Funkcionalna analiza blockchain čvora

Verbanac, Teo

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

FUNKCIONALNA ANALIZA BLOCKCHAIN ČVORA

Rijeka, rujan 2024.

Teo Verbanac
0069091329

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

FUNKCIONALNA ANALIZA BLOCKCHAIN ČVORA

Mentor: prof. dr. sc. Kristijan Lenac

Rijeka, rujan 2024.

Teo Verbanac
0069091329

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 24.03.2024.

Zavod: Zavod za računarstvo

Predmet: Algoritmi i strukture podataka

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Teo Verbanac (0069091329)**

Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Funkcionalna analiza blockchain čvora / Functional analysis of a blockchain node**

Opis zadatka:

Analizirati kod referentne implementacije punog čvora (eng. Full node) Bitcoin blockchain mreže s ciljem identificiranja osnovnih komponenti i međusobnih programskih zavisnosti. Razumijevanje ovih komponenata i načina na koji međusobno komuniciraju ključno je za efikasno testiranje i unapređenje sustava. Temeljiti analizu na principima funkcionalne analize softvera.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Mentor:
prof. dr. sc. Kristijan Lenac

Predsjednik povjerenstva za završni ispit:
prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024.

_____

Teo Verbanac

ZAHVALA

Zahvaljujem se svim profesorima Tehničkog fakulteta koji su me pratili tijekom studija. Posebno se zahvaljujem svojem mentoru prof. dr. sc. Kristijanu Lencu na pruženoj potpori i savjetima pri izradi završnog rada.

Sadržaj

POPIS SLIKA

Uvod

Bitcoin kao ideja javnosti postaje dostupan 2008. godine kada entitet Satoshi Nakamoto objavljuje „Bijelu knjigu” [1]. U ovoj knjizi opisane su osnovne ideje i rješenja problema koji se pojavljuju s kriptovalutama. Glavni motiv za razvoj ovakve tehnologije bit će decentralizacija, odnosno mogućnost slanja transakcije između dva korisnika na mreži bez potrebe posrednika kao npr. banaka. Vidjet ćemo da se cijela ideologija bazira na ideji međusobnog poštovanja i vjerovanja da čvorovi neće zloupotrijebiti sustav.

Glavni novitet ove tehnologije jest blockchain - javna knjiga koja bilježi sve transakcije u mreži u obliku blokova. Svaki novi blok u sebi će sadržavati zaglavlje prijašnjeg bloka, što rezultira strukturom podataka nalik na lanac, odakle i sam naziv „blockchain”. Prvi blok nastaje 2009. godine kada Nakamoto šalje deset bitcoina programeru Hal Finneyu. Ovaj blok naziva se „Genesis block” i predstavlja početak lanca. Danas se na blockchainu nalazi preko 850,000 blokova.

Još jedna bitna stavka ovakve tehnologije bit će rudarenje, odnosno proces stvaranja novoga bloka. Naime, ovaj proces zahtijevat će određenu količinu energije koje računalo mora potrošiti kako bi se stvorio novi ispravan blok. Ovo će biti temelj kojim ćemo osigurati iskreno ponašanje čvora. Količina rada koja će biti potrebna za stvaranje novoga bloka naziva se Proof of Work (dokaz o radu).

U prvome poglavlju opisat ćemo osnovne koncepte koji se koriste u Bitcoinu. Dobrim razumijevanjem ovih koncepata bit će lakše shvatiti i samu funkcionalnu analizu. Drugo poglavlje odnosi se na funkcionalnu analizu punog Blockchain čvora, te u njoj ćemo cijeli sustav podijeliti na komponente i ukratko ih opisati. Svaka komponenta imat će kratki opis, najvažnije funkcije koje sadrži, datoteke u kojima se nalazi kod funkcije te navod interakcija s drugim komponentama u sustavu. Analiza će se provesti nad Bitcoin Core sustavom. Treće poglavlje ostavit ćemo za prikaz najvažnijih pravila konsenzusa i ukratko ih objasniti. Ova pravila potrebna su kako bi se svi čvorovi na mreži mogli dogovoriti oko stanja blockchaina, budući da nemamo centralno tijelo koje može samo odrediti stanje. U četvrtom poglavlju spomenut će se dodatne komponente koje Bitcoin Core sadrži za bolju interakciju s korisnikom ili za poboljšanje sigurnosti.

Cilj ovoga rada bit će približavanje kompleksnog Bitcoin sustava čitatelju i pripremiti ga za samostalno testiranje i razvoj sustava. Prikazat će se najvažnije funkcionalnosti koje pruža Bitcoin kako bi se stvorila osnovna ideja o načinu na koji sustav funkcionira te način na koji je kod posložen. U radu će se naglasiti i određeni problemi s kojima se Bitcoin susretao te konkretnim rješenjima koje je predložila Bitcoin zajednica.

1 Osnovni pojmovi sustava Bitcoin

Glomazan i revolucionaran sustav kao što je Bitcoin sadržavat će brojne ideje i pojmove poput blockchaina, rudarenja, dokaza o radu (Proof of Work), javnih i privatnih ključeva i slično. Svaka od njih zaslužna je za ispravan i siguran rad sustava te samim razumijevanjem svake od njih dobivamo bolji uvid u rad kompletnog sustava. U ovome poglavlju ukratko ćemo objasniti neke od najvažnijih pojmova s kojima ćemo se susretati u radu.

1.1 Blockchain

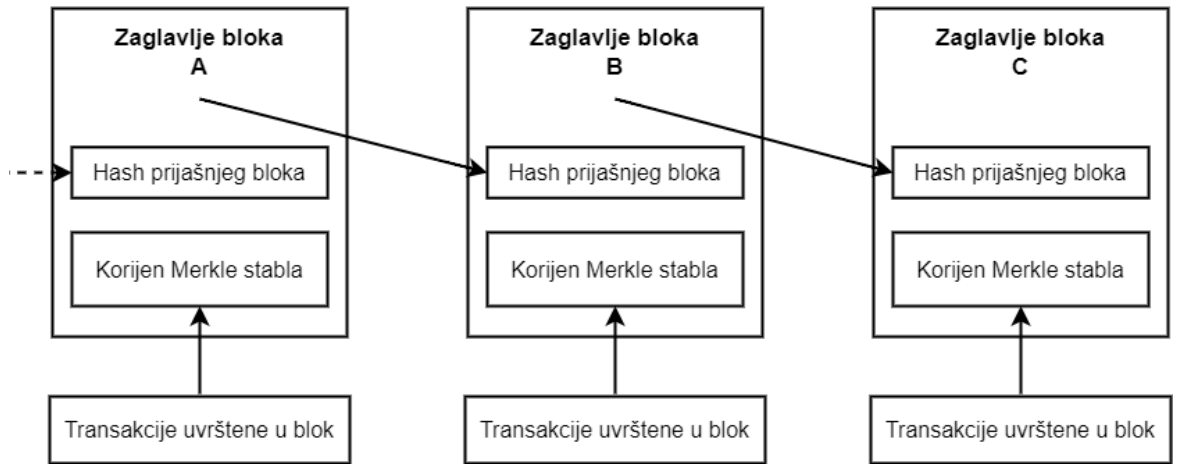
U osnovi, blockchain predstavlja distribuiranu digitalnu knjigu (ledger) koja bilježi transakcije na više računala [2] osiguravajući tako nepromjenjivost i sigurnost podataka koji su pritom javni i dostupni svima na mreži.

1.1.1 Sastav bloka

Svaki blok u lancu sadrži veličinu bloka, zaglavlje te sve transakcije na koje se taj blok odnosi, dok se zaglavlje bloka sastoji od [3]:

- Hash vrijednosti prijašnjeg bloka
- Korijena Merkle stabla
- Vremenske oznake
- Nivoa težine
- Nonce - broj korišten za Proof of Work kalkulaciju

Slika 1.1 prikazuje nam pojednostavnjenu ilustraciju blokova u blockchainu.



Slika 1.1 Pojednostavnjeni prikaz blockchaina [4]

1.1.2 Osnovne karakteristike blockchaina

U svom radu Pandey et al. ([5], 207 – 210) ovako navode najvažnija svojstva blockchaina i njihovu realizaciju:

1. **Decentralizacija**: Za razliku od tradicionalnih centraliziranih sustava, blockchain nema središnju točku kontrole. Podaci su pohranjeni na tisućama ili milijunima računala (čvorova) diljem svijeta, što povećava sigurnost i otpornost na napade.
2. **Transparentnost**: Sve transakcije na blockchainu vidljive su svim sudionicima mreže. Ovo povećava povjerenje jer svatko može provjeriti i pratiti transakcije.
3. **Nepromjenjivost**: Jednom kada su podaci zabilježeni u blockchainu, vrlo ih je teško promijeniti. To osigurava integritet podataka i smanjuje rizik od prijevara.
4. **Poboljšana sigurnost**: Blockchain koristi kriptografske tehnike za osiguranje podataka. Svaki blok sadrži kriptografski hash prethodnog bloka, što neovlašteno mijenjanje podataka čini gotovo nemogućim.

Jedno od najvažnijih svojstva blockchaina jest njegovo svojstvo **nepromjenjivosti**. Način na koji se to postiže možemo podijeliti na dvije tehnike. Prva tehnika naziva se **enkripcija**. To je tehnika koja neki običan tekst pretvara u šifrirani tekst, a način na koji se taj tekst dešifrira poznat je samo pošiljatelju i primatelju poruke. Za šifriranje samog teksta danas u računarstvu najčešće se koriste razne matematičke funkcije i pravila, a sve u svrhu zaštite informacije koja se prenosi.

Druga važna tehnika koja se koristi je **hashiranje podataka**. Ova tehnika pretvara ulaz, nevažno o njegovoj dužini, u izlaz čija će dužina uvijek biti jednaka. Velika prednost ovakve funkcije jest lakoća pretvorbe ulaza u izlaz, dok je pronalazak ulaza ako nam je poznat samo izlaz vrlo kompleksan, zbog čega se ovakva funkcija naziva „one-way“ funkcijom.

1.2 Peer to Peer mreža

Ranije u radu naveli smo kako je Bitcoin decentraliziran te da ne postoji centralna institucija koja provjerava ispravnost transakcija, već sama provjera ispravnosti dolazi od međudjelovanja i interakcija čvorova. Svi čvorovi povezani su u P2P mrežu, što na neki način podrazumijeva da su svi čvorovi jednakih prava, da ne postoje hijerarhije. Sam izraz „Bitcoin mreža” predstavlja sve uređaje spojene P2P protokolom u mrežu. Osim P2P načina povezivanja, postoji još primjerice Stratum protokol, koji se koristi za spajanje rudarskih

čvorova kako bi njihova suradnja bila što efikasnija, fleksibilnija i sigurnija. Ovakva mreža naziva se „proširenom Bitcoin mrežom" [2].

Budući da su sva računala u mreži jednakih prava, donošenje promjena konsenzusa predstavlja određene probleme. Kako bi promjena uopće bila prihvaćena, ona zahtijeva prihvaćanje od većine čvorova. Ako je većina čvorova prihvatila promjenu, sada je pitanje na koji će način unesene promjene utjecati na konsenzus.

1.2.1 Čvorovi u Bitcoin mreži

Za svako računalo spojeno na Bitcoin P2P mrežu možemo reći da je jedan čvor. Antonopoulos ([2], 172 – 182) objašnjava kako svi oni posjeduju ista prava, no mogu se razlikovati po funkcionalnostima koje izvršavaju. Primjerice **puni čvor** sadrži: novčanik, rudara, u cijelosti preuzetu kopiju blockchaina te mogućnost validacije i propagacije transakcija i blokova po mreži. Osim punog čvora postoje varijacije kao primjerice:

- **puni blockchain čvor** (full blockchain node) - sadrži kopiju blockchaina te validaciju i propagaciju
- **SPV novčanik** (SPV wallet) - sadrži novčanik te validaciju i propagaciju
- **samostalni rudar** (solo miner) - sadrži rudarenje, kopiju blockchaina te validaciju i propagaciju

1.3 Ažuriranje sustava i račvanja

Svako ažuriranje sustava neovisno o veličini promjene podložno je odbijanju od strane ostalih čvorova ako oni smatraju da je promjena nepotrebna. Za provedbu uspješne promjene smatra se da preko 85% čvorova mora prihvatiti promjene i ažurirati svoj sustav. Ovisno o tome je li promjena „forwards compatible" ili ne, nastat će hard i soft forkovi.

1.3.1 Hard fork

Ovakva promjena u konsenzusu rezultirat će račvanjem između staroga lanca i novoga koji će nastati u trenutku kada ova pravila stupe na snagu. Čvorovi koji se odluče zadržati staru verziju konsenzusa nastavit će koristiti stari blockchain. Jedan od poznatijih primjera ovoga bila je pojava Bitcoin Cash valute. Razlog zašto je došlo do pojave nove valute naveden je u potpoglavlju 3.4.

