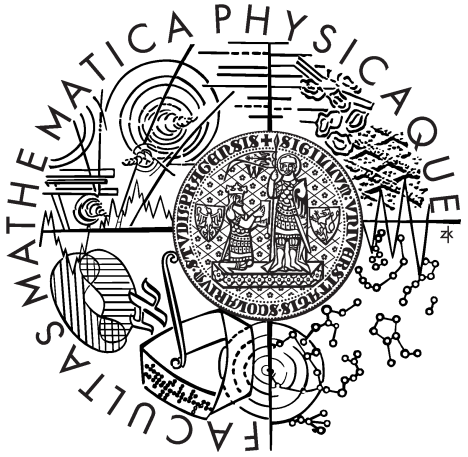
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Daniel Stahr

Sledování obarvených bitcoinů v transakčním grafu

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Petr Baudiš
Studijní program: Informatika
Studijní obor: Programování

Praha 2014

Na tomto místě bych rád poděkoval vedoucímu práce, Mgr. Petru Baudišovi, za cenné rady a podněty. Také bych chtěl poděkovat všem, kteří mě v průběhu psaní práce i celého studia podporovali.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Sledování obarvených bitcoinů v transakčním grafu

Autor: Daniel Stahr

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Petr Baudiš, Katedra aplikované matematiky

Abstrakt:

Práce popisuje, jak v bezhotovostní transakční síti Bitcoinu část prostředků označit, jakým způsobem je poté sledovat, obchodovat a k čemu se dá takové označení využít. Součástí práce je implementace navrženého řešení. Pro ukládání dat byla netradičně zvolena in-memory databáze, což se kladně projevilo na výkonu celé aplikace. Pozitivní přínos rychlého úložiště jsme doložili měřením na reálných datech.

Klíčová slova: Bitcoin, transakční grafy, barevné Bitcoiny, in-memory databáze

Title: Tracking colored bitcoins in the transaction graph

Author: Daniel Stahr

Department: Department of Applied Mathematics

Supervisor: Mgr. Petr Baudiš, Department of Applied Mathematics

Abstract: This thesis describes a way to mark, track and trade particular funds in the Bitcoin transaction network and mentions several uses of the marked, so-called colored Bitcoins. An application demonstrating the principles has been developed. An unusual choice of storing all the data in the memory was made, making a positive impact on the overall performance. We supported its positive contribution by measuring the performance using real blockchain data.

Keywords: Bitcoin, transaction graphs, colored Bitcoins, in-memory databases

Obsah

Úvod

Bezhotovostní finanční transakce jsou v dnešním světě považovány za samozřejmost a od určité výše zákony dokonce uvádějí povinnost je používat. Mohou se tak stát zajímavým objektem pozorování, například pro různé ekonomicko–sociální výzkumy. Jednou takovou transakční sítí (Bitcoin) a jedním konkrétním problémem se zabývá i tato práce.

Pro jeho nastínění použijme všeobecně známý příklad – sběratelské mince a chybotisky. Nominální hodnota je u nich jasně daná, v praxi je však lze často obchodovat za cenu mnohonásobně větší. A to právě díky jejich sběratelské, tedy jakési „virtuální“ hodnotě. O té sice občan používající peníze pro denní potřebu neví, ale pro určitou skupinu lidí je mnohem vyšší než nominální.

Pokud tento princip rozšíříme, nemusíme se omezovat pouze na fyzické, chybně vyražené mince. Jakákoliv skupina může libovolný objekt prohlásit za hodnotnější a jako takový jej posléze brát. Příkladem budiž dětské táborové peníze. Mimo osadu jsou bezcenné, zatímco v jejím rámci mají hodnotu větší, než malý kousek papíru.

Tato práce si klade za cíl navrhnout, jak toto označování zajistit v bezhotovostní transakční síti Bitcoinu a jakým způsobem poté označené peníze obchodovat. Součástí tohoto návrhu by měla být i základní implementace.

V první kapitole se zaměříme na základní vlastnosti Bitcoinu a jeho transakční sítě, ve druhé navážeme rozborem našeho problému a jeho aplikací na specifika naší sítě. Ve třetí kapitole popíšeme návrh programu a implementační detaily. Čtvrtá kapitola je zaměřena na používání a ovládání programu z pozice koncového uživatele. V páté kapitole uvádíme výsledky experimentálních měření.

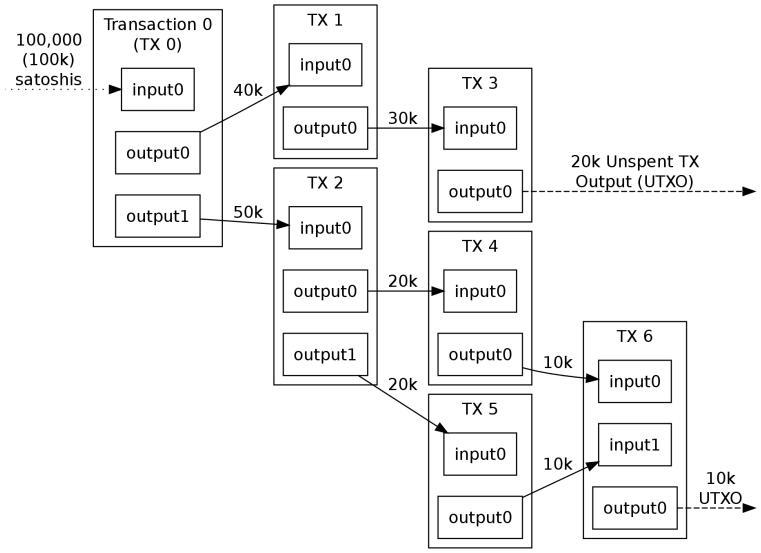
1. Bitcoin

Bitcoin (bitcoin) je nejstarší a nejrozšířenější z decentralizovaných kryptoměn, stejný název se používá také pro jeho transakční síť. Původní verzi softwaru byla uveřejněna v roce 2009 autorem pod pseudonymem Satoshi Nakamoto[1]. Struktura Bitcoinu byla od počátku navržena tak, aby se sítí nemohl jednotlivec nebo autorita nijak manipulovat, resp. aby tato manipulace nebyla reálně možná. Oproti jiným měnám (vázaným např. na politiku národních bank) tak Bitcoinu nehrozí zásahy shůry jako padělání měny, blokování účtů nebo umělé ovlivňování inflace. Stabilitě měny by mělo napomáhat i předem známé maximální množství bitcoinů v oběhu.

S absencí centrální autority souvisí i další podstatný rys decentralizovaných kryptoměn, a to anonymita (resp. pseudonymita). Ačkoliv je celá transakční síť volně přístupná všem, obecně nelze k transakcím přiřadit konkrétní osobu. I proto se stal Bitcoin populární při obchodování na černém trhu.

1.1 Základní terminologie

- *BTC* je zkratka pro Bitcoin, pokud je používán ve smyslu měny (jako USD).
- *Satoshi* je nejmenší jednotka, na kterou lze Bitcoin dělit. 1 satoshi = 10^{-8} BTC.
- *Adresa* je bitcoinová analogie pro číslo účtu a postačující identifikátor pro převod bitcoinů. Každý uživatel sítě může vlastnit neomezeně adres.
- *Transakce* je převod bitcoinů mezi adresami. Jedna transakce může mít více vstupních i výstupních adres.
- *Blok* je nejmenší samostatná část transakční historie. Sdružuje transakce a přídružná data (např. timestamp).
- *Block chain* je kompletní, veřejně přístupná historie všech transakcí obsahující všechny bloky v chronologickém pořadí.



Obrázek 1.1: Ukázka řetězení transakcí[3]

- *Testnet* je alternativní Bitcoin síť využívaná k testovacím účelům. Skutečné transakční síti se říká *livenet*.

1.2 Block chain

Jak již bylo nastíněno, block chain (blockchain) je „účetní kniha“ obsahující kompletní informace o všech potvrzených transakcích (transakci považujeme za potvrzenou, pokud je součástí nějakého bloku). Blockchain má k dispozici každý uživatel sítě a kompletní kopii obdrží po instalaci klienta. Distribuce transakční historie mezi všechny uživatele má velký význam pro zajištění decentralizace celé sítě a také pro ověřování její integrity. Dílčím prvkem blockchainu je blok.

1.2.1 Blok

Bloky sdružují nově příchozí transakce a udržují metadata nutná k zajištění integrity sítě. Jedná se zejména o hash hlavičky předchozího bloku a hash transakcí v bloku obsažených. Transakce příslušící danému bloku se hashují do jediné hodnoty užitím Merkle tree[2]. Pokud by tak někdo chtěl změnit transakci v historickém bloku, musel by změnit i všechny následující bloky. To je při výkonu současného hardware považováno za nemožné. Podrobnosti o tvorbě nových bloků uvedeme v sekci Těžba.

1.2.2 Transakce

Transakce jsou základní hybnou jednotkou každé finanční sítě a Bitcoin není výjimkou. Oproti běžné bankovní síti však má svá specifika.

