UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Uroš Hercog

Internet vrednosti: implementacija konektorja za protokol Interledger

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Matjaž Branko Jurič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Analizirajte tehnologijo veriženja podatkovnih blokov (blockchain) in proučite uporabno vrednost na področju prenosa sredstev. Naredite pregled obstoječih protokolov, ki se uporabljajo v finančnem svetu. Podrobno analizirajte protokol Interledger. Razvijte lasten konektor za protokol Interledger ter vtičnika za omrežji PayPal in Ethereum. Ovrednotite implementacijo in delovanje konektorja in vtičnikov ter identificirajte prednosti, slabosti in priložnosti za nadaljnje delo.

Zahvaljujem se mentorju prof. dr. Matjažu Branku Juriču in asist. dr. Janu Meznariču za vse napotke in potrpežljivost med pisanjem diplomske naloge. Hvala podjetju GateHub za zaupanje in okolje za rast.
Neizmerna hvala družini za vso podporo in spodbudo med študijem.

Kazalo

Kazalo slik

Kazalo tabel

Seznam uporabljenih kratic

kratica	angleško	slovensko
BIC	Bank Identifier Code	bančna identifikacijska koda
FTP	File Transfer Protocol	protokol za prenos datotek
BTC	Bitcoin	Bitcoin
IBC	Interblockchain Communication	komunikacija med verigami blokov
IP	Internet Protocol	internetni protokol
URL	Uniform Resource Locator	enolični identifikator vira
IANA	Internet Assigned Numbers Authority	organ za dodeljevanje internetnih naslovov
RTT	Round Trip Time	čas povratne poti
TRR	Transaction Account	transakcijski račun
ILP	Interledger Protocol	protokol Interledger
HTTPS	Secure Hyper Text Transfer Protocol	protokol, ki omogoča varno spletno povezavo
BTP	Bilateral Transfer Protocol	bilateralni prenosni protokol
API	Application Programing Interface	programski vmesnik
NAT	Network Address Translation	preslikava omrežnega naslova
OER	Octet Encoding Rules	pravila za kodiranje oktetov
ASN.1	Abstract Syntax Notation	abstraktna sintaksna notacija
DNS	Domain Name System	sistem domenskih imen
M2M	Machine To Machine	komunikacija med napravami
IPT	Inductive Power Transfer	prenos energije z indukcijo
DDoS	Denial of Service	porazdeljena ohromitev storitve
TCP	Transmission Control Protocol	protokol za nadzor prenosa

Povzetek

Naslov: Internet vrednosti: implementacija konektorja za protokol Interledger

Avtor: Uroš Hercog

Takojšnji prenos sredstev med različnimi plačilnimi sistemi danes predstavlja velik izziv. Njihova interoperabilnost bo z modernizacijo bančništva in s hitrim razvojem novih sistemov vedno bolj pomembna. Izmenjava sredstev med različnimi plačilnimi sistemi mora biti neposredna in instantna. Za to potrebujemo protokol, ki navedene zahteve neposredno naslavlja. Takšen protokol je Interledger, ki ga opišemo v nalogi.

V diplomski nalogi analiziramo področje interoperabilnosti plačilnih sistemov in podamo njegovo kritiko. Podrobno predstavimo plačilni protokol Interledger. Razvijemo in opišemo programsko rešitev, ki omogoča pošiljanje sredstev preko tega plačilnega protokola. Rešitev ovrednotimo tako, da izvedemo prenos sredstev med tremi različnimi plačilnimi sistemi. Na podlagi rezultatov vrednotenja rešitve sklenemo, da naša implementacija konektorja za protokol Interledger rešuje zastavljene probleme.

Ključne besede: Interledger, konektor, interoperabilnost, tehnologija veriženja blokov, glavna knjiga.

Abstract

Title: Internet of Value: implementation of Interledger protocol connector

Author: Uroš Hercog

Instant transfer of funds between different payment systems presents a great challenge. Interoperability of payment systems will play a major role in the modernization of existing financial systems and in the development of new payment systems, which are introduced daily. Interchange of funds between these systems should be direct and instant. To achieve this, we need a protocol that directly addresses these requirements. Such a protocol is Interledger, which we describe in detail.

In this thesis, we critically analyse the field of payment system's interoperability and present in full detail a new payment protocol, named Interledger. We develop and describe a software solution, which enables sending funds through this payment system. We evaluate the solution by executing a transfer over three different existing payment systems. Based on the results, we conclude that our implementation of an Interledger connector solves the given problems.

Keywords: Interledger, connector, interoperability, blockchain, ledger.

Poglavje 1

Uvod

1.1 Motivacija

Danes nihče ne razmišlja o tem, ali lahko pošlje elektronsko pošto z Gmaila nekomu, ki ima račun odprt pri Yahoo Mail ali katerem koli drugem ponudniku. Vse, kar potrebujemo, je prejemnikov e-poštni naslov in pošto mu lahko pošljemo. Takšno premikanje informacij med velikimi ponudniki je skoraj instantno in ga jemljemo za nekaj samoumevnega. Vendar pa prenos sredstev med različnimi plačilnimi sistemi terja neprimerno več časa in stroškov. Skoraj nemogoče je neposredno prenesti sredstva z računa Paypal na Venmo ali h kateremu koli drugemu podobnemu ponudniku. Izvedba tega terja depozit na osebni bančni račun s Paypala in dvig z osebnega bančnega računa na račun, ki ga imamo odprtega pri Venmu. Ti koraki lahko zaradi neposredne vpletenosti bank trajajo od nekaj ur pa do več dni. Kot odgovor na te težave so se začeli pojavljati različni globalni plačilni sistemi, ki večinoma temeljijo na tehnologiji veriženja blokov in obljubljajo takojšnje transakcije. Vendar že danes vidimo, da ta tehnologija brez delne centralizacije ne omogoča obdelave več deset tisoč transakcij na sekundo, kot jih omogočata Visa ali MasterCard. Namesto da se trudimo narediti skupni globalni plačilni sistem, s katerim se bodo vsi, vključno s finančnimi institucijami, strinjali in ga uporabljali, bi bilo bolje, da obstoječe sisteme povežemo na način, da bodo interoperabilni. S tem bi omogočili, da bi lahko nekomu na dru-

gem koncu sveta sredstva poslali v sekundah namesto v dneh. Obenem bi lahko pošiljatelj sredstva poslal v poljubni valuti na nekem omrežju, na drugi strani pa bi prejemnik ta sredstva prejel v valuti na omrežju, ki mu najbolj ustreza. Sistem bi sam poskrbel za vse potrebne pretvorbe in čim bolj optimalno pot skozi sisteme v omrežju do prejemnika. To lahko dosežemo s standardizacijo načina komunikacije in izmenjave sredstev med dvema popolnoma različnima sistemoma glavnih knjig. Rešitev težave standardizacije nudi protokol Interledger.

1.2 Cilji in prispevki naloge

Cilji diplomske naloge so predstaviti in poudariti aktualne tehnološke težave pri elektronskem prenosu finančnih sredstev, podati implementacijo plačilnega protokola Interledger in namenskega programa, imenovanega Orcus, ki ta protokol uporablja in predstavlja konektor v omrežju Interledger. Prav tako so cilji izvedba testnega plačila z uporabo več med seboj povezanih programov Orcus preko več različnih testnih glavnih knjig, integracija programa v obstoječ produkcijski plačilni sistem in izvedba plačila s pravimi sredstvi.
Rezultat diplomske naloge bosta analiza obstoječega tehnološkega stanja finančnih sistemov in delujoča implementacija konektorja za protokol Interledger.

1.3 Struktura naloge

V drugem poglavju so definirani in opisani koncepti, ki so pomembni za razumevanje osnov težave pošiljanja sredstev in protokola Inteledger. Podamo tudi opis obstoječih rešitev, ki vsaj v določeni meri rešujejo soroden problem. V tretjem poglavju natančno predstavimo arhitekturo protokola Interledger, opišemo posamezne komponente in način, kako se med sabo povezujejo v zaključeno celoto. Četrto poglavje je namenjeno predstavitvi naše implementacije protokola Interledger v programskem jeziku Golang, opisu integracije v obstoječo glavno knjigo in predlogom izboljšav naslednjih verzij konektorja. Zadnje poglavje zaključi diplomsko nalogo z mnenjem o mogočih načinih uporabe protokola v obstoječih

sistemih in poda načrt za nadaljnji razvoj konektorja.

Poglavje 2

Teoretično ozadje

V tem poglavju bomo na kratko predstavili osnovne pojme, potrebne za razumevanje naloge, pri čemer bomo najprej opisali temeljne finančne termine, nato pa razložili pojme, vezane na tehnologijo veriženja blokov in protokol Interledger. Poglavje bomo zaključili s predstavitvijo več obstoječih protokolov, ki že danes omogočajo podobne funkcionalnosti kot protokol Interledger, vendar v manjšem obsegu in z omejitvami glede na omrežje, na katerem lahko delujejo.

2.1 Razlaga pojmov

2.1.1 Glavna knjiga

Glavna knjiga (*ang. ledger*) je računovodska knjiga, v kateri so zapisani vsi prihodki in izdatki nekega računa za določeno valuto v kronološkem vrstnem redu, od najstarejše do najnovejše transakcije. Seštevek prihodkov in odhodkov računa nam pove trenutno finančno stanje tega računa. V zadnjem času so vedno bolj priljubljene porazdeljene glavne knjige (*ang. distributed ledger*) [6], v katerih so podatki replicirani na več neodvisnih strežnikih. S tem otežimo popravljanje zgodovine glavne knjige, ker nimamo več zgolj ene, ki bi predstavljala resnično stanje na računih.

2.1.2 Poravnava

Poravnava (*ang. settlement*) predstavlja proces premika sredstev med različnimi računi z namenom likvidacije terjatev, ki jih ima prejemnik do pošiljatelja. Ti premiki sredstev so nereverzibilni in nespremenljivi. V večini obstoječih bančnih sistemov se poravnave dogajajo po koncu delavnika. V Evropski uniji se za poravnave večinoma uporablja sistem TARGET2 [5], ki ga upravlja Evropska centralna banka.

2.1.3 Mikroplačila

Mikroplačila (*ang. micropayments*) so plačila v majhnih vsotah, po navadi v višini nekaj evrov, in se praviloma izvedejo na spletu. Definicija velikosti mikroplačila se razlikuje med različnimi ponudniki. Paypal na primer kot mikroplačila definira vsa plačila pod 10 € [13], medtem ko Interledger vsa v višini nekaj centov. Danes so taka plačila v večini predvsem bančnih sistemov praktično neizvedljiva. Stroški, ki nastanejo z eno transakcijo, so lahko večji od višine le-te. Kljub temu se je v zadnjih letih pojavilo več različnih sistemov, ki podpirajo taka plačila, ampak jim pri uporabnikih ni uspelo pridobiti prepoznavnosti in sprejetja. Predvidevamo, da je glavni razlog predvsem to, da sta integracija in uporaba takega sistema preveč kompleksni.

2.1.4 Veriga blokov

Veriga blokov (*ang. blockchain*) [16] je neskončno rastoči povezani seznam zapisov, imenovanih bloki. Vsak blok vsebuje časovni žig ustvaritve, podatke o transakcijah in zgoščeno vrednost predhodnega bloka. Verižno vključevanje te vrednosti omogoča, da lahko preverimo veljavnost celotne zgodovine verige, in preprečuje njeno popravljanje. Predstavlja nepopravljivo transparentno porazdeljeno javno knjigo in omogoča shranjevanje izmenjave sredstev med udeležencema na učinkovit in preverljiv način. Med najbolj prepoznavne implementacije te tehnologije spadata Bitcoin [10] in Ethereum [3].

2.1.5 Plačilni kanal

Plačilni kanal (*ang. payment channel*) je koncept, ki omogoča, da si soležnika informacije o medsebojnih terjatvah pošiljata neposredno. To idejo so popularizirali predvsem projekti s področja veriženja blokov, saj se lahko z uporabo plačilnega kanala izognemo zapisovanju vseh transakcij med soležnikoma na verigo blokov. To storimo le, kadar želita soležnika prenehati z izmenjevanjem terjatev. Takrat plačilni kanal zapreta in končno stanje sredstev soležnikov se zapiše v verigo blokov. Z uporabo plačilnega kanala zmanjšamo stroške transakcij, saj potrebujemo zgolj dve: eno za odpiranje in drugo za zapiranje kanala.

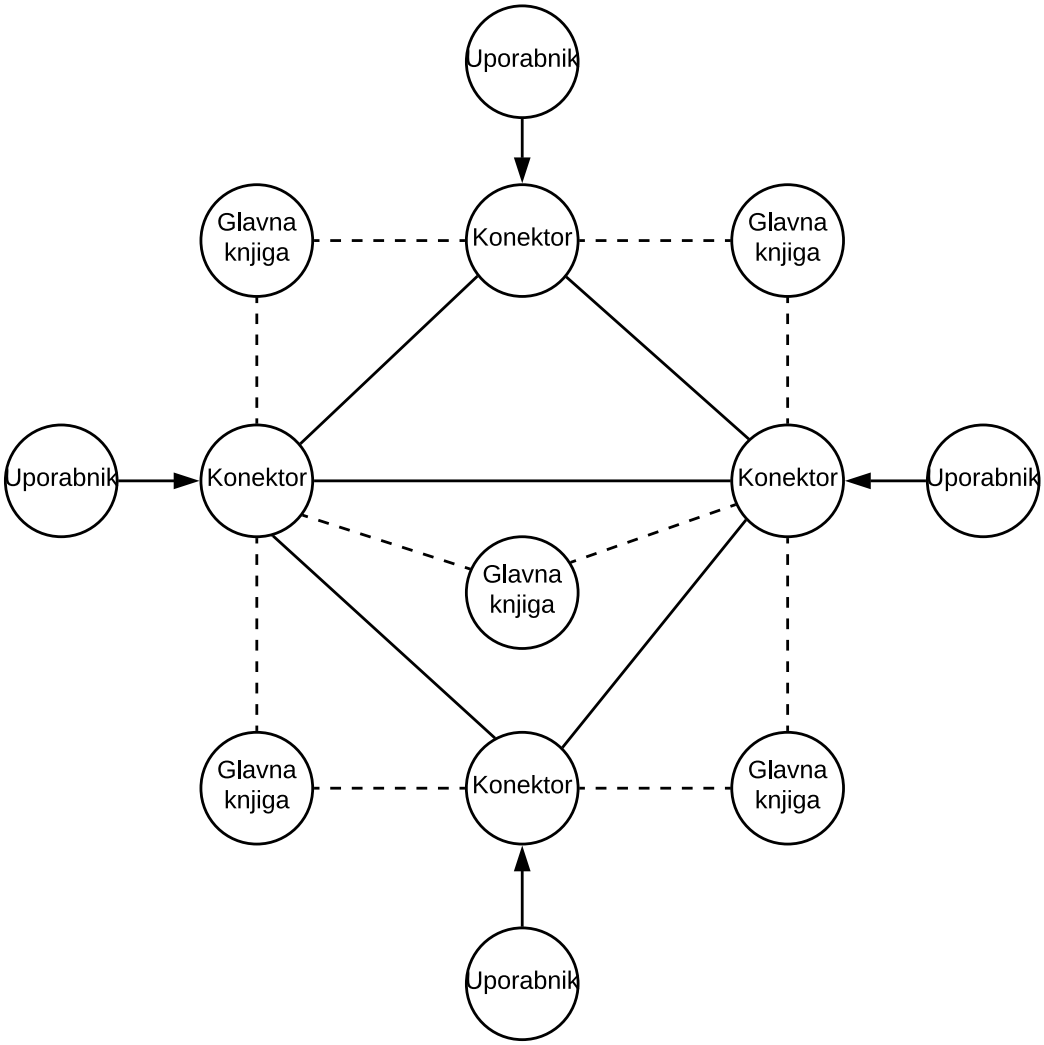
2.1.6 Konektor

Definicija konektorja (*ang. connector*) je zelo odvisna od področja, na katerem se uporablja. V naši nalogi bomo konektor definirali kot entiteto, ki je povezana z ostalimi podobnimi konektorji, z njimi tvori omrežje in preko katerega lahko uporabnik pošilja in prejema sredstva. Slika 2.1 prikazuje omrežje konektorjev, glavnih knjig in uporabnikov.

2.2 Obstoječi protokoli

Danes obstaja več različnih projektov, ki rešujejo problem povezovanja nepovezanih glavnih knjig in takojšnjega pošiljanja sredstev. Večina teh projektov je odprtokodnih in še v povojih, kar pomeni, da niso pripravljeni na vsakodnevno uporabo. Protokoli, ki jih uporabljajo finančne institucije, so veliko bolj dodelani, vendar niso odprtokodni in zato o njih vemo samo toliko, kot lahko izvemo iz dokumentacije.