1.3.2 Soft fork

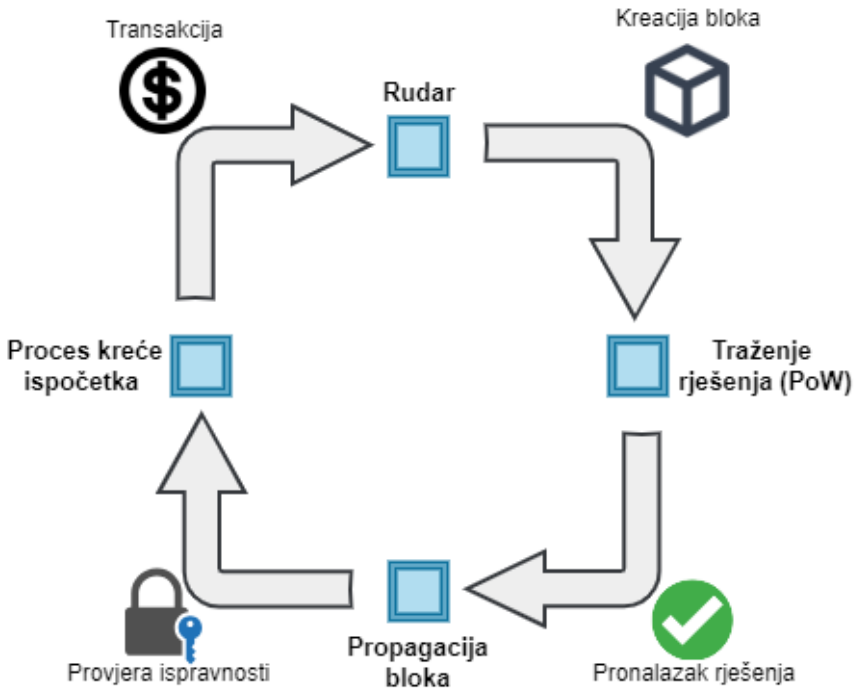
Za razliku od hard forka, ova će promjena biti „forwards compatible“, što podrazumijeva sposobnost neažuriranih čvorova da mogu validirati blokove ažuriranih čvorova. Sada će i čvorovi s ažuriranim konsenzusom i oni sa starijom verzijom koristiti isti lanac, iako mogu imati različita pravila za dodavanje blokova. Dobar primjer ovakvog račvanja dogodio se 2014. godine kada se promijenio način na koji se spremaju transakcije, nazvan „Segregated witness“, odnosno odvojeni svjedok. Ažurirani čvorovi mogli su prihvatiti i razumjeti nove transakcije, dok su stari čvorovi prihvatili nove transakcije iako ih nisu u potpunosti mogli razumjeti. I novi i stari čvorovi razumjeli su starije verzije transakcija.

1.4 Proof of Work

Proof of Work koncept nastaje 1993. kao rješenje protiv neželjene pošte na mail sustavima [6]. Svaki mail koji bi se tada poslao morao je riješiti kompleksnu matematičku slagalicu i to rješenje pridodati e-mailu. Bez pridodanog rješenja mailovi bi bili odbijeni od strane primatelja. Budući da pronalazak rješenja zahtijeva određeno vrijeme i snagu, slanje velike količine mailova odjednom zahtijeva i veliku količinu računalne snage.

Možemo reći da Bitcoinov PoW funkcionira na vrlo sličan način. Ovdje je cilj poticati čvor da se ponaša iskreno, jer u slučaju da čvor nadoda lažnu transakciju u novi blok, ostatak mreže može odbiti taj blok, pri čemu je čvor potrošio vrijeme i računalnu snagu na vlastiti trošak. Kada bi izrada bloka bila lagana i nezahtjevna, u slučaju odbijenoga bloka čvor bi jednostavno mogao predlagati nove blokove, koji bi opet sadržavali lažne transakcije. Budući da bi troškovi u ovome slučaju bili zanemarivi, nakon dovoljno pokušaja čvor bi uspio svoj blok dodati u blockchain. Bitcoin zahtijeva od rudara da riješe kompleksne kriptografske probleme kako bi nastao novi blok.

Način na koji se rješenje pronalazi jest modifikacijom vrijednosti koja se naziva nonce. Ova vrijednost je zapravo najobičniji broj. Pri analizi procesa rudarenja vidjet ćemo kako ta vrijednost utječe na sam hash. Na slici 1.2 vidimo korake stvaranja novog bloka.



Slika 1.2 Prikaz koraka za PoW koncept [7]

Redom koraci su: skupljanje transakcija, dolazak transakcija do rudara, dodavanje transakcije u blok, pronalazak rješenja za blok, propagacija bloka po mreži i validacija bloka od drugih čvorova. Vidimo da čim čvor prihvati blok, kreće proces rudarenja za novi blok.

1.5 Proces rudarenja

Bitcoin kao hash funkciju koristi SHA256, kojoj je izlaz veličine 256 bita. Ulazni podatak je zaglavlje bloka koje sadrži informacije kao što su primjerice: korijen Merkle stabla, nonce, vremensku oznaku te hash prijašnjeg bloka. Bitno je napomenuti da je izlaz ove funkcije praktički nepredvidljiv, te promjena samo jednog bita ulaza rezultirat će potpunom promjenom izlaza. Sve prije navedene vrijednosti za ulazni podatak su fiksne osim parametra nonce. Promjenom vrijednosti ovoga parametra dobivat ćemo potpuno različite izlaze, a cilj rudara bit će pronalazak odgovarajuće vrijednosti koja pruža izlaz gdje nekolicina početnih bitova sadrži vrijednost nula.

Hash funkcija SHA256, osim što proizvodi nasumičan izlaz, također sadrži još dva bitna svojstva koja održavaju stabilnost blockchaina:

1. **Otpornost na kolizije** - šanse kako će dva različita ulaza pružiti isti izlaz su gotovo nemoguće

2. **Nemogućnost rekreacije ulaza poznavajući samo hashiranu vrijednost** - budući da je veličina ulaza 2^{256}, napadač bi trebao iteracijom kroz sve kombinacije pronaći odgovarajuću

Za primjer ćemo koristiti riječ blockchain nad kojom ćemo provesti hash funkciju. Tablica 1.1 jasno prikazuje kako će i najmanja promjena znakova prouzročiti potpuno drugačiji i nepredvidljivi izlaz.

Tablica 1.1 Prikaz SHA256 funkcije na primjeru

Ulazni podatak	**Izlaz**
Blockchain	625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1
Blockchain0	1170bfd4266d3ff0472740483fd69223e04365054e621a72ef2482d5401d1f4d
blockchain	ef7797e13d3a75526946a3bcf00daec9fc9c9c4d51ddc7cc5df888f74dd434d1

1.6 Bitcoin ključevi i potpis

Koriste se za slanje i primanje bitcoin valute, uz njihovu pomoć korisnik dokazuje posjedovanje bitcoina. Hashiranjem javnoga ključa nastaje Bitcoin adresa s pomoću koje navodimo gdje/kome želimo poslati kriptovalutu.

1.6.1 Privatni ključ

Predstavlja broj između 1 i 2^{256} koji omogućuje kontrolu svih fondova za odgovarajuću Bitcoin adresu [8]. U slučaju da ovaj ključ dospije u krive ruke, ta osoba praktički dobiva kontrolu nad bitcoinom koji je zaštićen ovim ključem. Isto tako, u slučaju gubitka ključa svi fondovi bit će izgubljeni. Način na koji odabiremo broj zapravo nije bitan, već je bitno da to bude nepredvidljivo, nasumično i sigurno.

1.6.2 Javni ključ

Sam javni ključ je izvedenica privatnog ključa, te radi ovog matematičkog svojstva moguće je generiranje potpisa. Za puno razumijevanje ove formule bilo bi nam potrebno znanje o eliptičnim krivuljama koje se koriste za generaciju ključa. Za potrebe ovog rada spomenuti ćemo samo formulu koja se koristi ali nećemo ulaziti u detalje. Formula za javni ključ:

$$K = k * G \qquad (1.1)$$

- k predstavlja nasumično generirani broj (također predstavlja i privatni ključ)
- G predstavlja točku, konstantu koja se nalazi na eliptičnoj krivulji, G je jednak za sve Bitcoin korisnike
- K predstavlja javni ključ

Budući da je G jednak za sve Bitcoin korisnike, privatni ključ k pomnožen s G uvijek rezultira istim javnim ključem, no važno je da se privatni ključ ne može izračunati poznavajući javni ključ K. Jedini način izračuna jest pokušaj svih mogućih k vrijednosti, što je 2^{256} mogućnosti.

1.6.3 Digitalni potpis

Potpisi su zapravo način da korisnik potvrdi vlasništvo nad izlazom kojeg želi potrošiti. Bez digitalnih potpisa korisnik bi morao priložiti privatni ključ, pri čemu bi svi sudionici na mreži vidjeli taj ključ. Kako bi se ovo izbjeglo uvode se digitalni potpisi. Digitalan potpis je zapravo kombinacija privatnog ključa, nasumično generiranog broja i podataka transakcije nad kojima se provode određene matematičke funkcije. Nakon što se potpis kreira korisnik može dokazati da je on uistinu vlasnik javnoga ključa bez opasnosti da će neki čvor u mreži vidjeti njegov privatan ključ.

1.7 Bitcoin Script

Jezik koji služi za definiranje uvjeta pod kojim bitcoin može biti potrošen. Najveća prednost ovoga jezika jest korištenje stoga za izvođenje svih operacija i manipulacije podataka. Skripte imaju ograničene kompleksnosti kako bi vrijeme izvođenja bilo predvidljivo. Isto tako korištenje petlja je onemogućeno kako bi se izbjegla ranjivost prema DoS napadima.

Svaka transakcija ima dva dijela: skriptu zaključavanja (scriptPubKey) te skriptu otključavanja (scriptSig). Skripta zaključavanja postavlja se na izlaz transakcija i njome se specificiraju uvjeti koji moraju biti ispunjeni kako bi se novac mogao potrošiti. Skripta otključavanja ima svrhu ispunjenja uvjeta koje zadaje skripta zaključavanja. One se nalaze na ulazu svake transakcije. Ova skripta naziva se scriptSig jer sadrži digitalni potpis kojeg generira novčanik korisnika.

Za potvrdu transakcije čvor izvršava i skriptu zaključavanja i otključavanja. Prvo se na stogu izvodi skripta otključavanja. Ako izvršavanje prođe bez grešaka, cijeli taj stog se kopira i pokreće se skripta zaključavanja. U slučaju da je rezultat istinit, ulaz može potrošiti UTXO.

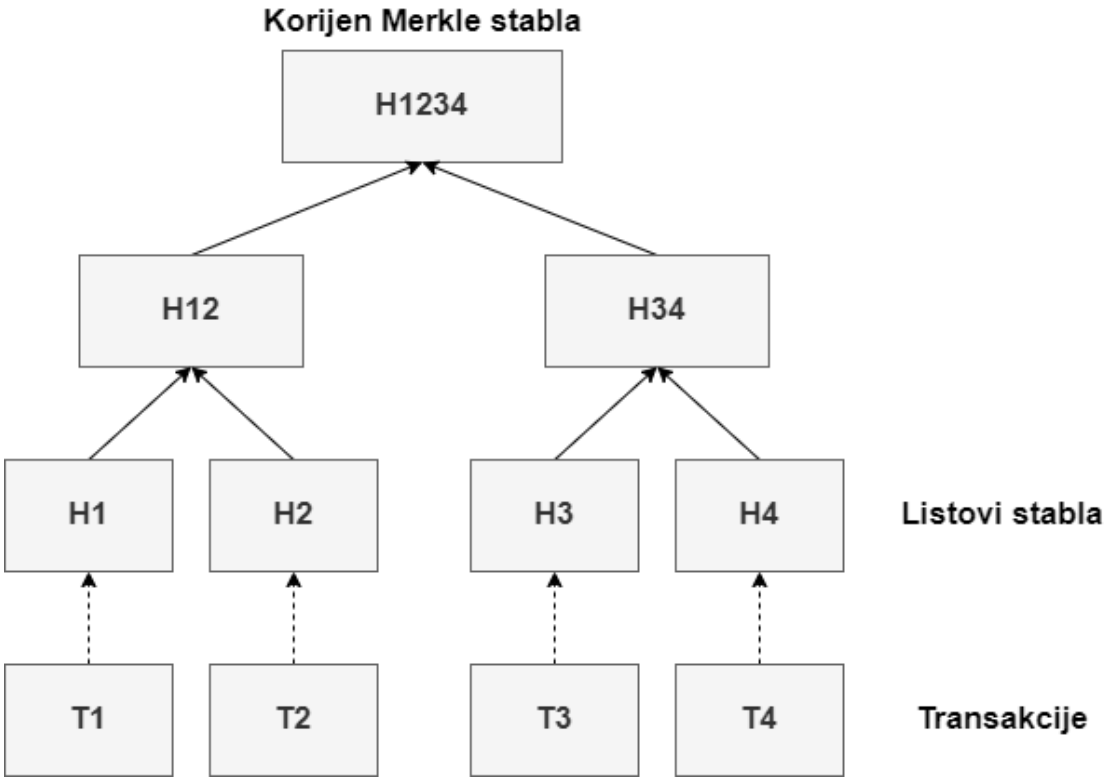
1.7.1 Pay to Public Key Hash(P2PKH)

Iako se danas češće koristi Pay to Witness Public Key Hash(P2WPKH), način na koji oni funkcioniraju je poprilično sličan. Jedina razlika je da P2WPKH ima manju proviziju te scriptSig se sada naziva „witness“, odnosno svjedok. P2PKH zaključava izlaz koristeći hashirani javni ključ, a za otključavanje potrebno je pružiti skriptu otključavanja u formatu: *<potpis korisnika><javni ključ korisnika>* [9].

