Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Vývoj darovací platformy využívající Lightning Network

Diplomová práce

Autor: Bc. David Nekolný
Studijní obor: Aplikovaná informatika

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové duben 2022

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29.4.2022 Bc. David Nekolný

Poděkování:

Tímto bych chtěl poděkovat vedoucí diplomové práce Mgr. Daniele Ponce, Ph.D. za metodické vedení práce.

Anotace

Diplomová práce se věnuje návrhu a vývoji darovací platformy za využití platební sítě Lightning Network. Účelem této platformy je umožnit internetovým tvůrcům jednoduché přijímání darů od svých podporovatelů v podobě bitcoinů. V teoretické části práce je popsána platební síť Lightning network a s ní úzce související technologie Bitcoin. Pozornost v této práci je věnována především těmto dvěma technologiím. Pouze okrajově se práce zabývá ostatními známějšími technologiemi, které jsou také součástí návrhu. V práci je implementována funkční ukázka navrhované darovací platformy, která se skládá z webové aplikace, rozšíření pro webový prohlížeč a Lightning network uzlu pro provádění plateb. Webová aplikace byla vyvinuta pomocí frameworků Node.js a Vuew.js a programovacího jazyka JavaScript.

Klíčová slova: Lightning network, Bitcoin, Blockchain, Webová aplikace, API

Annotation

Title: Development of a donation platform using the Lightning network

The diploma thesis focuses on the design and development of a donation platform using the Lightning Network for payment. The purpose of this platform is to enable Internet creators to easily accept donations from their supporters in the form of bitcoins. The theoretical part of the thesis describes the Lightning network and the closely related bitcoin technology. Across the thesis, the focus is primarily on these two technologies. Only peripherally, the thesis then discusses other more familiar technologies that are also used in the proposal. The thesis also includes the implementation of a functional demonstration of the proposed donation platform, which consists of a web application, a web browser extension and a Lightning network node for making payments. The web application was developed using the Node.js and Vue.js frameworks and the JavaScript programming language.

Keywords: Lightning network, Bitcoin, Blockchain, Web application, API

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Seznam zkratek

API	Application Programming Interface
	Rozhraní pro programování aplikací
BIP	Bitcoin Improvement Proposals
	Návrh na vylepšení Bitcoinu
BTC	Zkratka měny bitcoin
CLTV	Check LockTime Verify
CRUD	Create, Read, Update, Delete
	Vytvořit, Číst, Editovat, Smazat
CSV	Check Sequence Verify
DOS	Denial of Service
	Odepření služby
ECDSA	Elliptic Curve Digital Signature Algorithm
	Protokol digitálního podpisu s využitím eliptických křivek
gRPC	Google Remote Procedure Call
HTLC	Hashed TimeLock Contract
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token
LN	Lightning network
MTP	Median Time Past
PoW	Proof of Work
REST	Representational State Transfer
	Architektura rozhraní API
SegWit	Segregate Witness
TXID	Transaction ID
	ID transakce
UTXO	Unspent transaction output
	Neutracený transakční výstup
WTXID	Witness Transaction ID
	ID transakce se svědkem

1 Úvod

Virtuální měna Bitcoin je v současné době velmi diskutovaným technologickým tématem, které si postupně získává čím dál více příznivců. Jedná se o ojedinělou formu elektronických peněz, které je možné posílat takzvaně peer-to-peer neboli napřímo mezi účastníky. Bitcoin je decentralizovaná technologie, nad kterou nemá kontrolu žádná centrální autorita, a to ani její tvůrce Satoshi Nakamoto. Jelikož se jedná o elektronickou formu peněz, je možné tyto peníze používat prakticky všude po světě, kde je přístup k internetu. Jednou z hlavních výhod Bitcoinu je jeho vysoké zabezpečení, které však vyžaduje spotřebu velkého množství elektrické energie. U Bitcoinu platí, že čím více je energie spotřebováno, tím více je zabezpečený.

Přes řadu výhod, které Bitcoin přináší, však není příliš vhodný pro platby s nízkou hodnotou a také okamžité platby. To je zapříčiněno tím, že jsou Bitcoinové transakce schvalovány průměrně jednou za deset minut a zároveň je omezeno množství schvalovaných transakcí. Potvrzení transakce může být v době vytížení Bitcoinové sítě velice nákladné a také zdlouhavé. Tyto omezení jsou pro některé typy plateb nepřijatelné. Proto byla za účelem provádění instantních plateb s malou hodnotou vynalezena takzvaná „druhá vrstva“ Bitcoinu s názvem Lightning Network. Jedná se o platební síť, přes kterou lze posílat bitcoiny téměř okamžitě a za velice malé poplatky. Zároveň není snížena bezpečnost plateb.

Hlavním cílem této diplomové práce je vyvinutí darovací platformy, která umožní internetovým tvůrcům přijímat příspěvky v podobě bitcoinů. Za tímto účelem bude platforma integrovat právě platební síť Lightning network. Ta by měla být k posílání malých příspěvků díky svým vlastnostem vhodná.

Motivací k vytvoření darovací platformy je viditelné rozšiřování zpoplatněného obsahu napříč internetem. Tato platforma si tedy klade za cíl umožnit tvůrcům obsahu na internetu jednoduché přijímání příspěvků od svých příznivců. Tím by mohli být tvůrci motivováni k bezplatnému zpřístupnění svého obsahu všem a také

k vytváření kvalitnějšího obsahu. Přičemž příspěvky by mohly sloužit jako pozitivní zpětná vazba.

Teoretická část práce se zabývá právě technologiemi Bitcoin a Lightning Network s důrazem na jejich technické parametry a principy. První kapitola teoretické části práce se zaměřuje na fungování virtuální měny Bitcoin. Kapitola popisuje základní principy a datové struktury Bitcoinových transakcí spolu s jejich vlastnostmi. Dále se kapitola věnuje decentralizované databázi blockchain a jejímu zabezpečení proti neoprávněným úpravám. Další část teoretické práce analyzuje fungování platební sítě Lightning network. Součástí analýzy je mimo jiné popsání procesu provádění plateb přes tuto platební síť.

V rámci praktické části práce je následně navrhnuto řešení pro darovací platformu. Nejprve jsou definovány požadavky na výsledný systém na základě kterých je proveden samotný návrh. Součástí návrhu je datový model spolu s architekturou definující propojení integrovaných částí systému. Poslední část praktické práce se věnuje implementaci ukázkového řešení navrhované darovací platformy.

2 Cíl práce, metodika

Hlavním cílem této práce je vyvinout webovou darovací platformu s využitím technologie Lightning network. Hlavním účelem platformy je zpřístupnit možnost přijímání příspěvků ve virtuální měně bitcoin pro tvůrce obsahu na internetu. Tvůrcem může být autor článku, videa, přispěvatel na fórum nebo kdokoliv jiný, kdo tvoří nějaký obsah na internetu. Výsledná platforma by měla být jednoduchá a přívětivá jak pro tvůrce, který tak může svou práci zhodnotit, tak pro dárce, který chce podpořit tvůrce. Z pohledu dárce by proto měla být možnost přispění integrována přímo do platforem, které daný obsah prezentují. Vzhledem k tomu, že platforma pracuje s určitou formou peněz, měl by být kladen velký důraz na její zabezpečení.

K přesunu bitcoinů mezi účastníky platformy bude využívána platební síť Lightning network. Ta bude použita především z důvodu podpory drobných plateb s nízkými poplatky na provedení. Platební síť Lightning network úzce souvisí s technologií Bitcoin. Dílčím cílem práce je pak provedení technické analýzy technologie Bitcoin a Lightning network.

Prvním krokem při zpracování této diplomové práce bude provedení analýzy použitých technologií. Dojde k prozkoumání vlastností a principů dvou technologií Bitcoin a Lightning network. Vzhledem k tomu, že se jedná o poměrně komunitní sítě, informace budou čerpány především z open-source, otevřených zdrojů. Následuje stanovení požadavků na výsledný systém. Výsledné znalosti z technické analýzy Bitcoinové a Lightning network sítě spolu se stanovenými požadavky budou uplatněny v návrhu platformy. Následně bude na základě návrhu implementováno ukázkové funkční řešení. Posledním krokem práce bude shrnutí dosažených výsledků a popis fungování darovací platformy.

3 Bitcoin a Blockchain

Bitcoin je digitální platební síť umožňující přenos hodnoty napřímo mezi účastníky takzvaně peer-to-peer, tedy bez nutnosti jakéhokoliv prostředníka. Uživatelé spolu komunikují na základě Bitcoinového protokolu výhradně přes internet, ačkoliv je možné použít i jiné informační sítě. Bitcoin se skládá ze sady konceptů a technologií, které dohromady vytvářejí ekosystém digitálních peněz. Základní koncept byl představen v roce 2008 neznámým tvůrcem vystupujícím pod pseudonymem Satoshi Nakamoto.

Informace o všech Bitcoinových transakcích, které kdy proběhly, jsou udržovány v decentralizované databázi s názvem Blockchain, která se také někdy označuje jako účetní kniha Bitcoinu. Data v této databázi jsou strukturována do takzvaných bloků, které jsou z největší části tvořeny z transakčních dat. Při vkládání nového bloku do databáze je daný blok svázán s předchozím a vzniká tak pomyslný „řetěz“ bloků, proto také název blockchain. Z bezpečnostního důvodu je k uložení nového bloku na blockchain potřeba vypočíst složitou matematickou úlohu. Tento proces je znám jako těžení, a počítače, které ho vykonávají, se nazývají těžaři. Při těžení nových bitcoinů je spotřebováváno velké množství energie, a proto jsou těžaři za každý nový blok odměňováni.

Základní měnovou jednotkou v Bitcoinové sítí je bitcoin. Pro odlišení je jednotka bitcoin psána s malým písmenem na začátku. V kontextu celé sítě se pak používá slovo Bitcoin začínající velkým písmenem. Nejmenší jednotkou je satoshi se zkratkou sats, který nese název po Satoshi Nakamotovi, a představuje jednu sto-miliontinu bitcoinu. Bitcoiny nikdy neexistují samy o sobě, ale jsou vždy součástí transakcí, které přenášejí hodnotu od odesílatele k příjemci. Uživatelé Bitcoinové sítě vlastní kryptografické klíče, které slouží jako důkaz o vlastnictví bitcoinů v transakci. Při vytvoření transakce se bitcoiny uzamknou na klíč příjemce a k následnému utracení je nutné doložit důkaz o vlastnictví daného klíče pomocí digitálního podpisu. [4]

K jednodušší správě kryptografických klíčů postupně vznikla řada softwarů, kterým se říká peněženky, i když se spíše jedná o klíčenky. Peněženky totiž neuchovávají žádné bitcoiny, ale pouze klíče k bitcoinům, které jsou uloženy na blockchainu. Peněženky neslouží pouze k uchovávání klíčů, ale také generují nové. Vzhledem k obrovskému množství možných klíčů je zvykem generovat nový klíč pro každou novou transakci. Bitcoinové peněženky lze rozdělit na softwarové, internetové služby a hardwarové, které se řadí mezi nejbezpečnější variantu pro správu klíčů.

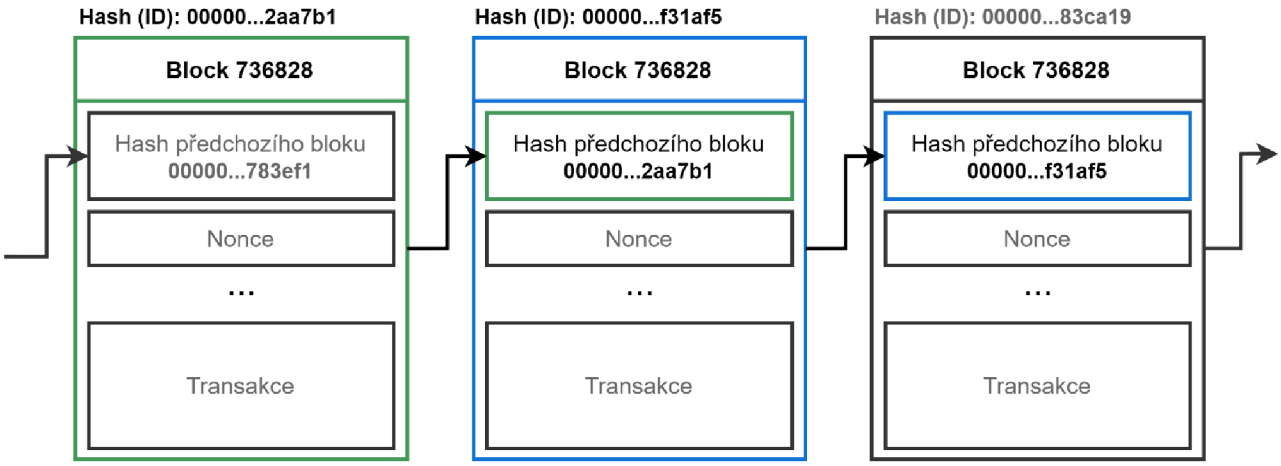
3.1 Datová struktura

Datová struktura blockchainu je ve své podstatě lineární spojový seznam, který je jednosměrný a bez možnosti cyklení záznamů. Jednotlivé záznamy v blockchainu se nazývají bloky, které jsou zpětně propojovány. Každý nově zapsaný blok na blockchain obsahuje odkaz na předchozí blok v blockchainu a tím kontinuálně vzniká pomyslný řetěz bloků. Hlavním účelem bloků je uchovávání informací o uskutečněných transakcích v Bitcoinové síti, proto jsou jednotlivé bloky tvořeny převážně z transakčních dat.

Každý blok v blockchainu má v podstatě dva identifikátory. Jedním z nich je výška bloku neboli index a druhým jeho otisk neboli hash. Výška je odvozena od pořadí bloku v blockchainu, kdy každý nový blok zapsaný do blockchainu má výšku o jedna větší než předchozí. První blok zapsaný na blockchainu měl výšku 0 a každý následující vždy o jedna více. Pro představu, první blok uložený na blockchainu v roce 2022 měl výšku 716599. Jinými slovy, blockchain v té době obsahoval celkem 716600 bloků.

K propojování bloků v blockchainu je používán druhý zmíněný identifikátor, a to hash bloku. Hash je digitální otisk bloku a je vypočítán pomocí hašovací funkce SHA-256. Hašovací funkce se používají napříč celou architekturou Bitcoinu. Proces použití hašovací funkce se nazývá hašování a jako výstup vznikne otisk neboli hash. Z výsledného hashe je prakticky nemožné získat původní data. Každý blok

v blockchainu obsahuje hash předchozího bloku a tím vzniká řetězec bloků, který je znázorněn na Obrázek 1.



Obrázek 1: Řetězení bloků v blockchainu [autor]

Díky této struktuře je při upravení jednoho bloku nutné upravit i všechny následující. Protože jakoukoliv změnou dat v bloku se změní také jeho hash, a ten je potřeba aktualizovat i následujícím bloku. Tím se však změní hash i následujícího bloku, a to se opakuje až do posledního bloku v řetězci. Tento kaskádový efekt zajišťuje, že při změně jakéhokoliv bloku v blockchainu je nutné přepočítat hashe všech následujících bloků. Čím je blok starší, a tudíž hlouběji v blockchainu, tím více bloků se při jeho změně musí přepočítat. V Bitcoinovém blockchainu je záměrně zvýšená obtížnost výpočtu hashe bloku a k jeho vypočtení je potřeba obrovský výpočetní výkon. Upravování bloků, které mají více než menší jednotky navazujících bloků, je tak prakticky nemožné. Obecně lze říct, že čím hlouběji v blockchainu je blok uložen, tím je těžší ho upravit. [4]

3.1.1 Blok

Blok je základní datová struktura v blockchainu agregující množinu bitcoinových transakcí. Skládá se z hlavičky, která obsahuje metadata o daném bloku a samotných transakcí. Velikost bloku je kvůli snížení datové náročnosti omezena na 1 MB, kde

pouze 80 bajtů tvoří hlavička a zbytek transakční data. Tabulka 1 popisuje datovou strukturu bloku.

Tabulka 1: Datová struktura bloku (převzato z [4])

Vlastnost	Velikost	Popis
Velikost	4 B	Velikost bloku v bajtech.
Hlavička	80 B	Hlavička obsahující metadata bloku.
Počet transakcí	1-9 B (VarInt)	Počet transakcí v bloku.
Transakce	Proměnná	Množina transakcí.

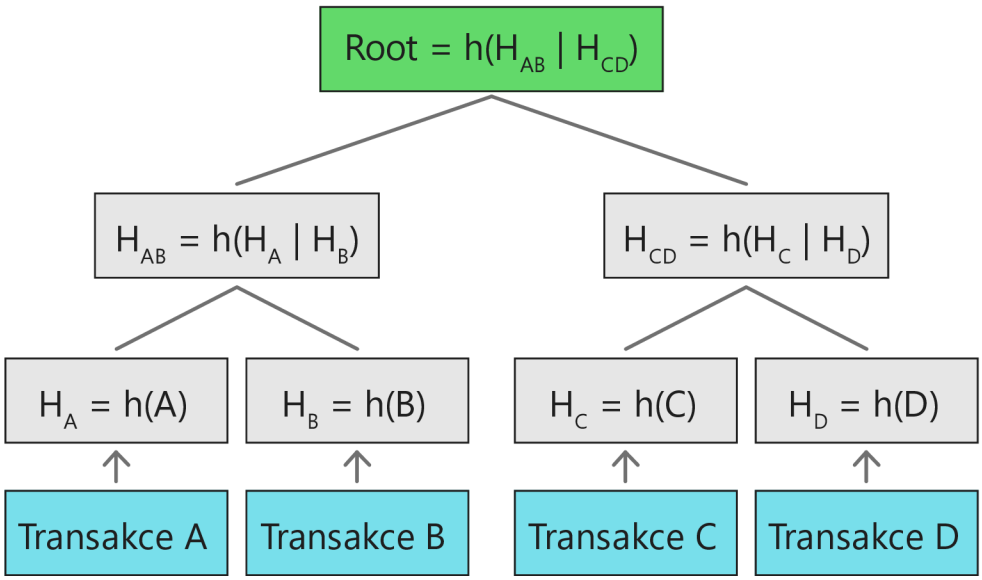
V Tabulka 2 je popsána datová struktura samotné hlavičky bloku, která sestává z důležitých a unikátních metadat o celém bloku. Hlavička je uložena na začátku bloku hned po velikosti. V první řadě hlavička obsahuje verzi, jejíž hodnota je s postupným vylepšováním Bitcoinových protokolů zvyšována tak, aby bylo jednoznačné, podle jakých pravidel je daný blok konstruován a jak má být validován. Hash předchozího bloku pak odkazuje na předchozí blok v blockchainu.

Tabulka 2: Datová struktura hlavičky bloku (převzato z [8])

Vlastnost	Velikost	Popis
Verze	4 B	Číslo verze bloku určuje, jakou sadou pravidel dodržovat při validaci bloku.
Hash předchozího bloku	32 B	Hash hlavičky předchozího bloku.
merkle root	32 B	Kořen hašového stromu obsahující všechny transakce tohoto bloku.
timestamp	4 B	Čas, kdy těžař začal hašovat hlavičku.
nBits	4 B	Cíl obtížnosti algoritmu Proof of Work pro tento blok.

