Sveučilište u Rijeci – Odjel za informatiku

Naziv preddiplomskog sveučilišnog studija

Gabriel Saganić

Jednostavni Blockchain sustav u c++

Završni rad

Mentor: doc. dr. sc. Marina Ivašić Kos

Rijeka, rujan, 2018

Sadržaj

Zadatak završnog rada

Uvod

Blockchain koncept teoretski je zamšljen još 90ih godina ali za praktičnu primjenu čekalo se sljedećih 20 godina. 2008. godine pod pseudonimom Satoshi Nakamoto stvorio je kod za kriptovalutu Bitcoin. Sam koncept i tehnologije koje se primjenjuju u ovome radu rezultat su brojnih istraživanja na području matematike, kriptografije i računalne sigurnosti.

U ovom radu objasniti ću sa teoretske strane koncept blockchaina i njegovih osnovnih elementa. Svaki element, funkcije i procesi koji se odvijaju u sustavu će biti teoretski objašnjen te će nakon toga biti prikazana implementacija osnovnih koncepata u C++ programskom jeziku. Objašnjeno je i pokazano kako se stvaraju jedinstvene korisničke adrese za novčanike koji se koriste za pohranu vrijednosti koje se prenose između korisnika i funkcije koje se koriste za transakcije u blockchainu i njihovo kriptiranje.

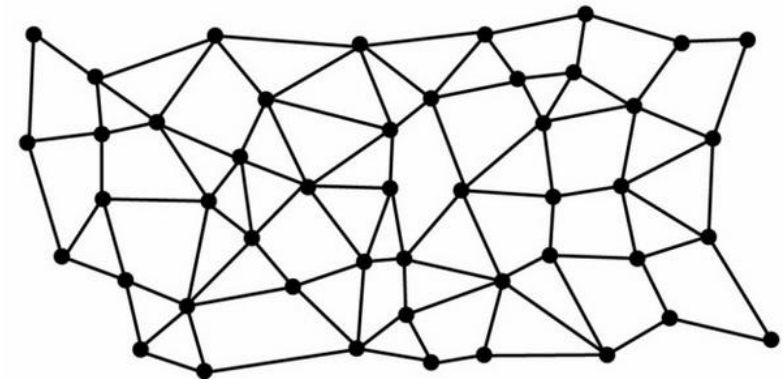
U radu će biti priložena jednostavna simulacija blockchain sustava u C++ koja služi za spremanje transakcija između dvije varijable u blockchainu te za provjeru i kriptiranje blokova i transakcija.

Ključne riječi(Blockchain, blok, distribuirani sustav, hash, transakcija)

Teorija blockchain tehnologije

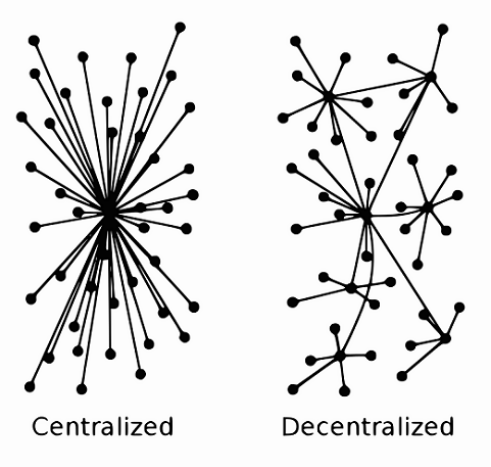
Distribuirani sustav i decentralizirani sustav

Distribuirani sustav je mreža koja se sastoji od autonomnih čvorova koja su spojena pomoću distribucijske mreže (Imran Bashir, Mastering Blockchain, 2017, str 10)24. Pomažu u dijeljenju različitih resursa i mogućnosti kako bi korisnicima pružili jedinstvenu mrežu. U distribuiranim sustavima nema središnjeg autoriteta. Svaki čvor je povezan sa svakim drugim čvorom i ima isti autoritet. Distribuirani sustavi sastoje se od velikog broja istovrsnih procesa, takozvanih čvorova, Slika 1. Čvorovi obavljaju zadane zadaće prema potrebama svojih korisnika. Kada je čvoru pri obavljanju neke zadaće potrebna pomoć on stupa u komunikaciju sa susjednim čvorom, a ti susjedi sa svojim susjedima i tako se komunikacija odvija na razini cijelog sustava.



Slika 1: Distribuirani sustav (Izvor: Usman M. Sheikh, Virtual currency: current regulatory and civil litigation trends, 2018)

Distribuirani sustavi mogu se podijeliti prema strukturi na centralizirane i decentralizirane, Slika 2. Kod centraliziranih sustava postoji poslužitelj koji osigurava zadovoljavajuću razinu performansi i kvalitete, kod decentraliziranih sustava poslužitelja nema, te sami korisnici i dobro promišljeni sustavi obavljaju poslove poslužitelja.



Slika 2: Centraliziran i decentraliziran sustav (Izvor : Stewart Martins, Understanding Blockchain: A Simple Approach, (2018), https://blockchainexplained.org/understanding-blockchain/)

U slučaju digitalnog novca, poslužitelj se brine za sigurnost valute kako bi se onemogućilo i spriječilo krivotvorenje, osigurava primjerenu kvalitetu i sigurnosnu razinu tehničkog rješenja i brine se za njegovo održavanje te povezuje klijente kako bi mogli međusobno komunicirati.

