UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Magistrsko delo

Algoritem za pošiljanje sporočil v decentraliziranih omrežjih, ki temelji na psevdo naključnosti

(A message passing algorithm based on pseudo randomness)

Ime in priimek: *Domen Vake*
Študijski program: *Računalništvo in informatika, 2. stopnja*
Mentor: *izr. prof. dr. Jernej Vičič*
Somentor: *asist. Aleksandar Tošić*

Koper, september 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Domen Vake
Naslov magistrskega dela: Algoritem za pošiljanje sporočil v decentraliziranih omrežjih, ki temelji na psevdo naključnosti

Kraj: Koper
Leto: 2021

Število listov: 73 Število slik: 39 Število tabel: 4
Število referenc: 26

Mentor: izr. prof. dr. Jernej Vičič
Somentor: asist. Aleksandar Tošić

UDK: 004.421(043.2)
Ključne besede: pošiljanje sporočil, porazdeljeni sistemi, algoritem, psevdo naključnost
Izvleček:
V magistrskem delu je predstavljen inovativen algoritem za diseminacijo sporočil v popolnoma decentraliziranih sistemih. Motivacija za raziskavo prihaja iz področja tehnologije veriženja blokov in razvoja decentraliziranih omrežnih protokolov. Ena izmed ključnih omejitev tovrstnih sistemov je učinkovita diseminacija sporočil, ki imajo s strani sistemskega pogleda lahko naključen izvor. Cilj dobrega algoritma je, da v čim krajšem času vsa vozlišča prejmejo sporočilo. Najbolj razširjena in enostavna implementacija je t.i "flooding algoritem", ki zagotavlja, da bodo vsa vozlišča, ki so povezana v omrežje prejela sporočilo. Algoritmi, ki temeljijo na principu poplavljanja so neučinkoviti. Učinkovitost tovrstnih algoritmov običajno ocenjujemo z mero, ki opisuje koliko krat vozlišče prejme enako sporočilo. Optimalen algoritem bi zagotavljal, da bodo ne glede na izvor sporočila vsa vozlišča zanj izvedela in ga bo vsako vozlišče prejelo natanko enkrat. Predstavljen algoritem deluje na principu psevdo naključnosti za določanje poti sporočila.

Key document information

Name and SURNAME: Domen Vake
Title of the thesis: A message passing algorithm based on pseudo randomness

Place: Koper
Year: 2021

Number of pages: 73 Number of figures: 39 Number of tables: 4
Number of references: 26

Mentor: Assoc. Prof. Jernej Vičič PhD
co-Mentor: Assist. Aleksandar Tošić

UDC: 004.421(043.2)
Keywords: message passing, distributed systems, algorithm, pseudo random
Abstract:
The master's thesis presents an innovative algorithm for the dissemination of messages in fully decentralized systems. The motivation for research comes from the field of blockchain technology and the development of decentralized network protocols. One of the key limitations of such systems is effective spread of data thorugh the system. The goal of good algorithms is to get every message received as soon as possible. The common and easy implementation is the so-called "flooding algorithm", which ensures that every transport option connected to the network will receive a message. Algorithms based on the principle of flooding are inefficient. The effectiveness of such algorithms is usually assessed by a measure that describes how many times the node receives the same message. The presented algorithm works on the principle of pseudo randomnes for determining the message path.

Kazalo vsebine

Kazalo preglednic

Kazalo slik in grafikonov

Seznam kratic

tj.	to je
npr.	na primer
SPOF	Kritična točka okvare

Zahvala

Zahvalil bi se mentorju doc. dr. Jerneju Vičiču in somentorju mag. Aleksandru Tošiću za vso podporo, strokovno pomoč in usmeritve tako pri zaključnem delu, kot v času izobraževanja. Prav tako bi se zahvalil družini in prijateljem za vso podporo, ki so mi jo izkazali na izobraževalni poti.
Hvala!

1 Uvod

Na področju računalniških sistemov od nekdaj spremljamo izredno hiter razvoj. Od leta 1945, ko se je začela sodobna računalniška doba, do približno 1985 so bili računalniki veliki in cenovno težje dostopni. Prav tako so ti računalniki zaradi nepoznavanja različnih načinov povezovanja delovali neodvisno drug od drugega.

Rezultat teh tehnologij je, da danes ni le izvedljivo, ampak je prav enostavno sestaviti računalniški sistem, sestavljen iz številnih omrežnih računalnikov. Velikost sistema se lahko razlikuje od peščice naprav do milijonov računalnikov. Povezava naprav je lahko žična, brezžično ali v kombinaciji obeh. Poleg tega se topologija sistemov pogosto dinamično spreminja z dodajanjem in odstranjevanjem novih naprav.

Vse te naprave morajo med seboj komunicirati in ta proces je lahko zelo zahteven. V zaključni nalogi je predstavljen inovativen algoritem za pošiljanje sporočil v porazdeljenih omrežjih, ki temelji na psevdo naključnosti.

2 Predstavitev področja

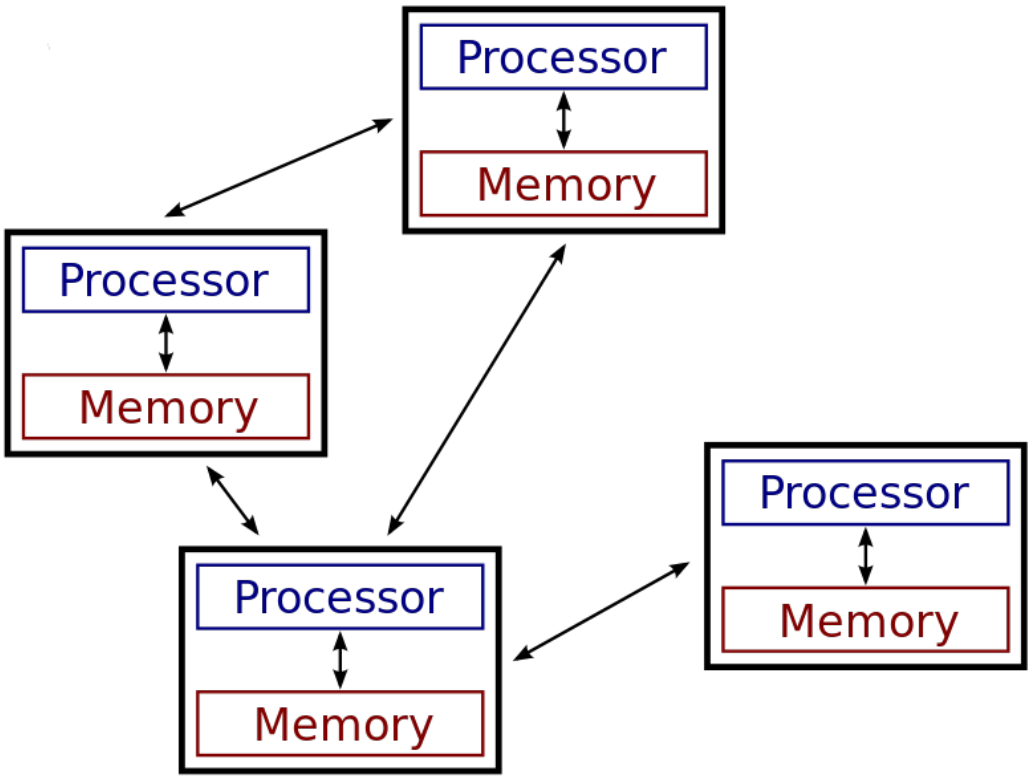
V magistrskem delu je predstavljen inovativen algoritem za diseminacijo sporočil v popolnoma decentraliziranih sistemih. Motivacija za raziskavo prihaja s področja tehnologije veriženja blokov in razvoja decentraliziranih omrežnih protokolov. V ta namen so v tem poglavju predstavljeni glavni koncepti, potrebni za razumevanje jedra naloge. Predstavljene so lastnosti porazdeljenih in decentraliziranih sistemov, glavni problemi le-teh in kako se z njimi spopadamo. Poglavje zajema predstavitev tehnologije veriženja blokov in njene glavne sestavne dele. Predstavljeni so tudi *Broadcast algoritem*, *Flooding algoritem* in *Push algoritem*, ki so prepoznani kot glavni za opravljanje enake naloge, kot jo opravlja naš algoritem, to je diseminacija sporočil v decentraliziranih sistemih.

2.1 Porazdeljeni sistemi

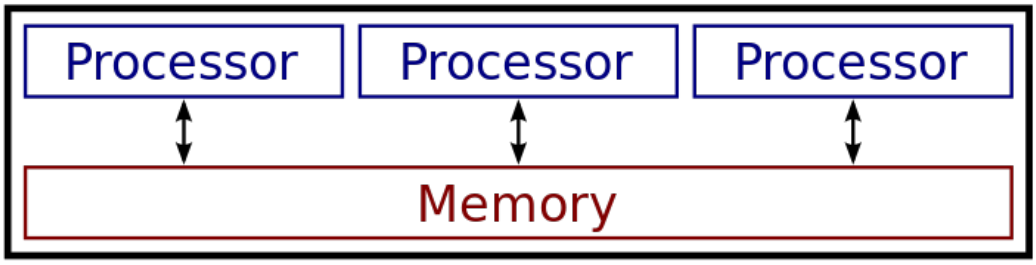
Porazdeljeni sistemi so pomembni za razvoj računalništva, saj je vse več problemov tako obsežnih in zapletenih, da jih en sam računalnik ne bi zmogel obravnavati sam. Tako porazdeljeno računalništvo ponuja tudi dodatne prednosti pred tradicionalnimi računalniškimi okolji. Porazdeljeni sistemi zmanjšujejo tveganja, povezana z eno samo kritično točko okvare (SPOF), saj povečujejo zanesljivost in toleranco napak. Sodobni porazdeljeni sistemi so na splošno zasnovani tako, da jih je mogoče prilagoditi in spreminjati v realnem času. Prav tako lahko vzporedno dodamo dodatne računalniške vire, s čimer povečamo zmogljivost in dodatno skrajšamo čas dela [1].

Ena od možnih definicij porazdeljenih sistemov je: *sistemi, v katerih komponente, ki se nahajajo v računalnikih v omrežju, komunicirajo in usklajujejo svoja dejanja s pomočjo pošiljanja sporočil. Komponente medsebojno delujejo, da bi dosegle skupni cilj* [2].

Koncepta *vzporedno računalništvo* in *porazdeljeno računalništvo* sta si na prvi pogled podobna in med njima razlika ni hitro jasna. V glavnem ju ločimo glede na način izmenjave podatkov. Pri vzporednem računalništvu imajo lahko vsi procesorji dostop do skupnega pomnilnika za izmenjavo informacij med procesorji, medtem ko ima pri porazdeljenem računalništvu vsak procesor svoj zasebni pomnilnik (porazdeljeni pomnilnik). Informacije se izmenjujejo s prenosom sporočil med procesorji. [3]



Slika 1: [1]Slika prikazuje shemo pomnilnika v porazdeljenem sistemu. Vsak črn kvadrat predstavlja računalnik, ki ima svojo procesorsko enoto (processor - moder kvadrat) in svoj pomnilnik (memory - rdeč kvadrat). Puščice v tem sistemu predstavljajo komunikacijski kanal med enotami.



Slika 2: [2]Slika prikazuje shemo pomnilnika v vzporednem sistemu. Črn kvadrat predstavlja računalnik, ki ima več procesorskih enot (processor - moder kvadrat) in skupni pomnilnik (memory - rdeč kvadrat). Puščice predstavljajo komunikacijski kanal med enotami. Procesorji komunicirajo preko skupnega pomnilnika

[1]Vir slike: `https://en.wikipedia.org/wiki/Distributed_computing#/media/File:Distributed-parallel.svg`

[2]Vir slike: `https://en.wikipedia.org/wiki/Distributed_computing#/media/File:Distributed-parallel.svg`

2.1.1 Arhitekture porazdeljenih sistemov

Za porazdeljeno računalništvo se uporabljajo različne strojne in programske arhitekture. Na nižji ravni je treba med seboj povezati več procesorjev z nekakšnim omrežjem, ne glede na to, ali je to omrežje natisnjeno na vezju ali je sestavljeno iz ohlapno povezanih naprav in kablov. Na višji ravni je potrebno procese, ki se izvajajo, povezati z nekakšnim komunikacijskim sistemom.

Porazdeljeni sistemi običajno spadajo v eno od več osnovnih arhitektur: odjemalec-strežnik, tristopenjsko, n-stopenjsko ali peer-to-peer. [6]

- **Odjemalec-strežnik:** arhitektura, kjer se pametni odjemalci obrnejo na strežnik za podatke, nato jih formatirajo in prikažejo uporabnikom. Vnos pri odjemalcu se vrne strežniku, ko predstavlja trajno spremembo.

- **Tristopenjski:** arhitektura, kjer med logiko odjemalca in strežnika postavimo vmesni sloj, ki ne hrani stanja (stateless). To poenostavi uvajanje aplikacij. Večina spletnih aplikacij je tristopenjskih.

- **n-stopenjske:** arhitektura, ki se običajno nanaša na spletne aplikacije, ki posredujejo zahteve drugim storitvam podjetja ali drugim aplikacijam.

- **Peer-to-peer:** arhitektura, kjer ni posebnih naprave, ki ponujajo storitve ali upravljajo z omrežnimi viri. Namesto tega so vse odgovornosti enakomerno porazdeljene med vse naprave v omrežju. Naprave imajo lahko tako enako vlogo kot odjemalci, in tudi kot strežniki. Primeri te arhitekture vključujejo BitTorrent in Bitcoin.

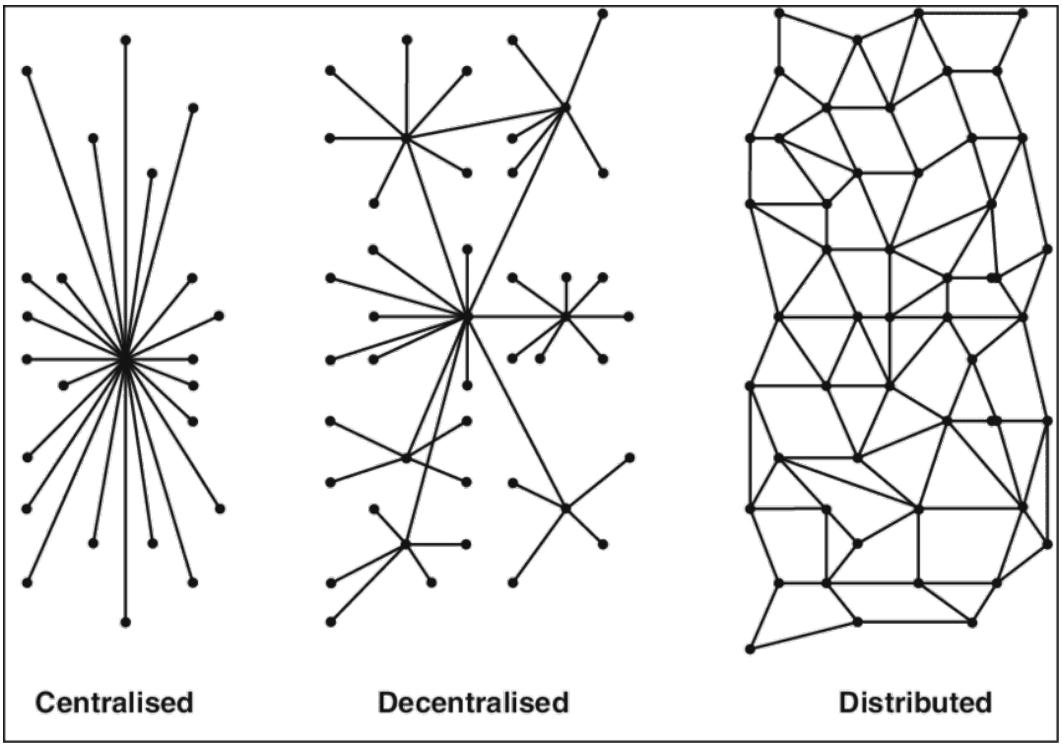
Razlogov za uporabo porazdeljenih sistemov je lahko več. Sama narava aplikacije lahko zahteva uporabo komunikacijskega omrežja, ki povezuje več računalnikov, na primer podatke, pridobljene na enem fizičnem mestu in zahtevane na drugem fizičnem mestu. Prav tako obstaja veliko primerov, v katerih bi bila uporaba enega računalnika načeloma mogoča, vendar je uporaba porazdeljenega sistema koristna iz praktičnih razlogov. Na primer, morda je stroškovno učinkoviteje doseči želeno raven zmogljivosti z uporabo skupine več nizkocenovnih računalnikov v primerjavi z enim visoko zmogljivim računalnikom. Porazdeljeni sistemi lahko zagotovijo tudi večjo zanesljivost kot neporazdeljeni sistemu, saj nima ene same centralne točke okvare. Poleg tega je porazdeljene sisteme lažje razširiti in upravljati kot monolitne, enoprocesorske sisteme.

2.2 Decentralizirani sistemi

Glavna razlika med centraliziranimi in decentraliziranimi omrežji je povezana z vprašanjem, kdo ima nadzor nad samim omrežjem. V centraliziranem sistemu ima edinstven sis-

tem ali skrbnik popoln nadzor nad vsemi vidiki omrežja. To običajno izvaja centralni strežnik, ki upravlja vse podatke in dovoljenja. Centralizirano omrežje prav tako hrani vso glavno procesorsko moč v tem primarnem strežniku.

Decentralizirana omrežja so organizirana na veliko bolj porazdeljen način. Vsako vozlišče v omrežju deluje kot ločen organ z neodvisnimi močmi odločanja o tem, kako deluje z drugimi sistemi. Ta omrežja prav tako distribuirajo procesorsko moč in funkcije delovne obremenitve med povezanimi napravami.



Slika 3: [3]Slika prikazuje topologije značilne za centralizirane (leva), decentralizirane (srednja) in porazdeljene sisteme (desna)

Leta 1979 je David Chaum zasnoval prvi koncept decentraliziranega računalniškega sistema, znan kot *Mix Network*. Zagotovil je anonimno e-poštno komunikacijsko omrežje, ki je decentraliziralo validacijo sporočil v protokolu. Ta tehnologija je kasneje postala predhodnik tako imenovanega *Onion Routing*, protokola brskalnika *TOR*. David Chaum je s tem začetnim razvojem anonimnega komunikacijskega omrežja uporabil svojo filozofijo Mix Network za oblikovanje prvega decentraliziranega plačilnega sistema na svetu in ga leta 1980 patentiral. Kasneje leta 1982 je za svojo doktorsko disertacijo pisal o potrebi po decentraliziranih računalniških storitvah v prispevku *Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups* [7]. Leta 1990 pa je uvedel prvi digitalni plačilni sistem na svetu, imenovan eCash.

[3]Vir slike: `https://www.researchgate.net/figure/Centralised-decentralised-and-distributed-networks-Ba`
`fig4_303659963`

2.2.1 Sinhronizacija decentraliziranih sistemov

Sestavni deli porazdeljenih sistemov se med seboj povezujejo, da bi dosegli skupen cilj. Tri pomembne značilnosti porazdeljenih sistemov so: sočasnost komponent, pomanjkanje globalne ure in neodvisna okvara komponent. Da pa komponente dosežejo sinhronost brez globalne ure, ima vsako vozlišče v omrežju svojo lokalno uro, po kateri se ravna. Za delovanje sinhronega sistema je ključno, da imajo komponente te ure usklajene. Različni sistemi to rešujejo na različne načine. Pogosto, ko se vozlišče pridruži sistemu, preveri ure drugih komponent v sistemu in svojo uro prilagodi na uro sistema. Za uro sistema je pogosta mediana ali srednja vrednost lokalnih ur ostalih komponent v sistemu.

V članku [4] avtorji predstavljajo razlike med sinhronimi in asinhronimi porazdeljenimi sistemi. Avtorji so kot sinhron proces definirali proces, katerega minimalna unikatna particija sistema zadostuje pogojema:

- nobena dva koraka v istem krogu ne vključujeta istega procesa
- nobena dva koraka v istem krogu nimata dostopa do iste spremenljivke

Ta pogoja zagotavljata, da lahko proces porazdelimo in ga rešujemo vzporedno. Takšne probleme pa lahko rešujejo tudi asinhroni sistemi, katerih glavna značilnost je neusklajenost ur. Vsaka komponenta omrežja ima svojo uro, ki je neodvisna od drugih komponent v omrežju. To lahko predstavlja probleme, saj so komponente v sistemu v določenem času lahko v različnih stanjih in ne rešujejo problema usklajeno. Asinhroni sistemi se pogosto pojavljajo zaradi pomanjkljive ali omejene procesorske moči, komunikacije ali energije. Takšne omejitve so posebno značilne za senzorska omrežja [5].

Ker je naš algoritem razvit za delovanje v sinhronih decentraliziranih omrežjih, se bomo v nadaljnjih poglavjih naloge z izrazom decentralizirana omrežja nanašali na sinhrona decentralizirana omrežja.

2.3 Pošiljanje sporočil v decentraliziranih sistemih

Velika večina aplikacij danes temelji na podlagi centraliziranih strežnikov za posredovanje sporočil med odjemalci, kjer se ti strežniki predstavljajo zaupanja vredne tretje osebe. Z naraščanjem tehnologije veriženja blokov v zadnjih nekaj letih je prišlo do odmika od centraliziranih strežnikov in tradicionalnih modelov v smer bolj decentralizirane enakovredne alternative. Vendar pa se zdi, da gre za dilemo med varnostjo, razširljivostjo in decentralizacijo. Po eni strani imajo določeni decentralizirani sistemi lahko težave s razširljivostjo, po drugi strani pa v centraliziranih sistemih vedno bolj pogosto prihaja do napadov, kar kaže na vprašljivost varnosti sistemov. [8]

Ker porazdeljeni sistemi v splošnem nimajo centralne avtoritete, je edini način za širjenje podatkov zanašanje na vsako vozlišče, da govorico posredujejo svojim sosedom. Slabost govoric je v tem, da je kakovost storitev (to je popolna in pravočasna distribucija podatkov) odvisna od zahteve, da vsak član ne diskriminira in zagotavlja hiter in zanesljiv prenos podatkov vsakemu članu svoje lokalne mreže.

V porazdeljenih sistemih poznamo mnogo algoritmov za pošiljanje sporočil po omrežju. Različni algoritmi imajo različne poudarke na varnosti, razširljivosti, zanesljivosti ter količini časa in virov, ki jih potrebujejo za dosego svojega cilja. Cilj dobrega algoritma je, da v čim krajšem času vsa vozlišča (naprave) prejmejo sporočilo. Najbolj razširjena in enostavna implementacija je t.i. "flooding algoritem", ki zagotavlja, da bodo vsa vozlišča, ki so povezana v omrežje, prejela sporočilo. Algoritmi, ki temeljijo na principu poplavljanja, so neučinkoviti. Učinkovitost tovrstnih algoritmov običajno ocenjujemo z mero, ki opisuje kolikokrat vozlišče prejme enako sporočilo. Optimalen algoritem bi zagotavljal, da bodo ne glede na izvor sporočila vsa vozlišča zanj izvedela in ga bo vsako vozlišče prejelo natanko enkrat. Decentralizirani sistemi so podvrženi nepredvidljivim napakam. Vozlišča lahko naključno zapustijo in se ponovno pridružijo omrežju. V primeru nepredvidene napake enega od vozlišč, povezana vozlišča potencialno ne bodo prejela sporočila.