Nonce	4 B	Libovolné číslo sloužící pouze k alternování výsledného hashe bloku.

Hlavička dále obsahuje vlastnost *merkle root*, která představuje otisk všech transakcí v daném bloku. Jedná se o kořen stromové struktury, která se nazývá hašový strom nebo také merklův strom. Listy hašového stromu představují jednotlivé transakce a větve jsou tvořeny postupným vzájemným hašováním těchto transakcí. Na Obrázek 2 je znázorněn postup formování hašového stromu z transakcí, kde funkce *h* představuje dvojí hašovací funkci SHA256, což lze zapsat jako SHA256(SHA256(data)). Tímto způsobem jsou zahašovány všechny transakce v bloku do jediné hodnoty.



Obrázek 2: Hašový (merklův) strom [autor]

Vlastnost *merkle root* má hodnotu kořene hašového stromu tvořeného ze všech transakcí daného bloku. Jedná se tedy o jednoznačný otisk těchto transakcí. Proto při jakékoliv změně libovolné transakce v bloku je změněna i hodnota *merkle root*. Vzhledem k tomu není nutné při hašování bloku hašovat celý blok i se všemi transakcemi. Stačí hašovat pouze hlavičku bloku, která již obsahuje otisk transakcí. K tomu se také používá dvojí hašování funkcí SHA256. Vlastnost hash předchozího

bloku tedy neobsahuje hash celého bloku, ale pouze jeho hlavičky. Zajímavé je, že hash samotného bloku není uložen v daném bloku, ale pouze v následujícím.

Poslední tři vlastnosti hlavičky *timestamp*, *nBits* a *nonce* se vztahují k těžbě a jsou detailněji popsány v kapitole *Těžba*.

3.2 Proof of Work

Satoshi Nakamoto [19] tvrdí, že „jediným způsobem, jak potvrdit, že nějaká transakce neproběhla, je o veškerých transakcích vědět“[1] (překlad autora). Právě proto existuje databáze blockchain, jejíž kopie jsou distribuovány po celé Bitcoinové síti tak, aby mohl kdokoliv a kdykoliv vědět o všech platných transakcích. Obecně je u decentralizovaných databází problém najít shodu o aktuálním stavu databáze napříč celou sítí. Proto byl v Bitcoinu zaveden takzvaný konsenzus mechanismus. Ten určuje několik jednoduchých pravidel, jejichž účelem je právě nalezení shody neboli konsenzu. K nalezení konsenzu je důležitý takzvaný „Proof of Work“ (PoW) algoritmus, což v překladu znamená „důkaz o práci“.

Proof of Work algoritmus slouží především k zabezpečení blockchainové databáze tak, aby nebylo jednoduché přidávat na blockchain nové bloky či retrospektivně upravovat stávající. Proto je k přidání nového bloku na blockchain potřeba vynaložit určitou práci a doložit důkaz o jejím provedení. Prací se v tomto kontextu myslí počítačový výkon, který je potřebný k výpočtu složité matematické úlohy. Jako důkaz o provedené práci je u PoW algoritmu potřeba nalézt správný hash bloku, který musí být menší než určitá hraniční hodnota. Naleznutí správného hashe se

[1] Původní text: „The only way to confirm the absence of a transaction is to be aware of all transactions.“

díky upravování hraniční hodnoty podaří v průměru jednou za deset minut. Proof of Work mechanismus umožňuje uzlům v síti jednoduše ověřit validitu bloku porovnáním hashe bloku s aktuální hraniční hodnotou, a dosáhnout tedy konsenzu napříč celou sítí.

3.3 Těžba

Těžba je proces provádění PoW algoritmu a jeho hlavní význam je zabezpečení databáze blockchain. Tento pojem odkazuje na fakt, že těžba je jediný způsob, jak se do oběhu dostávají nové bitcoiny, a přirovnává se tak k procesu těžby drahých kovů. Vytěžený blok je ten, který byl úspěšně zapsán těžařem na blockchain. Těžbu provádí speciální počítače optimalizované pouze pro tento účel. Těmto počítačům se také říká těžební stroje nebo zkráceně těžaři.

Těžaři validují nové transakce a ukládají je na blockchain v podobě bloků. K uložení nového bloku musí poskytnou důkaz o páci v podobě hashe bloku, který je menší než stanovená hraniční hodnota. K hašování bloku se používá dvojí hašování funkcí SHA256. Vzhledem k tomu, že je hašovací funkce ze své podstaty deterministická, existuje pro identický blok pouze jedna hodnota hashe. Proto hlavička bloku obsahuje vlastnost *nonce*, jejíž hodnota slouží k jedinému účelu, a to k upravení výsledného hashe bloku. K tomu je využívána vlastnost hašovacích funkcí, kdy každá nepatrná změna vstupních hodnot deformuje výstupní hash k nepoznání. Jakékoliv upravení hodnoty *nonce*, třeba jen o jeden bit, se tedy projeví kompletní změnou hashe bloku.

Těžaři tedy postupně upravují hodnotu vlastnosti *nonce* a následně hašují hlavičku bloku. Tento proces opakují do té doby, dokud nenaleznou správný hash, který má menší hodnotu, než je aktuální hraniční hodnota. Hlavní náplní práce těžařských strojů je tedy nepřetržité počítání nových hashů funkcí SHA256. Proto jsou k těžení využívány stroje optimalizovány právě k rychlému počítání hashů, kterým se říká ASIC. Výkonnost těžebních strojů je definována veličinou označovanou jako hashrate, která určuje kolik hashů je stroj schopen vypočítat za jednotku času.

Základní jednotkou veličiny hashrate je H/s, tedy hash za sekundu. Nejvýkonnější ASIC stroje dovedou spočítat přes 100 TH/s, tedy 100 biliónů hashů za sekundu. Celková hodnota hashrate všech těžařských strojů v celé síti v dnešní době přesáhla 200 ET/s neboli 200 triliónů hashů za sekundu.

Před zahájením těžení musí nejprve těžař nový blok zkonstruovat. Hlavní část bloku jsou samotné transakce. Ty těžař vybírá podle svého uvážení z takzvaného mempoolu. Mempoolu je databáze obsahující transakce, které jsou určeny k zapsání na blockchain. Z vybraných transakcí těžař následně generuje hašový strom, jehož kořen ukládá do vlastnosti *merkle root*. Těžař také nastavuje vlastnost *timestamp*, která by měla obsahovat aktuální čas. Tato hodnota musí být kvůli bezpečnosti větší než medián hodnot *timestamp* předchozích jedenácti bloků. Toto omezení zabraňuje těžařům vkládat do vlastnosti *timestamp* příliš vzdálené hodnoty. Těžař nakonec vkládá aktuální hraniční hodnotu do vlastnosti *nBits*. [8]

Hraniční hodnota slouží k upravování obtížnosti PoW algoritmu tak, aby byla dodržena desetiminutová perioda vytěžení nového bloku. Platný hash blok musí nabývat menší hodnoty, než je aktuální hraniční hodnota. Proto čím menší hraniční hodnota je, tím je obtížnější zkonstruovat platný blok se správným hashem. Hraniční hodnota je periodicky v čase upravována tak, aby obtížnost reflektovala rostoucí rychlost hardwaru a proměnlivý zájem o provozování těžby [19]. Přenastavení hraniční hodnoty probíhá jednou za 2016 bloků, což by mělo odpovídat 14 dnům. Toto časový interval se také nazývá epocha. Nová hraniční hodnota je proporcionálně upravena na základě skutečné doby vytěžení bloků v epoše. Pro zjištění skutečné doby těžby se využívá vlastností *timestamp* v hlavičce bloků, které nesou informaci o času vytěžení. [27]

Vzhledem k velké výkonnosti těžebních strojů je hraniční hodnota extrémně malá. K nalezení odpovídajícího hashe je tedy potřeba upravit hlavičku bloku nespočetněkrát. Bohužel vlastnost *nonce* není ani zdaleka dostačující k potřebnému množství alternací hlavičky. Vlastnost *nonce* může nabývat pouze 2^{32} možných hodnot. Takový počet hashů vypočte moderní těžařský stroj za přibližně 0.00004

sekundy. Proto těžaři pro upravení hashe hlavičky bloku využívají také upravování hodnoty *timestamp* nebo takzvané coinbase transakce, jejíž změna se promítne do hodnoty *merkle root*. [27]

Těžba je výkonnostně a energeticky velice náročná, a proto je potřeba těžaře k tomuto procesu motivovat. Za tímto účelem jsou těžaři za každý úspěšně zapsaný blok odměňováni. Tato odměna se skládá z nově vzniklých bitcoinů a transakčních poplatků. Výše odměny v podobě nově vzniklých bitcoinů je periodicky v čase snižována o polovinu, a to jednou za 210000 bloků, což přibližně odpovídá čtyřem rokům. Této události snižování odměny na polovinu se říká halving neboli půlení. Zpočátku byli těžaři odměňováni 50 bitcoiny za každý blok a po každém půlení se tato odměna snížila na polovinu. Tímto způsobem se budou odměny exponenciálně snižovat až do roku 2140, kdy odměna v podobě nových bitcoinů bude rovna nule. V této době bude v oběhu přibližně 21 miliónů bitcoinů a nebudou již vznikat žádné nové. Bitcoin se tedy stane deflačním.

Druhá část odměny, kterou těžaři získávají za vytěžený blok, je tvořena součtem všech transakčních poplatků v bloku. Každá Bitcoinová transakce obsahuje transakční poplatek, jehož hodnotu nastavuje odesílatel. Čím větší poplatek odesílatel určí, tím je větší motivace pro těžaře tuto transakci vytěžit. Tímto způsobem lze nepřímo ovlivnit pořadí zapsaných transakcí na blockchain.

Součet odměn pro těžaře je zapsán ve speciální transakci, která se nazývá coinbase transakce. Coinbase transakce musí být v seznamu transakcí v bloku vždy na prvním místě. Tuto transakci vytváří sám těžař, proto může určit libovolnou adresu příjemce, přičemž její hodnota je součtem transakčních poplatků a nově vzniklých bitcoinů. Jelikož v coinbase transakci neexistuje žádný odesílatel, těžaři tak na místo, kde je běžně ukládána adresa odesílatele, ukládají vlastní libovolná data. Těžaři tento prostor často využívají jako takzvanou *extra nonce*, která slouží pro stejný účel jako vlastnost *nonce*, tedy k upravení hlavičky bloku a následně výsledného hashe. [4]

Při těžbě může jednou za čas dojít k situaci, kdy těžaři vytěží dva bloky ve stejnou chvíli. Tehdy vyvstává otázka, na který blok mají ostatní těžaři navázat blok další. Tento problém má jednoduché řešení: těžaři si mohou vybrat, na který blok budou navazovat. Obvykle to je ten, který se k nim dostane dříve. Této situaci se říká rozštěpení řetězce nebo fork. Abychom mohli rozhodnout, který z rozštěpených řetězců je platný, existuje v Bitcoinu jednoduché pravidlo. Platný je ten, který je nejdelší, a tudíž na jeho těžbu bylo vynaloženo nejvíce energie. Toto pravidlo se podle tvůrce Bitcoinu nazývá Nakamotův konsenzus. Pokud tedy dojde k rozštěpení řetězce, část těžařů těží nové bloky v jedné větvi a část v druhé. Nakonec se uzná řetězec, ve kterém se těžařům jako prvním podaří vytěžit další blok, a tím se jejich řetězec stane nejdelším. [21]

3.4 Bitcoinové Uzly

Bitcoin je tvořen sítí propojených a navzájem komunikujících uzlů. Jako uzly jsou označovány zařízení, na kterých běží Bitcoinový program. Ke komunikaci uzly používají peer-to-peer síť prostřednictvím níž mohou objevovat nové uzly a předávat si napřímo mezi sebou potřebná data. Uzly jsou odpovědné především za provádění validace transakcí, sdílí ověřených a nových transakcí a uchovávání aktivního blockchainu databázi. [12]

Uzly se dělí na několik typů podle jejich účelu a podporovaných funkcí. Uzly mohou obsahovat celkem čtyři funkce, a to: směrování, blockchain databázi, těžení a služby peněženky. Všechny typy uzlů implementují funkci směrování, která zahrnuje komunikaci s ostatními uzly a validování a propagování transakcí. Uzly udržující celou aktuální kopii databáze blockchain jsou nazývány jako *úplné*. Úplné uzly mohou samostatně provádět ověřování transakcí bez nutnosti externího zdroje. Součástí úplného uzlu jsou také často funkce peněženky umožňující spravování privátních klíčů a podepisování transakcí. Existují také takzvané *SPV uzly*, které pro ověřování transakcí používají metodu zvanou *simple payment verification* (SPV). Tato metoda využívá při ověřování transakcí blockchain jiných uzlů. Proto SPV uzly

nemusí obsahovat kopii celého blockchainu. Nakonec jsou součástí sítě uzly provádějící proces těžby, kterým se říká těžební uzly. [4]

3.5 Vývoj Bitcoinové sítě

Základním kamenem Bitcoinové sítě je takzvaný Bitcoinový protokol. Ten definuje celou řadu pravidel, které musí účastníci této sítě dodržovat. Od Bitcoinového protokolu se odvíjí jednotlivé softwary, kde nejznámější je Bitcoin Core, jehož první verze byla vyvinuta samotným Satoshi Nakamotem. Jedná se o open-source software, proto k jeho kódu může kdokoliv přistupovat, popřípadě ho upravovat. Bitcoinový protokol je však čas od času potřeba upravit, a to buď za účelem opravení nežádoucího chování či vylepšení.

Všechny návrhy k vylepšení či změně nějaké části Bitcoinu jsou zaznamenávány do veřejně přístupných dokumentů, které se ukládají na server GitHub. Tyto dokumenty se nazývají „Bitcoin Improvement Proposal“ v překladu „Návrh na vylepšení Bitcoinu“, pro zjednodušení se používá zkratka BIP. Vzhledem k tomu že Bitcoin je open-source systém tak nový návrh na vylepšení může vytvořit prakticky kdokoliv. Po zveřejnění nového BIP však nelze zajistit že daný dokument bude schválen a implementován do Bitcoinové sítě. Právě naopak, většina BIP dokumentů jsou bitcoinovou komunitou odmítnuty.

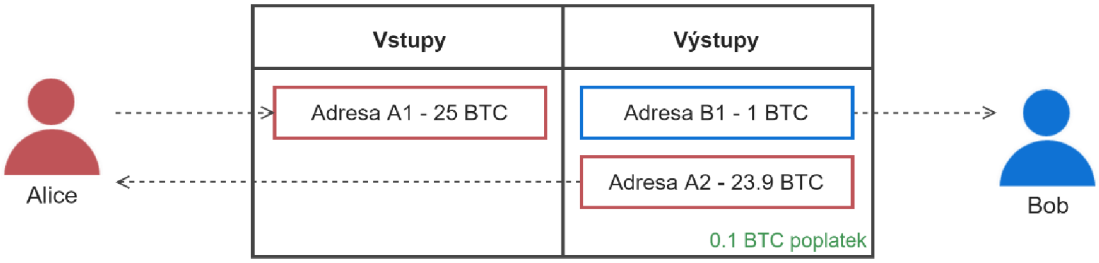
V některých případech mohou BIP dokumenty popisovat změnu samotného konsenzu, který určuje platnost transakcí a bloků. V tomto případě se pak změny dělí na hard fork a soft fork. Hard fork změny znamenají větší zásah do protokolu a vedou k rozdělení blockchainu na dvě větve. K rozštěpení blockchainové databáze dochází z důvodu nekompatibility nových bloků nebo transakcí V některých případech mohou BIP dokumenty popisovat změnu samotného konsenzu, který určuje platnost transakcí a bloků. V tomto případě se pak změny dělí na *hard fork* a *soft fork*. Hard fork jsou specifické tím že nejsou předně kompatibilní. Bloky vzniklé po přijetí hard fork změny by nebyli podle starých pravidel akceptovány. Proto nasazení nějaké hard fork změny často vede k rozdělení blockchainu na dvě větve.

Jedna větev je tvořena uzly, které přijali novou změnu a druhá ty ostatní. Hard forky znamenají velký zásah do Bitcoinového protokolu, a proto se jim většinou vývojáři snaží vyhnout a raději použít soft forky. Změny typu soft fork jsou naopak předně kompatibilní. Po nasazení těchto změn jsou nově vzniklé bloky a transakce validní i podle starých pravidel. Soft forky nejsou ve skutečnosti ani forky, protože nevedou k rozštěpení blockchain databáze. K aktivaci soft fork změn musí souhlasit většina těžařů v síti jinak je zamítnuta. K rozhodování o přijímání soft fork změn Bitcoinový protokol obsahuje hlasovací mechanismus pro těžaře, který byl navržen v rámci dokumentu BIP-34. [4]

3.6 Transakce

Bitcoinová transakce obsahuje informace o tom, kdo komu posílá kolik bitcoinů. Přičemž jedna transakce nemusí obsahovat pouze jednoho odesílatele a jednoho příjemce. Jediným limitem pro počet příjemců a odesílatelů je maximální velikost bloku.

Jednotlivé transakce se skládají ze vstupů a výstupů. Vstupy si lze zjednodušeně představit jako adresy, ze kterých jsou bitcoiny "odesílány". Výstupy jsou pak adresy, na které se dané bitcoiny připisují. Jednoduchý příklad je zobrazen na Obrázek 3, kde Alice posílá Bobovi 1 BTC. U transakcí platí pravidlo, že vstup je nutné utratit celý, a nelze utratit pouze část. V případě, kdy Alice má na adrese A1 25 BTC, musí být celá sumu zahrnuta do transakce. Pokud nechce Bobovi poslat celých 25 BTC, musí do výstupů zařadit svojí další adresu, na kterou budou odeslány zbylé bitcoiny. Alice by teoreticky mohla do výstupů zadat stejnou adresu jako ve vstupech, tedy A1, ale kvůli bezpečnosti a větší anonymitě je zvykem pro každou transakci generovat vždy nové adresy.



Obrázek 3: Transakce [autor]

Další nedílnou součástí transakce je poplatek, který slouží jako odměna pro těžaře, který danou transakci ověří. Poplatek je vypočítán jako rozdíl hodnoty vstupů a výstupů. V příkladu výše je na vstupu celkem 25 BTC a na výstupu 24.9 BTC, poplatek pro těžaře je tedy 0.1 BTC.

Tabulka 3 popisuje datovou strukturu Bitcoinových transakcí. Každá transakce obsahuje jako první vlastnost verzi. Hodnota verze určuje strukturování transakce a pravidla použitá pro její validaci. Dále transakce definuje vstupy, výstupy a jejich počty. Poslední vlastnost, kterou transakce obsahuje je *nLocktime*, která spolu s vlastností *nSequence* definovanou v jednotlivých vstupech, slouží k dočasnému zablokování transakce, tak aby nemohla být utracena. Tomu se podrobněji věnuje kapitola *Časové zámky*. [15]

Tabulka 3: Datová struktura transakce (převzato z [10])

Vlastnost	Velikost	Popis
Verze	4 B	Verze struktury dat v transakci.
Počet vstupů	1–9 B	Udává počet vstupů v transakci.
Vstupy	Proměnná	Množina vstupů.
Počet výstupů	1-9 B	Udává počet výstupů v transakci.
Výstupy	Proměnná	Množina výstupů.
nLocktime	4 B	Určuje čas, do kdy je daná transakce uzamčena (nelze vytěžit).

