TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Elektronická mýtná brána pod kapotou Blockchain

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Michal Kukla**
Vedoucí práce: Ing. Jan Hybš
Ústav nových technologií a aplikované informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Elektronická mýtná brána pod kapotou Blockchain

Jméno a příjmení: **Michal Kukla**
Osobní číslo: M18000086
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav nových technologií a aplikované informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Seznamte se s transakčním protokolem chytrých kontraktů a technologií decentralizovaných blockchain databází.
2. Navrhněte a realizujte aplikaci demonstrující chytré kontrakty v oblasti elektronických mýtných brán.
3. Demonstrujte výslednou aplikaci na ukázkové trase, definované protokolem GPX.
4. Proveďte bezpečnostní analýzu výsledné aplikace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Bhabendu Kumar Mohanta, Panda, S.S. and Jena, D. (2018). *An Overview of Smart Contract and Use Cases in Blockchain Technology*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/328581609_An_Overview_of_Smart_Contract_and_Use_Cases_in_Blockchain_Technology

[2] Maher Alharby and Aad van Moorsel (2017). *Blockchain Based Smart Contracts : A Systematic Mapping Study*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/319603816_Blockchain_Based_Smart_Contracts_A_Systematic_Mappi

[3] Smart Contracts: 12 Use Cases for Business & Beyond Prepared by: Smart Contracts Alliance -In collaboration with Deloitte An industry initiative of the Chamber of Digital Commerce. (2016). [online] Available at: http://digitalchamber.org/assets/smart-contracts-12-use-cases-for-business-and-beyond.pdf

[4] Bartoletti, M. (2020). Smart Contracts Contracts. *Frontiers in Blockchain*, [online] 3. Available at: https://www.frontiersin.org/articles/10.3389/fbloc.2020.00027/full

Vedoucí práce:	Ing. Jan Hybš Ústav nových technologií a aplikované informatiky
Datum zadání práce:	19. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

Ing. Josef Novák, Ph.D.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

4. května 2022 Michal Kukla

Poděkování

Nejprve bych chtěl poděkovat všem, kteří se podíleli na schválení tohoto zadání pro bakalářskou práci. Poděkování směřuje také vedoucímu práce za vedené konzultace během závěrečné práce.

Elektronická mýtná brána pod kapotou Blockchain

Abstrakt

Cílem Bakalářské práce je demonstrovat blockchain databázi a chytrý kontrakt v oblasti elektronických mýtných brán. Obsahem práce je rešeršní část vysvětlující základy technologie. Praktická část vysvětluje oblast elektronického mýta a výběru poplatků za využití silnic I. třídy. Jedná se o vozidla do 3,5 tuny a nad 3,5 tuny. Za použití GPS souřadnic pomocí GPX protokolu by bylo možné vybrat poplatek jen za ujetou trasu. Výběr mýta jen za ujetou trasu funguje u vozidel nad 3,5 tuny, u osobních vozidel do 3,5 tuny tomu tak nefunguje. Blockchain za pomocí chytrého kontraktu představuje řešení metody platby za trasu. Představené řešení by také mohlo spojit platbu mýta obou kategorií. V této práci jsou znázorněny nejdříve 3 návrhy. Návrhy jsou v dostatečné míře okomentovány zhlediska jejich výhod a úskalí. Z návrhů bylo vybráno jedno z řešení a to názorně zrealizováno. Vybrané řešení umožňuje vytvořit mýtnou bránou na jednotlivých bodech GPS souřadnic a následně uživatel si dokáže stáhnout informace o přítomnosti platbě mýta pomocí GPS souřadnic. Realizace byla vytvořena a testována na soukromé testovací síti Ethereum blockchain. Výsledkem se stalo řešení, které cílí na minimalizování počtu transakcí, zlepšení pseudoanonymity a rychlost. Při návrhu nebyla snížena míra bezpečnosti na úrovni blockchain nebo chytrého kontraktu. Aplikace dokáže z definované trasy protokolem GPX zpoplatnit trasu. Zpoplatnění je definované na jednotlivý bodech GPS souřadnic.

Klíčová slova: řetězec bloků, chytrý kontrakt, dohoda, databáze, elektronické mýto

Blockchain under the hood of E-toll

Abstract

Goal of the Bachelor thesis is to demonstrate blockchain database and smart contract in the area of e-toll. The thesis starts with a research, which explains basics of the blockchain technology. Next is practical part explaining functionality of current e-toll in Czech republic. The first system of e-toll is for lorries declared as vehicles above 3,5 tonne. The other system is for cars up to 3,5 tonne. With the help of GPS coordinates and GPX protocol is possible to pay fees for only driven route. Similarly works the system for lorries, the system for cars does not work in this way. Blockchain with smart contracts introduce the solution of payment for only driven route. The thesis also represents the solution for joining those two systems of e-toll in area of payments. Firstly in the thesis are described 3 proposals. The proposals are carefully discussed about their benefits and drawbacks. The one proposal is then chosen and implemented. The chosen proposal allows to create toll gate on each GPS coordinate seperately. User is able to download the information about presence of a toll road by GPS coordinates. Implementation was created and tested on a local blockchain network. The result of the implementation is the solution that aims for better pseudoanonymity and overall performance of the whole network. There is also no decreased of security level on a layer of blockchain nor smart contracts. Application demonstrates simulation of driving on a toll road and on top of that creating payment for the toll. Simulation of driving is done by using GPX protocol. Toll gate can be created for each GPS coordinate separately.

Keywords: blockchain, smart contract, consensus, database, e-toll

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam zkratek

Hash je hexadecimální řetězec s pevnou délkou vytvořený pomocí libovolného řetězce s variabilní délkou. K vytvoření hash se používá funkce např. sha3. V menší modifikaci se používá pod názvem keccak256.
SPZ Státní poznávací značka
JSON JavaScript Object Notation
XML eXtensible Markup Language

1 Úvod

V nedávné době došlo k digitalizaci dálniční známky pro vozidla do 3,5 tun, což dalo námět na tuto práci. Platba funguje pomocí koupě elektronické kupónu na určitou dobu (1 rok, 30 dnů, 10 dnů). Kupón se vztahuje k určité SPZ. Otázka ale zůstala nezměněna, proč by dva různí účastníci provozu by měli platit za silnici I. třídy stejně bez ohledu na ujetou vzdálenost? Proč platit i když se nemohlo jezdit[1]? Aktuální růst inflace(2022) by mohla nasvědčovat, že účastník provozu si může další rok rozmyslet koupi dálniční známky[6]. Což může zapříčinit nárůst hustoty dopravy ve městech a jejich okolí. Silnice I. třídy byly právě stavěny pro snížení hustoty dopravy na silnicích nižších tříd. Tomu by mělo zůstat i nadále. Tato práce dává řešení ve formě platby jen za ujetou trasu. S technologií sítí 5G by tato práce mohla být i jednodušeji realizovatelná, protože se sitím také přichází aplikační vrstva Vehicle-to-everything(V2X).

Technologie blockchain už tu funguje přes jedno desetiletí a má tendenci stále se vyvíjet. Jedna z hlavních vlastností, kterou tuto technologii odlišuje od ostatních, je absence vkládání důvěry v kohokoliv síti, že dělá věci tak, jak má. To je umožněno pomocí dohody(consensus) vepsané v algoritmu. Důvěra je tímto algoritmem nahrazena důkazem a ověřením. Jedinou třetí stranou pověřenou ke zpracování transakcí je samotná blockchain síť, ale bez vložení jakékoliv důvěry v kohokoliv. Aplikace používající blockchain se označují jako za decentralizované.

V průběhu práce je používán Ethereum blockchain. K vývoji chytrých kontraktů je použito vlastní vývojové prostředí. Vývoj probíhá na soukromé blockchain síti. Soukromá síť je zprovozněna klientem Geth[1] napsaný v jazyce Go. Síť používá consensus algoritmus PoW - důkaz práce. Chytrý kontrakt je psán v programovacím jazyce Solidity[2]. Použitá verze je 0.5.17 Solidity.

K vývoji chytrých kontraktů se nabízí řešení v podobě webového prostředí s názvem Remix IDE. Je to editor všemi potřebnými nástroji kolem vývoje kontraktů. Takové to prostředí se postará o všechny náležitosti kolem spouštění, testování a nasazování kontraktů. Vývoj decentralizované aplikace je poté mnohem rychlejší. Nevýhodou může být stálé připojení k internetu. Dalším poměrně užitečným nástrojem je Ganache. Nástroj nabízí podobné prostředí, ale není nutné mít stálé připojení k internetu.

Během práce nebyl použit ani jeden z výše uvedených nástrojů. Z důvodu k lepšímu poruzumění a prohlubení znalostí v blockchain technologii nebývá nic lepšího

[1] https://geth.ethereum.org/
[2] https://docs.soliditylang.org/

než použít samotnou technologii bez přidaných pomůcek. To určitě zpomalí vývoj, ale nabité znalosti mohou být později užitečnější.

Na propojenost a komunikaci všech klientů v síti je odkazováno jako na blockchain síť. Termínem blockchain se odkazuji na seznam spojených bloků obsahující transakce. V této práci nepoužívám termín kryptoměny, protože kryptoměna nemusí znamenat blockchain nebo blockchain síť. Kryptoměna může být chytrý kontrakt, nebo blockchain.

2 Blockchain - Distribuovaná účetní kniha

Blockchain je možné analogicky přirovnat účetní knize na papíře. Do účetní knihy se zapisují všechny proběhnuté transakce a dává přehled o rozpočtu. O to samé se snaží blockchain jen v elektronické podobě s důrazem na bezpečnost už v základu. Existují protokoly pro weby(http), soubory(ftp), e-mail(imap, pop3, smtp), vzdálený přístup(ssh,telnet), tak i existuje protokol pro uchování měny skrz internet - blockchain. Blockchain síť je protokol pro uchování hodnoty a dat, zároveň komunikuje s dalšími klienty pro zajistění decentralizovanosti, neměnitelnosti a transparentnosti. V takovém protokolu jsou definované adresy pro držení hodnoty, mince, čísla, data podle toho, jak je konkrétně blockchain definován.

2.1 Počátek

V roce 2008 byl zveřejněn Bitcoin whitepaper[1] autorem pod pseudonymem Satoshi Nakamoto[9], což se stalo základem pro další blockchain sítě. Nedávné události nasvědčujou, že autorem by měl být Craig Wright [10]. Hlavním hnacím motorem, proč vytvořit něco takového, byla světová ekonomická krize v roce 2007. I když první zmínky o elektronický pěnězích tzv. b-money je z roku 1997[2] a Chaum 1994. Dalším důležitým milníkem je rok 2014, kdy byl zvěřejněn blockchain s názvem Ethereum. Ethereum pracuje s myšlenkou, že vše co se ukládá do účetní knihy jsou jen čísla. Zároveň když zkompiluji kód například v jazyce C, získám posloupnost nul a jedniček tj. posloupnost čísel. Místo posílání peněz z adresy na jinou adresu se jedná o uložení libovolného programu. Nejlepším na tom je, že každý stav toho programu, ať se třeba jedná o nastavení nějaké proměnné, je uložen v blockchain. Technologie se nazývá chytrý kontrakt.

O chytrém kontraktu se prvně zmiňuje v článku autor Nick Szabo v roce 1996 [11]. Szabo nezmiňuje blockchain nebo decentralizovanost. Naopak chytrý kontrakt by měl být podle něho spravován jednim centralizovaným serverem. Jde o vnoření chytrého kontraktu do hardware nebo software. Příklad, na kterém to vysvětluje je, když osoba si pronajme vozidlo. Pokud splátky splácí, má v chytrém kontraktu nadefinováno, že má právo vozidlo využívát. Pokud ne, tak vozidlo propadne zpět původnímu majiteli. Poodkryl také svoji vizi o digitální identitě. To vše ještě na konci minulého století.