1.8 Merkle stablo

Apsolutno svaki blok koji se nalazi na blockchainu sadrži rezime svih transakcija u bloku koristeći Merkle stablo. Izraz stablo u programiranju podrazumijeva strukturu gdje će se podaci granati. Merkle stablo je binarno stablo koje koristi kriptografske hasheve transakcija kao listove. Budući da je stablo binarno svaki roditelj ima dva potomka, a sam podatak koji se nalazi na poziciji roditelja jest zbroj potomaka hashiran SHA256 funkcijom. Ovaj proces ponavlja se od najnižih razina prema najvišoj odnosno korijenu stabla. Kada izračunamo vrijednost korijena stabla tu vrijednost ćemo dodati u zaglavlje bloka koje će se s ostalim podacima hashirati i dati konačno zaglavlje.

Korištenjem ovakve strukture podataka omogućuje se brzi pregled transakcija bloka, bez potrebe za preuzimanjem cijelog bloka. Također promjenom bilo kakve transakcije korijen će promijeniti vrijednost, što nam osigurava sigurnost i integritet transakcija. Na slici 1.3 vidimo prikaz Merkle stabla.



Slika 1.3 Prikaz pojednostavnjenog Merkle stabla [10]

Možemo primijetiti da hash vrijednosti transakcija predstavljaju listove, te se njihovim zbrajanjem dobiva vrijednost roditelja.

1.9 Transakcija

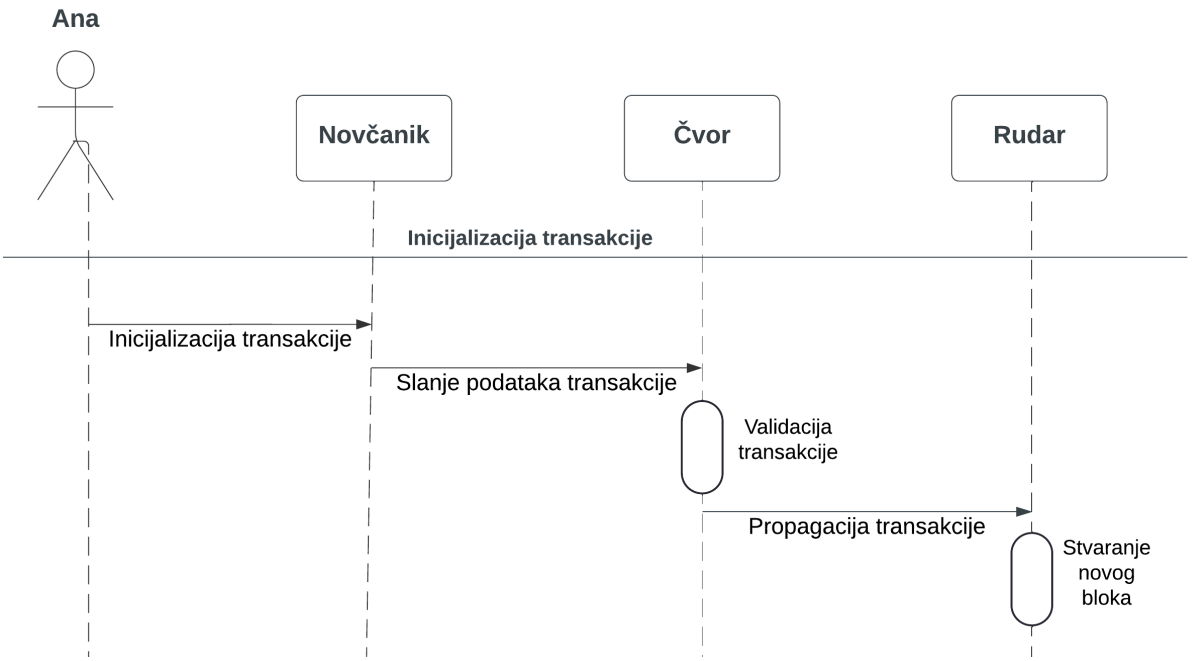
Transakcija u Bitcoinu podrazumijeva prijenos određene količine bitcoina koju čvor posjeduje drugome čvoru. Za svaku uplatu potrebno je znati javni ključ (gdje/kome šaljemo kriptovalutu) te količinu koju uplaćujemo. Za ovaj proces mogu se koristiti i QR kodovi koji imaju već definirani zahtjev za isplatu te se adresa primatelja i količina upisuju automatski.

Za opis transakcije možemo odabrati primjerice Anu i Josipa, gdje Ana želi Josipu uplatiti određeni iznos. Također ćemo pretpostaviti da Ana već unaprijed ima dovoljno bitcoina kako bi transakcija bila uspješna. Ana prvo skenira QR kod, pregledava transakciju te pritiskom na pošalji (send) odobrava plaćanje. Nakon par sekundi, Josipu će u novčaniku aplikacije biti vidljiva transakcija koju i on sam može potvrditi, iako ona još nije dodana u blockchain.

No kada će se zapravo ona upisati i kada ćemo moći sa sigurnošću reći da je transakcija validna? Za odgovor na ovo pitanje moramo se vratiti na trenutak kada Ana pritisne gumb

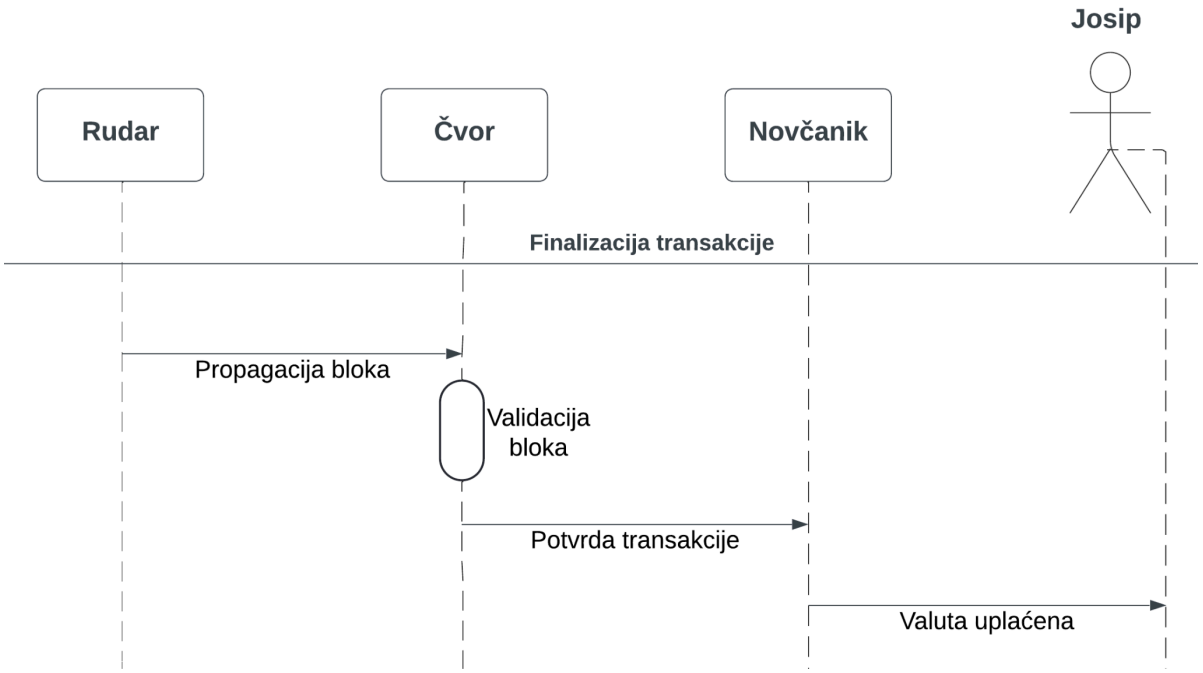
pošalji. Ta transakcija se propagira dalje na sve čvorove na koje je Anin mobitel spojen, te nakon toga oni dalje propagiraju tu informaciju dokle je svi čvorovi nisu zaprimili. Kada ta informacija pristigne do čvora koji ima funkciju rudara, ona se sprema s drugim transakcijama koje još nisu dodane u blockchain u takozvani transakcijski skup (mempool). Te transakcije bit će dodane pri stvaranju novoga bloka, procesu koji se događa približno svakih deset minuta.

Na slikama 1.4 i 1.5 prikazan je takav slučaj korištenjem sekvencijskog dijagrama.



Slika 1.4 Sekvencijski dijagram za prikaz inicijalizacije transakcije

Na prvome dijagramu prikazujemo nastanak transakcije koristeći novčanik te zatim propagaciju transakcije drugim čvorovima. Transakcija se propagira tako da Anin čvor šalje informacije dalje prema svim čvorovima za koje zna da postoje u mreži. Ovakav način širenja informacija P2P mrežom također se naziva i „gossip“ protokol, budući da se informacije šire nalik glasinama u društvu. Nakon što informacija o transakciji stigne do čvora koji ima svojstvo rudarenja, on će tu transakciju dodati u blok kojega će kreirati.



Slika 1.5 Sekvencijski dijagram za prikaz potvrde transakcije

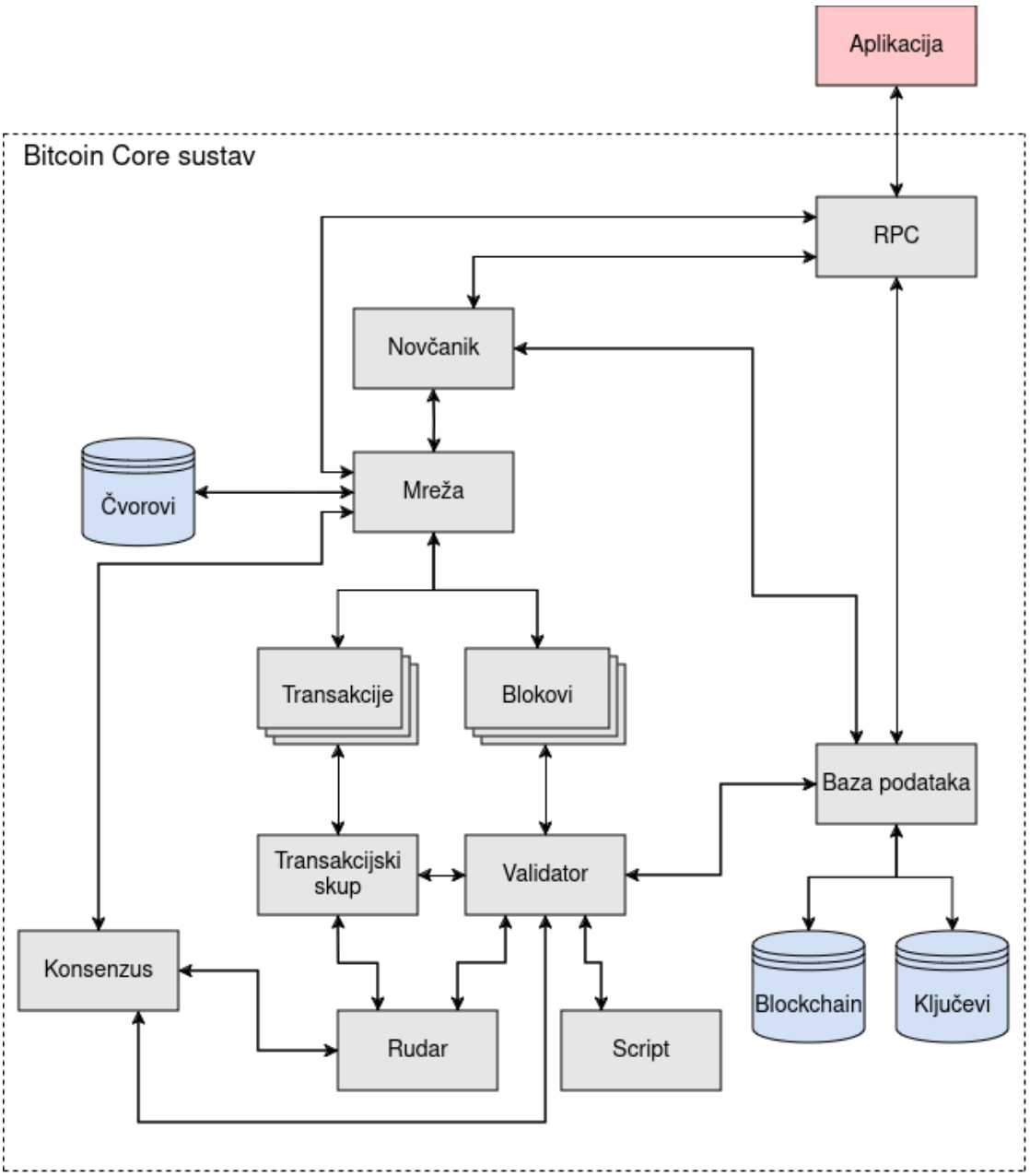
Čim rudar pronađe rješenje za PoW on taj blok propagira po mreži vrlo slično načinu na koji Anin čvor propagira transakciju. Kada Josipov čvor primi informaciju o novom bloku, on će ga validirati te dodati u svoj blockchain. Dodavanjem bloka u blockchain Josipov novčanik prikazat će iznos koji je nadodan pri Aninoj transakciji.