Každá transakce je jednoznačně identifikována pomocí TXID neboli „Transaction Identifier“. TXID má vždy velikost 32 bajtů a to proto, že jeho hodnota je vypočítána pomocí hašovací funkce SHA-256. Ta má vždy na výstupu hodnotu o velikost 256 bitů neboli 32 bajtů. Hodnota TXID je určena dvojitým zahašováním transakčních dat. Nejprve se tedy zahašují transakční data, a poté se vzniklý hash znovu zahašuje stejnou funkcí. Tím vznikne identifikátor transakce, který je jedinečný napříč celým blockchainem. [29]

3.6.1 Výstupy

V rámci Bitcoinové transakce je pomyslně přesouvána hodnota ze vstupů na výstupy. Výstupy tedy představují stranu příjemce transakce. Každý výstup tedy nabývá nějaké bitcoinové hodnoty. Výstupy mohou být následně použity jako vstupy jiných transakcí. Výstupy, které doposud nebyly použity jako vstupy se označují zkratkou UTXO. Tato zkratka je odvozena od anglického „Unspent Transaction Output“, v překladu „Neutracený transakční výstup“. [4]

Bitcoinové uzly uchovávají informace o všech aktuálně platných UTXO ve speciální databázi. Při zapsání nové transakce na blockchain jsou z UTXO databáze odstraněny vstupy, který byli touto transakcí utraceny. Zároveň se do databáze přidají výstupy, které touto transakcí vznikly. Celkový počet UTXO v databázi tedy stoupá, pokud nová potvrzená transakce obsahuje více výstupu než vstupů. Pokud má transakce více vstupů než výstupu, tak celkový počet UTXO klesá. UTXO databáze tedy neudržuje informace o žádných historických UTXO, které již byly utraceny. Díky tomu si databáze udržuje relativně malou datovou velikost. Na základě hodnot uložených v UTXO databázi je možné jednoduše určit celkové zůstatky na jednotlivých adresách. [21]

Každý výstup obsahuje pouze tři vlastnosti, které jsou vypsány v Tabulka 4. První vlastnost určuje, jakou hodnotu výstup nabývá, tedy kolik bitcoinů lze s použitím tohoto výstupu utratit. Následně výstup obsahuje takzvaný uzamykací kód neboli *scriptPubKey* a jeho velikost. Primárním účelem uzamykacího kódu je

kryptografické uzamčení vstupu tak, aby ho mohla utratit pouze vlastník. K odemknutí výstupu je většinou nutné prokázat vlastnictví odpovídajícího privátního klíč.

Tabulka 4: Datová struktura transakčního výstupu (převzato z [10])

Vlastnost	Velikost	Popis
Hodnota	8 B	Hodnota výstupu v satoshi.
Velikost scriptPubKey	1–9 B	Udává velikost uzamykacího kódu.
scriptPubKey (uzamykací kód)	Proměnná	Skript, který slouží k uzamčení výstupu.

K definování uzamykacího kódu se používá jednoduchý programovacího jazyka Skript. Tomuto tématu se věnuje kapitola *Skriptování*.

3.6.2 Vstupy

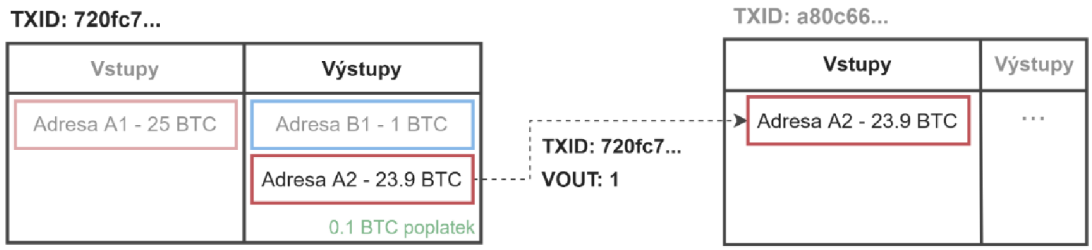
Hlavním účelem transakčních vstupů je identifikace a odemknutí výstupů, které mají být transakcí utraceny. Každý vstup musí také poskytnout takzvaný „proof of ownership“ neboli důkaz o vlastnictví daného UTXO. Jinými slovy musí mít nastavený odpovídající odblokovací kód, který splňuje podmínky definované v uzamykacím kódu daného výstupu. [4]

V Tabulka 5 níže jsou vypsány jednotlivé vlastnosti jednoho vstupu. TXID a VOUT jednoznačné identifikaci UTXO, který bude utracen. TXID identifikuje určitou transakci zapsanou v blockchainu a VOUT udává index výstupu. Dále transakce obsahuje *scriptSig* neboli odblokovací kód a jeho velikost. Ten se používá k pomyslnému odemknutí uzamykacího skriptu v UTXO. To slouží jako důkaz o vlastnictví bitcoinů v UTXO.

Tabulka 5: Datová struktura transakčního vstupu (převzato z [10])

Vlastnost	Velikost	Popis
TXID	32 B	Odkaz na transakci obsahující výstup, který chceme použít jako vstup.
VOUT	4 B	Index vybraného výstupu.
Velikost scriptSig	1–9 B (VarInt)	Udává velikost odblokovacího kódu.
scriptSig (odblokovací kód)	Proměnná	Skript k odblokování výstupu.
nSequnce	4 B	Udává váhu transakce při nahrazení.

Na Obrázek 4 je znázorněn příklad identifikace výstupu pomocí TXID a VOUT. V tomto případě odkazuje transakční vstup na transakci s TXID „*720fc7...*“. Transakce s tímto TXID je vyhledána v blockchainu, a na základě hodnoty VOUT je vybrán druhý transakční výstup. Na vstupu nové transakce se tedy použije výstup s adresou A2 o hodnotě 23.9 BTC.



Obrázek 4: Odkaz na UTXO [autor]

3.6.3 Skriptování

Bitcoinová síť používá k uzamykání a odemykání transakčních výstupu jednoduchý programovací jazyk nazývaný Skript. Ten vychází z jazyka Forth a proto používá postfixovou notaci, kde jsou operátory zapisovány až za operandy nebo funkcemi. Jednoduše to lze znázornit na příkladu sčítání dvou čísel, kde obě čísla, tedy operandy, jsou umístěny před operátorem plus. Tento příklad je zobrazen na Obrázek 5.

Obrázek 5: Příklad postfixové notace [autor]

Další vlastností jazyka Skript je jeho záměrná turingovská neúplnost, tedy není možné jím vypočítat veškeré možné úlohy. Jazyk také neobsahuje žádné cykly, aby nemohlo dojít k zacyklení. Skript je takzvaně stack-based, tedy založený na zásobníku. Při spuštění kódu se provádí jednotlivé hodnoty zleva doprava. Operandy jsou postupně vkládány do zásobníku a funkce jsou vykonávány nad hodnotami nejvýše v zásobníku. Tato metoda se také nazývá LIFO neboli „Last In, First Out“, v překladu „poslední dovnitř, první ven“. Při vykonání skriptu na Obrázek 5, by se do zásobníku nejprve vložila hodnota osm, poté čtyři. Následně by byl vykonán příkaz sečtení, který by na místo obou čísel vložil hodnotu 12.

V Bitcoinovém jazyce Skript je nadefinována řada operačních kódů neboli opkódů, které určují, jaké operace se mají nad hodnotami v zásobníku provést. Například pro sečtení dvou čísel by namísto operátoru „+“ byl použit opkód OP_ADD. V Tabulka 6 níže jsou vypsané běžně používané operační kódy v Bitcoinové síti. Ve Skriptu jsou jako nepravdivé hodnoty (false) představují pouze hodnoty nula, negativní nula a prázdné pole. Všechny ostatní hodnoty jsou pak pravdivé (true).

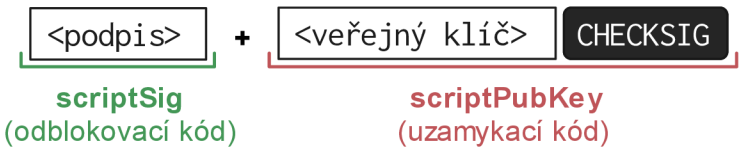
Tabulka 6: Příklad operačních kódů (převzato z [10])

Operační kód	Popis
OP_DUP	Duplikuje horní položku ze zásobníku.
OP_HASH160	Provede dvojité hašování horní položky, nejprve funkcí SHA-256 a poté funkcí RIPEMD-160.
OP_EQUAL	Pokud jsou horní dvě položky v zásobníku identické, je na místo nich vložena hodnota 1, v opačném případě hodnota 0.
OP_VERIFY	Zkontroluje horní položku v zásobníku a pokud je nepravdivá (false), celý skript je ukončen jako neúspěšný (transakce není validní).
OP_EQUALVERIFY	Provede nejprve opkód OP_EQUAL a poté OP_VERIFY. Nejprve tedy porovná dvě horní položky v zásobníku a pokud nebyli shodné, skript je ukončen.
OP_CHECKSIG	Na základě podpisu a veřejného klíče získaných ze zásobníku funkce ověřuje, zda podpis dešifrovaný veřejným klíčem odpovídá zahašované transakci. V případě shody byl podpis vygenerovaný stejným privátním klíčem jako veřejný klíč a funkce vloží do zásobníku hodnotu 1, v opačném případě 0.
OP_RETURN	Ukončí skript jako neúspěšný (transakce není validní).
OP_CHECKMULTISIG	Provede ověření vícero podpisů oproti veřejným klíčům a vyhodnotí, zda bylo nalezeno dostatečné množství shod definovaných uzamykacím kódem.

Jazyk Skript obsahuje desítky opkódů, které můžeme rozdělit na: logické, aritmetické, konstanty, řídící struktury, kryptografické, časové zámky a pro práci se zásobníkem. Za pomoci opkódů spolu s hodnotami je možné vytvořit rozsáhlou škálu uzamykacích kódů (*scriptPubKey*) tak, aby vyhovovaly danému scénáři. Transakční výstup je díky *scriptPubKey* zabezpečen tak, aby ho mohla utratit pouze entita se správným odblokovacím kódem *scriptSig*, který je také psán v jazyce Skript. K uzamčení vstupů se často využívá takzvaných digitálních podpisů, které slouží jako důkaz o vlastnictví privátního klíče. Konkrétně je v rámci Bitcoinové sítě

využíván algoritmus „The Elliptic Curve Digital Signature Algorithm“ zkráceně ECDSA, který k digitálním podpisům využívá eliptických křivek. [4]

Při validaci transakce, kterou provádí každý validační uzel v síti, se pro každý transakční vstup nalezne odpovídající UTXO. Pro každou dvojici vstupu a UTXO je spojením odblokovací kódu ze vstupu a uzamykací kódu z UTXO vytvořen validační skript. Ten je následně spuštěn z výsledku je vyhodnoceno, zda je odblokovací kód validní pro uzamykací kód a je tedy možné UTXO utratit. Na Obrázek 6 je znázorněno spojení odblokovacího kódu *scriptPubKey* a uzamykacího kódu *scriptSig* při validaci. Pro zjednodušení je z jednotlivých operačních kódů odstraněn prefix „OP_“. V tomto případě se jedná o vzor „Pay To Public Key“ (P2PK), kde *scriptPubKey* obsahuje veřejný klíč a *scriptSig* digitální podpis odpovídající danému klíči. UTXO, který obsahuje *scriptPubKey* v tomto formátu, může utratit pouze vlastník privátního klíče, kterým byl vygenerován veřejný klíč a jako důkaz poskytnout digitální podpis. [6].



Obrázek 6: scriptSig a scriptPubKey ve formátu P2PK [autor]

Při spuštění kódu ve formátu P2PK je do zásobníku nejprve vložen podpis, poté veřejný klíč a následně se provede funkce definovaná opkódem OP_CHECKSIG. Tato funkce bere ze zásobníku dvě hodnoty, a to veřejný klíč a podpis. Funkce následně kontroluje, zda podpis spolu s veřejným klíčem odpovídají dané transakci. Tím ověřuje, že transakční data nebyla změněna a podpis byl vygenerován stejným privátním klíčem jako klíč veřejný.

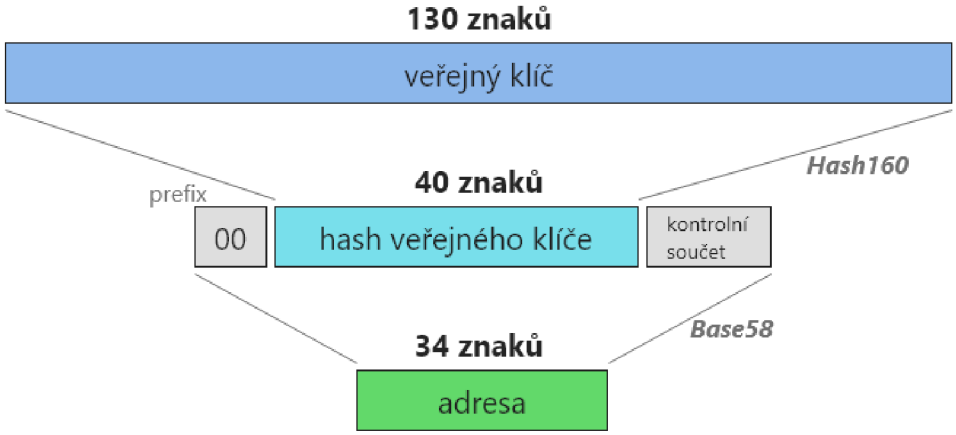
V Bitcoinové síti se mimo P2PK standardně používá několik dalších vzorů pro vytváření scriptPubKey. Tyto vzory jsou zpravidla zatíženy nějakým kryptografickým opkódem za účelem vysokého zabezpečení. Mezi základní standardní vzory se řadí:

- P2PKH (Pay to Public Key Hash) – platba na zahašovaný veřejný klíč,
- P2MS (Pay to Multisig) – platba na množinu podpisů,
- P2SH (Pay to Script Hash) – platba na zahašovaný skript,
- OP_RETURN – speciální vzor nesloužící k platbě, ale pouze k uložení dat do uzamykacího kódu.

Před zavedením rozšíření SegWit, kterému je věnována samostatná kapitola, byl podle Bistarelli et al. [6] nejběžněji používaný vzor P2PKH. Uzamykací kód *scriptPubKey* odpovídající tomuto vzoru obsahuje zahašovaný veřejný klíč a pro jeho odemknutí musí *scriptSig* obsahovat podpis spolu s veřejným klíčem, jak je znázorněno na Obrázek 7. Použití hashe veřejného klíče má řadu výhod. V první řadě je zajištěna bezpečnost v případě, že by byl prolomen podpisový algoritmus ECDSA. Mimo to je hash veřejného klíče značně menší, než původní klíč čímž se šetří místo na blockchainu. [27]

Obrázek 7: scriptSig a scriptPubKey ve formátu P2PKH [autor]

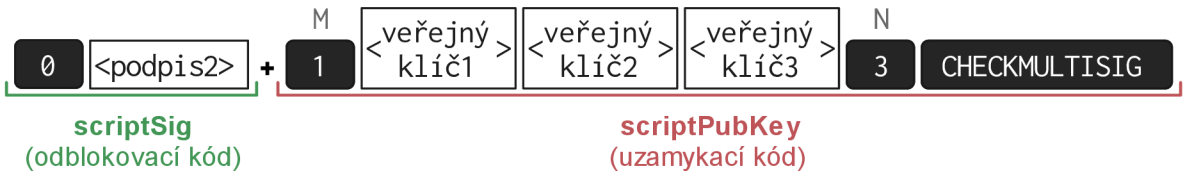
Vzor P2PKH také umožnil vznik takzvaných bitcoinových adres, které obsahují zakódovaný veřejný klíč. Ty umožňují uživatelsky přívětivější sdílení hashe veřejného klíče mezi příjemcem transakce a odesílatelem. Adresy jsou znatelně kratší než samotný hash veřejného klíče a díky tomu jsou lépe čitelné a hůře zaměnitelné. Při vytváření adresy se kódují pomocí base58 tři údaje, a to prefix, hash veřejného klíč a kontrolní součet. Hodnota prefixu určuje typ adresy a kontrolní součet slouží k ověření, že klíč nebyl změněn. Výsledná adresa má velikost pouhých 34 znaků.



