https://bitcoin.org/bitcoin.pdf [Πρόσβαση 10 10 2019]

[2] Vitalik Buterin and Virgil Griffith «Casper the Friendly Finality Gadget» Available: https://arxiv.org/pdf/1710.09437.pdf [Πρόσβαση 10 10 2019]

[3] J. Kurose και K. Ross, Δικτύωση Υπολογιστών, Αθήνα: M. Γκιούρδας.

[4] «Tokenized Networks: Web3, the Stateful Web» [Ηλεκτρονικό] Available: https://blockchainhub.net/web3-decentralized-web/ web3 [Πρόσβαση 10 10 2019]

[5] R. Schollmeier, «A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications» [Ηλεκτρονικό] Available: https://www.researchgate.net/publication/3940901 [Πρόσβαση 10 10 2019]

[6] «Client-Server Model» [Ηλεκτρονικό] Available: https://www.sciencedirect.com/topics/computer-science/client-server-model [Πρόσβαση 10 10 2019]

[7] K. Dooley, «Designing Large-scale LANs» – Page 31, O'Reilly Media.

[8] C.Buragohain, D.Agrawal, S.Suri, «A Game Theoretic Framework for Incentives in P2P Systems» [Ηλεκτρονικό] Available: https://arxiv.org/pdf/cs/0310039.pdf [Πρόσβαση 10 10 2019]

[9] G. Zyskind, O. Nathan, A. Pentland, «Decentralizing Privacy: Using Blockchain to Protect Personal Data» [Ηλεκτρονικό] Available: https://homepage.divms.uiowa.edu/~ghosh/blockchain.pdf [Πρόσβαση 10 10 2019]

[10] Stuart Haber, W. Scott Stornetta, «How to Time-Stamp a Digital Document» [Ηλεκτρονικό]
Available: https://www.anf.es/pdf/Haber_Stornetta.pdf [Πρόσβαση 10 10 2019]

[11] Bitcoin Wiki, «Transactions» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Transaction [Πρόσβαση 10 10 2019]

[12] Bitcoin Wiki, «Mining» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Mining [Πρόσβαση 10 10 2019]

[13] Bitcoin Wiki, «Controlled Supply» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Controlled_supply [Πρόσβαση 10 10 2019]

[14] Bitcoin Wiki, «Private Key» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Private_key [Πρόσβαση 10 10 2019]

[15] Bitcoin Wiki, «Proof of Work» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Proof_of_work [Πρόσβαση 10 10 2019]

[16] Proof of Stake [Ηλεκτρονικό] Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ#what-is-proof-of-stake [Πρόσβαση 10 10 2019]

[17] L. Lamport, R. Shostak, M Pease, «The Byzantine Generals Problem» [Ηλεκτρονικό]
Available: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf [Πρόσβαση 10 10 2019]

[18] «Byzantine Consensus Through Bitcoin's Proof-of-Work» [Ηλεκτρονικό] Available:
https://pdfs.semanticscholar.org/0d82/a314dd96265ac8542a9ca80e5c7db09594d2.pdf [Πρόσβαση 10 10 2019]

[19] «The energy consumption of the crypto world» [Ηλεκτρονικό] Available: https://medium.com/thebeammagazine/the-energy-consumption-of-the-crypto-world-b20e3628e0d2 [Πρόσβαση 10 10 2019]

[20] V. Buterin, «Nothing-at-Stake» [Ηλεκτρονικό] Available: https://blog.ethereum.org/2014/07/05/stake/?source=post_page [Πρόσβαση 10 10 2019]

[21] «Casper: A short description for validators» [Ηλεκτρονικό] Available: https://medium.com/@theNKBGroup/ethereum-releases-casper-v0-1-a-short-description-for-validators-3e0a7676d286 [Πρόσβαση 10 10 2019]