Hlavní zvláštností bitcoinové transakční sítě je řetězení transakcí. U běžného bankovního účtu se finance po příchodu na účet jednoduše přidají k zůstatku, který na účtě už je. Při odchozí transakci se výše tohoto zůstatku sníží o příslušný objem. Nepozorujeme tedy žádnou spojitost mezi příchozími a odchozími transakcemi (kromě nutné výše zůstatku). Transakční síť Bitcoinu funguje na mírně odlišném principu. Jak si můžeme všimnout na obrázku 1.1, jako vstupy transakce neslouží zůstatek na adrese, ale výstupy několika dalších transakcí. Zadavatel transakce pak může objem bitcoinů z jednotlivých vstupů rozdělit na jeden nebo více výstupů, které mohou být dále použity analogicky. Můžeme si všimnout, že celkový objem prostředků na vstupech nemusí být vždy stejný jako objem bitcoinů, který chceme poslat. Takové situace se typicky řeší přidáním dalšího výstupu vedoucího na adresu odesílatele. Celkový objem transakčních výstupů může být i menší než objem vstupů. Takové bitcoiny se neztrácejí, podrobnosti uvedeme v sekci Těžba.

1.3 Těžba

Těžba (těžení, mining) hraje v ekosystému Bitcoinu naprosto zásadní roli. Jejím prostřednictvím je totiž možno do blockchainu přidávat nové bloky.

Z principu decentralizace plyne potřeba nahradit centrální autoritu, která by ověřovala nově příchozí transakce. Autoři Bitcoinu se rozhodli navrhnout síť tak, aby o ní rozhodovali uživatelé s většinou výpočetního výkonu.

Ověřování transakcí a tvorba nových bloků je tedy otázkou vyřešení výpočetně složitého problému. A jelikož se valná většina návrhu celé sítě točí okolo hashování, byl i pro tento účel zvolen úkol ze stejné oblasti. Hlavička každého bloku obsahuje kromě již zmíněných položek také tzv. *nonce*. Úkolem tohoto parametru je doplňovat hash hlavičky bloku tak, aby na jeho počátku byl daný počet nul. Tím, že takový nonce uživatel sítě najde, prokáže svou výpočetní sílu a má právo zařadit takto nalezený blok do blockchainu.

Náklady na provoz těžebních počítačů jsou nezanedbatelné, neboť běží nepřetržitě na plný výkon. I na to tvůrci Bitcoinu mysleli a nabízejí úspěšnému nálezci dva způsoby odměny :

- *Nově vzniklé bitcoiny* – nálezce má právo vložit do bloku transakci bez vstupu. Její objem se v čase mění, každých 210000 bloků klesne na polovinu. Aktuální hodnota je 25 bitcoinů a další úprava je předpokládána v roce 2016. Tímto způsobem se autoři vypořádali jak s odměnou těžařům, tak se samotnou emisí nových bitcoinů.

- *Poplatky* – nálezci bloku připadnou rozdíly mezi vstupy a výstupy všech transakcí v bloku. Do budoucna by poplatky měly tvořit hlavní část odměny těžařům a začleňování do bloků by mělo prioritizovat transakce s vyššími poplatky. Zatím však v porovnání s novými bitcoiny tvoří zanedbatelný podíl.

Minimální počet nul, které musí hash bloku na začátku obsahovat, se nazývá *obtížnost* (difficulty). Síť je navržena tak, aby ji bylo možné v periodách po 2016 blocích měnit. To zaručí síti schopnost reagovat na výkon aktuálního hardwaru a udrží interval mezi novými bloky na přibližně deseti minutách.

Toto řešení se ukázalo jako velmi prozíravé. Brzy po vzniku Bitcoinu se začaly objevovat speciální aplikace využívající k výpočtům grafické karty a později dokonce hardwarové čipy vyvinuté přímo pro účel těžby. Pro ilustraci uveďme, že za poslední rok vzrostla obtížnost téměř 800×[4].

2. Obarvené bitcoiny

Již v úvodu jsme nastínili, že Bitcoin se může, stejně nako další média, stát nosičem přidané hodnoty. Jeho decentralizovanost a transparentnost transakcí umožňuje směnu takové hodnoty jednoduše, bezpečně a bez potřeby třetích stran. Každá směna navíc může být provedena atomicky v rámci jediné transakce. V neposlední řadě zmiňme, že Bitcoin je možné velmi dobře dělit. To je pro daný účel také vhodné, neboť nosné médium by mělo mít pokud možno co nejmenší hodnotu. Komunita začala pro bitcoiny nesoucí speciální hodnotu užívat termín *obarvené bitcoiny* (colored bitcoins). Meni Rosenfeld ve svém článku[5] uvádí několik příkladů jejich použití. Obarvené bitcoiny:

- ... mohou sloužit jako součást softwarového zabezpečení, např. pro auta a mobilní telefony.
- ... by mohly sloužit jako firemní akcie a bezpečná infrastruktura Bitcoinu by mohla být dále použita pro hlasování.
- ... mohou reprezentovat zatím neexistující hodnoty, jako např. očekávaná úroda, a podle vlastnických podílů pak může být rozdělen zisk.
- ... mohou sloužit také jako dluhopisy, jako speciální případ předchozího bodu.
- ... mohou reprezentovat úplně novou měnu, která může využívat existující infrastrukturu.
- ... mohou vyjadřovat vlastnictví určité movité či nemovité věci. Princip by mohl být podobný jako katastr nemovitostí či centrální registr vozidel.

Jak vidíme, možnosti použití jsou omezeny pouze mírou vynalézavosti uživatelů.

2.1 Protokoly

Jakkoliv jsou myšlenky zmíněné v předchozím odstavci lákavé, Bitcoin jako takový žádné podobné možnosti nenabízí. Je tedy třeba definovat vlastní protokol, podle kterého se obarvené bitcoiny budou transakční sítí šířit.

2.1.1 Genesis transakce

Pro praktickou použitelnost obarvených bitcoinů je třeba dobře definovat, jakým způsobem vznikají. Téměř všechny dále zmíněné protokoly používají princip tzv. *genesis transakce*. Ten spočívá v označení výstupů dané transakce určitou barvou, ze kterých se pak barva dále šíří dle specifikace protokolu. Za úvahu stojí i označování jednotlivých výstupů transakce nebo dokonce jednotlivých satoshi, ale v praxi se ukazuje, že označování celých transakcí poskytuje dostatečnou granularitu.

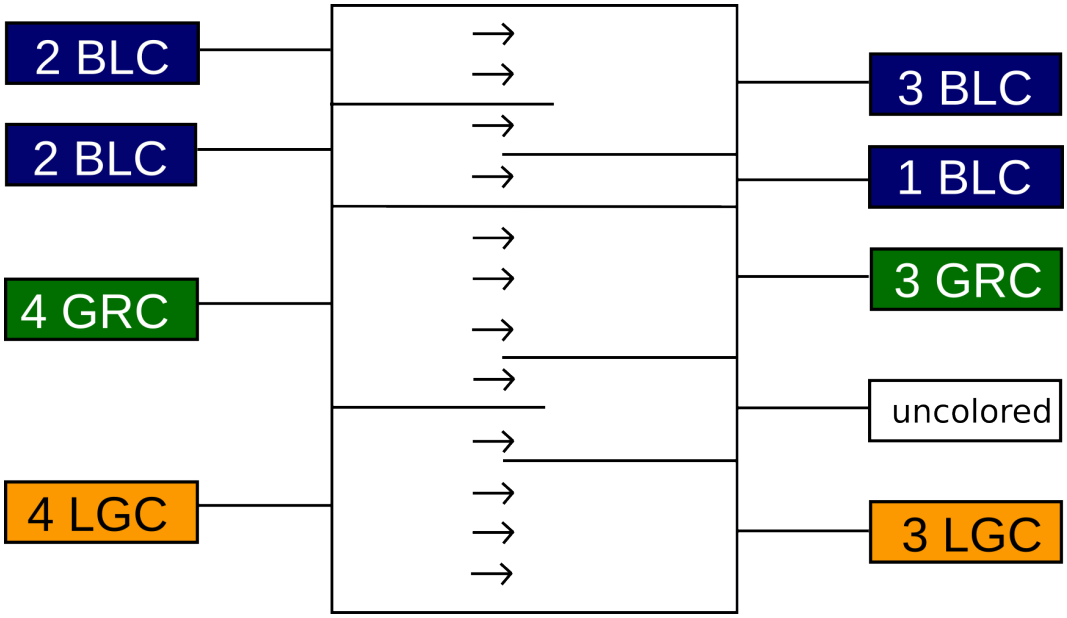
2.1.2 Existující protokoly

Během historického vývoje se několik jedinců či skupin pokoušelo definovat různé protokoly pro šíření barevných bitcoinů. Zmiňme několik z nich a poukažme na jejich výhody a nevýhody.

Šíření dle pořadí

Šíření dle pořadí (order-based coloring) je nejstarší a nejjednodušší protokol pro šíření barevných bitcoinů. Jelikož jsou vstupy i výstupy transakce seřazeny, je možné toto pořadí použít. Na obrázku 2.1 demonstrujeme základní princip šíření dle pořadí. Pořadí určuje provázanost mezi příchozími a odchozími transakcemi. Každý vstup i výstup má svou kapacitu a výstupy používají po řadě kapacitu z transakčních vstupů. Vstupy i výstupy jsou striktně jednobarevné a pokud by mělo na výstupu dojít k mísení barev, je celý označen jako bezbarvý.

Zásadní výhodou tohoto řešení je jeho jednoduchost a přímočarost jak pro uživatele, tak pro implementaci. Pravidla šíření jsou jednoduchá a uživatel může tímto protokolem posílat obarvené bitcoiny i bez specializovaného softwaru.