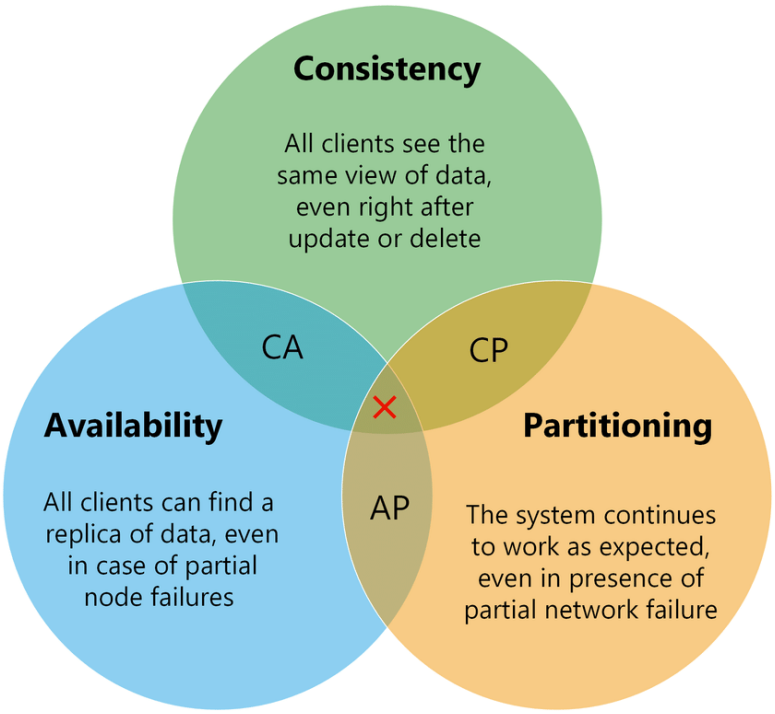
Izdavatelj valute, prije pojave digitalnih valuta baziranih na Blockchain tehnologiji, bio je određeni fizički subjekt koji je jamčio autentičnost i vrijednost valute. Više o izdavatelju valute i marketingu možete pročitati u knjizi Marc Levinson, Guide to financial markets. Nastankom Blockchaina odnosno Bitcoina, prve kripto valute, sustavi počinju koristiti decentralizirane distribuirane sustave. Time je omogućena razmjenu podataka kroz računalnu mrežu pri kojem čvorovi preuzimaju informacije jedni od drugih umjesto s jednog centralnog poslužitelja. Kad se dizajnira decentralizirani sustav mora se uzeti u obzir probleme koje rješava poslužitelj u centraliziranim sustavima. To su najčešće krivotvorini podaci i sigurnost sustava. Sigurnost se postiže konsenzusom između čvorava i CAP teoremom (CAP teorem). Što je veći broj čvorova potvrdilo transakciju, manji je rizik od krivotvorenih napada. Upotrebom POW (Proof of Work) mehanizma, čvorovi imaju veću motivaciju za održavanje sustava i snimanje transakcija. CAP teorem i POW mehanizam detaljnije ću objasnit dalje u seminaru.

CAP teorem

Sustav koji funkcionira na mreži s više čvorova zovemo distribuirani sustav. Jedan od glavnih problema takvih sustava odnose se na samu funkcionalnost mreže, odnosno veza između čvorova. Samim time nastale su brojne strategije vezane za dostupnost podataka na navedenim sustavima. CAP teorem prikazuje presjek različitih težišta važnosti kod raznih sustava, Slika 3.

Kao glavne težišne točke navode se:

1. Dosljednost (consistency) - svi klijenti vide trenutni identičan set podataka bez obzira na promjene i brisanje
2. Dostupnost (availability) – svi korisnici mogu vidjeti podatak bez obzira na grešku na čvoru
3. Tolerancija na fragmentiranje (partitoning) - sustav nastavlja raditi bez obzira na mrežnu grešku



Slika 3: Vizualni prikaz CAP teorema *(Izvor: How do i choose the right NoSQL solution? A comprehensive theoretical and experimental survey)*

U teorijskoj računarskoj znanosti, CAP teorem, također nazvan Brewerov teorem prema računalnom znanstveniku Ericu Breweru, tvrdi da je nemoguće da distribuirani sustavi istovremeno pružaju više od dvije težišne točke.

Kako je Blockchain distribuirani sustav bez centralne zaštite moguće je da svaki čvor napravi različitu listu transakcija. Ovaj problem je poznat pod nazivom račvanje (engl. forking). Moramo zaključiti koliko blokova mora doći poslije transakcije da da bi tu transakciju potvrdili. Što vise blokova dođe poslije transakcije veća je vjerojatnost da je blok dosljedan. U distribuiranim sustavima treba odlučiti dali će se čekati da blok postane dosljedan ili ne, što čini sustav AP (dostupnost i tolerancija na fragmentiranje) ili CP (dosljednost i tolerancija na fragmentiranje). Da bi se postao AP, klijent bi trebao prihvatiti transakciju čim se doda u Blockchain. Na taj način, ne postoji ovisnost o ostatku vršnjaka, što je čini dostupnim, ali