Obrázek 8: Generování Bitcoinové adresy z veřejného klíče [autor]

Vzor „Pay To Multisig“ (P2MS), je využíván pro vytvoření takzvané multisig platby, která uzamyká UTXO pomocí množiny veřejných klíčů. Pro odemknutí takového UTXO je zapotřebí několik (nebo všechny) podpisů odpovídajícím specifikovaným veřejným klíčům. Tyto transakce jsou také označovány jako M-of-N, kde M je počet potřebných podpisů a N celkový počet veřejných klíčů. Stěžejním opkódem tohoto vzoru je OP_CHECKMULTISIG, který kontroluje určený počet podpisů oproti veřejným klíčům. *ScriptPubKey* ve formátu P2MS má tvar „*M <veřejný klíč 1> ... <veřejný klíč N> N CHECKMULTISIG*“, kde *N* určuje celkový počet veřejných klíčů a *M* počet potřebných podpisů pro odemknutí. Platný *scriptSig* musí tedy obsahovat počet podpisů odpovídající parametru *M*. [1]

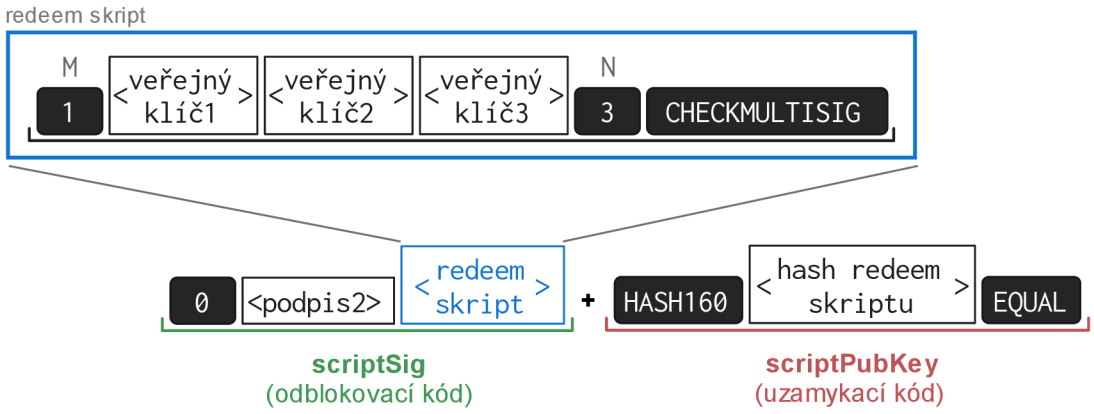
Na Obrázek 9 je znázorněn příklad skriptů ve vzoru P2MS. V tomto případě se jedná o 1-of-3 multisig platbu. K utracení takové platby je nutné předložit jeden ze 3 odpovídajících podpisů. Součástí odblokovacího kódu je také konstanta OP_0. Ta musí být v každé multisig transakci, kvůli historické chybě ve funkci



Obrázek 9: scriptSig a scriptPubKey ve formátu P2MS [autor]

OP_CHECKMULTISIG, která neopatřením ze zásobníku čte o jeden parametr navíc. Tuto chybu však není možné jednoduše vyřešit, protože by její řešení způsobilo neplatnost dosud vytvořených P2MS plateb. Od přijetí BIP-147 může tato konstanta nabývat pouze hodnoty OP_0 aby se zamezilo upravování konstanty při prostupování transakce sítí neboli tvárnosti transakce. [17]

Vzor P2MS je však samostatně využíván pouze ojediněle. Většinou se pro uzamknutí UTXO na více veřejných klíčů používá vzor P2MS zapouzdřený ve vzoru „Pay To Script Hash“ (P2SH), který uzamyká UTXO na hash samotného skriptu [4]. Na Obrázek 10 je zobrazen příklad skriptu ve formátu P2SH, který obdobně jako u předchozího příkladu uzamyká výstup na množinu veřejných klíčů, kde pro odemknutí je zapotřebí jeden platný podpis. Odblokovací kód v tomto vzoru obsahuje takzvaný redeem skript, který obsahuje původní uzamykací kód a jeho hash musí odpovídat hashi v uzamykacím kódu. Před redeem skriptem se vyskytují hodnoty, které „uspokojují“ samotný redeem skript. Validace *scriptPubKey* a *scriptSig* ve vzoru P2SH probíhá ve dvou fázích. Nejprve se spustí skript standardním způsobem, při které se potvrdí, že redeem skript odpovídá hashi ve *scriptPubKey*. Pokud hashe odpovídají, redeem skript se deserializuje a je spuštěn nad zásobníkem ze standardního spuštění, který obsahuje hodnoty, které jsou umístěny v *scriptSig* před redeem skriptem. Při spuštění redeem skriptu je ověřeno, že podpis odpovídá jednomu ze třech veřejných klíčů. [3]



Obrázek 10: scriptSig a scriptPubKey ve formátu P2SH [autor]

Použití vzoru P2SH má řadu výhod, především pak tu, že komplexní skript je ve výstupu nahrazen jeho hashem, a tím je daná transakce menší. Tímto způsobem se odesílateli snižují poplatky za zapsání transakce do blockchainu, a naopak se zvyšují entitě která chce daný UTXO následně utratit. Další výhodou je možnost zakódování *scriptPubKey* do bitcoinové adresy, stejně jak je tomu u vzoru P2PKH. [2]

3.6.4 Časové zámky

Časové zámky jsou nástroje umožňující dočasné zamezení utracení transakce nebo výstupu. Bitcoinový protokol definuje celkem čtyři typy časových zámků: *nLocktime* CLTV, *nSequence* a CSV.

Časový zámek na úrovni transakce může být definován vlastností *nLocktime*. Ta je definována buďto časovou hodnotu, nebo indexem bloku. Transakce, která má nastavený *nLocktime*, není možné do stanoveného času rozšířit po Bitcoinové síti a také zapsat na blockchain. Před vypršení časového zámku transakce je však možné její vstupy utratit v rámci jiné transakce. Z pohledu příjemce tedy není možné tuto transakci považovat za přijatou, dokud není skutečně zapsána na blockchainu. Příjemce se může spolehnout pouze na to, že po stanovenou dobu nemůže tuto transakci utratit. [9]

Dále je možné definovat časový zámek přímo v rámci uzamykacího kódu *scriptPubKey*. Za tímto účelem Skript obsahuje opkód CHECKLOCKTIMEVERIFY (dále jen CLTV), který byl navrhnut v rámci BIP-65. Před samotným opkódem se vyskytuje časová hodnota, která může být definována časem nebo indexem bloku. Při použití CLTV je transakční výstup uzamčený až po vytěžení transakce. Transakční výstup pak není možné po stanovenou dobu utratit. [28]

Bitcoinový protokol dále obsahuje takzvané relativní časové zámky, které uzamykají transakce po stanovenou dobu od vytěžení. K tomu je možné využít vlastnost *nSequence*, která je součástí každého transakčního vstupu. Hodnotu *n* této vlastnosti může obsahovat buď časovým údajem, nebo počtem bloků. Pokud hodnota *n* představuje počet bloků, daný vstup může být utracen až o *n* bloků později, než byl

vytěžen. V případě, že hodnota *n* představuje časový údaj, daný vstup může být utracen až o *n**512 sekund později, než byl vytěžen. [15]

Relativní časový zámek lze také určit na úrovni skriptu za pomoci funkce CHECKSEQUENCEVERIFY zkráceně CSV, která byla popsána v dokumentu BIP-112. UTXO obsahující CSV hodnotu může být utraceno pouze vstupem, který má nastavený relativní časový zámek vlastností *nSequence* na stejnou nebo vyšší hodnotu jako CSV. UTXO tedy není možné utratit po stanovenou dobu od vytěžení. Hodnota CSV i *nSequence* musí být definovány stejným časovým formátem. Na Obrázek 11 je ukázka uzamykacího kódu, který určuje, že dané UTXO nelze utratit po dobu 30 dnů od vytěžení. Toto UTXO lze utratit pouze vstupem, který má *nSequence* časový zámek s hodnotou 30 dní nebo více. CSV relativní časový zámek je obzvláště užitečný pro vytvoření závislosti typu předek-potomek mezi transakcemi. V tomto vztahu nemůže být potomek utracen dříve než předek nebo pokud neuplyne stanovená doba v CSV předka. Tento princip se využívá v síti Lightning network. [13]

Obrázek 11: Příklad použití CSV v uzamykacím kódu výstupu [autor]

Při validaci transakcí s časovými zámky je nu jsou jejich hodnoty porovnávány s hodnotou Median Time Past (MTP). Ta je vypočítána jako střední hodnota *timestamp* posledních jedenácti bloků zapsaných v blockchainu. Tato metrika se používá za účelem snížení vlivu jednotlivých těžařů na čas, který se používá k porovnávání. [16]

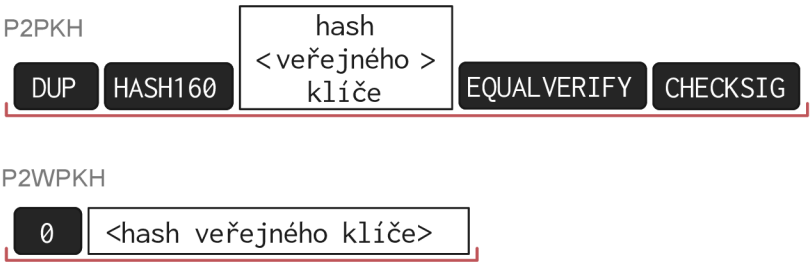
3.7 SegWit

SegWit znamená „Segregate Witness“, což lze volně přeložit jako „Oddělený svědek“. Jedná se o vylepšení Bitcoinové sítě BIP-141, které bylo úspěšně aktivováno soft forkem v roce 2017. Toto vylepšení umožnilo vytváření nového typu transakcí

nazývané SegWit transakce. Ty jsou specifické tím, že odblokovací kód je pomyslně oddělen od transakčního vstupu. SegWit transakce obsahují novou vlastnost s názvem *witness*, kam jsou přesunuty odblokovací kódy jednotlivých vstupů. [4]

V rámci přemístění podpisových dat z transakčních vstupů tyto data neovlivňují výpočet TXID, tím pádem nejsou ani součástí *merkle root*, a tedy neovlivňují výpočet hashe bloku. Každá SegWit transakce obsahuje mimo TXID také nově vlastnost WTXID. Hodnota WTXID je vypočítána dvojitým zahašováním (SHA256) serializovaných transakčních dat včetně podpisových dat *witness*. Každý blok obsahující SegWit transakce, nově také obsahuje vlastnost *witness merkle root*, který je oproti standardní vlastnosti *merkle root* vypočtena ze všech WTXID namísto TXID. Hodnota *witness merkle root* je uložena v *scriptPubKey* coinbase transakce pomocí vzoru OP_RETURN. [18]

SegWit umožnil vznik několika nových standardních vzorů pro uzamykací a odblokovací skript. První z nových vzorů vychází z P2PKH, a tím je „Pay To Witness Public Key Hash“ (P2WPKH) neboli platba na zahašovaný veřejný klíč na odděleném svědkovi. Tento vzor znatelně zjednodušuje a zmenšuje velikost jak scriptPubKey, tak *scriptSig*. Odblokovací kód *scriptSig* je u všech SegWit transakcí vždy prázdný, protože jak již bylo řečeno, hodnoty všech odblokovacích skriptů jsou přesunuta do vlastnosti *witness* přímo na transakci. Witness tedy obsahuje podpis a veřejný klíč. Oproti P2PKH je zjednodušen i uzamykací kód scriptPubKey, který obsahuje pouze dvě položky, a to konstantu OP_0 a hash veřejného klíče o velikosti 20 bajtů. Porovnání *scriptPubKey* ve vzoru P2PKH a P2WPKH je znázorněno na Obrázek 12. [18]

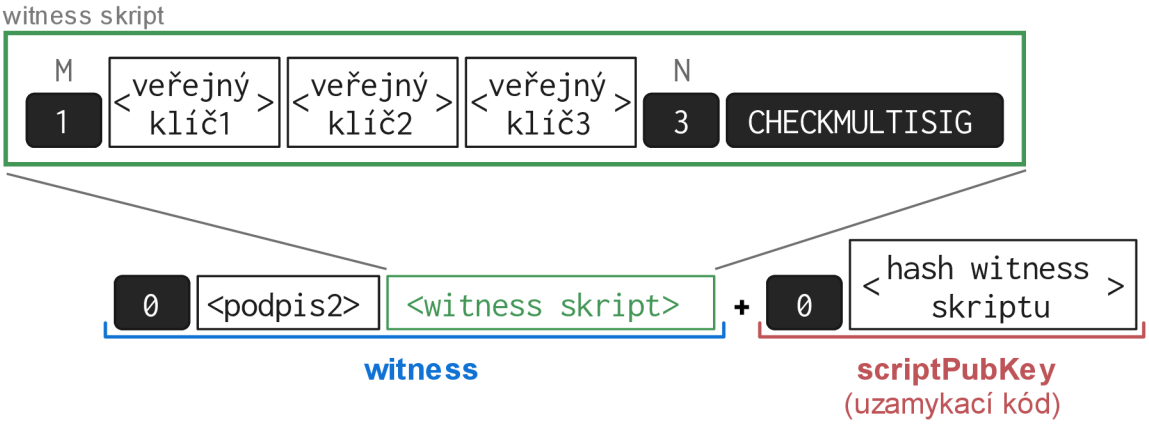


Obrázek 12: Porovnání scriptPubKey ve formátu P2PKH a P2WPKH [autor]

Při validaci skriptu ve formátu P2WPKH, validační uzel na základě struktury *scriptPubKey* rozpozná, že se jedná o tento vzor. Ten obsahuje pouze dvě položky, z toho první je konstanta OP_0 a druhá má velikost 20 bajtů. Uzel následně zkonstruuje virtuální skript odpovídající vzoru P2PKH, do kterého dosadí hash veřejného klíče ze *scriptPubKey* a odpovídající podpis s veřejným klíčem z vlastnosti *witness*. Validace dále probíhá stejným způsobem jako u vzoru P2PKH. Validační uzly, které nepodporují SegWit transakce provádí validaci jako u běžné transakce spuštěním skriptu, který vznikne spojením *scriptSig* a *scriptPubKey*. Protože *scriptSig* je u SegWit transakcí prázdný, tak výsledný skript obsahuje pouze *scriptPubKey*, který obsahuje již zmíněné dvě hodnoty OP_0 a hash veřejného klíče. Tyto hodnoty jsou při validaci postupně vkládány do zásobníku. Vzhledem k tomu, že vrchní položka v zásobníku je hash veřejného klíče, a je tedy nenulová, je skript označen jako validní. Tímto způsobem je zajištěna zpětná kompatibilita s uzly, které používají starší verzi Bitcoin Core, který nepodporuje SegWit. [27]

Další nový SegWit vzor nese název „Pay To Witness Script Hash“ (P2WSH) neboli platba na zahašovaný skript na odděleném svědkovi. Tento vzor vychází z P2SH, a stejně tak umožňuje vytváření komplexních uzamykacích skriptů, jako jsou například multisig platby. Uzamykací kód v tomto vzoru obsahuje opět pouze dvě položky, a to konstantu OP_0 a hash witness skriptu. Witness skript je obdobou redeem skriptu ze vzoru P2SH. Oproti němu je ovšem zahašován funkcí SHA256, což má za následek zvětšení velikosti na 32 bajtů. Velikost hashe je důležitá, kvůli odlišení vzoru P2WPKH a P2WSH. Použití hašovací funkce SHA256 navíc zvyšuje

zabezpečení proti útokům. Hodnota odblokovacího kódu je stejně jako u předešlého vzoru přesunuta do vlastnosti *witness*. Vlastnost *witness*, tedy stejně jako u P2SH, obsahuje hodnoty, které „uspokojují“ witness skript, který musí odpovídat hashi v uzamykacím kódu. Na Obrázek 13 je znázorněna struktura vzoru P2WSH. [18]



Obrázek 13: Vzor P2WSH [autor]

Při validaci skriptu ve vzoru P2SH musí validační uzel, stejně jako u vzoru P2WPKH, nejprve rozpoznat na základě struktury *scriptPubKey* že se jedná o daný vzor. Uzel poté vyhledá odpovídající odblokovací skript ve vlastnosti *witness* a zahašuje witness skript funkcí SHA256. Výsledný hash porovná s hashem, který se nachází ve scriptPubKey. V případě, že jsou hashe identické, pokračuje ve validaci spuštěním skriptu, který je tvořen z položek obsažených ve vlastnosti *witness* a deserializovaným witness skriptem. Tento vzor je obzvláště důležitý pro fungování sítě Lightning network. [27]

Spolu s novými vzory vnikl také v rámci vylepšení BIP-173 návrh na vytvoření nového typu bitcoinové adresy, která slouží k usnadnění používání SegWit transakcí pro koncové uživatele. Tento nový typ adresy se nazývá Bech32 a je v něm zakódován buďto hash witness skriptu v případě vzoru P2WSH, nebo hash veřejného klíče v případě vzoru P2WPKH spolu s konstantou OP_0. Oproti původnímu formátu Base58, neosahuje formát Bech32 velká písmena. Díky tomu lze adresu v tomto formátu zakódovat do QR kódu v alfanumerickém módu, což značně

redukuje velikost výsledného QR kódu. Nativní SegWit adresy ve formátu Bech32 jsou od ostatních adres jednoduše odlišitelné, protože začínají znaky „bc1“. Další výhodou je snazší, rychlejší kódování a dekódování formátu Bech32. [30]

3.7.1 Přínosy

SegWit je architektonická změna, která přispěla k vylepšení Bitcoinu z různých aspektů. Jeden z nich je lepší škálovatelnost bitcoinové sítě, tedy pomyslné zvětšení bloku, aby mohl pojmout více transakcí. Bloky nejsou nově limitovány maximální velikostí 1 MB, ale je zavedena nová metrika, takzvaná váha, označována jako WU. Bloky jsou nyní omezeny touto metrikou, a to s maximální hodnotou čtyři miliony WU. Váha je přiřazena každému bajtu transakce podle typu dat. Každý bajt *witness* dat odpovídá váze 1 WU a každý ostatní bajt v transakci odpovídá váze 4 WU. *Witness* data mají určenou menší váhu, především proto, že je uzly nemusí dlouhodobě udržovat, a nezatěžují tedy tolik Bitcoinovou síť. Díky tomu má SegWit transakce podstatně menší váhu než stejně velká obyčejná transakce, a tak se do bloku vejde odhadem o 70 % transakcí více. [7]

SegWit také zavedl verzování skriptů pomocí konstanty obsažené ve skriptu. Toto číslo umožňuje zpětně kompatibilní vylepšování skriptovacího jazyka pomocí soft fork. Výše zmíněné SegWit skripty vždy začínají konstantou OP_0. Právě tato konstanta určuje verzi skriptu a na základě její hodnoty se mohou vyvolat různá validační pravidla. Tak jako OP_0 upozorňuje uzel na přítomnost witness vlastnosti, tak odlišná konstanta může vyvolat různé jiné akce, které mohou být v budoucnu implementovány. Aktuálně nejnovější verze Bitcoin Core 0.22.0 podporuje verzi skriptu OP_1, která aktivuje takzvaný Taproot. [11]

Jedním z hlavních přínosů SegWit vylepšení je zamezení vytváření takzvaně tvárných transakcí. Takto označovány transakce, které může jakýkoliv uzel při jejím přenosu sítí nepatrně pozměnit před zapsáním na blockchain. Při jakékoliv drobné změně transakce se však změní její identifikátor TXID, a to může znamenat problémy s vyhledáním transakce na blockchainu. U tvárných transakcí není upravit

kritická data jako klíče a hodnoty, protože ty jsou opatřeny digitálním podpisem. Jediné, co lze upravit, je podpis samotný. SegWit tento problém řeší jednoduše tím, že podpisová data nejsou nadále součástí transakce, a tudíž jakákoliv změna jejich hodnoty nemá vliv na identifikátor transakce TXID. Bez zamezení vytváření tvárných transakcí by bylo velice obtížné používat platební síť Lightning Network. [4]

4 Lightning network

Bitcoin je navržen tak, aby umožňoval pouze omezený počet transakcí za sekundu. Proto jsou Bitcoinové transakce při vytížení sítě nákladné a jejich zapsání na blockchain může být zdlouhavý proces. Bitcoinová síť ze své podstaty není určena ke každodenním transakcím právě kvůli jeho špatné škálovatelnosti. Za tímto účelem vnikl nový protokol Lightning Network (LN), který pracuje nad Bitcoinovou sítí a je označován jako jeho druhá vrstva. Lightning network umožňuje provádět instantní platby s nízkými poplatky, a tím řeší špatnou škálovatelnost Bitcoinu. LN navíc přispívá k lepšímu soukromí a obdobně jako u Bitcoinu k tomu používá silné kryptografické algoritmy.

Koncept Lightning Network byl poprvé představen v roce 2015 a první implementace byla spuštěna v roce 2018. Tvůrci Lightning Network protokolu jsou Joseph Poon a Thaddeus Dryja [20]. Architektura Lightning Network je popsána v sadě dokumentů s názvem „Basic of Lightning Network“ zkráceně BOLT. Těchto dokumentů je celkem deset, kde každý definuje specifickou funkcionalitu Lightning network. Všechny BOLT dokumenty jsou open-source a jsou veřejně dostupné na platformě GitHub. Na základě BOLT dokumentů lze vyvinout implementace LN, které bez ohledu na běhové prostředí a programovací jazyk spolu mohou navzájem komunikovat. Dnes se mezi nejpoužívanější implementace řadí *LND*, *c-lightning* a *eclair*, kde každý využívá jiného programovacího jazyka.