Obrázek 2.1: Šíření barvy dle pořadí

Nevýhodou tohoto postupu je náchylnost k omezení malých transakcí v síti Bitcoinu. Bitcoin uplatňuje takzvaná *anti-dust* pravidla, která prakticky znemožňují posílání velmi malých částek(takové transakce čekají na začlenění do blockchainu i několik dní). Pokud bychom chtěli, aby naše nová jednotka překračovala hodnotu limitu malých částek, můžeme narazit na další problémy (najednou můžeme „dělit nedělitelné“, samotné nosné bitcoiny mohou mít příliš vysokou hodnotu). Dále tento protokol nepodporuje, aby jeden výstup přenášel více barev najednou. Pokud by se v jednom výstupu namíchalo více barev (popř. barevné a neobarvené bitcoiny), je celý tento výstup označen jako neobarvený. Konečně zmiňme také problém s tzv. obarvenými poplatky. Pokud bude poslední vstup transakce obarvený, je nemožné k transakci přidat poplatek bez ztráty části barevného zůstatku. To platí i pro případ, kdy by nějaký další vstup byl neobarvený.

Obalené šíření dle pořadí

Obalené šíření dle pořadí (padded order-based coloring) je rozšířením předchozího principu, které se primárně snaží řešit problém s anti-dust omezeními. Ke každému výstupu tak přidává konstantně veliké neobarvené množství bitcoinů za účelem překročení hranice pro aplikování filtru.

Oproti prostému šíření dle pořadí tento protokol úspěšně řeší problém s minimální velikostí transakčního výstupu a zachovává poměrně jednoduchou implementaci.

Zásadním problémem tohoto přístupu je jeho komplikovanost. Pro uživatele je nepřívětivé posílat Bitcoiny právě tímto způsobem, musejí dbát na správnou velikost obalení a navíc musejí přidávat ne úplně malou částku z vlastního, aby transakce vůbec proběhla. Konečně, tato metoda je mnohem restriktivnější a náchylnější například ke ztrátě barevných bitcoinů.

Šíření po jednom

Metoda *šíření po jednom* je založena na principu přiřazení unikátní barvy ke každému satoshi. Tento přístup je extrémně nevýhodný, pokud jej budeme používat na obecné transakce (množství přenášených bitcoinů se pohybují o několik řádů výš). Může se ale hodit například pro sledování reprezentace nemovitosti nebo jiného statku, který se typicky příliš nedělí a k jehož reprezentaci stačí jeden nebo málo podkladových satoshi. Pro takové účely pak šíření po jednom představuje jednoduchý a velmi účinný způsob.

2.2 Požadavky na protokol

Na základě výše provedené analýzy existujících protokolů bychom chtěli navrhnout nový nebo některý z těch existujících modifikovat. Námi zvolený protokol by měl splňovat následující kritéria :

- **Stabilita** – Pokud je některá část transakce označena jako nesoucí jistou barvu, je to trvalý a neměnný stav.
- **Uživatelská přívětivost** – Protokol by měl být jednoduše pochopitelný a použitelný pro koncového uživatele.
- **Samostatnost** – Uživatelská přívětivost by neměla být závislá na použití dodatečného softwaru.
- **Snadná implementace** – Protokol by neměl být příliš náročný na implementaci a jednoduchost by se měla projevit i v tomto ohledu.

- **Výkon** – Stav transakční sítě by mělo být možné udržovat inkrementálně s nově příchozími bloky, žádoucí je i možnost předzpracování pro dotazy na transakce.

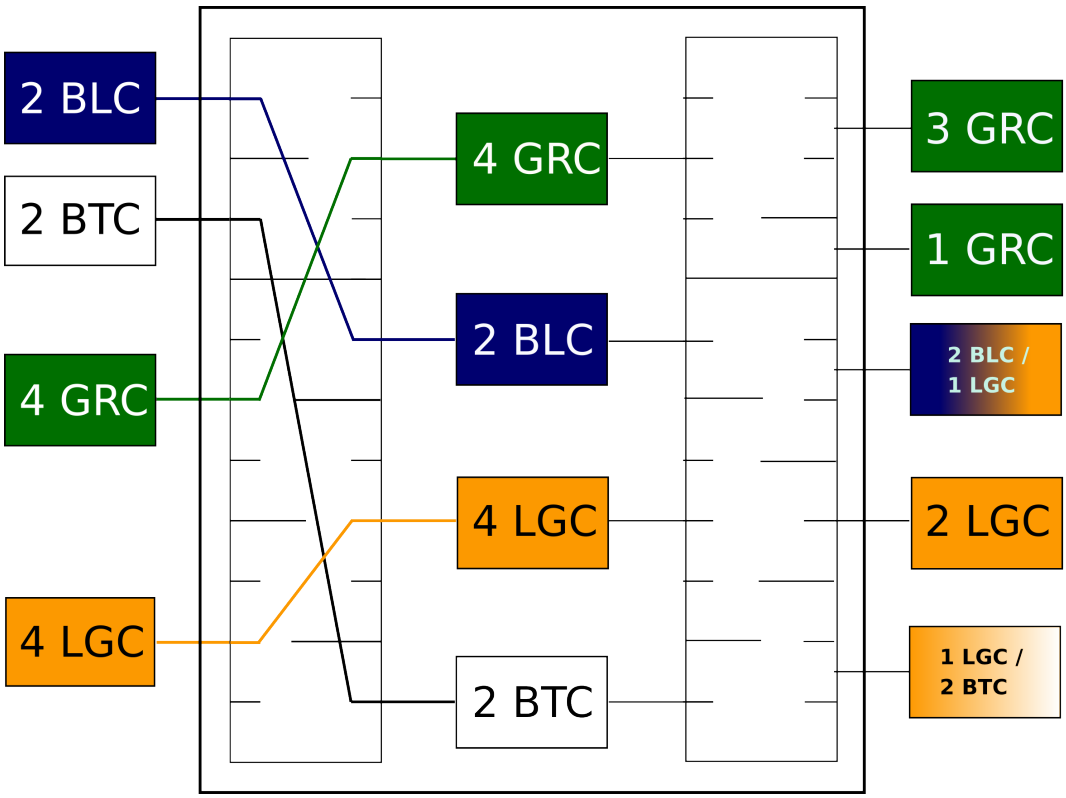
2.3 Návrh protokolu

Na základě výše uvedených požadavků dále rozveďme jednoduchou modifikaci šíření dle pořadí. Vzhledem k předpokládané deflaci bude do budoucna anti-dust pravidla nutné přepracovat. Navíc většina nastíněných použití není závislá na rychlém zpracování transakcí a ani současná pravidla tak nemusejí představovat závažnější komplikaci.

Modifikací, kterou uvažujme, je striktní přednost obarvených bitcoinů na výstupu. Označme součet všech příchozích transakcí včetně informací o barvě jako *zůstatek transakce*. Tento zůstatek budeme distribuovat do výstupů transakce podle identifikátorů barev vzestupně. Výstupy tedy budou postupně zaplněny obarvenými bitcoiny, až následně těmi neobarvenými.

Požadovanou stabilitu zajistíme již zmíněným vzestupným pořadím identifikátorů (id) barev podle toho, v jakém pořadí byly přidávány. Jelikož id nové barvy bude větší než všech ostatních, již přidaných, barev, nová barva nijak neovlivní existující barevnou síť. Tento princip musí platit i pro genesis transakce. Pokud tedy označíme jako genesis transakci nějakou, která už barevné výstupy má, budou novou barvou označeny jen dosud neobarvené výstupní bitcoiny. V krajním případě se tak může stát, že se nová barva v blockchainu vůbec neobjeví.

Tímto přístupem získáváme oproti původnímu šíření dle pořadí hned několik výhod. Za prvé můžeme v rámci jednoho transakčního výstupu bez problémů posílat několik různých barev a náš protokol tak bude méně restriktivní. Dále implicitně vyřešíme problém barevných poplatků – tento případ nastane jen pokud to bude nezbytně nutné. V neposlední řadě bude toto řešení uživatelsky přívětivější, uživatelé nemusí vůbec řešit pořadí vstupů a stačí jim znát celkový objem obarvených prostředků, které do transakce vstupují. Schéma námi navrhovaného protokolu předkládáme na obrázku 2.2, pořadí barev je v tomto případě zelená, modrá a žlutá. Můžeme si všimnout, že oproti původnímu protokolu nenastává



Obrázek 2.2: Šíření dle definovaného protokolu

žádná ztráta barevné informace. Pro úplnost uveďme také pseudokód převodu barev v rámci jedné transakce.

```
def transfer(transakce):
  vstupy = transakce.vstupy
  barvy = {}

  # spocitame zustatek transakce
  for vstup in vstupy:
    vstup_barvy = vstup.get_barvy()
    # vstup_barvy je kolekce dvojic (id barvy, mnozstvi)
    for barva, mnozstvi in vstup_barvy:
      barvy[barva] += mnozstvi

  # seradime barvy na vystup
  bar_sort = barvy.items().sort()
```

```
# Dale probiha sireni analogicky se sirenim dle poradi,
# jen bez restrikce na jednobarevnost vystupu

# barvy_serazene je kolekce dvojic (id barvy, mnozstvi)
# serazena vzestupne podle id barvy
vystupy = transakce.vystupy
barvy_pointer = 0
vystupy_pointer = 0

# postupne plnime neobarvene vystupy obarvenymi bitcoiny
while barvy_pointer < barvy_serazene.length()
    and vystupy_pointer < vystupy.length():
  if vystupy[vystupy_pointer].neobarvene < bar_sort[
     barvy_pointer]:
    # do teto transakce jeste muze prijit jina barva
    vystupy[vystupy_pointer].obarvi(bar_sort[
       barvy_pointer].barva, bar_sort[barvy_pointer].
       mnozstvi)
    barvy_pointer++
  else:
    # naplnili jsme neobarvenou kapacitu, je treba
       pouzit dalsi transakci
    vystupy[vystupy_pointer].obarvi(bar_sort[
       barvy_pointer].barva, vystupy[vystupy_pointer].
       neobarvene)
    bar_sort[barvy_pointer].mnozstvi -= vystupy[
       vystupy_pointer].neobarvene
    vystupy_pointer++
```

Nevýhodou tohoto postupu je závislost sítě na znalosti všech genesis transakcí. Pokud by někomu část informací chyběla, pak může některé bitcoiny chybně považovat za neoznačené a použít je pro šíření vlastní barvy. Pro reálné použití by tedy bylo třeba zajistit synchronizaci označování genesis transakcí, např. přes

centrální server. V případě opravdu masového rozšíření by připadala v úvahu i peer-to-peer varianta podobná té, kterou používá samotný Bitcoin.