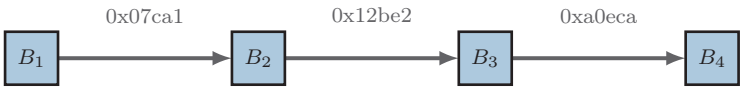
Podle Liptona [8] myšlenka blockchain určitě není nová. Znázorňuje to například

[1] Zpráva pojednávající o komplexním problému[12]

u rodu Hasburků, kdy předávání majetku a trůnu se řídilo podle předem definovaných pravidel.

2.2 Neměnitelnost

Název blockchain se skladá ze samotné struktury technologie - řetězec bloků. Bloky jsou zřetězeny pomocí hash z předchozího bloku, tak jak je znázorněno na obrázku 2.1. $B_1, .., B_4$ jsou jednotlivé bloky. Blok B_1 je nejstarší, blok B_4 je nejnovější. Vypsaný hash je vždy toho předchozího bloku, ke kterému hrana vede. Například hash `0x07ca1` pochází z bloku B_1, hash `0x12be2` pochází z bloku B_2...atd. První blok v blockchain síti se nazývá Genesis (angl. počátek). V terminologii grafů se tomu nazývá kořen. Je to speciální blok, který jako jediný nemá rodiče v celém seznamu. Nejnovějšímu bloku se říká list, protože nemá potomka.



Obrázek 2.1: Neměnitelný spojový seznam.

Hash je vypočítán z obsahu bloku, tudíž nějaká změna v bloku vytvoří rozdílný hash. Ten hash se poté vloží do nadcházejícího bloku a z obsahu(včetně hash) bloku opět vypočítá hash pro nastávající blok. To znamená, že změnu obsahu některého bloku z řetezce by znamenalo přepočítat všechny hashe v navazujících blocích. Pokud by hash neodpovídal v některém z bloků, tak by i další bloky na sebe nenavazovaly. Obsah bloku je tvořen transakcemi. Tímto postupem je zatím zjednodušeně vysvětlena neměnitelnost. Lze tedy definovat, že blockchain je neměnitelný spojový seznam. Blockchain síť poté zajišťuje decentralizovanost.

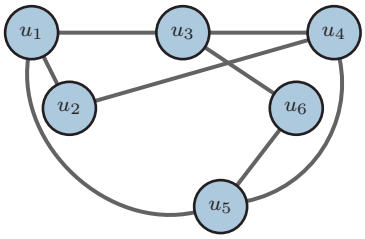
```
1 > eth.getBlockByNumber(105)
2 {
3 hash: "0x7873....2382",
4 nonce: "0x5ed1....ecda",
5 parentHash: "0x88d7....56d8",
6 transactions: ["0x9240....ed1f",...],
7 }
```

Zdrojový kód 2.1: Blok v ethereum

Příkaz z kódu 2.1 vypisuje obsah bloku. Mezi nejdůležitějšími atributy bloku pro teď jsou `hash`, `parentHash` a `transactions`. Atribut `hash` je toho daného bloku a předchozí(rodič) je `parentHash`. Právě tyto dva atributy zaručují návaznost. `Transactions` je pole transakcí opět má svůj hash.

K blockchain se přistupuje přes Javascript konzoli. Vše je zobrazováno v JSON formátu.

2.3 Decentralizovanost



Obrázek 2.2: Ukázka propojení klientů blockchain sítě

Stavebním prvkem blockchain sítě jsou klienti. Můžeme si představit, že jde o program(klienta), který neustále komunikuje s dalšími uzly(klienty) sítě a přeposílají si aktuální stav učetní knihy. Každý klient je server v blockchain síti. Odpojením jednoho z uzlů ze sítě nedojde k nefunkčnosti celé sítě, protože každý uzel má u sebe uloženou kopii účetní knihy. Záznam účetní knihy zůstává zachován v ostatních připojených uzlech. Každý se může do takové sítě připojit. Poté tu jsou sítě s centrálním úzlem, jejíž úkolem je kontrolovat kdo má oprávnění se do sítě připojit. Taková to síť už ale nepředstavuje plnou decentralizovanost.

```
1 > geth --georli --datadir "vas_adresar/" --syncmode "light
    "
```

Zdrojový kód 2.2: Připojení se k testovací síti blockchain

Příkaz 2.2 se připojí k testovací síti s názvem georli, pro lepší přehled je možné zadat adresář kam se uloží blockchain `vas_adresar`. Pro parametr `syncmode` je nutné prozatím zadat `light`, což zapříčiní stažení hlaviček bloků. Synchronizační proces může trvat pár minut, hlavičky bloků mohou zabrat kolem 0.5 GB. Takový to běžící klient ještě skoro nic nedělá, jenom si zapisuje, jaký další bloky byly přidány do blockchain.

```
1 > geth attach ipc:geth.ipc
```

Zdrojový kód 2.3: Připojení javascriptové konzole ke klientovi

Po spuštění klienta se vytvoří soubor `geth.ipc` v adresáři `vas_adresar/`. I když se to zdá jako soubor, spíš jde o komunikační kanál [13]. Skrz tento soubor můžeme komunikovat s klientem pomocí příkazu 2.3. Kód 2.4 ukáže ke komu je náš klient připojen.

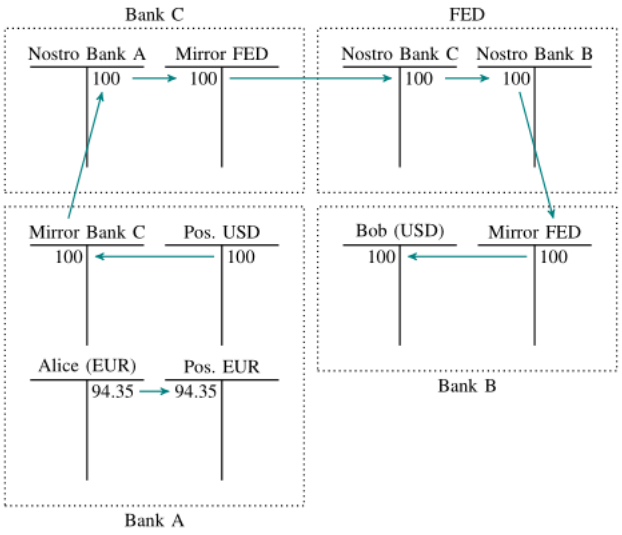
```
1 > admin.peers
```

Zdrojový kód 2.4: Výpis klientů, ke kterými jsme připojeni.

2.4 Bez důvěry (trustless)

Pokud v dnešní době posíláte peníze z účtu jedné banky na účet v druhé bance, vkládáte důvěru v bankovní systém, že peníze se odečtou a přičtou na správný účet. Jinými slovy k provedení jednoduché transakce je nutný zprostředkovatel neboli třetí strana(banka). Vkladána důvěra v třetí stranu, že dělá vše tak, jak má. U blockchain každý připojený uzel ověřuje, zda podpisy transakcí a hashe bloku souhlasí. Žádný uzel v síti nevkládá důvěru a nevěří žádnému dalšímu uzlu. Jinak řečeno, vše si každý uzel zkontroluje sám. Toto nové paradigma dovoluje obchodovat mezi dvěmi a více stranami, které si navzájem nedůvěřují a to bez absence zprostředkovatele(třetí strana). Tato klíčová vlastnost může zjednodušit veškeré jednání mezi stranami. V porovnání s aktuálním platebním systémem je nutno důvěřovat nejméně dvěma stranám, pokud se například platí v obchodě kartou. Zprostředkovatel karet např. Visa a samotná banka. Další strana může být populární placení přes Google Pay nebo Apple Pay.

Na obrázku 2.3 je znázorněno kolika zprostředkovatelům(bankám), Alice musí důvěřovat, aby mohla poslat peníze Bobovi na účet ve Spojených Státech. Můžeme vidět, že peníze proputují 4 bankami. Zde je ukázana jenom zjednodušená verze, tudíž těch bank a prostředníků může být ve skutečnosti ještě víc. Pokud by Alice platila přes bitcoin, transakce by zůstala jenom v jedné síti a potvrzená může být do 30 minut. Bez vložení důvěry v kohokoliv, že věci dělá tak jak má.



Obrázek 2.3: Alice posílá z evropské banky 94 euro Bobovi na účet v bance ve Spojených Státech. Aktuální bankovní systém. Zjednodušená verze.[14]

Pokud se podepisuje kupní smlouva, jsou dvě možnosti na výběr. Buď si celou smlouvu přečíst a zkontrolovat, nebo důvěřovat tomu, kdo psal kupní smlouvu. S kým se uzavírá smlouva, pokud dotyčný je upřímný, tak nebude vyžadovat důvěru k

podepsání. Pokud se důkladně čtou a kontrolujou všechny smlouvy co se podepisují, nevkládá se důvěra v ostatní autority. Takhle přesně funguje každý uzel blockchain sítě.

2.5 Transparentnost

```
1 > eth.getBalance(0x301e....e344)
2 '0'
```

Zdrojový kód 2.5: Zůstatek na účtu v jednotce Wei

U většiny sití je obsah blockchain transparentní. To znamená, že každý se může podívat na web např. bitcoin prohlížeč[2] nebo etherscan[3]. A zjistit si jakákoliv adresa kam posílá měnu dané sítě. Dokonce jaká adresa kolik vlastní měny. Ani u kontraktu tomu není jinak a tak každý má možnost číst data z jakéhokoliv kontraktu. Zároveň tímto způsobem, je každá transakce jednoduše sledována. To zapříčinuje zdanění kryptoměn v jednotlivých státech. Existují také sítě netransparentní, u kterých není možné zjistit pohyb měny na účtech druzích. Jenom odesílatel a příjemce ví o transakci. Příkaz 2.5 umožňuje vypsání zůstatku na jakékoliv adrese.

2.6 Adresy

Adresy se dělí na externě vlastněné a adresy kontraktů. Oba typy adres mohou držet měnu. Adresy externě vlastněné se skládají z veřejného a soukromého klíče. Soukromý a veřejný klíč se používá k asymetrickému šifrování. Jednim klíčem se zpráva šifruje a druhým dešifruje. Není možné šifrovat a dešifrovat jednim klíčem. Soukromý klíč je chráněn heslem.

Veřejný klíč v blockchain je ekvivalent k veřejné adrese a soukromý klíč poté slouží k útratě zůstatku na veřejné adrese. Soukromým klíčem se podepisuje transakce. Veřejnou adresou dojde ke zpětnému ověření transakce k útratě měny na adrese. V ethereum síti je používán hash algoritmus keccak256. Používá se také při vytváření adres. Jedná se o odnož algoritmu sha3 s malými úpravy. Pro vytvoření adresy se obvykle používá peněženka např.[4]. Jednou z hlavních výhod je, že při vytváření účtu se vygeneruje `seed passphrase`. Jedná se o pseudonáhodnou posloupnost libovolných slov. Takových slov pro adresu může být vygenerováno kolem 12. Pokud se ztratí zařízení s přístupem do účtu nebo dojde ke ztrátě hesla, tak seed passphrase dokáže získat přístup ke své adrese nehledě na jakém zařízení účet byl vytvořen.

Příkaz 2.6 vygeneruje soukromý a veřejný klíč a poté se zeptá na vložení hesla pro zabezpečení soukromého klíče. Výstup příkazu je poté uložen v adresáři `vas_adresar`.