Lightning network je obdobně jako Bitcoin tvořen sítí propojených uzlů, které mezi sebou komunikují napřímo takzvaně peer-to-peer. Tyto uzly mezi sebou vytváří takzvané platební kanály, které reprezentují virtuální propojení uzlů. Platební kanál může být vytvořen pouze mezi dvěma kanály, ne více. Uzly do kanálů uzamykají libovolnou hodnotu v bitcoinu za pomoci multisig Bitcoinových transakcí. Skrze kanál lze následně provádět libovolný počet Lightning network plateb, v celkové maximální výši uzamčených bitcoinů. V případě, že mezi odesílatelem a příjemcem LN platby neexistuje přímý platební kanál, je možné LN platbu takzvaně routovat skrz ostatní veřejné platební kanály, které nepřímo spojují tyto dva uzly.

Pokud mezi odesílatelem a příjemcem LN platby neexistuje žádná cesta sítí, není možné platbu provést. [12]

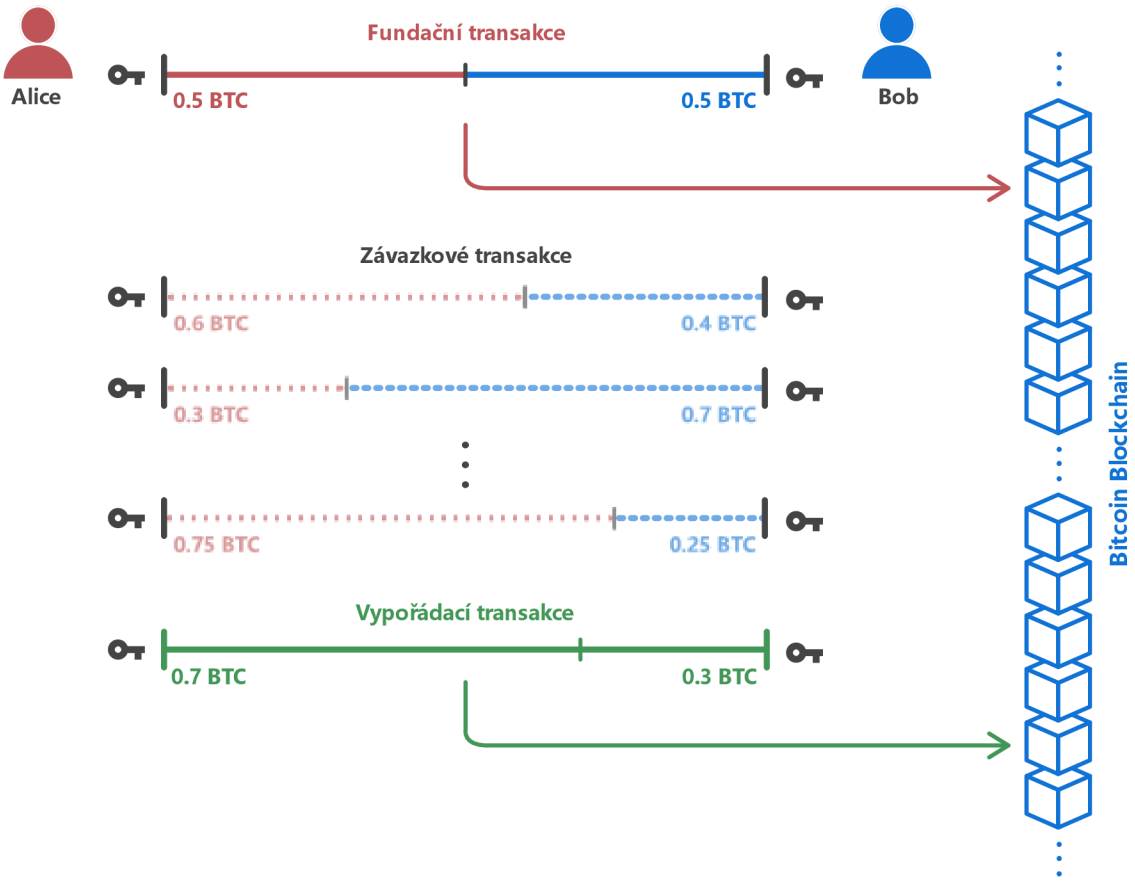
Oproti Bitcoin transakcím nejsou LN platby ukládány na veřejnou databázi. Blockchain je otevřený a přístupný komukoliv, a tím je do určité míry degradováno soukromí uživatelů. V případě Lightning network neexistuje žádné místo, kde by byly zapsány všechny platby. Jednotlivé uzly sítě pouze udržují informace o stavech jejich kanálů. Dokonce i v případě routování cizí platby přes některý z uzlů, se danému uzlu neodhalí příjemce ani odesílatel.

4.1 Platební kanál

Platební kanál v Lightning network síti představuje finanční vztah mezi dvěma uzly. Při vytváření nového kanálu každý z uzlů alokuje do kanálu určité množství bitcoinů, kde součet těchto prostředků je označována jako kapacita kanálu. Při LN platbě přes otevřený kanál jsou rebalancovány jeho prostředky ve prospěch jedné, nebo druhé strany podle toho, jakým směrem platba probíhá. Uzly se při spravování kanálů řídí podle kryptografického protokolu, který zajišťuje, aby žádná strana nemohla podvádět a uzly si tak nemusely důvěřovat. Tento protokol je definovány v dokumentech BOLT #2 a #3. Dokument BOLT #2 popisuje formu komunikace mezi kanály a BOLT #3 určuje strukturu Bitcoinových transakcí, které se při využívání kanálů generují. [5]

Základem platebního kanálu je multisig Bitcoinová transakce 2-of-2. K utracení této transakce jsou potřeba dva klíče, kde každý z uzlů vlastní jeden z nich. Uzly zřizující mezi sebou LN kanál vytváří multisig transakci, do které jsou alokovány libovolné prostředky v bitcoinu. Této transakci se také říká fundační transakce, protože slouží k pomyslnému financování kanálu. Po zapsání na blockchain tato transakce představuje kanál, skrze něj mohou jednotlivé uzly provést LN platby v maximální výši alokovaných bitcoinů. Pokud Alice a Bob mezi sebou zřídí kanál do kterého každý alokuje 0.5 BTC, pak Alice může provést LN platby Bobovi v celkové hodnotě 0.5 BTC. Při každé LN platbě je vytvořena nová multisig platba s rebalancovanými

prostředky. Tato transakce se označuje jako závazková. Pokud by například Bob poslal Alici 0.1 BTC, byla by vytvořena nová závazková transakce s výstupy 0.6 BTC pro Alici a 0.4 BTC pro Boba. Závazkové transakce uzly neprovádí, pouze si uchovávají nejnovější z nich. Tímto způsobem lze provést libovolné množství LN plateb. Zároveň je při vytvoření každé nové závazkové transakce zneplatněna předchozí závazková transakce. V případě, že by jeden z uzlů chtěl kanál ukončit, jednoduše provede poslední závazkovou transakci, čímž je zapsána na blockchain. Na Obrázek 14 je znázorněno otevření kanálu pomocí fundační transakce, následné provádění LN plateb generováním nových závazkových transakcí, a nakonec ukončení kanálu provedením poslední závazkové transakce. [20]



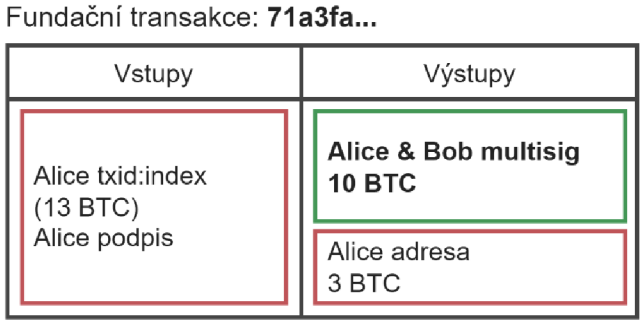
Obrázek 14: Rebalancování LN kanálu [autor]

4.1.1 Zřízení kanálu

Proces zřizování nového kanálu mezi uzly se řídí protokolem, který je definován v dokumentu BOLT #2. Vytvoření nového kanálu vždy iniciuje jeden z uzlů. Tomuto uzlu se také říká zřizovatel, a právě jeho prostředky budou uzamčeny do nového kanálu. V současné době mohou být při vytváření kanálu uzamčeny bitcoiny pouze jednoho z uzlů. Například v případě že by Alice (A) chtěla otevřít nový kanál z Bobe (B), může do nového kanálu alokovat prostředky pouze jeden z nich.

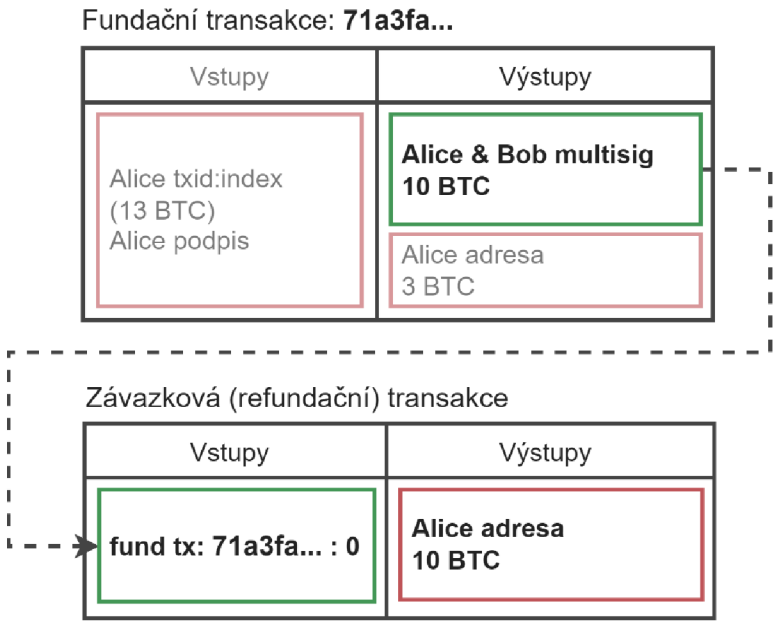
Před otevřením nového kanálu si uzly navzájem vymění několik důležitých dat, které jsou nezbytné k jeho konstrukci. Tyto data obsahují například: množství uzamčených bitcoinů, minimální zůstatek, veřejný klíč protistrany, hodnota časového zámku a další. Na základě těchto parametrů se oba uzly mohou rozhodnout, zda za stanovených podmínek chtějí nový kanál opravdu otevřít. [24]

Pokud se uzly dohodnou na vlastnostech nového kanálu, může zřizovatel zkonstruovat fundační transakci. Jako vstupy fundační transakce uzel A zvolí libovolné UTXO výstupy, které chce pro tuto transakci použít. Součet hodnot UTXO musí být stejný nebo větší než hodnota, na kterém se uzly dohodly. Výstupy jsou v transakci obvykle dva. První slouží k financování kanálu a druhý k navrácení zbylých prostředků zpět na Alicinu adresu. Příklad fundační transakce je na Obrázek 15. Touto transakcí Alice alokuje 10 BTC (1,000,000,000 satoshi) do kanálu a zbylé 3 BTC vrací zpět na její adresu.



Obrázek 15: Příklad fundační transakce [autor]

Výstup sloužící k financování prostředků je uzamknut multisig skriptem. Alice tento skript generuje ze svého a Bobova veřejného klíče. Po vytvoření fundační transakce není ihned odeslána k zapsání na blockchain. Pokud by byla ihned zapsána, Alice by se vystavovala riziku zablokování všech jejích uzamčených prostředků v případě, že by Bob zmizel nebo odmítl poskytnout svůj podpis. Aby Alice zablokování prostředků, musí před zapsáním fundační transakce na blockchain vygenerovat další transakci, kterou bude moct použít k utracení fundační transakce v případě problémů. Tato transakce se nazývá závazková, protože zavazuje oba partnery ke spravedlivému rozdělení zůstatku kanálu. Vzhledem k tomu, že Alice financovala kanál sama, tak závazková transakce zasílá veškeré prostředky právě zpět Alici. Příklad této transakce je znázorněn na Obrázek 16. [5]



Obrázek 16: Příklad závazkové refundační transakce [autor]

Před zavedením SegWit vylepšení by generování závazkové transakce bylo téměř nemožné kvůli tvárnosti Bitcoinových transakcí. U tvárných transakcí není možné jednoznačně určit jejich TXID, dokud nejsou zapsány na blockchain. Při generování závazkové transakce je však potřeba jednoznačně definovat TXID fundační

transakce, a to ještě předtím, než je vůbec zapsána na blockchain. Právě tento problém vyřešilo Bitcoinové vylepšení SegWit. [14]

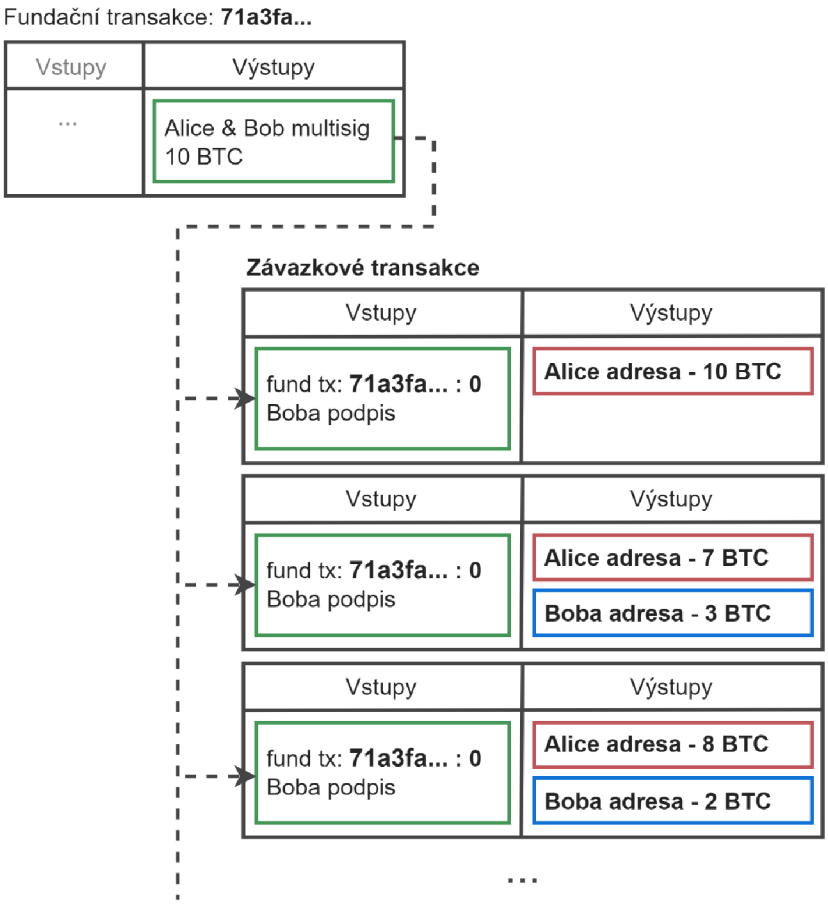
Závazková transakce v této podobě však není validní, neboť neobsahuje potřebné podpisy obou stran. Alice svůj podpis může kdykoliv doplnit, ale zatím nezná Bobův podpis, a proto stále nemůže bez rizika provést fundační transakci. Dalším krokem, který musí Alice učinit je odeslání informací o nové fundační transakci Bobovi. Součástí těchto informací je TXID fundační transakce, index výstupu a Alicin podpis. Získané informace Bob použije k vytvoření vlastní verze závazkové transakce. Rozložení prostředků této závazkové transakci musí vždy odpovídat Alicině transakci, jinak by Alicin podpis nebyl validní. [24]

Nakonec Bob odešle svůj podpis Alici. Díky tomu má Alice k dispozici oba potřebné podpisy k utracení závazkové transakce. Před provedením fundační transakce Alice samozřejmě ověří validitu Bobova podpisu. Pokud je podpis validní, může Alice bez jakéhokoliv rizika odeslat fundační transakci k zapsání na blockchain. V případě, že se po zapsání fundační transakce na blockchain cokoliv pokazí, může Alice použít závazkovou transakci k získání prostředků zpět bez nutnosti kooperace druhé strany. Po zapsání fundační transakce na blockchain je kanál úspěšně otevřen a připraven k používání. [5]

4.2 Platby přes kanál

Provádění plateb přes kanál je prakticky pouze přerozdělování prostředků kanálu generováním nových závazkových transakcí. Při každé LN platbě je hodnota platby přičtena na stranu příjemce a odečtena ze strany odesílatele. Po založení nového kanálu jsou běžně všechny prostředky pouze na straně zřizovatele, a proto je možné provést platbu pouze směrem od něj. K přerozdělování se využívají závazkové transakce, které byly vytvořeny při zřizování kanálu samotného. Při každé LN platbě obě strany generují nové závazkové transakce s aktualizovanými hodnotami výstupů, které reflektují směr a hodnotu dané platby.

V případě kanálu, který byl vytvořen v předchozí kapitole, by závazkové transakce na obou stranách obsahovaly výstup s hodnotu 10 BTC pro Alici a 0 BTC pro Boba. Pokud by Alice chtěla provést LN platbu v hodnotě 3 BTC ve prospěch Boba, oba uzly by vygenerovaly nové závazkové transakce s výstupy 7 BTC pro Alici a 3 BTC pro Boba. Uzly nově vzniklé závazkové transakce neodesílají do Bitcoinové sítě k zapsání na blockchain, ale pouze je ukládají na své lokální úložiště. Na Obrázek 17, je znázorněno provádění LN plateb generováním nových závazkových transakcí. Závazkovou transakci může provést kterákoliv strana v libovolném čase, a tím stvrdit aktuální stav zůstatků. Provedením závazkové transakce je však uzavřen kanál. Smyslem Lightning network je šetření místa na blockchainu a eliminace poplatků při posílání bitcoinů, proto se uzly snaží udržet kanál otevřený co nejdéle a využít jeho kapacitu na co největší počet LN plateb.