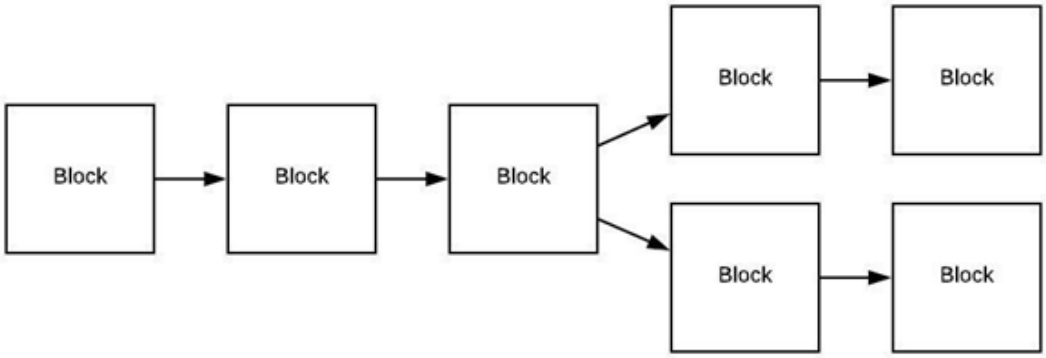
postoji rizik da će preostali vršnjaci odbiti transakciju, žrtvujući dosljednost. Da bi bio CP, klijent bi trebao prihvatiti transakciju nakon što je lanac blokova postigao konsenzus o tome. To ima invertni učinak, gdje su podaci dosljedni, ali postoji rizik da postane nedostupan ako postoji mrežna podjela koja sprečava konsenzus. Da bi bili sigurni da je sustav konzenstan na transakciju ne smijemo ništa tretirati kao „gotovo“ dok nije barem X blokova se povezalo nakon transakcije. Sposobnost konfiguriranja klijenta na ovaj način ne krši CAP teorem. To je zato što je ova vrsta konfiguracije nadoknađivanje između dostupnosti i dosljednosti, istodobno ne može imati oba.

Račvanje u blockchainu

Račvanje (engl. forking) je ono sto se desi kad se blockchain podjeli na dva lanca, tada dobijemo dva bloka koje dijele istog roditelja i identični su sve do točke račvanja, Slika 4. Račvanje je uvedeno u blockchain da se izbjegnu hakerski napadi ili greške nastale na lancu, što je bio slučaj na Bitcoinu 2010 godine. Račvanje je različito do svakog blockchaina, ovisno o arhitekturi i svrsi lanaca. U ovom radu objasnit ću opći način račvanja. Račvanje se može desiti u dva slučaja:

1. Dva rudara su potvrdili blok u relativno kratkom vremenskom razdoblju
2. Programer je promijenio pravilo po kojem se odlučuje konsenzus bloka

Prvi slučaj se desi kad prvi rudar koje je pronašao odgovarajući hash nije stigao obavijestiti ostale rudare u sustavu da je trenutni blok potvrđen i da mogu početi raditi na dodavanju novog bloka. Neki rudari će primiti blok od prvog rudara dok će drugi primiti blok od drugog rudara. Nakon nekog vremena će svi rudari dobiti i drugi blok i tada nastaje račvanje.



Slika 4: Račvanje u blockchainu (Henry He, Blockchain Basics – Fork,(2018), https://www.c-sharpcorner.com/article/blockchain-basic-fork/)

Budući da su oba bloka dobivani putem Proof of Work i oba su valjana, svi rudari prate oba bloka. Tada rudari rade na dodavanju novog bloka na onom lancu u kojem su prvi dobili blok. Kad rudar doda novi blok na lanac, po pravilu najduljeg lanca, svi rudari počinju raditi na produljenu tog lanca i drugi lanac se odbacuje. Podaci koji su se nalazili u lancu koje je odbačen se ignoriraju i ne smatraju se djelom blockchaina. Moguće je da su podaci koji su bili zapisani u bloku koji je odbačen već upisani u blockchain u blok koji je nastao u otprilike isto vrijeme kao i odbačeni. Ako to nije slučaj bit će zapisani nakon nekog vremena jer su rudari primili te podatke i oni se nalaze među kandidatima za novi blok.

Kod drugog slučaja, kad se blockchain verzija promjeni, prijašnja i nova verzija se potpuno razdvoje. Uglavnom nova verzija pokupi svu povijest podataka i na dalje svaka verzija ima svoju povijest podataka.

Proof of Work

Proof of Work (POW) je najpoznatiji algoritam koji koriste kripto valute poput Bitcoina i Ethereum. Proof of Work sustav ekonomska je mjera koja sprečava napade na sustav i druge zlouporabe poput spam-a. Koncept su predložili Cynthia Dwork i Moni Naor, te je prvi put prezentiran u novinama 1993. godine. Da bi razumjeli Proof of Work prvo trebamo poznavati pojmove poput Hash-a i rudara (engl. miners).

Hash

Hash funkcija je temeljni dio Blockchain tehnologije i da bi razumjeli Blockchain prvo moramo dobro poznavati hashiranje. Hash funkcije je bilo koja funkcija koja može „mapirati“ podatak proizvoljne veličine u podatak fiksne veličine. Preciznije rečeno, hash funkcija uzima poruku fiksne duljine i za izlaz daje šifriranu poruku.

Slika 5: Dijagram aktivnosti hash funkcije bez ključa

Hash funkcije možemo podijeliti na dvije vrste: hash funkcije bez ključa, Slika 5, koje za ulaz imaju samo poruku i hash funkcije s ključem koje uz poruku za ulaz imaju i privatni ključ. Kod hash funkcije s ključem uz poruku se šalje i MAC oznaka koju dobivamo upotrebom MAC algoritma i privatnog ključa. Kada primatelj, koji ima isti privatni ključ, primi poruku može MAC algoritmom i privatnim ključem dobiti MAC oznaku i tako potvrditi autentičnost poruke. Vrijednost dobivena hash funkcijom nazivamo hash. Hash je najčešće string nasumičnih slova

i brojeva. Najčešće korištena hash funkcija je SHA-256 (Imran Bashir, Mastering Blockchain, 2017, str 91), koju koristi Bitcoin, a korištena je i u jednostavnoj Blockchain simulaciji. Na primjer:
za ulaz „*Blockchain*“ izlaz SHA-256 funkcije je
625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1,
a za ulaz „*Blockchain2*“ dobijemo izlaz
4f07e031df167abda5623314c19c38e11358853f1c876b892a1d2ae494cc1058.

