ΕΘΝΙΚΌ ΜΕΤΣΌΒΙΟ ΠΟΛΥΤΕΧΝΕΊΟ

Σχολή Ηλεκτρολόγων Μηχανικών Μηχανικών Υπολογιστών

Κατανεμημένα Συστήματα

Εξαμηνιαία Εργασία

Βασίλειος Λυμπεράτος	03115034
Βασίλειος Αναγνωστούλης	03114790
Κωνσταντίνος Τερζής	03114844

Ακαδημαικό έτος 2019-2020

Τελική Ημερομηνία Παράδοσης: 03/04/20

ΠΕΡΙΓΡΑΦΗ

Αρχικά, δημιουργείται ο bootstrap κόμβος και στη συνέχεια προστίθενται και οι υπόλοιποι. Έχουμε ως δεδομένο ότι οι κόμβοι συνολικά θα είναι 5 το πλήθος. Ο κάθε ένας δημιουργεί ένα πορτοφόλι, δηλαδή λαμβάνει ένα public και ένα private key. Γνωστοποιεί στον bootstrap το public key, ο οποίος με τη σειρά του στέλνει σε όλους τους κόμβους τα public keys των υπολοίπων. Οι κόμβοι συνδέονται όλοι στην αρχή της εκτέλεσης, και μόνο τότε μπορούν να ξεκινήσουν τα transactions.

Αν ο κόμβος Α θέλει να στείλει noob coins στον κόμβο Β, τότε δημιουργεί ένα αντικείμενο της κλάσης Transactions, καθορίζοντας το public key του παραλήπτη και το ποσό που επιθυμεί να μεταφέρει. Πρώτα όμως, διασχίζει τη δική του λίστα με τα UTXOs και βεβαιώνεται ότι υπάρχουν αρκετά TransactionOutputs που να καλύπτουν το ποσό που θα μεταφερθεί. Αφαιρεί τις εγγραφές αυτές που τον περιλαμβάνουν από τη λίστα των UTXOs, και τις περνάει σαν parentTransaction στο καινούργιο transaction object που δημιουργεί, και σαν transactionOutput δημιουργεί ένα αντικείμενο που φέρει τη διεύθυνση του παραλήπτη και το ποσό που του μεταφέρει. Ακολούθως υπογράφει ολόκληρο το αντικείμενο με το ιδιωτικό του κλειδί και κάνει broadcast transaction σε όλο το δίκτυο. Οι κόμβοι που λαμβάνουν το μήνυμα, δεδομένου ότι σύμφωνα με την εκφώνηση της άσκησης είναι όλοι miners, κατά την παραλαβή του μηνύματος καλούν την συνάρτηση validate_transaction. Συγκεκριμένα, ελέγχουν την ορθότητα της υπογραφής και ελέγχουν στην προσωπική τους λίστα με τα UTXOs εάν υπάρχουν έξοδοι από προηγούμενα transactions που να δικαιολογούν το ποσό που ο Α θέλει να μεταφέρει στον Β. Με τη διαδικασία αυτή αντιμετωπίζουμε το double spending. Εάν επιβεβαιωθεί η ορθότητα της συναλλαγής, τότε δημιουργούνται δύο αντικείμενα TransactionOutput, ένα που περιλαμβάνει το ποσό που μεταφέρθηκε στον παραλήπτη Β και ένα που περιλαμβάνει το άθροισμα των UTXOs του Α,μείον το ποσό που δαπανήθηκε. Η διαδικασία αυτή βέβαια δεν συνεπάγεται ότι κρατιέται “balance” για κάθε wallet. Τα transactions που λαμβάνουν οι κόμβοι από τους υπόλοιπους τα κάνουν append στο προσωπικό τους transaction_pool. Αφού έχει γίνει validate, το transaction προστίθεται στο τρέχον block.