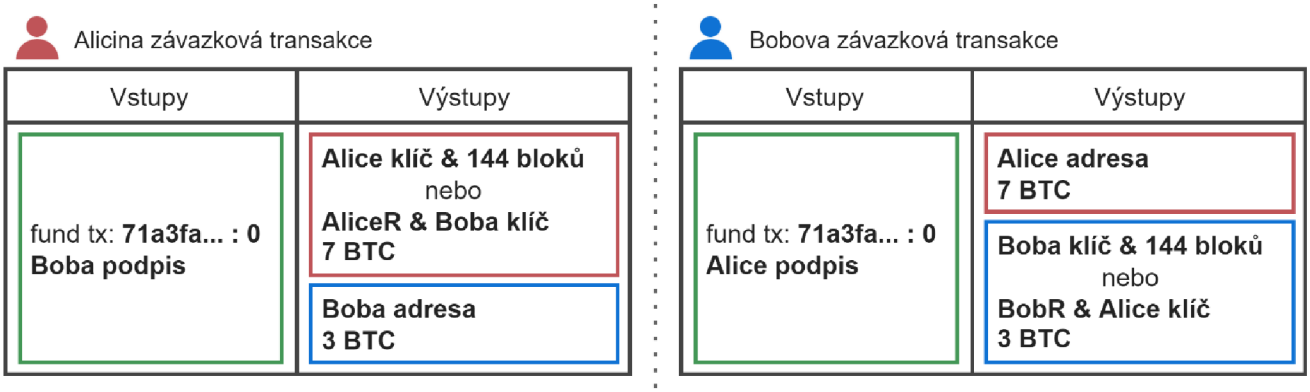


Obrázek 17: Generování závazkových transakcí [autor]

Po vygenerování nových závazkových transakcí však nastává situace, kdy oba uzly udržují dvě nebo více verzí validních transakcí, které může kdokoliv z nich provést. V ideálním případě by každý uzel měl udržovat pouze poslední závazkovou transakci, která reflektuje skutečný stav prostředků v kanálu. V případě, že by jeden z uzlů udržoval i staré transakce, mohl by kdykoliv odeslat ke schválení tu, která je pro něho nejvýhodnější. Přijímatel LN platby by v tomto případě neměl žádný důkaz o tom, že druhý uzel neodešle na blockchain původní transakci, která nereflektuje tuto platbu. V nejhorším případě by mohl zřizovatel kanálu kdykoliv provést první závazkovou transakci, a tím získat všechny prostředky z kanálu. Proto jsou závazkové transakce zatíženy mechanismem, díky kterému je možné starší transakce pomyslně zrušit. Takové transakce jsou takzvaně „revokable“ neboli zrušitelné.

Zrušitelné transakce ve skutečnosti není možné skutečně zrušit a ani zamezit jejich odeslání na blockchain. Tyto transakce pouze obsahují sankční mechanismus zajišťující pokutování uzlu, který se pokusí starší závazkovou transakci použít. I přesto je možné takovou transakci zapsat na blockchain, ale s největší pravděpodobností uzel, který tak učinil, přijde o svou část prostředků. [5]

Závazkové transakce reflektující stejný stav kanálu jsou ve skutečnosti u každého z uzlů trochu odlišné. Na Obrázek 18 je znázorněn příklad závazkových transakcí poté, co Alice poslala Bobovi první platbu prostřednictvím nového kanálu v hodnotě



Obrázek 18: Zrušitelné závazkové transakce [autor]

3 BTC. Každá transakce obsahuje vždy dva výstupy. Jeden z výstupů slouží k odeslání prostředků na adresu protistrany a není zatížen žádnými podmínkami. Pokud například Alice provede svoji závazkovou transakci, Bob může po poskytnutí svého podpisu jeho část transakce jednoduše utratit. Druhý výstup směřující směrem k vlastníkovi dané transakce je vždy zatížen sankčním mechanismem, aby v případě provedení staré transakce mohl tento výstup utratit i druhý uzel jako pokutu za podvod.

Výstup směřující k vlastníkovi je uzamčen skriptem, který je rozdělen na dvě větve, kde alespoň jedna z nich musí být pro odemknutí splněna. První větev skriptu lze odemknout poskytnutím podpisu vlastníka transakce, pouze pokud od jejího zapsání na blockchain uběhl nějaký čas, v tomto případě 144 bloků, což odpovídá přibližně jednomu dni. Tato větev se využívá v případě, kdy byla použita poslední závazková transakce, a jedná se tak o legitimní uzavření kanálu, přičemž žádný z uzlů nepodváděl. Druhou větev tvoří 2-of-2 multisig skript obsahující veřejný klíč protistrany a takzvaný odvolávací veřejný klíč R. Privátní klíč odpovídající odvolávacímu veřejnému klíči vždy vlastní strana generující transakci. Například Alice zná privátní klíč k veřejnému klíči AliceR a naopak Bob zná privátní klíč k veřejnému klíči BobR. Při odemykání této větve skriptu je však nutné znát jak privátní klíč k odvolávacímu klíči, tak privátní klíč protistrany. Alice tedy nemůže tuto větev využít k odemknutí výstupu, protože nezná a nikdy znát nebude Bobův privátní klíč. Právě proto je tato větev transakce určena k utracení výstupu protistranou, a to pouze v případě, že protistrana zná privátní odvolávací klíč, což nastává v případě provedení staré závazkové transakce.

V rámci generování nových závazkových transakcí při provádění LN platby, si oba uzly navzájem vyměňují privátní odvolávací klíče ke starší verzi závazkové transakce. Toto předání funguje jako důkaz o tom, že ani jedna strana nebude používat předchozí transakci, protože by tím ohrozila svoji část prostředků v transakci. Po vygenerování nových závazkových transakcí by Alice zaslala Bobovi

privátní klíč AliceR k předchozí neaktuální závazkové transakci a Bob udělá to samé se svým odvolávacím klíčem. V případě, že by Alice chtěla předchozí transakci zapsat na blockchain a pokusit se tak podvést protistranu, mohl by Bob použít získaný privátní klíč AliceR k utracení Alicin prostředků v dané transakci. V případě, že Alice použije poslední závazkovou transakci, která reflektuje skutečný stav kanálu, Bob nezná privátní klíč AliceR a Alice tak neriskuje ztrátu své části prostředků. Je důležité, aby uzly pro každou novu transakci generovaly vždy nový odvolávací klíč, aby nedošlo k tomu, že protistrana použije odvolávací klíč na nejnovější, a tedy platnou závazkovou transakci. [14]

Důležitou součástí sankčního mechanismu je relativní časový zámek. Ten určuje dobu, jakou musí uzel čekat před utracením vlastních prostředků. Hodnota relativního časového zámku je dána počtem, který udává kolik bloků musí být vytěženo, než bude daný výstup možné utratit. Například pokud Bob provede svoji verzi závazkové transakce, tak po jejím úspěšném zapsání na blockchain musí čekat, než bude zapsáno dalších 144 bloků. Až poté bude moct utratit svůj podíl transakce. Skript obsahuje relativní časový zámek z důvodu poskytnutí času na provedení sankce v případě podvodu. Během doby, kdy uzel, který provedl závazkovou transakci čeká až bude moct svůj díl utratit, může druhý uzel v případě podvodu použít odvolávací privátní klíč R, a tím si nárokovat i prostředky protistrany. Například, pokud Bob provede starší závazkovou transakci, Alice musí stihnout provést utracení Bobova výstupu předtím, něž vyprší časový zámek. Pokud to do té doby Alice nestihne, může Bob svou část transakce utratit, a to i když se jednalo o podvod. Proto musí uzly v sítí aktivně bránit svoje prostředky v kanálech. Hodnota časového zámku tedy určuje časový horizont, ve kterém může podvedená protistrana provést kompenzaci. Mimo to se jedná o dobu, kterou musí uzel čekat před utracením vlastních prostředků v případě, že právě on použil závazkovou transakci. Hodnotu relativního časového zamknu si uzly určují při zakládání nového kanálu a předávají ho protistraně prostřednictvím parametru *to_self_delay* ve zprávách *open_channel* a *accept_channel*.

Při provádění LN plateb by mohla nastat situace, kdy jsou všechny prostředky na straně jednoho uzlu. V tom momentu by druhý uzel nepodstupoval při podvádění žádné riziko, protože jeho výstup by neměl žádnou hodnotu. Aby k takové situaci nedošlo, je při zřizování kanálu určen minimální zůstatek na jeho obou stranách *channel_reserve_satoshis*. Díky tomu potencionální útočník čelí vždy nějakému riziku. [5]

Skriptu uzamykající výstup, který směřuje k vlastníkovi transakce, je definován v dokumentu BOLT #3 a má následující formát:

Výpis 1: Formát scriptPubKey v LN platbě [25]

```
1.  OP_IF
2.      <R_pubkey>
3.  OP_ELSE
4.      <to_self_delay>
5.      OP_CHECKSEQUENCEVERIFY
6.      OP_DROP
7.      <local_pubkey>
8.  OP_ENDIF
9.  OP_CHECKSIG
```

Jedná se o podmíněný skript, což znamená, že výstup může být odemknut, pokud je splněna jedna ze dvou větví. První větev umožňuje utracení výstupu v případě znalosti privátního klíče, který odpovídá veřejnému klíči *<R_pubkey>*. Klíč *<R_pubkey>* je odvozený od odvolávacího veřejného klíče a veřejného klíče protistrany a k jeho utracení je nutné znát jak odvolávací privátní klíč, tak privátní klíč protistrany. Druhá větev skriptu obsahuje relativní časový zámek CHECKSEQUENCEVERIFY s hodnotou *<to_self_delay>*, která zamezí odemknutí této větve po stanovenou dobu od vytěžení. Hodnota *<local_pubkey>* představuje veřejný klíč vlastníka transakce. Nakonec opkód CHECKSIG slouží k validaci podpisu s veřejným klíčem.[25]

Výstup uzamčený skriptem v předchozím formátu lze utratit dvěma způsoby. První způsob je použít transakční vstup s odblokovacím kódem,:

```
<R_signature> 1
```

kde parametr *<R_signature>* obsahuje podpis odpovídající veřejnému klíči *<R_pubkey>*. Hodnota 1 slouží k vybrání první větve skriptu. Tento kód je použit k utracení neplatné závazkové transakce, tedy v případě, že druhý uzel podváděl. Druhý způsob jak utratit výstup je použití vstupu, který obsahuje vlastnost *nSequence* s hodnotou *<to_self_delay>* a odblokovací kód,:

```
<local_signature> 0
```

kde parametr *<local_signature>* obsahuje podpis odpovídající veřejnému klíči *<local_pubkey>*. Tento kód je použit k utracení poslední, a tedy platné závazkové transakce.

Provedením závazkové transakce je automaticky uzavřen kanál a nelze ho nadále používat k LN platbám, protože výstupy fundační transakce jsou tímto utraceny. V případě, že žádná strana při uzavření kanálu nepodvádí, jsou k vytvoření a uzavření potřeba jen dvě transakce zapsané na blockchain. Jedna k vytvoření kanálu a jedna k jeho uzavření. V případě, že jeden z uzlů podvádí a použije k uzavření kanálu některou ze starších závazkových transakcí, je k uzavření kanálu potřeba ještě jedna transakce, která slouží k postihu. [14]

4.3 Routování platby

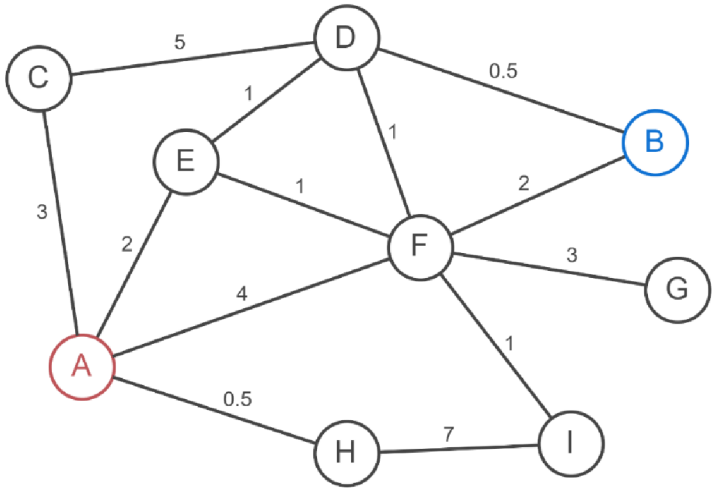
Routování je v Lightning network proces provádění platby mezi uzly, které nejsou přímo propojeny platebním kanálem. K routování je možné využít všechny veřejné kanály v síti. Za využití kanálu k routování si může vlastník kanálu účtovat drobné poplatky, což také funguje jako motivace k otevírání nových veřejných kanálů. Routování se řídí kryptografickým protokolem tak, aby žádný z účastněných uzlů nemohl platbu během routování odcizit.

Lightning network routování používá ke komunikaci mezi uzly takzvaný onion protokol k zajištění soukromí a anonymity účastníků. Díky tomu při routování platby nezná žádný ze zprostředkujících uzlů identitu odesílatele ani příjemce

platby. Zprostředkující uzly znají pouze předcházející uzel, který jim platbu předal a uzel, kterému mají platbu dále poslat. Onion routování je definováno v dokumentu BOLT #4. [22]

Před samotným procesem routování je nutné vyhledat vhodnou cestu od odesílatele k příjemci platby. Pokud žádná cesta neexistuje, není možné platbu provést. Proces hledání cesty nevyžaduje jakoukoliv koordinaci mezi jednotlivými implementacemi, a proto není součástí dokumentů BOLT. Hledání optimální cesty je v Lightning network obtížné vzhledem k tomu, že je v současnosti tvořena více než 35000 uzly, které jsou propojeny více než 85000 kanály. Optimální cesta je tvořena na základě tří kritérií: výše poplatků, likvidita a doba časového zámku.

Lightning network je možné reprezentovat jako neorientovaný graf, jehož vrcholy představují uzly a hrany kanály. Hrany mohou být ohodnoceny na základě výše poplatku daného kanálu případě hodnoty časového zámku. Příklad podoby grafu je znázorněn na Obrázek 19. K hledání optimální cesty je možné využívat algoritmy z teorie grafu jako například Dijkstrův algoritmus a jiné. V případě Lightning network je důležitá také likvidita kanálů, jejíž hodnota však není z bezpečnostních důvodů veřejná, a proto ji nelze využít v grafu. Při hledání cesty většina LN implementací nejprve pomocí grafu a algoritmu získá seznam kandidátních cest,



Obrázek 19: Graf uzlů a kanálů [autor]

které seřadí podle různých kritérií. Následně srze jednotlivé cesty zkouší routovat platbu a v případě, že platba neproběhne, zkouší další cestu ze seznamu. Na základě nepovedených pokusů routování mohou uzly aktualizovat lokální informace o likviditě dané cesty, a tím optimalizovat budoucí hledání cesty. Například v případě routování platby od uzlu A do uzlu B, by odesílatel zkoušel nejprve provést platbu cestou AEDB, která má nejmenší součet poplatků. Pokud by se routování přes přechozí cestu nezdařilo použil by cestu AEDFB a tak dále. [5]

Konstrukce grafu uzlů a kanálů je možná díky takzvanému gossip protokolu, který je popsán v dokumentu BOLT #7. Ten v Lightning network slouží k peer-to-peer sdílení informací o všech uzlech a kanálech v síti. Gossip protokol definuje tři typy zpráv, které mezi sebou uzly vyměňují. Zpráva *node_announcement* slouží k informování ostatních uzlů o vzniku nového uzlu. Zpráva *channel_announcement* informuje ostatní uzly o vzniku nového kanálu a *channel_update* o jeho aktualizaci. Na základě informací v těchto zprávách může jakýkoliv uzel v síti zkonstruovat odpovídající graf. Informace obsažené v jednotlivých zprávách jsou validovány tak, aby graf odpovídal skutečné podobě sítě. [26]

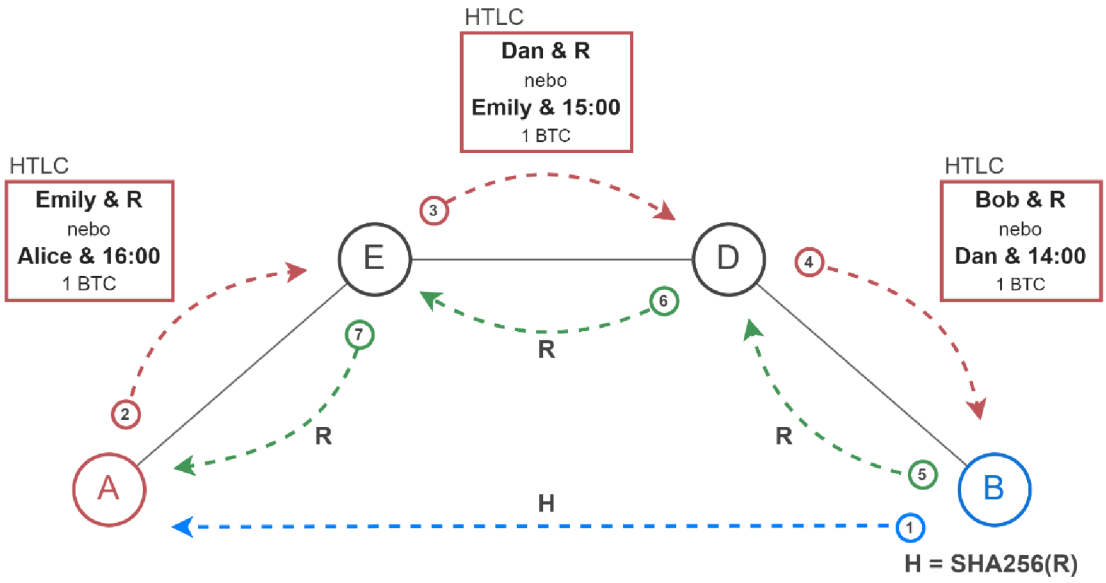
Routování probíhá jako postupné provádění LN plateb skrz jednotlivé kanály, které jsou součástí routovací cesty. Aby nedošlo k odcizení prostředků platby některým z prostředníků, je LN platba reprezentována dalším výstupem závazkové transakce. Tento výstup je uzamčen speciálním typem skriptu s názvem „Hash Time-Lock Contract“, zkráceně HTLC. Skript typu HTLC umožňuje stanovit lhůtu, před jejímž vypršení musí příjemce platby potvrdit její přijetí důkazem o provedení platby. Po vypršení lhůty je nárok příjemce ztracen a platba je navrácena plátci. HTLC výstup může utratit buďto příjemce, který podloží důkaz o provedení platby, nebo odesílatel v případě, že vyprší stanovená lhůta. Jako důkaz o provedení platby se využívá tajný klíč *R*, kterému se také říká *preimage* nebo *secret*. Hodnotu tohoto klíče zná pouze poslední uzel z cesty neboli příjemce routované platby. Zjednodušený HTLC skript by v případě routování platby mezi Alicí a Emily mohl vypadat následovně:

Výpis 2: Zjednodušený HTLC skript [autor]

```
1.  HASH160 <H> EQUAL
2.  IF
3.      <Emily_pubkey> CHECKSIG
4.  ELSE
5.      <locktime> CHECKLOCKTIMEVERIFY
6.      <Alice_pubkey> CHECKSIG
7.  ENDIF
```

Parametr *<H>* představuje hash tajného klíče *R* a *<locktime>* definuje expirační dobu skriptu v absolutní hodnotě. Parametr *<Emily_pubkey>* je veřejný klíč příjemce platby, tedy Emily, a <Alice_pubkey> je veřejný klíč odesílatele platby, tedy Alice. Tento skript může utratit buďto Emily se znalostí tajného klíče R, nebo Alice po expiraci CLTV. HTLC používaný při routování obsahuje také odvolávací privátní klíč, aby při vygenerování nové závazkové transakce byl HTLC skript neplatný.