Iz ovog primjera vidimo da je gotovo nemoguće otkriti ulaz promatranjem samo hash-a, te da samo najmanja promjena u ulazu kompletno mijenja izlaz. Hash funkcije su jednosmjerne funkcije, nemaju inverznu funkciju. Jedini način da se dobiju ulazni podaci hash funkcije iz izlaza je bruteforce algoritmom (pokušavanje svih mogućih kombinacija), ali za takvo nešto bi trebala sva računala na svijetu da rade od početka svijeta do danas. Bitno svojstvo hash funkcije je kompresija, veliki ulaz je komprimiran u vrlo kratki niz tog unosa. U blockchain tehnologiji je bitno da sto više podataka ulazi u hash funkciju da je teže ili gotovo nemoguće otkriti ulaz.

```
_hash=hashing(patch::to_string(index)+prev_hash+patch::to_string(_time)+patch::to_string(nonce)+tmpTransaction.getHash());
```

Slika 6: primjer informacija koje ulaze u hashiranje

Na slici 6 vidimo primjer formule za izračunavanje hasha koja koristi: redni broja bloka, hash prijašnjeg bloka, trenutno vrijeme, nonce, te hash svake transakcije. Nonce je broj koji se povećava jedan po jedan i tako mijenja izlaz hash funkcije. Nonce se povećava sve dok hash ne zadovolji zadani kriteriji.

Blockchain tehnologija koristi hash funkciju po svuda. Ako se blok mijenja, tj. netko je pokušao promijeniti zapisane podatke u svoju korist, hash vrijednost bi bila kompletno drugačija i svi mogu primijetiti da se nešto promijenilo. Hash vrijednost prijašnjeg bloka se koristi za računanje hash-a trenutnog bloka, te svaki blok ima hash prijašnjeg bloka te time stvaraju vezu između blokova koja se prekida čim netko izmjeni podatke.

Rudarenje

Proces rudarenja podrazumijeva traženje broja nonce pomoću kojeg će rudar, zajedno s ostalim podacima iz bloka, izračunati hash koji zadovoljava određene kriterije. Kriteriji se mijenjaju s obzirom na vrijeme koje je bilo potrebno da se riješi block kako bi se postigao jednak razmak između dodavanja novih blokova. Dobiveno rješenje zovemo Proof of Work. Rješenje potvrđuje da je rudar potrošio mnogo vremena i resursa na pronalaženje nonca. Onaj rudar kojem to uspije šalje novi blok u sustav i biva nagrađen za svoj rad. U Bitcoinu, najpoznatijoj kripto valuti, blok se dodaje svakih 10 minuta, kad se blok doda transakcije koje se nalaze u tom bloku su potvrđene i Bitcoini se mogu dalje trošiti.

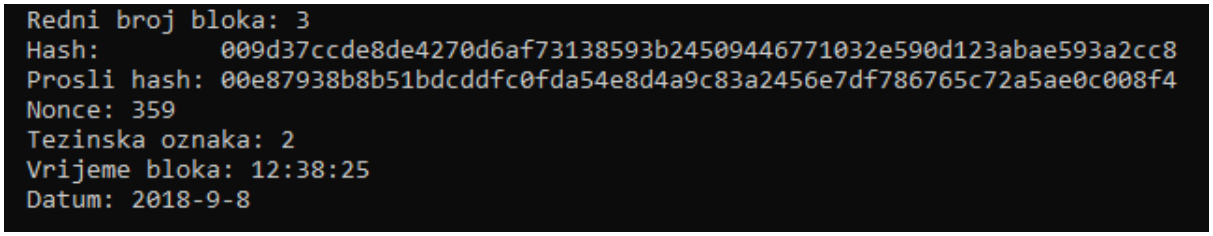
```
void Block::MineBlock()
{
    char cstr[difficulty +1];
    for (int i = 0; i < difficulty; ++i)
    {
        cstr[i] = '0';
    }
    cstr[difficulty] = '\0';
    string tmp_hash=_hash;
    string str(cstr);

    while (tmp_hash.substr(0, difficulty) != str)
    {
        nonce++;
        tmp_hash =hashing(_hash+patch::to_string(nonce)+prev_hash);
    }

    _hash=tmp_hash;
}
```

Slika 7: Primjer rudarenja

Na slici 7 vidimo pojednostavljen proces rudarenja koji je implantiran u Blockchain aplikaciji. Težinska oznaka je kriteriji koji rudari moraju zadovoljiti i to je najčešće broj nula na početku hasha. U ovom kodu težinska oznaka označena je s *difficulty*. U varijablu *cstr* spremamo određeni broj nula koji nam je zadan preko *difficulty* varijable. *Cstr* listu pretvaramo u string, da je poslije možemo uspoređivati s hashom, i spremamo je u varijablu *str*. U funkciji *while* se broj *nonce* povećava jedan po jedan sve dok početak hasha ,nonca , prošlog hasha i hasha svih podataka u bloku, nije jednak *str* varijabli. Jedino rješenje za pronalazak nonca je bruteforce algoritam (pokušavanje svih mogućih kombinacija). Hash koji zadovoljava uvjet spremamo i njega zovemo hash bloka.