V nadaljevanju opišemo enega izmed najbolj splošno sprejetih in uporabljenih protokolov v finančnem svetu kot tudi nekatere druge odprotokodne projekte, ki smo jih izbrali glede na popularnost in zrelost. Opis protokola Interledger bomo izpustili, saj mu bomo namenili celotno tretje poglavje.



Slika 2.1: Primer omrežja konektorjev

2.2.1 SWIFT

Organizacija Society for Worldwide Interbank Financial Telecommunication (SWIFT) je organizacija, ki so jo ustanovile večje svetovne finančne institucije in nudi omrežje, ki tem institucijam omogoča, da pošiljajo in prejemajo informacije o finančnih transakcijah na varen, standardiziran in zanesljiv način. V to omrežje je vključenih več kot tisoč svetovnih finančnih institucij, ki si dnevno izmenjajo več deset milijard sporočil. Ta sporočila ne predstavljajo dejanskega premika sredstev, ampak zgolj navodila za izvedbo nakazila, saj SWIFT ne omogoča poravnav. Finančne institucije izvedbo plačila dosežejo preko korespondenčne banke oz. računov, ki jih imajo ena pri drugi. SWIFT vsaki finančni instituciji, ki uporablja njihov sistem, dodeli edinstveno kodo po standardu ISO9362, ki jo poznamo pod imenom BIC (Business Identifier Code). Ta je sestavljena iz:

- štirih znakov, ki predstavljajo kodo institucije (na primer ABAN za Abanko);
- dveh znakov, ki predstavljata državo (na primer SI za Slovenijo);
- dveh znakov za lokacijo (na primer LJ za Ljubljano) in
- neobveznih treh znakov, ki predstavljajo podružnico banke.

SWIFT obenem tudi definira standardni format sporočil, ki si jih finančne institucije izmenjujejo. Gre za preprost format, ki je bil definiran za potrebe protokola SWIFT. Posamezno sporočilo je sestavljeno iz blokov, ki so definirani s parom zavitih oklepajev in delujejo po principu strukture *ključ-vrednost*. Primer takega sporočila je definiran v izseku kode 2.1.

Izsek kode 2.1: Primer veljavnega sporočila SWIFT

```
{1:F01MIDLGB22AXXX0548034693}
{2:I103BKTRUS33XBRDN3}
{3:{108:MT103}}
{4:
   :19A::DEAL//USD20,00
}
```

```
{5:{CHK:C77F8E009597}}
```

V splošnem je posamezno sporočilo definirano s petimi različnimi bloki, izmed katerih vsak predstavlja določen del navodila za izvedbo transakcije:

- osnovni blok glave (*ang. basic header block*) vsebuje podatke o izvoru sporočila,
- aplikacijski blok glave (*ang. application header block*) vsebuje podatke o tipu, cilju in smeri sporočila, pri čemer je smer lahko vhodna ali izhodna in se format sporočila glede na ta tip spremeni,
- uporabniški blok glave (*ang. user header block*) vsebuje izbirne podbloke, ki vsebujejo podatke o načinu procesiranja sporočila,
- tekstovni blok (*ang. text block*) podrobno opiše namen in strukturo sporočila. Format tega bloka je poseben in se ne sklada s formatom drugih. Sestavljen je iz zaporedja polj v formatu *:*{*tip*}{*opcija*}*:vsebina*{*crlf*}, kjer začetno dvopičje predstavlja začetek polja, *tip* dvomestno številko logičnega tipa sporočila (vsota, menjalni tečaj, datum in podobno) in izbirna vrednost *opcija* en znak za natančnejšo definicijo tipa. Primer kombinacije tipa in vrednosti je *19B*, ki predstavlja vsoto z valuto. Sledi komponenta *vsebina*, ki predstavlja strukturirano vsebino glede na kombinacijo tipa in vrednosti in ima strogo predpisan format. V primeru, da sta tip in vrednost *19B*, je format *:4!c//3!a15d* in pomeni, da dvopičju sledijo točno štirje znaki, dve poševnici, točno trije znaki in največ petnajst števk. Primer veljavnega bloka je naveden v 2.2.

  Izsek kode 2.2: Primer veljavnega polja v tekstovnem bloku sporočila

  ```
  :19A::DEAL//USD2973,54
  ```

- uporabniški končni blok (*ang. user trailer block*) lahko vsebuje poljubne podbloke. Primer takega bloka je {*5:*{*CHK:C77F8E009597*}}, kjer vrednost ključa CHK predstavlja zgoščeno vrednost sporočila za preverjanje avtentičnosti sporočila.

Sporočila se serializirajo in zapišejo v datoteko ter prenesejo naslovniku bodisi preko enostavnega varnega kanala za izmenjavo datotek s strukturirano vsebino, ki ga je razvil SWIFT, bodisi preko protokola FTP. Prejemnik sporočilo dekodira in na podlagi tipa in vrednosti ustrezno obdela. Podrobni opisi postopkov, povezanih z izmenjevanjem in obdelovanjem sporočil SWIFT, niso javno dostopni, saj je SWIFT popolnoma zaprto omrežje, dostopno zgolj licenciranim finančnim institucijam.

2.2.2 Lightning network

Bitcoin, leta 2017 svetovno najbolj znana in razširjena kriptovaluta, vsakomur omogoča pošiljanje sredstev brez potrebe po zaupanju vredni tretji osebi. Vendar se z naraščanjem priljubljenosti omrežja povečuje tudi število transakcij, ki jih uporabniki izvedejo. Zaradi fizične omejitve velikosti posameznega bloka velikokrat vseh transakcij, narejenih v periodi desetih minut, ni mogoče zapisati v blok velikosti 1 MB. Ob povprečni, približno 400 bajtov veliki transakciji (december 2017), to pomeni zgolj 2500 transakcij na blok oziroma 4 transakcije na sekundo. Za primerjavo, VISA je bila že leta 2016 sposobna obdelati več kot 65.000 transakcij na sekundo [18]. Zaradi pomanjkanja prostora v blokih rudarji, ki združujejo transakcije v bloke, dajejo prednost transakcijam, od katerih bodo več zaslužili. To posledično pomeni, da morajo uporabniki plačati višje stroške samo zato, da bo njihova transakcija vključena v naslednji blok. Kot odgovor na težavo Bitcoina pri obdelavi vedno večjega števila transakcij se je pojavil Lightning network. Predstavlja sloj nad omrežjem Bitcoin in omogoča instantno pošiljanje velikega števila transakcij, ne da bi se te vsakič zapisale v verigo, obenem pa obdrži vse dobre lastnosti Bitcoina. Lightning network uporabi dejstvo, da si lahko lastništvo nad nekim naslovom deli več uporabnikov oziroma da je za transakcijo iz takega naslova potrebnih več podpisov (*ang. multisignature*). Takšen naslov v kontekstu mikroplačil imenujemo plačilni kanal, odprtje tovrstnega kanala pa je v verigi blokov predstavljeno kot navadna transakcija. V splošnem velja, da se plačilni kanali odprejo med dvema uporabnikoma, kar pomeni, da sta za vsako transakcijo potrebna dva podpisa. Uporabnika se preko

poljubnega komunikacijskega kanala dogovorita, koliko sredstev bo vsak založil v plačilni kanal, in nato vanj ta sredstva pošljeta preko začetne transakcije za financiranje kanala (*ang. funding transaction*). Vsak od uporabnikov lahko na začetku drugemu uporabniku znotraj tega plačilnega kanala pošlje natančno toliko sredstev, kot jih je založil v kanal. Nato si lahko uporabnika začneta med seboj pošiljati sredstva na isti način, kot če bi to počela neposredno na glavni verigi. Uporabnik ustvari transakcijo, jo podpiše in pošlje prejemniku. Ta je ne pošlje v omrežje, temveč si jo shrani. S podpisom transakcije sta se strinjala o novem stanju sredstev, ki jih posameznik lahko porabi v kanalu. Na primer, Ana v začetku v kanal pošlje 0,5 BTC, isto naredi Bojan. Ana nato ustvari transakcijo, preko katere Bojanu pošlje 0.2 BTC. Ona jo podpiše in pošlje Bojana. S tem se stanje v kanalu spremeni za 0.2 BTC v korist Bojanu. Te transakcije si lahko pošiljata, dokler enemu od njiju ne zmanjka sredstev ali pa želi kanal preprosto zapreti. Takrat v omrežje pošljeta zadnjo transakcijo, ki predstavlja zadnje stanje. Sistem temelji na manj znanem dejstvu, da Bitcoin podpira preprost skriptni sistem, ki uporabnikom omogoča postavljanje pogojev, pod katerimi se lahko določena transakcija izvede. Na primer, sredstva se lahko na prejemnikov račun prenesejo po določenem pretečenem času. Primer take skripte je prikazan v izseku kode 2.3.

Izsek kode 2.3: Primer Bitcoin skripte, ki 1. januarja 2019 na naslov 1Mt2YT364znztTrCv1K6i6FUwrZKYNAnMW pošlje 1 BTC

```
1546300800 OP_CHECKLOCKTIMEVERIFY OP_DROP
    OP_DUP OP_HASH160 371
    c20fb2e9899338ce5e99908e64fd30b789313
    OP_EQUALVERIFY OP_CHECKSIG
```

2.2.3 Omrežje Cosmos

Ideja omrežja Cosmos je povezati več neodvisnih verig blokov, kot sta Bitcoin in Ethereum, na način, ki bi omogočil neposredni prenos sredstev med verigami z nizkimi transakcijskimi stroški. To naj bi dosegli s povezovanjem neodvisnih

verig blokov v osrednja vozlišča (*ang. hub*). Te verige se v omrežju Cosmos imenujejo cone (*ang. zone*) in se z vozliščem, v katerega so povezane, pogovarjajo preko protokola IBC (Inter-Blockchain Communication), ki je bil razvit za potrebe omrežja Cosmos. Osrednje vozlišče je veriga blokov, katere naloga je, da skrbi za sledenje premikanju sredstev in usklajevanju komunikacije med vsemi conami. To pomeni, da mora imeti osrednje vozlišče vedno aktualno stanje vseh sredstev na vseh conah, ki so vanj povezane. Na začetku naj bi obstajalo zgolj eno vozlišče, ki se mu bodo cone lahko priklopile. Vendar je arhitektura sistema zastavljena tako, da bo lahko kdor koli ustvaril novo vozlišče z novimi conami. Posledično naj bi čez čas obstajalo več vozlišč, ki naj bi se med seboj povezala v večje omrežje. To naj bi omogočalo interoperabilnost med različnimi verigami blokov.

Omrežje Cosmos uporablja validatorje za potrjevanje transakcij in sodelovanje pri procesu glasovanja izbire novih blokov. Za te dejavnosti so validatorji nagrajeni z žetoni Atom, ki so domorodna valuta v omrežju Cosmos. V primeru, da validator deluje proti omrežju, na primer da dvakrat potrdi isto transakcijo, je ta kaznovan in izgubi delež zastavljenih sredstev. To je velika motivacija za validatorje, da delujejo za dobrobit omrežja.

2.2.4 Polkadot

Polkadot je odprtokodni protokol in omrežje, namenjeno povezovanju glavnih knjig. Projekt je zasnoval Gavin Wood, ki je eden izmed avtorjev Ethereuma in denarnice Parity. Polkadot se poleg prenosa vrednosti posveča tudi prenosu podatkov med različnimi verigami, imenovanimi paraverige (*ang. parachains*). To pametnim pogodbam omogoča, da si lahko z drugimi pametnimi pogodbami na poljubnih platformah izmenjujejo podatke. Verige, ki se želijo priklopiti v omrežje Polkadot, morajo prevzeti njene obstoječe varnostne mehanizme in način sprejemanja soglasja. To pomeni, da se morajo verige prilagoditi, če se želijo vklopiti v omrežje. Drugi način predstavljajo mostovi, ki omogočajo, da v omrežje Polkadot priključimo tiste verige, ki se ne želijo prilagoditi. Ta način vključuje pisanje konektorjev, ki usklajujejo komunikacijo med sistemom Polkadot in drugo poljubno

verigo blokov, na primer Bitcoinom. Kot primer lahko navedemo, da bi moral Bitcoin spremeniti svoj način sprejemanja soglasja s *Proof of Work* na *Proof of Authority*, če bi želel delovati v omrežju na prvi način. Polkadot tudi ne definira virtualnih strojev, na katerih tečejo pametne pogodbe, ampak jih zgolj povezuje. Omrežje Polkadot uporablja idejo paraverig. To so vse verige blokov, ki so vključene v omrežje in uporabljajo isti sistem soglasja, kot ga nudi omrežje. Drugi koncept, ki ga uporablja Polkadot, je rele veriga (*ang. relay chain*), ki je osrednja veriga Polkadot in usklajuje vse paraverige v omrežju, kar prinaša visoko stopnjo centralizacije. Polkadot ima podobno kot ostali sistemi svoj žeton, imenovan Dot. Lastništvo teh žetonov uporabniku omogoči, da postane validator, ki ima glasovalno pravico glede dodajanja, urejanja in odstranjevanja paraverig. Validatorji potrjujejo bloke in jih zapisujejo v verigo, podobno kot rudarji v drugih verigah blokov. Zbiratelji (*ang. collators*) zbirajo nepotrjene transakcije na paraverigah in jih združujejo v bloke. Tretja vloga, ki jo lahko ima posamezni uporabnik v omrežju, je ribar (*ang. fisherman*). Ti v omrežju iščejo akterje, ki negativno vplivajo na omrežje (na primer lažejo o transakcijah) in jih poskušajo odstraniti. Validatorji, zbiratelji in ribiči so za svoje delo nagrajeni s provizijami od transakcij.

Poglavje 3

Protokol Interledger

Interledger [17] je protokol za pošiljanje paketov sredstev med različnimi plačilnimi omrežji in glavnimi knjigami. Prilagojen je za pošiljanje velike količine paketov z nizkimi zneski, zato mu pravimo tudi protokol za pošiljanje centov (*ang. penny-switching protocol*). Interledger lahko integriramo v kater okoli glavno knjigo, vključno s tistimi, ki niso bile ustvarjene za interoperabilnost, na primer bančne glavne knjige. Obenem je bil ustvarjen za uporabo v različnih uporabniških aplikacijah, ki bi jim takšne funkcionalnosti koristile.
Glavni cilji pri ustvarjanju protokola Interledger so bili nevtralnost, interoperabilnost, varnost, enostavnost in princip končnosti. Nevtralnost plačilnega protokola je ena najpomembnejših lastnosti, saj se s tem, ko protokol ni vezan na podjetje, valuto oziroma omrežje, rešimo pristranskosti in pridobimo zaupanje tako končnih uporabnikov kot tudi finančnih institucij, ki sodelujejo v omrežju. Interoperabilnost omogoča, da lahko v omrežje povežemo tudi glavne knjige, ki niso bile narejene za interakcijo z ostalimi podobnimi sistemi. Varnost je v plačilnem omrežju izjemnega pomena, saj lahko težave pri zagotavljanju le-te privedejo do finančnih izgub. Pošiljatelji, prejemniki in konektorji morajo biti zavarovani eden pred drugim in še posebej izolirani pred nevarnostmi, ki jih predstavljajo ostali udeleženci v omrežju. Konektorji ne smejo imeti možnosti kraje sredstev pošiljateljem in ti ne smejo imeti možnosti začasno zakleniti toliko konektorjevih sredstev, da bi to vplivalo na njihovo delovanje. Enostavnost je verjetno eden težje dosegljivih ci-

ljev vsakega protokola. Pomeni namreč stvari poenostaviti do tolikšne mere, da je implementacija v kateri koli drug sistem preprosta. S tem se zmanjša število razlogov za nestrinjanje med uporabniki, obenem pa se poveča splošno sprejetje uporabe protokola. Princip končnosti se zgleduje po trenutnem internetu in definira, da se vse implementacije funkcionalnosti, ki niso bistvene za delovanje jedra protokola Interledger prepustijo pošiljatelju in prejemniku.
Na Interledger smo se osredotočili iz dveh razlogov. Prvi razlog je, da za razliko od ostalih protokolov, kot je Lightning, ni vezan zgolj na eno valuto oziroma omrežje. Uporabnikom sistema omogoča, da lahko transakcije pošiljajo v kateri koli valuti, ki jo podpira vsaj en konektor v omrežju. Interledger tem protokolom ne predstavlja neposredne konkurence, temveč je komplementaren in na preprost način omogoči prenos sredstev med različnimi omrežji. Drugi razlog je, da obstaja zgolj ena implementacija, napisana v JavaScriptu, ki jo vzdržujejo avtorji protokola. Ta razlog nam je omogočil, da je naša implementacija prva neodvisna implementacija konektorja za protokol Interledger.

3.1 Arhitektura

V tem poglavju predstavimo bistvene komponente sistema Interledger, ki so definirani v uradni specifikaciji. Najprej opišemo sklad Interledger in opišemo posamezne plasti. Nadaljujemo s predstavitvijo bistvenih komponent protokola: z naslovi ILP, s paketi in transportno plastjo. Poglavje zaključimo s kritiko protokola in podamo predloge možnih izboljšav.