[2]Bitcoin - https://www.blockchain.com/explorer
[3]Ethereum - https://etherscan.io/
[4]Bitcoin Electrum Wallet - https://electrum.org/#home

```
1 > geth account new --datadir vas_adresar/
```

Zdrojový kód 2.6: Vytvořit účet v ethereum

2.7 Poplatky

```
1 > eth.gasPrice
2 1000000000
```

Zdrojový kód 2.7: Hodnota Gas v jednotce Wei

Blockchain sítě mají, jako centrální platební systém, poplatky za transakce. Ty zajišťují bezpečnost a eliminují spamovací útoky. Poplatky jsou také jako jeden z příjmů pro klienty co přidávájí bloky a transakce do blockchain. Ten další příjem je pak tvorba nových mincí. Poplatek se vyměřuje podle toho, jak výpočetně náročná transakce je. Ethereum má pro to jednotku, které se říká gas. Zde se jedná o analogii paliva v autě. Jeden Gas má nějakou hodnotu a je potřeba několik takových jednotek pro transakci. Konkrétněji v ethereum síti je potřeba 21 000 Gas jednotek pro odeslání měny na jinou adresu. Počet Gas jednotek určuje, jak moc procesorově náročné je zpracování transakce. Pro vložení kontraktu do ethereum se pohybujeme v řádově od sta tisíců až k několika milionů Gas jednotek podle rozsáhlosti kontraktu.

Poplatky je jedna z příčin, co dělá blockchain nepoužitelný pro běžnou platbu. Konkrétněji to dělá cena za jednu Gas jednotku. Za běžnou transakci můžete zaplatit v přepočtu kolem 300,- Kč a ve špičce se cena transakce může vyšplhat přes 1000,- Kč a víc.

2.8 Škálovatelnost

```
1
2 static const unsigned int MAX_BLOCK_SIZE = 1000000;
```

Zdrojový kód 2.8: Řádek všech řádků v blockchain sféře. Řádek, který od 2010 mění svět blockchain technologie.

U blockchain se škálovatelnost měří podle toho, kolik transakcí je možné zpracovat za 1 sekundu(TPS - transactions per second). Píše se rok 2010 a Satoshi Nakamoto dál vyvíjí už běžící bitcoin. V té době šlo ještě o vedlejší projekt poskladaný po večerech. Tudíž probíhalo jenom pár transakcí za hodinu a hodnota bitcoin měny byla kolem nuly[4], takže největší hrozbou sítě byly spamovací útoky. Proto byl už před tím

zaveden právě řádek s maximální povolenou velikosti bloku2.8, která činí 1 MB. Což bylo na dlouhou dobu velice dostačující limit. Nicméně ve 2010 některý vývojáře už tento limit znepokojoval a tak byla navrhnuta záplata s variabilní velikosti bloku viz kód2.9.

```
1  diff --git a/main.h b/main.h
2  index c5a0127..c92592a 100644
3  --- a/main.h
4  +++ b/main.h
5  @@ -14,7 +14,10 @@ class CBlockIndex;
6   class CWalletTx;
7   class CKeyItem;
8
9  -static const unsigned int MAX_BLOCK_SIZE = 1000000;
10 +static const unsigned int TX_PER_MINUTE = 1400;
11 +static const unsigned int TX_AVG_SIZE_GUESS = 256;
12 +static const unsigned int MAX_BLOCK_SIZE =
13 + TX_PER_MINUTE * TX_AVG_SIZE_GUESS * 10 * 2;
14  static const unsigned int MAX_BLOCK_SIZE_GEN =
       MAX_BLOCK_SIZE/2;
15  static const int MAX_BLOCK_SIGOPS = MAX_BLOCK_SIZE/50;
16  static const int64 COIN = 100000000;
```

Zdrojový kód 2.9: Záplata z roku 2010

Kód odstraňuje konstatní velikost bloku 1 MB. Výpočet velikosti bloku spočívá ve vynásobení `TX_PER_MINUTE` a `TX_AVG_SIZE_GUESS`. Proměnná `TX_PER_MINUTE` obsahuje počet transakcí za minutu. Proměnná `TX_AVG_SIZE_GUESS` obsahuje průměrnou velikost transakce.

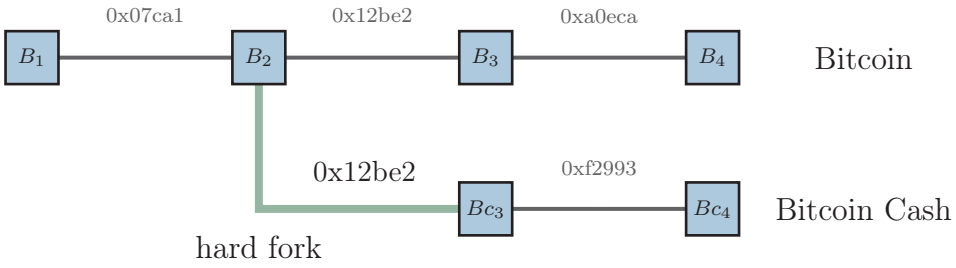
Záplata byla samotným Satoshim zamítnuta, protože by verze s touto záplatou by byla nekompatibilní s ostatními verzemi klienta. Satoshi argumentuje tím, že bude možné v průběhu let tento limit zvýšit [3]. Problém je v tom, že aktualizovat kód klienta není tak jednoduché jako u programů klient-server. V blockchain každý klient je serverem a zároveň klientem. K tomu aby klient s novější verzí se mohl připojit do sítě, tak i ostatní uzly musejí mít stejnou verzi klienta. Ostatními uzly se myslí většina. Jde o dohodnutí, kdy většina uzlů bude mít stejnou verzi. Problém s velikostí bloků má většina sítí ošetřeno. Opak se stal pravdou a v případě bitcoin se tímto limitem se nedá hnout. Tento řádek kódu zapříčinil

- mizernou škálovatelnost,
- vysoké poplatky za transakci,
- prodlevy ve schválení transakce,
- tvorba nových blockchain sítí řešících tento limit(např. Bitcoin Cash, Ethereum),

- tvorba nových vrstev a aplikací nad bitcoinem(Lightning network[5]).

A to vše jen kvůli jednomu řádku kódu. Jsou názory pro a proti, proč (ne)zvýšit limit. Větší bloky by podle některých zahltilo celkovou internetovou síť. Aktuálně centrální platební systém jako například PayPal dokáže ve špičce zpracovat 50 000 transakcí za sekundu. V současnosti Bitcoin dokáže jenom 6 transakcí za sekundu a ethereum kolem 12 transakcí za sekundu. Je to veliký rozdíl mezi centralizovaným a decentralizovaným systémem. Ve zkratce všechny ostatní blockchain sítě se snaží najít lepší kompromis mezi decentralizováním, škálováním, bezpečností a nižšími poplatky za transakce.

2.9 Aktualizace blockchain - hard a soft fork



Obrázek 2.4: Hard fork v podobě Bitcoin Cash

Jakýkoliv software běžící v internetu je nutné aktualizovat, není tomu jinak u blockchain sítě. Avšak proces aktualizace je úplně odlišný než u klasického client-server architektury. Otázka zní, jak aktualizovat tisíce propojených klientů skrze internet? K tomu je potřeba dohodu v podobě kódu v klientovi. Například, že od nějakého bloku bude potřeba používat zaktualizovaného klienta, jinak se klient nepřipojí do sítě. K aktualizacím se právě vážou termíny soft a hard fork. Soft fork by se dalo přeložit jako odlehčená aktualizace, kdy nedojde k nekompatibilnosti s původní sítí. Vůbec pod aktualizací můžeme představit, že jde o samotnou údržbu kódu tak i úprava samotné vepsané dohody v kódu. To může ovlivnit jaké transakce a bloky jsou považovány za validní. Hard fork aktualizace právě vytvoří nekompatibilnost s původní sítí. Příkladem může být síť Bitcoin Cash. Bitcoin Cash je hard fork sítě Bitcoin. Tyto dvě sítě jsou nekompatibilní z důvodu velikosti bloků, které můžou být zapsány v blockchain. Bitcoin Cash dovoluje bloky 8 MB velké, Bitcoin jenom kolem 1 MB.

[5] https://lightning.network/

3 Chytrý kontrakt

```
1
2  // SPDX-License-Identifier: GPL-3.0-or-later
3
4  pragma solidity ^0.5.17;
5
6  contract SimpleStorage {
7      uint storedData;
8
9      function set(uint x) public {
10         storedData = x;
11     }
12     function get() public view returns (uint) {
13         return storedData;
14     }
15 }
```

Zdrojový kód 3.1: Hello World kontrakt

Chytrý kontrakt je libovolný kód uložen v blockchain jako názorná ukázka3.1. Jedná se o kontrakt, který dovoluje komukoliv v síti nastavit proměnnou `storedData` pomocí metody `set` a zároveň ji i číst metodou `get`. Pro vykonání kontraktu se používájí operační kódy[1]. K vykonání takového kódu slouží ethereum virtuální stroj(EVM). Ten se nachází v každém připojeném klientovi.

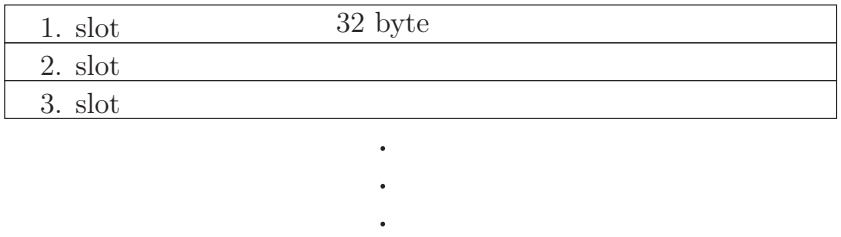
V rámci sítě Ethereum se kód nejčastěji píše v programovacím jazyce Solidity. Jde o vysoko-úrovňový, kompilovací jazyk se zaměřením na kontrakty. Jediný rozdíl je, že místo objektů se píšou kontrakty. Každý stav v kontraktu je zaznamenán v blockchain podobě transakce. I když je to vysoko-úrovňový jazyk, tak lze psát kód ve strojovém jazyce tzv. assembly. Důvodem pro použití assembly může být snížení ceny nasazení kontraktu do blockchain.

Kontrakty nejsou určené pro náročný výpočet, velký počet cyklů atp. Hlavním důvodem jsou vysoké ceny transakcí za provedení náročné operace.

[1] Operační kódy - https://ethervm.io/

3.1 Paměť a úložiště

V kontraktu funguje uložení dat na dočasné místo tj. paměť. K tomu se používá modifikátor `memory`. Dočasné místo znamená, že data v tom místě existují jen v průběhu vykonávání kódu. Poté se vymažou. Paměť se používá pro funkční parametry a navratový hodnoty funkcí.



Obrázek 3.1: Úložiště v blockchain pro chytrý kontrakt

Na konec je možno data uložit i na trvalé místo, na které se zapíšou transakcí do blockchain. Takovému místu se říká úložiště (angl. storage). Úložiště se dělí na sloty potažmo zásobníky. Každý slot má velikost 32 byte viz obrázek 3.1. Prostředí blockchain začíná přidělovat úložiště atributům kontraktu od 1. slotu. To znamená viz kód 3.1, že atributu s názvem `storedData` bude deklarováno místo v 1. slotu. Pokud jsou proměnné menší než je velikost slotu, může být v jednom slotu více proměnných.

3.2 Mapování

Mapování je hash tabulka, kde každý hash je virtuálně inicializovaný na samé nuly při deklaraci. Datový typ se nazývá `mapping` viz kód 3.2. Pro vytvoření hash se používá funkce keccak256.

```
1 mapping(address => uint256) public balances;
```

Zdrojový kód 3.2: Použití mapování pro přiřazení celého čísla k adrese

Místo kam se uloží data je spojení čísla slotu mapovaní a klíče. Na to se následně použije hash funkce keccak256. Výsledný hexadecimální řetězec je adresa úložiště, kde data jsou uložená. K ověření slouží funkce getStorageAt. Mapování je možné ukladát jenom do úložiště, nikoliv do paměti. Tudíž není možné použít mapování jako navrátovou hodnotu, nebo parametr funkce.