Dok neki servisi za manja plaćanja znaju čekati za jednu potvrdu, ili čak nijednu, većina kompanija koje koriste Bitcoin plaćanja će čekati duže, budući da se svakom potvrdom smanjuje šansa da isplata nije validna. Najčešće se kao omjer sigurnosti i brzine uzima barem tri validacije, što traje oko pola sata.

2 FUNKCIONALNA ANALIZA

Funkcionalna analiza u programiranju predstavlja razdvajanje sustava na njegove najvažnije funkcije i promatranje odnosa među njima, pružajući tako uvid u način na koji elementi unutar sustava djeluju. Kvalitetno provedena analiza može uvelike pomoći pri shvaćanju, testiranju i unapređenju sustava. Za funkcionalnu analizu Bitcoina odredit ćemo deset glavnih funkcionalnosti: **konsenzus**, **blockchain**, **mreža**, **rudar**, **transakcijski skup**, **novčanik, validator**, **RPC**, **baza podataka** i **Script**. U nastavku ćemo detaljno opisati svaku od njih, gdje se one nalaze u kodu te u kakvim su odnosima s drugim komponentama.

Na slici 2.1 vidimo raspodjelu i međusobne interakcije komponenta. Podjela je izrađena na način kako bi objašnjenje komponenti bilo čim jasnije, jednostavnije i što preciznije. Primjerice, moglo se odrediti da komponenta konsenzus u sebi sadrži komponentu validator, što bi također bilo točno, no zbog jednostavnosti ovdje su razdvojene. Blok „aplikacija“ predstavlja klijenta koji preko RPC poziva koristi Bitcoin sustav kao server za dobivanje potrebnih informacija. Potpoglavlje 2.8 sadrži više informacija o ovoj temi.



Slika 2.1 Podjela komponenti i prikaz njihovih interakcija

Prije nego krenemo sa samom analizom, napomenuti ćemo da za prikazivanje datoteka nisu korišteni nastavci poput „.cpp“ i „.h“, već se naziv odnosi na obje datoteke (ako postoje). U većini slučajeva te datoteke bit će usko povezane i nadopunjavat će jedna drugu. Primjerice klasa i deklaracija funkcije nalazit će se u „.h“ datoteci, dok će se njene funkcionalnosti (odnosno njena konkretna implementacija) nalaziti u „.cpp“ datoteci.

2.1 Konsenzus

Sama riječ konsenzus podrazumijeva slaganje ili dogovor većine. Slično tome, konsenzus u Bitcoinu svodi se na blockchain i sporazum oko lanca koji ima najveći akumulirani PoW. **Ovdje se nalazi konkretna implementacija Proof of Work algoritma** koji je najvažnije pravilo unutar cijelog Bitcoina. Pod konsenzusom još možemo pronaći **pravilo najdužeg lanca**, koje osigurava kako će svaki čvor sadržavati lanac s najvećim PoW.

2.1.1 Analiza konsenzusa

Korištene datoteke:

- src/consensus/params - sadrži klasu params koja nam služi za postavljanje raznih parametara koji se koriste za izračun u algoritmu prilagodbe težine
- src/consensus/merkle - služi za stvaranje Merkle stabla
- src/consensus/tx_check - služi za provjeru ispravnosti transakcije

Možemo reći da svaka datoteka unutar mape consensus ima svoj točno određeni zadatak vidljiv iz imena datoteke.

Najvažnije interakcije:

- Mreža - provjerava slažu li se lokalne kopije blockchaina među čvorovima
- Validator - koristi za validaciju novih transakcija i blokova
- Rudar - provjerava ako je novi blok ispravan, to jest zadovoljava li taj blok navedena pravila

Najvažnije funkcije:

- CheckTransaction() - provjeri transakciju koristeći pravila konsenzusa
- ComputeMerkeTree() - računa korijen Merkle stabla za dani niz transakcija

2.2 Blockchain

Već smo ranije spomenuli blockchain kao strukturu podataka i neke njegove specifikacije. **Glavna funkcija ovoga modula jest upravljanje blockchainom, dodavanje novih blokova, organizacija samih blokova te njihovo spremanje.** U slučaju da čvor zaprimi lanac, čiji je ukupni akumulirani Proof of Work veći od onoga lokalne kopije blockchaina, koristeći pravilo najdužeg lanca lokalni blockchain će se ažurirati kako bi sadržavao lanac gdje je PoW najveći.

2.2.1 Analiza blockchaina

Korištene datoteke:

- src/chain - nalaze se funkcije poput bool Contains() koji služi za provjeru je li određeni blok nadodan u blockchain
- src/chainparams - sadrži parametre poput maksimalne veličine bloka i slično kojima upravljamo samom strukturom blockchaina
- src/validation – ovdje se nalaze funkcije ActivateBestChain() i FindForkInGlobalIndex()

Najvažnije interakcije:

- Baza podataka - spremanje blockchaina na medij
- Validator - provjera ispravnosti blokova
- Konsenzus - koristeći ovu komponentu rješava probleme račvanja te provjerava ispravnost blokova

Najvažnije funkcije:

- Contains() - provjerava je li određeni blok već nadodan u blockchain
- FindForkInGlobalIndex() - vraća pokazivač na blok nakon kojega slijedi račvanje lanca
- ProcessNewBlock() - dodaje novi blok u blockchain
- ActivateBestChain() - ova funkcija se pokreće svaki put kada postoji mogućnost da lokalna kopija blockchaina nije ona s najvećim akumuliranim PoW, primjerice pri zaprimanju novoga bloka ili pri pokretanju čvora. Funkcija provjera je li lokalna kopija lanca zaista ona s najvećim PoW i ako postoji lanac s većim PoW pokreće reorganizaciju

2.3 Mreža

Budući da je Bitcoin decentraliziran, svaki čvor u mreži ima značajnu ulogu. Dijeljenjem podataka između čvorova i njihovom komunikacijom osiguravamo integritet i sigurnost da jedan zlonamjerni čvor ne može ugroziti mrežu. Ova komponenta ključna je za Bitcoin, budući da i kratkotrajni kvar može onesposobiti cijelu mrežu i uzrokovati probleme poput račvanja lanca.

Funkcija ovoga modula jest upravljanje P2P mrežom i komunikacijom između čvorova. Također mora održavati veze, pronaći i pridodati nove čvorove te ukloniti neaktivne. Nadalje, ovaj modul prati ponašanje čvorova te bilježi pokušaje širenja lažnih transakcija po mreži. Za ovakvo ponašanje pridodaju se kazneni bodovi, koji će nakon prelaska određenog praga rezultirati vremenskom zabranom korištenja Bitcoin mreže.

2.3.1 Analiza mrežne komponente

Korištene datoteke:

- src/net - upravljanje portovima, dodavanje, brisanje i blokiranje pristupa mreži čvorovima uz pomoć addrman i banman datoteka
- src/banman - služi za kažnjavanje neiskrenih čvorova
- src/net_processing - odlučuje što učiniti za svaku primljenu informaciju i u skladu s odlukom koordinira druge komponente
- src/protocol - definira strukture poruka koje se izmjenjuju između čvorova koji su spojeni na mrežu

Najvažnije interakcije:

- Prosljeđivanje blokova i transakcija drugim čvorovima u mreži
- Primanje blokova i transakcija koji se propagiraju mrežom
- Validator - slanje primljenih transakcija i blokova validatoru kako bi se potvrdila ispravnost istih

Najvažnije funkcije:

- AddNode() - dodavanje nove adrese u skup poznatih adresa
- Ban() - zabranjuje određenom čvoru sudjelovanje u mreži na određeno vrijeme
- ProcessMessage() - procesira primljenu poruku te određuje što treba dalje raditi, primjerice odgovor nazad čvoru koji je poslao poruku
- ProcessBlock() - procesira zaprimljeni blok, pregledava je li blok novi ili već od prije poznat čvoru
- ProcessHeadersMessage() - procesira zaglavlja blokova koje zaprimi, ovo je bitna stavka jer po mreži se općenito prvo šalju zaglavlja, a nakon toga cijeli blokovi
- OpenNetworkConnection() - otvara novu vezu prema čvoru

2.4 Rudar

Proces rudarenja najjednostavnije možemo objasniti kao kontinuirano hashiranje zaglavlja bloka dokle prvih nekoliko znakova u rezultatu ne počinju nulom. Broj potrebnih nula određuje se algoritmom prilagodbe težine (potpoglavlje 3.2). Najvažniji parametri koji ulaze u proces hashiranja su nonce, korijen Merkle stabla i hash vrijednost prijašnjeg bloka. Merkle stablo slaže se nad transakcijama koje se nalaze u transakcijskom skupu te se slažu po određenim pravilima.

Funkcija ovoga modula jest proizvesti novi, ispravni blok koji će se nadodati na blockchain te pritom potvrditi određene transakcije.

2.4.1 Analiza rudara

Korištene datoteke:
src/node/miner - glavna datoteka za ovaj modul, sadrži sve najvažnije funkcionalnosti

Najvažnije interakcije:

- Transakcijski skup - rudar pri izradi svakog bloka informacije o transakcijama pronalazi u mempoolu
- Validator - svaki novo izrađeni blok mora proći provjeru ispravnosti prije njegovog dodavanja u blockchain i daljnje propagacije po mreži

Najvažnije funkcije:

- CreateNewBlock() - najvažnija funkcija za ovaj modul, služi za kreaciju novog bloka

2.5 Skup transakcija

Mempool ili skup transakcija, služi čvoru kako bi pratio transakcije koje još nisu dodane u blockchain, a izvršene su. Tako će primjerice novčanik uz pomoć transakcijskog skupa moći pratiti transakcije koje su povezane s korisnikom ali se još nisu potvrdile dodavanjem u blockchain. Ova stavka posebno je važna pri rudarenju, budući da sve transakcije koje se nalaze ovdje trebaju čim prije biti uključene u blockchain. Transakcije koje imaju veću proviziju vrlo vjerojatno će biti prije nadodane u blok, budući da tako rudar ostvaruje najveću zaradu (zarada rudara sastoji se od nagrade bloka i provizije svih transakcija koje je uvrstio u blok). Također postoji mogućnost vraćanja transakcija natrag u mempool, ako se određeni

blok ne pridoda glavnome blockchainu. U slučaju da neka transakcija provede 72 sata unutar mempoola i ne uvrsti se u blockchain, ona se briše iz mempoola i smatra se nevaljanom.

Glavna funkcija jest dodavanje i spremanje transakcija koje se smatraju važećima do trenutka njihova uvrštavanja u blockchain.

2.5.1 Analiza transakcijskog skupa

Korištene datoteke:

- src/txmempool - najvažnija datoteka za ovu komponentu, sadrži klasu CTxMemPool gdje se nalaze bitne varijable i funkcije za upravljanje skupom transakcija
- src/policy/fees i src/policy/feerate - zajedno služe za izračun provizije na određenu transakciju i pripomažu rudaru koliko je određena transakcija bitna, koliko će brzo biti dodana u blockchain

Najvažnije interakcije:

- Rudar - rudar pomoću mempoola odabire transakcije koje će dodati u novi blok
- Mreža - primanje novih transakcija koje treba nadodati
- Validator - svaku nadolazeću transakciju validira prije dodavanja u mempool

Glavne funkcije:

- RemoveForBlock() - pri stvaranju novog bloka ukloni transakcije koje su uključene u taj blok iz transakcijskoga skupa
- IsSpent() - provjerava je li slučajno došlo do „double spenda“

2.6 Novčanik

Ova komponentna omogućava pregled, slanje i primanje kriptovaluta preko mreže. S pomoću javnog ključa drugi korisnici mogu na naš račun poslati kriptovalutu. Pomaže i ubrzava proces izrade transakcije tako što automatski pronalazi dovoljnu količinu UTXO kako bi korisnik isplatio iznos. Novčanik je zadužen i za automatsko potpisivanje transakcije s izračunom provizije. Bitno je napomenuti da niti jedan Bitcoin novčanik u sebi ne sadrži bitcoin, već korištenjem privatnih ključeva oni osiguravaju siguran pristup bitcoinu.