[22] Bitcoin Wiki, «Genesis Block» [Ηλεκτρονικό] Available: https://en.bitcoin.it/wiki/Genesis_block [Πρόσβαση 10 10 2019]

[23] «Casper Fork Choice Rule» [Ηλεκτρονικό] Available: https://ethresear.ch/t/immediate-message-driven-ghost-as-ffg-branch-choice-rule/2561 [Πρόσβαση 10 10 2019]

[24] V. Buterin, «Safety Under Dynamic Validator Sets» [Ηλεκτρονικό] Available: https://medium.com/@VitalikButerin/safety-under-dynamic-validator-sets-ef0c3bbdf9f6 [Πρόσβαση 10 10 2019]

[25] O. Moindrot, C. Bournhonesque, «Proof of Stake Made Simple with Casper» [Ηλεκτρονικό]

Available: https://www.scs.stanford.edu/17au-cs244b/labs/projects/moindrot_bournhonesque.pdf [Πρόσβαση 10 10 2019]

[26] C. Cachin, M. Vukoli´c «Blockchain Consensus Protocols in the Wild» [Ηλεκτρονικό]
Available: https://arxiv.org/pdf/1707.01873.pdf [Πρόσβαση 10 10 2019]

[27] I. Bentov, C. Lee, A. Mizrahi, M. Rosenfeld, «Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake» [Ηλεκτρονικό]
Available: https://eprint.iacr.org/2014/452.pdf [Πρόσβαση 10 10 2019]

[28] M. Sharples, J. Domingue, «The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward» [Ηλεκτρονικό]
Available: https://link.springer.com/chapter/10.1007/978-3-319-45153-4_48 [Πρόσβαση 10 10 2019]

[29] John V. Guttag, «Introduction to Computation and Programming Using Python: With Application to Understanding Data», The MIT Press .

[30] Roy Thomas Fielding, «Architectural Styles and the Design of Network-based Software Architectures – Chapter 5: Representational State Transfer (REST)» [Ηλεκτρονικό]
Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm [Πρόσβαση 10 10 2019]

[31] «RESTful API» [Ηλεκτρονικό]
Available: https://searchapparchitecture.techtarget.com/definition/RESTful-API [Πρόσβαση 10 10 2019]

[32] «Introducing JSON» [Ηλεκτρονικό]
Available: https://www.json.org/ [Πρόσβαση 10 10 2019]

[33] «Flask» [Ηλεκτρονικό]
Available: https://palletsprojects.com/p/flask/ [Πρόσβαση 10 10 2019]

[34] «What Is Neo4j?» [Ηλεκτρονικό]
Available: https://neo4j.com/neo4j-graph-database/?ref=home-banner/ [Πρόσβαση 10 10 2019]

[35] «The Neo4j Cypher Manual» [Ηλεκτρονικό]
Available: https://neo4j.com/docs/cypher-manual/current/ [Πρόσβαση 10 10 2019]

[36] «About Node.js» [Ηλεκτρονικό]
Available: https://nodejs.org/en/about/ [Πρόσβαση 10 10 2019]

[37] «Java Native Interface (JNI)» [Ηλεκτρονικό]
Available:
https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html [Πρόσβαση 10 10 2019]

[38] «Using Neo4j from Python» [Ηλεκτρονικό]
Available: https://neo4j.com/developer/python/ [Πρόσβαση 10 10 2019]

[39] Neo4j Desktop Application [Ηλεκτρονικό] Available: https://neo4j.com/developer/neo4j-desktop/ [Πρόσβαση 10 10 2019]

[40] D. Khovratovich, C. Rechberger A. Savelieva, «Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family» [Ηλεκτρονικό] Available: https://eprint.iacr.org/2011/286.pdf [Πρόσβαση 10 10 2019]

[41] «hashlib - Secure hashes and message digests» [Ηλεκτρονικό] Available: https://docs.python.org/2/library/hashlib.html [Πρόσβαση 10 10 2019]