3.3 Transparentnost kontraktu

K prozkoumání úložiště kontraktu se používá příkaz 3.3. Prvním argumentem je `slot` a druhý je `adresa kontraktu`. Funkce vrátí data uložená ve slotu v hexadecimální soustavě.

```
1 > eth.getStorageAt(1, contractAddress);
```

Zdrojový kód 3.3: Příkaz pro výpis dat z úložiště

4 Consensus

Slovo consensus by se dalo volně přeložit jako dohoda v našem kontextu. V podstatě jde o dohodnutí se mezi uzly sítě na tom, jaké bloky a transakce samotné uzly schvalují a je pak tedy možné přidat transakci do bloku a blok na konec posledního validního bloku. Dohoda je vepsána v kódu každého klienta. To vše bez vkladání důvěry v kohokoliv. Zde dochází ke změně paradigma, kdy důvěra je nahrazena důkazem a ověřením. Proto je celá síť tzv. trustless.

4.1 Proof Of Work - PoW

Jedná se o vyřešení matematické rovnice. Nalézt číslo(od teď jenom nonce), který ve výsledku se rovná správnému výstupu(hash). Nonce je číslo použité jednou (angl. number used once).Tímto je zajištěna neměnitelnost. Ten, kdo hledá nonce se nazývá miner. U proof of work se měří kolik takových možností vyzkouší výpočetní jednotka(CPU, GPU a FPGA) za sekundu - počet vygenerovaných hash řetězců za sekundu(Hash rate per second).

Algoritmus má parametr obtížnost. Obtížnost udává jak dlouho bude trvat najít nonce. Parametr je zároveň závislý na hash rate celé sítě. Bitcoin vytváří nový blok každých 10 minut. Důvod proč zrovna 10 minut je ten, aby každý klient připojen v síti stihnul aktualizovat svůj blockchain [9]. Například u Bitcoinu dochazí neustálému vyvažování obtížnosti podle hash rate, tak aby vytvoření nového bloku trvalo cca 10 minut. Pokud stoupne hash rate sítě, stoupne i obtížnost. Pokud klesne hash rate sítě, klesne i obtížnost. Jinak řečeno, hash rate stoupne a tím se rychleji vytvoří nový blok v blockchain, proto zvyšíme obtížnost. Mezi hash rate a obtížností vládne přímá úměra.

Dalším důležitým parametrem je kolik práce(energie) bylo vynaloženo na výpočet konkrétního bloku. Tento parametr souvisí s rozlišením jaký blockchain je ten pravý v síti. Právě blockchain, na kterým bylo vynaloženo nejvíce energie a to znamená také, že je nejdelší ze všech. Všichni uzle stavějí další blok nad nejdelším blockchainem, na kterém bylo vynaloženo nejvíce energie.

Důkaz prace je velmi silným zároveň jednoduchým nástrojem jak zachovat blockchain neměnitelný. Vysoká daň je především za velkou spotřebou elektrické energie, která je vyplýtvána za prosté generování náhodných čísel. [1]

[1] Pro těžbu na území ČR je potřeba mít živnostenský list.

4.2 Proof Of Stake (PoS)

Pokud důkaz práce je velice náročná na spotřebu elektrické energie, tak následující algoritmus je jeho řešením. Algoritmus důkaz kapitálu má za úkol nahradit důkaz práce. V tomto typu jsou jenom validatoři. Jde o vložení nějaké částky do algoritmu. Částka se poté uzamkne v síťi. Po uzamčení častky se účet stane validátorem. Síť poté pseudonáhodně vybere ověřovatele, který ověří a přidá další blok na konec blockchain. Ostatní oveřovatelé poté kontrolují(hlasují) jestli jsou pro nebo proti přidání bloku na konec blockchain. Pokud je většina souhlasí, block se přidá do blockchain. Validace totiž není ani z daleka tak naročná jako PoW algoritmus. Tudíž je možné validaci provozovat i na starším počítači. To znamená, že kdokoliv může být ověřovatelem. Tímto ztrácejí smysl velké haly s velkým výpočetním výkonem pro těžbu.

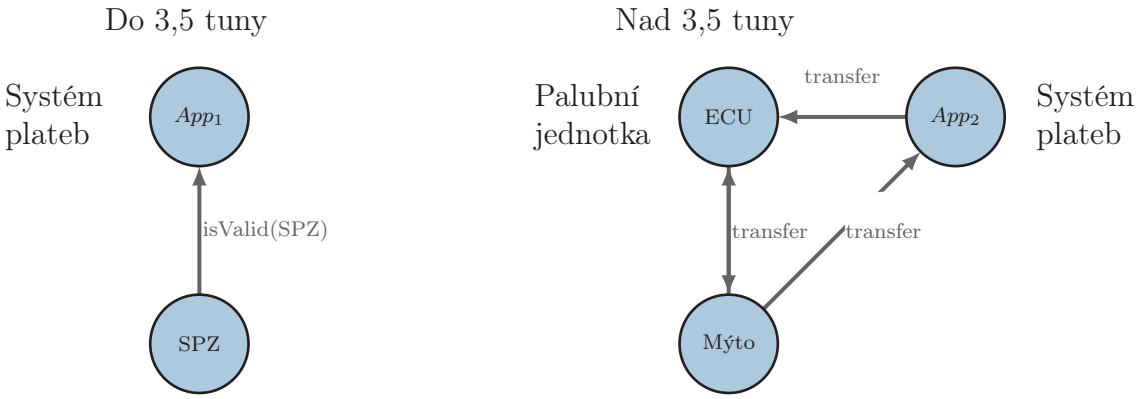
V tuto chvíli (únor 2022), je v provozu síť ethereum 2.0. Jde o velice rannou verzi síťě s PoS algoritmem. Celkově jde vývojářům a celé komunitě zvýšit škálovatelnost sítě pro větší propustnost transakcí za sekundu. Mluví se až o 100 000 transakcí za sekundu, které bude možné validovat přidávat do blockchain. To už dokáže konkurovat centralním platebním systémům. Finální verze by měla přijít někdy koncem roku 2023, ale očekává se další zpoždění termínu, protože zde jde o velice komplexní řešení, které nechce nikdo uspěchat.

4.3 Proof Of Authority

Tento konsensus algoritmus je celkově kontraproduktivní vůči blockchain jako takovému. Jedná o autoritu, která má právo validovat bloky do blockchain. Stejné jako u Proof of Stake, jen kdo bude validovat je předem určený. Autorita se obvykle určí při vytvoření sítě. Z důvodu kontrakproduktivnosti se používá jenom u testovacích sítí.

5 Elektronické mýto

Elektronická mýtná brána má za úkol vybrat poplatek za využití silnic I. třídy pro vozidla nad 3,5 tuny. Pro motorová vozidla do 3,5 tuny existuje elektronická dálniční známka. Je nezbytné, aby takové systémy vytvářely zisk, jak pro údržbu a opravu silnic tak i pro údržbu samotného systému.



Obrázek 5.1: Metody výběru poplatků

Na obrázku 5.1 je zobrazen aktuální systém výběrů poplatků pro vozidla do 3,5 tuny a nad 3,5 tuny. Vrchol s názvem App_1 je systém platby pro vozidla do 3,5 tuny. Systém umožňuje si koupit na SPZ dálniční známku dobou platnosti na 1 rok, 1 měsíc nebo 10 dnů. Vrchol s názvem SPZ je kontrola ověření platnosti dálniční známky. Kontrola je prověděna pomocí kamerového systému pro rozpoznávání SPZ. Kontrola se poté dotáže systému edalnice ohledně zaplacené známce.

U vozidel nad 3,5 tuny to je jiné. Každé vozidlo má palubní jednotku s kreditovým systémem. Kredit si majitel vozidla dobije přes aplikaci a poté se přičte do palubní jednotky skrz mobilní síť. Jízdou pod mýtnou bránou se poplatek odečte z palubní jednotky vozidla viz obrázek 5.1.

Řešení představené v této práci by mimo jiné umožnilo tyto dva systémy spojit do jednoho. Výměra poplatku může být vepsána v chytrém kontraktu. Každý si může zjistit, jak poplatek byl vyměřen.

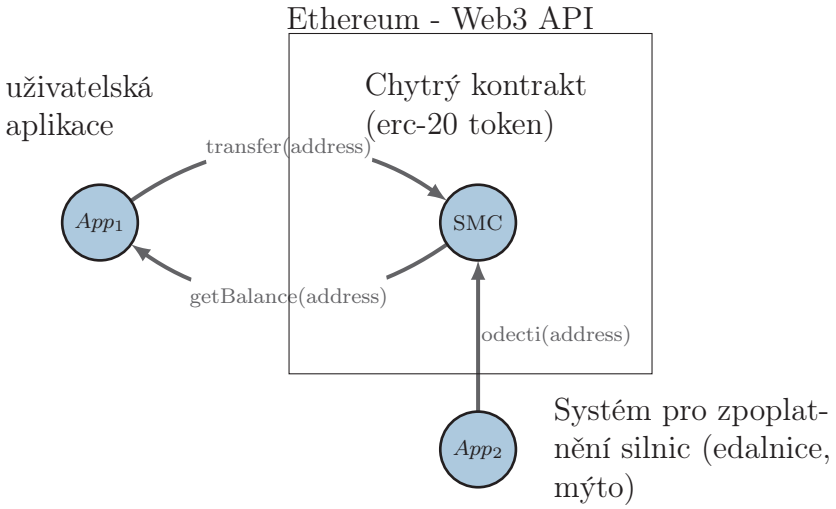
5.1 ERC-20 token

Jedno z řešení, které nabízí samotná síť ethereum, je token ERC-20. ERC-20 je standard pro tokeny k výměně za měnu ethereum sítě. Token je chytrý kontrakt. Existují i webové stranky pro tvorbu ERC-20 tokenu, tudíž každý si může vytvořit svůj vlastní token. Kontrakt je považován za validní ERC-20 token, pokud kontrakt dědí od rozhraní `IERC20`. Základem kontraktu je atribut s datovým typem `mapping` viz kód 5.1. Atribut `balances` má uložen zůstatek pro každou možnou adresu. Při deklaraci je celý interval možných adres automaticky inicializován na 0. Atribut v kódu 5.1 je deklarován jako soukromý. Ostatní kontrakty tento atribut nebudou vidět, ale každý klient v blockchain síti se může podívat do úložiště prozkoumat proměnnou co obsahuje, jak je znázorněno v sekci 3.3.

```
1
2     mapping (address => uint256) private _balances;
```

Zdrojový kód 5.1: Zůstatek na adrese

Adresa pomocí ERC-20 tokenu je propojena s majitelem vozidla. Majitel koupí token, který se přičte na adresu v kontraktu. Pokud vozidlo projede po zpoplatněné trase, odečte se mu poplatek ve výši ujeté vzdálenosti. Výpočet poplatku může být zadefinován v kontraktu. Každý se může podívat na internetu do kontraktu a zjistit si jak je poplatek vypočítán. Zde je využita transparentnost blockchain sítě. Kontrakt se může tímto způsobem napojit na systém mýtných brán. Palubní jednotka vyšle data směrem k mýtné bráně, buď už s adresou majitele, nebo se adresa zjistí z jiných informací vozidla. Systém vytvoří transakci a odečte se token na adresu majitele. Jakékoliv vozidlo se může propojit s aplikací pomocí mobilního internetu, nebo satelitního systému. Zpoplatnění se provede díky GPS souřadnic. Pokud GPS souřadnice vozidla rovnají zpoplatněné trase, odečte se token z adresy majitela vozidla. Nedílnou výhodou je, že tyto dva systémy by mohly používat totožnou logiku placení.