Funkcija ovoga modula jest praćenje stanja računa korisnika te spremanje lokalne kopije privatnog ključa.

2.6.1 Analiza novčanika

Korištene datoteke:

- src/wallet/wallet - služi za stvaranje, pokretanje i brisanje novčanika, također može kreirati transakciju. Ova datoteka puno surađuje s walletdb datotekom
- src/wallet/walletdb - služi za spremanje podataka poput primjerice javnih i privatnih ključeva te transakcija
- src/wallet/fees - koristi se za izračun provizije
- src/wallet/spend - sadrži funkcije za stvaranje i potpis transakcija

Sve datoteke unutar mape src/wallet možemo smatrati značajnim za ovu komponentu.

Najvažnije interakcije:

- Baza podataka - spremanje podataka novčanika, kada se korisnik ponovno spoji na mrežu iz baze se učitavaju podaci kako bi se korisniku prikazalo trenutno stanje računa
- Mreža - nakon što stvori i potvrdi transakciju propagira je dalje po mreži kako bi je ostali čvorovi zaprimili
- Blockchain - praćenje transakcija koje su bitne korisniku

Najvažnije funkcije:

- CreateTransaction() - funkcija za stvaranje novih transakcija
- CommitTransacion() - propagacija transakcije po mreži
- LoadWallet() - učitavanje novčanika uz pomoć prije spremljenih podataka
- WriteTx() - zapisuje transakciju u bazu podataka

2.7 Validacija / validator

Provjerava jesu li blokovi i transakcije u skladu s Bitcoin protokolom. Svaki puni čvor provodi ovaj proces nad svim transakcijama i blokovima koje zaprimi. U slučaju da validator uspostavi da transakcija ili blok nisu ispravni, oni se odbacuju i ne dijele se dalje po mreži.

Glavni cilj jest provjera ispravnosti transakcija provjeravajući potpise i ulaze te provjera ispravnosti blokova provjeravajući krši li blok pravila konsenzusa i nalaze li se u bloku „double spend“ transakcije.

Ukratko, možemo reći da validacija potvrđuje ispravnost transakcija, da nije došlo do „double spend“ problema, a konsenzus podrazumijeva da se članovi u mreži slažu oko poretka validiranih transakcija [11].

2.7.1 Analiza validatora

Korištene datoteke:

- src/validation - dodavanje ili brisanje bloka iz blockchaina, funkcije koje pregledavaju ispravnost blokova i transakcija
- src/consensus/validation - surađuje sa src/validation za validaciju blokova
- src/consensus/tx_verify - surađuje sa src/validation za validaciju transakcija

Najvažnije interakcije:

- Script - pokretanjem skripta određuje se ispravnost transakcije
- Konsenzus - već na primjeru datoteka koje smo naveli možemo vidjeti kako ova dva modula surađuju u svrhu provedbe protokola

Najvažnije funkcije:

- CheckBlock() - provjerava ispravnost bloka prije njegova dodavanja
- ConnectBlock() - dodavanje bloka u blockchain te ažuriranje UTXO seta
- DisconnectBlock() - uklanjanje bloka iz blockchaina i poništavanje transakcija koje su prethodno bile dodane tim blokom
- AcceptToMemoryPool() - pregledava ispravnost transakcije, te u slučaju da je transakcija validna dodaje se u mempool

2.8 RPC (Remote Procedure Call)

Sam RPC, ne vezano za Bitcoin predstavlja komunikacijski protokol između dva računala koji prvom računalu, zvanom klijent, omogućava korištenje servisa i resursa drugog računala, zvanog server. Klijent preko mreže šalje zahtjev serveru za određenu funkciju i određene parametre. Kada server zaprimi zahtjev, obrađuje ga i šalje klijentu nazad rezultate.

Glavna zadaća RPC-a jest pružanje sučelja vanjskim programima kako bi mogli komunicirati s čvorom. Ta komunikacija se najčešće sastoji od komandi i upita za određene podatke.

2.8.1 Analiza RPC komponente

Korištene datoteke:

- src/rpc/client i src/rpc/server - odnose se na komunikaciju između klijenta i servera
- src/rpc/protocol - služi pri komunikaciji između klijenta i servera, način na koji oni šalju upite i primaju odgovore
- src/rpc/blockchain - služi za upite o blockchain podacima
- src/rpc/rawtransaction - služi za rukovanje transakcijama

Svaka datoteka unutar src/rpc mape predstavlja pozive prema točno određenoj komponenti Bitcoina vidljivoj iz imena datoteke.

Najvažnije interakcije:

- Ova komponenta olakšava korisniku korištenje novčanika, blockchaina, mreže i rudara te su to komponente s kojima ona najviše surađuje, iako se može reći da surađuje s gotovo svim komponentama.

Najvažnije funkcije za sam RPC:

- StartRPC() - pokreće RPC server
- ExecuteCommand() - pokreće izvršavanje određenog RPC poziva koji se navodi kao parametar pri pozivu funkcije

2.8.2 Podjela RPC poziva

RPC naredbe na temelju komponente s kojom komuniciraju možemo podijeliti na sljedeći način [12]:
Konsenzus:

- getblockhash - vraćanje hash vrijednosti bloka određene visine
- getblockchaininfo - vraća najbitnije informacije o stanju blockchaina

Transakcije:

- createrawtransaction - kreacija transakcije

- gettransaction - vraća informacije o transakciji
- listunspent - vraća sve izlaze transakcija koji još nisu potrošeni

Mreža:

- getaddednodeinfo - vraća informacije o spojenim čvorovima
- getconnectioncount - vraća broj povezanih čvorova

Novčanik:

- getbalance - vraća stanje na računu
- sendtoadress - slanje određene količine bitcoina na navedenu adresu
- getnewadress - stvara novu adresu za uplate bitcoina

Rudar:

- getmininginfo - vraća informacije vezane za rudarenje
- submitblock - predlaže novi blok mreži

2.9 Baza podataka

Služi nam kako bismo sigurno i efektivno mogli lokalno spremati i pristupati podacima poput blockchaina i privatnih adresa. Uz to ova komponenta održava UTXO skupove, prati transakcije te u skladu s njima ažurira skup. Koristeći lokalno spremljene podatke, čvorovi međusobno komuniciraju i pregledavaju jesu li njihovi podaci validni te jesu li postigli konsenzus. Bitcoin Core koristi LevelDB za mapiranje i dohvat spremljenih podataka.

2.9.1 Analiza baze podataka

Korištene datoteke:

- src/txdb - služi za spremanje UTXO skupova te blockchaina
- src/dbwrapper – pruža interakciju sa samom bazom podataka
- scr/wallet/walletdb - služi kao posrednik između novčanika i baze podataka, zapravo sadrži i funkcije za obje komponente
- src/wallet/db - direktan pristup memoriji za određene podake, funkcije Read(), Write(), Erase(), Exists() i slično
- src/node/blockstorage – obavlja pohranu blokova i učitavanje istih

Najvažnije interakcije:

- Novčanik - spremanje transakcija i adresa koje novčanik sadrži
- Transakcijski skup - spremanje transakcijskoga skupa na disk
- Blockchain - spremanje lokalne kopije blockchaina kako bi čvor posjedovao točne i aktualne podatke

Najvažnije funkcije:

- Read() i Write() - čitanje i zapisivanje u bazu podataka
- Erase() - brisanje vrijednosti iz baze podataka
- WriteTx() - zapis transakcije u bazu podataka
- LoadBlockIndexDB() i WriteBlockIndexDB() - funkcije za indeksiranje blokova, indeks bloka pomaže pri pronalasku i dohvaćanju određenog bloka, jer se blokovi ne spremaju redom po njihovim visinama, već po redoslijedu po kojem su stigli na čvor

2.10 Script

Ranije u radu detaljno smo naveli svojstva i način korištenja Script jezika. Najčešće korištene skripte su Pay to Public Key Hash (P2PKH) te Pay to Witness Public Key Hash(P2WPKH) koja je modernija verzija P2PKH. Također nudi se i Pay to Script Hash (P2SH) za kompleksnije transakcije. Ovaj modul priziva se pri svakoj transakciji.

Glavna funkcija ove komponente bit će provjera može li korisnik potrošiti izlaze koristeći razne skripte.

2.10.1 Analiza Script komponente

Korištene datoteke:

- scr/script/script - glavna datoteka po pitanju Script jezika, upravlja stvaranjem i formiranjem skripti
- scr/script/interpreter - ova datoteka služi za samo izvršavanje i prikaz rezultata skripte

Najvažnije interakcije:

- Ova komponenta najčešće se koristi s validatorom kako bi se potvrdila ispravnost transakcije

Najvažnije funkcije:

- EvalScript() - pregledava ispravnost same skripte
- VerifyScript() - koristi se za validaciju transakcije
- EvalChecksig() - provjera potpisa na javni ključ kako bi se uvidjelo je li transakcija odobrena od strane vlasnika privatnog ključa

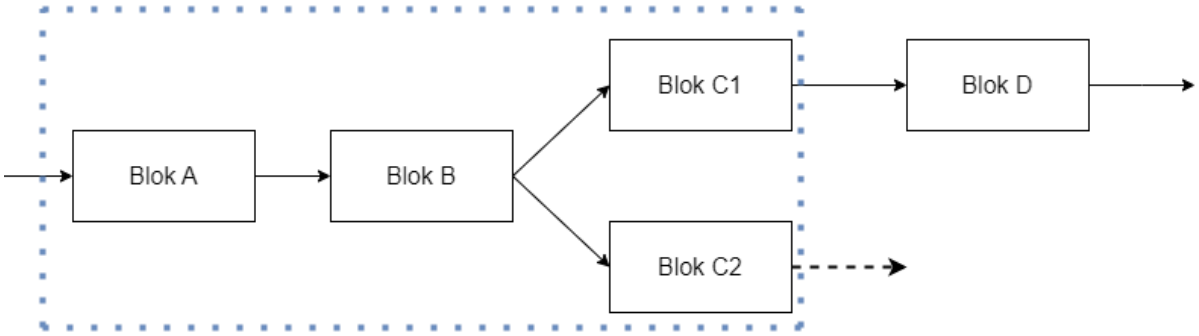
3 PRAVILA ZA POSTIZANJE KONSENZUSA

Ranije smo naveli najvažnije pravilo za postizanje konsenzusa, odnosno Proof of Work. Dogovoreno je da će svaki čvor provjeravati količinu akumuliranog PoW za sve lance za koje zna da postoje. Uz ovo najvažnije pravilo, postoji još nekolicina pravila koja služe za uspostavljanje konsenzusa.

3.1 Pravilo najdužeg lanca

Vrlo jednostavno rečeno, ovo pravilo svakome čvoru naređuje kako se ponašati u slučaju račvanja. Budući da znamo da će u Bitcoin mreži biti slučajeva kada dvojica rudara proizvedu novi blok gotovo istovremeno, sigurno je da ćemo imati dvije različite, no ispravne verzije blockchaina. Ovakav slučaj najlakše je objasniti na primjeru.

Pretpostavimo da još nije došlo do račvanja te da su zadnja dva bloka za sve čvorove u mreži blokovi A i B. Nakon što je blok B dodan, čvorovi započinju rudarenje novog bloka C. Dva rudara pronašla su rješenje u isto vrijeme i svaki od njih prosljeđuje svoj blok dalje. Blokove ćemo nazvati C1 i C2. Ovisno o tome koji blok čvor zaprimi prvo, dodaje ga u svoj lokalni blockchain i propagira dalje, dok ovaj drugi blok pri zaprimanju odbacuje. Sada su dva moguća lanca A→B→C1 i A→B→C2 (slika 3.1 označeno plavim točkastim linijama).