3.1.1 Sklad

Sklad Interledger je razdeljen v štiri plasti, podobno kot je razdeljen sklad TCP/IP. Vsaka plast ima svojo nalogo in je popolnoma neodvisna od drugih. V nadaljevanju te plasti od najnižje do najvišje definiramo in na kratko predstavimo.

Plast vtičnikov glavnih knjig

Plast vtičnikov glavnih knjig skrbi za usklajevanje prenosa sredstev za poravnavo med soležnikoma. Ta se sama odločita, kateri protokol za poravnavo bosta uporabljala, ker ga specifikacija Interledger ne definira.

Plast protokola Interledger

Plast protokola Interledger je odgovorna za posredovanje paketov Interledger med pošiljateljem in prejemnikom. Na tej plasti je definiran zgolj en protokol, ILP (*ang. Interledger protocol*). Ta protokol predstavlja jedro sklada Interledger in definira vse od standardnega formata naslovov ILP do formatov paketov.

Transportna plast

Transportna plast pošiljatelju in prejemniku transakcije Interledger omogoča, da se dogovorita o pogoju izvršitve. Protokol Interledger definira transportni protokol, imenovan STREAM, ki ga predstavimo v poglavju 3.1.4.

Aplikacijski nivo

Aplikacijska plast predstavlja najvišjo plast sklada protokola. Glavne naloge te plasti so iskanje računa prejemnika, izbira transportnega protokola in inicializacija transakcije z izmenjavo avtentikacijskih podatkov, kot je pogoj, pod katerim se ta transakcija lahko izvede. Primer aplikacijske plasti je protokol SPSP (Simple Payment Setup Protocol) [4], ki je nastal v okviru protokola Interledger. Predstavlja protokol za izmenjavo plačilnih podatkov med pošiljateljem in prejemnikom ter uporablja protokol Webfinger [8] za odkrivanje soležnika in STREAM [3.1.4] kot transportni protokol.

3.1.2 Naslovi

Naslovi Interledger pomagajo pri usmerjanju plačil med različnimi konektorji v omrežju do prejemnika in imajo podobno vlogo kot naslovi IP na omrežnem ni-

voju interneta. Ti naslovi, čeprav so videti podobno kot URL-ji, niso namenjeni neposredno uporabnikom, temveč zgolj konektorjem.

Zgradba naslova

Vsak naslov je zgrajen iz več delov, ki so vnaprej definirani na podlagi preprostih pravil in med seboj ločeni s piko. Vsebujejo lahko zgolj črke, številke, podčrtaj, tildo in vezaj. Vsi ostali znaki so prepovedani. Prvi del mora biti vedno razporeditvena shema, ki je lahko zgolj ena izmed sledečih:

- **g**, ki predstavlja globalno shemo in je namenjena uporabi pri naslovih za pošiljanje in prejemanje sredstev. To shemo uporablja večina konektorjev v omrežju in nima osrednje avtoritete, ki bi izdajala naslove, kot je na primer IANA za internetne domene,
- **private**, ki predstavlja zasebno shemo in je namenjena uporabi pri naslovih za pošiljanje in prejemanje sredstev v zasebnem omrežju,
- **example**, ki se uporablja v primerih in dokumentaciji,
- sheme **test** in **test**{**1-3**}, ki se uporabljajo za povezovanje v javna testna omrežja ali v integracijskih testih programa,
- shema **local** je veljavna zgolj znotraj konteksta konektorja, podobno kot velja za “link-local” naslove IP,
- shema **peer** je namenjena uporabi ob odpiranju plačilnih kanalov z drugimi konektorji in
- **self**, ki predstavlja povratno povezavo na trenutni konektor.

Razporeditveni shemi lahko sledi eden ali več delov, imenovanih okolice, ki definirajo, kje deluje konektor. Cilj okolic je, da lahko na preprostejši način povežemo konektorje, ki so si geolokacijsko blizu. Konektor z okolico *eu* se bo najverjetneje raje povezal z drugim konektorjem, ki ima to okolico, kot s tistim,

ki ima okolico *us*, saj bo imel do njega verjetno krajši RTT. To pomeni, da bo konektor lahko hitreje pošiljal transakcije in bo povezava zaradi lokalnosti in krajše fizične razdalje tudi bolj zanesljiva. Treba je poudariti, da je izbor okolice popolnoma odvisen od upravljavca konektorja in mu nič ne preprečuje, da si ne bi izbral okolice *eu*, čeprav je v ZDA. Delom okolice sledi vsaj en del, ki predstavlja unikaten identifikator računa, ki ga upravlja konektor. Zadnji del naslova ima lahko še več s tildami pripetih dodatnih delov, imenovanih interakcije. Predstavljajo metapodatke konektorju in mu olajšajo delo pri pošiljanju. Primer interakcije je TRR prejemnika, če želimo, da prejemnik sredstva prejme neposredno na bančni račun.

3.1.3 Paketi

Specifikacija Interledger definira štiri različne tipe paketov, ki se pošiljajo med konektorji in predstavljajo osnovne gradnike komunikacije. V nadaljevanju pakete opišemo in razložimo njihovo vlogo v sistemu.

Paket za pripravo transakcije

Paket za pripravo transakcije se uporablja za pripravo celotne verige konektorjev, ki bodo sodelovali v transakciji. Vsak konektor, ki ta paket dobi v plačilnem kanalu, preko katerega je paket dobil, zaklene toliko sredstev, kot je navedeno v tem paketu. Nato ta paket posreduje naprej naslednjemu konektorju v verigi. V paketu so zapisane naslednje vrednosti:

- **amount** predstavlja količino sredstev, ki jih pošiljatelj pošilja prejemniku. Višina je predstavljena v najmanjši enoti valute, v kateri se pošilja,
- **expiresAt** predstavlja čas ,do katerega se mora transakcija izvesti, preden velja za neveljavno,
- **executionCondition** je pogoj, pod katerim se transakcija lahko izvede. Je zgoščena vrednost izpolnitve pogoja (*ang. image*), ki jo pozna zgolj prejemnik,
- **destination** predstavlja prejemnikov naslov ILP in

- **data** so poljubni podatki, ki jih lahko pošiljatelj pošlje prejemniku.

Paket za izvedbo transakcije

Paket za izvedbo transakcije je del drugega koraka izvedbe transakcije. S tem paketom konektor konektorju, od katerega je dobil paket za pripravo transakcije, dokaže, da se je transakcija res izvedla. V paketu sta zapisani naslednji vrednosti:

- **fulfillment** je vrednost izpolnitve pogoja (*ang. preimage*). Prejemnik izračuna zgoščeno vrednost te vrednosti in jo primerja s pogojem, pod katerim se transakcija lahko izvede in
- **data** so poljubni podatki, ki jih lahko prejemnik pošlje nazaj pošiljatelju.

Paket za zavrnitev transakcije

Paket za zavrnitev transakcije je namenjen prekinitvi koraka priprave ali izvedbe transakcije. V paketu so zapisane naslednje vrednosti:

- **code** je tričrkovna oznaka napake iz seznamov možnih napak A.1, A.2 in A.3,
- **triggeredBy** je naslov ILP konektorja oziroma plačilnega kanala, ki je zavrnil transakcijo,
- **message** je človeško berljiv razlog za napako in
- **data** so poljubni podatki, ki se lahko pošljejo ob zavrnitvi transakcije.

3.1.4 Transport

V specifikaciji protokola Interledger so avtorji definirali tudi osnovni transportni protokol, imenovan STREAM. Je protokol, ki omogoča vzpostavljanje tokov plačil in podatkov med soležnikoma. Nudi zelo širok nabor funkcionalnosti, ki jih lahko aplikacije, ki potrebujejo podporo za plačila, izkoriščajo. Med pomembnejše funkcionalnosti spadajo:

- dvosmerno pošiljanje sredstev in podatkov preko protokola ILP,
- deljenje velikih transakcij v več manjših in sestavljanje posameznih delov nazaj, podobno kot deluje protokol TCP,
- multipleksiranje več tokov preko ene odprte povezave,
- nadzor hitrosti pretakanja podatkov oziroma sredstev in s tem preprečevanje preobremenjenosti,
- avtentikacija in enkripcija paketov ILP in
- generiranje in preverjanje izpolnitve pogojev za izvedbo transakcij.

3.1.5 Bilateralni prenosni protokol

Izmenjava paketov ILP med dvema konektorjema lahko poteka na več različnih načinov. Pakete si lahko izmenjata preko zahtev HTTP/S, kar pomeni, da morata oba, če si želita pakete pošiljati v obe smeri, imeti zagnan strežnik HTTP/S. Težava nastane, ko je eden izmed konektorjev za NAT-om, ker to pomeni, da je brez ustrezne konfiguracije nedostopen. To težavo lahko premostimo z uporabo dolgotrajnih povezav s pomočjo protokola Websocket. Na ta način mora biti javno dostopen zgolj eden izmed konektorjev.
Protokol Websocket ne nudi mehanizma za povezovanje zahtev in odgovorov, temveč zgolj uokvirjanje podatkov. Bilateralni prenosni protokol (*ang. Bilateral Transfer Protocol – BTP*) doda ta manjkajoči člen.
BTP uporablja koncept zahteve in odgovora, podobno kot protokol HTTP. Za vsako zahtevo pričakuje odgovor, tudi če je to zgolj potrdilo prejetja. Vsaka zahteva mora imeti identifikator, ki jo enolično predstavi. Prejemnik mora na zahtevo odgovoriti z odgovorom, ki mora vsebovati identifikator zahteve, da lahko pošiljatelj odgovor poveže s poslano zahtevo.
Pri procesiranju zahtev in odgovorov lahko pride do zapletenih primerov:

- V primeru, da konektor dobi nepričakovan paket, torej odgovor ali napako, nanj ne sme odgovoriti.

- V primeru, da konektor dobi neberljiv paket, nanj ne sme odgovoriti. Neberljiv paket je tisti paket, ki je strukturno nepravilen; na primer zapisana dolžina se ne ujema z dejansko dolžino vsebine.

Pravilno obdelovanje teh robnih primerov je ključno, da konektor prepreči neskončne zanke. V primeru, da dobimo nepričakovan paket in nanj odgovorimo z napako, bo ta paket za prejemnika prav tako nepričakovan in bo nanj ponovno odgovoril z napako. To se nadaljuje v nedogled in lahko povzroči nezmožnost pošiljanja ostalih paketov.

Tipi sporočil

Protokol definira štiri različne tipe sporočil, ki jih lahko pošiljamo preko odprte povezave:

- sporočilo **message** predstavlja zahtevo in je namenjeno pošiljanju splošnih informacij soležniku. Te informacije so lahko na primer posodobitve usmerjevalne tabele soležnika: s kom je ta vzpostavil povezavo, s kom jo je prekinil in katere obstoječe poti so še vedno aktivne. Prejemnik lahko na to sporočilo odgovori zgolj s sporočilom napake ali prejema. Odgovor s katerim koli drugim tipom je neveljaven.

- sporočilo **error** predstavlja tip sporočila "odgovor" in je namenjeno pošiljanju informacij o napaki, ki se je zgodila ob obdelavi pošiljateljeve zahteve. Koda napake in ime sta definirana v dodatku A.4.

- sporočilo **response** predstavlja tip odgovora, ki je namenjen potrjevanju prejetja. Vsebuje zgolj sklic na zahtevo, ki jo potrjuje.

- sporočilo **transfer** predstavlja tip zahteve, ki je namenjena pošiljanju dokaza o izvedbi dela transakcije, zahteve po poravnavi plačilnega kanala ali katere koli druge informacije o prenosu sredstev na kanalu. Prejemnik lahko

na to sporočilo odgovori zgolj s sporočilom napake ali potrditvijo prejema. Odgovor s katerim koli drugim tipom je neveljaven.

Struktura paketa BTP

Vsako sporočilo, ki ga želimo poslati preko protokola BTP, ima enako strukturo in je zakodirano s kodirnim algoritmom OER. Prvo polje v sporočilu je velikosti enega bajta in predstavlja številski tip sporočila. Response ima vrednost 1, Error 2, Message 6 in Transfer vrednost 8. Vsi vmesni manjkajoči tipi niso definirani, temveč zgolj rezervirani, če se v prihodnje pojavi potreba po dodatnih tipih. Zatem sledi identifikator sporočila dolžine štirih bajtov, ki omogoča povezovanje zahtev in odgovorov. Pošiljatelj mora poskrbeti, da nikoli ne pošlje dveh paketov z istim identifikatorjem, saj sicer ne bo mogel pravilno povezati odgovora z zahtevo. Tretje polje je dolžina podatkov, ki jih pošiljamo v paketu. Zadnje polje so dejanski podatki, ki se pošiljajo.

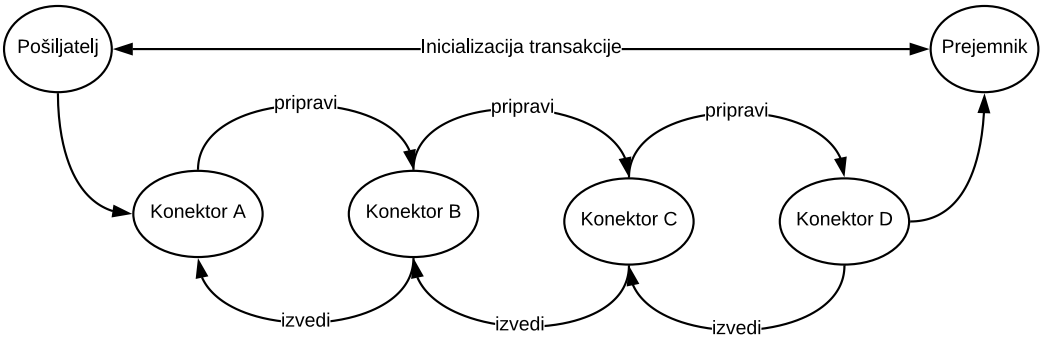
3.2 Izvedba transakcije

Celoten potek izvedbe transakcije, od inicializacije do izvedbe, poteka v več korakih, ki se morajo izvršiti zaporedoma. Slika 3.1 prikazuje potek izvedbe transakcije. Prvi korak je inicializacija transakcije, ki se zgodi izven protokola Interledger. Pošiljatelj in prejemnik se dogovorita o vsoti, valuti, omrežju in pogoju izvedbe transakcije. Pošiljatelj nato izvede transakcijo na svojem konektorju. Ta sestavi paket za pripravo transakcije glede na informacije, ki mu jih je posredoval pošiljatelj. Nato v svoji usmerjevalni tabeli poišče konektor, ki pozna pot do prejemnika, in mu posreduje paket. Prejemnik tega paketa preveri, ali ima pošiljatelj na plačilnem kanalu dovolj prostih sredstev za transakcijo. Če jih ima, mu zmanjša količino prostih sredstev za velikost paketa. V nasprotnem primeru transakcijo zavrne in zaključi z obdelovanjem. Konektor nato v svoji usmerjevalni tabeli poišče konektor, ki pozna pot do prejemnika. Če poti ne najde, z obdelovanjem zaključi in napako sporoči pošiljatelju. V prejetem paketu priredi višino sredstev tako,

da vsoto pomnoži z menjalnim tečajem za valuto plačilnega kanala, preko katerega je prejel paket od pošiljatelja, in valuto, za katero ima odprt plačilni kanal s konektorjem, ki mu bo posredoval paket. Od vsote odšteje še stroške transakcije, kar predstavlja njegov dobiček. Popravi tudi čas do izteka transakcije, da si zagotovi dovolj časa, da lahko prejeto izpolnitev transakcije posreduje nazaj pošiljatelju. Nato paket posreduje naprej. Vsi nadaljnji konektorji v verigi ponovijo isti postopek, dokler paket ne pride do konektorja, ki pozna prejemnika. Ta konektor paket posreduje prejemniku, ki se odloči, ali bo transakcijo sprejel ali zavrnil. Če se odloči za sprejem transakcije, konektorju posreduje potrditev izvršitve. Ta potrditev ustreza pogoju, s katerim sta pošiljatelj in prejemnik inicializirala transakcijo. Konektor posreduje potrditev nazaj konektorju, od katerega je prejel paket za transakcijo. Konektor, ki je prejel potrditev, to potrditev nato preveri s pogojem. Če se ujemata in je potrditev dobil pred iztekom transakcije ter ima dovolj časa, da potrditev posreduje nazaj pošiljatelju, to pomeni, da je bil ta del transakcije uspešen in se lahko zaklenjena sredstva v njunem plačilnem kanalu prestavijo s pošiljateljevega na prejemnikov račun. V nasprotnem primeru konektor transakcije ne sprejme in jo pusti, da poteče. V tem primeru se sredstva v plačilnem kanalu po preteku transakcije samodejno sprostijo. Enak postopek posredovanja izvršitve transakcije izvedejo vsi konektorji v verigi, dokler potrditev ne prispe do pošiljatelja. Med celotnim procesom lahko pride do težav, kot je nezmožnost posredovanja potrditve izvršitve transakcije nazaj pošiljatelju. Te težave in možne rešitve predstavimo v naslednjem podpoglavju.