Obrázek 5.2: Návrh 1. verze

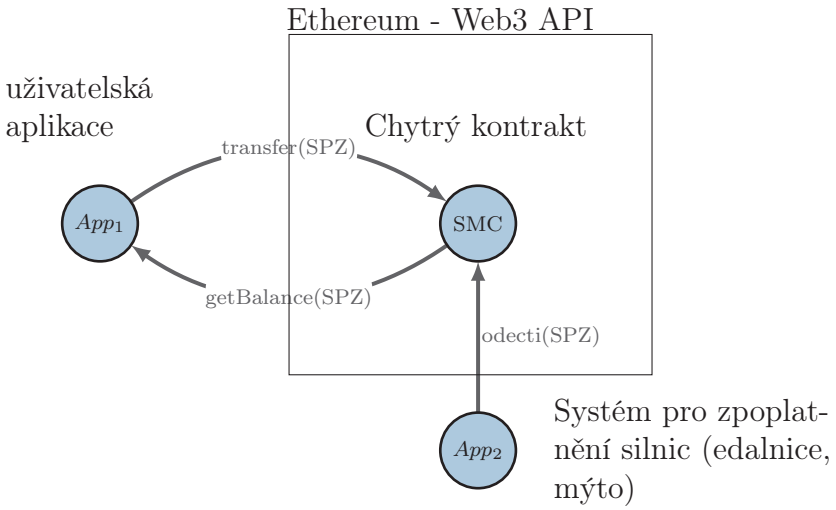
Funkčnost systému je znázorněna v obrázku 5.2. Vrchol s názvem SMC je chytrý kontrakt existující v blockchain síti ethereum. Vrchol App_1 je uživatelská aplikace s přístupem do ethereum sítě skrze Web3 API. K provedení transakce nebo zavolání funkce z kontraktu musí mít uživatel svoji adresu. Hlavní funkce kontraktu pro uživatele jsou pro koupi tokenu `transfer` a zjišťění zůstatku na adrese `getBalance`. Samotné zpoplatnění a kontrolu by prováděla aplikace App_2. K zpoplatnění by došlo až po zavolání funkce `odecti`. Vrchol App_2 je systém vlastníka silnic zpoplatněných v tomto případě Stát. Vlastník má také svoji adresu k ověření subjektu a může být i majitelem kontraktu SMC. Vlastníkem kontraktu může být i někdo třetí, ale jak už bylo řečeno není zde vkládana důvěra v kohokoliv, že dělá věci tak jak má.

Demonstrující aplikace může fungovat následovně. Aplikace se připojí k síti ethereum s adresou uživatele. To znamená, že aplikace si načte zůstatek pro adresu uživatele. Dále se připoji ke kontraktu pro zpoplatnění trasy. Tímto aplikace umožní vytvořit transakci pro koupení tokenu na adresu. Pro simulaci trasy se použijou GPS souřadnice ve formátu GPX. Jedná se o xml soubor pro ukládání tras pomocí GPS souřadnic. Tento formát podporujou všechny internetové mapy a navigace.

Hlavním argumentem, proč řešení 5.2 není vhodné, je narušení pseudoanonymity. Propojenost adresy a vozidla by znamenala částečné propojení dat o uživatelích, kdo co vlastní. Je nutné zachovat pseudoanonymitu jednotlivých uživatelů ve veřejné síti. Nikdo nechce zveřejňovat vše co vlastní. I když někdo může říčt, že nemá co skrývat. Tato míra propojení dat je na krypto směnárnách jako coinbase.com, kde potřebujete pro vytvoření účtu formu identifikace osoby. Poté osoba je spojena s adresou v blockchain síti. Od té chvíle je pak vše vůči osobě zpětně dohledatelné, protože jak je řečeno v rešeršní části - blockchain je neměnitelný spojový seznam bloků. Historie platných transakcí je v blocích navždy zaznamenána. Další komplikovanost spočívá v prodeji vozidla, kdy by muselo dojít k přepsání adresy majitele v systému (další operace navíc). V poslední řadě by došlo vytvoření příliš mnoha transakcím. Transakce pro koupi tokenu a poté spousta mikrotransakcí za ujetou trasu. Je nutné

zdůraznit, že za každou transakci se platí poplatek.

5.2 Zlepšení pseudoanonymity



Obrázek 5.3: Návrh 2. verze

V předchozí sekci je v kontraktu přiřazena adresa k zůstatku. Ke zlepšení pseudoanonymity je lepší rovnou přiřadit SPZ vozidla k zůstatku. Zbytek logiky zůstává bez změny. Tak, jak to je v systému edalnice. Tato změna zredukuje další operace kolem adres, ke které SPZ patří. Od teď každý může na jakoukoliv SPZ koupit token. Nemusí tedy znamenat, ten kdo koupí na SPZ token, že vozidlo je ve vlastnictví majitela adresy v kontraktu.

Řádek, který tuto příležitost řeší, může vypadat následovně viz kód 5.2.

```
1
2    mapping (string => uint256) private _balances;
```

Zdrojový kód 5.2: Zůstatek na SPZ

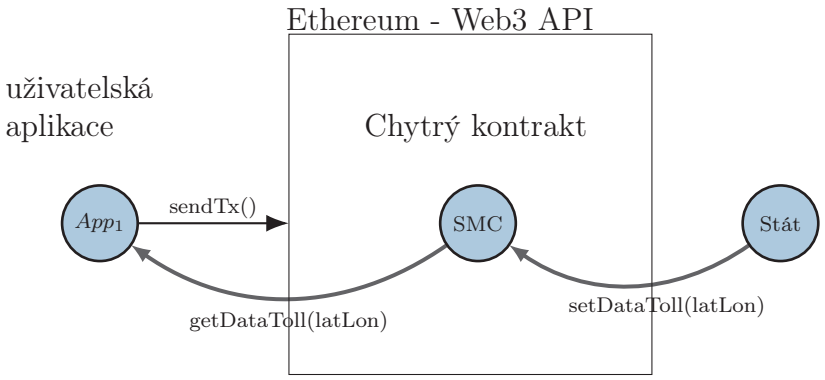
Kontrakt s zůstatkem definovaný v kódu 5.2 už nebude validní token podle standardu ERC-20. To znamená, že token nebude možné směnit nebo vyměnit za libovolně jiný ERC-20 token, ale v oblasti zpoplatnění silnic I. třídy to ničemu nevadí. Naopak nepožadujeme, aby token byl směnitelný. Ethereum měnu je stále možné směnit s jakýkoliv kontraktem. Záleží jak je definovaný konkrétní kontrakt.

5.3 Snížení počtu transakcí

Další nevýhodou minulých návrhů v sekci 5.2 a 5.1 je vytváření velké množství transakcí za převod měny a následné mikrotransakce pro odečtení poplatku za

použití zpoplatněné silnice. Teď nastává rozhodnutí jestli toto vyřešit s komplexní nadstavbou nad kontraktem, nebo se tzv. zpátky vrátit k rýsovacímu prknu. Komplexní nadstavba by představovala, že poplatek se odečte až po ukončení zpoplatněné trasy. Což by markantně snížilo počet transakcí, ale stále by docházelo ke spousta transakcím za koupi tokenu. Hlavní důvodem, proč je to důležité téma, není stále rostoucí velikost blockchain sítě, ale spíš poplatky za jednotlivé transakce. V tomto případě se musí při koupi tokenu a odečtení poplatku platit poplatek, protože dochází k zápisu do blockchain ve formě transakce. Při čtení z blockchain nedochází k vytvoření nové transakce, proto se nic platit nemusí.

Pro pochopení dalšího návrhu je nutné shrnout základní vlastnosti (výhody) kontraktu a samotné blockchain sítě. Čtení z blockchain je zadarmo. Čtení nezapisuje do blockchain a tím nevytvoří transakci, za kterou by se muselo platit. Operace nastavení a čtení v datovém typu `mapping` je v čase $O(1)$, nebo velice blízké k časové náročnosti $O(1)$. V závislosti na použité Hash funkce. Čím více instrukcí kontrakt obsahuje tím dražší transakce bude. Jinak řečeno, čím víc je kontrakt komplexnější tím více gas jednotek je spotřebováno. Oproti tomu běžná transakce při posílání měny spotřebuje pevný počet gas jednotek. Další faktor v ceně transakce je cena za 1 gas jednotku. Ta se liší v průběhu celého dne. Cena může záviset na počtu transakcí čekající na vložení do bloku.



Obrázek 5.4: Návrh 3. verze

Návrh zobrazen na obrázku 5.4 funguje tak, že stát jako autorita vlastnicí silnice vloží potřebné informace o tom jaká trasa je zpoplatněna pro uživatele pomocí metody `setDataToll`. Tyto data si uživatel stáhne `getDataToll` a pokud se nachází na trase zpoplatněné podle kontraktu, tak App_1 pošle transakci `sendTx()` na úrovni blockchain. Proto tento návrh nepotřebuje žádnou tvorbu tokenu pro uživatele. Transakce bude obsahovat cílovou adresu a vyměřený poplatek z kontraktu. Uživatel zaplatí poplatek jenom za obyčejnou transakci pro platbu mýta. V kontraktu se neukládá kdo, co a jakým vozidlem uživatel jezdí. Nedochází tedy uniku propojení dat do veřejné sítě. Vložená data v kontraktu jsou jen veřejná data, které má právo vidět každý. Jediné transakce na úrovni kontraktu budou pocházet od státu. Zde je vidět výhoda chytrého kontraktu, kdy je možno jednoduše zajistit, aby jeden kontrakt mohlo používat více států najednou. Ten bude nastavovat nebo

aktualizovat hodnotu zpoplatněné trasy.