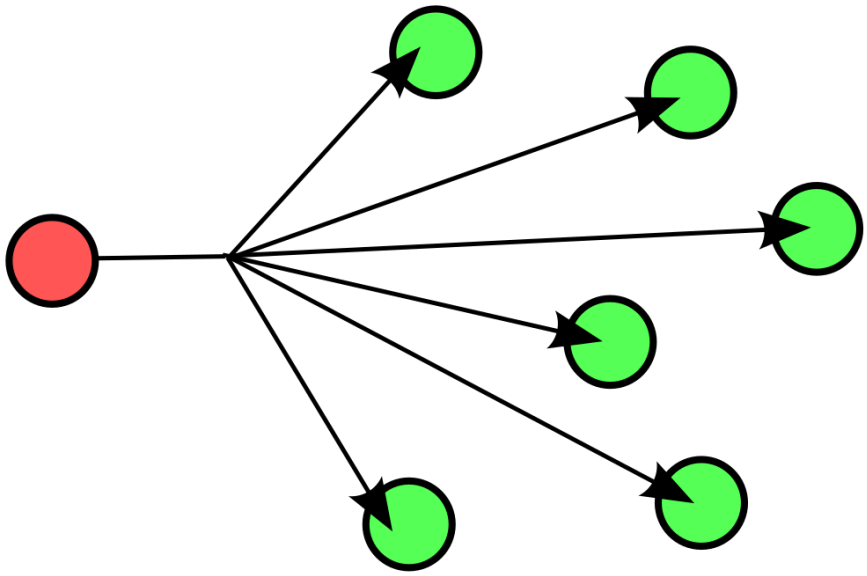
V naslednjih poglavjih so predstavljeni pogosto uporabljeni algoritmi za pošiljanje sporočil med napravami, kateri so v kasnejših poglavjih primerjani z našo implementacijo algoritma.

2.3.1 Broadcast algoritem

Broadcast ali oddajanje je metoda, ki se uporablja v računalniških omrežjih in zagotavlja, da bo vsaka naprava v omrežju prejela (oddajan) paket. Komunikacija poteka po principu "vsi z vsemi"in je komunikacijska metoda, pri kateri vsak pošiljatelj pošilja sporočila vsem prejemnikom v skupini. To je v nasprotju z metodo "od točke do točke", pri kateri vsak pošiljatelj komunicira z enim prejemnikom [10]. Prav tako pri metodi vsaka naprava potrebuje povezavo z vsemi drugimi napravami v omrežju. Pošiljatelj ali oddajnik pošlje paket vsem napravam, ki so v omrežju. Ta metoda porabi zelo veliko količino virov, saj mora imeti pošiljatelj dostop do velike pasovne širine. Ker lahko pošiljanje ali oddajanje negativno vpliva na delovanje, oddajanja ne podpira vsaka omrežna tehnologija. Večinoma se uporablja v omrežjih LAN (lokalna omrežja in ethernetu), redko pa v večjih omrežjih, kot so WAN (širokopasovna omrežja). Pomanjkljivost oddajanja je, da se lahko uporablja za napade DoS (zavrnitev storitve). Na primer, napadalec lahko pošlje lažne zahteve za ping z uporabo naslova računalnika žrtve kot izvornega naslova. Nato bodo vsa vozlišča v tem omrežju odgovorila na to zahtevo računalnika žrtve in povzročila okvaro celotnega omrežja.

Algoritem 1: Broadcast algoritem

```
1 message ← createOrRecieveMessage()
2 for v ∈ V do
3 |   sendMessage(v, message)
4 end
```



Slika 4: [4]Slika prikazuje shemo pošiljanja sporočila po principu broadcast. Oddajnik (rdeč) hkrati pošlje sporočilo vsem napravam v omrežju (zelena).

2.3.2 Flooding algoritem

Flooding algoritem ali poplavljanje se uporablja za usmerjanje paketov v računalniških omrežjih. Pri tem se vsak dohodni paket pošlje skozi vsako odhodno povezavo pri povezanih napravah, razen tiste, preko katere je prispel. V primerjavi z metodo oddajanja, poplavljanje ne potrebuje direktne povezave med vsemi napravami, temveč potrebuje le dostopno pot med vsemi napravami. [11]

Obstaja več različic algoritmov poplavljanja, vsi pa izpolnjujejo spodnja pogoja:

- Vsako vozlišče deluje kot oddajnik in sprejemnik.
- Vsako vozlišče poskuša posredovati vsako sporočilo vsakemu od svojih sosedov, razen izvornemu vozlišču.

[4]Vir slike: `https://en.wikipedia.org/wiki/Broadcasting_(networking)#/media/File:Broadcast.svg`

Posledica takšnega delovanja je, da je vsako sporočilo po določenem času dostavljeno v vse dosegljive dele omrežja. Kljub temu pa so algoritmi v splošnem bolj zapleteni, saj je v nekaterih primerih potrebno implementirati varnostne ukrepe, da bi se izognili nepotrebnim podvojenim prejemanjem sporočil in neskončnim zankam ter tako omogočili, da sporočila sčasoma tudi potečejo iz sistema. Pogosto pa pri algoritmu poplavljanja zmanjšamo porabo virov, preko implementacije FAN-OUT koncepta. Fan-out nam pove, kolikim naključnim prejemnikom, naj pošiljatelj pošlje sporočilo. To zmanjša zanesljivost algoritma, vendar zmanjša porabo virov.

Algoritem 2: Broadcast algoritem

```
1 message ← createOrRecieveMessage()
2 I ← V
3 for 0..FAN-OUT do
4 |   v ← randomElement(I)
5 |   I ← I - v
6 |   sendMessage(v, message)
7 end
```

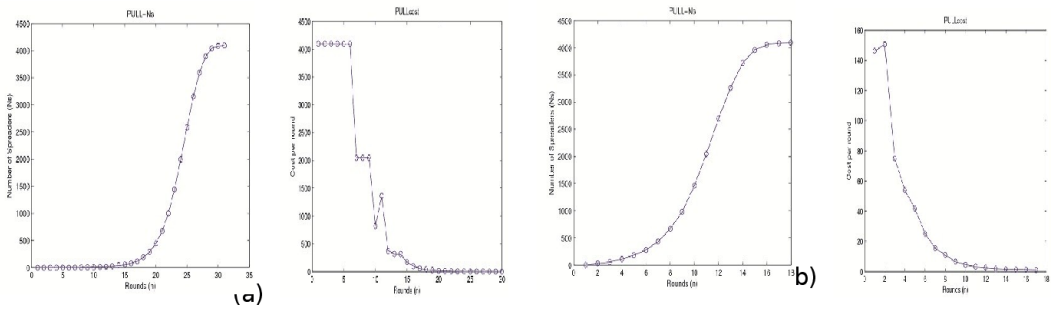
2.3.3 Push-Pull algoritem

Motivacija za uporabo naključne komunikacije je, da zagotavlja robustnost, preprostost in razširljivost [12]. Tako imenovani *push* algoritem začne v stanju, kjer je informirano le vozlišče, v katerem je govorica ustvarjena. Nato vozlišče naključno izbere neko drugo sosednje vozlišče kot partnerja in mu posreduje podatek. Tako vsako informirano vozlišče ponavlja v krogih, dokler proces ni končan po fiksno določenem številu krogov. Po $O(\ln n)$ krogih je z visoko verjetnostjo informiranih vseh n vozlišč [13]. *Push* algoritem deluje izjemno učinkovito, dokler v omrežju ni informiranih $\frac{n}{2}$, saj je visoka verjetnost, da vozlišče ob naključni izbiri soseda izbere vozlišče, ki je še neinformirano. Ko pa je v omrežju informiranih več kot polovica vozlišč, ta verjetnost začne upadati.

Za kontrast lahko pogledamo shemo *pull* algoritma, kjer vozlišča namesto propagiranja svoje govorice, prevzamejo govorico od naključnega soseda. V tem primeru bo morda vozlišče, ki začne govorico, moralo počakati nekaj krogov, preden ga prvič pokliče prvo sosednje vozlišče. Toda sčasoma, z visoko verjetnostjo po $O(\ln n)$ krogih, bo obveščenih $\frac{n}{2}$ vozlišč [12].

Od tega trenutka ima algoritem *pull* prednost pred algoritmom *push*, saj se število obveščenih vozlišč približno kvadrira iz kroga v krog. To je zato, ker ima v krogu, ki se začne z ϵn informiranimi igralci, vsak posamezni igralec verjetnost $1 - \epsilon$, da bo prejel govorico od naključno izbranega soseda. Tako obstaja verjetnost ϵ, da bo vozlišče ostalo neinformirano. Posledično je pričakovano število neobveščenih vozlišč na koncu kroga $\epsilon^2 n$ [12]. Tako lahko pričakujemo, da ta faza (od $\frac{n}{2}$ informiranih vozlišč) traja

$\theta(\ln \ln n)$ in se v tem času pošlje $\theta(n \ln \ln n)$ sporočil.



Slika 5: [5]Slika prikazuje število oddajnikov sporočil v algoritmu *pull* za omrežje velikosti N = 212 za pobudnika s stopnjo vozlišča (a) $k = 2$ in (b) $k = 82$. Prva slika (levo) prikazuje delovanje algoritma *pull* za iste parametre v istem omrežju. Število povezav začetnega oddajnika je $k = 2$ in $k = 82$. Število krogov za popolno razširjanje sporočila se močno razlikujejo glede na začetno stopnjo širjenja. Pri začetni razpršitvi z višjo stopnjo so stroški in krogi manjši v primerjavi z začetnim primerom razpršitve z nižjo stopnjo.

Da združimo predvidljivost algoritma *push* s kvadratnim informiranjem algoritma *pull*, govorico preprosto pošljemo v obe smeri, kadar je le mogoče. Algoritem *push-pull* deluje namreč tako, da ustvarjalec govoric začne časovni števec govorice z 0, kar predstavlja starost govorice. V vsakem krogu se starost inkrementalno zveča in pošlje s sporočilom oziroma z govoricami. V vsakem krogu vsak obveščeni igralec potisne in potegne ("push and pull") govorico od naključnih sosedov, razen če je starost govorice večja od $t_{max} = \log_3 n + O(\ln \ln n)$, saj faza eksponentne rasti traja $\log_3 n + O(\ln \ln n)$ krogov z visoko verjetnostjo. Med to fazo število prenosov govorice raste eksponentno iz kroga v krog. Tako v tej fazi pošljemo $O(n)$ sporočil. Vse druge faze trajajo $O(\ln \ln n)$, tako da lahko med njimi pričakujemo $2n$ prenosov govorice za vsako fazo. Tako lahko razberemo, da je celotno število prenosov enako $O(n \ln \ln n)$ [12].

2.4 Psevdo naključnost

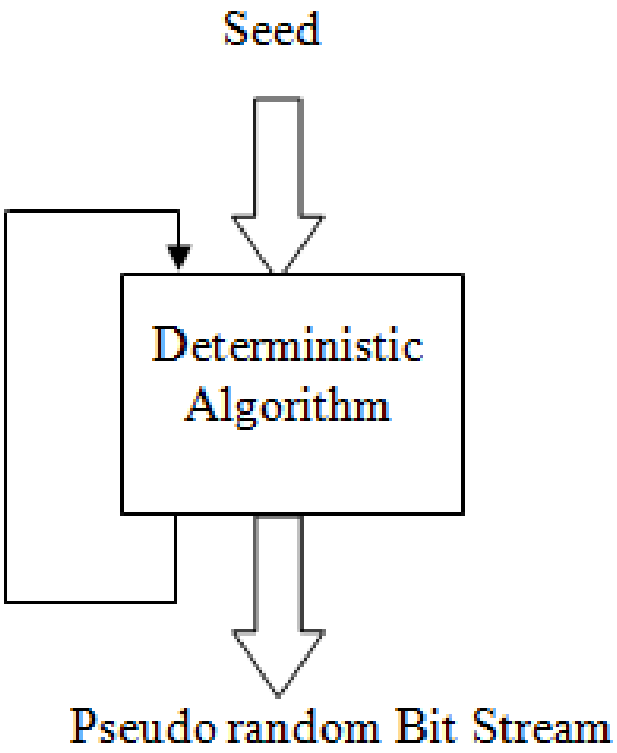
Generacija naključnih števil je proces, ki pogosto z generatorjem naključnih števil (RNG) ustvari zaporedje števil ali simbolov, ki jih ni mogoče razumno predvideti bolje, kot po naključnem ugibanju. Generatorji naključnih števil so lahko resnično naključni strojni generatorji naključnih števil (HRNG), ki ustvarjajo naključna števila kot funkcijo trenutne vrednosti nekega naključnega vira iz fizičnega okolja. Ta se nenehno spreminja na način, ki ga je v praksi nemogoče modelirati. Števila lahko generirajo

[5]Vir slike: `https://www.semanticscholar.org/paper/Adaptive-Push-Then-Pull-\`
`\Gossip-Algorithm-for-Gupta-Maali/7db5d8e50eac3b0e854ead460ff5adcf5498bd59/figure/0`

generatorji psevdo naključnih števil (PRNG), ki ustvarjajo številke, ki so videti naključno, a so dejansko deterministične, in jih je mogoče reproducirati, če je stanje generatorja znano.

Obstaja več računskih metod za ustvarjanje psevdo naključnih števil. Vsi ne dosegajo cilja resnične naključnosti, čeprav lahko z različnim uspehom izpolnijo nekatere statistične teste naključnosti. Ti testi so namenjeni merjenju, kako nepredvidljivi so rezultati generatorja (to je, v kolikšni meri so njihovi vzorci razpoznavni). Zaradi tega so na splošno neuporabni za aplikacije, kot je kriptografija. Obstajajo pa tudi skrbno oblikovani kriptografsko varni generatorji psevdo naključnih števil (CSPRN) s posebnimi funkcijami, ki so posebej zasnovane za uporabo v kriptografiji.

Obstaja več načinov za ustvarjanje naključnega števila na podlagi funkcije gostote verjetnosti (probability density function). Te metode navadno vključujejo pretvorbo naključnega števila. Zaradi tega te metode delujejo enako dobro pri ustvarjanju tako psevdo naključnih kot resničnih naključnih števil. Ena metoda, imenovana inverzijska metoda, vključuje integracijo do območja, ki je večje ali enako naključnemu številu (ki ga je treba ustvariti med 0 in 1 za pravilne porazdelitve). Druga metoda, imenovana metoda sprejema-zavrnitve, vključuje izbiro vrednosti x in y ter preverjanje, ali je funkcija x večja od vrednosti y. Če je, je vrednost x sprejeta. V nasprotnem primeru se vrednost x zavrne in algoritem poskusi znova. [14]



Slika 6: [6]Slika prikazuje enostavno shemo psevdo naključnega generatorja števil PRNG(Deterministic Algorithm), ki kot vhodni podatek prejme seme, s katerega v zanki generira naključno zaporedje, do želene dolžine.

Večina računalniško generiranih naključnih števil uporablja PRNG, to so algoritmi, ki lahko samodejno ustvarijo dolge serije števil z dobrimi naključnimi lastnostmi, vendar se sčasoma zaporedje ponovi (ali pa se poraba pomnilnika poveča brez omejitev). Ta naključna števila so v mnogih situacijah sprejemljiva [15]. Niz vrednosti, ki jih ustvarijo takšni algoritmi, je na splošno določen s fiksno številko, imenovano seme (seed), kot je prikazano na sliki 6. Eden najpogostejših PRNG je linearni kongruencialni generator, ki za generacijo števil uporablja ponovitev $X_{n+1} = (aX_n + b) \mod m$, kjer so a, b in m velika cela števila in je X_{n+1} naslednik X v vrsti psevdo naključnih števil. Največje število številk, ki jih formula lahko ustvari, je eno manjše od modula, $m-1$. Razmerje ponavljanja lahko razširimo na matrike, saj imajo le-te veliko daljša obdobja in boljše statistične lastnosti [14].

2.5 Tehnologija veriženja blokov

V sledečem poglavju je za namen boljšega razumevanja okolja, v katerem deluje razvit algoritem, podrobneje predstavljena tehnologija veriženja blokov (blockchain). Pred-

[6]Vir slike: `https://www.researchgate.net/figure/Pseudo-Random-Number-Generator_fig2_277905704`

stavljeni so glavni koncepti in protokoli, ki omogočajo delovanje sistema. Na koncu so predstavljeni še glavni problemi, s katerimi se razvijalci in znanstveniki spopadajo pri razvoju tehnologije. Po definiciji je tehnologija veriženja blokov naraščajoč seznam zapisov, imenovanih bloki, ki so med seboj povezani s pomočjo kriptografskih tehnik [16].

Veriženje blokov je leta 2008 izumila oseba (ali skupina ljudi) pod imenom Satoshi Nakamoto in je služila kot knjiga javnih transakcij kripto valute Bitcoin [16]. Identiteta Satoshija Nakamota do danes ostaja neznana. Tehnologija je postala prva digitalna valuta, ki je rešila problem dvojne porabe sredstev brez potrebe po zaupanja vrednem organu ali centralnem strežniku. Zasnova Bitcoina je navdihnila tudi druge aplikacije, saj so verige, ki so odprte javnosti pogosto uporabljene tudi v drugih kriptovalutah [17].

Tehnologija veriženja blokov je uporabljena in integrirana na večih področjih. Primarna uporaba tehnologije je kot porazdeljena transakcijska knjiga v kriptovalutah, kot je Bitcoin. Od leta 2016 nekatera podjetja preizkušajo tehnologijo in preučujejo učinke blockchaina na organizacijsko učinkovitost na nižjih nivojih v svojem zaledju [18].

Leta 2019 je bilo ocenjeno, da je bilo v tehnologijo veriženja blokov vloženih približno 2,9 milijarde dolarjev, kar predstavlja 89% povečanje v primerjavi z 2018. Poleg tega je *International Data Corp* ocenil, da bodo naložbe podjetij v tehnologijo veriženja blokov do leta 2022 dosegle 12,4 milijarde USD [19]. Poleg tega po podatkih *PricewaterhouseCoopers* (PwC), drugega največjega omrežja profesionalnih storitev na svetu, tehnologija blockchain lahko do leta 2030 ustvari letno poslovno vrednost več kot 3 bilijone dolarjev. Oceno PwC dodatno dopolnjuje študija iz leta 2018, v kateri je PwC anketiral 600 vodstvenih delavcev in ugotovil, da je 84% vsaj delno izpostavljenih uporabi tehnologije veriženja blokov, kar kaže na veliko povpraševanje in zanimanje za to tehnologijo [23].

Uporaba tehnologije veriženja blokov se je močno povečala tudi od leta 2016. Po statističnih podatkih iz leta 2020 je bilo v letu 2020 več kot 40 milijonov denarnic v verigi blokov v primerjavi s približno 10 milijoni denarnic v verigi blokov v letu 2016.

2.6 Tehnologije v veriženju blokov

2.6.1 Razpršilna funkcija

Razpršilna funkcija je vsaka funkcija, ki jo lahko uporabite za preslikavo podatkov poljubne velikosti v vrednosti fiksne velikosti. Vrednosti, ki jih vrne razpršilna funkcija, se imenujejo razpršene vrednosti, kode razpršitve, izvlečki ali preprosto razpršitve. Vrednosti se običajno uporabljajo za indeksiranje tabele s fiksno velikostjo, imenovane razpredelnica.

Razpršilne funkcije in z njimi povezane zgoščevalne tabele se uporabljajo v aplikacijah za shranjevanje in iskanje podatkov za dostop do podatkov v majhnem in v

skoraj konstantnem času. Njihovo iskanje zahteva le delno večjo količino prostora kot je potrebno za shranjevanje celotnih podatkov. Razprševanje je računalniško in shranjevalno učinkovita oblika dostopa do podatkov, ki se izogiba nelinearnemu času dostopa urejenih in neurejenih seznamov in strukturiranih dreves ter pogosto eksponentnim zahtevam po shranjevanju za neposreden dostop.

Razpršilna funkcija vzame vnos kot ključ, ki je povezan z datumom ali zapisom in se uporablja za njegovo identifikacijo v aplikaciji za shranjevanje in iskanje podatkov. Ključi so lahko fiksne dolžine, na primer celo število, ali spremenljive dolžine, kot je na primer ime. V nekaterih primerih je ključ lahko datum. Izhod je koda razpršitve, ki se uporablja za indeksiranje razpredelnice, ki vsebuje podatke, zapise ali kazalce nanje.

Za razpršilno funkcijo se lahko šteje, da opravlja tri funkcije:

- Ključe spremenljive dolžine pretvori v vrednosti s fiksno dolžino (običajno dolžino strojne besede ali manj) tako, da jih zloži z besedami ali drugimi enotami z uporabo operaterja za ohranjanje parnosti, kot sta ADD ali XOR.

- Mešanje bitov podatka, tako da dobljene vrednosti enakomerno porazdeli po prostoru rešitev.

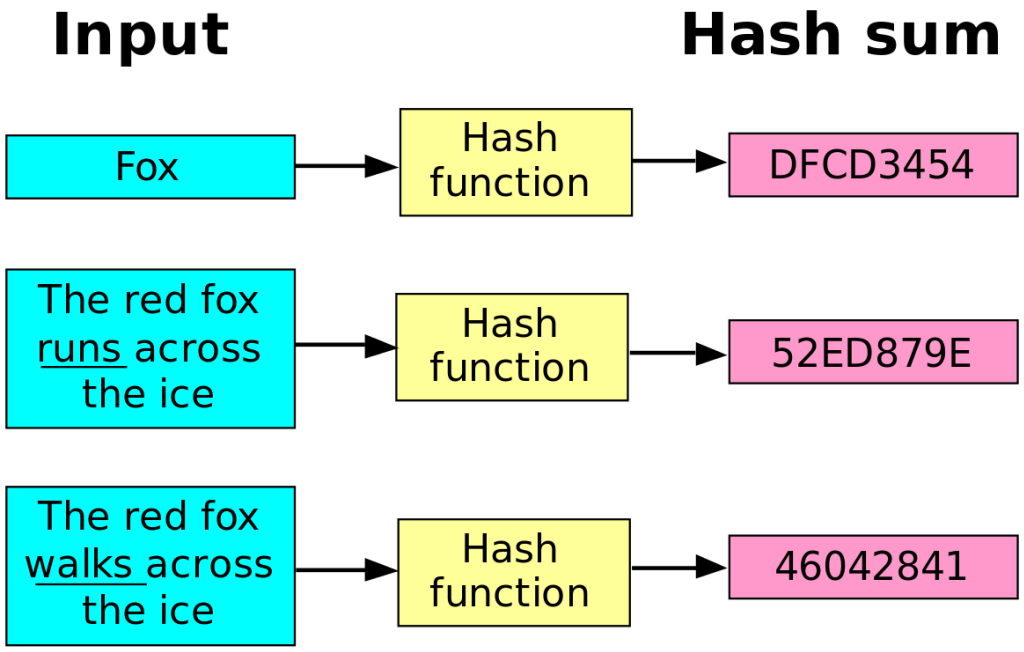
- Preslika vrednosti ključev v vrednosti, manjše ali enake velikosti tabele

Dobra razpršilna funkcija izpolnjuje dve osnovni lastnosti:

- izračunana mora biti zelo hitro za poljuben vhodni podatek in

- imeti nizko količino podvajanj izhodnih vrednosti (trki).