3.3 Zmanjševanje tveganj procesiranja transakcij

Konektorji so pri obdelavi transakcij izpostavljeni določenemu tveganju, vendar v zameno za to izpostavljenost od vsake transakcije generirajo dobiček. Konektor se mora med izvedbo plačila najprej zadolžiti pri konektorju, ki mu posreduje transakcijo, preden je upravičen do sredstev konektorja, od katerega je prejel transakcijo. Šele ko prejme potrditev izvršitve, lahko od pošiljatelja terja sredstva, saj ima takrat potrdilo, da je transakcijo res posredoval naprej in se je do določenega



Slika 3.1: Potek izvedbe transakcije

dela že izvršila. Za dokazovanje in terjanje sredstev od pošiljatelja ima na voljo zgolj določeno časovno okno. Tak potek izvajanja transakcije ima lahko resne posledice, če konektor ne deluje pravilno.
V nadaljevanju predstavimo tri različne tipe tveganj, ki so jim konektorji lahko izpostavljeni, in možne ukrepe za preprečevanje škode.

3.3.1 Nezmožnost izpolnitve transakcije

Konektorju eno največjih tveganj predstavlja nezmožnost izpolnitve vhodne transakcije po tem, ko se je izhodna transakcija že izvršila. V tem primeru konektor ostane brez sredstev, ki jih je poslal soležniku, ki je naslednji v verigi, ampak ni dobil sredstev od soležnika, ki je njegov predhodnik v verigi konektorjev za transakcijo. Do te situacije lahko pride, ko transakcija poteče, preden uspe konektor posredovati izpolnitev transakcije nazaj predhodnemu konektorju.
Konektor lahko posledice te težave olajša tako, da

- se povezuje zgolj s konektorji, ki so splošno znani, da so zanesljivi in odzivni,
- neprestano ocenjuje količino časa, potrebnega za izvedbo transakcije, in že na začetku zavrne vse transakcije, ki bi ga lahko ogrozile,
- poskrbi za preprečevanje napadov DDoS, ki bi ga lahko ogrozili s stališča

omrežnega prometa in posledično nezmožnostjo posredovanja izvršitev transakcij,

- daje prednost posredovanju izvršitev transakcij pred vsemi ostalimi dejavnostmi, kot je posredovanje posodobitev poti,
- uporablja zaprta omrežja, do katerih imajo dostop zgolj ostali zaupanja vredni konektorji.

3.3.2 Izčrpavanje likvidnosti

Konektorji morajo med delovanjem skrbeti, da imajo v vsakem trenutku na voljo dovolj sredstev v plačilnih kanalih, da lahko sprejemajo nove transakcije, saj z njimi služijo. Vendar lahko pride do situacije, v kateri konektor prostih sredstev nima na voljo. To se lahko zgodi, če napadalec izvaja transakcije, ki so neizpolnljive. Na primer, napadalec sredstva pošilja sam sebi, vendar nobene transakcije kot prejemnik ne potrdi. Te transakcije bodo čez čas potekle, vendar bodo sredstva do takrat zaklenjena v plačilnih kanalih vseh konektorjev v verigi.
Konektor lahko posledice te težave olajša tako, da

- neprestano nadzoruje količino zaklenjenih sredstev za določenega pošiljatelja in zavrne vse njegove transakcije, ki bi povzročile, da bi vsota njegovih aktivnih transakcij presegla določeno mejo,
- daje prednost transakcijam z manjšo količino sredstev in tistim, ki gredo do bolj zanesljivih konektorjev,
- ne procesira transakcij za splošno znane problematične pošiljatelje in prejemnike.

3.3.3 Volatilnost menjalnih tečajev

Konektor se ob inicializaciji transakcije zaveže, da bo zagotovil določen menjalni tečaj. Vendar se lahko ta menjalni tečaj med izvajanjem transakcije spremeni. V

primeru, da se menjalni tečaj poslabša, konektor razliko pokrije z lastnimi sredstvi.
Konektor lahko posledice te težave olajša tako, da

- v stroških transakcije, ki jih zasluži ob vsaki transakciji, upošteva tudi dejstvo, da se lahko menjalni tečaj obrne proti njemu,
- daje prednost transakcijam s krajšimi časovnimi omejitvami, ker predstavljajo manjše tveganje, da bi se menjalni tečaj spremenil.

3.4 Omrežje konektorjev

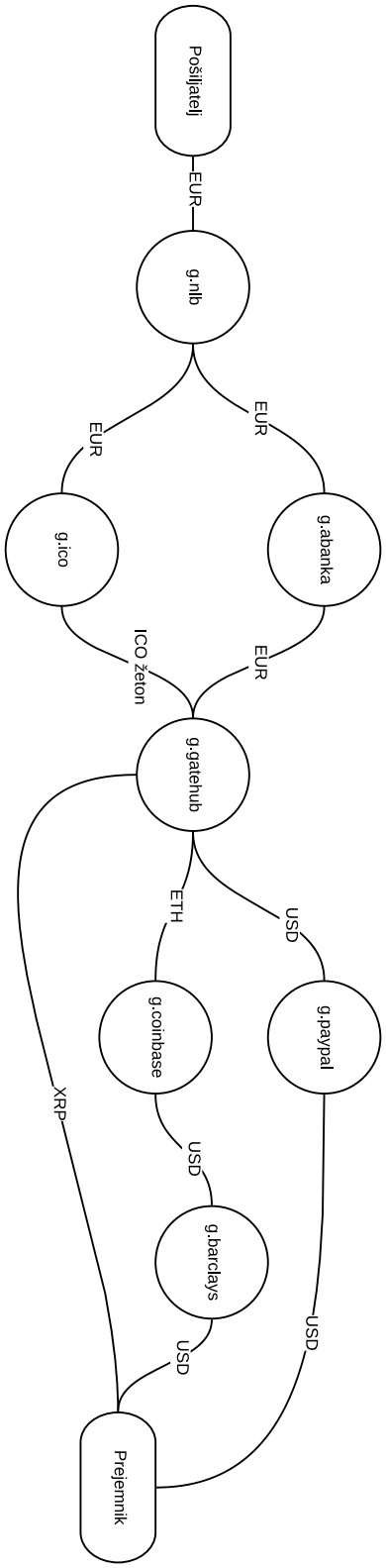
Konektorji se med seboj povežejo in s tem ustvarijo lastno omrežje, v katerem lahko vsak pošlje sredstva komur koli v omrežju. Primer takega enostavnega omrežja predstavlja slika 3.2. V prikazanem omrežju lahko pošiljatelj prejemniku pošlje USD ali XRP, za to pa porabi EUR. Med prenosom od pošiljatelja do prejemnika se lahko ta sredstva pretvorijo v žeton ICO in Ether, če je pot preko teh valut najprimernejša.

3.5 Predlogi izboljšav protokola

Med branjem in interpretiranjem specifikacije protokola Interledger smo naleteli na precej nejasnosti, večinoma povezanih s podrobnostmi implementacije posameznih delov protokola. Nerešeni del težav v nadaljevanju definiramo, opišemo in predlagamo možne rešitve.

3.5.1 Definicija interakcij

Specifikacija naslova ILP interpretacije interakcij naslova ne definira dovolj natančno. Specificira, da so interakcije vsi deli za simbolom ~. Vendar regularni izraz za validacijo dela dovoljuje uporabo tega simbola znotraj ostalih delov. To



Slika 3.2: Primer enostavnega omrežja

Del	Prvi način	Drugi način
razporeditvena shema	g	g
okolica oz. račun	acme	acme, **~bo**
interakcije	**~bo**, ~i	~i

Tabela 3.1: Tabela interpretacije interakcij

pomeni, da pri trenutni specifikaciji ne moremo dovolj natančno določiti, kje se začnejo interakcije, kar lahko povzroči težave, saj so interakcije namenjene pošiljatelju in prejemniku. Primer dvoumne interpretacije interakcij je prikazan v tabeli 3.1. V primeru, da prejemnik od konektorja dobi napačne interakcije, lahko ta transakcijo zavrne, čeprav jih je pošiljatelj pravilno pripel. Naslov *g.hercog.~foo.~bar* lahko interpretiramo na vsaj dva načina. V prvem načinu vse za prvim simbolom ~ tretiramo kot interakcije, namenjene prejemniku, v drugem primeru pa vse za drugo pojavitvijo. Rešitev te težave je enostavna. Specifikacija se spremeni na način, da simbol ~ ni dovoljen v posameznih delih, temveč je namenjen zgolj za pripenjanje interakcij na konec naslova. V primeru, da se zgledujemo po prvi interpretaciji naslova, bi nov naslov v skladu s popravljeno specifikacijo izgledal *g.hercog~foo~bar*.

3.5.2 Globalna bijekcija naslovov

Specifikacija protokola ne definira, da morajo biti vsi naslovi v omrežju unikatni in da lahko pripadajo zgolj enemu konektorju. Avtorji protokola te omejitve niso dodali, saj ne vidijo težav v tem, da bi si dva konektorja naslov delila in da obstaja zelo majhna verjetnost, da bi prišlo do težav pri razločevanju takih konektorjev. To pomeni, da naslovi ILP niso bijektivni, kar lahko predstavlja varnostno grožnjo. Predpostavimo, da obstajata dva konektorja, A in B, ki si delita naslov *g.eu.foo.bar*. Ko eden izmed njiju vzpostavlja povezavo s tretjim konektorjem C, ta ne more vedeti, ali je to konektor A ali B. To težavo lahko rešimo s pomočjo globalnega registra naslovov ILP, predstavljenega v poglavju 4.8.2.

3.5.3 Možnost prejema točne vsote

Trenutna specifikacija ne omogoča pošiljanja točne vsote prejemniku. To pomeni, da pošiljatelj ne more vnaprej natančno vedeti koliko mora poslati, da bo prejemnik prejel na primer točno 10 €, lahko zgolj oceni, koliko bo stala pot, in pošlje toliko več. Vendar nič ne zagotavlja, da do prejemnika ne bo prišlo manj, kot je zahteval. Ta se nato lahko odloči, da bo prejel premalo in transakcijo zavrne. Avtorji kot rešitev te težave predlagajo, da pošiljatelj prejemniku najprej pošlje neizpolnljivo transakcijo za želeno vsoto in počaka, da mu prejemnik sporoči, koliko bi dejansko prejel. Pošiljatelj nato lahko izračuna, koliko bi moral poslati, da bi prejemnik prejel točno vsoto. Ta rešitev podvoji število transakcij v omrežju in razpolovi število transakcij, ki prestavljajo sredstva. Konektorji, ki sodelujejo pri transakcijah, od teh neizpolnljivih transakcij ne zaslužijo ničesar, kar je zelo slaba motivacija za sodelovanje v sistemu.

Kot rešitev predlagamo propagacijo menjalnih tečajev skupaj z informacijami o aktualnih poteh. Ko konektor dobi informacije o novih poteh, lahko nov menjalni tečaj izračuna tako, da prejetega pomnoži s svojim za določeno valuto in nato to informacijo o novi poti posreduje naprej ostalim soležnikom. Na ta način ustvari kumulativni menjalni tečaj, ki je veljaven toliko časa, kot je veljaven najmanj veljaven menjalni tečaj. Izsek kode, ki poskrbi za preračunanje menjalnega tečaja, je prikazan v 3.1.

Vendar ima tudi ta rešitev nekaj pomanjkljivosti. V primeru, da ima pot do končnega konektorja veliko skokov, to pomeni veliko različnih menjalnih tečajev z različnimi časovnimi obdobji veljavnosti. Lahko se zgodi, da veljavnost menjalnega tečaja poteče, še preden se ta propagira do zadnjega konektorja. To lahko rešimo na način, da vsi konektorji v omrežju zagotavljajo zaklenjene menjalne tečaje za časovno obdobje več sekund.

Izsek kode 3.1: Primer izračuna novega menjalnega tečaja

```
0   func CalcExchangeRate(p types.PaymentChannel, r types.RouteUpdate) types.Rate
        {
1       n := getExchangeRate(p.Currency, r.SourceCurrency)
2       rt := n.Rate.mul(r.Rate.Rate)
3
4       e := n.Rate.ExpiresAt
5       if e.After(n.ExpiresAt) {
6           e = n.ExpiresAt
7       }
8
9       return types.Rate {
10          SourceCurrency: p.Currency,
11          TargetCurrency: t.TargetCurrency,
12          Rate: rt,
13          Provider: n.Provider,
14          ExpriesAt: e,
15      }
16  }
```

Poglavje 4

Razvoj konektorja Orcus

Cilj tega poglavja je predstaviti zasnovo in implementacijo arhitekture konektorja za protokol Interledger, ki smo ga poimenovali Orcus. Najprej posebno pozornost namenimo podrobni predstavitvi posameznih storitev, ki predstavljajo jedro konektorja, in opišemo njihovo medsebojno delovanje. Nadaljujemo z opisom vtičnikov in njihove vloge v sistemu. Sledi opis postopkov priprave, prevajanja, konfiguracije in poganjanja konektorja. Na koncu poglavja podamo kritiko naše implementacije, predloge izboljšav in načrt za nadaljnje delo.

4.1 Uporabljene tehnologije

V tem poglavju na kratko predstavimo uporabljene tehnologije in razložimo, zakaj smo se odločili, da jih uporabimo za razvoj konektorja Orcus.

4.1.1 Golang

Golang je odprtokodni programski jezik, ki je nastal leta 2009 pod okriljem Googla. Je statično tipiziran, preveden C-jevski tip jezika z vgrajenim samodejnim upravljanjem s pomnilnikom in preprostim upravljanjem s sočasnim izvajanjem operacij. Ta jezik smo za razvoj izbrali zaradi dobre podpore omreženja s strani standardne knjižnice, preglednosti kode, vgrajenega sočasnega izvajanja in lastnih večletnih izkušenj.

4.1.2 gRPC

gRPC (gRPC Remote Procedure Calls) je odprtokodni sistem za klice za oddaljene postopke (*ang. remote procedure call*), ki ga je razvil Google in nudi lažjo komunikacijo s strežnikom. Kot transportni protokol uporablja HTTP/2, omogoča dvosmerno pretakanje podatkov v realnem času, avtentikacijo odjemalca in samodejno prekinjanje klicev po določenem času, če se klic še ni zaključil. Je zelo dobro podprt v vseh bolj splošno uporabljenih programskih jezikih. Iz teh razlogov smo se odločili, da to tehnologijo uporabimo pri razvoju.

4.1.3 Protocol buffers

Protocol buffers oziroma protobuf je Googlov format za izmenjevanje podatkov. Je nevtralen glede na programski jezik, v katerem se uporablja, in omogoča hitrejšo serializacijo in deserializacijo podatkov v primerjavi z ostalimi podobnimi formati, kot sta XML in JSON. V primerjavi z XML-jem sta serializacija in deserializacija 60-krat hitrejši, serializirani podatki pa v povprečju zavzamejo 6-krat manj prostora [2]. Ta dejstva so nas prepričala, da smo ga uporabili za glavni format izmenjave sporočil.

4.2 Storitve

Vse storitve so definirane kot vmesniki in so zato popolnoma neodvisne z vidika implementacije poslovne logike. Med seboj komunicirajo zgolj preko funkcij, definiranih v teh vmesnikih, kar jim omogoča, da razkrivajo nadvse majhno podmnožico funkcij in visoko stopnjo zmožnosti testiranja. Taka zasnova ustvari zaključen abstraktni sistem. S tem drugim razvijalcem omogočimo, da lahko na zelo preprost način poljubno implementirajo kateri koli del sistema, in ta bi moral delovati popolnoma enako kot z drugo implementacijo neke storitve. S tem dosežemo visoko stopnjo neodvisnosti in avtonomnosti posameznih storitev.
V naslednjih podpoglavjih opišemo posamezne storitve, njihove glavne naloge in razložimo posamezne metode storitev.

4.2.1 Register

Register je storitev, ki komunicira z zunanjim registrom naslovov ILP. Njegova naloga je tako rezervacija želenega naslova ILP v oddaljenem registru kot tudi razrešitev poljubnega naslova ILP v naslova HTTP in BTP. Ta naslov lahko nato konektor uporabi za vzpostavitev povezave BTP ali inicializacijo plačila z oddaljenim konektorjem. Ker je register definiran kot vmesnik, ga lahko implementiramo tudi kot zelo preprost lokalni register, ki podatke shranjuje v datoteki, podobno kot deluje */etc/hosts* v operacijskem sistemu Linux. Definicija vmesnika registra je zapisana v izseku kode 4.1.