2.4 Existující software

V současné době se barevnými bitcoiny zabývá několik open-source projektů, o jejichž koordinaci poměrně čerstvě usiluje iniciativa ColoredCoins[6]. Většina z nich však cílí spíše na uživatele barevných bitcoinů. Programy jsou z hlediska koncového uživatele především rozšířením základní Bitcoin peněženky o zabudovanou podporu různých protokolů. Jako zástupce můžeme jmenovat například projekty Coinprism[1], ChromaWallet[2], Iridis[3] nebo Bitcoin Armory[4].

Přirozenou potřebou těchto programů je udržování barevných zůstatků pro účely ověřování transakcí. Tento problém řeší existence centrálního úložiště, kde se data shromažďují (v tomto světle je naše synchronizace genesis transakcí standardním postupem). Tyto služby ale nemají potřebu uchovávat historická data, neboť pro zachování funkčnosti sítě si stačí pamatovat údaje o dosud neutracených výstupech transakcí. Zmiňme také fakt, že u těchto programů lze obarvit bitcoiny pouze novou transakcí, není možné označit nějakou historickou transakci jako genesis. V neposlední řadě uveďme, že iniciativa ColoredCoins vznikla až po zadání této práce. Např. Coinprism se vyvíjí pouze posledních pár měsíců a prozatím nemá volně přístupný zdrojový kód.

Pokud zavítáme mezi programy komplexně analyzující blockchain, nalezneme několik proprietárních (blockchain.info) i open-source zástupců (např. bitcoin-abe[5] nebo Insight[6]). Podporu pro barevné transakce ale nenabízí ani jeden z nich. Jelikož jsou tyto programy zaměřeny na zpracovávání a zobrazení dat výhradně z blockchainu, nejsou připraveny uživatelsky ani architekturou na další zdroje dat a přidání koncepce barev by vyžadovalo nemalé změny v jejich struktuře.

Po provedení analýzy existujících programů tedy můžeme konstatovat, že nej-

[1] https://www.coinprism.com/
[2] http://chromawallet.com/
[3] https://github.com/owlen/iridis
[4] https://bitcoinarmory.com/
[5] https://github.com/bitcoin-abe/bitcoin-abe
[6] http://insight.is/

lepším řešením pro demonstraci našeho protokolu a pro komplexní prohlížení všech transakcí bude implementovat vlastní aplikaci.

3. Implementace

3.1 Analýza

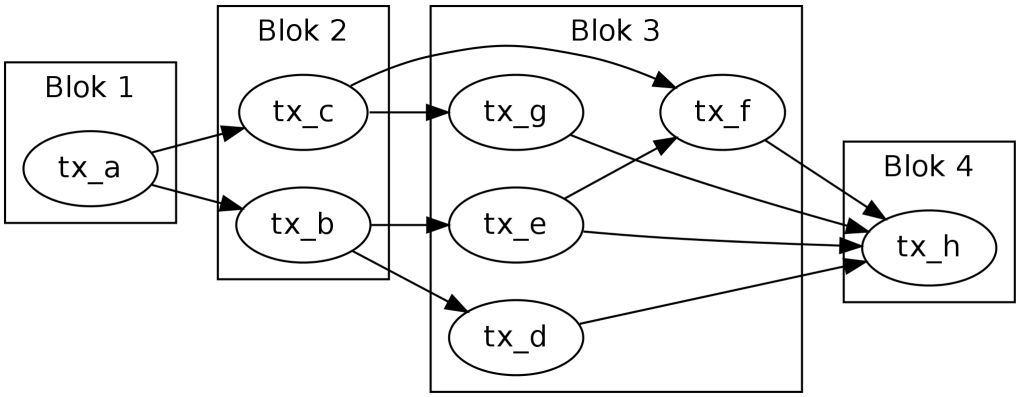
V minulé kapitole jsme se rozhodli nerozšiřovat žádný existující program a raději naimplementovat vlastní řešení. To s sebou nese potřebu analyzovat daný problém a zvolit vhodné prostředky.

3.1.1 Algoritmus

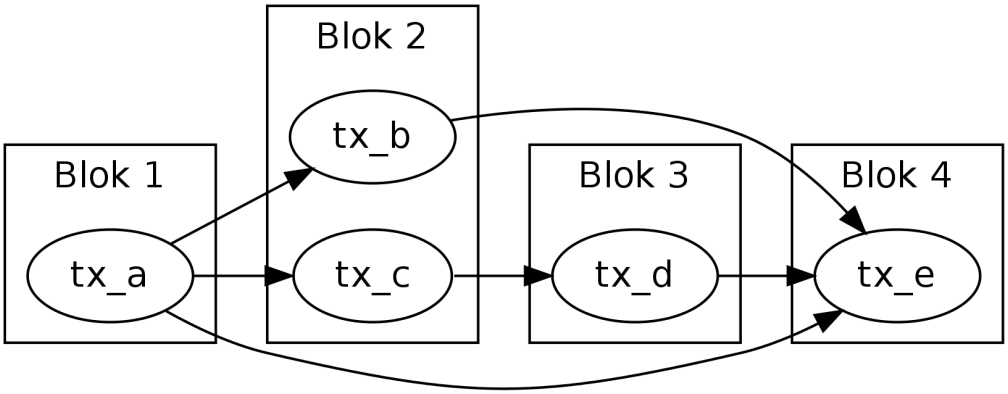
V první řadě je třeba navrhnout algoritmus, kterým budeme simulovat šíření barvy v transakční síti. Šíření barvy v rámci jedné transakce je dobře popsáno protokolem, zásadní roli tak hraje především pořadí zpracování dotčených transakcí.

Prohledávání do šířky

Prvním a nejjednodušším algoritmem, který můžeme uvažovat, je prosté prohledávání do šířky. Pro danou genesis transakci můžeme najít všechny transakční výstupy, kam se barva šíří, a ty dále zpracovat analogicky. Výhodou tohoto řešení je jeho jednoduchost a přímočarost. Významnou nevýhodou však je několikanásobné zpracování transakcí. Uvážíme-li transakční strukturu na obrázku 3.1, transakce *tx_h* (a případné další v řetězci) může být zpracována až pětkrát, což významně prodlouží dobu zpracování.



Obrázek 3.1: Řetěz transakcí nevhodný pro prohledávání do šířky



Obrázek 3.2: Řetěz transakcí nevhodný pro vylepšené prohledávání do šířky

Vylepšené prohledávání do šířky

Drobným vylepšením prohledávání do šířky je kontrola, zda se daný prvek již ve frontě nenachází. V triviálních případech tak můžeme zabránit opakovanému zpracovávání, ale praktickými testy se ukázalo, že v blockchainu se poměrně často nacházejí řetězce transakcí, kdy i toto vylepšení selhává. Jeden takový příklad můžeme vidět na obrázku 3.2.

Topologické zpracování

Jako ideální se ukazuje zpracovávat jednotlivé transakce v topologickém pořadí. Poté se nemůže stát, že bychom některou transakci zpracovávali vícekrát. V praxi nám Bitcoin poskytuje takovou topologii triviálně – budeme transakce zpracovávat chronologicky po blocích a v rámci jednoho bloku pak v pořadí, ve kterém byly v bloku uvedeny. To nám zajistí, že pokud jsme jednu transakci již zpracovali, nikdy už ji nebudeme zpracovávat znovu. Tento postup je v algoritmech poměrně běžný (ze všech příkladů jmenujme například Dijkstrův algoritmus) a zaručí nám příjemnou časovou složitost v nejhorším i průměrném případě.

Zaměřme se na časovou složitost navrhovaného postupu blíže. Pro zpracovávání transakcí ve správném pořadí budeme potřebovat prioritní frontu. Nejpřímočařejším řešením je použití binární haldy[7]. Vzhledem k omezenému univerzu (počet bloků v dalších deseti letech nepřesáhne milion a počty transakcí v jednom bloku se pohybují maximálně v řádu tisíců) můžeme uvažovat i o použití technik založených na přihrádkovém třídění. Asymptoticky lepší výsledek by však

byl v praxi pravděpodobně zastíněn multiplikativními konstantami a o výsledném přínosu se dá úspěšně pochybovat. Použijeme-li tedy binární haldu, časová složitost pro zajištění správného pořadí zpracování bude $\Theta(n \cdot \log n)$, kde n je počet navštívených transakcí.