Razpršilne funkcije se za učinkovitost zanašajo na ustvarjanje ugodnih porazdelitev verjetnosti, kar skrajša čas dostopa na skoraj konstantno. Uporaba razpršilnih funkcij temelji na statističnih lastnostih interakcije ključa in funkcije. Vedenje je v najslabšem primeru neznosno slabo, vendar z izjemno majhno verjetnostjo. Obnašanje v povprečnem primeru pa je lahko skoraj optimalno (minimalno trčenje) [24].



Slika 7: [7]Slika prikazuje primer razpršilne funkcije. Vsi vhodi so različnih dolžin, toda imajo veliko podobnosti. Vsi rezultati so enakih dolžin, vendar so si med seboj zelo različni.

2.6.2 Merklova drevesa

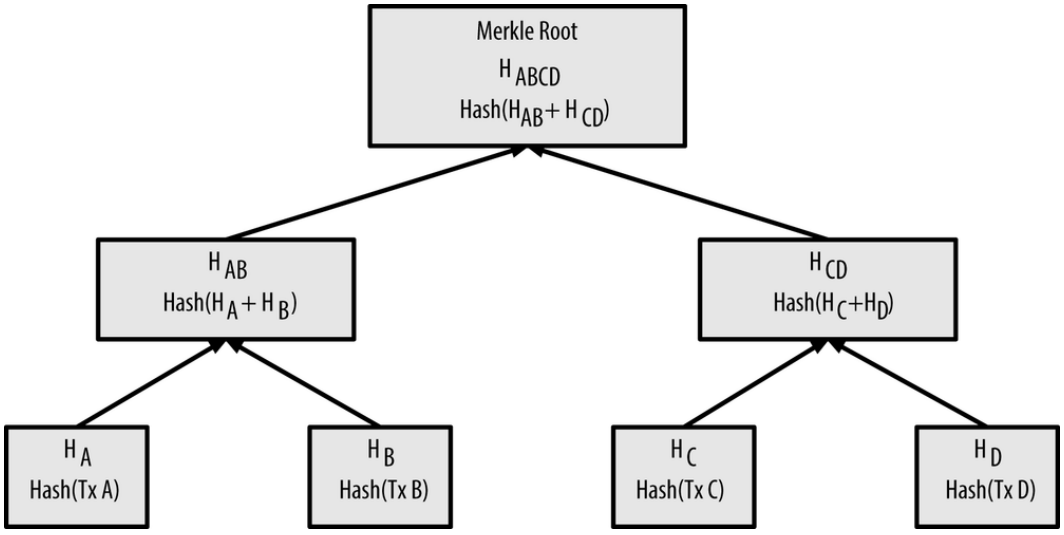
V kriptografiji in računalništvu je drevo razpršitve ali Merkle drevo tisto drevo, v katerem je vsako listno vozlišče označeno s kriptografskim hashom podatkovnega bloka. Vsako vozlišče, ki ni list, je označeno s kriptografskim hashom hashov svojih podrejenih vozlišč. Razpršena drevesa omogočajo učinkovito in varno preverjanje vsebine velikih podatkovnih struktur. Razpršena drevesa so posplošitev zgoščenih seznamov in verig zgoščevanja.

Če želimo dokazati, da je listno vozlišče del danega binarnega drevesa razpršitve, moramo izračunati število hashov, sorazmernih z logaritmom števila listnih vozlišč drevesa. To je v nasprotju s seznami razpršitev, kjer je število sorazmerno z številom listnih vozlišč [27].

Koncept razpršenih dreves je dobil ime po Ralphu Merkleu, ki ga je leta 1979 patentiral [26].

Na vrhu Merkle drevesa se nahaja korneski hash ali glavni hash. Ko iz peer-to-peer omrežja prenašamo datoteke, navadno iz zanesljivega vira najprej pridobimo korenski

[7]Vir slike: `https://sl.wikipedia.org/wiki/Zgo\%C5\%A1\%C4\%8Devalna\_funkcija`
[8]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt_0702.png`



Slika 8: [8]Slika prikazuje shemo Merkle drevesa. Puščice nakazujejo smer gradnje drevesa, saj je starševsko vozlišče sestavljeno iz hashov svojih otrok.

hash. Ko smo pridobili hash, lahko datoteke prenesemo od poljubnega nezanesljivega vira v omrežju. Nato lahko primerjamo korenski hash prejete datoteke z zanesljivim. V primeru neskladja smo prejeli poškodovano ali ponarejeno različico datoteke.

2.7 Struktura

Blokovna veriga je decentralizirana, porazdeljena in pogosto javna digitalna knjiga, sestavljena iz zapisov, imenovanih bloki. Bloki se uporabljajo za beleženje transakcij v številnih računalnikih, tako da ni mogoče spremeniti podatkov preteklih blokov brez spreminjanja vseh naslednjih blokov [17]. To udeležencem v omrežju omogoča neodvisno in relativno nezahtevno preverjanje in revizijo transakcij. Baza podatkov blockchain se upravlja avtonomno z uporabo omrežja peer-to-peer in porazdeljenega strežnika za časovno označevanje. Poganja jo množično sodelovanje, ki temelji na kolektivnih interesih [20]. Takšna zasnova olajša robusten potek dela, kjer je negotovost udeležencev glede varnosti podatkov majhna. Po strukturi lahko tehnologijo veriženja blokov razdelimo v 5 logičnih plasti [25]:

- infrastruktura (naprave)
- omrežje (odkrivanje vozlišč, širjenje informacij in preverjanje)
- soglasje ali konsenz (dokaz o delu, dokaz o vložku)
- podatki (bloki, transakcije)
- aplikacije (pametne pogodbe in decentralizirane aplikacije)

Tabela 1: Tabela sestave individualnega bloka v verigi

Velikost[bajti]	Polje	Opis
4	velikost bloka	Velikost bloka v bajtih po tem polju
80	glava bloka	Vsa polja v glavi bloka
1-9	Število transakcij	Število vseh transakcij zajetih v bloku
spremenljivo	transakcije	vse transakcije zapisane v bloku

Tabela 2: Tabela sestave glave bloka

Velikost[bajti]	Polje	Opis
4	Različica	Številka različice protokola v katerem je blok generiran
32	Prejšnji hash	Referenca na hash prejšnjega bloka v verigi
32	Merkle koren	Korenski hash Merkle drevesa transakcij v bloku
4	časovni žig	Približek časa ob katerem je bil generiran blok v Unix časovnem formatu
4	Težavnost	Težavnostna tarča dokaza o delu za ta blok
4	Nonce	Števec, ki je uporabljen pri dokazu o delu

2.8 Blok

Bloki vsebujejo serije veljavnih transakcij, ki so zgoščene in kodirane v Merkle drevo [17]. Vsak blok je s prejšnjim blokom povezan tako, da vključuje kriptografski zapis (hash) le-tega. Povezani bloki tako tvorijo verigo. Ta ponavljajoči se postopek potrjuje celovitost prejšnjega bloka vse do začetnega bloka, ki je znan kot izvorni blok [17].

Blok je sestavljen iz glave, ki vsebuje metapodatke, čemur sledi dolg seznam transakcij, ki sestavljajo večino njegove velikosti. Glava bloka je velika 80 bajtov, povprečna transakcija je najmanj 250 bajtov, povprečni blok pa vsebuje več kot 500 transakcij [21].

Število za enkratno uporabo ("nonce"), težavnostna tarča in časovni žig se uporabljajo v rudarskem procesu in bodo obravnavani podrobneje v naslednjih poglavjih.

Pomembno je vedeti, da hash posameznega bloka dejansko ni vključen v podatkovno strukturo bloka, niti takrat, ko se blok prenaša v omrežju, niti ko je shranjen v trajnem pomnilniku vozlišča kot del verige blokov. Namesto tega hash bloka izračuna vsako vozlišče, ko prejme od omrežja nov blok. Razpršitev bloka je lahko shranjena v ločeni tabeli zbirke podatkov kot del metapodatkov bloka, da se olajša indeksiranje in omogoča hitrejši prenos blokov z diska. [21]

Listing 2.1: Hash prvega bloka

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

Drugi način za identifikacijo bloka je njegov položaj v verigi blokov, imenovan višina bloka. Prvi blok, ki je bil ustvarjen, je na višini bloka 0. Blok je tako mogoče identificirati na dva načina, in sicer s sklicevanjem na hash bloka ali s sklicevanjem na višino bloka. Vsak naslednji blok, ki je dodan "na vrhu"bloka, je za eno mesto "višje"v verigi blokov. Višina bloka 1. januarja 2014 je bila približno 278.000, kar pomeni, da je bilo 278.000 blokov zloženih na vrhu prvega bloka, ustvarjenega januarja 2009. [21]

Vozlišča na omrežju vzdržujejo lokalno kopijo verige blokov, ki se začne z izvornim blokom. Lokalna kopija verige blokov se nenehno posodablja, ko najdemo nove bloke in jih uporabimo za razširitev verige. Ko vozlišče sprejema dohodne bloke iz omrežja, jih bo potrdilo in jih nato povezalo z obstoječo verigo blokov. Za vzpostavitev povezave bo vozlišče pregledalo glavo dohodnega bloka in poiskalo hash prejšnjega bloka. Ta blok je starš novega bloka, zato je ta novi blok podrejen zadnjemu bloka v verigi in razširja obstoječo verigo blokov. Vozlišče doda nov blok na konec verige. [21]

2.9 Rudarjenje in konsenz

Vsi tradicionalni plačilni sistemi so odvisni od modela zaupanja, ki ima nek osrednji organ za opravljanje storitev, ki preverja in izvaja vse transakcije. Bitcoin nima osrednjega organa, vendar ima kljub temu vsako vozlišče popolno kopijo javne knjige, ki ji lahko zaupa kot verodostojen zapis. Blokovne verige ne ustvari osrednji organ, ampak jo vsako vozlišče v omrežju sestavi neodvisno. Vsako vozlišče v omrežju, ki deluje na podlagi informacij, ki se prenašajo prek nezaščitenih omrežnih povezav, pride do istega zaključka in sestavi kopijo iste javne knjige kot vsi ostali. [21] To poglavje obravnava postopek, s katerim omrežja kot so bitcoin in ethereum dosežejo globalno soglasje brez osrednjih pooblastil. Bolj natančno sta opisana koncepta *dokaz o delu* in *dokaz o vložku*.

Bitcoin decentralizirano soglasje izhaja iz medsebojnega delovanja štirih procesov, ki se pojavljajo neodvisno na vozliščih v omrežju: [21]

- Neodvisno preverjanje vsake transakcije s strani vsakega vozlišča na podlagi izčrpnega seznama meril;
- Neodvisno združevanje teh transakcij v nove bloke z rudarskimi vozlišči, skupaj z dokazanim izračunom po algoritmu za preverjanje dela;
- Neodvisno preverjanje novih blokov od vsakega vozlišča in sklopa v verigo;
- Neodvisna izbira verige, v katero je bilo kumulativno vloženega več dela (preko dokaza o delu) v primeru večih predlog verig.

Nekatera vozlišča v bitcoin omrežju so specializirana vozlišča, imenovana rudarji. Vsako vozlišče, ki prejme transakcijo, bo najprej preverilo transakcijo. To zagotavlja, da se po omrežju širijo samo veljavne transakcije, neveljavne transakcije pa se zavržejo na prvem vozlišču, na katere naleti. Po potrditvi bo vozlišče transakcijo shranilo v področje transakcij, kjer počakajo, dokler jih ni mogoče vključiti v blok. Vozlišče zbira, potrjuje in posreduje nove transakcije tako kot vsako drugo vozlišče. Za razliko od drugih vozlišč bo vozlišče rudarja te transakcije nato združilo v tako imenovani kandidatni blok. [21]

Ko rudarsko vozlišče prejme nov blok in ga potrdi, preveri vse svoje zabeležene transakcije in jih referencira z novimi transakcijam v prejetem bloku. Transakcije v bloku so izvršene, zato jih vozlišče izbriše iz čakajočih transakcij. Vse transakcije, ki so ostale so še vedno nepotrjene in čakajo na nov blok. Po validaciji rudarsko vozlišče začne graditi naslednji kandidatni blok. Te bloke imenujemo kandidatni bloki, ker še niso potrjeni, saj ne vsebujejo dokaza o delu. Blok postane veljaven šele, ko rudarju uspe izvesti dokaz dela za dani blok. [21]

Prva transakcija, ki jo rudar vstavi v kandidatni blok je posebna transakcija, ki jo imenujemo transakcija generiranja. To transakcijo je zgradilo rudarsko vozlišče in je njegova nagrada za trud pri rudarjenju. Transakcija je ustvarjena kot plačilo v rudarjevo denarnico. [21]

Rudarji prejemajo dve vrsti nagrad za rudarjenje, to so novi kovanci, ustvarjeni z vsakim novim blokom in provizije za transakcije iz vseh transakcij, ki so vključene v blok. Za pridobitev te nagrade rudarji tekmujejo pri reševanju težkega matematičnega problema, ki temelji na kriptografskemu algoritmu razpršitve. Rešitev problema, imenovanega dokaz dela, je vključen v nov blok in deluje kot dokaz, da je rudar porabil veliko računalniškega napora. Rudarstvo je izum, zaradi katerega je bitcoin poseben, decentraliziran varnostni mehanizem, ki je osnova za peer-to-peer digitalno valuto. Nagrada za nove kovance in transakcijske provizije je spodbujevalna shema, ki usklajuje dejanja rudarjev z varnostjo omrežja, hkrati pa ustvarja denarno ponudbo.

Preprosto povedano je rudarjenje postopek večkratnega preslikavanja glave bloka skozi razpršilno funkcijo s spreminjanjem enega parametra, dokler nastalo razpršitev ne ustreza določenemu cilju. Rezultata funkcije razpršitve ni mogoče vnaprej določiti, niti ni mogoče ustvariti vzorca, ki bo ustvaril določeno vrednost zgoščevanja. Ta funkcija zagotavlja, da je edini način za ustvarjanje razpršenega rezultata, ki se ujema z določenim ciljem, poskusiti znova in znova ter naključno spreminjati vnos, dokler se po naključju ne pojavi želeni rezultat razpršitve. [21]

2.9.1 Dokaz o delu

2.9.1.1 Razprševanje

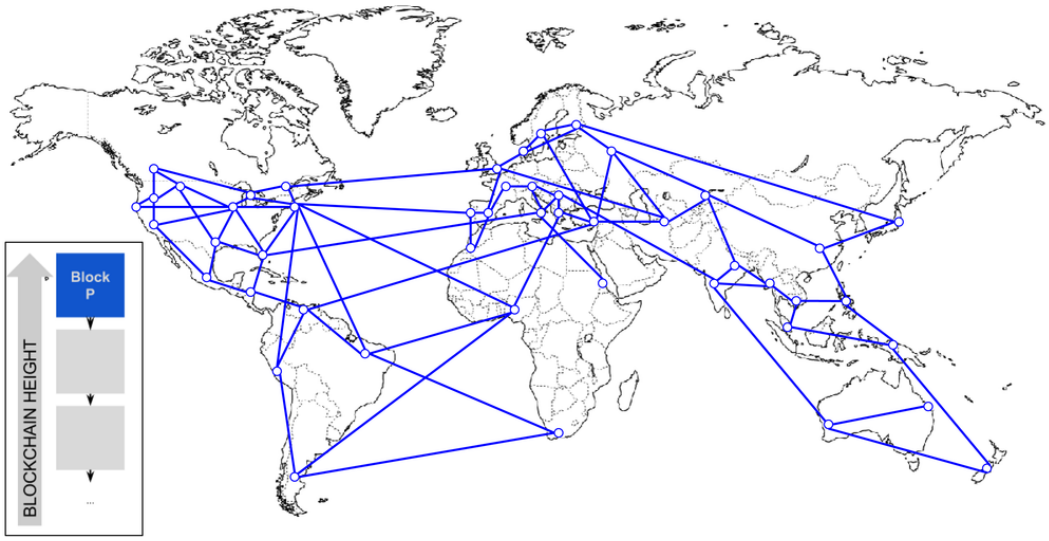
Kot je predstavljeno v prejšnjih poglavjih, razpršilni algoritem sprejme vnos podatkov poljubne dolžine in ustvari deterministični rezultat s fiksno dolžino ali digitalni prstni odtis vhoda (Bitcoin specifično uporablja razpršilni algoritem pod imenom SHA256). Za vsak enak vnos bo nastala vedno enaka razpršitev, ki jo lahko vsakdo, ki izvaja isti algoritem razpršitve, zlahka izračuna in preveri. Ključna značilnost kriptografskega algoritma razpršitve je, da je skoraj nemogoče najti dva različna vhoda, ki proizvajata isti prstni odtis. Kot posledica je tudi skoraj nemogoče izbrati vnos tako, da se ustvari želeni prstni odtis, razen poskusov naključnih vnosov. [21] Število, ki se v takem scenariju uporablja kot spremenljivka, se imenuje nonce, kar je izpeljanka iz *"number only used once"* ali število, ki je uporabljeno samo enkrat.

Če želimo iz tega algoritma narediti izziv, nastavimo poljuben cilj: poiščite nonce, ki ustvari šestnajstiško razpršitev, ki se začne z ničlo. Če je izhod hash funkcije enakomerno porazdeljen, bi pričakovali, da bomo našli rezultat z 0 kot šestnajstiško predpono enkrat na vsakih 16 hashov (eno od 16 šestnajstiških številk od 0 do F). Temu pragu pravimo cilj in naloga rudarja je najti razpršitev, ki je številčno manjša od cilja. Če zmanjšamo cilj, postane naloga iskanja razpršitve, ki je manjša od cilja, vse težja.

Prav tako deluje dokazilo o delu v omrežju Bitcoin. Rudar sestavi kandidatni blok, napolnjen s transakcijami. Nato rudar izračuna hash glave tega bloka in preveri, ali je manjši od trenutnega cilja. Če razpršitev ni manjša od cilja, bo rudar spremenil nonce (običajno ga le povečal za eno) in poskusil znova. Ob trenutnih težavah v omrežju bitcoin morajo rudarji povprečno poskusiti štiri milijarde-krat, preden najdejo nonce, ki povzroči dovolj nizko razpršitev glave bloka. [21]

2.9.1.2 Težavnostni cilj

Cilj določa težavnost in zato vpliva na to, kako dolgo traja, da se najde rešitev za algoritem dokazovanja dela. Bloki Bitcoina se v povprečju ustvarijo vsakih 10 minut. To je utrip omrežja in določa pogostost izdajanja novih kovancev in hitrost poravnave transakcij. Hitrost te ure ne sme ostati konstantna le kratkoročno, ampak v dolgih desetletjih. V tem času se pričakuje, da se bo moč računalnikov še naprej hitro povečevala. Poleg tega se bo nenehno spreminjalo tudi število udeležencev rudarjenja in računalnikov, ki jih uporabljajo. Če želite ohraniti čas ustvarjanja blokov pri 10 minutah, je treba glede na te spremembe prilagoditi težavnost rudarjenja. Dejansko je težavnostni cilj dinamični parameter, ki se občasno prilagodi, da bo dosegel 10-minutni cilj bloka. Preprosto povedano, cilj težavnosti je nastavljen na kakršno koli komutativno moč rudarjev, ki bo povzročila 10-minutni interval bloka. [21]

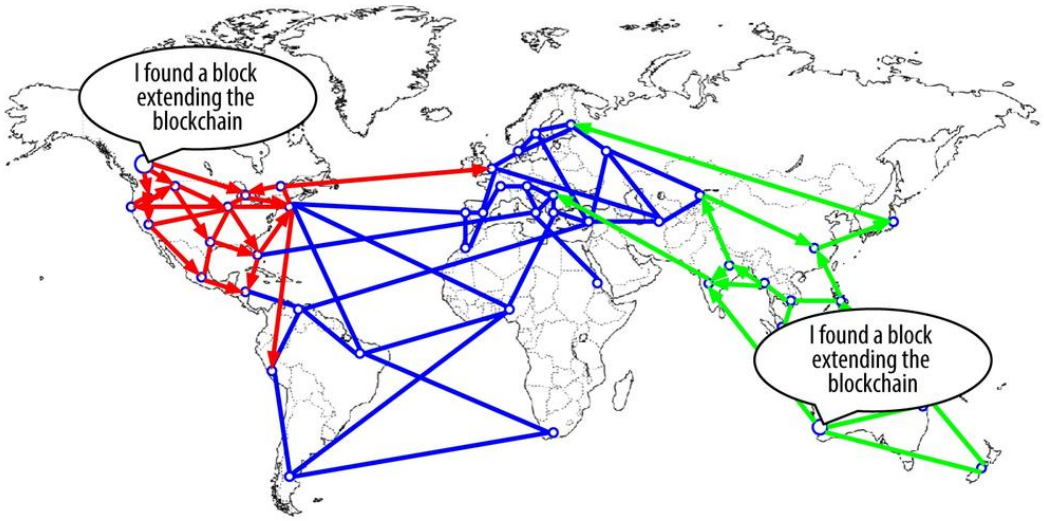


Slika 9: [9]Slika prikazuje razcep verige: Predstavitev geografske topologije je poenostavitev predstave omrežja, ki se uporablja za ponazoritev razcepa. Slika prikazuje stanje pred razcepom, kjer vsa vozlišča poznajo enako različico verige.

2.9.1.3 Razcep verige

Ker je blockchain decentralizirana struktura podatkov, njene različne kopije niso vedno skladne. Bloki lahko pridejo do različnih vozlišč ob različnih časih, zaradi česar imajo lahko vozlišča različno perspektivo verige blokov. Da bi to rešili, vsako vozlišče vedno izbere in poskuša razširiti verigo blokov, ki predstavlja največji vložek v dokaz dela, znano tudi kot najdaljša veriga ali veriga največje kumulativne težavnosti. S seštevanjem matematičnega dela, zabeleženega v vsakem bloku v verigi, lahko vozlišče izračuna skupno količino vložka v dokazilo o delu, ki je bilo vloženo za ustvarjanje te verige. V tem času, ko vsa vozlišča izberejo najdaljšo kumulativno težavno verigo, globalno omrežje sčasoma konvergira v skupno stanje. Razcepi se pojavijo kot začasna nedoslednost med različicami verige blokov in se odpravijo s končno konvergenco, ko se enemu od razcepov doda več blokov. [21]

V naslednjih nekaj slikah je predstavljen razplet dogodka razcepa verige v celotnem omrežju. Diagram je poenostavljena predstavitev bitcoina kot globalnega omrežja. V resnici topologija omrežja bitcoin ni geografsko organizirana. Namesto tega tvori mrežno mrežo medsebojno povezanih vozlišč, ki bi lahko bila geografsko zelo oddaljena drug od drugega. Predstavitev geografske topologije je poenostavitev, ki se uporablja za ponazoritev razcepa. V resničnem omrežju se "razdalja"med vozlišči meri v tako imenovanih skokih od vozlišča do vozlišča, ki predstavljajo število vmesnih vozlišč po najbližji poti od enega do drugega, ne pa na njihovi fizični lokaciji. Za ilustracijo so različni bloki prikazani kot različne barve, ki se razprostirajo po omrežju in obarvajo povezave, ki jih prečkajo.