Izsek kode 4.1: Definicija vmesnika za implementacijo registra

```
0  type Registry interface {
1      Run(ctx context.Context) error
2      Register(ctx context.Context, addr types.Address) error
3      Resolve(adr types.Address) (btp, api string, err error)
4  }
```

4.2.1.1 Opis metod

Run

Metoda poskusi vzpostaviti povezavo z oddaljenim registrom naslovov ILP. Ob uspešni vzpostavitvi povezave zažene periodično preverjanje dosegljivosti registra. V primeru težav vrne napako in zaključi z izvajanjem.

Register

Metoda poskusi v oddaljenem registru ILP shraniti naslova HTTP in BTP na želeni naslov ILP. Podatki o naslovih HTTP in BTP se podajo konstruktorski funkciji, ker sta za vse naslove ILP identična. V primeru napake, da je na primer določen naslov ILP že registriran in ne pripada konektorju, vrne napako.

Resolve

Metoda poskusi pridobiti naslova BTP in HTTP za določen naslov ILP iz oddaljenenega registra. V primeru, da naslova ne more razrešiti, vrne napako.

4.2.1.2 Oddaljeni register

Oddaljeni register konektorjev omogoča prevedbo naslovov ILP v naslove HTTP in BTP, podobno kot DNS omogoča prevedbo URL-jev v naslove IP. S tem konektorju olajša delo, da lahko operira zgolj z naslovi ILP kot primarnimi naslovi v omrežju. Naša implementacija zunanjega registra je izjemno preprosta in omogoča zgolj shranjevanje in pridobivanje podatkov. Pri shranjevanju ne podpira dokazovanja istovetnosti oziroma lastništva konektorja nad nekim naslovom ILP, ampak deluje po principu "kdor prvi pride, prvi melje". Trenutna implementacija zadostuje potrebam našega sistema, vseeno pa v zadnjem delu tega poglavja podamo predlog varnejše in učinkovitejše implementacije z uporabo verige blokov.

4.2.2 Usmerjevalna storitev

Glavna naloga usmerjevalne storitve je, da vzdržuje glavno usmerjevalno tabelo. Od soležnikov prejema informacije o novih konektorjih, ki so dostopni preko njih, in o konektorjih, ki niso več aktivni in so zato nedosegljivi. Vse te podatke obdela in posodablja svojo usmerjevalno tabelo. Način posodabljanja te tabele je odvisen od posamezne implementacije storitve. Naša implementacija uporablja "Distance-vector routing protocol", ki poti med seboj primerja na podlagi števila skokov, ki so potrebni, da dosežemo prejemnika. Pri tem manjše število skokov pomeni primernejšo pot do prejemnika. Definicija vmesnika usmerjevalne storitve je zapisana v izseku kode 4.2.

Izsek kode 4.2: Definicija vmesnika za implementacijo usmerjevalne storitve

```
0       type Routing interface {
1           Run(ctx context.Context) error
2           Route(r types.Address, b, c types.Currency) ([]types.Route, error)
3           RouteAny(r types.Address, c types.Currency) ([]types.Route, error)
4           Update(rd ... types.RouteUpdate)
5           Updates() <-chan []types.RouteUpdate
6           Table() []types.Route
7       }
```

4.2.2.1 Opis metod

Run

Metoda zažene več različnih funkcij, ki periodično skrbijo za preverjanje ustreznosti zapisov v usmerjevalni tabeli. Ena takšnih funkcij je *removeOldRoutes(ctx context.Context)*, ki vsakih 5 sekund pregleda poti in odstrani tiste, ki jih soležniki niso osvežili več kot 10 sekund. Funkcija Run obenem skrbi, da se vse funkcije, ki se periodično izvajajo, ob preklicu izvajalnega konteksta, podanega kot argument funkciji, tudi zaključijo.

Route

Metoda vrne vse možne poti, ki vodijo do določenega prejemnika za izbrano izvorno in ciljno valuto. Zapisi, ki jih funkcija vrne, morajo biti urejeni od najboljše do najslabše poti. V trenutni implementaciji to pomeni, da so zapisi urejeni od poti, ki ima najmanjše število skokov, do poti, ki jih ima največ in je zato najmanj primerna.

RouteAny

Metoda deluje podobno kot *Route* in vrne vse poti do prejemnika glede na ciljno valuto, vendar se ne ozira na izvorno valuto.

Update

Metoda kot argument sprejme poljubno število posodobitev poti in jih poskusi dodati v usmerjevalno tabelo. Te posodobitve so lahko tipa “dodaj” ali “odstrani”. V primeru tipa “dodaj” funkcija ovrednoti trenutno stanje povezovalne tabele in v primeru, da bi dodajanje te poti pozitivno vplivalo na zmožnost usmerjanja plačil, to pot tudi doda oziroma zamenja enakovredno že obstoječo pot. Posodobitev tipa “odstrani” iz usmerjevalne tabele določeno pot brezpogojno izbriše.

Updates

Metoda vrne kanal, v katerega se zapisujejo vse spremembe usmerjevalne tabele, ki jih prejemnik nato lahko posreduje naprej vsem svojim soležnikom.

Table
Metoda vrne kopijo celotne trenutne usmerjevalne tabele in se uporablja predvsem za vizualizacijo.

4.2.3 Storitev skladišča

Storitev skladišča skrbi za shranjevanje in pridobivanje podatkov. Definicija vmesnika storitve skladišča je zapisana v izseku kode 4.3.
Naša implementacija podpira več različnih skladiščnih zaledij, ki komunicirajo s podatkovnimi bazami:

- **v pomnilniku** omogoča shranjevanje podatkov zgolj v pomnilniškem prostoru programa. To skladiščno zaledje ne omogoča trajnega hranjenja podatkov in se uporablja zgolj za testiranje.

- Skladiščno zaledje **MongoDB** je primer implementacije uporabe baze NoSQL za shranjevanje podatkov.

- Skladiščno zaledje **PostgreSQL** je primer implementacije uporabe relacijske podatkovne baze. Definirane ima tudi migracije, potrebne za vzpostavitev podatkovne baze in vseh potrebnih tabel.

Izsek kode 4.3: Definicija vmesnika za implementacijo storitve skladišča

```
0  type Storage interface {
1      Run(ctx context.Context) error
2      StoreTransaction(transaction types.Transaction) error
3      LoadTransaction(txID string) (types.Transaction, error)
4      HasTransaction(txID string) bool
5      StorePrepare(txPrepare types.TransferPrepare) error
6      LoadPrepare(txID string) (types.TransferPrepare, error)
7      StoreFulfillment(txFulfill types.TransferFulfill) error
8      LoadFulfillment(txID string) (types.TransferFulfill, error)
9  }
```

4.2.3.1 Opis metod

Run

Metoda poskusi vzpostaviti povezavo s podatkovno bazo in skrbi, da se v primeru, da se povezava izgubi, ta tudi ponovno vzpostavi.

StoreTransaction

Metoda shrani podatke o transakciji. V primeru, da transakcija z določenim identifikatorjem že obstaja, vrne napako.

LoadTransaction

Metoda poskusi naložiti transakcijo z določenim identifikatorjem. Če taka transakcija ne obstaja, vrne napako.

HasTransaction

Metoda vrne logično vrednost, ki pove, ali je transakcija z določenim identifikatorjem prisotna v podatkovni bazi.

StorePrepare

Metoda shrani podatke o koraku transakcije “prepare”. V primeru, da korak “prepare” za določeno transakcijo že obstaja, vrne napako.

LoadPrepare

Metoda poskusi naložiti podatke o koraku “prepare” za določeno transakcijo. Če ta korak še ni bil izveden oziroma podatki o tem koraku še niso bili shranjeni, vrne napako.

StoreFulfill

Metoda shrani podatke o koraku transakcije “fulfill”. V primeru, da korak “fulfill” za določeno transakcijo že obstaja, vrne napako.

LoadFulfill

Metoda poskusi naložiti podatke o koraku "fulfill" za določeno transakcijo. Če ta korak še ni bil izveden oziroma podatki o tem koraku še niso bili shranjeni, vrne napako.

4.2.4 Storitev plačilnih kanalov

Storitev plačilnih kanalov ima popoln nadzor nad konektorjevimi plačilnimi kanali. Skrbi za pravilno odpiranje in zapiranje kanalov, pošiljanje sporočil drugih storitev, inicializiranje transakcij s konektorji, izvajanja korakov priprave in izvrševanje transakcij ter poravnav plačilnih kanalov. Definicija vmesnika storitve plačilnih kanalov je zapisana v izseku kode 4.4.

Izsek kode 4.4: Definicija vmesnika za implementacijo storitve plačilnih kanalov

```
0  type Channel interface {
1      Run(ctx context.Context) error
2      Pay(rc types.Address, sc, tc types.Currency, am uint64, dt []byte) error
3      OpenChannel(target types.Address, currency types.Currency) error
4      CloseChannel(target types.Address) error
5      Initiate (tc types.Currency, amount uint64) (*types.Transaction, error)
6      List () []types.PaymentChannel
7  }
```

4.2.4.1 Opis metod

Run

Metoda zažene strežnik BTP in začne sprejemati in obdelovati zahteve za odpiranje plačilnih kanalov.

Pay

Metoda sproži plačilo v določeni valuti in vsoti ciljnemu konektorju. Preprečiti, da bi konektor izvedel plačilo samemu sebi.

OpenChannel

Metoda poskusi odpreti nov plačilni kanal s ciljnim konektorjem. Ta funkcija

ukaz za odprtje novega plačilnega kanala zgolj doda v vrsto in ne zagotavlja, da se bo ta dejansko odprl.

CloseChannel
Metoda poskusi zapreti obstoječi plačilni kanal. Zagotavlja, da se bo kanal sčasoma zaprl, ne zagotavlja pa točnega časa, ko se bo to zgodilo. Razlog za to je, da se morajo vse transakcije, ki uporabljajo ta plačilni kanal, zaključiti, soležnika pa poravnati.

Initiate
Metoda ustvari novo transakcijo in vrne pogoje, pod katerimi je konektor pripravljen, da izvede določeno transakcijo.

List
Metoda vrne seznam s podrobnostmi o vseh trenutno vzpostavljenih plačilnih kanalih.

4.2.4.2 Plačilni kanal

Plačilni kanali omogočajo dvosmerno izmenjavo sredstev med dvema soležnikoma, ne da bi ta sredstva zamenjala glavno knjigo. To se zgodi zgolj takrat, ko enemu izmed soležnikov zmanjka sredstev v kanalu, kar povzroči poravnavo med njima na dejanskem omrežju (npr. SEPA). Po uspešni poravnavi se plačilni kanal sprosti in soležnika si lahko ponovno pošiljata sredstva.

Omejitve
Posamezen plačilni kanal omogoča pošiljanje sredstev zgolj v eni valuti na enem omrežju. V primeru, da si soležnika želita pošiljati še neko drugo valuto, morata za to odpreti namenski plačilni kanal.

Življenjski cikel
Plačilni kanal je lahko med obstojem v več različnih stanjih, kar imenujemo življenjski cikel. Prehod med stanji je razviden iz slike 4.1.

- **Created** je začetno stanje kanala, ki se nastavi ob inicializaciji. V tem stanju povezava BTP s soležnikom še ni vzpostavljena.

- **Connected** je stanje, ko se vzpostavi povezava BTP s soležnikom in je kanal pripravljen na rokovanje BTP. V primeru napake pri vzpostavljanju povezave se kanal zapre.

- **Authenticating** je stanje, ko si odjemalec in strežnik izmenjujeta avtentikacijske podatke. Odjemalec strežniku pošlje svoj avtentikacijski žeton in naslov ILP. Strežnik podatke preveri, in če ustrezajo odjemalcu pošlje svoj naslov ILP. V primeru, da je pri avtentikaciji prišlo do napake, strežnik to odjemalcu sporoči in plačilni kanal zapre.

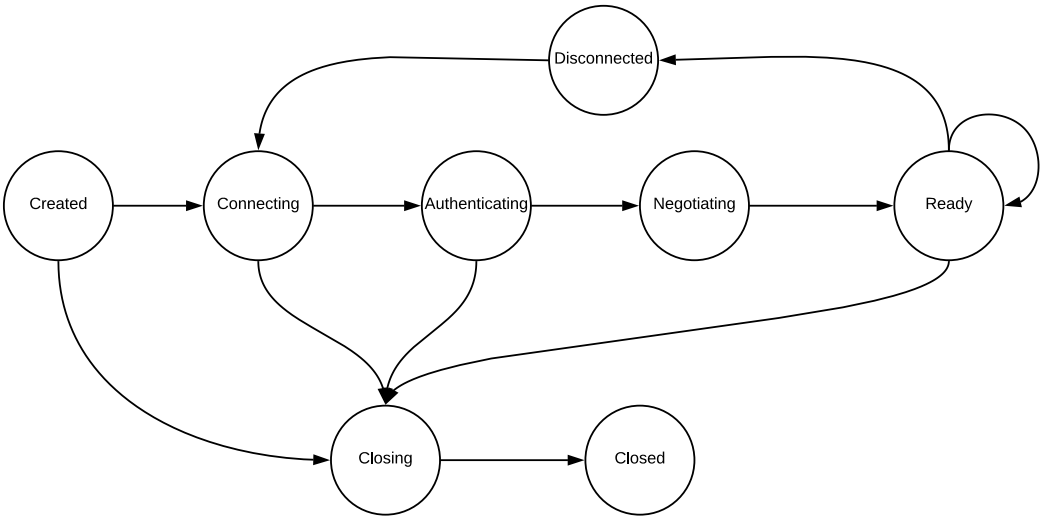
- **Negotiating** je stanje plačilnega kanala, ko odjemalec strežniku pošlje podatke o valuti, za katero želi odpreti plačilni kanal. Strežnik odjemalcu odgovori ali z zavrnitvijo, v tem primeru tudi zapre plačilni kanal, ali z odobritvijo, ki vključuje kapaciteto plačilnega kanala.

- **Ready** je stanje, ko je plačilni kanal pripravljen za prenašanje podatkov in sredstev. V tem stanju plačilni kanal preživi največ časa.

- **Disconnected** je stanje, ko se iz neznanega razloga povezava BTP s soležnikom nenadoma zapre. V tem primeru se plačilni kanal začasno onemogoči in povezava poskusi ponovno vzpostaviti. V primeru, da se povezava ne more ponovno vzpostaviti in se vse aktivne transakcije niso zaključile, lahko pride do izgube sredstev konektorja.

- **Closing** je stanje, ko preko kanala ni več možno pošiljati sredstev in kanal čaka zgolj na transakcije, ki so trenutno aktivne. Po končani izvedbi vseh transakcij se kanal lahko dokončno zapre.

- **Closed** je stanje, ko je plačilni kanal zaprt. To pomeni, da sta si soležnika poplačala vse dolgove na dejanskih plačilnih sistemih.



Slika 4.1: Shema prehajanja stanj plačilnega kanala

4.2.5 Storitev za upravljanje omrežij

Storitev za upravljanje omrežij skrbi za upravljanje z vtičniki, ki komunciririajo z nativnimi omrežji. Predstavlja most med temi omrežji in dejanskim omrežjem ILP, obenem pa skrbi za proženje transakcij na nativnih omrežjih. Definicija vmesnika storitve za upravljanje z omrežji je zapisana v izseku kode 4.5.

Izsek kode 4.5: Definicija vmesnika za implementacijo storitve upravljanja omrežij

```
0        type Network interface {
1            Run(ctx context.Context) error
2            Transfer(it []string, mt []byte, am uint64, cx types.Currency) error
3            RegisterPlugin(ctx context.Context, ad string) error
4            PluginsList() []types.Plugin
5            Negotiate(ad types.Address, tc string, nt string) (uint64, *types.Currency
                 , error)
6        } fig : payment_channel_state_diagram
```

4.2.5.1 Opis metod

Run

Metoda za vsak v konfiguraciji definiran vtičnik za nativna omrežja pokliče funkcijo RegisterPlugin.

Transfer

Metoda sproži transakcijo določenemu prejemniku na določenem omrežju. Ta funkcija izvede dejansko transakcijo, ki je v primerih kriptoomrežij nepovratna, kar pomeni, da lahko konektor izgubi dejanska sredstva.

RegisterPlugin

Metoda se poskusi povezati z določenim vtičnikom. Ob uspešni povezavi njim aktivira tudi periodično preverjanje njegovega stanja.

Negotiate

Metoda vrne kapaciteto kanala, ki ga je konektor pripravljen odpreti z odjemalcem za neko določeno valuto, in naslov ILP, ki ga vtičnik uporablja za to valuto.