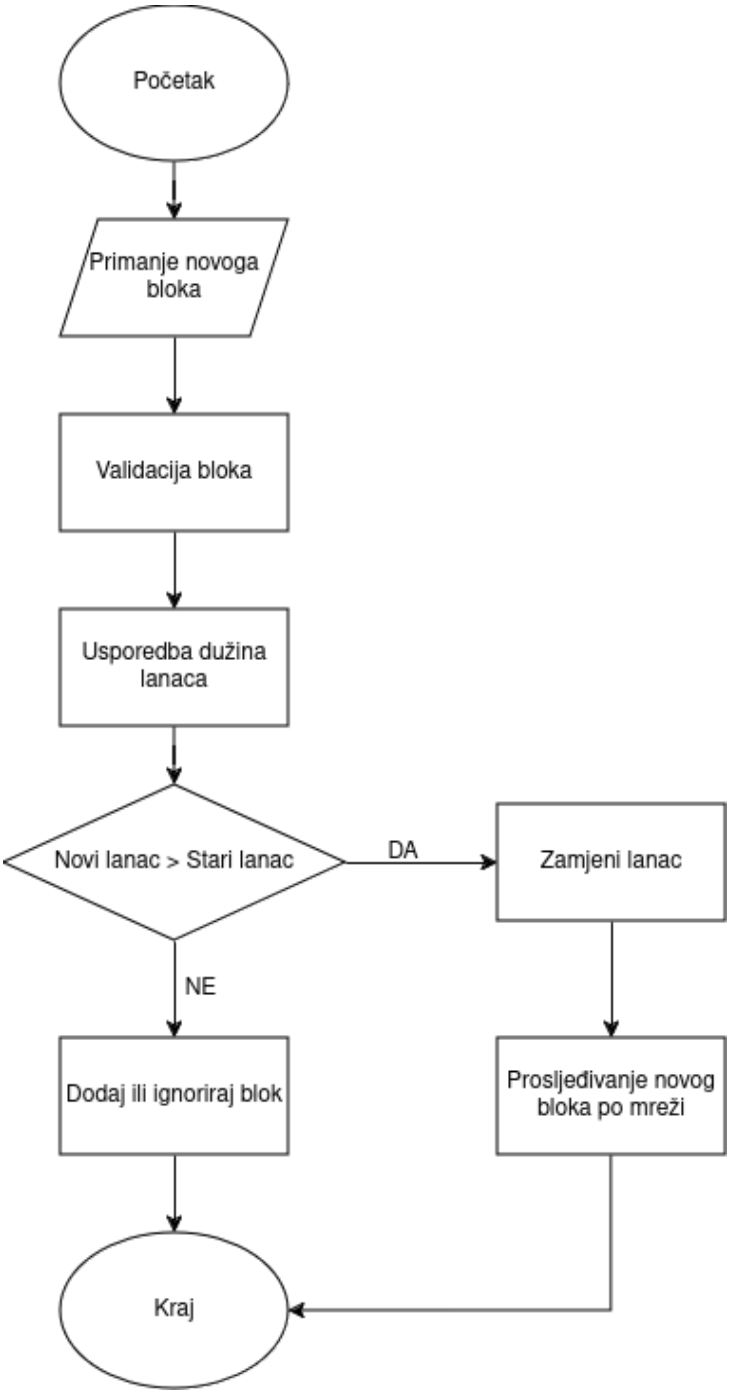


Slika 3.1 Prikaz račvanja lanaca

Svaki rudar kao parametar za blok D uzeti će zadnji blok u svojoj lokalnoj kopiji blockchaina (C1 ili C2). Recimo da je proizveden blok D kojemu je prijašnji blok C1. Sada, propagacijom bloka D čvorovi čiji je lanac sadržavao blok C2 kao vrh mogu uvidjeti da njihova lokalna kopija lanca nije ona s najvećim Proof of Work. Sada će uslijediti proces reorganizacije,

nakon kojega će propagirati dalje blok D. Kada ga svi čvorovi uvrste u svoj blockchain postići ćemo konsenzus [13].

Slika 3.2 prikazuje dijagram toka za pravilo najdužeg lanca, odnosno što će čvor učiniti kada zaprimi novi blok.



Slika 3.2 Dijagram toka za pravilo najdužeg lanca

Sama implementacija ovoga pravila nalazi se u validation.h i validation .cpp datotekama.

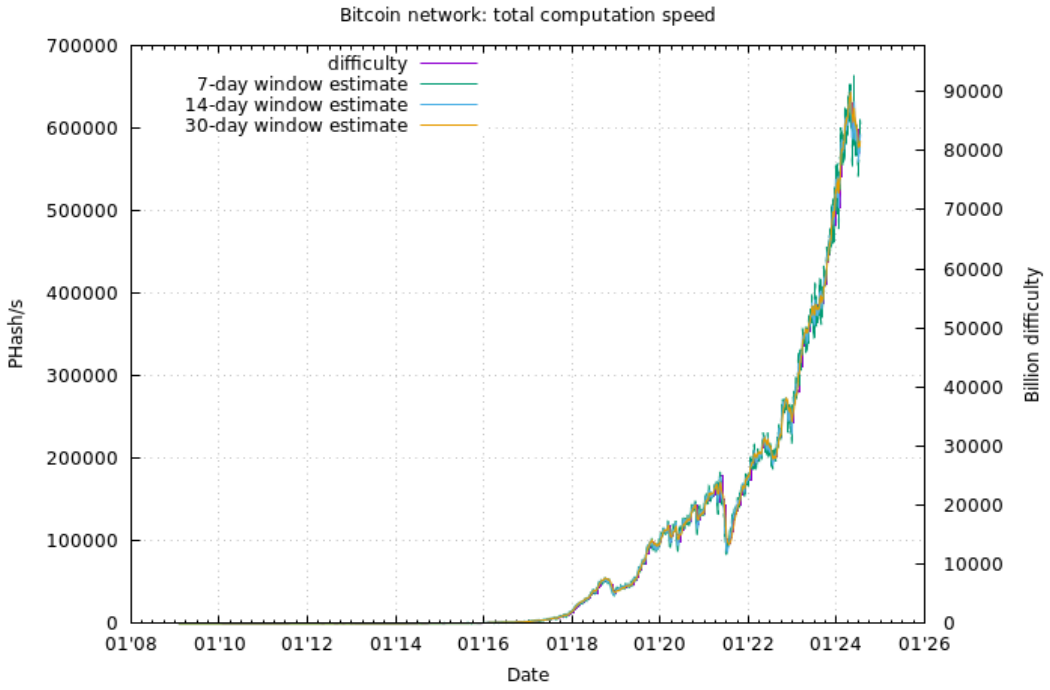
3.2 Algoritam prilagodbe težine

Još jedan bitan algoritam korišten u Bitcoinu jest algoritam prilagodbe težine (difficulty adjustment algorithm). Njime osiguravamo sigurnost i stabilnost mreže. Satoshi navodi u svome radu [1] kako će blokovi biti generirani otprilike svakih desetak minuta. Glavni motiv za implementaciju ovoga algoritma jest razvoj tehnologije, točnije porast količine hasheva u sekundi koju računala mogu izvesti, pri čemu bi se blokovi generirali prebrzo.

Dogovoreno je da se provjera težine izvodi nakon što se generiralo 2016 novih blokova. Budući da se blokovi generiraju svakih deset minuta, možemo zapaziti da se provjera dešava svakih 20160 minuta (oko četrnaest dana).

3.2.1 Mehanizam prilagodbe

Pojam težine je broj vodećih bitova u hashu koji moraju biti nula. Proporcionalno, povećanjem broja vodećih nula povećava se i potrebno vrijeme i računalna snaga potrebna za pronalazak hasha. Primjerice 2009. godine samo prvih osam bitova morali su biti nula, dok se danas zahtijeva između 72 i 80 bita. Slika 3.3 prikazuje tehnološki razvoj koji je rezultirao većom količinom obrađenih hasheva po sekundi.



Slika 3.3 Prikaz razvoja računalne snage za Bitcoin mrežu [14]

Na y-osi prikazana je snaga u mjernoj jedinici Peta hash, odnosno 10^{15} hash operacija po sekundi.

Formula kojom se računa nova težina:

$$Nova\ težina\ = Stara\ težina \times \frac{Ciljano\ vrijeme}{Stvarno\ trajanje} \qquad (3.1)$$

Ciljano vrijeme je 20160 kao što smo spomenuli prije, dok se stvarno vrijeme mjeri od trenutka kada je izvršena zadnja promjena težine do trenutka kada se nadoda još 2016 blokova. Pretpostavka jest da je broj 2016 odabran kao ravnoteža između responzivnosti i stabilnosti, te se u praksi pokazao efektivan.

Primjerice, ako smo izmjerili stvarno trajanje od 18327 minuta, a stara težina iznosi 10, imamo jednadžbu.

$$Nova\ težina\ = 10 \times \frac{20160}{18327} \qquad (3.2)$$

$$Nova\ težina\ = 11 \qquad (3.3)$$

Nagle promjene težine mogu negativno djelovati na mrežu, jer primjerice rudari koji ulaze ili izlaze iz mreže mogu uzrokovati nestabilnosti. U svrhu sprječavanja naglih promjena u težini, ona je ograničena na faktore broja četiri (4 ili ¼). Ovakvo ograničenje sprječava napad gdje bi skupina čvorova mogla ciljano prilagoditi svoju snagu rudarenja kako bi promijenili težinu prema vlastitim potrebama. Ograničavanjem postižemo stabilnost i sigurnost, jer bez naglih promjena napadači moraju potrošiti više energije kako bi izmanipulirali algoritam. Faktor četiri zapravo se kroz vrijeme pokazao kao najbolje rješenje.

3.2.2 Zašto baš deset minuta?

Satoshi već u svome radu [1] navodi da će zaglavlje bloka zauzimati 80 bajtova. Tada će godišnji rast ukupne memorijske veličine blockchaina biti oko 4.2 megabajta ne računajući transakcije, te ako se svaki blok generira u intervalu od deset minuta. No, zašto primjerice nije odabrano pet minuta, i što bismo time postigli?

Najveća prednost bila bi upravo brzina kojom možemo potvrditi transakciju. Ranije u radu (potpoglavlje 1.9) smo naveli kako se transakcija smatra sigurnom ako imamo tri potvrde, odnosno ako se na blok koji sadrži našu transakciju nadodala još tri nova bloka. Naravno, bržim generiranjem blokova ukupno vrijeme za potvrdu naše transakcije bilo bi smanjeno.

Iako se ovo čini kao vrlo bitna stavka, zapravo za većinu transakcija niskih vrijednosti uopće nije potrebno čekati više od jedne potvrde, dok bismo primjerice za transakcije većih iznosa morali čekati oko 15 minuta (tri potvrde).

Glavni problem smanjenja intervala predstavlja količina nepotrebne potrošnje hash snage, koja u Proof of Work mehanizmu ionako predstavlja veliki problem. Također, sam blockchain bi imao duplo više blokova pri čemu bi količina memorije koju blockchain zauzima bila veća. Ako bismo interval povećali potrebno vrijeme čekanja za potvrdu transakcije bi se povećalo, no količina potrošnje nepotrebne hash snage bi se smanjila, kao i memorijsko zauzeće blockchaina.

3.3 UTXO skup

UTXO (Unspent Transaction Output) skup predstavlja sve bitcoine koji su poslani i zapisani na blockchainu, no još nisu potrošeni. Ranije smo spomenuli Bitcoinove transakcije i njihove ulaze i izlaze. Možemo reći da su na UTXO skupu svi bitcoini koji se nalaze na izlazu blokova, a nisu prisutni ni u jednom ulazu u blokovima nakon. Sam zapis sadrži količinu koja je prenesena na izlazu te uvjete koji moraju biti ispunjeni kako bi se mogla potrošiti. Svi čvorovi sadrže UTXO skup pomoću kojeg i oni sami mogu validirati transakcije i blokove te ih potom propagirati dalje po mreži. Bitno je da svi poštuju ista pravila te da se slažu o trenutačnom stanju blockchaina.