Slika 10: [10]Slika prikazuje razcep verige: Predstavitev geografske topologije je poenostavitev predstave omrežja, ki se uporablja za ponazoritev razcepa. Dve vozlišči sočasno opravita dokaz o delu in propagirata blok.(rdeči in zeleni del omrežja)

Razcep se pojavi, kadar se propagirata dva kandidatna bloka, ki tekmujeta za oblikovanje najdaljše verige blokov. To se zgodi v normalnih pogojih, kadar dva rudarja rešita algoritem dokazovanja dela v skoraj enakem času. Ko oba rudarja odkrijeta rešitev za ustrezne bloke kandidatov, takoj oddata svoj žmagovalni"blok svojim neposrednim sosedom, ki začnejo širiti blok po omrežju. Vsako vozlišče, ki prejme veljaven blok, ga bo vključilo v svojo verigo blokov in razširilo verigo blokov za en blok. Če to vozlišče kasneje vidi drugi blok kandidatov, ki razširja istega nadrejenega, poveže drugega kandidata v sekundarni verigi. Posledično bodo nekatera vozlišča najprej »videla« en kandidatni blok, druga pa bodo videla drugi kandidatni blok in nastali bosta dve konkurenčni različici verige blokov. [21]

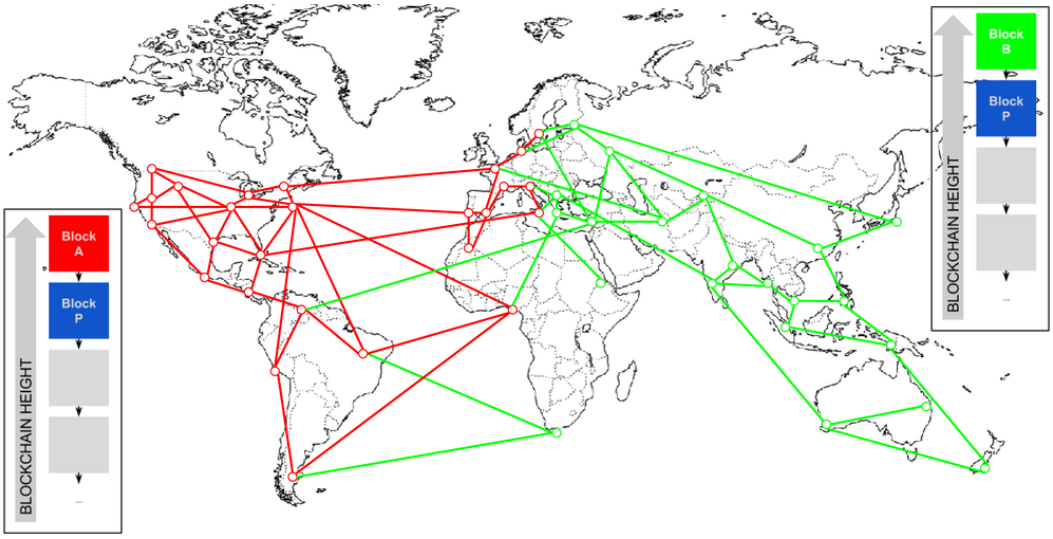
Na sliki 10 vidimo dva rudarja, ki skoraj istočasno dokažeta dva različna bloka. Oba bloka sta otroka modrega bloka, ki naj bi podaljšal verigo z gradnjo na vrhu modrega bloka. Za lažje sledenje je eden vizualiziran kot rdeči blok iz Kanade, drugi pa kot zeleni blok iz Avstralije. Kot je prikazano na sliki 11, ko se oba bloka širita, nekatera vozlišča najprej prejmejo blok "rdeče", nekatera pa najprej želena". Omrežje razdeli na dve različni perspektivi verige blokov, na eni strani je rdeči blok, na drugi pa je zeleni blok. [21]

Od tega trenutka bodo vozlišča omrežja, ki so najbližja (topološko, ne geografsko)

[9]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt\_0802.png`

[10]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt\_0803.png.jpg`

[11]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt\_0804.png`



Slika 11: [11]Slika prikazuje razcep verige: Predstavitev geografske topologije je poenostavitev predstave omrežja, ki se uporablja za ponazoritev razcepa. Bloka rdečega in zelenega vozlišča se propagirata skozi celotno omrežje in razcepita omrežje na dva dela.

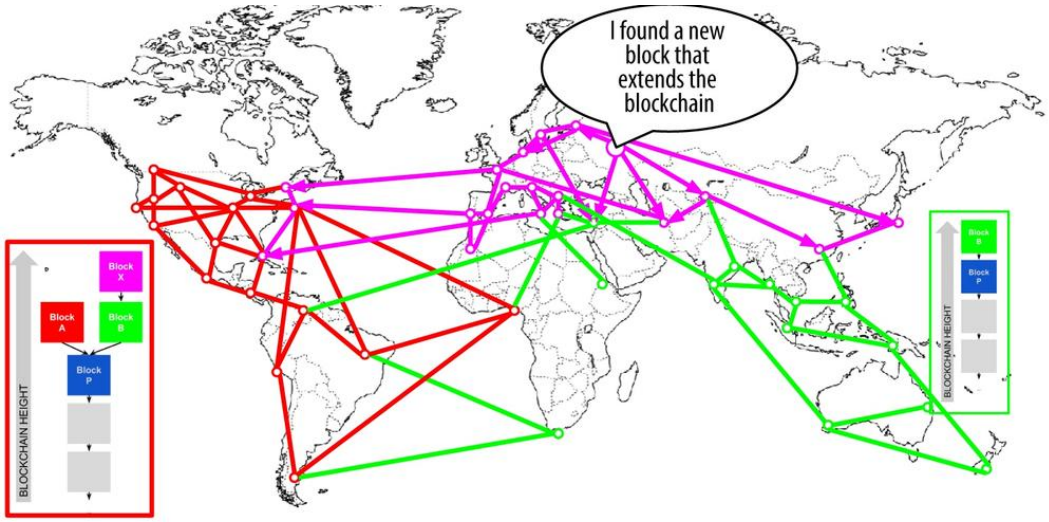
kanadskemu vozlišču, najprej slišala za rdeč blok in bodo ustvarila novo verigo največje kumulativne težavnosti z rdečim kot zadnjim blokom v verigi, pri čemer se ignorira zeleni kandidatni blok, ki pride malo kasneje. Medtem bodo vozlišča, ki so bližje avstralskemu vozlišču, vzela ta blok za zmagovalca in razširila verigo blokov z zelenim kot zadnjim blokom, pri tem pa prezrla "rdečo", ko pride nekaj sekund kasneje. Vsi rudarji, ki so prvi videli "rdeče", bodo takoj zgradili kandidatne bloke, ki se sklicujejo na rdeč blok, in začeli poskušati rešiti dokaz o delu teh blokov kandidatov. Rudarji, ki so namesto tega sprejeli zeleno, bodo začeli graditi na zeleni in razširjati to verigo. [21]

Razcep se skoraj vedno razreši v enem bloku. Ker je del večje moči omrežja namenjen gradnji na vrhu rdeče verige, je drugi del moči razprševanja osredotočen na nadgradnjo na zeleni. Tudi če je moč razprševanja skoraj enakomerno razdeljena, bo verjetno en niz rudarjev našel rešitev in jo razširil, preden bo druga skupina rudarjev našla svoje rešitve. Ta novi blok takoj razširijo in celotno omrežje ga vidi kot veljavno rešitev, kot je prikazano na sliki 13. [21]

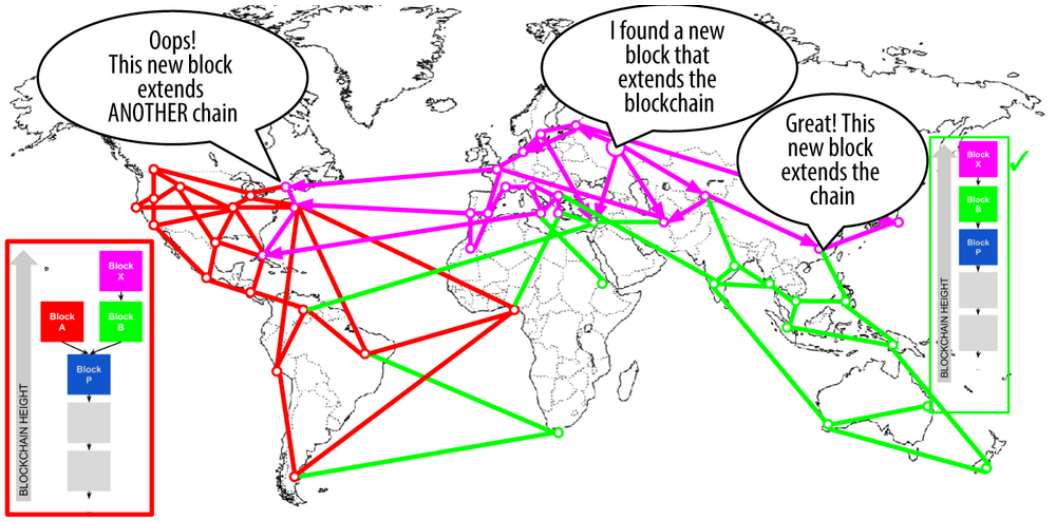
Teoretično je mogoče, da se razcepi podaljšajo na dva bloka, če rudarji na nasprotnih štraneh"prejšnjega razcepa skoraj istočasno najdejo nova dva bloka. Vendar je verjetnost, da se to zgodi, zelo majhna. Medtem ko se razcepi z enim blokom lahko pojavijo skoraj vsak teden, so vilice z dvema blokoma izjemno redke. [21]

Bitcoin blokovni interval 10 minut je oblikovni kompromis med hitrim časom po-

[12]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt\_0805.png.jpg`
[13]Vir slike: `https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/images/msbt\_0806.png`



Slika 12: [12]Slika prikazuje razcep verige: Predstavitev geografske topologije je poenostavitev predstave omrežja, ki se uporablja za ponazoritev razcepa. Novo vozlišče opravi dokaz o delu v zelenem podomrežju in propagira nov blok.



Slika 13: [13]Slika prikazuje razcep verige: Predstavitev geografske topologije je poenostavitev predstave omrežja, ki se uporablja za ponazoritev razcepa. Omrežje konvergira nazaj v skupno stanje ob sprejetju najdaljše verige.

trditve (poravnavo transakcij) in verjetnostjo razcepa. Hitrejši čas generacije bloka bi pospešil hitrejše transakcije, vendar bi privedel do pogostejših razcepov omrežja, medtem ko bi počasnejši čas generiranja bloka zmanjšal število razcepov, a upočasnil poravnavo.

3 Algoritem za pošiljanje sporočil

V tem poglavju je razloženo delovanje algoritma za pošiljanje sporočil po decentraliziranih sistemih. Algoritem deluje na osnovi drevesne strukture in kot vodilni koncept uporablja psevdo naključnost.

3.1 Izzivi pri pošiljanju sporočil v decentraliziranih sistemih

Kot je razloženo v prejšnjih poglavjih, se pošiljanje sporočil spopada z mnogimi izzivi. Ti izzivi so ponavadi algoritmične narave, kot so potrjevanje verodostojnosti sporočil in spopadanje s problemom dveh generalov [9]. Takšni problemi so višje nivojski in njihovo reševanje se v splošnem izvaja na samem vozlišču. To pomeni, da lahko predpostavimo pravilo delovanje zunanjih dejavnikov in povezanih tehnologij. Ko pa govorimo o dejanskem pošiljanju sporočil iz vozlišča do vozlišča, se v praksi spopadamo še z mnogimi drugimi problemi. Da podatki uspešno preidejo iz ene naprave do druge, mora naprava izvesti mnogo protokolov in procedur v nižjih nivojih delovanja računalnikov. Nato sporočilo preko protokola TCP/IP(Transmission Control Protocol/Internet Protocol). Kljub robustnosti teh tehnologij, pa še vedno lahko pride do mnogih napak.

Glavni težavi s katerimi se spopadamo pri pošiljanju sporočil, sta obe povezani z napakami, katere niso direktno napake algoritma, temveč so ponavadi strojne napake pri napravi, ki je udeležena v procesu prenašanja sporočila. Pri uporabi UDP protokola, pri katerem pošiljatelj ne preverja ali je prejemnik sporočilo prejel, se pogosto lahko zgodi, da naprava ne prejme sporočila ali pa sporočilo ni preneseno pravilno zaradi izgube ali korupcije paketa.

Prav tako moramo paziti na napake, kjer naprava popolnoma izgubi povezavo z omrežjem. Razlogov za takšno stanje je lahko mnogo. Lahko pride do strojne okvare na napravi ali pa naprava preprosto izgubi povezavo z internetom. Takšne napake lahko ovirajo ali celo onemogočijo pravilno delovanje algoritma, če se zgodijo ob nepravem času.

3.2 Predpostavke

Algoritem zajema dve glavni predpostavki za svoje delovanje. Prva predpostavka zajema okolje delovanja algoritma. Okolje, v katerem deluje algoritem, je decentralizirano

porazdeljeno omrežje. V tem omrežju vse naprave uporabljajo enak algoritem za posredovanje sporočil. Vsa vozlišča so si enakovredna. V takšnem omrežju vsa vozlišča delujejo kot strežnik in odjemalec istočasno. To pomeni, da poljubno vozlišče v kateremkoli trenutku pošlje sporočilo poljubnemu drugemu vozlišču, dokler so poslana sporočila skladna s protokoli komunikacije v omrežju. To okolje je mreža, ki temelji na tehnologiji veriženja blokov.

Druga predpostavka je, da vsako vozlišče pozna vsa druga volišča v omrežju. Poljubno vozlišče ima v pomnilniku shranjeno polje vseh znanih aktivnih vozlišč v omrežju. Vsa vozlišča se med seboj sinhronizirajo. Tako se seznam udeleženih spreminja skladno z vstopanjem in izstopanjem vozlišč iz skupine aktivnih vozlišč. Ta vozlišča so v omrežju shranjena v blokih. Vsak vstop in izstop naprave med aktivna vozlišča je zapisana kot transakcija. Tako se vsa vozlišča med seboj sinhronizirajo preko pošiljanja blokov.

3.3 Cilji algoritma

Pri grajenju algoritma za pošiljanje sporočil smo si zadali več ciljev. Ti cilji nam pomagajo pri evalvaciji uspešnosti grajenja protokola in nam ponujajo načela, ki smo se jih držali. Imeli smo tri glavne cilje:

- Minimiziranje porabljenih virov
- Zagotavljanje čim hitrejšega informiranja celotnega omrežja
- Uspešno obravnavanje napak v omrežju

Prvi cilj se osredotoča na čim manjšo porabo virov naprav in omrežnih virov. To pomeni, da mora biti kompleksnost algoritma čim nižja pri računanju poti sporočila. Prav tako pa je zelo pomembno, da algoritem pošlje čim manj sporočil. Kot je predstavljeno v prejšnjih poglavjih, ima algoritem poplavljanja zelo enostavno strukturo in napravi ne vzame veliko komputacijskih virov, vendar po omrežju lahko pošlje ogromne količine sporočil, kar je zelo požrešno in lahko stane veliko časovne širine.

Drugi cilj se osredotoča na najhitrejšo mogočo dostavo sporočila od izvornega vozlišča do vseh drugih vozlišč. To lahko dosežemo zelo enostavno, če bi izvorno vozlišče poslalo sporočilo vsakemu drugemu vozlišču v omrežju. Ta pristop bi sicer optimiziral število korakov, ki so potrebni za dostavo sporočila, vendar bi bil za dano vozlišče izjemno drag. Potrebno je bilo poiskati rešitev, hitre dostave sporočil, ki se uspešno razširi na ogromna števila vozlišč in ne predstavlja dragih procesov.

Tretji cilja pa se navezuje na izzive, s katerimi se soočamo pri pošiljanju sporočil pri decentraliziranih omrežjih. Algoritem mora uspešno zaznati napake pri pošiljanju

sporočil, kot je nedosegljivost prejemnika. V primeru napake mora algoritem le-to tudi popraviti oziroma negirati.

3.4 Ideja algoritma

Algoritem izrablja dejstvo, da vsako vozlišče pozna naslove vseh ostalih vozlišč, ki so v danem trenutku aktivna na omrežju. Vsako vozlišče ob povezavi na omrežje zgradi skupino aktivnih vozlišč preko branja verige blokov. Ko je celotna veriga prenesena, je vozlišče sinhronizirano in spada med aktivna vozlišča. To pomeni, da začne sodelovati v konsenzu omrežja in mora prejeti vsa sporočila, ki se po mreži propagirajo. Ob prejemanju sporočil z novimi bloki se polje vozlišč, ki jih dano vozlišče pozna, osvežuje.

Algoritem temelji na gradnji vpetih dreves v grafu, ki predstavlja omrežje. Graf, ki predstavlja omrežje je polno povezan, saj vsako vozlišče pozna vsako drugo vozlišče. Kljub temu mora graf zaradi cilja o minimiziranju porabe virov zadostovati določenim potrebam. Oblika vpetega drevesa mora biti podobna strukturi binarnega drevesa v pomenu, da ima vsako vozlišče največ dva otroka in je uravnovešeno. To nam zagotavlja, da bo vsako vozlišče poslalo največ 2 sporočili. Prav tako bo sporočilo doseglo liste drevesa (zadnja neobveščena vozlišča) v največ $\log n$ posredovanjih sporočila, kjer je n število vseh vozlišč v omrežju.

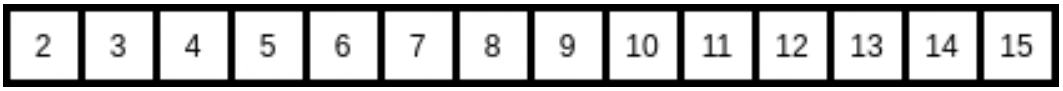
Če algoritem sestavi vpeto drevo za omrežje, to drevo zadostuje pogojem hitre dostave sporočila le za vozlišče, ki je v korenu drevesa. Če bi bil izvirnik sporočila drugo vozlišče in bi drevo "obesili"tako, da bi izvorno vozlišče predstavljalo koren, drevo ne bi bilo več uravnoteženo, prav tako pa bi imelo korensko vozlišče 3 otroke namesto 2. To pa ne izpolnjuje naših ciljev, tako da moramo za vsako vozlišče zgraditi novo drevo na način, da zadostuje pogojem.

Upoštevali smo, da bi dejstvo, da ima vsako vozlišče svoje vpeto drevo, potencialno predstavljalo visoko varnostno tveganje, saj bi napadalci lahko za poljubno sporočilo poznali pot pod pogojem, da vedo, katero vozlišče je sporočilo ustvarilo. To bi jim omogočilo prestrezanje sporočil ter napad na natančno določena vozlišča v namen blokiranja prejema sporočil tistemu vozlišču. Potencialno bi napadalci lahko to izrabili še na številne druge načine, vendar je to že izven področja te magistrske naloge. Za namen varnosti naš algoritem ustvari novo vpeto drevo za vsako vozlišče in za vsako sporočilo.

Vse te pogoje zadostimo z generacijo vpetih dreves s pomočjo uporabe tehnologije psevdo naključnosti, kjer uporabimo sporočilo in vozlišče v namen generacije unikatnega drevesa za pot sporočila.

3.5 Pošiljanje sporočila v omrežju

Kot omenjeno algoritem za vsako sporočilo natančno izračuna pot sporočila skozi omrežje. Vsako vozlišče v omrežju ima v pomnilniku shranjene naslove vseh ostalih vozlišč v omrežju, kot je prikazano na sliki 14.



Slika 14: Slika prikazuje polje vozlišč v pomnilniku vozlišča 1. Vozlišče pomni vsa ostala vozlišča, vendar na vsebuje sebe. V realni implementaciji so v polju shranjeni IP naslovi vozlišč, tu pa smo vozlišča označili z identifikacijskimi številkami za lažjo predstavo sheme.

Ko vozlišče ustvari ali prejme novo sporočilo mora najprej v polje števil vstaviti sebe in izvzeti vozlišče, ki je sporočilo ustvarilo. Tako ima vozlišče polje, v katerem so vsi prejemniki sporočila. Ko je polje urejeno, lahko polje premešamo. To mešanje bo omogočilo, da je za vsako sporočilo ustvarjena nova unikatna pot. Ker pa mora vsako vozlišče imeti sposobnost ustvariti enako drevo, moramo za dosego cilja uporabiti psevdo naključno mešanje polja. To pomeni, da bomo pri mešanju polja uporabili ključ. To nam zagotavlja, da lahko vsako vozlišče, ki ima enako polje in ima hkrati dostop do ključa, generira enako mešanje polja.

Zaradi varnostnih razlogov je pomembno, da ključ, ki generira dotično mešanje, ni splošno znan, temveč je znan le vozliščem, ki sporočilo pošiljajo. Tu lahko kot ključ uporabimo hash sporočila, ki ga dobimo preko uporabe razpršilne funkcije. Celotno sporočilo lahko vstavimo v razpršilno funkcijo. To pomeni, da vsa vozlišča, ki imajo enak ključ, imajo tudi dostop do veljavnega, neoskrunjenega sporočila.

Mešanje polja dosežemo, kot prikazuje slika 15. Vsak ključ proizvede drugačno mešanje (v kolikor je to mogoče v odvisnosti od velikosti polja), prav tako pa enak ključ vedno proizvede enako mešanje vozlišč.

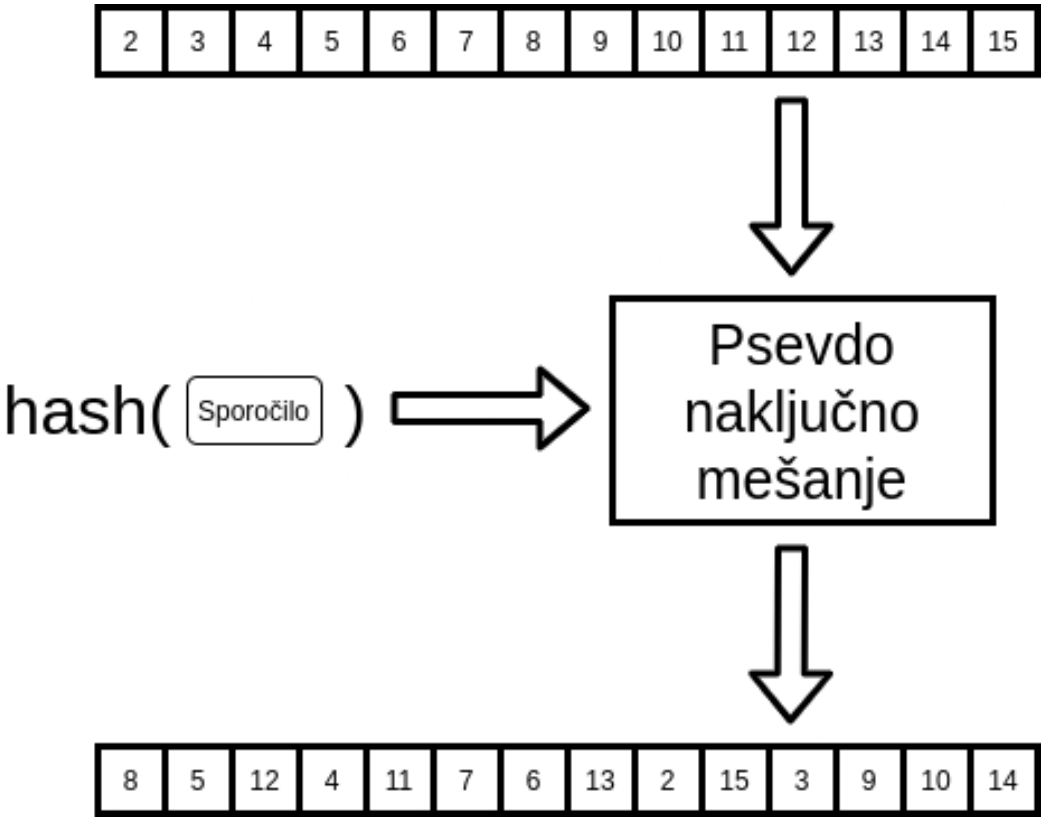
Ob prejemu sporočila tako vsako vozlišče izvede postopek mešanja polja prejemnikov, iz katerega bo zgrajena pot sporočila.