4.2.6 Storitev za upravljanje računov

Storitev za upravljanje računov nudi vmesnik za upravljanje z računi, ki jih imajo uporabniki odprte pri konektorju. Ti uporabniki so po navadi aplikacije, ki želijo na preprost način prejeti plačilo za na primer prikaz vsebine ali plačilo za fizični produkt. Primer uporabe računa za prikaz vsebine je predstavljen v podpoglavju 4.2.6.2. Definicija vmesnika storitve za upravljanje računov je zapisana v izseku kode 4.6.

Izsek kode 4.6: Definicija vmesnika za implementacijo storitve za upravljanje računov

```
0        type Service interface {
1          Run(ctx context.Context) error
2          Create(ctx context.Context, adr []types.Address) (string, []types.Address,
               error)
3          Remove(ctx context.Context, accID string) error
4          Subscribe(ctx context.Context, accID string) (<-chan types.AccountPayment,
               <-chan error, error)
5          List(ctx context.Context) []*types.Account
6        }
```

4.2.6.1 Opis metod

Run

Metoda požene storitev za upravljanje računov s podano konfiguracijo in poskusi naložiti obstoječe račune iz podatkovnega zaledja.

Create

Metoda ustvari nov račun in ga poveže na naslove ILP, podane kot argument. Preveri, da vsi podani naslovi ILP pripadajo vtičnikom, s katerimi je konektor povezan. Ob uspešni ustvaritvi računa metoda vrne njegov unikatni identifikator in seznam naslovov ILP za ta račun. Vsa plačila na te naslove bodo pripisana temu računu.

Remove

Metoda odstrani račun in preneha s posredovanjem plačil na ta račun.

Subscribe

Metoda ustvari dolgotrajni kanal, preko katerega naročnik dobiva informacije o plačilnih na določen račun. Ob vzpostavitvi kanala se vanj najprej vpišejo vsa plačila, ki so bila poslana na račun, ampak še niso bila posredovana nobenemu

naročniku. Nato se v kanal v realnem času vpisujejo vsa plačila za ta račun. V danem trenutku je lahko za določen račun odprt zgolj en kanal.

List

Metoda vrne seznam vseh ustvarjenih, neizbrisanih računov. V primeru, da računov ni, vrne prazen seznam.

4.2.6.2 Primer uporabe računa

Preprost in zelo nazoren primer uporabe računa pri konektorju je plačevanje za prikaz vsebine spletne strani. Za to smo implementirali spletni strežnik, ki od obiskovalca zahteva, da za prikaz plača. V nasprotnem primeru mu vsebine ne prikaže.

Pred zagonom strežnika moramo najprej pognati svoj konektor in ga povezati z ostalimi konektorji. Opis tega se nahaja v poglavju 4.4.

Nato z njim vzpostavimo dolgotrajno povezavo, ki jo bomo uporabljali med celotnim izvajanjem aplikacije. Izsek kode 4.7 prikazuje odpiranje povezave s konektorjem Orcus, ki posluša na *localhost:8080*.

Izsek kode 4.7: Primer vzpostavitve povezave gRPC s konektorjem Orcus

```
0  conn, err := grpc.Dial("localhost:8080", grpc.WithInsecure())
1  defer conn.Close()
2  if err != nil { return err }
```

Po uspešni povezavi na konektor moramo ustvariti nov račun. To storimo tako, da najprej ustvarimo odjemalca, ki se samodejno zgenerira z ukazom *make compile-protobuf*, opisanim v 4.4.2. Nato izvedemo klic za kreiranje računa, ki ga vežemo na naslov ILP *g.ethereum.eth*. Izsek kode tega koraka je prikazan v izseku kode 4.8.

Izsek kode 4.8: Primer kreiranja novega računa

```
0        ac := orcus.NewAccountServiceClient(conn)
1        rspAc, err := ac.AccountCreate(ctx, &orcus.AccountCreateRequest{
2          IlpAddresses: []string{"g.ethereum.eth"},
3        });
4        if err != nil { return err }
```

Kot rezultat tega klica dobimo identifikator računa in naslov ILP, na katerega bodo morali obiskovalci nakazati sredstva v zameno za vsebino. Nato se naročimo na vsa plačila, namenjena na naš račun. Ob prejemu sporočila njegovo vsebino pod poljem *Data* poskusimo dekriptirati. Rezultat uspešne dekripcije je identifikator zahtevka, za katerega je uporabnik plačal. Potrditev plačila si shranimo v asociativno tabelo, v kateri imamo shranjene vse zahtevke v teku. V primeru neuspešnega poizkusa dekripcije sporočilo zavržemo, saj je za nas neveljavno. Izsek kode, ki prikazuje navedeni postopek, je prikazan v izseku kode 4.9.

Izsek kode 4.9: Primer kreiranja novega računa

```
0        asCl, err := ac.AccountSubscribe(ctx, &orcus.AccountSubscribeRequest{
1          AccountId: rspAc.AccountId,
2        })
3        if err != nil { return err }
4        var mx sync.Mutex
5        m := make(map[string]*request)
6        go func() {
7          for {
8            v, err := asCl.Recv()
9            if err != nil { fmt.Println(err.Error()); return }
10
11           d, err := AesGcmDecrypt(encryptKey, v.Data)
12           if err != nil { fmt.Println(err.error()); continue }
13
14           if id := findByPtr(m, string(d)); id != "" { setRequestPaid(id) }
15         }
16       }()
```

Na koncu kreiramo nov spletni strežnik, ki ob zahtevi na korensko pot zahteva

plačilo oziroma preveri, ali je uporabnik že plačal. Postopek prikaza vsebine poteka v dveh korakih. V prvem koraku strežnik uporabniku na zahtevo odgovori s statusno kodo *402 – PaymentRequired* [15] in petimi glavami HTTP:

- **X-Pay-Pointer** predstavlja enolični identifikator zahteve. Uporabnik to glavo in vrednost pošlje strežniku, ko je že plačal in želi videti vsebino,
- **X-Pay-To** predstavlja naslov ILP, na katerega mora uporabnik nakazati določeno vsoto. V našem primeru je to naslov ILP, ki smo ga dobili, ko smo kreirali račun pri konektorju,
- **X-Pay-Amount** predstavlja vsoto, ki jo mora plačati uporabnik,
- **X-Pay-Currency** predstavlja valuto, v kateri mora uporabnik nakazati sredstva in
- **X-Pay-Data**, ki predstavlja podatke, ki jih mora uporabnik vključiti v transakcijo. Ti podatki se uporabijo za potrditev plačila, kot je razvidno iz izseka kode 4.9.

V drugem koraku uporabnik zahtevo ponovi, vendar tokrat vključi glavo *X-Pay-Pointer* z vrednostjo, ki jo je prejel od strežnika. Ta nato preveri, ali je že prejel plačilo za ta zahtevek. V primeru, da je že dobil potrdilo o plačilu, uporabniku vrne vsebino, sicer pa statusno kodo *202 – Accepted* [15], ki pomeni, da je uporabnikov identifikator veljaven, ampak plačilo še ni prispelo. Izsek kode, ki prikazuje opisani postopek, je prikazan v izseku kode 4.10.

Izsek kode 4.10: Primer strežnika ki za prikaz vsebine zahteva plačilo

```
0   r := httprouter.New()
1   r.GET("/", func(w http.ResponseWriter, r *http.Request, p httprouter.Params) {
2     if v := r.Header.Get("X-Pay-Pointer"); v != "" {
3       mx.Lock(); defer mx.Unlock();
4       vx, ok := m[v]
5       if !ok { w.WriteHeader(http.StatusBadRequest); return }
6       if vx.(*requested).isPaid { showContent(&w); return }
7       w.WriteHeader(http.StatusAccepted)
8     } else {
9       id := uuid.NewV4().String()
10      pt := hex.EncodeToString(AesGcmEncrypt(key, uuid.NewV4().Bytes()))
11      mx.Lock();
12      m[id] = &requested{ptr: pt, isPaid: false }
13      mx.Unlock()
14
15      w.Header().Set("X-Pay-Pointer", id)
16      w.Header().Set("X-Pay-To", "g.ethereum.eth")
17      w.Header().Set("X-Pay-Amount", "100")
18      w.Header().Set("X-Pay-Currency", "ETH+ethereum")
19      w.Header().Set("X-Pay-Data", pt)
20      w.WriteHeader(http.StatusPaymentRequired)
21    }
22  })
23  log.Fatal(http.ListenAndServe(":6060", r))
```

4.3 Vtičniki

Vtičniki so namenjeni komunikaciji konektorja z nativnimi omrežji, kot so Ripple, SEPA in druga. Konektorju omogočajo, da izvede transakcijo s svojega računa na določenem omrežju na nek drug račun na istem omrežju. To omogoča, da se sredstva prestavijo v najkrajšem možnem času, ki ga dopušča določeno omrežje.

V naši implementaciji vtičniki za določena omrežja niso del konektorja, ampak so na voljo posamično, kot lastni moduli. Upravljalec konektorja se pred zagonom sam odloči, katere od uradno podprtih vtičnikov bo uporabil, lahko pa uporabi

tudi svojega. Komunikacija med konektorjem in vtičnikom poteka preko protokola gRPC in uporablja Protobuf kot format sporočil. Zaradi uporabe tega protokola je lahko vtičnik napisan v katerem koli jeziku, ki podpira gRPC in ni omejen zgolj na jezik Golang.

Opis metod

Da je vtičnik združljiv s konektorjem, mora ta implementirati storitev gRPC, ki jo definira konektor. V tem podpoglavju bomo opisali posamezne metode, ki jih mora vtičnik implementirati, in jih podprli s primeri izsekov kode zahtev in odgovorov dotične metode. Struktura vmesnika s temi metodami je definirana v izseku kode 4.11.

Izsek kode 4.11: Protobuf definicija vtičnika

```
service Plugin {
    rpc Info (Empty) returns (InfoResponse);
    rpc Health (Empty) returns (HealthResponse);
    rpc Transfer (TransferRequest) returns (TransferResponse);
    rpc Rate (RateRequest) returns (RateResponse);
    rpc Currency (Empty) returns (CurrencyResponse);
    rpc Negotiate (NegotiateRequest) returns (NegotiateResponse)
        ;
}
```

Info

Metoda vrne splošne podatke o vtičniku, kot so ime, verzija vtičnika, verzija prevajalnika in čas prevajanja. Konektor ta klic izvrši, takoj ko se želi povezati z vtičnikom, in se na podlagi odgovora odloči, ali bo s povezovanjem nadaljeval. Na primer, konektor bi lahko prekinil nadaljnje povezovanje, če je verzija vtičnika prestara. Izsek kode protobuf definicije odgovora je zapisan v izseku kode 4.12.

Izsek kode 4.12: Protobuf definicija odgovora klica info metode

```
message InfoResponse {
    string name = 1;
    string plugin_version = 2;
    string compiled_at = 3;
    string compiler_version = 4;
}
```

Health

Metoda vrne podatke o splošnem zdravju vtičnika. Konektor jo kliče periodično in se na podlagi odgovora odloči, ali bo ta vtičnik še uporabljal za nadaljnje transakcije. Na primer, če vtičnik ne more več delovati na omrežju Bitcoin, ker je izgubil povezavo z vozliščem, vrne status *NODE_CONNECTION_LOST*. Izsek kode protobuf definicije odgovora je zapisan v izseku kode 4.13.

Izsek kode 4.13: Protobuf definicija odgovora klica health metode

```
message HealthResponse {
    enum Status {
      UNKNOWN = 0;
      OK = 1;
      NODE_CONNECTION_LOST = 2;
    }
    Status status = 1;
}
```

Transfer

Metoda izvede transakcijo na omrežju, na katerem deluje vtičnik. Na primer, če vtičnik deluje na omrežju Ethereum, ta funkcija ustvari transakcijo za določeno vsoto in valuto ter jo pošlje v omrežje Ethereum. Pri implementaciji te funkcije moramo biti zelo previdni, saj lahko v primeru napačne implementacije konek-

tor ostane brez sredstev. To še posebej velja, če deluje na omrežju, ki temelji na tehnologiji veriženja blokov, kjer so transakcije, za razliko od bančnih, nepovratne. Izseka kode zahteve in odgovora za izvršitev transakcije sta zapisana v izseku kode 4.14. Klicatelj v zahtevi posreduje podatke o naslovu, na katerega naj se pošljejo sredstva, koliko naj se pošlje in v kateri valuti. Pošiljatelj lahko poda tudi metapodatke, ki se bodo vpisali neposredno v transakcijo. Na primer, pri transakciji XRP se ti podatki zapišejo v polje *memo*. Kot odgovor se v primeru uspešne izvedbe transakcije vrne unikatni identifikator transakcije na omrežju; v primeru neuspešne izvedbe se vrne napaka.

Izsek kode 4.14: Protobuf definicija zahteve in odgovora klica transfer methode

```
message TransferRequest {
    string address = 1;
    bytes meta = 2;
    uint64 amount = 3;
    string ticker = 4;
}
message TransferResponse {
    string id = 1;
}
```

Rate

Metoda vrne aktualni menjalni tečaj za določen par valut, ki ju vtičnik podpira. Izseka kode zahteve in odgovora pridobitve menjalnega tečaja sta zapisana v izseku kode 4.15. Klicatelj v zahtevi posreduje izvorno in ciljno valuto. Kot odgovor se vrnejo podatki o menjalnem tečaju, njegovi zadnji posodobitvi in njegovem ponudniku. V primeru, da vtičnik ne podpira menjave med želenima valutama, vrne napako.

Izsek kode 4.15: Protobuf definicija zahteve in odgovora klica rate metode

```
message RateRequest {
    string source_currency = 1;
    string target_currency = 2;
}

message RateResponse {
    string rate = 1;
    string provider = 2;
    string last_update = 3;
}
```

Currency

Metoda vrne vse valute, ki jih vtičnik podpira. Izsek kode odgovora klica je definiran v izseku kode 4.16. Kot odgovor se vrne seznam s podrobnimi podatki o valutah, ki so podprte. Te informacije vključujejo naslov ILP, ki ga lahko konektor oglašuje v omrežju, tričrkovno oznako valute v standardu ISO 4217[1], omrežje, človeško berljivi naziv in skalo valute, ki predstavlja število decimalnih mest.

Izsek kode 4.16: Protobuf definicija odgovora klica currency metode

```
message CurrencyResponse {
    message Currency {
        string ilp_address = 1;
        string ticker = 2;
        string network = 3;
        string name = 4;
        uint32 scale = 5;
    }

    repeated Currency currencies = 1;
}
```

Negotiate

Metoda vrne velikost plačilnega kanala, za katerega vtičnik dovoli, da ga konektor odpre s soležnikom za neko določeno valuto, ki jo ta vtičnik podpira. Izseka kode zahteve in odgovora klica te metode sta zapisana v izseku kode 4.17. Klicatelj v zahtevi posreduje podatke o naslovu ILP soležnika, ki želi odpreti plačilni kanal, in valuti tega kanala. Kot odgovor se vrne velikost kanala, predstavljena v najmanjši enoti valute. V primeru EUR in velikosti kanala 10 to pomeni 10.000 v centih.

Izsek kode 4.17: Protobuf definicija zahteve in odgovora klica negotiate metode

```
message NegotiateRequest {
    string address = 1;
    string ticker = 2;
}

message NegotiateResponse {
    uint64 amount = 1;
}
```

4.4 Poganjanje konektorja

Za zagon konektorja in povezavo z ostalimi konektorji v omrežju je potrebnih kar nekaj korakov: od vzpostavitve lokalnega razvojnega okolja in prenosa izvorne kode, preko prevajanja do konfiguracije in zagona. Vse te korake podrobno predstavimo v naslednjih podpoglavjih.

Ker najverjetneje večina uporabnikov ne bo sama prevajala konektorja in vtičnikov, v zadnjem podpoglavju predstavimo še alternativno možnost zagona konektorja s pomočjo vnaprej pripravljenih slik Docker.

4.4.1 Priprava okolja

Za vzpostavitev lokalnega razvojnega okolja je na računalniku treba imeti prednameščene naslednje programe:

- program *git* za upravljanje z repozitoriji kode,
- prevajalnik za *Golang* različice 1.9.4 ali novejše,
- platformo za virtualizacijo vsebnikov *Docker* različice 17.12.0-ce ali novejše,
- program *docker-compose* različice 1.19.0 ali novejše,
- program za upravljanje s paketi *dep* različice devel ali novejše,
- program *make* za poganjanje ponavljajočih se nalog verzije 4.1 ali novejše.

Priprava konektorja

Repozitorij z izvorno kodo konektorja, ki se nahaja na GitHubu na `https://github.com/uroshercog/orcus`, kloniramo na računalnik. Nato v kloniranem direktoriju poženemo ukaz *dep ensure*, ki namesti vse potrebne knjižnice. Te so zapisane v datotekah Gopkg.toml oziroma Gopkg.lock. Vse knjižnice se prenesejo v direktorij vendor/ v trenutnem direktoriju.

Priprava vtičnikov

Podobno kot kodo konektorja tudi repozitorij vtičnikov prenesemo z GitHuba (`https://github.com/uroshercog/orcus-plugins`) in ponovno z ukazom *dep* namestimo vse knjižnice.