5.4 Realizace

Z předchozích návrhů byl zvolen návrh 5.4, z důvodu pseudoanonymity, počtu transakcí a spotřebě gas jednotek. Realizace bude vytvořena v programovacím jazyce Solidity ve verzi 0.5.17. Pro prvotní nasazení se použije soukromá síť blockchain. Pro simulaci blockchain sítě stačí vytvořit 2 lokální klienty a následně je propojit. Za odeslání transakce z zpoplatněné trasy bude uživateli stačit si načíst cílovou adresu a hodnotu. Dle zadání se bude průjezd po trase simulovat pomocí GPX protokolu. Na začátek bude tedy nejjednodušší přiřadit bodům GPS souřadnice, adresu a hodnotu poplatku viz kód 5.3.

```
1   address public owner;
2   struct toll_info{
3      address addr;
4      uint256 value;
5   }
6   mapping(string => toll_info) private toll_list;
```

Zdrojový kód 5.3: Struktura dat a mapování GPS souřadnic

Struktura `toll_info` obsahuje datový typ `address` a `uint256`. Typ `uint256` má velikost 256 bitů a je `unsigned`, tudíž uložené číslo může být jenom nezáporné. Neočekává se záporná hodnota poplatku ve smyslu platba směrem ke klientovi. I když ze širšího nadhledu i tuto variantu není možné vyloučit. Do proměnné s názvem `addr` se uloží cílová adresa a do proměnné `value` se uloží hodnota poplatku. Strukturu `toll_info` použijeme v datovém typu `mapping`. Řetězec tzv. mapujeme na strukturu. Každý řětězec má inicializovanou strukturu `toll_info`. Do řetězce později se bude vkládat GPS souřadnice ve formátu `xx.xxxxxxNyy.yyyyyyE`. Zeměpisná šířka je zde reprezentována písmenem x a délka písmenem y. Zeměpisná šířka a délka bude zadávana s přesností na 6 desetinných míst. Přesnost je kolem 10 cm, což je pro účely mýtné brány dostačující. Modifikátor přístupu k proměnné je zde použit `private`. Přístup k proměnné `toll_list` bude řízen podle veřejných metod v kontraktu viz 5.4. Funkcemi je vytvořen prostor k ošetření, kdo má přístup k proměnné nebo-li jaká adresa má přístup. Tomuto se říká zapouzdření.

```
1   function setToll(address addr, string memory latLon,
       uint256 _value) public {
2      require(msg.sender == owner, "only owner function");
3      toll_list[latLon] = toll_info(addr, _value);
4
5   }
6   function setValueToll(string memory latLon, uint256
       _value) public {
```

```
7         require(msg.sender == owner, "only owner function");
8         /* Zde je mozno pridat dodatecnou kalkulaci poplatku
              */
9         toll_list[latLon].value = _value;
10    }
```

Zdrojový kód 5.4: Funkce pro nastavení mýtné brány

Funkce `setToll` má tři parametry a nic nevrací. Parametry jsou adresa, GPS souřadnice ve formátu `xx.xxxxxxNyy.yyyyyyE` a hodnota poplatku. Funkce nejdříve ověří toho, kdo zavolal tuto funkci, pomocí funkce `require`. Prvním parametrem funkce je `boolean` hodnota vloženého výrazu. Pokud výraz vyhodnotí jako `False`, výpíše se uživatelovi druhý parametr datovým typem `string`. V tomto případě se příkaz ptá jestli se jedná o majitela kontraktu. Pokud ne, oznámí se, že se jedná o metodu pro majitela kontraktu. Kontrakt tedy skončí na 2. řádku. Neplatí se žádný poplatek za volání metody.

Pokud funkce `require` vrátí `True`, kontrakt vykoná kód na řádku č. 3. Na tomto řádku se určitému GPS bodu deklaruje struktura obsahující cílovou adresu a hodnotu poplatku. Kontrakt zde skončí a vytvoří se transakce. Transakce obsahuje jaká hodnota se změní. Dokud tato transakce nebude vložena do bloku, tak k žádné změně v kontraktu nedojde.

V poslední řadě je deklarovaná funkce `setValueToll`. Funkce má za úkol změnit hodnotu poplatku v bodě definovaném v parametru metody s názvem `latLon`. Nová hodnota je vepsána v proměnné `_value` předanou opět parametrem metody. Ještě před tím než se zapíše hodnota, je opět zkontrolováno jestli jde o majitela kontraktu. Také tato funkce nic nevrací.

Pro získání informací z jednotlivých GPS bodů budou použity metody `getAddressToll` a `getValueToll` viz kód 5.5. Bohužel verze jazyka 0.5.17 neumožňuje vracet proměnnou datovým typem `struct`. Je nutné tudíž vytvořit metody pro jednotlivé proměnné nacházející se ve struktuře na místo jedné metody. Jediným parametrem obou metod je proměnná typu `string`. Parametr obsahuje GPS bod. Funkce `getAddressToll` vrátí cílovou adresu. Funkce`getValueToll` vrátí hodnotu poplatku. Za volání těchto dvou metod není vyměřen žádný poplatek. To je umožněno modifikátorem s názvem `view`. Modifikátor `view` neumožňuje příkaz, který by měnil stav kontraktu. Zde neověřujeme přístup k funkcím, protože je vyžadováný přístup pro kohokoliv.

```
1     function getAddressToll(string memory latLon) public
          view returns(address){
2         return toll_list[latLon].addr;
3     }
4     function getValueToll(string memory latLon) public view
          returns(uint256){
5         return toll_list[latLon].value;
6     }
```

Zdrojový kód 5.5: Metody get

Tímto je základ kontraktu deklarovaný. Celý kontrakt je přístupný v příloze 8.2. Kontrakt obsahuje také pomocné funkce pro přístup jednotlivých adres úložišť, kde mýtné brány jsou zapsány. Protože je to kompilovaný jazyk, je nutné ho teď zkompilovat do strojového kódu. Ke kompilaci je použit kód 5.6.

```
1 > solc --pretty-json --combined-json abi,bin,interface
    toll.sol
```

Zdrojový kód 5.6: Kompilace kontraktu

Soubor s kontraktem je nazván `toll.sol`. Výstup kompilace je ve formátu JSON. Kompilátor `solc` totiž směruje celý výstup na standardní výstup. To znamená, že pokud se příkaz spustí zobrazí se výstup v příkazové řádce. Výstup můžeme přesměrovat například do souboru, to se bude hodit až dojde k nasazení kontraktu do blockchain. Důležitými parametry kódu jsou `abi, bin, interface`. Parametr `bin` je pro výstup binárního kódu z kontraktu.

Abi je zkratka pro Application Binary Interface (Aplikační binární rozhraní). Binární rozhraní předává informaci o proměnných a metodách v binární podobě. Informace jsou název, typ a měnitelnost proměnné. U metod se ukládá název, vstupy, výstupy a typ. Typ může být funkce k placení `payable` nebo čtení `view`. Tyto informace jsou důležité v rámci připojování se ke kontraktu.

```
1 > echo "var output=`solc --pretty-json --combined-json abi
    ,bin,interface toll.sol`" > output.js
```

Zdrojový kód 5.7: Pokročilá verze kompilace kontraktu

V kódu 5.7 je výstup přesměrován pomocí operátora > do souboru `output.js`. Samotný výstup kompilátoru je zabalen do javascript proměnné s názvem `output`.

5.5 Nasazení kontraktu do blockchain

Strojový kód je nutné teď nasadit do blockchain. Kontrakt se může nasadit pomocí Web3 API, nebo za použití konzole klienta připojeného k blockchain síti. Jednodušší a pro testování přívětivější je použít klienta viz kód 5.8.

```
1 loadScript("./output.js");
2 contractAbi = output.contracts[nameFile+':'+contractName].
    abi;
3 contractEth = eth.contract(JSON.parse(contractAbi));
```

```
4 contractBinCode = '0x' + output.contracts[nameFile+':'+
      contractName].bin;
5
6 deployTransactionObject = { from: eth.coinbase, data:
      contractBinCode, gas: petrol };
7 contractInstance = contractEth.new(deployTransactionObject
      );
8 start_stop_mine();
```

Zdrojový kód 5.8: Nasazení kontraktu do blockchain

Nejdříve se načte soubor `output.js` jako výsledek ze sekce 5.4. Ze souboru je pak vytaženo binární rozhraní `contractAbi`, ze kterého je vytvořen objekt `contractEth`, a binární kód `contractBinCode`. V tuto chvíli jsou k dispozici všechny ingredience k potřebě vložení kontraktu do transakce. Ingredience k vytvoření transakce jsou atributy `from` a `gas`. Atribut `from` je majitel transakce a potažmo kontraktu. Bude to adresa, která bude nakonec vepsána také do kontraktu v proměnné `owner`. Následně je nutné přidat atribut `gas` s množstvím gas jednotek. V případě tohoto kontraktu bude stačit zadat kolem 2 milionů gas jednotek. Z provedených odhadů by mělo kontraktu k nasazení stačit kolem 1 milionu gas jednotek. Pokud se zadá počet gas jednotek rovno k odhadovaných, v nejčastějším případě poté nedojde k vytvoření transakce. Po vytvoření transakce se provede její vytěžení a zařazení do bloku funkcí `start_stop_mine` . Poznámka na závěr, pro vytvoření transakce je potřeba zpřístupnit soukromý klíč příkazem `personal.unlockAccount(address)`. Příkaz se zeptá na heslo. Tímto je umožněno podepsat transakci soukromým klíčem. Následné ověření probíhá veřejným klíčem.

5.6 Výsledek řešení

Kontrakt je v tento moment nasazený na blockchain. Teď je možné se ke kontraktu připojit pomocí kódu 5.9. Bez připojení není možné volat funkce kontraktu. První řádky kódu jsou podobné jako u nasazení kontraktu, jinými slovy je nutné získat rozhraní kontraktu(ABI) tj. jaké funkce kontrakt obsahuje. Z bínárního kódu uložený v blockchain se to nevyčte. Do proměnné `contractAddress` se ukládá adresa kontraktu vyčtena z bloku, ve kterém kontrakt je uložen jako transakce. Na posledním řádku pak už jenom se proměnná `tollContract` připojí k rozhraní, které je spojené s adresou kontraktu. Tímto připojení se ke kontraktu hotové.

```
1 loadScript('./output.js');
2 contractAbi = output.contracts[nameFile+':'+contractName].
      abi;
3 contractEth = eth.contract(JSON.parse(contractAbi));
4 contractAddress = eth.getTransactionReceipt(eth.
      getBlockByNumber(blockNumber).transactions[txNumber]).
      contractAddress;
```

```
5 console.log(contractAddress);
6 if(contractAddress == null){
7       return;
8 }
9 tollContract = contractEth.at(contractAddress);
```

Zdrojový kód 5.9: Připojení ke kontraktu

Připojení je možné otestovat zavolání některou funkcí z kontraktu viz kód 5.10. Zde se volá funkce pro nastavení mýtné brány `setToll` s parametry adresy majitela mýtný brány `ownerToll`, GPS souřadnice mýtný brány `latLon` a výše poplatku `value`. Funkce nastavuje dvě proměnné pro uložení adresy a poplatku. Nastavení hodnoty v hash tabulce typu `mapping` spotřebuje kolem 30 000 gas jednotek. To je skoro o 10 000 gas jednotek víc než za transakci na úrovni blockchain. Celková spotřeba metody v gas jednotkách bude kolem 60 000 gas jednotek. Toto číslo je možné vynásobit aktuální cenou za 1 gas jednotku, poté dostaneme výši poplatku za transakci.

```
1 tollContract.setToll(ownerToll, latLon, value).send({from:
      ownerContract});
```

Zdrojový kód 5.10: Nastavení kontraktu

Protože je to funkce měnící obsah proměnné v kontraktu, vytvoří se transakce příkazem `send`. Parametr je majitel transakce v tomto případě i majitelem kontraktu, protože nikdo jiný nemá oprávnění k zapisování. Poté je nutné, aby transakce se opět vytěžila a vložila do blockchain. Až potom jsou vložená data do funkce vepsána v proměnné kontraktu. K tomu můžeme použít funkci `start_stop_mine` ze sekce Nasazení kontraktu 5.5.

Dále kontrakt obsahuje metody pro uživatele pozemních komunikací viz kód 5.11. Metody jenom čtou obsah proměnných z kontraktu a tím nevytváří novou transakci, proto se neplatí žádný poplatek na straně uživatele. Pro návrat hodnoty je nutné ještě zavolat funkci `call`.

```
1 var fee = tollContract.getValueToll(latLon).call();
2 var address = tollContract.getAddressToll(latLon).call();
```

Zdrojový kód 5.11: Získání dat z kontraktu

Pomocí těchto dat může uživatel vytvořit transakci za využití silnice I. třídy na úrovni blockchain viz kód 5.12. Zde se mohou použít proměnné z předešlého kódu 5.11.

```
1 eth.sendTransaction({from: sender,to: address, value: fee,
      gas: 21000})
```

Zdrojový kód 5.12: Vytvoření transakce

Až po vytěžení samotné transakce je mýto zaplaceno.