Iz ustvarjenega premešanega polja vozlišč lahko enostavno zgradimo uravnovešeno drevo. Za izgradnjo polja sledimo naslednjim korakom:

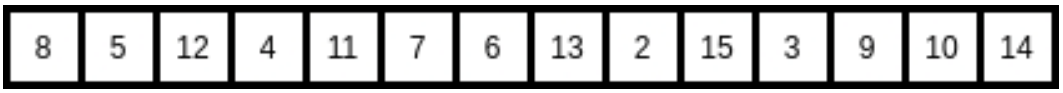
1. Vozlišče, ki je ustvarilo sporočilo, vrinemo na sredinski index polja

2. Sredinsko vozlišče vstavimo v drevo kot koren

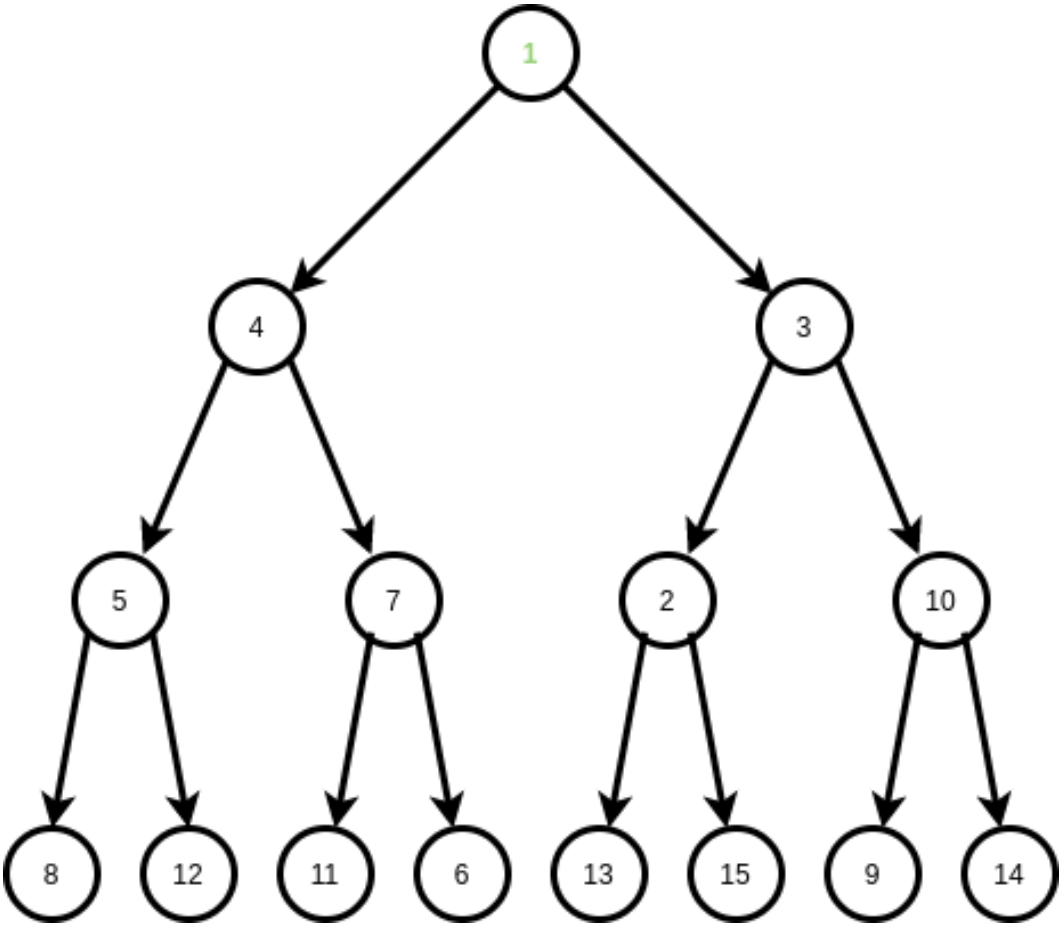
3. Rekurzivno za levo in desno polovico ponavljamo spodaj prikazana 2 koraka



Slika 15: Slika prikazuje mešanje polja vozlišč, ki predstavljajo prejemnike sporočila. Algoritem za psevdo naključno mešanje kot vhod vzame dva parametra. Prvi parameter je polje vozlišč, ki jih mešamo. Drugi argument je rezultat razpršilne funkcije sporočila ali hash. Algoritem nam vrne unikatno mešanje polja, ki ga dosežemo z uporabo ključa tega sporočila.



Slika 16: Slika prikazuje polje vozlišč, ki je bilo premešano z uporabo psevdo naključnega mešanja, kot je prikazano na sliki 15.



Slika 17: Slika prikazuje drevo, ki ga pridobimo z izvajanjem algoritma za pretvarjanje polj v uravnotežena drevesa. Kot vhodno polje, ki smo ga pretvarjali, smo vzeli polje prikazano na sliki 16. Vozlišče, ki je ustvarilo sporočilo je v korenu označeno z zeleno barvo.

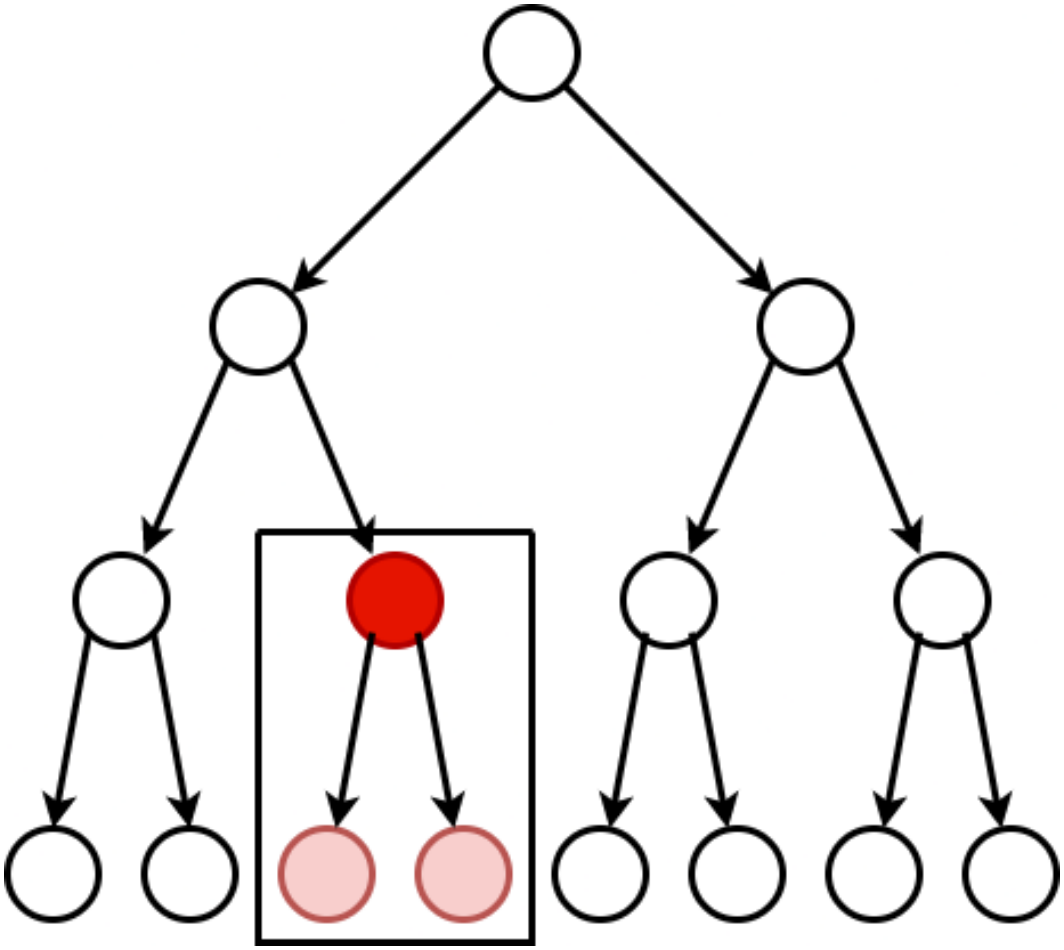
(a) Vzamemo sredino leve polovice in vstavimo kot levega otroka sredinskega elementa, vstavljenega v prejšnjem koraku

(b) Vzamemo sredino desne polovice in vstavimo kot desnega otroka sredinskega elementa, vstavljenega v prejšnjem koraku

Ko vozlišče ustvari drevo, lahko enostavno razbere pot sporočila. Vozlišče tako poišče svoja podrejena vozlišča in jima posreduje sporočilo.

3.6 Soočanje z napakami

Ko iz simulacije preidemo na produkcijska omrežja, se zelo pogosto soočamo z napakami, ki v simulaciji niso bile predvidene. To močno vpliva na delovanja različnih algoritmov, saj je skoraj nemogoče predvideti vsa stanja, v katerih lahko algoritem pristane. Kljub temu pa je naloga razvijalcev, da se pripravijo na kar se da veliko število

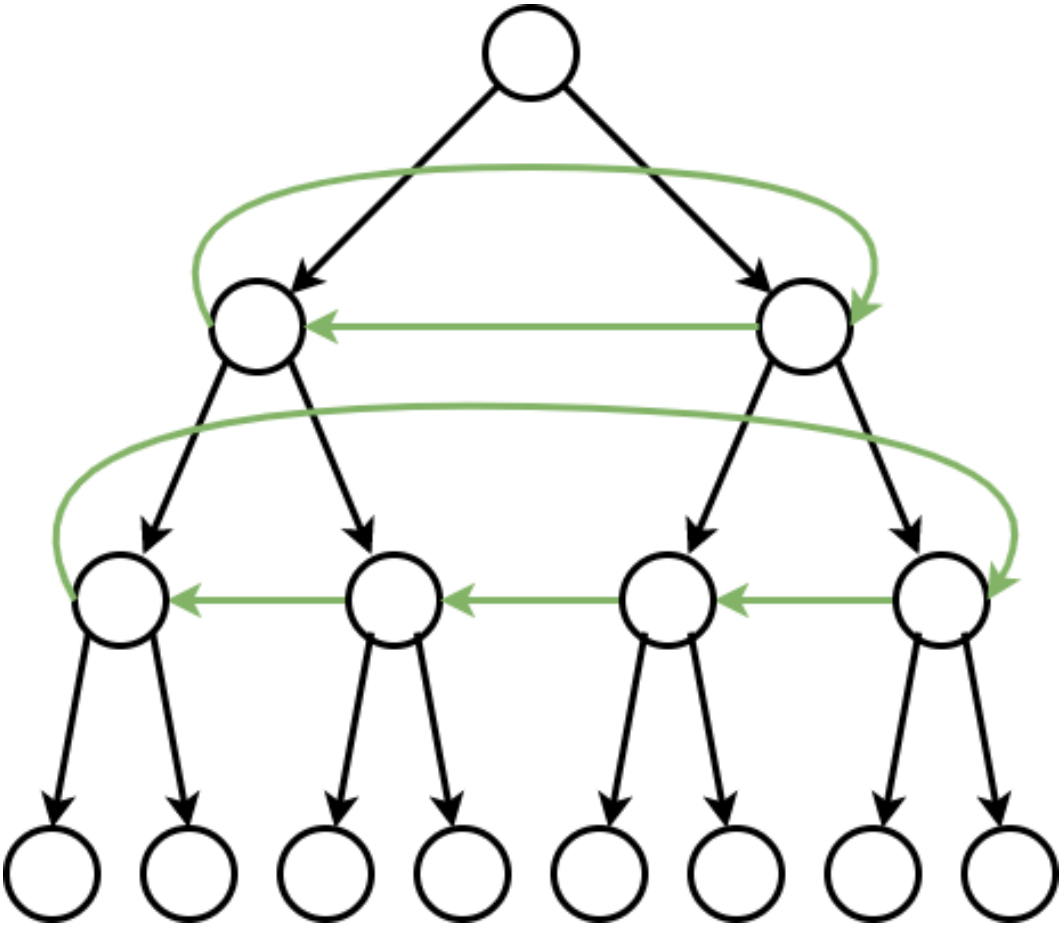


Slika 18: Slika prikazuje drevo, ki prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. V pravokotniku so z rdečo označena 3 neinformirana vozlišča. Pri temno rdečem vozlišču je prišlo do napake in je izgubilo povezavo z omrežjem. Ker je neodzivno, ni niti prejelo niti posredovalo sporočila svojima otrokoma, ki so obarvana svetlo rdeče.

različnih napak. Pri tem algoritmu smo se globlje osredotočili na potencialno izgubo povezave vozlišč z omrežjem.

Glavni problem, s katerim se algoritem sooča v primeru naključne prekinitve povezave vozlišča, ni neinformiranost tega vozlišča. Glavni problem je, da to vozlišče ne posreduje sporočil svojim otrokom v drevesu poti sporočila, kot je prikazano na sliki 18.

Preden se lotimo popravljanja napak, potrebuje algoritem najprej sposobnosti preverjanja samega obstoja napak. To v algoritmu naredimo tako, da vsakemu vozlišče dodamo nalogo preverjanja sosednjih vozlišč. Po prejemu sporočila od svojega starša vozlišče pošlje sporočilo svojim otrokom. Potem isto sporočilo posreduje svojemu sosedu, seveda v primeru, da ga ta ni prejel. Pomembno je, da sporočilo pošljemo, saj obstaja verjetnost, da se sosednje vozlišče odziva in je prišlo do napake pri komunikaciji



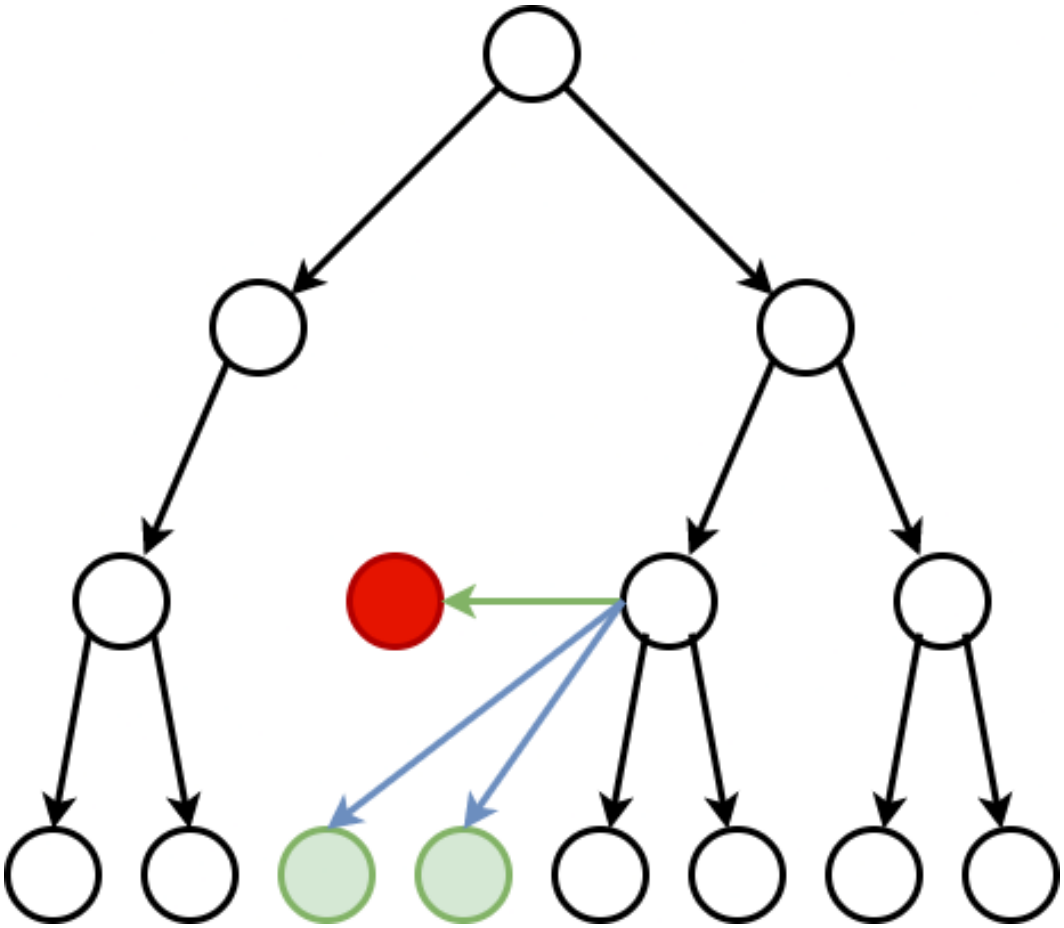
Slika 19: Slika prikazuje drevo, ki prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. Z zelenimi puščicami so označena sporočila, ki jih vozlišča pošljejo svojim sosedom v namen pregleda odzivnosti soseda. Enako preverjanje se dogaja na vseh nivojih drevesa. Na tej sliki na najnižjem nivoju preverjanje ni prikazano zaradi razumljivosti sheme.

med njim in njegovim staršem.

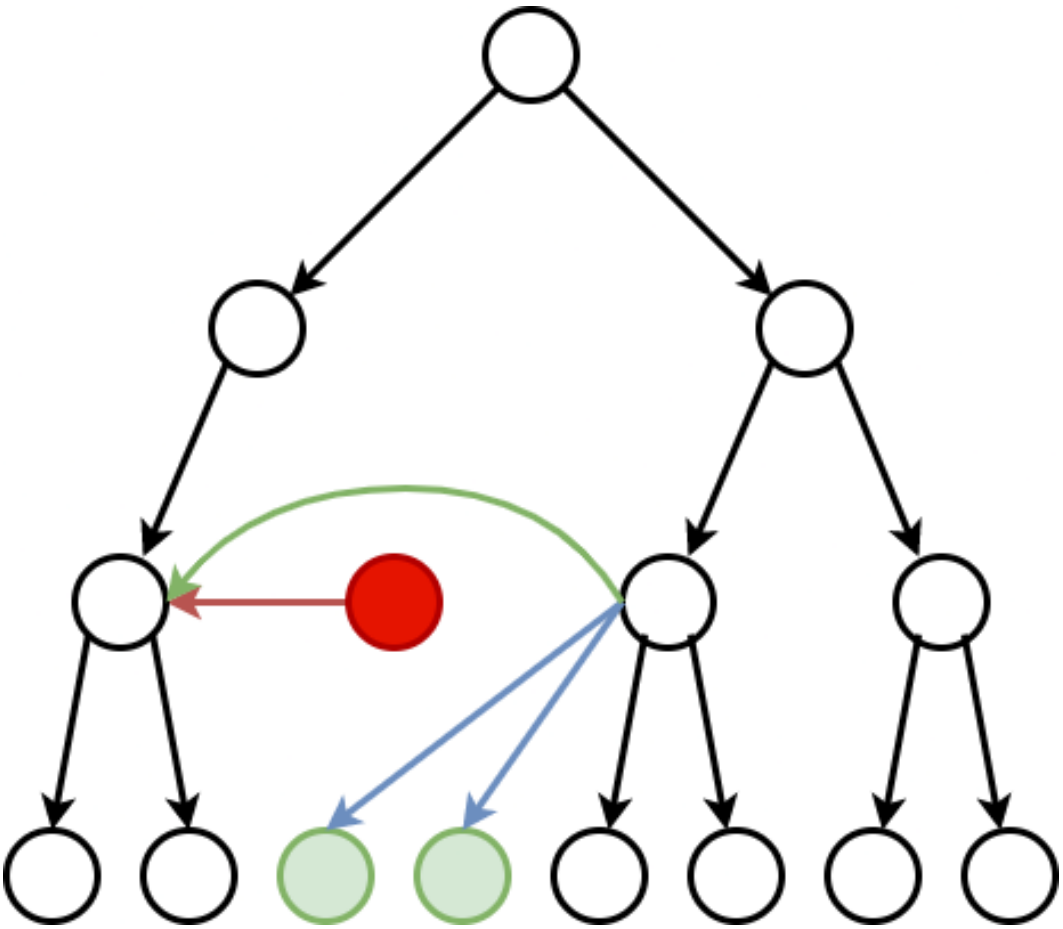
V primeru neodzivnosti sosednjega vozlišča mora aktivno vozlišče informirati otroke neodzivnega vozlišča. Tako bodo vozlišča prejela podatek in sporočilo bo naseljevalo svojo pot, kot je prikazano na sliki 20.

Tako je vozlišče, ki je zaznalo napako, popravilo napako informiranja neodzivnega sporočila. To pa ni dovolj, saj lahko da je vozlišče, ki je sosed neaktivnega vozlišča, prav tako neodzivno. Tega vozlišča ni preverilo nobeno drugo vozlišče, saj je vozlišče, ki je bilo zanj zadolženo, neodzivno. To nalogo tako kot informiranje otrok prenesemo na vozlišče, ki zazna neodzivno vozlišče, kot je prikazano na sliki 21

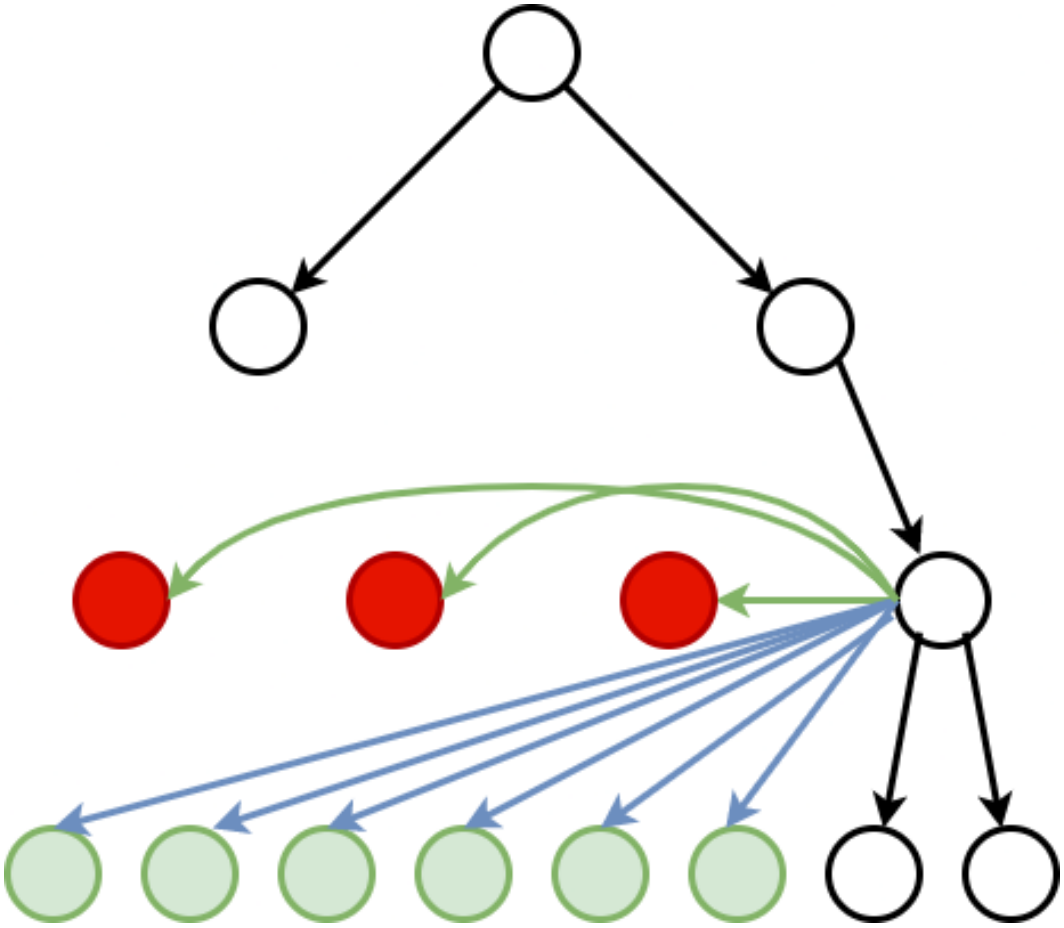
Hitro lahko vidimo, da v primeru mnogih napak lahko eno vozlišče opravi kar veliko količino dela, kot je prikazano na sliki 22. To sicer ni optimalno, vendar zagotavlja, da se algoritem ne ustavi, dokler vsaj eno vozlišče v vsakem nivoju deluje. Verjetnost da pride do takšnih napak je majhna. Malo verjetno je že, da pride do neodzivnosti



Slika 20: Slika prikazuje drevo, ki prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. Ko vozlišče preveri sosednje vozlišče (rdeče vozlišče) in opazi, da je neodzivno, pošlje sporočilo njegovim otrokom, da se veriga ne prekine (modre puščice).