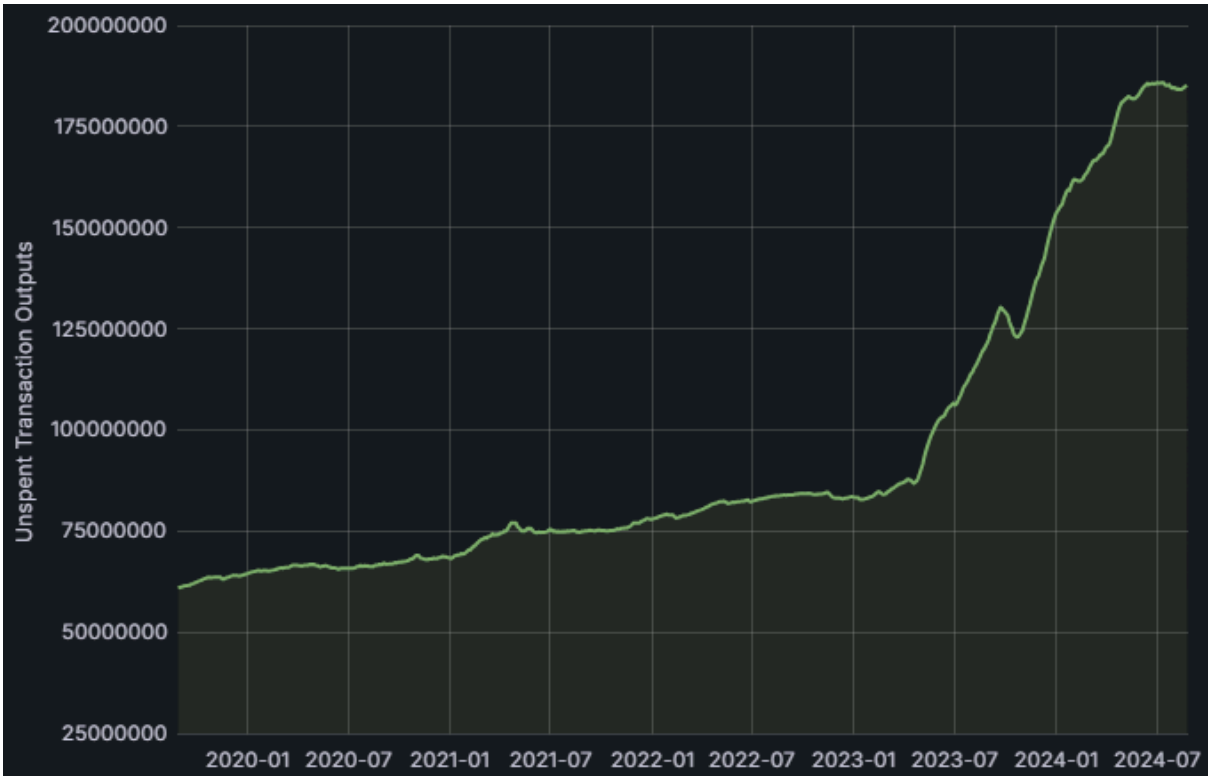
3.3.1 Upravljanje i korištenje UTXO skupa

Skup se ažurira pri dodavanju novih blokova u blockchain. Provjeravajući izlaze transakcija možemo dodati nove adrese koje sadrže UTXO ili možemo povećati iznos koji određena adresa već posjeduje. Pregledom ulaza transakcija, smanjuje se iznos koji adresa posjeduje, te u slučaju da je on jednak nuli, ta adresa se briše iz UTXO skupa.

Pri samoj validaciji transakcije, provjerava se u UTXO skupu jesu li ulazi prije potrošeni, kako bi se izbjegao problem dvostruke potrošnje. Čvorovi moraju čim prije pristupiti skupu i validirati transakciju, te je važno da mehanizmi dohvaćanja podataka i indeksiranja budu optimalni. Također čvorovi mogu odrezati dijelove blockchaina i zadržati cijeli UTXO skup kako bi uštedjeli na prostoru.

3.3.2 Problem memorije i rješenja

Povećanjem UTXO skupa, direktno se povećava i potrebna količina RAM-a koju čvor mora imati. Zadnjih par godina ovaj problem je naročito vidljiv. Slika 3.4 na y-osi prikazuje nam količinu izlaza transakcija koja je pohranjena u UTXO skupu.



Slika 3.4 Prikaz porasta veličine UTXO skupa kroz zadnjih 5 godina [15]

Neka od rješenja za uštedu prostora:

- Razdvajanje potpisa transakcije od podataka transakcije kako bi podaci o transakciji zauzimali manje memorije
- Spajanje više transakcija u jednu

3.4 Ograničenje veličine bloka

Ovo pravilo odnosi se na maksimalnu veličinu bloka koja trenutačno iznosi jedan megabajt. Uvedeno je 2010. godine kako bi se osigurala stabilnost mreže i onemogućila namjerna preopterećenja mreže. Zlonamjerni rudari mogli bi proizvesti iznimno veliki blok pri čemu bi njegova validacija i propagacija kroz mrežu prouzročila probleme.

Iako je ova ideja dodana kako bi se osigurao ispravan rad mreže, ona ima svoj veliki nedostatak. Naime, razvojem i popularizacijom Bitcoina, na mreži se broj transakcija naglo

povećavao. Budući da je dogovoreno da će se svaki novi blok stvoriti za otprilike deset minuta, a veličina bloka je fiksna, dolazi do zagušenja transakcijskog skupa. Kao najlogičnije rješenje nudilo se povećanje veličine bloka, što bi ubrzalo proces dodavanja transakcija u blockchain. Novonastali problem ovoga rješenja jest centralizacija mreže, odnosno nemogućnost prijenosa velikih blokova kod čvorova slabijeg hardvera. Radi ove odluke Bitcoin je doživio hard fork, te nastaje Bitcoin Cash kao nova valuta. Čvorovi koji se nisu slagali sa ovom odlukom zadržali su stari blockchain, dok je Bitcoin Cash blockchain prihvaćao blokove do osam megabajta.

Ova tema predstavljala je jedan od najvećih problema prošlog desetljeća u vezi Bitcoina, no danas se stanje normaliziralo razvojem novih ideja kao primjerice SegWit (odvojeni svjedok) i batching tehnika čiji je primarni cilj smanjiti veličinu transakcije.

3.5 Block halving

Block halving možemo prevesti kao prepolovljenje bloka, no ovo pravilo ne odnosi se na blok već na nagradu koju rudari dobivaju kada pronađu rješenje za PoW. Implementirano je kako bi se omogućila kontrola nad inflacijom i postigla dugoročna stabilnost bitcoina. Satoshi je u svome radu [1] spomenuo kako će ovo pravilo ograničiti količinu ukupnih bitcoina na dvadeset i jedan milijun što bi onemogućilo inflaciju koja je prisutna kod tradicionalnih valuta.

Naime, dogovoreno je da se svakih 210,000 blokova (od prilike četiri godine) nagrada koju rudari dobivaju smanjuje za 50%. Početna nagrada bila je 50 bitcoina za pronalazak rješenja, a za vrijeme pisanja ovog rada iznosi 3.125 bitcoina. Budući da vidimo da su rudari sve manje i manje plaćeni, postavlja se pitanje isplati li se rudariti. Ovo će predstavljati problem za rudare jer će njihovi troškovi rudarenja ostati isti dok će se nagrada dobivena za blok smanjiti. Rješenje za ovaj problem Nakamoto vidi u provizijama za transakcije. Smatra da će razvojem Bitcoin mreže doći će do porasta broja transakcija i samim time većom zaradom od provizije, jer se bloku uz isplatu novih bitcoina isplaćuje i zarada od provizije za sve transakcije uvrštene u blok.

4 DODATNE KOMPONENTE

Do sada smo nabrojali komponente koje su krucijalne za uspješan rad Bitcoina. Sada ćemo spomenut komponente koje nam daju dodatne mogućnosti ili pozitivno utječu na sigurnost mreže.

4.1 Grafičko korisničko sučelje (GUI)

Iako je Bitcoin Core poznat po svojoj komandnoj liniji, nudi opciju za grafičko sučelje kako bi se olakšalo korištenje aplikacije. Po nazivu mape vidimo da se koristi Qt razvojni okvir (framework) za kreiranje grafičkog sučelja.

4.1.1 Analiza grafičkog sučelja

Korištene datoteke:

- src/qt/main i src/qt/bitcoin - služe za pokretanje glavne aplikacije i definiranje funkcionalnosti kojima grafičko sučelje upravlja
- src/qt/walletview - služi za prikaz podataka koji se nalaze u komponenti novčanik koristeći grafičko sučelje
- src/qt/bitcoingui – glavna datoteka za sinkronizaciju grafičkog sučelja s ostalim komponentama, također generira osnovne komponente grafičkog sučelja (menu, toolbar itd.)

Naravno sve datoteke unutar src/qt mape imaju svoju važnost, neke se odnose na komunikaciju između grafičkog sučelja i ostatka sustava, dok neke služe za vizualizaciju i sam izgled komponenti.

Najvažnije interakcije:

- RPC - preko ovoga modula grafičko sučelje dobiva informacije od Bitcoin Core sustava
- Novčanik - grafičko sučelje daje korisniku mogućnost da upravlja svojim novčanikom kao primjerice pregled transakcija i slanje bitcoin valute

Najvažnije funkcije:

- GuiMain() - pokreće GUI aplikaciju

- UpdateWalletStatus() - pri svakoj promjeni stanja novčanika poziva se ova funkcija kako bi se pokazale najnovije ispravne informacije

4.2 Tor mreža

Tor mreža pomaže korisniku ostati anoniman pri spajanju na mrežu. U slučaju Bitcoina osigurat će anonimnu komunikaciju između čvorova. Tor može sakriti IP adresu pri komunikaciji čime je praćenje čvora otežano.

4.2.1 Analiza Tor mreže

Korištene datoteke:

- src/torcontrol - gotovo sav kod vezan za samu implementaciju Tor mreže nalazi se ovdje

Najvažnije interakcije:

- Mreža - sam Tor dodaje još jedan sigurnosni sloj na mrežu kako bi se zaštitili podaci čvora

Najvažnije funkcije:

- StartTorControl() - uspostavlja vezu s Tor mrežom
- StopTorControl() - zaustavlja vezu s Tor mrežom

4.3 Sigurnosni mehanizmi

Bitcoin Core implementira sigurnosne mehanizme u svrhu zaštite podataka i sigurnog rada mreže.

- **Zaštita od Dos napada** - ranije spomenuti mrežni modul sadrži implementaciju ovoga mehanizma. Ukratko, čvorovi prate ponašanje drugih čvorova, te u slučaju slanja velike količine poruka ili slanja krivih poruka mogu korištenjem kaznenih bodova kazniti taj čvor
- **Zaštita od ponovljenih napada** - korištenje unikatnih oznaka transakcija kako bi se onemogućilo ponavljanje transakcije nakon pojave račvanja. Cilj je izbjeći slučaj gdje napadač može poslati transakciju iz jednog lanca koja nije prisutna u drugome lancu, gdje će je drugi lanac smatrati ispravnim što bi rezultiralo financijskim gubitkom

5 ZAKLJUČAK

Funkcionalna analiza u programiranju ima svrhu podjele sustava i analiziranja svakog pojedinog dijela. Bitno je da analiza odgovara na pitanje što se očekuje od komponente, no ne i kako će se to rješenje implementirati. Izrađenom funkcionalnom analizom lakše se možemo sporazumjeti s ostalim sudionicima na projektu.

U ovome radu cjelokupni Bitcoin sustav sveden je na osnovne funkcije kako bi se stekao dojam o načinu na koji je sustav posložen. Način podjele samoga sustava na komponente je subjektivan, no bazira se na glavnim funkcionalnostima koje su prisutne u Bitcoinu.

Završetkom funkcionalne analize kao sljedeći korak nameće se implementacija, primjerice izmjena dijelova koda za osobne svrhe i testiranje. Također jedna od mogućnosti jest naprimjer usporedba s Ethereum sustavom i osnovnim funkcijama koje su tamo prisutne.

U izvršavanju ove analize stekao sam nove spoznaje i produbio znanje o Bitcoinu i samoj funkcionalnoj analizi kao alatu. Stečena znanja uvelike će mi pomoći za profesionalni razvoj u struci.

Literatura

[1] Nakamoto, S.: "Bitcoin: „A Peer-to-Peer Electronic Cash System“ ", 2008.

[2] Andreas, A.: "Mastering Bitcoin 2nd edition", O'Reilly Media, 2017.

[3] Investopedia team: "What Is a Block in the Crypto Blockchain, and How Does It Work?", s Interneta, https://www.investopedia.com/terms/b/block-bitcoin-block.asp, 20. kolovoza 2024.

[4] Lemieux, V.: "Blockchain for Recordkeeping: Help or Hype?", s Interneta, https://www.researchgate.net/figure/Simplified-Blockchain-Source-Bitcoinorg-2015_fig1_309414363, 7. kolovoza 2024.

[5] Pandey, R. i dr.: "Distributed Computing to Blockchain", Academic Press, 2023.

[6] Naor, M.; Dwork, C.: "Pricing via Processing or Combatting junk mail", Springer, Berlin, 1993.

[7] Shuaib, M. i dr.: "A Novel Optimization for GPU Mining Using Overclocking and Undervolting", MDPI, Basel, 2022.

[8] Walker, G.: "Keys", s Interneta, https://learnmeabitcoin.com/technical/keys/#private-key, 19. kolovoza 2024.

[9] Rybarczyk, R.: "Bitcoin P2PKH Transaction Breakdown", s Interneta, https://medium.com/coinmonks/bitcoin-p2pkh-transaction-breakdown-bb663034d6df, 12. kolovoza 2024.

[10] "Blockchain Merkle Trees", s Interneta, https://www.geeksforgeeks.org/blockchain-merkle-trees/, 29. srpnja 2024.

[11] "How Are Blockchain Transactions Validated? Consensus VS Validation", s Interneta, https://medium.com/hupayx/how-are-blockchain-transactions-validated-consensus-vs-validation-ada9c001fd0a, 12. kolovoza 2024.