Prvním krokem routování platby je vygenerování náhodného tajného klíče R přijímatelem platby. Klíč R je následně zahašován a výsledný hash je odeslán odesílateli routové platby. Například v situaci routování platby od Alice k Bobovi za použití cesty AEDB z předchozího přikladu, by Bob vygeneroval náhodný tajný klíč *R* a jeho hash *H* by odeslal Alici. Tato situace je znázorněna na Obrázek 20. Po přijetí hashe H, může Alice vytvořit HTLC výstup do kterého uzamkne prostředky, které chce zaslat Bobovi. Alice následně odešle HTLC Emily, která vygeneruje další HTLC s obdobným *H* a nižší hodnotou časového zámku. Nové HTLC odešle dále Danovi, který opět vygeneruje nové HTLC, které předá Bobovi. Jakmile Bob přijme HTLC od Dana, může použít HTLC a zapsat ho na blockchain, čímž získá prostředky v něm. Provedením transakce by se však uzavřel kanál, a proto namísto provedení HTLC Bob odešle Danovi klíč *R*, jako důkaz o tom, že může HTLC utratit. Dan nyní ví, že Bob může HTLC utratit, a proto se mezi sebou dohodnou a přerozdělí prostředky v kanálu na Bobovu stranu. Dan provede stejný postup s Emily, a ta poté s Alicí. Tímto způsobem jsou přerozděleny prostředky ve všech kanálech a routování platby je úspěšně dokončeno.



Obrázek 20: Routování LN platby [autor]

Důležitou vlastností routování je, že žádný uzel nemůže provést HTLC bez toho, aniž by zveřejnil hodnotu klíče *R*, protože provedením transakce je hodnota R veřejně zapsána na blockchain. Například, pokud by Dan neodeslal hodnotu klíče *R* Emily a namísto toho utratil její HTLC, vystaví tím hodnotu klíče R na blockchain a Emily tak může utratit Alicino HTLC, čímž nikdo nepřijde o žádné prostředky.

Časové zámky jsou součástí HTLC především kvůli vyrovnání prostředků v situaci nedostupnosti některého z uzlů, nebo při odmítnutí spolupráce. Pokud taková situace nastane, časové zámky umožní v momentě jejich vypršení navrácení prostředků odesílateli. V HTLC se používá absolutní časový zámek CHECKLOCKTIMEVERIFY neboli CLTV, který umožňuje uzamčení skriptu do určité doby v čase. Hodnota CLTV je v případě Lightning network definována výškou bloku. Odesílatel tedy může HTLC utratit až transakcí, která bude zapsána do bloku s minimální výškou stanovenou v CTLV.

4.4 Uzly

Uzly představují zařízení využívající peer-to-peer ke komunikaci s ostatními uzly za účelem provádění LN plateb. Uzly vyžadují neustálý přístup k aktuální verzi

blockchain databáze tak, aby mohly kontrolovat Bitcoinové transakce. LN uzly také udržují informace o ostatních veřejných uzlech v síti pro podporu routování plateb. Ke komunikace nejčastěji používají protokol TCP/IP nebo anonymizovaný protokol Onion.

Jednotlivé uzly v síti jsou identifikovány veřejným klíčem spolu se síťovou adresou s portem. Identifikátor je psán ve formátu: *NodeID@Adresa:Port*, kde *NodeID* představuje veřejný klíč uzlu. Hodnota veřejného klíče je odvozena od privátního klíče, který je generován při inicializaci uzlu. Síťová adresa může být buďto ve formátu IP adresy, nebo Onion adresy, podle toho jakou síť uzel používá ke komunikaci s ostatními uzly. V obou případech je součástí síťové adresy port, který je v Lightning Network nastavený na výchozí hodnotu 9735. Uzel však může tuto hodnotu změnit a přijímat síťové požadavky přes jiný port. Identifikátor uzlu může vypadat například takto: [5]

> *028ffdb448b1a54ec3b8f6adc46c842475ea03626ec5d35df6e0343fca214be8b2@gmc3qv2ggczlgpmbvhkv2dcdhcebd2zrhxlm5rsqcp2xettmupn6xdqd.onion:9735*

Identifikátor uzlu je potřebný k otevření kanálu s daným uzlem. Je možné ho získat buďto přímo od provozovatele uzlu, nebo prostřednictvím takzvaných LN prohlížečů. Ty udržují informace o všech veřejných uzlech v síti spolu s jejich statistikami o počtu otevřených kanálů, dostupnosti, stáří, likviditě a dalších. Tyto prohlížeče jsou užitečné především při hledání vhodného uzlu pro kanál.

4.5 Lightning network peněženka

Lightning network peněženka je software, který spravuje uživatelské privátní klíče, sleduje zůstatek účtu a zpracovává LN platby. K zajištění těchto funkcí je součástí peněženky mimo jiné LN uzel, kdy platí, že jeden uzel může využívat vícero peněženek. Většina peněženek je určena pro koncové uživatele, a proto obsahují přívětivé grafické rozhraní. Lightning network peněženka se skládá z následujících komponent:

- úložiště privátních klíčů,
- LN uzel, který komunikuje peer-to-peer s ostatními uzly,
- Bitcoin uzel udržující aktuální blockchain data a komunikující s ostatními Bitcoin uzly,
- databáze udržující informace o uzlech a kanálech v Lightning Network síti,
- správce kanálů umožňující otevírání a uzavírání LN kanálů,
- systém k hledání cesty mezi odesílatelem a příjemcem. [5]

Lightning network peněženky mohou namísto jedné nebo více komponent používat služby třetích stran, které tyto funkce zprostředkovávají. Peněženky, které obsahující všechny komponenty a nejsou závislé na žádné třetí straně, jsou označovány jako úplné Lightning network peněženky. Peněženky se také dělí na *custodial* a *noncustodial* podle toho, kde se nachází privátní klíče uživatelů. V případě *custodial* peněženky jsou klíče uloženy u třetí strany a uživatel této peněženky musí důvěřovat této třetí straně. Oproti tomu *noncustodial* peněženky drží klíče přímo u uživatele, který k nim má přístup jako jediný. Nicméně i noncustodial peněženka může jednu nebo více částí peněženky, mimo správu klíčů, outsourcovat na důvěryhodnou třetí stranu.

Lightning network peněženky mohou být nainstalovány na různá zařízení jako notebooky, servery, desktopové počítače a mobilní zařízení. Úplné Lightning network peněženky mohou být nainstalovány pouze na servery, případně desktopové počítače, především kvůli nutnosti stálého připojení k internetu a velké kapacitní zátěži. [5]

Vzhledem k tomu, že jsou LN peněženky určeny koncovým uživatelům jsou navrženy tak, aby byly jednoduše použitelné a přehledné. Proto většina LN peněženek obsahuje tlačítka *Odeslat* a *Přijmout*, které slouží k jednoduchému provádění LN plateb. Uživatel je tak odkloněn od složitých procesů probíhajících na pozadí. LN peněženky k lepší použitelnosti používají QR kódy ke sdílení platby. Z principu fungování LN plateb není možné odesílatelem odeslat platbu bez kooperace příjemce, ten musí vždy platbu iniciovat. Platba tedy probíhá tak, že

příjemce zvolí volbu *Přijmout,* čímž se vygeneruje nová platba neboli LN faktura, kterou je nutné odeslat odesílateli k zaplacení. LN faktura je zakódována do řetězce, který je určen ke sdílení odesílateli. Zakódovaná LN faktura může vypadat následovně:

```
lnbc5250n1p39h5d2pp54nd22a0j6zvsn8602p60k8zsum32z8kr5mnzte24hmpxf00
4etvqdqqxqyjw5q9qtzqqqqqq9qsqsp5sta74l6n9fn3hzgudf9avm683evm2myn4jx
ll3rj20wk3ta2thwqrzjqwryaup9lh50kkranzgcdnn2fgvx390wgj5jd07rwr3vxej
e0glcllem4lzeenewhcqqqqlgqqqqqeqqjq8csjuuk3k0gpyxt3j7dpfq0r05r52hmy
apy26w5yaav0zmt9ntn8js4eq3lrmsvvyqr484f8jreggcuedlagx0t44hwxucljarw
f2fsqh4wnfs
```

Struktura zakódované LN faktury je definována v dokumentu BOLT #11. Řetězec vždy začíná čitelnou částí, která obsahuje prefix *lnbc* následovaný částkou LN faktury s násobitelem. Násobitel je definován jedním písmenem a určuje hodnotu, kterou je částka vynásobena . Násobitel může nabývat hodnoty *m* (mili), *u* (mikro), *n* (nano), nebo *p* (piko). V příkladu výše je částka 5250 následována násobitelem *n*, tedy nano. Hodnota LN faktury se tedy vypočítá jako $5250 * 10^{-9}$ což je 0,00000525 bitcoinů neboli 525 satoshi. Zakódovaná LN faktura dále obsahuje časové razítko, množinu otagovaných informací a podpis. Každá otagovaná informace se skládá z tagu, velikosti a hodnoty. Tag určuje sémantický význam dané hodnoty. Otagované informace obsahují několik povinných údajů jako například hash tajného klíče R. [23]

Za účelem přívětivějšího sdílení zakódované LN faktury využívají LN peněženky zpravidla QR kódy. QR kód je generován při vytváření LN faktury a následně sdílen odesílateli, který jej může jednoduše oskenovat. Na Obrázek 21 je zobrazen QR kód odpovídající výše zakódované LN faktuře.

Obrázek 21: Ukázka LN faktury v podobě QR kódu [autor]

Odesílatel platby tento QR kód oskenuje svojí LN peněženkou, která dekóduje informace o faktuře a po potvrzení od uživatele provede LN platbu mezi peněženkou odesílatele a příjemce. Při provádění LN platby musí být obě strany online a v případě, že je například příjemce platby nedostupný, není možné platbu provést.

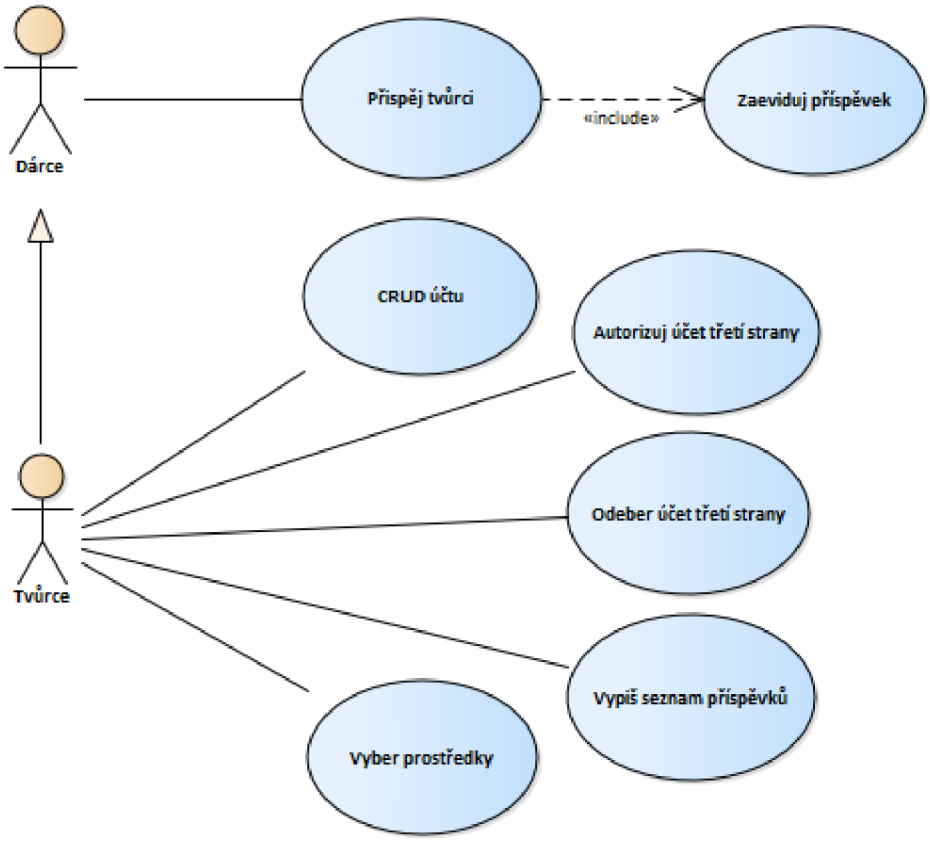
5 Analýza a návrh řešení

Tato kapitola se zabývá analýzou a návrhem vyvíjené platformy. Nejprve jsou definovány požadavky, na základě kterých je dále navrženo řešení platformy včetně datového modelu a fyzické architektury.

5.1 Požadavky

Vyvíjená platforma (dále jen aplikace) má za cíl poskytnout možnost přispívání internetovým tvůrcům prostřednictvím sítě Lightning network (dále také jako LN). Jako internetový tvůrce je označován uživatel vytvářející libovolný obsah na internetu na různých platformách třetích stran (dále jen platformy) jako například YouTube, Medium, StackExchange a dalších. Řešení by mělo být z pohledu uživatele přímo integrováno do zmíněných platforem tak, aby bylo co nejvíce uživatelsky přívětivé, jednoduché a intuitivní.

Základní požadavky na aplikaci jsou reprezentovány jednoduchým diagramem případů užití na Obrázek 22. Součástí systému jsou dva typy uživatelů, a to dárce a tvůrce. Dárce má pouze jediný případ užití a tím je přispění tvůrci. Mezi případy užití tvůrce patří mimo přispění jiným tvůrcům také autorizace a odebrání účtu třetí strany, vypsání seznamu příspěvků, výběr prostředků získaných příspěvky a provádění CRUD operací jeho účtu.



Obrázek 22: Diagram případů užití [autor]

V případě, že tvůrce chce přispět jinému tvůrci, automaticky se stává aktérem typu dárce. Scénář případu užití *Přispěj tvůrci* vypadá následovně:

1. Dárce iniciuje přispění Tvůrci
2. Systém vyhledá Tvůrce
3. Pokud je Tvůrce zaregistrovaný v aplikaci
   3.1. Systém zaeviduje příspěvek
   3.2. Systém zobrazí QR kód s LN platbou
   3.3. Dárce oskenuje QR kód libovolnou LN peněženkou
   3.4. Systém sleduje stav LN platby
   3.5. Pokud LN platba proběhla
      3.5.1. Systém zaeviduje příspěvek a přičte jeho hodnotu na virtuální peněženku Tvůrce
      3.5.2. Systém zobrazí hlášku Dárci o proběhlém příspěvku

3.6. Pokud LN platba neproběhla

3.6.1. Systém označí příspěvek jako neproběhlý

4. Pokud Tvůrce není zaregistrovaný v aplikaci

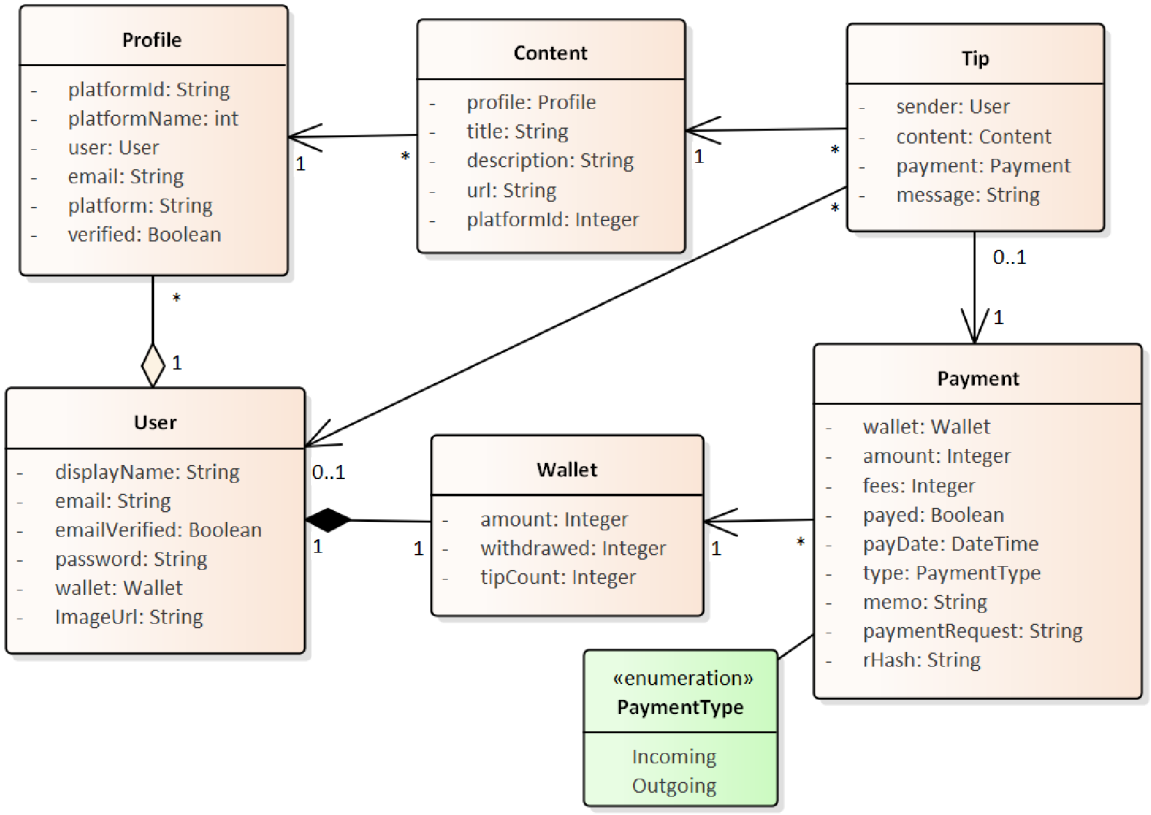
4.1. Sytém zobrazí hlášku Dárci, že není možné provést příspěvek

Současně se splněním funkčních požadavků, které byly vymezeny v rámci diagramu případu užití, musí aplikace splňovat i několik non-funkčních požadavků:

- Aplikace musí poskytovat možnost přispívání nepřetržitě,
- aplikace musí být uživatelsky přívětivá a jednoduchá na používání,
- aplikace musí poskytovat možnost přispívání i bez registrace uživatele, tedy anonymně.

5.2 Datový model

Datový model představuje návrh objektové struktury dat definované množinou tříd. Na Obrázek 23 jsou tyto třídy zobrazeny pomocí diagramu tříd, který popisuje jejich vztahy a vlastnosti. Diagram obsahuje třídy: *User* (uživatel), *Profile* (profil), *Content* (obsah), *Tip* (příspěvek), *Payment* (platba) a *Wallet* (peněženka).