Μόλις μαζευτεί ικανοποιητικός αριθμός από transactions, στην περίπτωση της άσκησης 10, ο κάθε κόμβος κάνει hash το block και ξεκινά να κάνει mining. Συγκεκριμένα, πρόκειται για το proof-of work. Δηλαδή, αφού έχει διαμορφώσει το block σε μορφή dictionary, αναζητά ένα nonce, έναν αριθμό δηλαδή τον οποίο αν προσθέσει στο μήνυμα που φέρει το dictionary και περάσει το συνδυασμό τους από τη hash function, να δώσει συγκεκριμένο αριθμό μηδενικών στην έξοδό της. Στην άσκηση ορίζουμε ως difficulty, δηλαδή τον αριθμό των μηδενικών που θέλουμε να προηγούνται σε τέσσερα. Το κάθε ένα φέρει το hash του προηγούμενου block στο chain για λόγους διασφάλισης εγκυρότητας. Μόλις έχει βρεθεί το nonce του block γίνεται broadcast σε όλο το δίκτυο. Το πρώτο block δημιουργείται από τον bootstrap node, ο οποίος μεταφέρει 100 noob coins σε όλους τους κόμβους. Είναι το μοναδικό που δεν επαληθεύεται από τους κόμβους και διαθέτει σαν αύξοντα αριθμό το 1, και σαν nonce το 0. Στη συνέχεια οι κόμβοι διατηρούν το δικό τους αντίγραφο για την αλυσίδα. Ωστόσο, είναι πιθανό αν δύο ή περισσότεροι κόμβοι προλάβουν να κάνουν mine σχεδόν ταυτόχρονα, να γίνουν broadcast διαφορετικά blocks στο δίκτυο και τελικά οι κόμβοι να έχουν μεταξύ τους διαφορετικές εκδοχές του chain, καθώς όλοι προσθέτουν στην αλυσίδα το πρώτο block που λαμβάνουν. Η συγκεκριμένη περίπτωση αποτελεί το φαινόμενο των conflicts.

CLI

- t <recipient address> <amount> : ο client στέλνει <amount> noob coins στη <recipient address>.

- view : τυπώνει τα transactions που περιέχονται στο τελευταίο επικυρωμένο block.

- balance : τυπώνει το υπόλοιπο του wallet του κόμβου..

- help : επεξηγεί τις παραπάνω εντολές.

ΠΕΙΡΑΜΑΤΑ

MaxTransactions = 5 Difficulty = 4			
	Average Mine Time Per Block (s)	**Total Time (s)**	**Throughput (trans/s)**
Node 0	4.108	394.319	1.268
Node 1	3.631	395.765	1.263
Node 2	4.617	393.162	1.271
Node 3	3.465	395.744	1.263
Node 4	4.127	393.146	1.271

MaxTransactions = 1 Difficulty = 4			
	Average Mine Time Per Block (s)	**Total Time (s)**	**Throughput (trans/s)**
Node 0	2.613	5278.453	0.094
Node 1	2.662	5281.049	0.094
Node 2	1.780	5268.328	0.094
Node 3	2.027	5269.432	0.094
Node 4	1.964	5269.446	0.094

MaxTransactions = 10 Difficulty = 4			
	Average Mine Time Per Block(s)	**Total Time (s)**	**Throughput (trans/s)**
Node 0	6.141	325.866	1.534
Node 1	7.430	324.813	1.539
Node 2	6.975	325.930	1.534
Node 3	7.408	324.812	1.539
Node 4	6.268	324.912	1.538

MaxTransactions = 5 Difficulty = 4			
	Average Mine Time Per Block (s)	**Total Time (s)**	**Throughput (trans/s)**
Node 0	3.676	1304.224	0.766
Node 1	10.582	1333.468	0.749
Node 2	3.177	1326.825	0.753
Node 3	3.611	1266.997	0.789
Node 4	3.128	1239.712	0.806
Node 5	4.056	1317.316	0.759
Node 6	4.058	1297.261	0.770
Node 7	2.466	1266.708	0.789
Node 8	2.925	1337.217	0.747
Node 9	3.356	1333.494	0.749