[12] "RPC API Reference", s Interneta, https://developer.bitcoin.org/reference/rpc/, 17. kolovoza 2024.

[13] Walker, G.: "Longest Chain", s Interneta, https://learnmeabitcoin.com/technical/blockchain/longest-chain/, 31. srpnja 2024.

[14] "Bitcoin", s Interneta, https://bitcoin.sipa.be/, 18. kolovoza 2024.

[15]

"Unspent Transaction Output Set", s Interneta, https://statoshi.info:3000/d/000000009/unspent-transaction-output-set?viewPanel=6&orgId=1&from=now-5y&to=now&refresh=5s, 24. kolovoza 2024.

POJMOVNIK

DAA – Difficulty Adjustment Algorithm

GUI – Graphical User Interface

IP – Internet Protocol

PoW – Proof of Work

P2PKH - Pay To Public Key Hash

P2SH – Pay To Script Hash

P2P – Peer To Peer

P2WPKH - Pay To Witness Public Key Hash

RAM – Random Access Memory

RPC – Remote Procedure Call

SHA 256 – Secure Hash Algorithm 256

UTXO - Unspent Transaction Output

Sažetak

Kvalitetno provedena funkcionalna analiza uvelike nam olakšava razumijevanje kompleksnih sustava kao što je Bitcoin. Svojom decentraliziranom prirodom sam sustav sadrži jedinstvene koncepte koje ne nalazimo često u drugim sustavima. Funkcionalna analiza u ovome radu temeljila se na podjeli sustava u manje cjeline kako bi korisniku pružila uvid svih potrebnih komponenti za uspješan rad sustava. Rad pruža dokumentaciju svih komponenti s kratkim objašnjenjem čemu one služe i interakcijama s drugim komponentama. Prikazana su i pravila konsenzusa koja donose stabilnost mreži i spomenuti su neki od najvećih problema s kojima se Bitcoin sustav susretao od svojih početaka pa sve do danas.

Ključne riječi: Bitcoin čvor, funkcionalna analiza, dokumentacija, blockchain, konsenzus, Bitcoin sustav

Abstract

A good functional analysis can be very useful for understanding a complex system such as Bitcoin. With its decentralized nature, the system contains unique concepts we won't usually find in other systems. In this thesis, functional analysis was made by splitting the system into its core functions to give the reader a better understanding of all important components that allow the system to work properly. This thesis provides documentation for all those components while briefly explaining them and mentioning their most important interactions with other components. Rules of consensus that bring agreement to the network were also shown, along with some of the problems Bitcoin has faced since its release.

Keywords: Bitcoin node, functional analysis, documentation, blockchain, consensus, Bitcoin system

Dodatak A: GitHub repozitorij i stablasti prikaz mapa i datoteka

Sve navedene datoteke i njihov kod nalaze se na online platformi GitHub u repozitoriju bitcoin/bitcoin/ (https://github.com/bitcoin/bitcoin, zadnje pristupano 28.8.2024.).
Budući da se i u samoj src/ mapi nalazi više tisuća datoteka, sama vizualizacija nije jednostavna. Za pregled cjelokupne strukture predlaže se preuzimanje koda u cjelini i korištenje file explorera. U nastavku je nadodan pojednostavnjeni stablasti prikaz direktorija i datoteka unutar mape src/. Prvo su prikazane datoteke unutar same src/ mape, a nakon njih dolazi lista važnijih podmapa. Za određene podmape bit će prikazane i neke datoteke koje se nalaze u njima.

```
├── src/
│   ├── addrdb.cpp
│   ├── addrdb.h
│   ├── addrman.cpp
│   ├── addrman.h
│   ├── addrinfo.cpp
│   ├── addrinfo.h
│   ├── alert.cpp
│   ├── alert.h
│   ├── amount.h
│   ├── args.cpp
│   ├── args.h
│   ├── assets.cpp
│   ├── assets.h
│   ├── banman.cpp
│   ├── banman.h
│   ├── base58.cpp
│   ├── base58.h
│   ├── base64.cpp
│   ├── base64.h
│   ├── bech32.cpp
│   ├── bech32.h
│   ├── blockfilter.cpp
│   ├── blockfilter.h
│   ├── blockencodings.cpp
│   ├── blockencodings.h
│   ├── blockpolicy.cpp
│   ├── blockpolicy.h
│   ├── bloom.cpp
│   ├── bloom.h
│   ├── chain.cpp
│   ├── chain.h
│   ├── chainparams.cpp
│   ├── chainparams.h
│   ├── chainparamsbase.cpp
│   ├── chainparamsbase.h
│   ├── chainparamsconstants.cpp
│   ├── chainparamsconstants.h
│   ├── chainparamsseeds.cpp
│   ├── chainparamsseeds.h
│   ├── checkqueue.cpp
│   ├── checkqueue.h
│   ├── coins.cpp
│   ├── coins.h
│   ├── compressor.cpp
│   ├── compressor.h
│   ├── compress.cpp
│   ├── compress.h
│   ├── core_io.cpp
│   ├── core_io.h
│   ├── core_memusage.h
│   ├── core_read.cpp
│   ├── core_read.h
│   ├── core_write.cpp
│   ├── core_write.h
│   ├── cpuid.cpp
│   ├── cpuid.h
│   ├── dbwrapper.cpp
│   ├── dbwrapper.h
│   ├── deserializer.h
│   ├── dstencode.cpp
│   ├── dstencode.h
│   ├── fs.cpp
│   ├── fs.h
```

```
│   ├── hash.cpp
│   ├── hash.h
│   ├── httprpc.cpp
│   ├── httprpc.h
│   ├── httpserver.cpp
│   ├── httpserver.h
│   ├── init.cpp
│   ├── init.h
│   ├── key.cpp
│   ├── key.h
│   ├── key_io.cpp
│   ├── key_io.h
│   ├── libbitcoinconsensus.cpp
│   ├── libbitcoinconsensus.h
│   ├── mapport.cpp
│   ├── mapport.h
│   ├── memusage.h
│   ├── merkleblock.cpp
│   ├── merkleblock.h
│   ├── miner.cpp
│   ├── miner.h
│   ├── net.cpp
│   ├── net.h
│   ├── netbase.cpp
│   ├── netbase.h
│   ├── net_permissions.cpp
│   ├── net_permissions.h
│   ├── net_processing.cpp
│   ├── net_processing.h
│   ├── netaddress.cpp
│   ├── netaddress.h
│   ├── netgroup.cpp
│   ├── netgroup.h
│   ├── netmessagemaker.h
│   ├── outputtype.cpp
│   ├── outputtype.h
│   ├── protocol.cpp
│   ├── protocol.h
│   ├── psbt.cpp
│   ├── psbt.h
│   ├── pubkey.cpp
│   ├── pubkey.h
│   ├── random.cpp
│   ├── random.h
│   ├── rest.cpp
│   ├── rest.h
│   ├── reverse_iterator.h
│   ├── scheduler.cpp
│   ├── scheduler.h
│   ├── shutdown.cpp
│   ├── shutdown.h
│   ├── signet.cpp
│   ├── signet.h
│   ├── streams.h
│   ├── sync.cpp
│   ├── sync.h
│   ├── threadinterrupt.cpp
│   ├── threadinterrupt.h
│   ├── threadsafety.h
│   ├── timedata.cpp
│   ├── timedata.h
│   ├── torcontrol.cpp
│   ├── torcontrol.h
│   ├── trafficgraphdata.cpp
```

```
│   ├── trafficgraphdata.h
│   ├── txdb.cpp
│   ├── txdb.h
│   ├── txmempool.cpp
│   ├── txmempool.h
│   ├── ui_interface.h
│   ├── uint256.cpp
│   ├── uint256.h
│   ├── undo.h
│   ├── validation.cpp
│   ├── validation.h
│   ├── validationinterface.cpp
│   ├── validationinterface.h
│   ├── version.cpp
│   ├── version.h
│   ├── warnings.cpp
```

```
│   ├── compat/
│   ├── consensus/
│   │   ├── amount.h
│   │   ├── block.h
│   │   ├── consensus.h
│   │   ├── merkle.cpp
│   │   ├── merkle.h
│   │   ├── params.h
│   │   ├── tx_verify.cpp
│   │   ├── tx_verify.h
│   │   └── validation.h
│   ├── crypto/
│   ├── interfaces/
│   ├── logging/
│   ├── mempool/
│   ├── node/
│   │   ├── blockstorage.cpp
│   │   ├── blockstorage.h
│   │   ├── chainstate.cpp
│   │   ├── chainstate.h
│   │   ├── coin.cpp
│   │   ├── coin.h
│   │   ├── mempool_args.cpp
│   │   ├── mempool_args.h
│   │   ├── psbt.cpp
│   │   ├── psbt.h
│   │   ├── transaction.cpp
│   │   ├── transaction.h
│   │   ├── utxo_snapshot.cpp
│   │   └── utxo_snapshot.h
│   ├── policy/
│   │   ├── fees.cpp
│   │   ├── fees.h
│   │   ├── fee_estimator.cpp
│   │   ├── fee_estimator.h
│   │   ├── packages.cpp
│   │   ├── packages.h
│   │   ├── policy.cpp
│   │   ├── policy.h
│   │   ├── settings.cpp
│   │   └── settings.h
│   ├── pow/
│   ├── primitives/
│   ├── qt/
│   ├── rpc/
│   │   ├── blockchain.cpp
│   │   ├── blockchain.h
│   │   ├── context.cpp
│   │   ├── context.h
│   │   ├── getblocktemplate.cpp
│   │   ├── getblocktemplate.h
│   │   ├── miner.cpp
│   │   ├── miner.h
│   │   ├── misc.cpp
│   │   ├── misc.h
│   │   ├── net.cpp
│   │   ├── net.h
│   │   ├── rawtransaction.cpp
│   │   ├── rawtransaction.h
│   │   ├── server.cpp
│   │   ├── server.h
│   │   ├── util.cpp
│   │   └── util.h
│   ├── script/
```

```
│   │   ├── bitcoinconsensus.cpp
│   │   ├── bitcoinconsensus.h
│   │   ├── descriptor.cpp
│   │   ├── descriptor.h
│   │   ├── ecdsa.cpp
│   │   ├── ecdsa.h
│   │   ├── interpreter.cpp
│   │   ├── interpreter.h
│   │   ├── miniscript.cpp
│   │   ├── miniscript.h
│   │   ├── script.cpp
│   │   ├── script.h
│   │   ├── sign.cpp
│   │   ├── sign.h
│   │   ├── standard.cpp
│   │   └── standard.h
│   ├── support/
│   ├── util/
│   ├── test/
│   ├── wallet/
│   │   ├── context.cpp
│   │   ├── context.h
│   │   ├── crypter.cpp
│   │   ├── crypter.h
│   │   ├── db.cpp
│   │   ├── db.h
│   ├── ├── external_signer_scriptpubkeyman.cpp
│   ├── ├── external_signer_scriptpubkeyman.h
│   │   ├── feebumper.cpp
│   │   ├── feebumper.h
│   │   ├── flatdb.cpp
│   │   ├── flatdb.h
│   │   ├── init.cpp
│   │   ├── init.h
│   │   ├── interfaces.cpp
│   │   ├── interfaces.h
│   │   ├── ismine.cpp
│   │   ├── ismine.h
│   │   ├── keystore.cpp
│   │   ├── keystore.h
│   │   ├── load.cpp
│   │   ├── load.h
│   │   ├── outputtype.cpp
│   │   ├── outputtype.h
│   │   ├── rpcwallet.cpp
│   │   ├── rpcwallet.h
│   │   ├── sapling.cpp
│   │   ├── sapling.h
│   │   ├── scriptpubkeyman.cpp
│   │   ├── scriptpubkeyman.h
│   │   ├── tx.cpp
│   │   ├── tx.h
│   │   ├── wallet.cpp
│   │   ├── wallet.h
│   │   ├── walletdb.cpp
│   │   ├── walletdb.h
│   │   ├── wallettool.cpp
│   │   ├── wallettool.h
│   │   ├── walletutil.cpp
│   │   └── walletutil.h
```

U slučaju da čitatelj želi stablo s apsolutno svim datotekama evo što mora učiniti (navedeni koraci odnose se na Windows korisnike, iako je postupak vrlo sličan i za Mac i Linux korisnike):

1. Preuzeti repozitorij s githuba
2. Pronaći odgovarajuću mapu na računalu
3. Stisnuti tipku shift te desni klik na spomenutu mapu
4. Odabrati opciju otvori u terminalu
5. Upisati komandu „tree /f /a > tree.txt“ i pritisnuti enter
6. Unutar mape pronaći datoteku tree.txt i otvoriti je u MS wordu
7. U pop-up prozoru odabrati unicode jezik i otvoriti datoteku