Slika 8: Primjer potvrđenog bloka

Na slici 8 vidimo primjer jednog potvrđenog bloka. Vidimo da je težinska oznaka 2 i da su prva dva znaka hasha jednaka nuli.

Kada za rudanje novog bloka ne bi bilo potrebna implementacija Proof of Work algoritma tada bi najjače računalo u mreži uvijek prvi riješio blok što bi značilo da sustav više nije decentraliziran i da jedno računalo kontrolira cijeli sustav. Dakle Proof of Work možemo opisati sljedećim koracima:

1. Sakupljene transakcije spremi u novi blok
2. Odaberi nonce
3. Izračunaj hash bloka i nonca
4. Usporedi da li izračunati hash zadovoljava kriteriji
5. Ako zadovoljava potvrdi blok, ako ne vrati se drugi korak.

Problem bizantijskih generala

Problem bizantskog generala (Leslie Lamport, Robert Shostak, Marshall Pease, The Byzantine Generals Problem, 1982) je sporazumni problem koji su najprije opisali Leslie Lamport, Robert Shostak i Marshall Pease u svom radu iz 1982. godine "Problem bizantskog generala". Problem je misaoni eksperiment koji ima za cilj ilustrirati poteškoće postizanja konsenzusa u distribuiranom sustavu.

Problem: Skupina generala, svaki zapovijedajući s dijelom bizantske vojske, okružuje grad. Generali moraju razviti plan da napadnu ili da se povuku iz grada. Neki generali možda žele napasti, dok se drugi možda žele povući, ono što je potrebno je da svaki general dođe do kolektivne odluke. Ako se ne postigne jedinstvena odluka, neki se generali odluče za napad, dok se drugi odluče povući, tada neusklađeni napad ili povlačenje neće uspjeti i većina vojske će poginuti. Generali su također fizički odvojeni jedni od drugih i moraju prenijeti svoju odluku putem glasnika. Postoji mogućnost da glasnici ne uspiju isporučiti glasove, ili mogu krivotvoriti glasove, što bi rezultiralo neusklađenim odlukama. Problem je dodatno kompliciran od strane izdajnički generala koji mogu poslati mješovite poruke o njihovim željama, navodeći neke generale na napad, a druge na povlačenje.

Zamislimo da je svaki general blok u blockchainu i oni se moraju dogovoriti koji blokovi su ispravni, a koji su izmijenjeni - izdajice. Kako je blockchain decentralizirani sustav, nema središnjeg autoriteta da ukloni pogreške u slučaju neuspjeha. Proof of Work algoritam je prvi put pronađen u Bitcoinu te pruža rješenje problema Bizantskog generala. Svaki general kada prvi puta primi poruku o napadu ili povlačenju pokrene na svom računalu izračunavanje hash-a koji sadrži informaciju o tom podatku. Kada neki general nađe hash koji zadovoljava dane kriterije pošalje ga ostalima u mreži, kako bi ga oni upisali u blockchain. Ostali generali tada počinju izračunavati novi hash koji u sebi sadrži prethodno generirani hash. Na taj način generali žele izgraditi što dulji lanac s istom odlukom i poštuju pravilo da se u jednoj grani lanca nalaze samo hash-evi koji imaju u sebi zapisano identične odluke. Ako neki general izdajnik želi upisati neku drugu odluku on mora potaknuti račvanje u blockchainu. No, budući da ima više dobrih generala prema pravilu najduljeg lanca grana izdajnika će nakon nekog vremena biti ignorirana.

Nakon nekoliko sati postoji dovoljno dugi lanac s istom odlukom napada. Svi generali sada imaju dokaz da se na izgradnju tog lanca potrošila velika količina vremena i može vjerovati da je odluka zapisana u tom lancu dogovorena od strane svih generala.

Struktura blockchaina

Nakon sto smo objasnili osnovni princip kako blockchain radi, možemo krenuti na jednostavnu strukturu Blockchaina, blokova i na objašnjene koda.

Blockchain

U najosnovnijem slučaju, Blockchaina je veoma jednostavni opis potvrđenih blokova gdje je svaki blok povezan sa svojim predvodnikom sve do prvoga bloka(engl. genesis block) u lancu koji se koristi za inicijaliziranje podataka.

Blockchain je često vizualiziran kao vertikalni stog. Blokovi su slojevito poredani jedan na drugi dok prvi blok služi kao temelj tog stoga. Prilikom ovakve vizualizacije blokova poredanih jedan na drugog pojavljuju se pojmovi kao što je “visina” koja se odnosi na udaljenost od prvoga bloka i pojam “vrh” koji se odnosi na zadnji dodani blok na stog. U blockchain mreži pohranjen je veliki broj zapisa. Održavanje ovih zapisa pomaže korisnicima pratiti sve informacije od početka zadanog blockchaina.

Blockchain se sastoji od tri glavna dijela:

Blok- lista sastavljana od raznih podataka

Lanac – Hash koji povezuje jedan blok s drugim

Mreža – mreža se sastoji od čvorova. Svaki čvor sadrži cjeloviti zapis o svim podacima koje su ikad zabilježene u tom blockchainu. Čvorovi se mogu nalaziti diljem svijeta i mogu biti upravljane od bilo koga. Na žalost, u ovom primjeru mreža nije implementirana u blockchain.

Objašnjene koda

```
Blockchain::Blockchain()
{
    chain.emplace_back(Block(0));
}

void Blockchain::addBlock(Block newBlock)
{
    newBlock.prev_hash=chain.back().getHash();
    chain.push_back(newBlock);
}
```

Slika 9: Funkcije korištene u Blockchain klasi

Na prvoj konstruktorskoj funkciji *Blockchain*(), Slika 9, kreira se vektor koji sam nazvao *chain* i dodavamo mu prvi blok u lanac, genesis blok.