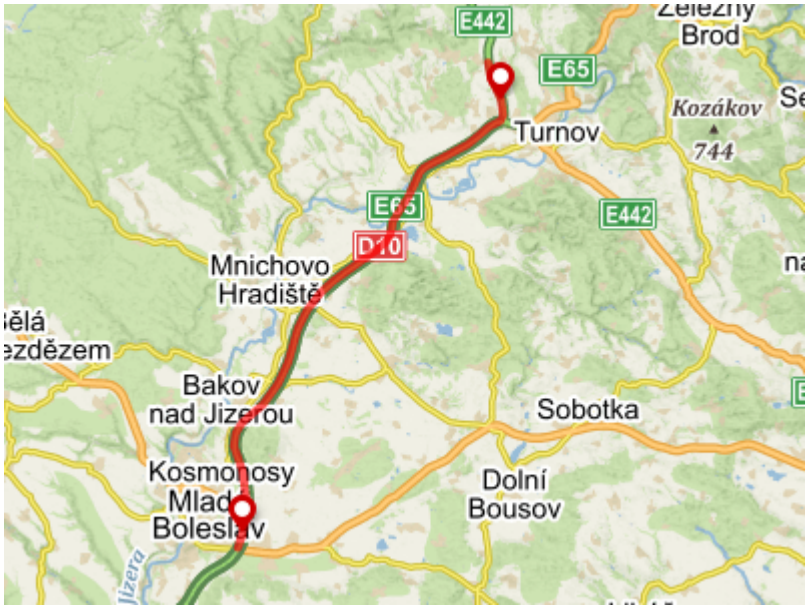
5.7 Trasa GPX

Ruční testování funkcionality kontraktu v blockchain síti je zdlouhavý proces. Ve zkratce půjde o vytvoření simulace pro zpoplatnění trasy. Kód bude simulovat jízdu po trase. K simulaci se použije trasa definovanou protokolem GPX viz kód 5.13. GPX protokol používá XML formát. XML je rozšiřitelný značkovací jazyk. V XML se používají párové a nepárové tagy, které se mohou vnořovat, a také mohou obsahovat parametry. Na začátku souboru se definují základní atributy a tagy pro kompatibilitu mezi zařízeními. Mezi nimi patří kodování `utf-8`, xml verze `1.0`, tvůrce gpx trasy `mapy.cz`, verze gpx protokolu `1.1` a další. Poté následuje párový tag `trk` (zkratka ze slova track), který už obsahuje danou trasu. Ke každé trase můžeme přiřadit název pomocí tagu `name`. Tag `trkseg` je zkratka pro track segment a obsahuje jednotlivý GPS body `trkpt` trasy. Tag `trkpt` obsahuje parametry zeměpisnou šířku `lat` a zeměpisnou délku `lon`. Tento tag poté obsahuje informaci o nadmořské výšce `ele`. Název GPS bodu je tvořen zkraceným tvarem zeměpisné šířky a délky.

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <gpx creator="mapy.cz" version="1.1" xmlns="http://www.
       topografix.com/GPX/1/1" xmlns:xsi="http://www.w3.org
       /2001/XMLSchema-instance" xsi:schemaLocation="http://
       www.topografix.com/GPX/1/1 http://www.topografix.com/
       GPX/1/1/gpx.xsd">
4      <trk>
5         <name></name>
6         <trkseg>
7            <trkpt lat="50.60606628656387" lon
                ="15.118529945611954">
8               <ele>300</ele>
9               <name>50.606066N, 15.118530E</name>
10           </trkpt>
```

Zdrojový kód 5.13: GPX protokol

Pro ukázku byl vybrán úsek dálnice D10 směrem z Liberce na Prahu viz obrázek 5.5. Délka úseku je kolem 33 kilometrů. Aktuálně se jedná o úsek zpoplatněný [7].



Obrázek 5.5: Výbraný úsek dálnice D10

Z GPX trasy se pro simulaci použijou jednotlivé souřadnice GPS bodů z tag `trkpt`. Tato práce si vystačí se zkráceným tvarem souřadnic, tak jak je definovaný tag `name`. Je nutné tedy vybrat informace z tag `name` vnořený v `trkpt` tag. Jedna z možností je použít nodejs a knihovnu `xml2js`, nebo kteroukoliv jinou knihovnu pracující s XML soubory. Mezi elegantním řešením patří také použití Bash skriptu. Za zmínku také stojí zmínit řešení pomocí Bash skriptu viz kód 5.14 [5]. Nejedná se o robustní řešení, ale pro jednoduchý XML soubor v dostatečné míře stačí.

```
1
2 rgpx () { local IFS=\> ; read -d \< E C ;}
3
4 while rgpx; do
5    if [[ $E = name ]]; then
6       echo $C
7    fi
8 done < export.gpx
```

Zdrojový kód 5.14: Výběr dat z XML souboru

Do cyklu `while` vstupuje soubor s názvem `export.gpx` pomocí operátoru `<`. Pro každý řádek souboru se zavolá funkce `rgpx`, která oddělí název tagu a data uložená v tagu do proměnných `E` a `C`. Tyto proměnné se pak využijí v těle cyklu. V příkazu `if` si porovnáme proměnnou `E` v tomto případě tag s názvem `name`. Pokud se výraz vyhodnotí jako `true`, vypíše se na standardní výstup data z proměnné `C`.

```
1
2 50.606066N, 15.118530E
```

```
3 50.606094N, 15.118511E
4 50.606459N, 15.118295E
5 ...
```

Zdrojový kód 5.15: Výstup z Bash skriptu 5.14

Výstup ze skriptu by měl vypadat podobně viz 5.15. První řádek je prázdný z důvodu nalezení `name` v tagu `trkseg`, který ukládá název daného úseku. V tomto případě je prázdný, což ničemu nevadí. Teď stačí tento výstup dát do vhodného formátu, ideálně zbavením se mezery a čárky mezi zeměpisnou šířkou a délkou. V prostředí nodejs je potřeba načíst tyto data do datového typu pole. To umožní iterativně vkládat data do kontraktu podle GPS souřadnice. Je možné tímto například si zadefinovat mýtnou bránu pro každou n-tou souřadnici. Funkce a metody pro vložení těchto dat do kontraktu jsou poté totožné ze sekce 5.6.

5.8 Simulace

Pro simulaci je nutné nejdříve definovat mýtný brány. Pro ukázku se použije trasa ze sekce 5.7, takže bude stačit definovat mýtnou bránu jenom na této trase. Nastavení mýtné brány se provede metodou s názvem `setToll`. Vstupními parametry jsou GPS souřadnice mýtné brány a hodnota poplatku.

```
1 for(var i = 0; i < latLonList.length; i++){
2       var latLon = latLonList[i][0].concat(latLonList[i
            ][1]);
3       bonytoken.setToll(ownerToll, latLon, value).send({
            from: ownerContract});
4 }
```

Zdrojový kód 5.16: Vložení dat do kontraktu

V kódu 5.16 se prochází iterativně `for` cyklem proměnná `latLon` datovým typem pole, obsahující jednotlivé GPS souřadnice trasy a ty následně se vloží do kontraktu. Ještě před vložením se upraví formát souřadnic a uloží se do proměnné `latLon`. Konkrétněji byl zvolen formát typu `50.606066N15.118530E`. Poté samotné volání kontraktu se podobá ze sekce 5.6. Opět je nutné vytvořit transakci funkcí `send` a tu poté nechat vytěžit do blockchain.

Projetí po zpoplatněném úseku bude simulovat funkce. Opět se bude procházet `for` cyklem trasa uložena v poli, tak jako v případě vložení dat do kontraktu 5.16. Jen s tím rozdílem, že uživatel si stáhne výše poplatku a cílovou adresu z konkrétních souřadnic a zkontroluje jestli mýtná brána je přítomna. To se dá zjistit buď, že hodnota poplatku je 0 nebo uložená adresa se rovná `0x0`. Adresa `0x0` je speciální typ adresy, která se používá ve speciálních případech. Používá se například ke

spálování(burning) ERC-20 tokenů v kontraktech. Pokud mýtná brána existuje, uživatel pošle obyčejnou transakci danými údaji pro zaplacení mýta. Přesnou definici funkce je možné najít v příloze viz 8.1.

Kód 5.17 ukazuje výstup funkce, která simuluje jízdu po zpoplatněné trase. Do funkce vstupujou parametry trasa `latLonList` stejného typu jako v předchozích krocích a adresa plátce mýta. Každý řádek na výstupu funkce značí detekci mýtné brány. Následně se vypíše její adresa, poplatek v jednotkách `Gwei` a souřadnice.

```
> payTestContract(latLonList, address)

Address: 0x9cfD...bd237, Fee: 0.002 Gwei, LatLon:
   50.606066N15.118530E
Address: 0x9cfD...bd237, Fee: 0.002 Gwei, LatLon:
   50.615558N15.111280E
```

Zdrojový kód 5.17: Ukázka zpoplatnění

Po úspěšném vytěžení všech transakcí je možno konstatovat, že mýto bylo úspěšně zaplaceno. Transakce jsou potom zapsány v blockchain. Transakce můžou být vypsány pomocí kód 5.18. Do parametru funkce se vkládá číslo bloku `blockNumber`, který obsahuje dané transakce ze simulační trasy. Výstup funkce je pole adres odkazující se na jednotlivé transakce.

```
> eth.getBlockByNumber(blockNumber).transactions

["0xd943....42de", "0x12a5....62ad", "0x12d8....59ec
   ",....]
```

Zdrojový kód 5.18: List transakcí

Na konec zpětná kontrola výpisem obsahu jedné z transakcí viz kód 5.19. Pro získání obsahu transakce se používá funkce `eth.getTransaction` s parametrem adresy transakce.

```
> eth.getTransaction(eth.getBlockByNumber(blockNumber).
   transactions[0])