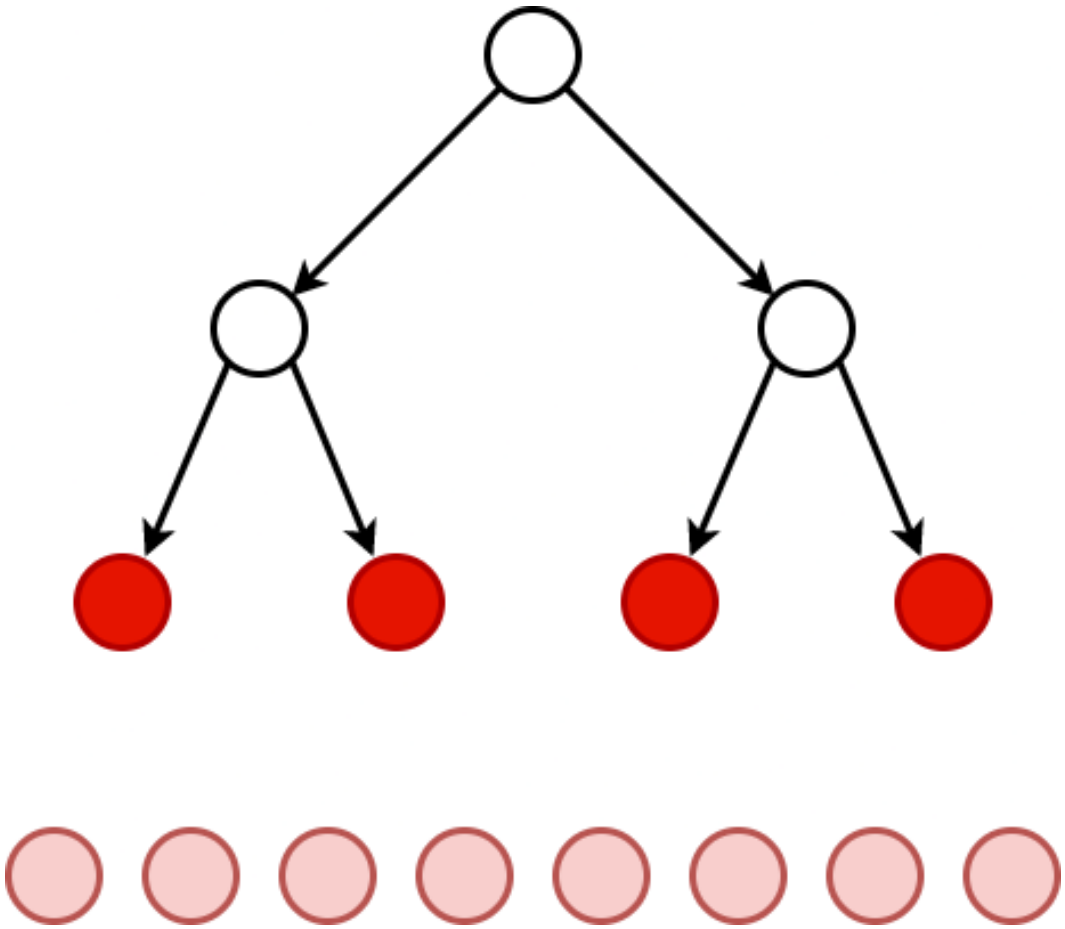
Slika 21: Diagram prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. Ko vozlišče preveri sosednje vozlišče (rdeče vozlišče) in opazi, da je neodzivno, pošlje sporočilo njegovim otrokom, da se veriga ne prekine (modre puščice). Na koncu preveri še soseda svojega soseda, saj zaradi neodzivnosti rdečega vozlišča ni bil preverjen.



Slika 22: Diagram prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. Vozlišče zaporedno obvesti otroke soseda (modre puščice) in preverja sosede neodzivnih vozlišč (zelene puščice).

večjega števila vozlišč še preden se podatek o tem prenese skozi celotno omrežje, da bi ga vozlišča izvzela iz polja prejemnikov. Še manj pa je verjetno, da so vsa neaktivna vozlišča v zaporedju zmešana v isti nivo.

Vidimo, da kljub izjemno nizki verjetnosti obstaja možnost terminalne napake algoritma. V primeru, da na nekem nivoju povezavo izgubijo vsa vozlišča, se algoritem ustavi in vozlišča v nižjih nivojih ostanejo neinformirana, kot je prikazano na sliki 23.



Slika 23: Slika prikazuje drevo, ki prikazuje pot sporočila pri informiranju vseh vozlišč v omrežju. Shema prikazuje terminalno napako. Na tretjem nivoju ni odzivno nobeno vozlišče (temno rdeče), zato se algoritem ustavi. Vozlišča v zadnjem nivoju ostanejo neinformirana (svetlo rdeče).

4 Implementacija

V poglavju je predstavljena implementacija predstavljenega algoritma. Algoritem je implementiran v simulaciji. To pomeni, da algoritem ne deluje na večih napravah, temveč se izvaja na eni napravi. Naprava simulira delovanje omrežja in poganja več vozlišč, ki med seboj komunicirajo na način, kakršnega bi imeli v realnem porazdeljenem sistemu. Vsako vozlišče se izvaja v svoji niti.

Takšna implementacija nam omogoča nekaj prednosti. Prva prednost je skupnost podatkov na enem mestu, zato je vizualizacija omrežja veliko bolj enostavna. V ta namen smo poleg simulacije implementirali tudi grafični vmesnik, preko katerega lahko spremljamo obratovanje algoritma. Druga večja prednost implementacije pa je, da imamo popolno kontrolo nad sistemom, kar pomeni, da lahko okolje sistema manipuliramo tako, da vidimo, kako se algoritem obnaša v različnih pogojih.

4.1 Uporabljene tehnologije

Za implementacijo smo uporabili jezik Java. Različica jezika je Java 11. JDK 16 je odprta kodna referenčna izvedba različice 16 platforme Java SE, kot jo določa JSR 390 v postopku skupnosti Java [22]. Java poganja zaledni sistem aplikacije, kar je sama simulacija okolja in algoritma. Prav tako pa s pomočjo Javalin ogrodja aplikacija deluje kot strežnik, preko katerega serviramo grafični vmesnik za vizualizacijo simulacije. Vmesnik je napisan v jezikih HTML, CSS in JavaScript.

Implementacija uporablja nekaj zunanjih knjižnic, ki so nam pri implementaciji olajšale delo. Pri grafičnem vmesniku uporabljamo VivaGraphJS knjižnico, ki omogoča enostavnejše načine vizualizacije grafov za prikaz našega omrežja. V zalednem sistemu uporabljamo že prej omenjeno ogrodje Javalin in Commons-codec, ki ga je razvila Apache Software Foundation. Le-tega uporabljamo za enostaven dostop do algoritma SHA256. To je glavni algoritem, ki ga naš sistem uporablja za razprševanje podatkov.

Za komunikacijo med zalednim sistemom in grafičnim vmesnikom uporabljamo tehnologijo odprtih priključkov (web socket), preko katere imata sistema odprto povezavo. Preko tega komunikacijskega kanala si pošiljata sporočila, kjer pa za prevajanje podatkov iz JSON formata v Java razrede (in obratno) uporabljamo knjižnico Gson, ki jo je v ta namen razvil Google.

Tabela 3: Tabela prikazuje atribute vozlišča v simulaciji

Ime atributa	Tip podatka	Opis
ID	String	Identifikator vozlišča. V realnem omrežju vsako vozlišče predstavlja javni ključ za kriptiranje sporočil.
Obveščeno	boolean	Je atribut, ki označuje vozlišče kot informirano ali neinformirano. Vozlišče je informirano, ko prejme sporočilo. V realnem svetu te podatke predstavlja veriga blokov.
Aktivno	boolean	V omrežju veriženja bloka ta podatek ne obstaja, vendar v simulaciji predstavlja stanje vozlišča. V primeru, da simuliramo, da je vozlišče nedosegljivo, je atribut postavljen na 1, drugače na 0.
Vozlišča	polje vozlišč	To polje vsebuje druga vozlišča v omrežju, ki jih vozlišče pozna.

4.2 Vozlišče

V aplikaciji je razred vozlišča glavni akter v okolju. Vozlišče predstavlja napravo, ki bi v realnem svetu poganjala sistem veriženja blokov in bi si z drugimi napravami izmenjevala podatke. Vsaka naprava o sebi hrani nekaj podatkov. Te atributi omogočajo vozlišču opravljanje algoritma. Vozlišča v mreži veriženja blokov hranijo več podatkov, kot jih vozlišča v simulaciji, vendar so podatki vezani na druge operacije, ki se dogajajo v omrežju. Podatki, ki jih hranijo vozlišča v simulaciji, se zaradi dokaza koncepta navezujejo le na pošiljanje podatkov. Podrobneje so predstavljeni v tabeli 3.

4.3 Sporočilo

Sporočilo je ovojnica za prenos podatkov v omrežju. Sporočilo je sestavljeno iz 2 glavnih kosov: glava in telo. V telesu sporočila so shranjeni vsi podatki, ki jih pošiljatelj želi poslati prejemniku. V glavi pa so shranjeni meta podatki sporočila, katere uporabljajo algoritmi, za uspešno navigacijo sporočila. Podatki nam niso pomembni, saj direktno ne vplivajo na delovanje našega algoritma. V algoritmu tako polje telesa puščamo prazno, v glavi pa pošiljamo podatke, ki jih potrebujemo za delovanje, kot je prikazano v tabeli 4.

4.4 Omrežje

Za potrebe te naloge omrežje predstavlja kar se da verjetno sliko dejanskega porazdeljenega decentraliziranega omrežja. Omrežje lahko predstavimo z usmerjenim acikličnim

Tabela 4: Tabela prikazuje podatke v glavi sporočila

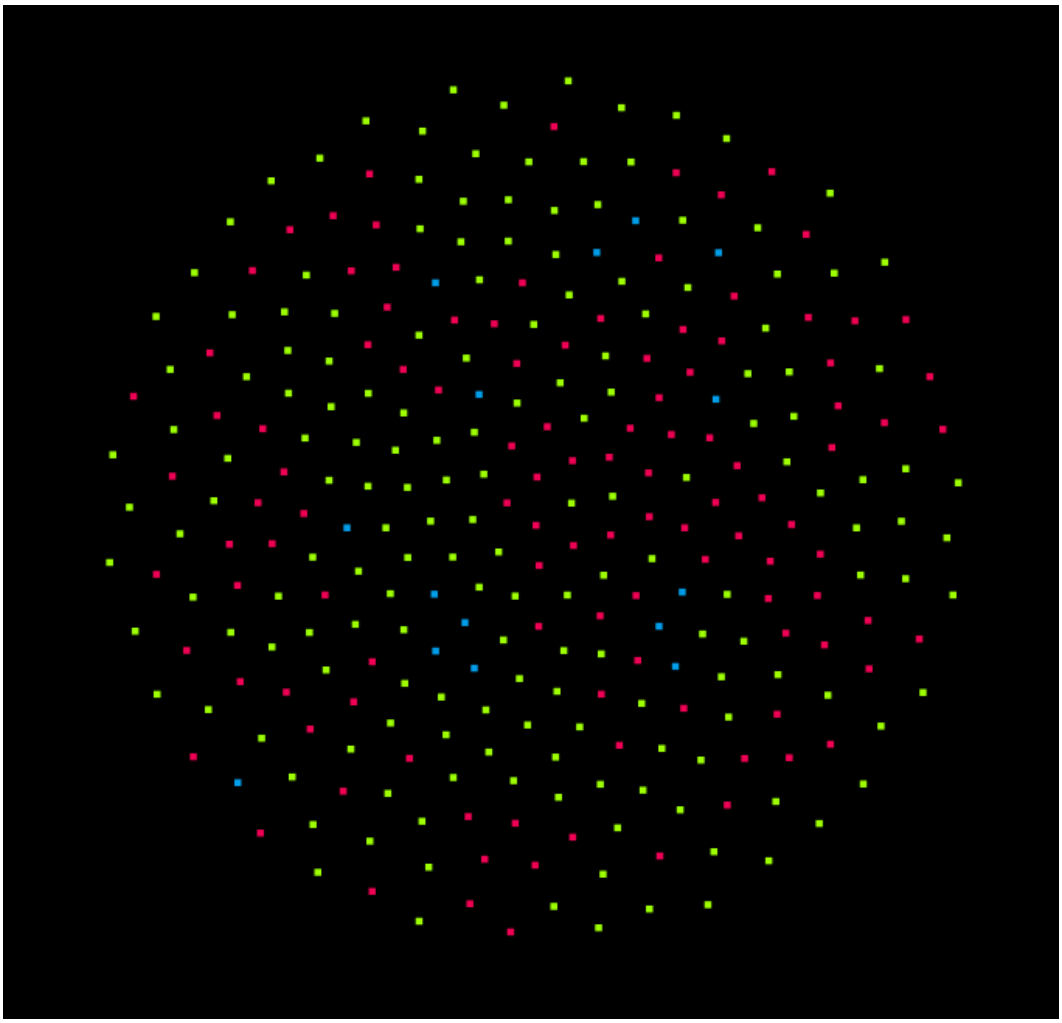
Ime podatka	Tip podatka	Opis
ID	String	Unikatni identifikator sporočila, ki je generiran ob stvaritvi sporočila.
ID stvarnika	String	Identifikator vozlišča, ki je sporočilo generiralo. V realnem omrežju vsako vozlišče predstavlja javni ključ za kriptiranje sporočil.
Pot	zaporedje bitov	Pot sporočila od stvaritelja do trenutnega prejemnika. V binarnem drevesu 0 pomeni v drevesu spust k levemu otroku in 1 k desnemu otroku. Več o sistemu je razloženo v prihodnjih poglavjih.
Tip	DATA / ACK	Polje predstavlja tip sporočila. Vrednost DATA pomeni, da je sporočilo navadno sporočilo, ki prenaša podatke v telesu. Vrednost ACK pa pomeni, da je telo prazno in da je namen sporočila le obveščanje o aktivnosti volišča.
Preskoki	število	V polju so šteta posredovanja sporočila v omrežju. Podatek je hranjen v namen statistike.

grafom. Sestavljeno je iz večih vozlišč in usmerjenih povezav (komunikacijskih kanalov). Vozlišča preko komunikacijskih kanalov pošiljajo sporočila. Simulacija omrežja poteka v ciklih. V vsakemu ciklu vsa vozlišča začnejo kot neobveščena vozlišča. V neki točki naključno vozlišče generira sporočilo, naloga algoritma pa je, da to sporočilo čim hitreje pošljejo vsem vozliščem. Ob prejemu sporočila vozlišče postane informirano. Cikel se konča, ko so vsa aktivna vozlišča obveščena. V primeru, da algoritmu ne uspe informirati vseh vozlišč je vsak cikel časovno omejen na eno minuto. Po minuti se cikel uspešno ali neuspešno konča in se začne nov.

Vsako vozlišče v omrežju je lahko v enem od treh stanj:

- Obveščeno
- Neobveščeno
- Neodzivno

V algoritmu se vsako vozlišče glede na stanje obnaša drugače. Obveščena vozlišča pošiljajo sporočila neobveščenim vozliščem, medtem ko neobveščena vozlišča ne počnejo nič, saj se v resnici ne zavedajo, da sporočilo obstaja, dokler jih ne doseže. Neodzivna vozlišča simulirajo realno okolje, kjer lahko naključna naprava izgubi povezavo z omrežjem. Vozlišča ne vedo, katera bodo neodzivna, zato mora biti algoritem pripravljen na njihovo obravnavanje.



Slika 24: Slika prikazuje omrežje. V omrežju so točke, ki predstavljajo naprave v omrežju. Barva naprave označuje njeno stanje. Zelene točke prikazujejo obveščena vozlišča, rdeče neodzivna vozlišče in modre vozlišča, ki sporočila še niso prejela.

4.5 Latenca

V realnem omrežju vsako pošiljanje vsakega sporočila potrebuje različne vire. Pri veliki količini sporočil je posebej pomembna poraba časovne širine, saj je ta lahko zelo omejena. Potencialno vsako vozlišče lahko pošilja velike količine podatkov. Da pa sporočilo prejme prejemnika, pa se to ne zgodi takoj, vendar traja nekaj časa. Temu pojavu pravimo latenca ali časovni zamik. To simuliramo tudi v omrežju. Med vsemi vozlišči je zamik in zamik ni med vsemi enak. Vedno pa imata dve vozlišči enak zamik ne glede na smer potovanja sporočila. To pomeni, da če vozlišče A pošlje sporočilo vozlišču B, bo potovalo enako količino časa, kot če vozlišče B pošlje sporočilo vozlišču A.

Če bi v simulaciji vnaprej generirali vrednosti latence, bi bil to zelo zahteven proces, ker imamo lahko ogromno število vozlišč. To bi pomenilo, da bi potrebovali tabelo, ki shrani vrednosti latenc za vsako vozlišče z vsakim drugim vozliščem, kar pa bi lahko zasedlo ogromno količino pomnilnika.

Problema smo se zopet lotili z uporabo psevdo naključnosti. Za vsak par vozlišč smo generirali latenco v realnem času s pomočjo razpršilne funkcije in identifikatorjev vozlišč. To poteka v 4 korakih:

1. Razporedi ključa
2. S pomočjo razpršilne funkcije generiraj ključ, ki je unikaten paru
3. Pretvori ključ v seme za prevdo naključno funkcijo
4. V rangu od minimalne do maksimalne latence s semenom generiraj latenco

Vsako vozlišče ima svoj identifikator, ki je 32-mestno šestnajstiško število. Identifikatorja obeh vozlišč uporabimo za generacijo ključa. Pomembno je, da identifikatorja vozlišč v hash funkcijo vedno vnesemo v istem zaporedju, saj bi menjava zaporedja producirala različne rezultate. Ker je identifikator v šestnajstiškem sistemu, lahko identifikatorja razvrstimo po velikosti. Tako bosta poljubna ključa vedno vnesena v enakem zaporedju.

Ko sta ključa razvrščena, jih konkatiniramo in vstavimo v razpršilno funkcijo, ki nam vrne novo 32-bitno šestnajstiško število. To predstavlja hash, ki je unikaten paru vozlišč.

Ker v naslednjem koraku psevdo naključna funkcija kot seme potrebuje 64-bitno število, moramo ključ najprej pretvoriti v primeren format. To je enostaven proces, vendar moramo biti pozorni na dolžine števil. Ena števka v šestnajstiškem sistemu je v binarnem sistemu predstavljena s štirimi biti. To pomeni, da bo količina bitov potrebnih za zapis števila štirikrat večja. Če želimo 64-bitno število, to pomeni, da za

pretvorbo potrebujemo največ 16 znakov dolgo šestnajstiško število. V naši implementaciji smo tako vstavili prvih 16 znakov ključa.

```
long hashToSeed(String hash) {
    return Long.parseLong(hash.substring(0, 15).trim(), 16);
}
```

V zadnjem koraku tako pridobljeno seme vstavimo v generatorja naključnih števil in generiramo vrednost latence, kjer imamo dano minimalno in maksimalno vrednost latence.

```
int calcLatencyBetweenNodes(String id1, String id2) {
    long seed = hashToSeed(hash(mergeId(id1, id2)));
    r.setSeed(seed);
    return r.nextInt(MAX_LATENCY - MIN_LATENCY) + MIN_LATENCY;
}
```

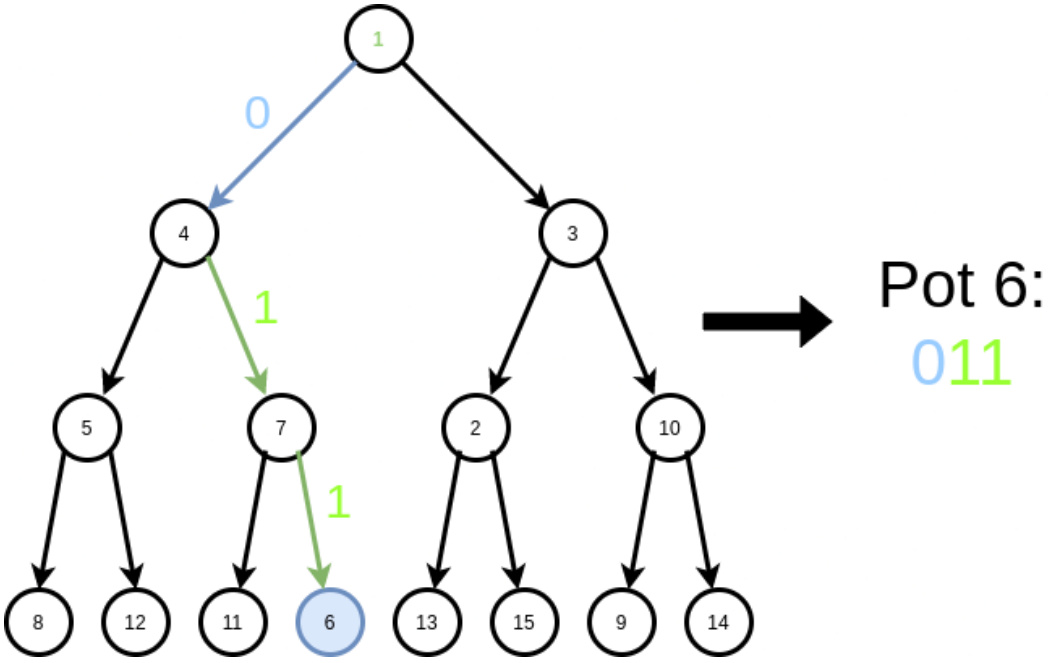
4.6 Računanje poti

Glavni problem, s katerim se soočamo z algoritmom, je določanje prejemnikov sporočila. Kot predstavljeno to deluje na podlagi psevdo naključnosti. Ob prejemu sporočila vsako vozlišče opravi tri glavne korake:

1. Psevdo naključno generiranje polja prejemnikov
2. Določanje svoje pozicije v drevesu in pozicije otrok
3. Določanje pozicije sosednjega vozlišča v drevesu

Ko vozlišče prejme sporočilo, najprej pripravi polje potencialnih prejemnikov. Vsako vozlišče ima polje vseh vozlišč v sistemu. Naloga vozlišča je generirati enako pot sporočila, kot ga je generiralo vozlišče, ki je sporočilo ustvarilo. Najprej iz polja vseh vozlišč vozlišče odstrani izvornika sporočila. Nato identifikatorja sporočila spremeni v seme in ga vstavi v generatorja naključnih števil. Nato s pomočjo izvornika psevdo naključno premeša polje.

```
List pseudoRandomMix(List vozlisca, long seed) {
    Random r = new Random();
    r.setSeed(seed);
    for (int i = vozlisca.size() - 1; i > 0; i--) {
        int index = r.nextInt(i + 1);
        zamenjaj(vozlisca, i, index);
    }
```



Slika 25: Slika prikazuje diagram zapisa poti sporočila po omrežju. Pot sporočila je linearna in celotno omrežje lahko predstavimo z usmerjenim binarnim drevesom. Vsako vozlišče mora obvestiti do dva otroka, zato lahko enega označimo z 0 in drugega z 1. Pot tako zapišemo kot zaporedje vozlišč, skozi katera je sporočilo prešlo.