Kad se blok napuni transakcijama poziva se funkcija *addBlock(Block newBlock)* koja povezuje novi blok s prijašnjim preko hash-a, te novi blok stavlja na vrh lanca.

```
void Blockchain::print()
{
    for(int j=0; j<chain.size(); j++)
    {
        if(chain[j].n_transaction!=0)
        {
            tm *ptm=localtime(&(chain[j]._time));
            cout<<endl<<"\t\t\tBLOCK " << j<<":"<<endl;
            cout <<"TRANSAKCIJE:"<<endl;
            for(int i=0; i<chain[j].getTransaction().size();i++)
                {
                cout <<"\tFrom "<<chain[j].getTransaction()[i].getFrom();
                cout <<"\tto "<<chain[j].getTransaction()[i].getTo();
                cout <<"\tamount: "<<chain[j].getTransaction()[i].getAmount()<<endl;
                }
            cout <<"BLOCK:"<<endl;
            cout <<"\tRedni broj bloka: " << chain[j].getIndex() << endl;
            cout <<"\tHash:         " << chain[j].getHash()<<endl;
            cout <<"\tProsli hash: " << chain[j].prev_hash <<endl;
            cout <<"\tNonce: " << chain[j].getNonce() <<endl;
            cout <<"\tTezinska oznaka: " << chain[j].difficulty <<endl;
            cout <<"\tVrijeme bloka: " << ptm->tm_hour << ":" << ptm->tm_min<< ":" << ptm->tm_sec << endl;
            cout <<"\tDatum: " << (ptm->tm_year+1900) << "-" << (ptm->tm_mon+1) << "-" << ptm->tm_mday << endl;
        }
    }
}
```

Slika 10: Funkcija za ispis svih podataka u blockchainu

Na slici 10 je prikazan dio koda koji služi za ispisivanje svih podataka koji su zapisani u blockchain.

```
                        BLOCK 1:
TRANSAKCIJE:
        From robi       to gabriel      amount: 40
        From gabriel    to nikola       amount: 50
BLOCK:
        Redni broj bloka: 1
        Hash:         00300bc8d9db2f09cd2fac750f2dd872a093a39fcacde599adea55691d339694
        Prosli hash: 006b876df90e21053bad22de3a8945418788f5db2644be9fc39cf846001365a2
        Nonce: 247
        Tezinska oznaka: 2
        Vrijeme bloka: 12:20:36
        Datum: 2018-9-10
```

Slika 11: Primjer ispisa

Slika 11 prikazuje primjer izlaza *Print()* funkcije.

Blok

Blokovi su datoteke u kojima su trajno pohranjeni podatci izvršeni u određenom vremenskom periodu. Svaki blok sastoji se od zapisa o nekim podacima i referencama na blok koji je prethodio. Blokovi su organizirani u linearni niz koji nazivamo blockchain. Jednom zapisani podaci u blok trajno su tamo pohranjeni, ne mogu se mijenjati ili unositi.

Svaki blok unutar blockchaina jednoznačno je definiran hash funkcijom. Svaki blok sadrži svoj "identitet" koji je generiran korištenjem SHA256 kriptografskog algoritma za hashiranje. Taj je "identitet" pohranjen u zaglavlju bloka. Svaki blok referencira na prethodni blok u lancu tako što sadrži hash identitet prethodnog. Prethodni blok nazivamo roditelj, *engl. parent block*. Ovim referenciranjem blokova sa njihovim prethodnicima stvaramo lanac koji povezuje sve blokove u lanac do početnog, prvog bloka kojeg nazivamo *engl. Genesis block.* Blok može imati samo jednoga roditelja (*parent block)* i više "djece" blokova. Svaki blok može imati samo jednoga roditelja zbog toga što u svakom bloku može biti pohranjena samo jedna vrijednost u polju "prethodni block hash". Situacija većeg broja djece u blockchainu se javlja u procesu račvanja kada se pojavi više blokova koji sadrže istu hash referencu od prethodnog bloka. To je privremena situacija u kojoj su različiti blokovi otkriveni praktički istovremeno od strane različitih rudara (*engl. miners).*

Blok predstavlja trenutno stanje, sadrži informacije o podacima u prošlosti i sadrži poveznicu na budući novi blok. Svaki put kada je blok završen on daje prostor novom bloku u lancu. Dovršen blok stalni je zapis podataka u prošlosti i nove podatke bilježe se u tekućem bloku. Ovim principom cijeli sustav radi u ciklusu, a podaci se stalno pohranjuju.

Struktura bloka

Tablica 1: Struktura bloka

Naziv	Opis	Veličina (bajtovi)
Veličina bloka (Blocksize)	Veličina bloka u bajtovima	4
Zaglavlje bloka (Blockheader)	Sastoji se od 6 stavki koje su navedene u tablici niže	80
Brojač podataka	Koliko zapisa sadrži blok	1-9
Podaci	Zapisi pohranjeni u bloku	Ovisi o količini podataka

U zaglavlju bloka sadržani su ovi podatci

Tablica 2: Zaglavlje bloka

Naziv	Svrha	Ažurira se...	Veličina (bajtovi)
Verzija	Verzija boka (specifično za blockchain)	Prilikom nadogradnje softvera i on specificira novu verziju	4
Hash Prošlog bloka	256 bitnia hash adresa od prethodnog bloka	Kada dolazi novi blok	32
Hash sadržaja	256 bitni hash svih zapisa iz bloka	Kada je prihvaćena transakcija	32
Vrijeme	Trenutna vremenska oznaka u sekundama od 1.1.1970. 00:00 UTC	Svake sekunde	4
Težinska oznaka	Težina algoritma čije je rješenje potrebno za uključivanje bloka u blockchain	Kada je prihvaćen novi blok	4
Nonce	32 bitni broj, počinje na 0		4