{
 blockHash: "0xbb67...a44ff",
 blockNumber: 124,
 from: "0xefed4....bc9a2",
 gas: 21000,
 gasPrice: 1000000000,
 hash: "0xd943....42de",
 nonce: 79,
 to: "0x9cfd....bd237",
```

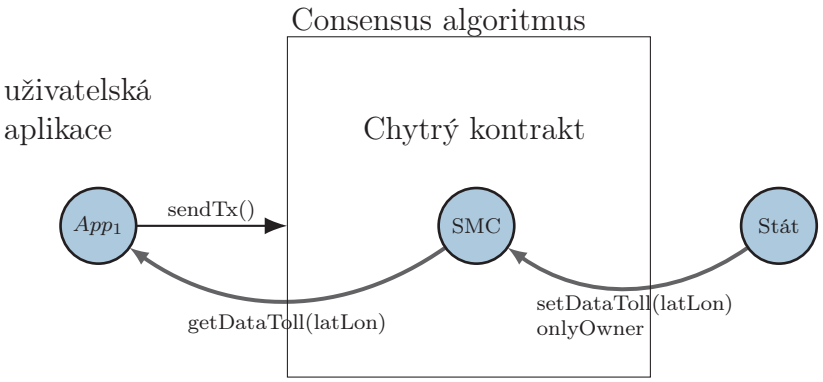
```
12   v: "0x9c7",
13   value: 2000000
14 }
```

Zdrojový kód 5.19: Obsah transakce

6 Bezpečnost

Bezpečnost kontraktu se dělí na blockchain síť a samotný chytrý kontrakt. Je nutné, aby obě oblasti byli zabezpečeny. Není možné spustit finální verzi kontraktu v nezabezpečené blockchain síti. Pokud není dostatečné míře zabezpečen chytrý kontrakt, je nesmysl se spoléhat na zabezpečení blockchain sítě. Tyto dvě úrovně musejí být zabezpečeny nezávisle na sobě. Základ zabepečení blockchain sítě se skrývá Consensus algoritmu.

V obrázku 6.1 je vidět princip zabezpečení aplikace. Consensus algoritmus se stárá o zabezpečení sítě na úrovni blockchain. Každý consensus algoritmus má své zranitelnosti. Na úrovni kontraktu poté může být zranitelná jedině funkce `setDataToll`. Ta je zabezpečena ověřením majitela kontraktu. Jinými slovy, jenom majitel má možnost zapisovat do kontraktu.



Obrázek 6.1: Bezpečnost aplikace

6.1 Chytrý kontrakt

Jednou z hrozeb může být zkopírování vepsané adresy v proměnné `owner`. Toto může každý, jak bylo znázorněno v sekci 3.3. Uživatel se identifikuje jako majitel a tím mít šanci změnit data. Uživatel tedy zavolá funkci pro nastavení dat s adresou majitele. Tímto, ale musí i vytvořit transakci pod adresou majitele. Pokud uživatel nemá soukromý klíč k podepsaní transakce, bude tato transakce v kategorii nevyřízených. Pokud se transakce podepíše jiným klíčem, uzel vytvářící bloky tuto transakci vyhodnotí jako chybnou a nezařadí ji do bloku. Pokud by se to uživatelovi povedlo, přepsaly by data v kontraktu a poplatek za transakci by zaplatil majitel ne

uživatel, který transakci vytvořil. To by se mohlo povést v případě, pokud uživatel našel např. hrubou silou soukromý klíč k adrese v proměnné `owner`. Takový přístup vyžaduje vyzkoušet velké množství kombinací.

6.2 Blockchain

Nejdříve je dobré říct o co se takový útočník snaží na úrovni blockchain. Je nemožné v blockchain vygenerovat nějakou měnu, protože logika měny je zapsána v klientovi. Pokud by si někdo upravil klienta do takové podoby, kdy si své volně vygeneruje nějakou tu měnu, nemohl by se s takovým klientem připojit do sítě. Důvod je nekompatibilnost klienta vůči klientům v síti.

Nejčastějším útokem u sítí s důkazem práce je získání 51% hashovacího výkonu. Vyjádřeno jinak, útočník generuje náhodná čísla rychleji než všichni účástníci blockchain dohromady. Pokud překročí hranici 51 % dokáže rozhodovat jaké transakce budou zahrnuty do učetní knihy. Nebo-li 51 % stačí, aby útočník přesvědčil dostatek počet ostatních uzlů sítě, že jde o věrohodný blok, a že i u sebe si ho mají zapsat a na něm dál budovat další blok. Je prakticky nemožné, aby se toto stalo u bitcoin. Pro dosažení útoku, potřebujete více výkonů, než vyprodukují některé státy dohromady. I když pokud se domluví většina učastníků sítě můžou tento útok použít jako stávku, kdy dají najevo, že s něčím nesouhlasí. Útočníkovi jde nejčastěji o přepsání konkrétní transakce. Útočník se může pokusit vymazání transakce, nebo přepsání cílové adresy transakce. Klíčem k většímu zabezpečení sítě je fakt, že takový útok by se ekonomicky nevyplatil útočníkovi. Více by se mu vyplatilo zůstat upřímným uzlem a přidávat bloky do blockchain jako všichni ostatní.

7 Závěr

Na začátku práce byly vysvětleny základní vlastnosti blockchain databáze. Mezi hlavními patří decentralizovanost, neměnitelnost, transparentnost a absence důvěry. Rozebralo se žhavé téma škálovetelnost. Bez dobré škálovatelnosti se cena transakci výrazně nesníží. Byla představena transparentnost blockchain pomocí jednoho příkazu. Příkaz umožňuje si vypsat jakýkoliv data z kontraktu. Bezpečnost a neměnitelnost blockchain je vysvětlena consensus algoritmem. Bezpečnost blockchain sítě zavisí na počtu uzlů a celkové velikosti sítě.

Výsledná aplikace dokáže na základě trasy definované GPX protokolem zpoplatnit jízdu po trase. Autorita vlastnící silnice může vytvářet mýtné brány na jednotlivých GPS souřadnicích. Je možné následně mýtný brány upravovat nebo i vymazat. Uživatel zavolá metodu kontraktu pro získání dat s parametrem GPS souřadnicemi, poté blockchain vratí odpověď. Na základě získaných dat uživatel pošle transakci. Aplikace se také může využít k placení parkování. Uživatel zaparkuje na parkovací místo a dotáže se s GPS souřadnicemi a časovým intervalem kontraktu. Poté pošle transakci pro zaplacení parkování.

Největší vyzvou práce bylo samotné pochopení blockchain technologie a následné nalezení silných stránek chytrého kontraktu a ty pak přenést do aplikace.

Samotnou kapitolou samo o sobě bylo také vytvoření vlastního prostředí pro vývoj kontraktu. Jednalo se o vytvoření lokální blockchain sítě, spuštění a její zastavení. Následně zkompilovat a nasadit kontrakt do blockchain, připojit se ke kontraktu a nakonec otestovat. Také bylo nutné si vytvořit pomocné funkce k lepšímu zobrazení dat. To zapříčinilo méně času na vytvoření komplexnějšího řešení mýta. Naopak vlastní prostředí poté urychlilo samotný vývoj při řešení a testování chyb v kontraktu. Není možné říct, že jít cestou vlastního prostředí byl špatný krok. Znalosti co tato cesta dala byly k nezaplacení v průběhu testování. Po získaných znalostech je možné se poohlížet po nějakém už existujícím nástroji pro zjednodušení a zrychlení vývoje. Všechny zdrojové kódy související s prací jsou dostupné na odkazu `https://gitlab.tul.cz/michal.kukla/bp-blockchain`. Na dotaz mohou být zpřístupněny `michal.kukla@tul.cz`.

Představené řešení by se dalo nazvat za nadčasové v době psaní této práce. I když ostatní technologie by mohly k tomuto řešení pomoct. Mezi ně patří síť 5G. Síť 5G slibuje datovou rychlost až 10 Gps a podporu různých druhů zařízení. Mezi novinky také patří implementace aplikační vrstvy s názvem Vehicle-to-everything(V2X). Pokud se vyřeší problém škálovatelnost a stane se blockchain jako jedno z možných platebních systémů pro ostatní služby, nedalo by se toto řešení vyloučit jako jedno z možných způsobů platby za mýto.

8 Příloha

```
1  if ( bonytoken === 'undefined' ){
2        console.log("No defined contract for variable
            bonytoken");
3        return;
4  //  loadContract(blockNumber);
5  }
6  sender = accounts[0];
7  amountGas = 21000;
8  for(var i =0; i < latLonList.length; i++){
9        var latLon = latLonList[i][0].concat(latLonList[i
            ][1]);
10       var recipient = await bonytoken.getAddressToll(
            latLon).call();
11       if( recipient != "0
            x0000000000000000000000000000000000000000" ){
12          var fee = await bonytoken.getValueToll(latLon)
               .call();
13          console.log("Address: "+recipient+", Fee: "+
               weitogwei(fee)+" Gwei, LatLon: "+latLon);
14          eth.sendTransaction({from: sender,to:
               recipient, value: fee, gas: amountGas})
15       //   await timer(2000);
16       }
17 }
```

Zdrojový kód 8.1: Definice funkce pro simulaci jízdy po trase

```
1  // SPDX-License-Identifier: GPL-3.0-or-later
2
3  pragma solidity ^0.5.17;
4
5  contract SimpleStorage {
6     address public owner;
7  // mapping(string)
8     mapping(address => uint256) public a;
9
```

```
10 // [lat][lon] = address
11    struct toll_info{
12 // address to pay fee
13       address addr;
14 // fee in wei
15       uint256 value;
16    }
17    mapping(string => toll_info) public toll_list;
18
19    constructor() public{
20       owner = msg.sender;
21    }
22
23    function setToll(address addr, string memory latLon,
         uint256 _value) public {
24       require(msg.sender == owner, "only owner function");
25       toll_list[latLon] = toll_info(addr, _value);
26
27    }
28    function setValueToll(string memory latLon, uint256
         _value) public {
29       require(msg.sender == owner, "only owner function");
30       //Zde je mozno pridat dodatecnou kalkulaci value
31       toll_list[latLon].value = _value;
32    }
33    function getAddressToll(string memory latLon) public
         view returns(address){
34       return toll_list[latLon].addr;
35    }
36    function getValueToll(string memory latLon) public view
          returns(uint256){
37       return toll_list[latLon].value;
38    }
39    function setMapping(address recipient, uint256 balance)
          public payable{
40       a[recipient] = balance;
41    }
42    function getA(address addr) public view returns(uint256
         ) {
43       return a[addr];
44    }
45    function getAbiEncode(uint256 key, uint256 slot) public
          pure returns(bytes memory){
46       return abi.encodePacked(key,slot);
47    }
48    function getKeccak(uint256 key, uint256 slot) public
         pure returns (uint256) {
49       return uint256(keccak256(abi.encode(key, slot)));
```

```
50 // return uint256(keccak256(addr))
51     }
52 }
```

Zdrojový kód 8.2: Kontrakt mýtných brán

Použitá literatura

[1] *Covid-19: Vláda vyhlásila na 3 týdny přísný lockdown*. 2021. URL: https://www.kurzy.cz/zpravy/581304-covid-19-vlada-vyhlasila-na-3-tydny-prisny-lockdown/.

[2] W. Dai. "b-money". In: (1998). URL: http://www.weidai.com/bmoney.txt.

[3] Jeff Garzik. *[PATCH] increase block size limit*. 2010. URL: https://bitcointalk.org/index.php?topic=1347.

[4] Joseph Gibson. *Ten Years Ago, Two Pizzas Were Purchased For 10,000 Bitcoins. Today, That's The Equivalent Of Over $400 Million*. Jan. 17, 2017. URL: https://web.archive.org/web/20210126081415/https://www.celebritynetworth.com/articles/how-much-does/ten-years-ago-two-pizzas-were-purchased-for-10000-bitcoins-today-thats-the-equivalent-of-over-400-million/.

[5] *How To Parse XML in Bash*. 2010. URL: https://stackoverflow.com/questions/893585/how-to-parse-xml-in-bash.

[6] *Inflace je nejvyšší za téměř 30 let. Zdražilo hlavně bydlení, benzin a potraviny*. 2022. URL: https://ct24.ceskatelevize.cz/ekonomika/3486618-inflace-je-nejvyssi-za-temer-30-let-zdrazilo-hlavne-bydleni-benzin-a-potraviny.

[7] Státní fond dopravní infrastruktury. *Kde platí elektronická dálniční známka*. 2022. URL: https://edalnice.cz/kde-plati/index.html.

[8] Alexander Lipton. *Blockchains and distributed ledgers in retrospective and perspective*. 2017. URL: https://static1.squarespace.com/static/57af6f83893fc027c794e637/t/5ac3ce0c575d1fa836835376/1522781718170/88+2018+Lipton+Blockchains+and+distributed+ledgers+in+retrospective+and+perspective+JRF.pdf.

[9] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. 2008. URL: www.bitcoin.org.

[10] Jamie Redman. "Crypto Patent Alliance Questions Craig Wright's White Paper Copyright Claim". In: (2021). URL: https://news.bitcoin.com/crypto-patent-alliance-questions-craig-wrights-white-paper-copyright-claim/.

[11] Nick Szabo. *Smart Contracts: Building Blocks for Digital Markets*. Tech. rep. 1996.

[12] *Whitepaper definice*. 2022. URL: https://en.wikipedia.org/wiki/White_paper.

[13] Wikipedia, ed. *Everything is a file*. Sept. 28, 2021. URL: https://en.wikipedia.org/wiki/Everything_is_a_file.

[14] Karl Wüst and Arthur Gervais. "Do you need a blockchain". In: (2017). URL: doyouneedablockchain.com.