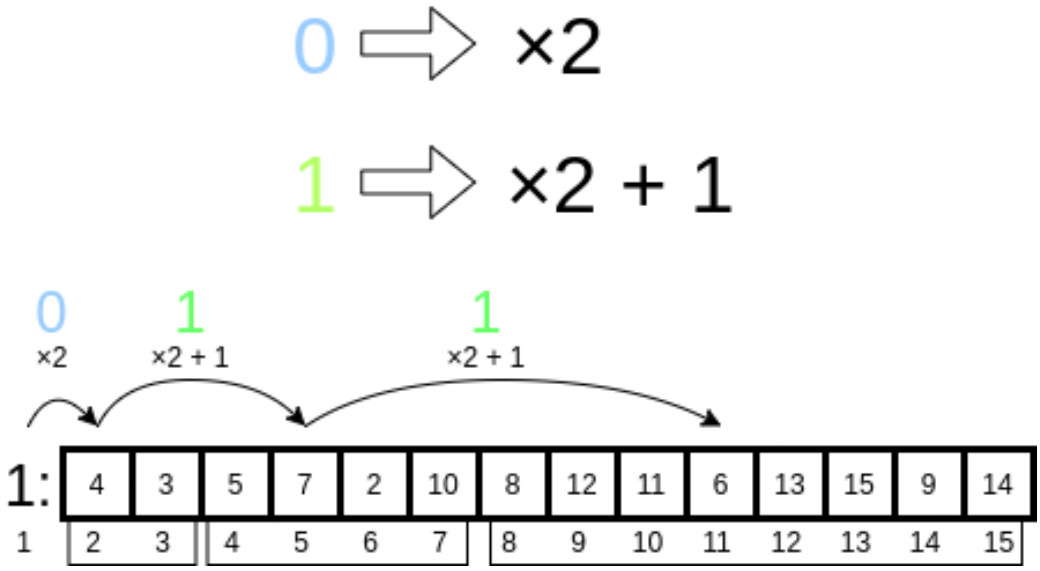
```
    return vozlisca;
}
```

Ko ima vozlišče pravilno premešano polje prejemnikov, lahko preko njega določi pot sporočila. Kot je razloženo v teoretični razlagi algoritma, pot sporočila predstavlja drevesna struktura. Ker pa bi samo konstrukcija drevesa potencialno zahtevala preveč virov (izgradnja in iskanje svoje pozicije v najslabšem primeru O(n)), moramo to nalogo opraviti drugače. Tu nastopi atribut sporočila, ki zapisuje pot sporočila do vozlišča. Oblika zapisa poti je v sekvenci bitov, ki predstavljajo pot sporočila, kjer je pomen 0 sprehod do levega otroka in 1 sprehod do desnega otroka, kot je prikazano na sliki **??**.

Ker pa v vozlišču ne zgradimo dejanskega drevesa, moramo opraviti enako delo s samim poljem. To lahko dosežemo z izrabo dejstva, da se v vsakem nivoju podvoji število vozlišč.

Kot je prikazano na sliki 26 predpostavljamo, da je koren število 1 in potem z enostavnim množenjem lahko izračunamo svojo pozicijo v drevesu. Ko imamo svojo pozicijo, z istim algoritmom izračunamo svoje otroke (pomnožimo z 2 in dobimo indeks prvega otroka, kateremu prištejemo še 1 in dobimo še drugega otroka).

V tretjem koraku pa moramo določiti pozicijo sosednjega vozlišča v v drevesu. To vozlišče moramo preveriti, v kolikor je njegov starš neodziven in ni prejel sporočila. Ker imamo niz bitov poti do trenutnega vozlišča, lahko ta niz pretvorimo v niz do



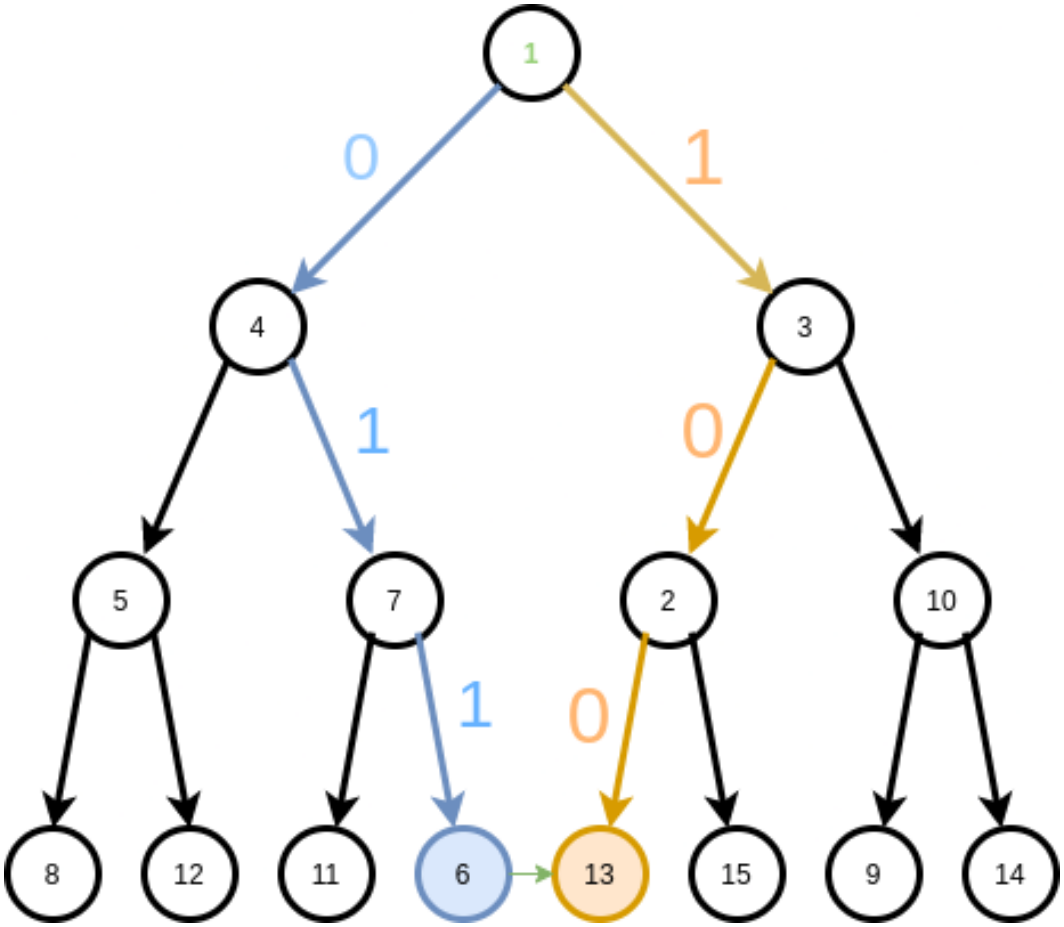
Slika 26: Slika prikazuje diagram izračuna poti sporočila v omrežju. Koren drevesa se začne s številom 1. Potem lahko preko bitov binarnega zapisa poti sporočila izračunamo indekse naslednjih elementov. Indeks zgolj pomnožimo z 2 in mu prištejemo bit poti (0 ali 1).

sosednjega vozlišča.

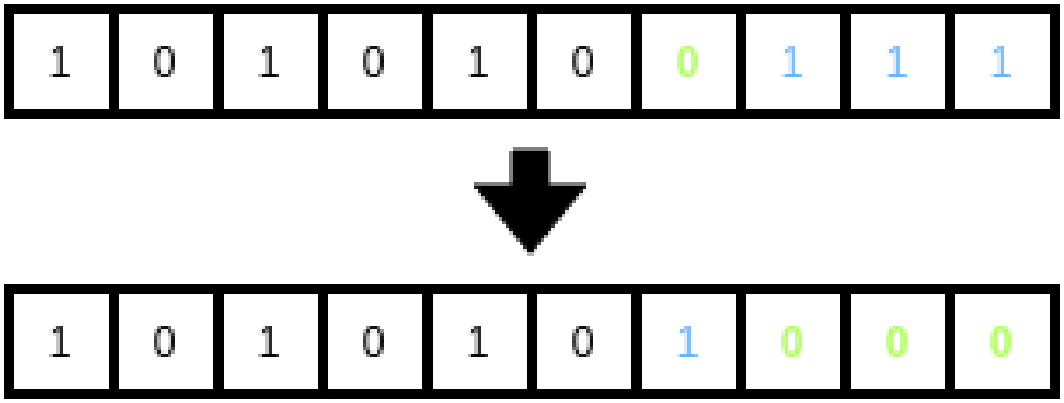
Kot je razvidno iz slike 27 imata pot vozlišča in pot soseda obratni strukturi. To velja za vsa sosednja vozlišča, ampak le od skupnega starša naprej, vključno z njim. Če želimo narediti transformacijo poti, kot je prikazana na sliki 28, se držimo sledečega algoritma:

- Iz desne proti levi preverjamo vsak bit poti vozlišča
- Dokler ne dosežemo prve 0, spreminjamo enice v ničle
- Ko dosežemo prvo 0, jo spremenimo v enico in končamo proces

```
String retracePathOneNodeToTheRight(String path) {
    for (int i = path.length() - 1 ; i >= 0 ; i--) {
        if (path.charAt(i) == '0') {
            path.setCharAt(i, '1');
            break;
        }
        path.setCharAt(i, '0');
    }
    return path;
}
```



Slika 27: Slika prikazuje diagram poti do trenutnega aktivnega vozlišča 6 (modra). Prikazuje tudi pot do soseda modrega vozlišča, vozlišča 13 (oranžna). Vidimo, da sta poti od razcepa skupnega starša (koren) do vozlišča (011) in njegovega soseda (100), obratni.



Slika 28: Slika prikazuje diagram transformacije poti vozlišča, za katerega računamo pot do njegovega soseda. Zadnji biti se do prve ničle (vključno z ničlo) pretvorijo v obraten bit.

V primeru, da vozlišče prejme sporočilo od svojega soseda, prejem sporočila zahteva potrditev, saj se v nasprotnem primeru vozlišče šteje kot neodzivno. Kot potrditev vozlišče v odgovor pošlje prazno sporočilo, ki ima v glavi zapisan tip ACK. Ko vozlišče pošlje sporočilo sosedu na potrditev, čaka 1.3 sekunde, po tem času pa ravna s sosedom kot neodzivnim. To pomeni, da enak proces pošiljanja sporočila ponovi, le da kot svojo pot uporablja pot do soseda. To pomeni, da prevzame naloge soseda, obvesti njegove otroke in preveri njegovega soseda.

5 Testiranje

V poglavju testiranje je opisan proces evalvacije delovanja predstavljenega algoritma. Opravili smo več testov z različnimi parametri, ki nam omogočajo zanesljiv vpogled v kvaliteto algoritma.

Vsako sporočilo, ki je poslano med poljubnima vozliščema, ima latenco. Najmanjša mogoča latenca je 300ms in največja latenca je 500ms. Testi so potekali na štirih različnih velikostih omrežja. Testirane velikosti so 100, 500, 1000 in 2000 vozlišč.

Za primerjavo je bil implementiran referenčni algoritem poplavljanja, ki je predstavljen v razdelku 5.1. Vsi podatki so primerjani s to referenco. Algoritem poplavljanja je opravljal vse teste z enakimi parametri kot naš algoritem. Kombinacijo variabilnih spremenljivk, smo testirali dvajsetkrat za vsak algoritem.

5.1 Algoritem poplavljanja

Algoritem poplavljanja (flooding algorithm) ob vsakem prejemu sporočila posreduje to sporočilo sosedom. Ker je ob višjem številu poznanih sosedov verjetnost, da je nekega soseda že obvestilo neko drugo vozlišče, ne pošiljamo sporočila vsem poznanim vozliščem. Za algoritem poplavljanja je pomembno vedeti, da je bila minimalna stopnja vsakega vozlišča v omrežju nastavljena na 7. Parameter fan-out, ki nam pove količino naključnih sosedov, ki prejmejo sporočilo od vozlišča, pa je nastavljeno na 5 za namen zmanjševanja porabe pasovne širine in manjšega števila poslanih vseh sporočil.

5.2 Delovanje v primeru napak

Glavna prepreka pri delovanju, s katerim se algoritem sooča pri informiranju, je neodzivnost vozlišč. Pri inicializaciji omrežja v simulaciji aplikacija naključna vozlišča označi kot neaktivna. Neaktivno vozlišče ne odgovarja na sporočila in jih ne posreduje drugim vozliščem. Aktivna vozlišča ob začetku testa ne vedo, katera vozlišča so neaktivna in se morajo soočati z njimi tekom testa.

```
for (Node n : network) {
    if (DISCONNECT_ODDS > r.nextFloat()) {
        n.deactivate();
    }
```

}

Število neaktivnih vozlišč je odvisno od spremenljivke DISCONNECT_ODDS. Ta vrednost predstavlja verjetnost, da se vozlišče deaktivira. To pomeni, da pri več testih z enako verjetnostjo ne bomo imeli identičnega števila neaktivnih vozlišč, ampak bo le-to rahlo odstopalo.

Testirali smo različne vrednosti verjetnosti, da smo videli, kako se na to odzivata algoritma. Testirane vrednosti so 0%, 5%, 10%, 25%, 50% in 75%. V realnem omrežju je ta verjetnost nizka, vendar smo testirali tudi visoke verjetnosti (25%, 50% in 75%) v namen testiranja algoritma v ekstremnih situacijah.

5.3 Hitrost algoritma

Simulacija se začne, ko naključno vozlišče generira sporočilo. Konča se lahko uspešno ali neuspešno. Uspešno se simulacija konča, ko vsa aktivna vozlišča prejmejo sporočilo in postanejo informirana. Aktivna vozlišča so vozlišča, ki imajo vzpostavljeno povezavo z omrežjem. Prav tako pa se test lahko konča neuspešno, kar pomeni, da niso bila obveščena vsa vozlišča. V tem primeru je bil test ustavljen zaradi pretečenega časa. Čas, ki je dovoljen, da algoritem izpolni svojo nalogo, je 60 sekund.

Končni čas testa se šteje od prvega poslanega sporočila do prejetja zadnjega poslanega sporočila. Tudi če se algoritem neuspešno ustavi, čas štejemo do zadnjega sporočila in ne 60 sekund. Za preverjanje uspešnosti algoritma štejemo vsa informirana vozlišča v končnem stanju.

5.4 Poraba virov

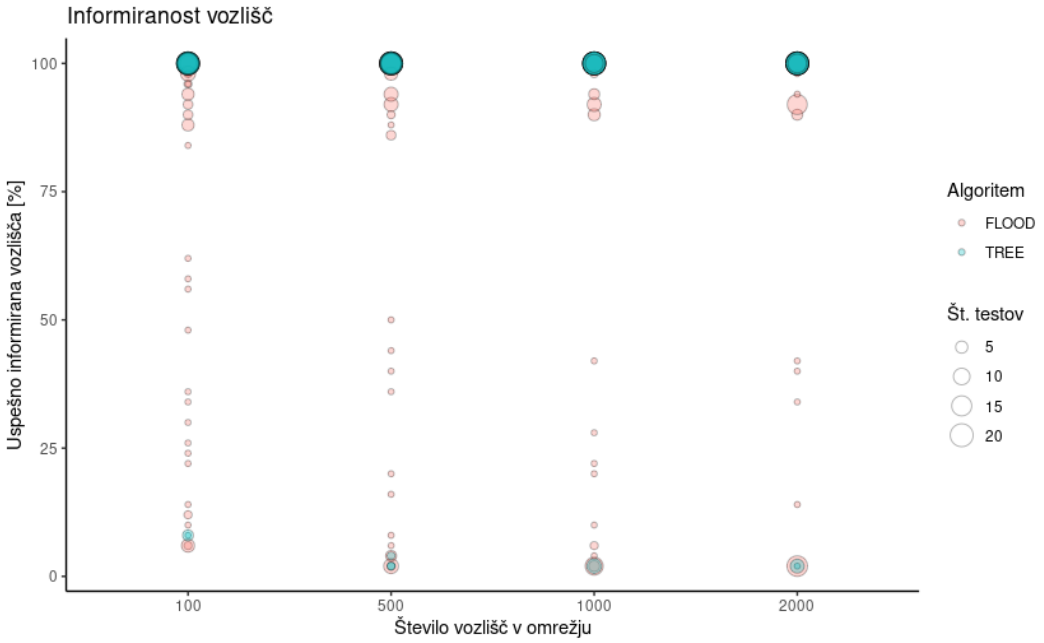
Pri porabi virov smo spremljali 3 parametre. To je štetje skupnih sporočil in število preskokov sporočila, ki je zapisano v glavi sporočila. Glavni parameter je skupno število poslanih sporočil v algoritmu. Ta parameter najbolj vpliva na porabo virov, saj predstavlja skupno porabo virov celotnega omrežja. Poleg tega pa smo spremljali tudi pot sporočila. Tu smo ločili povprečno dolžino poti sporočila ter najdaljšo pot sporočila. Če je največje število skokov večje od logaritma števila vozlišč, to pomeni, da je na poti sporočila vsaj eno vozlišče postalo neaktivno. To vemo zato, ker je moralo sporočilo narediti vsaj en skok v istem nivoju drevesne strukture, ki predstavlja pot sporočila. To se zgodi, kadar neko vozlišče od svojega starša ni prejelo sporočila in je obveščen od soseda. To pomeni, da je neko vozlišče moralo opraviti dvojno delo in porabiti svoje vire namesto neaktivnega vozlišča. Daljša kot je pot, več virov je bilo neenakomerno porazdeljenih v omrežju.

6 Rezultati

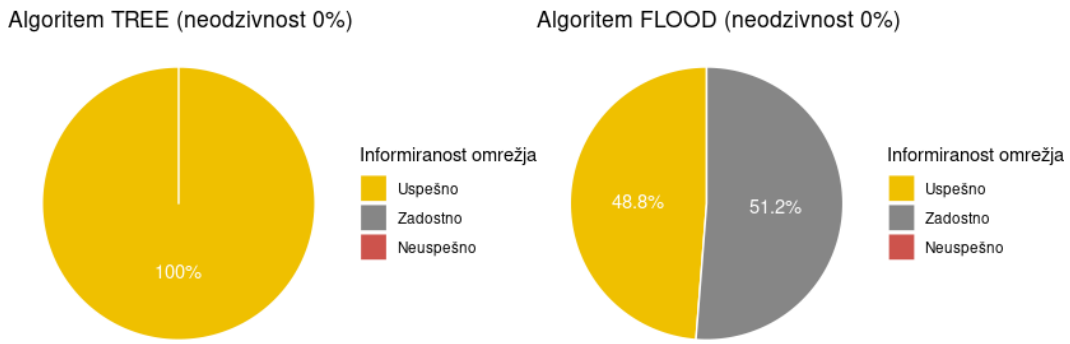
V poglavju so predstavljeni rezultati testiranj algoritmov ter njihova primerjava. Glavne teme primerjave so uspešnost informiranja omrežja, število poslanih sporočil ter hitrost doseganja tega cilja.

6.1 Uspešnost informiranja omrežja

Iz grafa 29 je razvidno, da je predstavljen algoritem v primerjavi z algoritmom poplavljanja zelo polaren. Vidimo, da so vse meritve našega algoritma (TREE) locirane pretežno na dveh točkah, ne glede na velikost omrežja. Algoritem je v večini informiral celotno omrežje. Če pa tega ni storil uspešno, se je ustavil že v samem začetku, kjer je informiranih le nekaj vozlišč. To lahko pripišemo temu, da se algoritem ustavi, če je nedosegljiv celotni sloj drevesa, ki predstavlja pošiljanje sporočila. Ker imajo višji sloji (najbližje generatorju sporočila) najmanj vozlišč, je verjetnost zaustavitve algoritma največja takoj na začetku deseminacije.

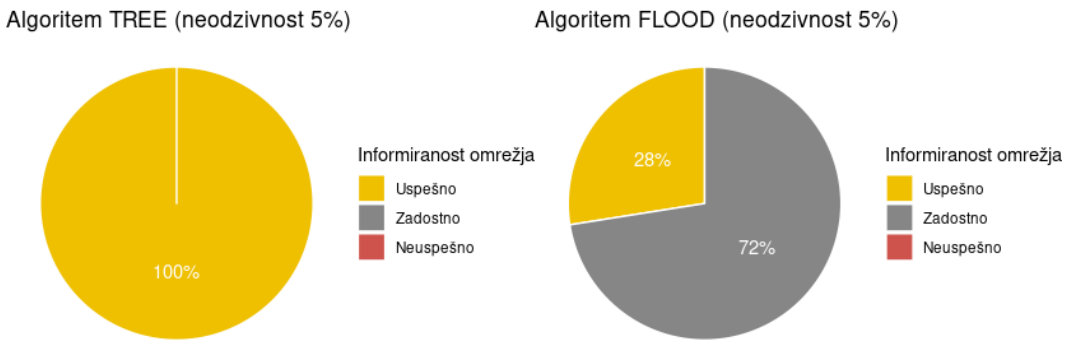


Slika 29: Graf prikazuje uspešnost informiranja vozlišč v omrežju, predstavljeno v procentih v odvisnosti števila vozlišč v omrežju za predstavljen algoritem (TREE) in algoritem poplavljanja (FLOOD). Velikosti točk označujejo število meritev, ki si delijo enako vrednost.