Παρατηρούμε ότι στη μετάβαση από blocksize 5 -> 1 υπάρχει σημαντική αύξηση στο χρόνο εξυπηρέτησης των transactions επειδή δημιουργούνται 5 φορές περισσότερα blocks. Σε ένα blockchain ο μεγαλύτερος όγκος του υπολογιστικού χρόνου αφιερώνεται στο mining. Στην περίπτωση ωστόσο της μετάβασης από blocksize 5 ->10 δεν παρατηρούμε ουσιαστική χρονική διαφορά, την οποία θα περιμέναμε αφού υπάρχει υποδιπλασιασμός των blocks που γίνονται mine. Το φαινόμενο αυτό στην υλοποίησή μας μπορεί να εξηγηθεί ως αποτέλεσμα διαφόρων παραγόντων όπως: single-threaded mining, communication overhead (μεγάλα μπλοκ), λεπτά σημεία συγχρονισμού που οδηγούν σε ανεξήγητες καθυστερήσεις. Για difficulty 5 ο χρόνος εξόρυξης των νέων μπλοκ αυξάνεται εκθετικά και ως εκ τούτου ο συνολικός χρόνος εκτέλεσης είναι απαγορευτικός για τη διεξαγωγή μετρήσεων με τους συγκεκριμένους υλικούς πόρους. Για τα πειράματα με 10 κόμβους παρατηρήσαμε αύξηση του χρόνου εκτέλεσης, καθώς οι ίδιοι πόροι μοιράζονται σε περισσότερα nodes και έτσι ο καθένας εργάζεται με την μισή περίπου ταχύτητα σε σχέση με την προσέγγιση των 5 κόμβων.

Κλάσεις

BLOCK

Class block:

Μεταβλητές:

1. previousHash: το hash του προηγούμενου block στο blockchain.
2. timestamp: η χρονοσφραγίδα της στιγμής της δημιουργίας του block.
3. listOfTransactions: τα transactions που απαρτίζουν το συγκεκριμένο block.
4. blocknumber: ο αύξων αριθμός του block.
5. nonce: ο αριθμός με τον οποίο εξασφαλίζεται το proof-of-work.

Συναρτήσεις:

1. getblocknum(): επιστρέφει τον αύξοντα αριθμό του block.
2. to_dict() : επιστρέφει ένα dictionary με τα πεδία ‘transactions’, ‘previousHash’, ‘number’, ‘timestamp’ και ‘nonce’, εάν ζητηθεί κατά την κλήση της.
3. add_nonce() : όταν βρεθεί το nonce που κάνει mine το block το προσθέτει στο αντίστοιχο self πεδίο.
4. add_hash() : θέτει τιμή στο πεδίο currenthash του self.
5. getHash() : επιστρέφει το hash του block.

Class GenesisBlock:

Μεταβλητές:

1. previousHash : default τιμή 0.
2. timestamp : η χρονοσφραγίδα της στιγμής της δημιουργίας του block.
3. listOfTransactions : τα transactions που απαρτίζουν το συγκεκριμένο block.
4. blocknumber : default τιμή 0.
5. nonce : default τιμή “0”.

Συναρτήσεις:

1. to_dict() : επιστρέφει ένα dictionary με τα πεδία ‘transactions’, ‘previousHash’, ‘number’, ‘timestamp’ και ‘nonce’, εάν ζητηθεί κατά την κλήση της.

WALLET

Class wallet:

Μεταβλητές:

1. publickey: το δημόσιο κλειδί του node.
2. privatekey: το ιδιωτικό κλειδί του node.
3. signer: αντικείμενο του crypto.

Συναρτήσεις:

1. address(): Γυρνάει τη δημόσια διεύθυνση του.
2. balance(): βρίσκει το άθροισμα των TransactionOutputs στη λίστα UTXO με παραλήπτη τον ίδιο τον client.
3. sign_transaction() : υπογράφει το transaction.

Συνάρτηση:

1. verify_signature() : επιβεβαιώνει την ορθότητα αποστολής μηνύματος από αποστολέα.

NODE

Class Node:

Μεταβλητές:

1. ip : η ip του κόμβου.
2. port : το port του κόμβου.
3. bootstrapip : η ip του bootstrap.
4. bootstrapport : η ip του bootstrap.
5. current_block: δηλαδή το τρέχον block που ο κάθε κόμβος “χτίζει” την τρέχουσα στιγμή.
6. chain: η αλυσίδα που γνωρίζει ο κόμβος, ως λίστα από Class block objects.
7. wallet: το wallet του κόμβου, το οποίο αρχικοποιείται δημιουργώντας ένα αντικείμενο τύπου wallet.
8. verified_transactions : λίστα με τα transactions που έχουν επαληθευτεί αλλά δεν έχουν προστεθεί ακόμη σε κάποιο block.
9. UTXO : λίστα με τα UTXOs που διαθέτει ο κόμβος. Είναι αντικείμενα τύπου TransactionOutput.
10. ring : λίστα από dictionaries που περιέχουν πληροφορία της μορφής ‘public-key’, ‘ip’, ‘port’ και ‘id’.
11. bcounter: μετράει τα μπλοκ που έχει κάνει mine.
12. isMining: κάνει minig το block.
13. usingChain: χρησιμοποιεί την αλυσίδα.
14. resolvingConflicts: κάνει resolve.
15. blockWhileMining.
16. mining_useless.