4.4.2 Prevajanje programa

Za prevajanje programa imamo na voljo več različnih ukazov:

- **make compile-develop** pokliče prevajalnik *go* z argumenti, ki program prevedejo v razvojnem načinu. Izklopijo se različne optimizacije, ki jih izvede prevajalnik, obenem pa se uporabi tudi zastavica *-race*, ki v program namesti detektor za zaznavanje nesinhroniziranega sočasnega branja in pisanja pomnilnika. Program, preveden na ta način, deluje počasneje in je tudi večji.

- **make compile-release** pokliče prevajalnik *go* z argumenti, ki program prevedejo v produkcijsko verzijo za arhitekturo trenutnega sistema.

- **make compile-alpine** pokliče prevajalnik *go* z argumenti, ki program prevedejo za sistem Linux Alpine. Pri tem se izklopi *CGO*, uporabi se implementacija omrežnega sklada iz standardne knjižnice in ne iz sistema, na sistema na katerem prevajamo, ter se prevede kot statičen program, brez odvisnosti od sistemskih knjižnic. To se uporablja za pakiranje programa v zelo majhno sliko Docker, ki je popolnoma samozadostna.

Pri vseh treh načinih prevajanja se v program ob povezovanju zapišejo tri vrednosti:

- **Version** predstavlja trenutno verzijo konektorja Orcus. Kot vrednost se zapiše prvih 8 znakov zgoščene vrednosti zadnje spremembe, ki je bila uveljavljena v sistemu za nadzor izvorne kode GIT,

- **CompiledAt** predstavlja čas, ko je bil program preveden, in

- **GoVersion**, ki predstavlja verzijo prevajalnika, s katerim je bil program preveden.

Ob zagonu katerega koli od teh treh načinov prevajanja konektorja se najprej samodejno požene ukaz ‘make compile-protobuf’. Ta zažene prevajalnik za Protobuf z vtičnikoma gRPC in Golang in datoteke s protobuf definicijami, ki se nahajajo v direktorijih pkg/plugin in rpc/proto, prevede v kodo Golang. Ta koda vsebuje strežnik in odjemalca gRPC. Po končanem prevajanju konektorja se samodejno požene ukaz ‘make compile-orcusctl’, ki ustvari program *orcusctl*, na-

menjen upravljanju s konektorjem. Po uspešnem prevajanju se v trenutnem direktoriju ustvari mapa, imenovana *bin*, in v njej programa *orcusd* in *orcusctl*.

4.4.3 Prevajanje vtičnikov

Podobno kot konektor moramo prevesti tudi vtičnike. Poleg treh identičnih ukazov, ki jih uporabljamo tudi za prevajanje konektorja, imamo na voljo še dodatnega, *compile-all*, ki prevede vse vtičnike naenkrat. Ob klicu ukaza moramo podati še argument *TYPE*, ki definira, kateri način prevajanja se uporabi. Na voljo so *develop*, *release* in *alpine*. Ukazom *compile-develop*, *compile-release* in *compile-alpine* moramo ob klicu podati argument *PLUGIN*, ki pove, katerega od vtičnikov želimo prevesti. Izsek kode ukaza *compile-all* je prikazan v 4.18.

Izsek kode 4.18: Primer kode ukaza *compile-all* za prevajanje vtičnikov

```
0   @files:=$(find ./ -maxdepth 1 -mindepth 1 -type d | grep -i -P '^\./(?!vendor
        |bin|\..* $$).*')
1   @while read r; do
2       $(MAKE) PLUGIN=$$r compile-$TYPE
3   done <<< "$files"
```

4.4.4 Konfiguracija

V direktoriju *orcus* projekta datoteko examples/orcus-connector.yaml prekopiramo v direktorij bin/. V prekopirani datoteki se v obliki YAML nahajajo konfiguracijske opcije, ki so razložene v dodatku B.1. Konfiguracija vtičnikov je odvisna od posameznega vtičnika. V dodatku B.2 je razložena konfiguracija za vtičnik GateHub.

4.4.5 Zagon programa

Program zaženemo tako, da iz terminala poženemo *bin/orcusd*. Konektor se zažene, naloži konfiguracijsko datoteko, poskusi zagnati vse storitve, se povezati z vsemi

definiranimi vtičniki in z vsemi vnaprej definiranimi soležniki. V primeru napake ob zagonu se ta izpiše in konektor zaključi z izvajanjem.

4.4.6 Alternativna možnost zagona

Kot alternativna ročnemu prevajanju celotnega sistema sta na voljo vnaprej pripravljeni javni sliki Docker na repozitoriju Docker `uroshercog/orcus` in `uroshercog/orcus-plugins`. Ti sliki se samodejno zgenerirata ob vsaki novi izdaji verzije konektorja in vtičnikov. Primer datoteke *docker-compose* v izseku kode 4.19 prikaže zagon vnaprej pripravljene slike *orcus*. Naloži se tudi slika z vsemi uradno podprtimi vtičniki. Treba je zgolj zagotoviti konfiguracijske datoteke.

Izsek kode 4.19: Primer datoteke docker-compose

```
version: "2"
services:
  redis:
    image: "redis"
    networks:
      - "orcus"
  orcus:
    image: "uroshercog/orcus"
    volumes:
      - "${PWD}/bin/orcus-config.yaml:/opt/orcus-
        config.yaml"
    networks:
      - "orcus"
    depends_on:
      - redis
      - plugin-ethereum
  plugin-ethereum:
    image: "uroshercog/orcus-plugins"
```

```
    volumes:
      - "${PWD}/bin/ethereum-config.yaml:/opt/
        plugins/ethereum-config.yaml"
    command: ["./plugins/ethereum"]
    networks:
      - "orcus"
networks:
  orcus:
    driver: "bridge"
```

4.5 Upravljanje s konektorjem

Ob prevajanju konektorja se poleg programa *orcusd* ustvari še program *orcusctl*, namenjen upravljanju s konektorjem. Predstavlja referenčno implementacijo odjemalca, s katerim lahko na aktivnem konektorju izvajamo administrativne naloge. Program podpira klice večine metod, ki jih implementirajo naslednje storitve:

- **account** omogoča ustvarjanje, brisanje in pridobivanje trenutno odprtih računov pri konektorju. Obenem omogoča tudi pošiljanje sredstev na poljuben naslov ILP in naročnino na vsa plačila za določen račun,
- **channel** omogoča odpiranje, zapiranje in pridobivanje vseh trenutno odprtih plačilnih kanalov,
- **plugin** omogoča tako pridobivanje trenutno povezanih vtičnikov kot tudi povezovanje na nove in zapiranje povezav z že povezanimi vtičniki,
- **registry** ima zgolj ukaz za razreševanje naslovov ILP v oddaljenem registru in
- **route**, ki omogoča pridobivanje seznama vseh trenutno aktivnih poti, ki so shranjene v usmerjevalni tabeli usmerjevalne storitve. Obenem omogoča tudi iskanje poti glede na prejemnikov naslov in izvorno ter ciljno valuto.

Primer izvedbe ukaza za pridobitev vseh odprtih računov konektorja je prikazan v izseku kode 4.20.

Izsek kode 4.20: Primer ukaza za pridobitev odprtih računov

```
orcusctl --rpc-url=localhost:9000 account list
```

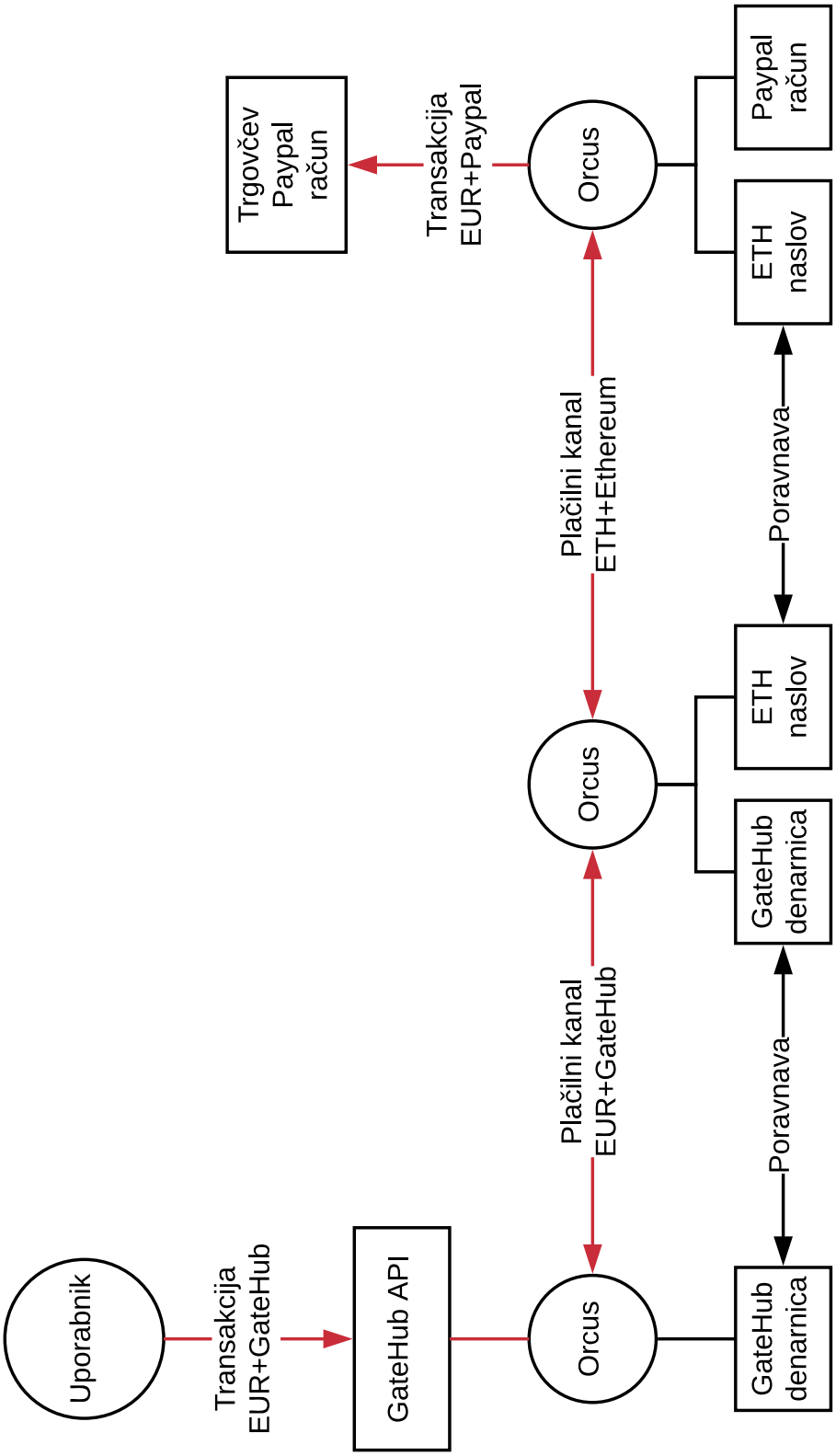
4.6 Integracija v obstoječi plačilni sistem

Kot primer integracije konektorja Orcus za prenos sredstev preko protokola Interledger bomo predstavili plačilo storitve trgovcu. Uporabnik za dogovorjeno storitev plača v evrih iz svoje denarnice GateHub, trgovec pa sredstva dobi na svoj račun, odpr pri Paypalu. Sredstva se na poti najprej nevidno pretvorijo v Ether na omrežju Ethereum, nato pa v končne evre na omrežju Paypal. Celotna pretvorba se izvede v manj kot sekundi z zelo nizkimi transakcijskimi stroški. Podomrežje, ki omogoča to transakcijo, je prikazano na sliki 4.2. Pri implementaciji testnega plačila smo se omejili zgolj na prenos sredstev v eno smer, od GateHuba preko Ethereuma do Paypala. To pomeni, da za izvedbo transakcije ne potrebujemo vtičnika za GateHub, saj v tem primeru sredstva na omrežju GateHub nikoli ne sodelujejo v poravnavah med konektorji.

4.6.1 Integracija v omrežje GateHub

Integracija konektorja Orcus v omrežje GateHub je bila s stališča delovanja transakcijega sistema GateHub zelo preprosta, saj je bilo treba dodati zgolj popdporo za novo omrežje, podobno kot so to trenutno Bitcoin, Ethereum in ostala podprta omrežja. Za integracijo novega omrežja smo se omejili na razvojno okolje, saj konektor še ni pripravljen na pravo produkcijsko okolje, v katerem bi upravljal dejanska sredstva.

Zaradi arhitekturne zasnove platforme GateHub smo morali razviti preprosto dodatno storitev, imenovano most (*ang. bridge*). Glavna naloga te storitve je, da



Slika 4.2: Primer plačila trgovcu

se naroči na sporočila o novih transakcijah za določeno omrežje pri sporočilni vrsti (*ang. messaging queue*) in jih ustrezno sprocesira. V našem primeru storitev izvede klic za novo transakcijo na konektorju Orcus. Most nato čaka na sporočilo konektorja o uspešnosti ali neuspešnosti izvedbe transakcije. Informacijo o uspešnosti transakcije nato sporoči nazaj glavni knjigi, ki jo obdela.
Podrobnejše razlage implementacije in kode mostu zaradi visoke stopnje integracije z internimi sistemi in s transakcijsko logiko GateHuba ne moremo pokazati.

4.6.2 Vtičnik za omrežje Ethereum

Za programiranje vtičnika Ethereum smo uporabili odjemalca iz uradne implementacije verige blokov Ethereum, ki omogoča zelo preprosto kreiranje in podpisovanje transakcij. Pri razvoju smo uporabljali javno dostopna vozlišča testnega omrežja Ethereum, imenovanega Rinkeby. Za to omrežje je na voljo sistem samodejnega dodeljevanja sredstev naslovom na omrežju (*ang. faucet*). To nam je omogočilo, da smo lahko pretesirali celoten potek transakcije. Izsek kode generiranja, podpisovanja in pošiljanja transakcije je prikazan v 4.21.

Izsek kode 4.21: Primer izvedbe transakcije Ethereum

```
0   toAddr := common.HexToAddress(req.Interactions[0])
1   nonce, err := s.cl.NonceAt(ctx, *s.from, nil)
2   if err != nil { return nil, err }
3
4   g, err := s.cl.EstimateGas(ctx, ethereum.CallMsg{From: *s.from, To: &toAddr
        })
5   if err != nil { return nil, err }
6
7   tx := types.NewTransaction(nonce, toAddr, big.NewInt(int64(req.Amount)), g,
        nil, nil)
8   tx, err = types.SignTx(tx, types.NewEIP155Signer(nil), s.pk)
9   if err != nil { return nil, err }
10
11  s.cl.SendTransaction(ctx, tx)
```

4.6.3 Vtičnik za omrežje Paypal

Programiranje vtičnika za omrežje Paypal nam ni predstavljajo večjega izziva, saj je transakcija izvedljiva zgolj z enim klicem HTTP na njihov API. Za implementacijo vtičnika smo uporabljali njihovo testno okolje, imenovano peskovnik (*ang. sandbox*), v katerem smo ustvarili testne račune. Sistem tem računom samodejno dodeli testna sredstva. Primer izseka kode izvedbe zahtevka za transakcijo je prikazan v 4.22. Podrobna razlaga posameznih polj zahtevka in glav HTTP je na voljo v uradni dokumentaciji [12].

Izsek kode 4.22: Primer izvedbe zahtevka za transakcijo na omrežju Paypal

```
0   re := requestEnvelope{ DetailLevel: "ReturnAll", ErrorLanguage: "en_US" }
1   rl := receiverList {{ Email: "receiver@orcus.io" Amount: "10" }}
2
3   pl := &payRequest{
4     ActionType:      "PAY",
5     SenderEmail:     "sender@orcus.io",
6     CurrencyCode:    "USD",
7     ReturnURL:       "http :// orcus.io",
8     CancelURL:       "http :// orcus.io",
9     ReceiverList:    rl,
10    RequestEnvelope: re,
11  }
12
13  u := "https :// svcs.sandbox.paypal.com/AdaptivePayments/Pay"
14  rq, err := http.NewRequest("POST", u, bytes.NewReader(
        mustMarshalJsonToBytes(pl)))
15  if err != nil { return nil, err }
16
17  rq.Header.Set("X-PAYPAL-SECURITY-USERID", "user id")
18  rq.Header.Set("X-PAYPAL-SECURITY-PASSWORD", "password")
19  rq.Header.Set("X-PAYPAL-SECURITY-SIGNATURE", "signature")
20  rq.Header.Set("X-PAYPAL-APPLICATION-ID", "application id")
21  rq.Header.Set("X-PAYPAL-REQUEST-DATA-FORMAT", "JSON")
22  rq.Header.Set("X-PAYPAL-RESPONSE-DATA-FORMAT", "JSON")
23
24  rsp, err := http.DefaultClient.Do(rq)
```

4.7 Vrednotenje implementacije

Naša implementacija konektorja Interledger izpolnjuje vse funkcionalne cilje, ki smo si jih zadali, in je v skladu s specifikacijo protokola Interledger. Uspešno smo izvedli transakcijo preko več testnih glavnih knjig, kar je bilo do sedaj možno zgolj z uporabo več različnih konektorjev, specifičnih za posamezno omrežje. S tem smo pokazali, da je premikanje sredstev med glavnimi knjigami tako preprosto, kot pošiljanje e-pošte. Dokazali smo, da sta zagon novega konektorja in povezovanje z obstoječimi nadvse preprosta. Podali smo kritiko in predloge izboljšav protokola Interledger. V poglavju 4.8 podamo predloge izboljšav naše implementacije, ki jih bomo v prihodnjih verzijah konektorja upoštevali in implementirali. Načrt nadaljnjega razvoja konektorja podamo v poglavju 4.10.
V sklopu naloge cilja izvedbe transakcije na produkcijskih sistemih nismo izvedli. Razlog za to je, da konektorju zaradi premajhne pokritosti kode s testi še ne zaupamo upravljanja s pravimi sredstvi.