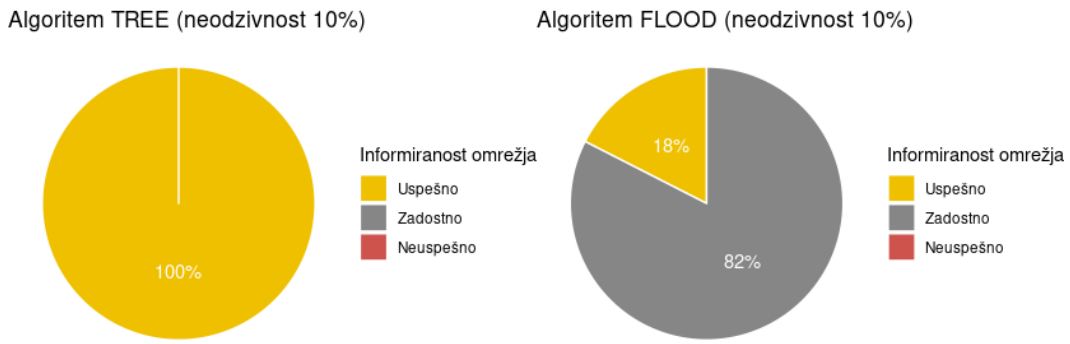


Slika 30: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer so aktivna vsa vozlišča. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano, kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

Ko pogledamo primerjavo na sliki 30, je hitro razvidno, da algoritem poplavljanja ne zagotavlja popolne informiranosti omrežja, če zmanjšamo *fan-out* parameter. Lahko se zgodi, da gre sporočilo v grafu omrežja "okrog"vozlišča. To pomeni, da je visoka verjetnost, da bo omrežje zadostno informirano, vendar ne bodo informirana vsa vozlišča.

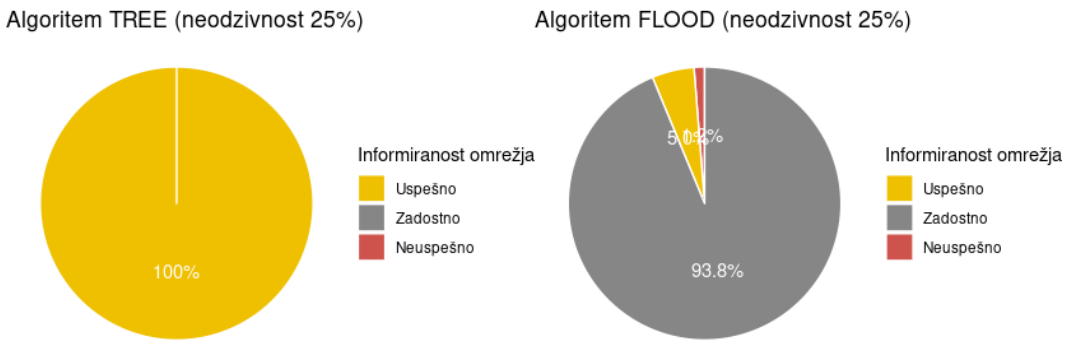


Slika 31: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer je neaktivnih naključnih 5% vozlišč. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano, kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.



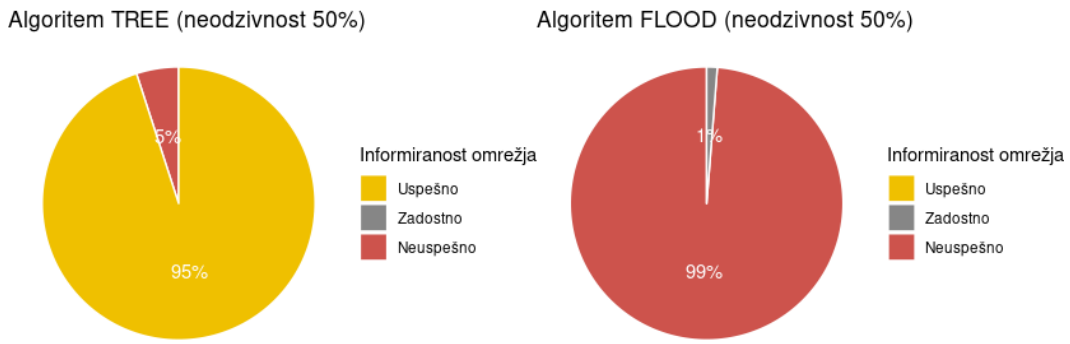
Slika 32: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer je neaktivnih naključnih 10% vozlišč. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

Pri testih, kjer je 5 odstotkov vozlišč neaktivnih, lahko vidimo znižano uspešnost informiranja pri algoritmu poplavljanja(graf 31). Znižanje je kar 20% že pri 5% neodzivnosti. Kljub temu pa so vsa ostala merjenja še vedno zadovoljiva, saj imajo omrežja v tehnologiji veriženja blokov redundanco pri informiranju omrežja. Za razliko od algoritma poplavljanja na uspešnost predstavljenega algoritma takšen odstotek neodzivnih volišč ni vplival.



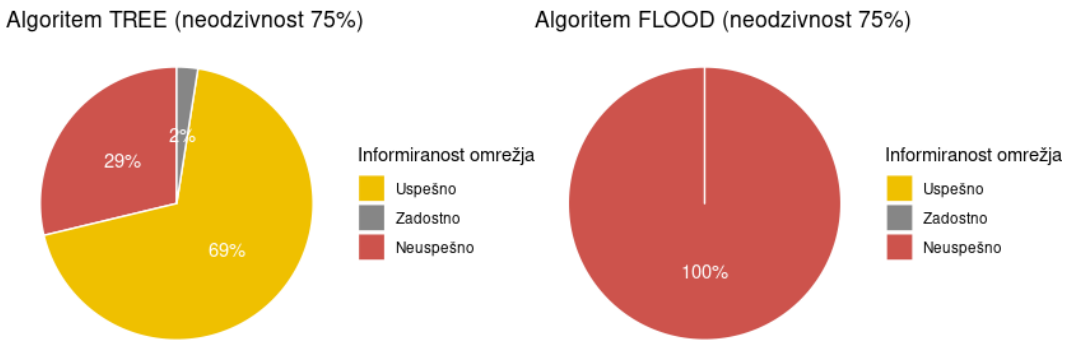
Slika 33: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer je neaktivnih naključnih 25% vozlišč. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

Podoben trend opazimo tudi pri 10% in 25% neodzivnosti vozlišč na grafih 32 in 33. Predstavljen algoritem je pri obeh primerih še vedno 100% uspešen pri informiranju celotnega omrežja, medtem ko algoritmu poplavljanja odstotek uspešnosti konstantno upada, in to na kar 5% uspešnost pri 25% neodzivnosti. Prav tako pa pri algoritmu poplavljanja začnemo opažati mejo algoritma v danem okolju. Pri 25% neodzivnosti ima algoritem poplavljanja 2% neuspešnost informiranja.



Slika 34: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer je neaktivnih naključnih 50% vozlišč. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

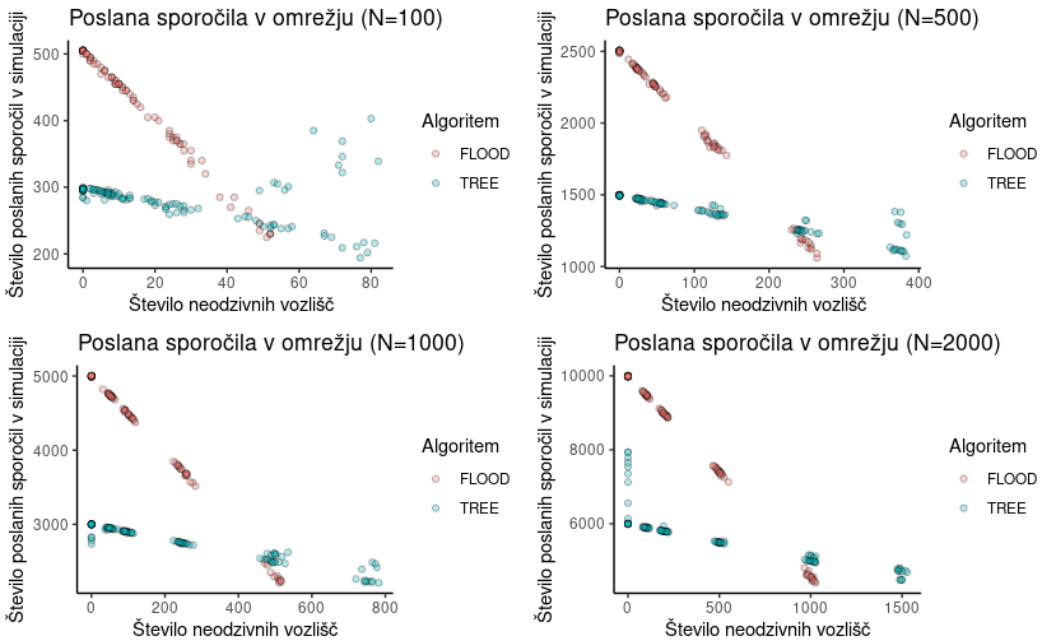
Ko pogledamo grafa 34 in 35 opazimo, da tudi predstavljen algoritem ne prikazuje več 100% uspešnosti. Pri 50% neodzivnih vozlišč ima predstavljen algoritem 5% neuspešno informiranih testov. To pa je še vedno veliko boljša uspešnost, ko jo primerjamo z algoritmom poplavljanja. Le-ta ima 1% zadostno informiranih testov, medtem ko so bili vsi ostali poskusi neuspešni. Pri 75% neodzivnosti pa algoritem poplavljanja niti zadostno ne opravi več nobenega testa. Predstavljeni algoritem pa ima tudi v teh ekstremnih pogojih kar 69% uspešnost. Takšni pogoji so sicer v praksi izjemno redki, vendar so kljub temu rezultati predstavljenega algoritma zelo pozitivni.



Slika 35: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Primerjava prikazuje uspešnosti informiranja omrežja, kjer je neaktivnih naključnih 75% vozlišč. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano, kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

6.2 Število poslanih sporočil

Merjenje števila poslanih sporočil je zelo pomembno, saj nam izmera pove, kako zahteven je algoritem za izvajanje svojega dela na vozliščih glede na zahtevano porabo virov. Iz grafov na sliki 36 je razvidno, da za vsa omrežja ob porastu neodzivnih vozlišč število poslanih sporočil upade. To dejstvo ni samoumevno, saj naš algoritem pošilja dodatna sporočila za preverjanje aktivnosti vozlišč. Zaradi tega je tudi opazno, da količina sporočil v našem algoritmu upada počasneje, kot pri algoritmu poplavljanja. Pri manjših omrežjih pa opazimo, da razporeditev neodzivnih vozlišč v omrežju vpliva na število poslanih sporočil v našem algoritmu. Ta porast sporočil se v večjih omrežjih stabilizira.

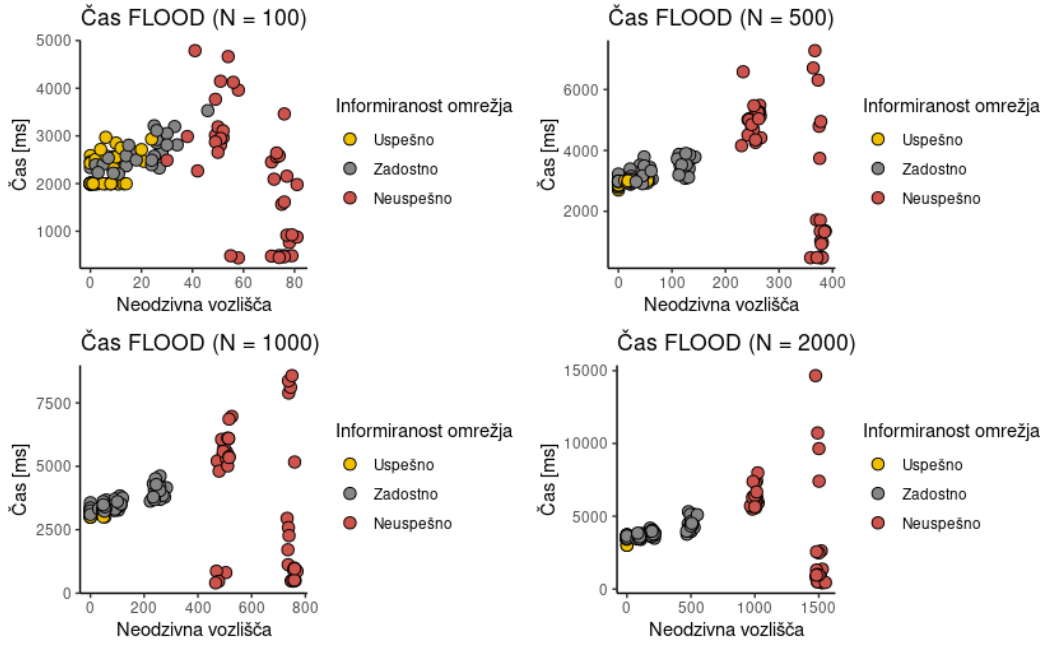


Slika 36: Na sliki je prikazana primerjava našega algoritma (TREE) in algoritma poplavljanja (FLOOD). Vsak graf prikazuje število poslanih sporočil v testu v odvisnosti s številom neodzivnih vozlišč v omrežju. Prikazani so štirje grafi, ki prikazujejo štiri različne velikost omrežja (N). Prikazani so le podatki meritev, kjer je informiranost omrežja uspešna ali vsaj zadostna.

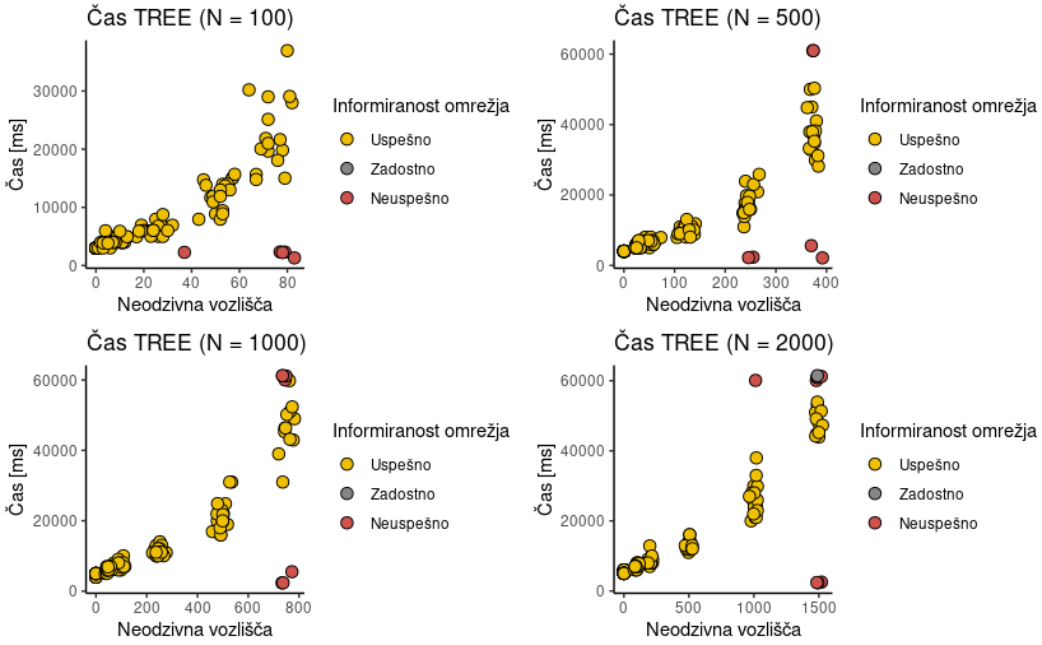
6.3 Hitrost informiranja omrežja

Zadnja mera, ki smo jo obravnavali, je sama hitrost informiranja omrežja. Tu je ob primerjavi grafov na slikah 37 in 38 naš algoritem razvidno počasnejši. To je smiselno, saj vsako vozlišče v našem algoritmu obvesti 2 novi vozišči, medtem ko v algoritmu poplavljanja obvesti vsako vozlišče 5 drugih. To pomeni, da v najboljšem primeru algoritem poplavljanja konča simulacijo v $\log_5 n$ korakih, medtem ko naš algoritem v $\log_2 n$. Iz grafov lahko razberemo, da je hitrost našega algoritma veliko bolj konstanta, kot pri poplavljanju. Poplavljanje svoje prejemnike izbira naključno, medtem ko so pri predstavljenem algoritmu psevdo naključno izračunani. To stabilizira čas izvajanja našega algoritma.

Zanimiva opazka pri našem algoritmu je, da je pri večjih omrežjih veliko neuspešnih testov in zadostnih testov lociranih na maksimalni časovni omejitvi. To pomeni, da algoritem verjetno v tisti točki še ni končal informiranja, vendar je bil prisilno ustavljen zaradi iztečenega časa. V primeru daljšega časa, bi algoritem predvidoma uspešno končal več testov.



Slika 37: Na sliki so prikazani podatki algoritma poplavljanja (FLOOD) v različnih velikostih omrežja (N). Graf prikazuje čas, ki je potreben za celotno simulacijo v odvisnosti s številom neodzivnih vozlišč v omrežju. Vsak zapis (točka) je obarvana različne barve glede na uspešnost informiranja omrežja v simulaciji. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano, kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.



Slika 38: Na sliki so prikazani podatki predstavljenega algoritma (TREE) v različnih velikostih omrežja (N). Graf prikazuje čas, ki je potreben za celotno simulacijo v odvisnosti s številom neodzivnih vozlišč v omrežju. Vsak zapis (točka) je obarvana različne barve glede na uspešnost informiranja omrežja v simulaciji. Omrežje je uspešno informirano, če so sporočilo prejela vsa aktivna vozlišča. Omrežje je zadostno informirano, kadar je sporočilo prejelo vsaj 95% aktivnih vozlišč. Če je sporočilo prejelo manj kot 95% vozlišč, je informiranje omrežja neuspešno.

7 ZAKLJUČEK

V nalogi je predstavljen algoritem za pošiljanje sporočil v porazdeljenih omrežjih. Algoritem je primeren za uporabo v omrežjih, kot so tehnologije veriženja blokov. Z algoritmom smo se primarno osredotočili na minimiziranje števila poslanih sporočil v omrežju. Število sporočil je glavni porabnik pasovne širine, ki je v mnogih algoritmih ne optimizirano. To smo tudi uspešno dosegli, poleg tega pa je v testih algoritem pokazal zelo visoko mero redundance v primerjavi z algoritmom poplavljanja.

V prihodnosti bi bilo potrebno več raziskav algoritma. Ker smo na testih opazili, da se je v primerjavi z algoritmom poplavljanja naš algoritem slabo odrezal v hitrosti informiranja omrežja, bi bilo to zaželeno izboljšati. Ugotovili pa smo, da hitrost lahko dosežemo s povečanjem količine obveščenih vozlišč s strani enega vozlišča.

Menimo, da je bila raziskava uspešna in algoritem deluje po pričakovanjih, na določenih področjih celo nad pričakovanji.

8 LITERATURA IN VIRI

[1] Coulouris, George F and Dollimore, Jean and Kindberg, Tim, *Distributed systems: concepts and design*, 2005. *(Citirano na strani 2.)*

[2] Van Steen, Maarten and Tanenbaum, A, *Distributed systems principles and paradigms*, 2002. *(Citirano na strani 2.)*

[3] PAPADIMITRIOU, Christos H, *Computational Complexity, chapter Approximation and Complexity*, 1994. *(Citirano na strani 2.)*

[4] Arjomandi, Eshrat and Fischer, Michael J and Lynch, Nancy A, *Efficiency of synchronous versus asynchronous distributed systems*, Journal of the ACM, 1983.

[5] Boyd, Stephen and Ghosh, Arpita and Prabhakar, Balaji and Shah, Devavrat, *Randomized gossip algorithms*, IEEE/ACM Transactions on Networking, 2006. *(Citirano na strani 6.)*

[6] The age of cryptocurrency: how bitcoin and the blockchain are challenging the global economic order, *VIGNA, Paul; CASEY, Michael J.*, Macmillan, 2016. *(Citirano na strani 6.)*

[7] CHAUM, David Lee, *Computer Systems established, maintained and trusted by mutually suspicious groups*, Electronics Research Laboratory, University of California, 1979. *(Citirano na strani 4.)*

[8] HALPIN, Harry., *Deconstructing the Decentralization Trilemma*, arXiv preprint arXiv:2008.08014, 2020. *(Citirano na strani 5.)*

[9] Lamport, Leslie and Shostak, Robert and Pease, Marshall, *The Byzantine generals problem*, Concurrency: the Works of Leslie Lamport, 2019. *(Citirano na strani 6.)*

(Citirano na strani 26.)

[10] PADUA, David, *Encyclopedia of parallel computing*, Springer Science & Business Media, 2011. *(Citirano na strani 7.)*

[11] TANENBAUM, Andrew, *Computer networks*, Prentice Hall international editions, 1996. *(Citirano na strani 8.)*

[12] Karp, Richard and Schindelhauer, Christian and Shenker, Scott and Vocking, Berthold, *Randomized rumor spreading*, Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000. *(Citirano na straneh 9 in 10.)*

[13] Agrawal, Divyakant and El Abbadi, Amr and Steinke, Robert C, *Epidemic algorithms in replicated databases*, Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997.

[14] NAG, C, *g05–Random Number Generators*, Citeseer, 2008. *(Citirano na strani 9.)*

[15] *RANDOM.ORG - True Random Number Service*, `http://www.random.org`. (Datum ogleda: 14. 1. 2016.) *(Citirano na straneh 11 in 12.)*

[16] NARAYANAN, Arvind, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*, Princeton University Press, 2017. *(Citirano na strani 12.)*

[17] *The Economist Blockchains: The great chain of being sure about things*, `https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things`. (Datum ogleda: 18. 6. 2016.) *(Citirano na strani 13.)*

[18] Katie Martin, *Financial Times - CLS dips into blockchain to net new currencies*,
`https://www.ft.com/content/c905b6fc-4dd2-3170-9d2a-c79cdbb24f16`.
(Datum ogleda: 7. 11. 2016.) *(Citirano na straneh 13, 16 in 17.)*

[19] Michael Castillo, *Blockchain 50 Billion Dollar Babies*,
`https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain50-billion-dollar-babies/sh=5e8e206857cc`. (Datum ogleda: 1. 2. 2021.) *(Citirano na strani 13.)*

[20] Tapscott, Don in Tapscott, Alex, Fortune - Hereś Why Blockchains Will Change the World.
https://fortune.com/2016/05/08/why-blockchains-will-change-the-world/ 8 (5) 2016. *(Citirano na strani 13.)*

[21] *The Blockhain - Masering Bitcoin[Book]*,
`https //www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html`. (Datum ogleda: 3. 8. 2021.) *(Citirano na strani 16.)*

[22] *OpenJDK - JDK 16*,
`https://openjdk.java.net/projects/jdk/16/`. (Datum ogleda: 7. 8. 2021.) *(Citirano na straneh 17, 18, 19, 20, 21, 22 in 23.)*

[23] Steve, Davies, *PwCś Global Blockchain Survey*, `https://www.pwc.com/gx/en/industries/technology/blockchain/blockchain-in-business.html`. (Datum ogleda: 1. 2. 2021.) *(Citirano na strani 38.)*

[24] KNUTH, Donald, *The Art of Programming*, Addison-Wesley, 1973. *(Citirano na strani 13.)*

[25] CHEN, Huashan, *A survey on ethereum systems security: Vulnerabilities, attacks, and defenses*, ACM Computing Surveys, 2020. *(Citirano na strani 14.)*

[26] MERKLE, Ralph, *A digital signature based on a conventional encryption function*, Conference on the theory and application of cryptographic techniques, 1987. *(Citirano na strani 16.)*

[27] BECKER, George, *Merkle signature schemes, merkle trees and their cryptanalysis*, Ruhr-University Bochum, Tech. Rep, 2008. *(Citirano na strani 15.)*

(Citirano na strani 15.)