Při zpracovávání jedné transakce při šíření provedeme seřazení barev plus nejvíce tolik operací, kolik má daná transakce na vstupu barev plus počet jejích výstupů. To je přímo patrno z pseudokódu u definice šíření. Vzhledem k tomu, že šíříme barvu pouze z jedné genesis transakce, počet barev automaticky vypadává[1]. Celková nejhorší časová složitost našeho algoritmu při sledování toku z jedné genesis transakce je tedy $\mathcal{O}(n \cdot \log n + T)$, kde T je počet výstupů všech navštívených transakcí. Vzhledem k faktu, že transakce mají typicky malé množství výstupů, je možné asymptotický rozdíl mezi počtem výstupů a počtem transakcí zanedbat. Náš algoritmus tedy má očekávanou časovou složitost $\Theta(n \cdot \log n)$. Závěrem ještě podotkněme, že této složitosti dosáhneme v případě konstantního přístupu k datům o transakci. Níže v této kapitole problém rozebereme a zvolíme řešení, které tento požadavek splňuje.

Jelikož topologické zpracování v porovnání s ostatními nabízí prakticky samá pozitiva, zvolme pro implementaci právě jej.

3.1.2 Technologie

Programovací jazyk

Na trhu je k dispozici velké množství programovacích jazyků a nedílnou součástí analýzy by měla být i jeho volba. Námi zvolený jazyk by měl představovat kompromis mezi rychlostí a programátorskou přívětivostí, např. množstvím dostupných knihoven nebo správou paměti. Při konzultaci s předními českými zástupci bitcoinové komunity jsme jako takový jazyk zvolili Python. Interpreter tohoto jazyka je běžnou součástí linuxových distribucí a je k němu k dispozici velké množství nejrůznějších knihoven. V případě potřeby lze kritické pasáže přepsat v jazyce C, což umožní další vylepšení výkonu. V neposlední řadě je jazyk široce zastoupen v bitcoinové komunitě, což s sebou přináší výhody jak pro samotné

[1] Více barev najednou zpracováváme pouze při zjištění nového bloku, kde ale zase nehrají roli další transakce v řetězci.

programování (k dispozici jsou knihovny pro práci s Bitcoinem), tak pro následné přijetí programu komunitou.

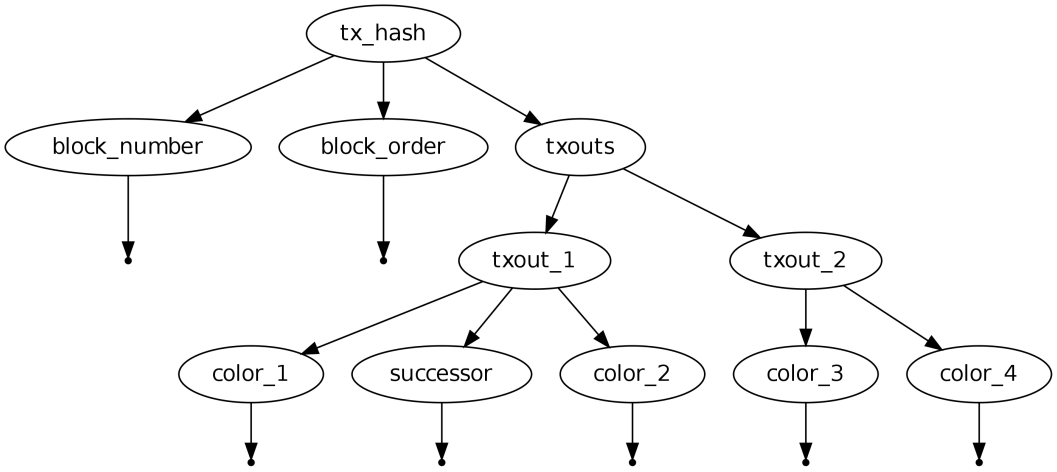
Datové úložiště

Náš program bude zajisté potřebovat uchovávat zůstatky o barevných tocích pro jednotlivé transakce. Také by bylo vhodné předzpracovat dopředné informace pro transakce, protože tato informace v čistých datech z blockchainu není. Naskýtá se tedy otázka, kam tato data ukládat.

V úvahu připadá některá z tradičních relačních databází, kupříkladu MySQL nebo Postgre. Po pokusech s python-abe, který relační databázi používá, jsme se rozhodli tuto možnost dále nerozvíjet. Naše aplikace nedisponuje množstvím různých tabulek, aby dokázala plně využít možnosti relačních databází. Ty s sebou navíc přinášejí zbytečnou režii. Při zpracování blockchainu pomocí python-abe se ukázalo, že režie v databázi svými výpočetními nároky výrazně převyšuje samotný funkční kód.

Další možností je využít nějakou NoSQL databázi. Například LevelDB, kterou mimo mnoha dalších aplikací používá také Bitcoin k ukládání blockchainu. Key-value přístup by vyhovoval jednoduché struktuře našich dat. Analýza aplikace Insight, která LevelDB používá, ukázala, že oproti relačnímu přístupu je tento výrazně méně náročný na výpočetní výkon. Pokud bychom však chtěli použít key-value databázi zapisující na disk, narazíme na problém. Všechny součásti naší aplikace vč. Bitcoin klienta typicky poběží na jednom počítači. Čtení a zápisy na disk od naší aplikace a Bitcoin klienta se tedy budou vzájemně zbytečně zpomalovat.

Na základě předchozího odstavce si položme otázku, zda je možné zápis a čtení nějak oddělit. Řešení se naskýtá v podobě in-memory databází. Zatímco databáze využívající jako úložné médium disk provádějí zápis průběžně, in-memory databáze synchronizují svůj stav s diskem v pravidelných intervalech uložením své kopie, tzv. snapshotu. Jistou nevýhodou těchto databází je nutnost držet svůj kompletní obsah v paměti, což přináší zvýšené nároky na hardware a náchylnost ke ztrátě dat. Výměnou za to však získáme výrazný nárůst výkonu. Vzhledem k nízkým cenám operačních pamětí tak in-memory přístup dostává více



Obrázek 3.3: Struktura dat ukládaných do databáze – návrh

a více prostoru i v komerční sféře a velikosti takových databází již dnes dosahují ke stovkám gigabytů[8].

Z důvodů zmíněných v předchozích dvou odstavcích budeme ukládat data do databáze v paměti. Vzhledem k obnovitelnosti dat by jejich případná ztráta nebyla kritická (údaje o genesis transakcích lze ukládat na perzistentní úložiště) a velikost databáze by se měla pohybovat v jednotkách gigabytů, což je v dnešní době běžná kapacita operační paměti i u levnějších zařízení. Jako konkrétní databáze byl zvolen Redis[9] jakožto software, který je svobodný, stabilní, velmi dobře zdokumentovaný a nabízí knihovní podporu pro velké množství programovacích jazyků.

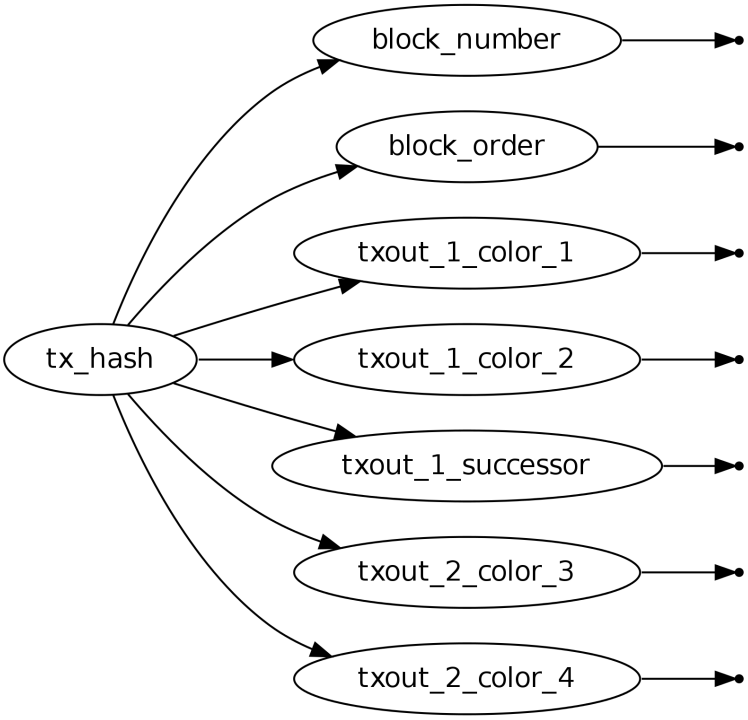
3.1.3 Architektura

Na základě poznatků zmíněných výše v této kapitole se jeví jako ideální aplikaci logicky rozdělit do třech větších celků.

Datová vrstva

Datová vrstva zajišťuje převážně synchronizaci lokální databáze s daty z blockchainu. Jejím úkolem je příprava databáze před prvním spuštěním a následná aktualizace databáze na základě údajů o nově příchozích blocích.

Jako základní jednotku uloženou v databázi zvolme transakci. Informace o bloku je pro nás zajímavá pouze pro určení pořadí a to si mohou jednotlivé transakce nést bez potřeby dalších objektů. Pro účely algoritmu je pro nás podstatné pro



Obrázek 3.4: Struktura dat ukládaných do databáze – úprava

každý výstup znát konkrétní barevné toky a případného následníka, pro celou transakci pak číslo bloku a pořadí v jeho rámci. Modelový příklad takového záznamu vidíme na obrázku 3.3 (uložené hodnoty jsou reprezentovány tečkami). Pro praktické účely a lepší využití architektury Redisu tuto strukturu zploštíme na dvě úrovně. Klíč první úrovně bude nadále hash transakce, klíče druhé úrovně budou vytvářeny pomocí unikátních prefixů. Zmíněný příklad po převedení vidíme na obrázku 3.4. Závěrem provedeme ještě optimalizaci ukládání hashe. Jelikož jsou všechny hashe zapsány pouze pomocí hexadecimální abecedy, je možné sloučit dva znaky hashe do jednoho a ušetřit tak až polovinu diskového prostoru. Tato kompresní metoda je použita pro všechny výskyty hashů transakcí v lokální databázi, a to jak pro klíče, tak pro hodnoty.