4.8 Predlogi izboljšav

Med razvojem konektorja smo naleteli na ogromno težav, ki smo jih uspešno rešili. Vendar je večina naših rešitev zaradi obsežnosti projekta in pomanjkanja časa zelo osnovnih in včasih neoptimalnih. V nadaljevanju podamo predloge izboljšav obstoječih funkcionalnosti kot tudi predloge implementacije novih funkcionalnosti, ki bi pripomogle k bolj dovršenemu in zanesljivemu delovanju konektorja Orcus.

4.8.1 Redundančni konektorji

Od vseh sistemov, ki so del finančnega ekosistema, se pričakuje, da so neprestano dosegljivi in da se skoraj nikoli ne ugasnejo nepričakovano. Trenutna implementacija konektorja Orcus tem zahtevam ne ustreza, saj vedno teče zgolj ena replika. To pomeni, da če se ta instanca ugasne, cel sistem preneha z delovanjem. Da bi dosegli želeno redundantnost, bi morali to v konektor implementirati. Dodali bi

način delovanja, v katerem bi lahko sočasno pognali več konektorjev in jih povezali med seboj tako, da bi eden izmed njih deloval v aktivnem načinu in bi procesiral transakcije. Ostali konektorji bi delovali v pasivnem načinu in bi se neprestano sinhronizirali s trenutno aktivnim konektorjem. V primeru, da bi se aktivni konektor ugasnil, bi lahko eden izmed pasivnih konektorjev takoj prevzel njegovo vlogo. Tak način delovanja bi lahko implementirali z uporabo algoritma Raft [11] za izbiro aktivnega konektorja.

4.8.2 Podpora za blockchain register konektorjev

Trenutna implementacija zunanjega registra je zelo osnovna in centralizirana. Omogoča zgolj shranjevanje in razreševanje naslovov ILP. Kot možno izboljšavo bi implementirali shranjevanje naslovov ILP v verigo blokov. S tem bi konektorjem omogočili, da bi imeli nad svojimi naslovi popoln nadzor, saj bi za spreminjanje vrednosti, na katere kažejo naslovi, potrebovali ustrezen zasebni ključ. Obenem bi preprečili, da bi imelo več konektorjev isti naslov ILP. Kot osnovo za implementacijo bi lahko uporabili že obstoječo rešitev, imenovano DNSChain [14], ki to funkcionalnost omogoča na ravni sistema DNS.

4.8.3 Avtentikacija vtičnika in konektorja

Povezovanje konektorja na vtičnik predstavlja možno točko napada na konektor ali vtičnik. Napadalec lahko ustvari svoj konektor in se poveže na naš vtičnik ter mu pošlje ukaze za izvedbo transakcij. Na ta način lahko izprazni račune, ki jih upravlja vtičnik. Trenutna implementacija tega ne preprečuje. Možna implementacija medsebojne avtentikacije vtičnika in konektorja bi lahko bila z uporabo preverjanja certifikata strežnika in odjemalca. Vtičnik bi dovolil povezavo zgolj konektorju, katerega certifikat pozna.

4.9 Primeri uporabe

V tem podpoglavju bomo predstavili nekaj primerov uporabe konektorja Orcus in protokola Interledger. Ti primeri prikazujejo instantno plačevanje za storitev oziroma dobrino brez posrednikov. Prejemnik plačila lahko prejeta sredstva takoj porabi za svoje izdatke in mu ni treba čakati, da se sredstva prestavijo na njegov račun.

4.9.1 IoT

Živimo v svetu, v katerem se število pametnih naprav iz leta v leto eksponentno povečuje [7]. S tem se pojavlja tudi težnja po transakcijah M2M, s katerimi si bodo naprave lahko med seboj plačevale za podatke, storitve ali dobrine. Primer plačevanja za dobrine med napravami je polnjenje električnega avtomobila s pomočjo tehnologije IPT [9]. Avtomobil parkiramo na parkirno mesto, ki ima podporo za polnjenje s pomočjo indukcije, ta se začne polniti in plačevati za vsako porabljeno vatno uro.

4.9.2 Strani brez oglasov

Danes skoraj ne vidimo spletne strani, ki ne bi na tak ali drugačen način prikazovala oglasov. Ti oglasi so za lastnike spletnih strani mesečni prihodek, brez katerega teh strani ne bi mogli več podpirati. Vendar veliko ljudi zaradi oglasov, ki so velikokrat moteči, uporablja vtičnike v brskalniku, ki oglase pred prikazom strani izbrišejo. Če je takih uporabnikov dovolj, lastnik spletne strani ostane brez prihodkov in stran ugasne. Kot rešitev te težave lahko uporabimo protokol Interledger in statusno kodo *402 – PaymentRequired* iz protokola HTTP [15]. Uporabnik obišče spletno stran in strežnik mu vrne kodo 402 skupaj z vsoto, ki jo mora plačati, če želi stran brez oglasov, in naslovom ILP, na katerega naj nakaže sredstva. Uporabnikov brskalnik se na podlagi uporabnikovih nastavitev odloči, ali bo izvedel plačilo. Če plača, mu strežnik vrne stran brez oglasov, v nasprotnem primeru pa stran z oglasi.

4.9.3 Pretakanje vsebin

Pretakanje vsebine v živo je danes ob pomoči platform, kot sta Twitch in YouTube, postalo zelo popularno in predvsem preprosto za uporabo. Ljudje so se začeli s tem profesionalno ukvarjati in to jim predstavlja glavni vir dohodka. Ta dohodek temelji predvsem na reklamah, ki se prikazujejo uporabnikom, in mesečnih naročninah, ki jih ti uporabniki plačujejo. Z uporabo Interledgerja bi lahko dodali tretji način. Na primer, uporabnik bi lahko plačal za vsak megabajt vsebine, ki se pretoči k njemu. To pomeni, da bi uporabnik lahko odločil, v kakšni kakovosti želi spremljati vsebino. Če bi izbral nižjo kakovost, bi zaradi manjše količine prenesenih podatkov plačal manj, kot če bi prenašal vsebino v večje kakovosti.

4.10 Nadaljnje delo

Z delom na konektorju Orcus bomo nadaljevali tudi v prihodnje, saj želimo dodati vse funkcionalnosti, ki nam jih v sklopu te naloge ni uspelo implementirati. Eden večjih ciljev, ki jih želimo doseči, je vzpostaviti združljivost konektorja s trenutno edino drugo implementacijo, imenovano *ILP-Connector*, ki jo vzdržujejo avtorji protokola Interledger. Želimo si tudi, da bi se v razvoj prostovoljno vključilo več razvijalcev, saj projekt postaja preobsežen in prekompleksen, da bi ga lahko vzdrževala in nadgrajevala zgolj ena oseba.

Poglavje 5

Zaključek

Preprost pretok sredstev med različnimi glavnimi knjigami še danes, čeprav smo v dobi tehnološkega razcveta, predstavlja enega večjih tehnoloških izzivov. S porastom števila različnih projektov s področja veriženja blokov se bo ta težava samo še povečala in naivno bi bilo pričakovati, da bodo vse glavne knjige interoperabilne. Eno izmed možnih rešitev zagotavljanja interoperabilnosti prinaša protokol Interledger, ki je predstavljal rdečo nit naše naloge.

V diplomski nalogi smo obravnavali težavo premikanja sredstev med različnimi glavnimi knjigami. Pregledali in analizirali smo področje obstoječih plačilnih sistemov in elektronskega poslovanja. Predstavili smo njihove najpomembnejše lastnosti in podali njihovo kritiko. V praktičnem delu naloge smo razvili konektor za protokol Interledger, imenovan Orcus, ki omogoča plačevanje preko tega plačilnega protokola. Kot dokaz uporabnosti protokola in enostavnosti integracije v obstoječe plačilne sisteme smo izvedli testno transakcijo preko kombinacije tako tradicionalnih kot distribuiranih glavnih knjig. Podali smo evalvacijo svojega dela in predloge izboljšav konektorja, ki jih bomo upoštevali pri nadaljnjem razvoju.

Protokol Interledger ne predstavlja končne rešitve težave, ki smo jo obravnavali v diplomski nalogi. Zavedamo se, da ima določene pomanjkljivosti, ki smo jih opisali. Vendar pa upamo, da bo Interledger oziroma podoben sistem prešel v splošno uporabo. Želimo si, da bi finančnim institucijam predstavljal odskočno desko za modernizacijo svojega poslovanja za vedno bolj povezano in avtomati-

zirano prihodnost bančništva.

Literatura

[1] Currency codes - iso 4217. Dosegljivo: https://www.iso.org/iso-4217-currency-codes.html. [Dostopano: 28. 7. 2018].

[2] Developer guide. Dosegljivo: https://developers.google.com/protocol-buffers/docs/overview. [Dostopano: 28. 7. 2018].

[3] A next-generation smart contract and decentralized application platform. Dosegljivo: https://github.com/ethereum/wiki/wiki/White-Paper. [Dostopano: 28. 7. 2018].

[4] Simple payment setup protocol (spsp). Dosegljivo: https://github.com/interledger/rfcs/blob/master/0009-simple-payment-setup-protocol/0009-simple-payment-setup-protocol.md. [Dostopano: 28. 7. 2018].

[5] Target2. Dosegljivo: http://www.ecb.europa.eu/paym/t2/html/index.en.html. [Dostopano: 28. 7. 2018].

[6] Sinclair Davidson, Primavera De Filippi, and Jason Potts. Disrupting governance: The new institutional economics of distributed ledger technology. 2016.

[7] Gartner. Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. Dosegljivo: https://www.gartner.com/newsroom/id/3598917. [Dostopano: 15. 7. 2018].

[8] Internet Engineering Task Force (IETF). Webfinger. Dosegljivo: https://tools.ietf.org/html/rfc7033. [Dostopano: 28. 7. 2018].

[9] S. Lukic and Z. Pantic. Cutting the cord: Static and dynamic inductive wireless charging of electric vehicles. *IEEE Electrification Magazine*, 1(1):57–64, Sept 2013.

[10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[11] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[12] Paypal. Adaptive payments api: Getting started. Dosegljivo: https://developer.paypal.com/docs/classic/adaptive-payments/gs_AdaptivePayments/. [Dostopano: 23. 7. 2018].

[13] Paypal. What are micropayments? Dosegljivo: https://www.paypal.com/us/smarthelp/article/what-are-micropayments-faq664. [Dostopano: 15. 7. 2018].

[14] Greg Slepak. Dnschain + okturtles, 2013. [Dostopano: 28. 7. 2018].

[15] The Internet Society. Hypertext transfer protocol – http/1.1. Dosegljivo: https://tools.ietf.org/html/rfc2616. [Dostopano: 15. 7. 2018].

[16] Melanie Swan. *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.", 2015.

[17] Stefan Thomas and Evan Schwartz. A protocol for interledger payments. 2016.

[18] VISA USA. Visa fact sheet. Dosegljivo: https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-

technology/aboutvisafactsheet.pdf, 2017. [Dostopano: 10. 7. 2018].

Dodatek A

Seznami napak

A.1 Zavrnitev transakcije

A.1.1 Končne napake

Koda	Ime	Opis
F00	Bad Request	Splošna napaka pošiljatelja.
F01	Invalid Packet	Paket ILP je sintaktično nepravilen.
F02	Unreachable	Nemogoče posredovati paket prejemniku, ker je ali njegov naslov ILP neveljaven ali pa pot do njega ne obstaja.
F03	Invalid amount	Vsota je semantično nepravilna.
F04	Insufficient Destination Amount	Prejemnik je prejel premalo sredstev.
F05	Wrong Condition	Pogoj izvedbe transakcije je napačen.
F06	Unexpected Payment	Transakcija je bila nepričakovana.
F07	Cannot Receive	Prejemnik iz nekega razloga ne more sprejeti transakcije.
F08	Amount Too Large	Višina sredstev v paketu je previsoka.

Tabela A.1: Tabela končnih napak zavrnitve transakcije

A.1.2 Začasne napake

Koda	Ime	Opis
T00	Internal Error	Splošna začasna napaka pošiljatelja.
T01	Peer Unreachable	Soležnik je nedosegljiv.
T02	Peer Busy	Soležnik je prezaposlen.
T03	Connector Busy	Konektor je prezaposlen.
T04	Insufficient Liquidity	Konektor nima dovolj sredstev za izvedbo transakcije.
T05	Rate Limited	Pošiljatelj je presegel določeno mejo števila transakcij v nekem obdobju.

Tabela A.2: Tabela začasnih napak zavrnitve transakcije

A.1.3 Relativne napake

Koda	Ime	Opis
R00	Transfer Timed Out	Transakcija je potekla.
R01	Insufficient Source Amount	Višina sredstev v paketu je prenizka.
R02	Insufficient Timeout	Količina časa pred potekom transakcije je premajhna.

Tabela A.3: Tabela relativnih napak zavrnitve transakcije

A.2 Napake BTP

Koda	Ime	Opis
T00	UnreachableError	Začasna napaka, konektor lahko čez čas ponovno poskusi.
F00	NotAcceptedError	Poslani podatki so semantično napačni.
F01	InvalidFieldsError	Vsaj ena poslana vrednost vsebuje strukturno napačne podatke.
F03	TransferNotFoundError	Transakcija s poslanim identifikatorjem ne obstaja.
F04	InvalidFulfillmentError	Poslana izpolnitev pogoja ne ustreza pogoju za izvršitev transakcije.
F05	DuplicateIdError	Identifikator transakcije že obstaja.
F06	AlreadyRolledBackError	Transakcija je že bila preklicana.
F07	AlreadyFulfilledError	Transakcija je že bila izvedena.
F08	InsufficientBalanceError	Konektor nima dovolj sredstev za izvršitev transakcije.

Tabela A.4: Tabela podprtih kod napak protokola BTP

Dodatek B

Konfiguracija

B.1 Konektor Orcus

Atribut	Opis	Privzeto
btp.host	Ime gostitelja strežnika BTP	localhost
btp.port	Vrata gostitelja strežnika BTP	50000
rpc.host	Ime gostitelja strežnika RPC	localhost
rpc.port	Vrata gostitelja strežnika RPC	50000
tls.certificate	Pot do certifikata TLS	/
tls.key	Pot do ključa TLS	/
peers[].ilpAddress	naslov ILP konektorja s katerim se mora konektor avtomatsko povezati	/
peers[].currency	Valuta, v kateri želi pošiljati sredstva soležniku	/
storage.type	Tip podatkovne baze	redis
storage.host	Ime gostitelja podatkovne baze	localhost
storage.port	Vrata gostitelja podatkovne baze	6379
storage.username	Uporabniško ime za dostop do podatkovne baze	/
storage.password	Geslo za dostop do podatkovne baze	/
storage.db	Ime baze v podatkovni bazi	0
plugins	Seznam naslovov vtičnikov, katere želimo, da jih konektor uporablja	[]
maxTxTime	Najdaljši čas izvajanja transakcije	0
maxChannels	Največje število sočasno odprtih plačilnih kanalov	100

Tabela B.1: Tabela konfiguracijskih opcij konektorja

B.2 Vtičnik GateHub

Atribut	Opis	Privzeto
api	URL do API GateHuba	/
apiKey	Avtentikacijski žeton za dostop do API-ja	/
ilpAddressPrefix	Predpona naslova ILP za vse valute, ki jih podpira	/
vaults[].id	ID vault objekta na GateHubu	/
vaults[].gateway	ID gatewaya, na katerem je prisoten *vault* na GateHubu	/
vaults[].address	Naslov denarnice GateHub, iz katere pošiljamo	/

Tabela B.2: Tabela konfiguracijskih opcij vtičnika GateHub