Συναρτήσεις:

1. contiuous_mining() : τρέχει συνέχεια.
2. create_wallet() : αρχικοποιεί ένα instance της κλάσης wallet, δημιουργεί το πορτοφόλι του κόμβου.
3. register_node_to_ring() : είναι μια συνάρτηση που καλείται μόνο από τον bootstrap node, και προσθέτει κάθε νέο κόμβο στο δίκτυο. Ακόμη στέλνει post requests στους κόμβους ζητώντας να του στείλουν τα public keys τους.
4. create_transaction() : δημιουργεί ένα transaction instance, το προσθέτει στη λίστα με τα verified_transactions που διαθέτει ο κόμβος και επιστρέφει το hash του transaction.
5. create_genesis_block() : δημιουργεί το genesis block που αποτελείται από τις genesis transactions.
6. create_genesis_transactions() : δημιουργεί ένα transaction για κάθε node, μεταφέροντάς του 100 noob coins.

7. broadcast_transaction() : κάνει broadcast μόλις το δημιουργήσει.
8. getGenesisBlock() : προσθέτει το genesisblock στην αλυσίδα και τα transactions στη λίστα UTXO.
9. verify_signature() : επιβεβαίωση υπογραφής.
10. validate_transaction() : ελέγχει ένα transaction ως προς την ορθότητά του. Συγκεκριμένα επαληθεύει τον αποστολέα, εξετάζει αν το ποσό μπορεί να μεταφερθεί βάσει του δικού του UTXO, και αν αυτά συντρέχουν, ενημερώνει τη λίστα με τα UTXOs με τα νέα transactionOutputs που έχουν δημιουργηθεί για τον αποστολέα και τον παραλήπτη του ποσού.
11. add_transaction_to_block() : αφού καλέσει τη validate_transaction, εάν δεν έχει φτάσει τα 10 transactions τότε το προσθέτει στο block. Διαφορετικά κάνει hash το block και ξεκινά τη διαδικασία του mining καλώντας την mine_block.
12. mine_job() : καλεί την mine_block, προσθέτει τις verified_transactions στο τρέχον block.
13. mine_block() : ψάχνει ένα nonce που να εγγυάται το proof-of-work μέσω της search_proof, συμπεριλαμβάνει το block στην προσωπική έκδοση της αλυσίδας και το κάνει broadcast καλώντας την παρακάτω συνάρτηση.
14. broadcast_block() : κάνει broadcast μόλις τελειώσει το mining.
15. receive_block() :λαμβάνει ένα block που έχει γίνει mined.
16. search_proof() : βρίσκει το nonce του mining.
17. valid_proof() : ελέγχει αν είναι σωστό το nonce.
18. validate_block() : ελέγχει την εγκυρότητα του block. Εάν το previousHash του τρέχοντος block δεν ταιριάζει με το αντίστοιχο πεδίο του τελευταίου block που έχει προστεθεί στην αλυσίδα του κόμβου, τότε έχουμε conflict και καλεί την resolve_conflicts για την επίλυσή του. Διαφορετικά ελέγχει όλα τα transactions του block αν είναι valid. Εάν ισχύει αυτό τότε τα αφαιρεί από τη λίστα verified_transactions.
19. valid_chain() : ελέγχει ολόκληρη την αλυσίδα, ένα block τη φορά. Ελέγχει ότι το hash του block συμπίπτει με το πεδίο previousHash του επόμενου block.
20. resolve_conflicts() :λυνει περιπτώσεις conflict.

TRANSACTION

Class TransactionInput:

Μεταβλητή:

1. parentOutputId: το id του γονέα - transaction βάσει του οποίου μπορούμε να προχωρήσουμε σε νέο transaction.