Jelikož je plánovaná struktura dat jednoduchá a přímočará, nepředpokládáme, že by datová vrstva sloužila jako rozhraní pro komunikaci s databází. Taková architektura by zbytečně zavlekla další kód s minimální přídavnou hodnotou.

Algoritmická vrstva

Se zpracovanými daty operuje **algoritmická vrstva** mající za úkol simulaci převodu barevných bitcoinů. V algoritmické vrstvě implementujeme šíření barevných bitcoinů a přidružené úkoly. Algoritmická vrstva používá data zpracovaná datovou vrstvou a naopak datová vrstva používá tu algoritmickou (např. po příchodu nového bloku odsimulovat tok do nově příchozích transakcí).

Uživatelská vrstva

Poslední vrstvou je **uživatelská vrstva**, která zpracovaná data pohodlně zpřístupňuje uživateli a také mu umožní vkládat informace o genesis transakcích.

Ostatní

Výše zmíněné vrstvy pak doplňuje sada funkcí společných pro všechny. Připomeňme také externí závislosti – databázi a běžícího Bitcoind klienta.

3.2 Realizace

Jak již bylo zmíněno při návrhu architektury, program je rozdělen na tři základní logické celky – databázový, algoritmický a uživatelský. Ty po řadě reprezentují soubory `blocks_synchronizer`, `colors_processor` a `user_interaction`. Hlavní funkční celky jsou dále doplněny soubory `authproxy` (rozhraní pro komunikaci s Bitcoin klientem), `halfnode` (deserializace dat z blockchainu), `setup` (konfigurační soubor), `shared` (sdílené konstanty) a `util` (sdílená funkcionalita). Spouštění a hlavní smyčku má na starosti soubor `main`.

3.2.1 Synchronizace bloků

Datovou vrstvu zajišťující synchronizaci bloků představuje soubor `blocks_synchronizer`, konkrétně třída `BlocksSynchronizer` v něm obsažená.

Synchronizace z databáze Bitcoinu do lokální databáze může probíhat dvěma způsoby – asynchronně a synchronně. První způsob je rychlejší, druhý oproti tomu umožňuje zohlednit již existující transakce s barevnými bitcoiny a propagovat je do těch nově příchozích.

Asynchronní zpracování

Asynchronní způsob se používá při úvodní inicializaci, jelikož není třeba uvažovat žádné existující obarvené transakce. Využívá třech pracovních front a vzájemně nezávislých pracovních vláken, která data mezi nimi převádějí. V programu jej lze spustit pomocí metody `start`.

První frontou je fronta na čísla bloků. Její aktuálnost zajišťuje metoda `blockWorkFiller`. Ta periodicky monitoruje počet bloků v blockchainu a pokud zjistí nějaké nově příchozí bloky, zapíše jejich čísla do fronty.

Frontu s čísly bloků monitorují vlákna, jejichž pracovní kód je specifikován v metodě `workBlock`. Tato vlákna mají za úkol na základě čísla bloku načíst z Bitcoin klienta data o bloku, zparsovat je a jednotlivé transakce zapsat do druhé, transakční fronty. Pro samotné parsování byl použit volně dostupný zdrojový kód z projektu stratum-minning[2].

Převod mezi transakční frontou a frontou zapisující do databáze je prováděn metodou `workTransaction`. Ta z jednotlivých transakcí vyextrahuje jednotlivé atomické údaje, které mají být později zapsány do databáze. Pod jedním atomickým údajem si můžeme představit list (tečku) z obrázku 3.4.

Poslední součástí řetězce převodu mezi frontami jsou vlákna zapisující do databáze. Ve zdrojovém kódu je reprezentuje metoda `workTransactionToStorage`. Narozdíl od ostatních pracovních vláken nezpracovávají data z front po jedné položce, ale ukládají data po blocích. Přínos tohoto přístupu je zřejmý, výrazně jím snížíme podíl komunikační režie.

Počet vláken pro jednotlivé úlohy je pevně daný a nevyužívá se žádná technika dynamického přidělování volných vláken. Praktické testy ukázaly,že to není nutné. Při malém počtu zpracovávaných bloků a transakcí není podobná technika třeba, přidělená vlákna výkonnostně bez problémů stačí. Při vysoké zátěži je úzkým hrdlem zápis do databáze – rychlejší zpracování dat z blockchainu nemá příliš smysl a při použití příliš velkého počtu vláken zapisujících do databáze narazíme na její interní výkonnostní limity. Většina implementací Pythonu navíc neumí využít více procesorových vláken současně, paralelizace tedy není v rámci jedné instance interpreteru možná.

[2] https://github.com/slush0/stratum-mining

Závěrem ještě uveďme, že pro bezpečné přerušení synchronizace je třeba všechny fronty korektně zpracovat, což může v závislosti na velikosti vstupních dat trvat poměrně dlouho. Je tak nutné dbát na velikost těchto front. Program proto nabízí možnost konfigurace jejich velikosti.

Synchronní zpracování

Synchronní zpracování nově příchozích dat z blockchainu se používá po úvodní inicializaci. Spoští se metodou `start_blocking`. Narozdíl od asynchronního zpracování nepoužívá fronty a výstupy jednotlivých fází jsou přímo předávány dalšímu stupni zpracování, jinak je postup analogický s asynchronním zpracováním.

Oproti asynchronnímu zpracování přináší dvě výrazné změny. První z nich je exkluzivita pro zápis do databáze mezi datovou a algoritmickou částí, což zajišťuje integritu dat. Druhou změnou je šíření barvy do nově synchronizovaných transakcí. Závěrem celé synchronizace se zavolá metoda z algoritmické části, která zajistí rozšíření barev do nově příchozích transakcí. To umožňuje udržovat údaje o barvách stále aktuální.

3.2.2 Šíření barvy

Na algoritmickou část se zaměřuje soubor `colors_processor` a jeho třída `ColorsProcessor`.

Třídě dominuje metoda `color`, která zajišťuje veškeré šíření barevných bitcoinů a jejich zápis do databáze. Jak již bylo zmíněno, pro zajištění integrity dat je nutné nespouštět metodu zároveň s probíhající synchronizací. Ze stejných důvodů také není možné spouštět tuto metodu několikrát zároveň z různých vláken. Samotný kód metody je pouze programovým přepisem topologického zpracování (sekce 3.1.1) a námi definovaného protokolu (sekce 2.3) doplněným o ukládání do databáze.

Mimo hlavní metody se ve třídě nacházejí ještě metody `genesis` a `load_txs`. První jmenovaná slouží k určení identifikátoru barvy a množství mincí, které mají být obarveny, druhá pro načítání dat o transakcích z databáze.

3.2.3 Interakce s uživatelem

Jako uživatelské rozhraní slouží soubor `user_interaction`, resp. třída `UserInteraction` v něm obsažená.

Třída je koncipována tak, aby byla přívětivá pro uživatele, který bude s programem aktivně komunikovat. Uživatelské rozhraní tedy není vhodné např. pro začlenění do skriptů a podobně. Pro tento účel by však stačilo přímo volat jednotlivé dále zmíněné metody.

Hlavním vstupním bodem do uživatelského rozhraní je metoda `poll`. Ta periodicky žádá uživatele o příkazy a volá podle uživatelova vstupu adekvátní metody.

Pokud vynecháme programátorsky nezajímavé metody starající se o ukončení smyčky, ošetření vstupu a vytištění nápovědy, nabízí třída dvě funkční metody. Metoda `color` zajišťuje vytvoření nových barevných bitcoinů na výstupech zadané transakce. Funkčně je závislá na algoritmické vrstvě, sama o sobě pouze zajišťuje ošetření vstupu.

Metoda `balance` vypisuje detailní informace o vstupech a výstupech jednotlivých transakcí. Jelikož informace o vstupech nemáme přímo uložené v databázi, je nutné je získat z výstupů transakcí, které jsou následně použity jako vstupy té naší. Proto se tedy nejprve dotážeme na naši transakci Bitcoin klienta. Údaje o vstupech transakce použijeme pro identifikaci předků, na které se následně dotážeme lokální databáze. Poté již jednoduše zpracujeme detaily o barevných tocích na příslušných výstupech a použijeme je jako údaje o vstupech do dotazované transakce. S ohledem na zaměření celé třídy není výstupní formát primárně zaměřen na strojové zpracování, ale neklade mu žádné překážky v podobě nepravidelností nebo zbytečně bohatého formátování.

3.2.4 Konfigurace

Jelikož je program závislý na externích parametrech (minimálně připojení do databáze a na Bitcoin klienta), je součástí aplikace také konfigurační soubor. Po vzoru jiných aplikací psaných v Pythonu (např. Django) nebo samotného Python interpreteru se konfigurace provádí v samostatném Python souboru. Ten si v případě potřeby jednotlivé funkční moduly importují jako jednu z knihoven.

Kromě již zmíněných údajů o připojení k externím službám lze konfigurovat

nastavení logování, velikosti jednotlivých front a velikosti bloků pro zápis do databáze.

4. Uživatelská dokumentace

Jako referenční operační systém budeme dále uvažovat Ubuntu 14.04 LTS a jeho uživatelskou instalaci. Pro většinu ostatních linuxových systémů budou příkazy stejné či velmi podobné, mohou však vyžadovat instalaci dalších závislostí nebo využití jiného balíčkovacího systému. V případě použití jiného operačního systému se doporučujeme držet slovního popisu.