Objašnjene koda

```
Block::Block(int nIndex)
{
    index=nIndex;
    _time=time(0);
    nonce=0;
    n_transaction=0;
    difficulty=2+(0.5*nIndex);
    _hash=hashing(patch::to_string(index)+prev_hash+patch::to_string(_time)+patch::to_string(nonce));
}

void Block::addTransaction(Transaction tmpTransaction)
{
    _time=time(0);
    _hash=hashing(_hash+patch::to_string(_time)+tmpTransaction.getHash());
    vector_transaction.push_back(tmpTransaction);
    n_transaction++;
}
```

Slika 12: Funkcije korištene u Blok klasi

U konstruktoru *Blok(int nIndex)* kreira se novi blok, Slika 12. Konstruktoru prosljeđujemo veličinu lanaca te time znamo redni broj bloka. Sprema se vrijeme kad je nastao blok, nonce i broj transakcija postavljamo na 0. Težinsku oznaku povećavamo za svaka dva bloka za jedan. U varijablu *_hash* spremamo hash generiranog od indexa bloka, hash-a prijašnjeg bloka, vremena i nonca. Kako još nemamo zapisanih transakcija u bloku hash će se mijenjati unosom svake transakcije.

Kada se obavi nova transakcija funkciji *addTransaction(Transaction tmpTransaction)* prosljeđujemo novu transakciju, te je zapisujemo u blok. Ažurira se vrijeme bloka i hash koji

se sad generira od prijašnjeg generiranog hash-a, novog vremena i o hash-a transakcije. Transakcije spremamo u vektor varijablu *vector_transaction* i povećavamo broj transakcija u bloku.

```
void Block::MineBlock()
{
    char cstr[difficulty +1];
    for (int i = 0; i < difficulty; ++i)
    {
        cstr[i] = '0';
    }
    cstr[difficulty] = '\0';
    string tmp_hash=_hash;
    string str(cstr);

    while (tmp_hash.substr(0, difficulty) != str)
    {
        nonce++;
        tmp_hash =hashing(_hash+patch::to_string(nonce)+prev_hash);
    }

    _hash=tmp_hash;
}
```

Slika 13: Funkcija za rudanje bloka

Kad se blok napuni transakcijama moramo potvrditi trenutni blok da možemo dodati novi blok u lanac, to radi funkcija *MineBlock()*, Slika 13. Inicijaliziramo novu *char* listu veličine *difficulty*, te je „napunimo“ nulama. U zadnjem polje liste dodajemo „end of line character“ da funkcije uoče kad je kraj stringa. Hash bloka spremamo u novu varijablu *tmp_hash*, te listu pretvaramo u string, da je možemo uspoređivati s hash-om, i spremamo je u *str* varijablu. *While* funkcija se ponavlja sve dok početak hash nije isti *str* varijabli. Svako ponavljanje *while* funkcije povećava *nonce* jedan po jedan te generira hash s novim noncem. Kad pronađemo zadovoljavajući hash spremamo ga u *_hash* varijablu i time je trenutni blok potvrđen, te se poslije dodaje novi blok u lanac.

Transakcije i novčanici

Blockchain se može koristi za spremanje mnogo raznovrsnih podataka, ali ipak najpoznatije je za pohranu transakcija između korisnika. Prvi sustav koji je popularizirao Blockchain i kripto valute je Bitcoin. Bitcoin koristi novčanike (engl. wallet) gdje se pohranjuju Bitcoin tokeni, te se preko njegove adrese šalju i primanju Bitcoini. Bitcoine prebacujemo Bitcoin transakcijama koje se spremaju u blokove. Svaki put kad se pokrene novčanik, novčanik se ažurira tako da čita sve transakcije sa svojom adresom u bitcoin blockchainu, te ih zbraja ili oduzima da stvori ukupan broj Bitcoina.

Upoznajmo najprije najvažnije pojmove za funkcioniranje transakcija i novčanika. Svaki novčanik ima svoj javni ključ, privatni ključ i adresu. Privatni ključ se može generirati na razne načine, najčešće se generira odabirom nasumičnog broja između 1 i 2^{256} na koje se primjenjuje hash algoritam SHA-256. Kad smo generirali privatni ključ, pomoću njega i eliptične krivulje generiramo javni ključ. Javni ključ je točka na krivulji dobivena operacijom množenja točke G skalarom privatnog ključa, gdje javni ključ= G * privatni ključ. Kada smo dobili javni ključ, primjenom hash algoritma SHA-256 na javni ključ dobijemo adresu novčanika. Operacije na eliptičnoj krivulji i hash funkciji su jednosmjerne, odnosno ne možemo iz adrese generirati javni ključ i iz javnog ključa privatni ključ. Pri kreiranje transakcije koristi se privatni ključ, te pomoću njega korisnik potvrđuje da je on vlasnik određene adrese.

Transakcija je zapis u Bitcoin mreži (ili bilo kojoj drugoj kripto valuti) kojim se određeni iznos bitcoina prenosi s jedne adrese na drugu adresu. Transakcije su javne i moguće ih je slobodno pregledavati. Novčanik omogućava korisniku da kreira novu transakciju, da bi se transakcija izvršila potrebno je da korisnik navede adresu primatelj i iznos koji želi uplatiti. Posjedovanjem privatnog ključa korisnik potvrđuje da određena svota novca pripada njemu.