Συναρτήση:

1. to_dict() : επιστρέφει το παραπάνω πεδίο σε μορφή dictionary.

Class TransactionOutput:

Μεταβλητές:

1. recipient : η διεύθυνση του παραλήπτη.
2. amount : το ποσό που θα μεταφερθεί.
3. transactionId : το id του transaction.

Συναρτήσεις:

1. to_dict() : επιστρέφει σε μορφή dictionary τα πεδία recipient και amount.
2. fill_id() : είναι setter του πεδίου transactionId.
3. get_receiver() : είναι getter του πεδίου recipient.

Class GenesisTransaction:

Μεταβλητές:

1. receiver_address : διεύθυνση παραλήπτη.
2. amount : ποσό αλλαγής.
3. transaction_id : default τιμή 1.

Συνάρτηση:

1. to_dict() : επιστρέφει ένα dictionary με τα πεδία ‘transactions’, ‘receiver_address’, ‘amount’, ‘transaction_id’.

Class Transaction:

Μεταβλητές:

1. sender_address : το public key του wallet από το οποίο προέρχονται τα χρήματα.
2. receiver_address : το public key του wallet στο οποίο θα καταλήξουν τα χρήματα.
3. amount : το ποσό που θα μεταφερθεί.
4. transaction_inputs : η λίστα με τα UTXOs του node.
5. transaction_outputs : καλεί την createOutputs, αποτελείται τελικά από 2 αντικείμενα της κλάσης transactionOutputs.
6. signature : περιέχει το hash των πεδίων ‘sender’, ‘receiver’, ‘amount’, ‘inputs’, ‘outputs’ και ‘signature’.
7. transaction_id : το hash των πεδίων ‘sender’, ‘receiver’, ‘amount’, ‘inputs’, ‘outputs’, ‘signature’ και ‘id’.

Συναρτήσεις:

1. createOutputs() : δημιουργεί δύο transactionOutputs, ένα για τον receiver και ένα για τον sender, αφαιρώντας από το συνολικό balance που της έχει δοθεί από τα UTXOs το ποσό που μεταφέρθηκε.
2. getid() : είναι getter του πεδίου transaction_id.
3. to_dict1() : επιστρέφει σε μορφή dictionary τα πεδία ‘sender’, ‘receiver’, ‘amount’, ‘inputs’, ‘outputs’, και αν ζητηθεί από την κλήση της και τα πεδία ‘signature’ και ‘id’.
4. hash_transaction() : επιστρέφει το hash του transaction συμπεριλαμβανομένης της υπογραφής.
5. get_receiver() : είναι getter του πεδίου receiver_address.
6. add_id_to_output() : είναι setter του πεδίου transactionId της Class TransactionOutput, το οποίο το θέτει ίσο με το id του transaction.

REST

1. get_wallets() : ακούει στο /getwallets, επιστρέφει το dictionary με τα στοιχεία του κάθε κόμβου.
2. get_utxos(): ακούει στο /getutxo, επιστρέφει τη λίστα με τα utxos.
3. get_chain(): επιστρέφει το chain του node, ακούει στο /getchain.
4. add_node(): μπορεί να την τρέξει μόνο ο bootstrap node, προσθέτει νέο κόμβο στο δίκτυο, ακού στο /addnode.
5. self_register(): κάνουν εγγραφή οι νέοι κόμβοι στο δίκτυο. Στέλνουν request στο /addnode όπου ακούει ο bootstrap.
6. receive_genesis(): παίρνουμε το αρχικό block που δε χρειάζεται να γίνει validate.
7. receive_wallets(): από το /receivewallet οι nodes λαμβάνουν άλλα wallets.
8. create_transaction():δημιουργείται συναλλαγή.
9. receive_transaction(): από το /receivetransaction οι nodes λαμβάνουν άλλα transactions.
10. receive_block(): από το /receiveblock οι nodes λαμβάνουν άλλα blocks.
11. chain_len(): επιστρέφει το μήκος της αλυσίδας.
12. try_mine():
13. chain_send(): τρέχει τη main, αρχικοποιεί τη θύρα που ακούνε οι κόμβοι και τις ip, δημιουργεί ένα object node