Na DVD, které je přiloženo k práci, je k dispozici obraz virtuálního počítače pro program VirtualBox, ve kterém je program nainstalován a připraven k použití včetně synchronizované testnet sítě. Pomocí skriptu `/home/bitcoin/colored_bitcoins_env/runall.sh` je možné spustit program včetně obou jeho závislostí.

4.1 Hardwarové požadavky

Pro komfortní používání programu je nejdůležitější dostatek operační paměti. Pro zpracování a analýzu livenetu je doporučeno minimálně 8 GB operační paměti, ideálních je však 16 GB a více.

Aplikace neklade žádné speciální požadavky na typ či výkon procesoru, při použití méně výkonného procesoru lze ovšem očekávat adekvátní zpomalení aplikace.

Ke spuštění a provozování aplikace není třeba žádný specializovaný hardware.

4.2 Prerekvizity

Aplikace je napsána v jazyce Python, je tedy nutné mít nainstalovaný interpreter tohoto jazyka a správce balíčků `pip`. Ve většině linuxových distribucí je předinstalovaný jeho oficiální interpreter CPython. Pro vyšší výkon doporučujeme instalaci alternativního interpreteru PyPy[1]. Je možné, že k nainstalování některých knihoven budou potřeba hlavičkové soubory Pythonu nebo další závislosti. V případě Ubuntu 14.04 LTS stačí nainstalovat balík `python-dev`.

[1] http://pypy.org/

Pro korektní funkčnost programu je dále nutné zajistit RPC připojení k Bitcoin klientovi (verze 0.9 a novější) a k databázi Redis (verze 2.8 a novější). Níže uvádíme postupy instalace a nastavení těchto programů.

4.2.1 Databáze Redis

V případě Ubuntu 14.04 LTS je Redis (ve starší verzi) k dispozici v balíčkovacím systému jako `redis-server`. Pokud není možné z jakéhokoliv důvodu program nainstalovat s pomocí systémových nástrojů nebo pokud vyžadujeme aktuální verzi, postupujeme podle níže uvedeného návodu.

- Z domovské stránky Redisu[9] stáhneme aktuální stabilní verzi[2].

```
wget http://download.redis.io/releases/redis-2.8.13.
    tar.gz
```

- Stažený soubor rozbalíme a otevřeme rozbalenou složku.

```
tar -xvzf redis-2.8.13.tar.gz
cd redis-2.8.13
```

- Zkompilujeme program.

```
make
```

- Pokud kompilace proběhla v pořádku, spustíme program.

```
./src/redis-server
```

4.2.2 Klient bitcoind

Bitcoind poskytuje aplikaci rozhraní pro přístup k datům z blockchainu a stará se také o jejich synchronizaci.

Oproti starším verzím Ubuntu se tento program nenachází v repozitářích, je tedy třeba jej nainstalovat manuálně.

- Z domovské stránky Bitcoinu[3] stáhneme aktuální verzi klienta[4]

[2] V době psaní práce 2.8.13.
[3] https://bitcoin.org
[4] V době psaní práce 0.9.2.1.

```
wget https://bitcoin.org/bin/0.9.2.1/bitcoin-0.9.2.1-
    linux.tar.gz
```

- Stažený soubor rozbalíme.

```
tar -xvzf bitcoin-0.9.2.1-linux.tar.gz
```

- V domovské složce vytvoříme složku `.bitcoin`.

```
mkdir ~/.bitcoin
```

- Otevřeme k editaci soubor `/.bitcoin/bitcoin.conf`.

```
nano ~/.bitcoin/bitcoin.conf
```

- Vyplníme konfiguraci bitcoind klienta. Důležité je nastavit indexování transakcí `txindex=1`, RPC server `server=1`, port `rpcport` a RPC přihlašovací údaje `rpcusername` a `rpcpassword`. V příkladu je též nastavena testovací síť. Soubor uložíme.

```
txindex=1

server=1
rpcport=8332
rpcuser=user
rpcpassword=pass

testnet=1
```

- Spustíme bitcoind klienta. V případě použití 32bitového systému stačí v příkazu nahradit 64 za 32.

```
./bitcoin-0.9.2.1-linux/bin/64/bitcoind
```

Po spuštění doporučujeme vyčkat, než se blockchain plně synchronizuje. To může trvat několik hodin až dní.

4.3 Instalace a spuštění

- Z repozitáře projektu stáhneme aktuální verzi. Pokud máte nainstalovaný program pro správu verzí Git, můžete jej použít. Staženou složku otevřeme.

```
# s~nainstalovanym Gitem
git clone https://github.com/Danstahr/colored-bitcoins
    -project.git

# bez nainstalovaneho Gitu
curl -L -o colored_coins_code.zip http://github.com/
    Danstahr/colored-bitcoins-project/zipball/master/
unzip colored_coins_code.zip
```

- Spustíme instalaci závislostí pomocí skriptu `install_dependencies.sh`. Pro korektní instalaci knihoven je v závislosti na konfiguraci systému možné, že bude vyžadováno administrátorské oprávnění.

```
sudo ./install_dependencies.sh
```

- Program spustíme pomocí skriptu `run.sh`. V případě, že jste nepostupovali podle návodu na zprovoznění prerekvizit výše, bude pravděpodobně nutné provést změny v konfiguračním souboru (viz příslušná sekce).

```
./run.sh
```

4.4 Ovládání

Po spuštění programu a úspěšné synchronizaci bude uživateli nabídnuto jednoduché interaktivní prostředí, ve kterém zadává příkazy. Jejich seznam uvádíme níže.

- `help` – Vypíše seznam všech příkazů včetně stručného popisu.
- `color` – Jako parametr přijímá hash transakce. Danou transakci (resp. její dosud neobarvené výstupy) označí jako genesis transakci pro novou barvu, jejíž identifikátor je po provedení obarvení vypsán uživateli. Níže uvádíme příklad použití.

```
Enter a command ('help' for list of available
    commands) : color tx_hash
Uncolored outputs of transaction tx_hash set to
    color 1
```

- `balance` – Jako parametr přijímá hash transakce. Zobrazí podrobnosti o vstupech a výstupech dané transakce, včetně informací o barvách. Pro ilustraci uvádíme výstup z příkazu.

```
=== tx_hash transaction details : ===

Contained in block 270946

INPUTS :
    0 -> coming from block 270945, transaction tx_hash
        , output 0
         Color 0 -> 239810000 satoshi
OUTPUTS :
    0 -> spent in block 270946, transaction tx_hash
         Color 1 -> 239799454 satoshi
    1 -> unspent yet
         Color 1 -> 546 satoshi

=== End of transaction details ===
```

- `exit` – Ukončí program.

4.5 Konfigurace

Jelikož je program cílen na čistě interaktivní použití, nespouští se se žádnými parametry nebo přepínači. Veškerá nastavení probíhají pomocí editace konfiguračního souboru. Ten je umístěn ve složce `colored_bitcoins` jako soubor `setup.py`. Pro korektní funkčnost programu je nutné ponechat všechny parametry definované a měnit pouze jejich hodnoty.

- `redis_setup` – Slouží k nastavení připojení do databáze. Nejčastěji budeme chtít měnit adresu serveru (`host`), port (`port`) nebo číslo databáze(`db`). Pro úplný výčet parametrů odkažme na dokumentaci ke knihovně redis-py[5]. Jednotlivé parametry zapisujeme ve formátu slovníku jazyka Python[6]. Celou konfiguraci tedy zapisujeme do složených závorek, klíče i textové hodnoty ohraničujeme uvozovkami a vzájemně je oddělujeme dvojtečkou. Jednotlivé dvojice klíč : hodnota pak oddělujeme čárkou. Níže je uveden jednoduchý příklad konfigurace.

```
redis_setup = {
    "host": "example.com",
    "port": 6379
}
```

- `bitcoind_connection` – Slouží k nastavení připojení k bitcoind RPC serveru. Korektním nastavením je jediný řetězec ve standardním URL formátu `http://uzivatelske_jmeno:heslo@host:port`. Níže uvádíme příklad takového řetězce.

```
bitcoind_connection = "http://user:pass@example.com
    :18332"
```

- `bitcoind_timeout` – Slouží k nastavení timeoutu pro dotazy na bitcoind klienta. Potýkáte-li se s pomalým připojením či s vysokou zátěží počítače, zvyšte tuto hodnotu.

- `blocks_queue_length`, `transactions_queue_length` a `database_queue_length` – Slouží k nastavení velikosti vyrovnávací paměti pro jednotlivé stupně zpracování. Zvýšení hodnot může přinést malý nárůst výkonu za cenu vyšší spotřeby paměti. Doporučujeme ponechat na výchozích hodnotách, v případě nedostatku paměti pak hodnoty snížit.

- `database_buffer_size` – Určuje počet záznamů, která jsou zapisovány do databáze během jednoho volání. Vyšší hodnota může přinést lepší výkon, obzvlášť pokud je databáze nainstalována na jiném stroji.

[5] http://redis-py.readthedocs.org/en/latest/
[6] https://docs.python.org/2/tutorial/datastructures.html#dictionaries

Mimo výše zmíněných parametrů se v konfiguračním souboru provádí nastavení logování. V aplikaci je použita knihovna `logging`, která je součástí standardních knihoven Pythonu. Pro podrobnosti o jejím nastavení tak odkažme na její dokumentaci[7].