Kako Bitcoin koristi transakcije tako će se i u jednostavnoj blockchain simulaciji koristiti novčanici i transakcije. Novčanici se kreiraju u konzoli bez ikakve kontrole sigurnosti.

Objašnjenje koda

```
Wallet::Wallet(double n_balance)
{
    private_key=hashing(patch::to_string(rand() % sha + 1));
    own_balance=n_balance;
}

void Wallet::synx_wallet(Blockchain FullCoin)
{
    balance=own_balance;
    for(int j=0; j<FullCoin.chain.size(); j++)
    {
        for(int i=0; i<FullCoin.chain[j].getTransaction().size(); i++)
        {
            if(FullCoin.chain[j].getTransaction()[i].getFromID()==private_key)
            {
                balance=balance-FullCoin.chain[j].getTransaction()[i].getAmount();
            }

            if(FullCoin.chain[j].getTransaction()[i].getToID()==private_key)
            {
                balance=balance+FullCoin.chain[j].getTransaction()[i].getAmount();
            }

        }
    }
}
```

Slika 14: Funkcije korištene u Wallet klasi

Klasa *Wallet* ima dvije funkcije, jednu konstruktorsku *Wallet(double n_balance)* kojoj prosljeđujemo svotu valute te je sprema u *own_balance*, Slika 14. Privatni ključ se generira tako da nasumično odabrani broj između 1 i 2^{256}, hashiramo pomoću SHA-256 algoritma. U funkciji *synx_wallet(Blockchain FullCoin)* čita se sve zapisane transakcije i uspoređuju se ako je korišteni trenutni novčanik, te ako je svota novaca se povećava ili smanjuje ovisno o vrsti transakcije.

```
Transaction::Transaction(string nFrom, string nTo, string nFromID, string nToID, int nAmount)
{
    fromID=nFromID;
    toID=nToID;
    from=nFrom;
    to=nTo;
    amount=nAmount;
    transaction_hash=hashing(from+to+char(amount));
}

Transaction::Transaction()
{
    from="error";
    to="error";
    amount=0000;
    transaction_hash="error";
}
```

Slika 15: Funkcije korištene u Transaction klasi

Klasa *Transaction* ima jedan konstruktor bez parametra i jedan s pet parametra koje joj prosljeđujemo, Slika 15. Kad proslijedimo parametre funkcija ih sprema, te je transakcija spremna da se zapiše u blok.

Zaključak

U suvremenom, informacijskom društvu informacije imaju veliku važnost. Svake godine količina informacije koje razmjenjujemo se povećava, s time se povećava i potreba za pohranu tih informacija. Blockchain je idealno rješenje za pohranu informacija. Nudi sigurnost, informacije se ne mogu modificirani nakon unosa, pravo za korištenje informacije imaju samo ovlašteni korisnici, te je pregled informacija jednostavan i brzi.

U ovom radu opisao sam osnovni princip kako blockchain tehnologija funkcionira, te njezina dobra svojstva. Blockchain je relativno nova tehnologija i svakim danom sve više napreduje. Smatram da ljudi neće imate veliko povjerenje u blockchain kao u banku, samim time što banka postoji već duži niz godina i time sto plaćaju poslužitelja koji im garantira sigurnost. Tako mislim da blockchain neće zamijeniti banku, ali da je i da će još promijeniti način slanja transakcija i informacija se ne može usporiti.

Inicijalna ideja bila je napraviti funkcionalnu distribuiranu mrežu. Pokazalo se da je to preambiciozno za sada, ali planiram nastaviti proučavati blockchain tehnologiju i napraviti funkcionalan blockchain sustav.

Popis slika

Popis tablica

Popis priloga

Uz ovaj rad priložen je CD na kojem se nalazi ovaj, te izvorni kod jednostavne blockchain simulacije u c++.

Literatura

1. antonylewis2015, A gentle introduction to blockchain technology, (2015), https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/

2. Dave Nash, Build a blockchain with C++, (2017), https://davenash.com/2017/10/build-a-blockchain-with-c/

3. Domina Hozjan, Blockchain, PMF, Zagreb, 2017

4. hak8or, SHA-256 basic implementation in C++ with a test

5. Hamzeh Khazaei, Marios Fokaefs, Saeed Zareian, Nasim Beigi-Mohammadi, Brian Ramprasad, Mark Shtern, Purwa Gaikwad and Marin Litoiu, How do i choose the right NoSQL solution? A comprehensive theoretical and experimental survey, 2016

6. Henry He, Blockchain Basics – Fork, (2018), https://www.c-sharpcorner.com/article/blockchain-basic-fork/

7. https://en.bitcoin.it/wiki/Block

8. https://en.bitcoin.it/wiki/Block_hashing_algorithm

9. Imran Bashir, Mastering Blockchain, 2017

10. Leslie Lamport, Robert Shostak, Marshall Pease, The Byzantine Generals Problem, 1982

11. Melanie Swan, Blockchain: Blueprint for a New Economy, 2015

12. Sasha Ivanović, Jacob Saur, Lisa Thomson, What is Blockchain Technology? A Step-by-Step Guide For Beginners, (2018), https://blockgeeks.com/guides/what-is-blockchain-technology/

13. Stewart Martins, Understanding Blockchain: A Simple Approach, (2018), https://blockchainexplained.org/understanding-blockchain/

14. Usman M. Sheikh, Virtual currency: current regulatory and civil litigation trends, 2018