[7] https://docs.python.org/2/library/logging.html

5. Praktické testy

V této kapitole uvádíme výsledky experimentálních měření výkonu. K testování byl použit virtuální počítač ve VirtualBoxu se čtyřmi dostupnými jádry, 2 GB RAM a operačním systémem Ubuntu 14.04 LTS 64bit. Virtuální stroj byl nastaven podle návodu v kapitole Uživatelská dokumentace, včetně instalace interpreteru PyPy. Podkladový systém tvořil čtyřjádrový procesor AMD™Phenom II 945 taktovaný na 3 GHz, 6 GB operační paměti a operační systém Debian Wheezy. Veškerá měření probíhala s použitím testnetu.

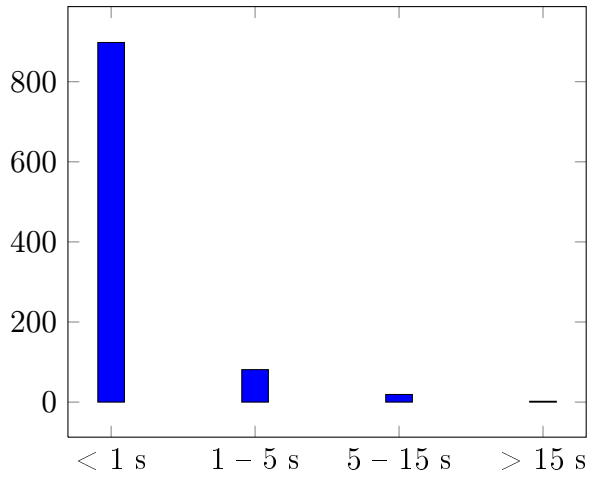
5.1 Úvodní synchronizace

Měření času potřebného na úvodní synchronizaci proběhlo pětkrát, což eliminovalo náhodné výkyvy zapříčiněné jinými běžícími procesy. Ve výsledku se však jednotlivé hodnoty lišily maximálně o jednotky minut. Nejrychleji byl blockchain synchronizován za **1 hodinu, 32 minut a 28 sekund**. V průměru vyžadovala synchronizace **1 hodinu, 36 minut a 15 sekund**.

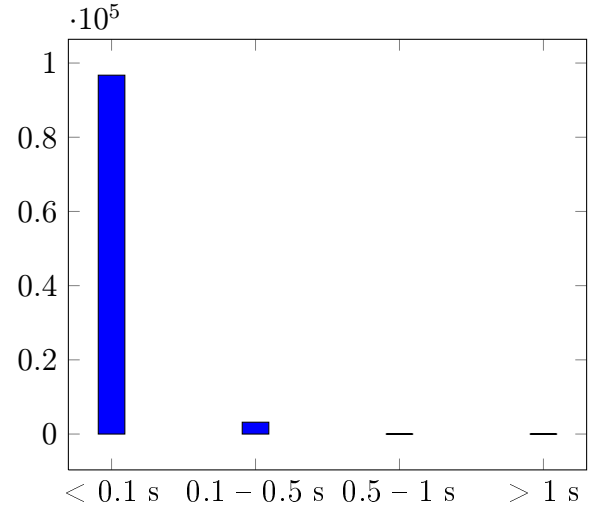
5.2 Označování genesis transakcí

Zadávání nových genesis transakcí je výpočetně nejnáročnější činnost, jelikož musí proběhnout propagace nové barvy po celé transakční síti. Jako testovací model jsme zvolili obarvování 1000 náhodných transakcí s běžící synchronizací bloků. V testu jsme tak zohlednili i situace, kdy bude v pozadí probíhat synchronizace nového bloku, což se negativně projeví na odezvě. Zdrojový kód testu je umístěn ve složce s programem v souboru `test.py`.

Z grafu 5.1 vidíme, že valná většina požadavků je zpracována v čase pod jednu sekundu. Do pěti sekund, což je stále z hlediska uživatele přijatelný čas, je pak zpracováno 99,8% všech požadavků.



Obrázek 5.1: Histogram délek zpracování nových genesis transakcí



Obrázek 5.2: Histogram délek zpracování dotazů na zůstatky

5.3 Dotazy na transakce

Jelikož dotazy na barevné zůstatky budou mnohem běžnější operací než zadávání genesis transakcí, bylo pro testování použito 100000 dotazů na náhodné transakce. Testování probíhalo za stejných podmínek jako označování genesis transakcí a zdrojový kód testu je taktéž možno nalézt v souboru `test.py`.

Naměřené hodnoty jsou zaneseny do grafu 5.2. Opět si můžeme všimnout, že valná většina dotazů je vyřízena téměř okamžitě. Dotazy trvající déle než 0.5 sekundy pak můžeme označit za anomálie, neboť je jejich podíl vskutku mizivý.

5.4 Livenet

V průběhu práce na projektu bylo provedeno také několik měření s použitím livenetu. Považujme je pouze za orientační, neboť neprobíhala s finální verzí kódu.

V porovnání s testnetem trvala výrazně déle úvodní synchronizace. To je vzhledem k cca desetinásobnému počtu transakcí logické. Přesto byla synchronizace hotova za méně než den, což je plně srovnatelné s jinými projekty zaměřenými na zpracování blockchainu. Navíc – oproti bitcoin-abe (také napsanému v Pythonu) je na tom naše aplikace výrazně lépe.

Poměrně překvapivým zjištěním bylo, že počet transakcí nemá velký vliv na výkon označování genesis transakcí ani na dotazy na zůstatky. S největší pravděpodobností je to zapříčiněno faktem, že livenet používá mnohonásobně více uživatelů, jejichž transakce se vzájemně nekříží. Jako další možný důvod uveďme fakt, že značné množství bitcoinů vůbec není v oběhu a mnoho transakčních výstupů tak zůstává neutracených. Na tuto skutečnost poukazují např. Dorit Ron a Adi Shamir[10].

Závěr

Cílem práce bylo navrhnout vlastní protokol pro šíření barevných Bitcoinů a implementovat aplikaci, která na něm bude stavět. Tyto požadavky se podařilo splnit – využili jsme již existujícího protokolu, který jsme modifikovali tak, abychom eliminovali jeho neduhy a zároveň zachovali jeho jednoduchost.

V implementační části jsme pak demonstrovali použití databáze Redis i pro jiný účel, než je běžně používána (tj. kešování). Podařilo se ukázat, že in-memory databáze může pozitivně přispět k celkovému výkonu aplikace. Ukázali jsme, že použitím vhodného formátu dat můžeme výrazně snížit paměťovou náročnost. Tím zpřístupníme in-memory databáze i pro aplikace, které pracují s velkým množstvím dat.

Pokud bychom chtěli aplikaci dále rozšiřovat, pravděpodobně bychom se zaměřili především na integraci s ostatními, již existujícími programy. Do našeho programu bychom mohli např. zakomponovat podporu dalších protokolů (pro každý protokol by bylo třeba implementovat vlastní alternativu třídy `ColorsProcessor`).

Dalším možným rozšířením by mohlo být využívání dat z programů zpracovávajících blockchain. Mohli bychom kupříkladu využívat jejich indexy adresa → transakce a nabídnout uživateli prohlížení zůstatků podle adres.

Nakonec zmiňme rozšíření možností interakce s programem. Programátorům bychom mohli zpřístupnit API pro obarvování a zjišťování zůstatků. Uživatelé by mohli uvítat například grafické nebo webové rozhraní.

Seznam použité literatury

[1] Nakamoto, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. [2009] [cit. 2014-07-28]. Dostupné z https://bitcoin.org/bitcoin.pdf.

[2] Becker, Georg. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis* [online]. 18.07.08 [cit. 2014-07-28]. Dostupné z http://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf.

[3] Bitcoin.org. *Triple-Entry Bookkeeping (Transaction-To-Transaction payments) As Used By Bitcoin* [online]. [cit. 2014-07-07]. Dostupné z https://bitcoin.org/img/dev/en-transaction-propagation.svg.

[4] Blockchain.info. *Difficulty Chart* [online]. 2011 [cit. 2014-07-06]. Dostupné z http://blockchain.info/charts/difficulty.

[5] Rosenfeld, Meni. *Overview of Colored Coins* [online]. December 4, 2012 [cit. 2014-07-06]. Dostupné z https://bitcoil.co.il/BitcoinX.pdf.

[6] coloredcoins.org. *Connect all Colored Coins initiatives to share resources and ideas* [online]. 2013 [cit. 2014-07-06]. Dostupné z http://www.coloredcoins.org.

[7] Williams, John William Joseph. ALGORITHM-232-HEAPSORT. *Communications of the ACM* 7.6, 1964, s. 347-348.

[8] Couch, Courtney. *Redis as the primary data store? WTF?!* [online]. April 8 2013 [cit. 2014-07-16]. Dostupné z https://muut.com/blog/technology/redis-as-primary-datastore-wtf.html.

[9] Redis. *An open source, BSD licensed, advanced key-value store* [software]. [cit. 2014-07-16]. Dostupné z http://redis.io/.

[10] Ron, Dorit a Adi Shamir. *Quantitative Analysis of the Full Bitcoin Transaction Graph* [online]. [2012] [cit. 2014-07-29]. Dostupné z https://eprint.iacr.org/2012/584.pdf.

A. Obsah přiloženého DVD

- /
  - `src` ..... Zdrojové kódy aplikace
  - `text`
    - `text.pdf` ..... Text práce ve formátu PDF
    - `src` ..... Zdrojové kódy textu ve formátu LaTeX
  - `VirtualBoxVM` ..... Komprimnovaný obraz pro VirtualBox s připraveným prostředím