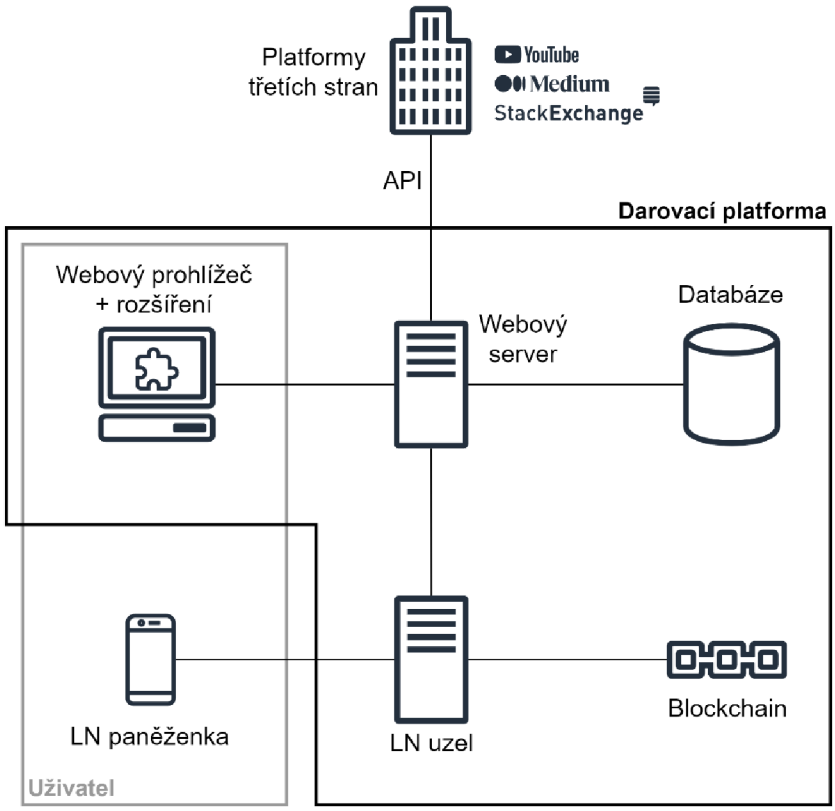
Obrázek 23: Datový model aplikace [autor]

Třída *User* představuje registrovaného uživatele na vyvíjené platformě. Může se jednat jak o tvůrce, tak dárce. V případě tvůrce jsou pak uchovávány informace o připojených účtech třetích stran, které se za účelem odlišení nazývají profily. Jeden účet může obsahovat libovolný počet profilů, aby mohl tvůrce přijímat dary z více platforem na jediný účet. Každému uživateli je vytvořena virtuální peněženka definovaná třídou *Wallet*, která slouží k evidování celkové výše přijatých příspěvků. Třída *Payment* pak představuje LN platbu, která může sloužit jako příchozí platba k připsání prostředků na virtuální peněženku tvůrce nebo jako odchozí platba k vyplácení nashromážděných prostředků z virtuální peněženky. Samotný příspěvek pak představuje třída *Tip*, která eviduje zprávu pro tvůrce, dárce, LN platbu a internetový obsah, který chce dárce svým příspěvkem ocenit. Internetový obsah pak definuje třída *Content* a může představovat libovolné dílo veřejně

přístupné na internetu, čímž může být například video, fotografie, článek či jiný typ tvorby.

5.3 Architektura

Fyzická architektura vyvíjené platformy je zobrazena na Obrázek 24. Ta mimo jiné znázorňuje propojení mezi uživatelem a platformami třetích stran. Stěžejní částí architektury je webová aplikace běžící na webovém serveru, která na základě požadavků přijatých od uživatele provádí všechny potřebné operace. Webová aplikace přímo komunikuje s LN uzlem, a díky tomu může sledovat stav LN plateb, vytvářet LN faktury nebo odesílat LN platby. Webová aplikace dále zprostředkovává autorizaci účtů třetích stran za použití jejich API. Nakonec webová aplikace ukládá všechny potřebné informace na databázi.



Obrázek 24: Návrh fyzické architektury platformy [autor]

Důležitou částí vyvíjené platformy je LN uzel, který umožňuje provádění plateb přes Lightning network. Při provádění LN plateb je nutné, aby obě strany byly v dané chvíli online. Většina běžných uživatelů však používá LN peněženky ve formě mobilních aplikací, které jsou online pouze v moment používání peněženky uživatelem. V momentě nedostupnosti LN peněženky tvůrce by pak dárce nemohl provést LN platbu a přispět tak danému tvůrci. Právě kvůli této vlastnosti LN plateb je nutné, aby aplikace používala vlastní LN uzel. Ten bude online nepřetržitě a veškeré LN platby budou odesílány právě na tento uzel. Dárce tedy může přispívat nezávisle na stavu LN peněženky tvůrce. To však znamená, že prostředky odeslané dárcem nejsou přímo odeslány tvůrci, ale na LN uzel. Proto je každý příspěvek evidován do databáze a jeho hodnota připsána na virtuální peněženku daného tvůrce, která je vedena v rámci vyvíjené platformy. Tvůrci je kdykoliv umožněno provézt výběr nashromážděných příspěvků z LN uzlu na jeho soukromou LN peněženku.

Dalším důvodem pro použití vlastního LN uzlu je fakt, že před každou LN platbou musí nejprve příjemce vygenerovat LN fakturu. V případě že by byli platby prováděny přímo mezi dárcem a tvůrcem, tak by tedy musel tvůrce před každým příspěvkem vygenerovat novou LN fakturu. Kvůli tomu by aplikace musela komunikovat se všemi peněženkami případně uzly tvůrců za účelem vytváření těchto LN faktur. To je však v dnešní době prakticky nemožné vzhledem k velkému množství různých peněženek, kde většina z nich ani neimplementuje žádné rozhraní kvůli bezpečnosti uživatele.

Architektura znázorňuje také uživatelskou část, která sestává z LN peněženky a webového prohlížeče. LN peněženka slouží dárci k zasílání příspěvků a tvůrci k výběru nashromážděných prostředků. Prostřednictvím webového prohlížeče uživatel komunikuje s webovou aplikací a iniciuje všechny potřebné operace. Pro jednodušší darování příspěvků obsahuje webový prohlížeč na straně dárce rozšíření, které slouží k integraci tlačítka pro přispívání přímo do platforem třetí strany. Běžně je k integrování vizuálního prvku do platformy potřebný přístup k

jejím zdrojovým kódům. Rozšíření webového prohlížeče však umožňuje úpravu chování prohlížeče včetně grafického rozhraní a zároveň je podporováno většinou známých webových prohlížečů jako Google Chrome, Mozilla Firefox, Safari a dalšími. Nevýhodou tohoto řešení je nutnost implementace rozšíření pro každý prohlížeč zvlášť vzhledem k jejich odlišným požadavkům.

5.3.1 Lightning network uzel

Lightning network uzel je software, který umožňuje provádět LN platby s ostatními uzly v síti. Jak již bylo zmíněno v teoretické části práce, LN uzel musí ke správnému fungování mít neustálý přístup k aktuální verzi blockchainu a být nepřetržitě dostupný v síti. Většina implementací LN uzlu zároveň implementují také funkcionality LN peněženek. Obsahují tak citlivá data jako privátní klíče, proto je důležité, aby LN uzel běžel v bezpečném prostředí.

Existuje řada řešení poskytujících virtuální LN uzly jako službu. Tato řešení jsou sice nejjednodušší na nasazení, ale zároveň nejméně bezpečné, protože nejsou spravovány lokálně a nelze zajistit, aby k nim neměl přístup nikdo jiný. Proto navrhovaná darovací platforma využívá LN uzel běžící na vlastním zařízení v privátní síti.

Přístup k aktuální databázi blockchain je možné zajistit několika způsoby. První způsob je připojit LN uzel k Bitcoinovému úplnému uzlu, který udržuje kompletní blockchain a může LN uzlu potřebná data z blockchainu poskytnout. Pokud je Bitcoinový uzel spravovaný stejnou entitou jako LN uzel, jedná se o nejbezpečnější způsob získávání dat z blockchainu bez nutnosti důvěry třetí straně. V případě, že se jedná o cizí Bitcoinový uzel, nelze ověřit validitu poskytovaných dat a je tedy nutná důvěra v provozovatele uzlu. Alternativním způsobem je využití protokolu SPV nebo jeho vylepšené verze Neutrino, který umožňuje získat ověřitelné informace o transakcích v blockchainu přímo od těžařů. Použitím jednoho z protokolů je však redukován počet uzlů v síti, která se tímto stává více centralizovanou. Proto je tento způsob vhodný pouze v případě, že není možná implementace vlastního

Bitcoinového uzlu. Navíc Neutrino protokol nadměrně zatěžuje internetovou síť při komunikaci s těžaři.

Vyvíjená aplikace bude k přístupu na blockchain využívat vlastní Bitcoinový úplný uzel. Toto řešení je nejbezpečnější, jelikož nevyžaduje žádnou důvěru v třetí stranu. Zařízení, na kterém běží Bitcoinový úplný uzel, musí poskytovat dostatečnou kapacitu k uložení celého blockchainu, který v dnešní době přesahuje velikost 400 GB a každým dnem se zvětšuje. Blockchain musí být zároveň průběžně aktualizován, a proto je vhodné, aby bylo zařízení nepřetržitě zapnuté a připojené k síti.

Implementace Lightning uzlu a Bitcoinového úplného uzlu poběží na jediném zařízení, které musí:

- Poskytovat úložiště s minimální kapacitou 500 GB k uložení celého blockchainu,
- být dostatečně výkonný k ověřování Bitcoinových transakcí,
- být připojen stabilním připojením k internetové síti,
- být nepřetržitě v provozu.

K implementaci Bitcoinového úplného uzlu existují pouze dva softwary, a to *Bitcoin Core* a *btcd*. Bitcoin Core je nejběžněji používaný software, který byl zpočátku vyvíjen Satoshi Nakamotem. Jedná se o open-source software psaný převážně v jazyce C++. Jediná známá alternativa k *Bitcoin Core* je software s názvem *btcd*, který je také open-source a je psaný v jazyce Go. Projekt *btcd* vznikl za účelem separování blockchainových funkcionalit uzlu a Bitcoinové peněženky, které například *Bitcoin Core* implementuje společně. Oddělení těchto funkcionalit je vhodně v případě, že uzel pracuje s více peněženkami. Uzel, který bude součástí vyvíjené aplikace bude používat *Bitcoin Core*, a to především proto, že tento software určuje standard celé sítě a při použití jiné implementace není do budoucna jistá její kompatibilita s ostatními nástroji.

Pro Lightning network uzel existuje celá řada možných softwarů implementující LN protokol. Mezi implementace, které splňují všechny BOLT specifikace patří:

Lightning Network Deamon (*LND*), *Core Lightning* (*CLN*), *Eclair* a *Rust-Lightning*. Kromě *Rust-Lightning*, který je spíše zaměřený na integraci vlastních funkcí do LN uzlu, se od sebe jednotlivé implementace příliš neliší a k běžnému používání lze tedy využít libovolnou z nich. LN uzel, který je součástí vyvíjené platformy používá implementaci *LND,* a to především kvůli podpoře vzdáleného volání funkcí přes protokol gRPC, který umožňuje efektivně komunikovat s LN uzlem z libovolného běhového prostředí.

6 Implementace

Tato kapitola obsahuje ukázkovou implementaci navrženého řešení, která sestává z několika částí. Každé části implementace se věnuje samostatná podkapitola. První popisuje implementaci rozšíření pro vybraný webový prohlížeč Google Chrome. Následující podkapitola je věnována implementaci Lightning network uzlu. Nakonec je popsána implementace webové aplikace, které je věnována poslední podkapitola.

6.1 Implementace rozšíření pro Google Chrome

Implementované rozšíření má za cíl integraci vizuálního prvku do platforem třetích stran pro jednoduchou podporu přispívání. V tomto případě se jedná o platformy YouTube, Medium a StackExchange.

Rozšíření webového prohlížeče je obecně software sloužící k rozšíření či upravení funkcionality prohlížeče. Rozšíření také umožňuje přistupovat a upravovat webový obsah jednotlivých stránek. Google Chrome rozšíření se skládá z několika komponent:

- Manifest soubor – specifikuje základní vlastnosti rozšíření,
- backgroun skripty – spouští se při vybrané události a běží v odděleném prostředí na pozadí webu,
- obsahové skripty – spouští se v kontextu webových stránek, a mohou tak přistupovat k jeho prvkům pomocí DOM,
- stránka s nastavením rozšíření,
- uživatelské rozhraní – libovolné grafické rozhraní zobrazující se jako popup okno, kontextové menu, ikona v horní liště či omnibox.

Jednotlivé komponenty jsou vytvořeny pomocí webových vývojových technologií HTLM, CSS a JavaScript (JS).

Implementované rozšíření je tvořeno ze všech zmíněných komponent kromě stránky s nastavením. Jediná povinná komponenta každého rozšíření je manifest soubor, který definuje základní vlastnosti rozšíření jako název, popis, ikony, oprávnění a cesty k souborům ostatních komponent. Ukázka manifest souboru vyvíjeného rozšíření je ve Výpis 3. V první řadě definuje cestu k background skriptu s názvem *background.js*. Vlastnost *permission* specifikuje výčet API ke kterým bude mít rozšíření oprávnění. V tomto případě má rozšíření přístup k API *tabs*, které umožňuje práci s kartami prohlížeče a API *scripting*, které slouží k injektování JS a CSS souborů do stránek. Hodnota vlastnosti host_permission určuje že rozšíření bude aktivní na všech stránkách s https protokolem. Nakonec ukázkový manifest obsahuje cesty k ikonám rozšíření.

Výpis 3: Ukázka manifest souboru webového rozšíření [autor]

```
1.  {
2.     ...
3.       "background": {
4.         "service_worker": "background.js"
5.       },
6.       "permissions": [ "tabs", "scripting" ],
7.       "host_permissions": [ "https://*/*" ],
8.       "icons": {
9.         "16": "/icons/16.png",
10.        "32": ...
11.      },
12.    ...
13. }
```

Hlavní komponentou vyvíjeného rozšíření je background skript, který slouží k detekci stránek třetí platformy a následně injektuje do stránky odpovídající obsahový skript. K detekci stránky využívá skript tabs API, konkrétně funkci *tabs.onUpdated.addListener*, která je volána při každé změně libovolné karty prohlížeče. Při každé změně karty poskytuje API informace jako titulek, URL stránky, URL favicony a mnoho dalších. Tyto informace pak skriptům slouží ke zjištění, o jakou platformu se jedná a na základě toho injektovat správný obsahový skript odpovídající dané platformě. Pokud uživatel navštíví stránku, která nepatří

mezi zmíněné tři podporované platformy, rozšíření neprovádí žádné další operace. Pokud se jedná o stránky v rámci platformy YouTube a StackExchange, pak tyto stránky skript identifikují jednoduše podle URL. V případě stránek Medium se nelze řídit pouze podle URL stránky, protože tyto weby mohou mít URL libovolné. V případě těchto stránek se k identifikaci využívá URL favicony, které vždy obsahují doménu druhé úrovně *medium* následovanou doménou nejvyššího řádu *.com*.

Po identifikaci platformy je do stránky injektován CSS soubor a obsahový skript odpovídající dané platformě. Pro každou platformu rozšíření definuje samostatné CSS i obsahové skripty, aby byl zajištěn individuální vzhled, pozice a chování tlačítka s ohledem na design a strukturu stránky. U každé platformy vyvolává tlačítko stejnou událost, a tou je otevření nového okna prohlížeče s vygenerovanou LN platbou sloužící k přispění tvůrci obsahu dané platformy.

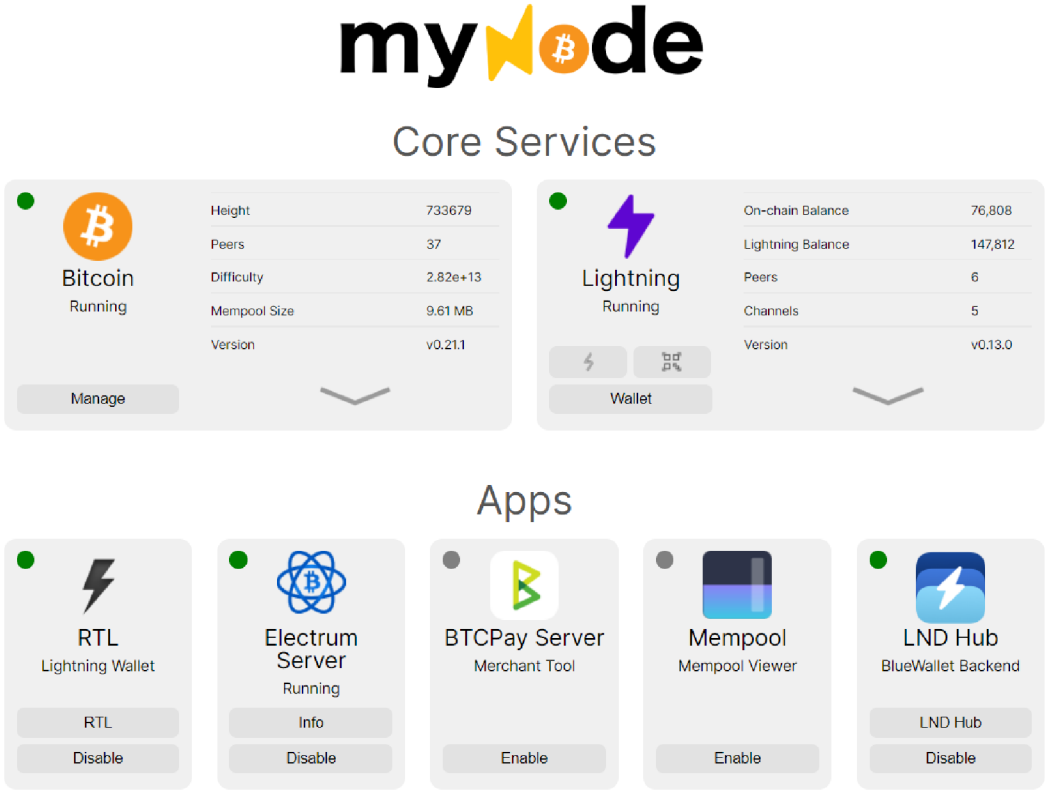
6.2 Implementace Lightning network uzlu

V této kapitole je popsána implementace Lightning network uzlu. V první části kapitoly je vybrán vhodný operační systém pro zařízení, na kterém poběží implementace Lightning network uzlu. Na základě vybraného operačního je navržen hardware zařízení, a nakonec je popsán proces otevírání LN kanálů.

6.2.1 Operační systém

K jednodušší implementaci Lightning network uzlu existuje několik distribucí linuxového operačního systému umožňujících jednoduchou správu softwarů týkajících se Lightning network a Bitcoinu. Tyto operační systémy automaticky nainstalují a propojí implementaci Lightning network uzlu a Bitcoin Core s minimální nutností konfigurace. Navíc obsahují sadu doplňkových nástrojů rozšiřujících funkcionalitu uzlu. Mezi tyto operační systémy patří: *Umbrel*, *MyNode*, *RaspiBlitz*, *RaspiBolt* a *EmbassyOS*.

Implementovaný LN uzel používá distribuci linuxového operačního systému *MyNode*, který obsahuje implementaci Lightning uzlu *LND* a implementaci Bitcoinového úplného uzlu *BitcoinCore*. Mimo tyto dva softwary distribuce obsahuje desítky volitelných aplikací k lepší správě a monitorování LN uzlu i Bitcoinového blockchainu. Jednou z volitelných aplikací je nástroj *Ride The Lightning* (*RTL*), který umožňuje provádění LN operací přes webové rozhraní jako otevírání kanálů, odesílání a přijímání LN plateb a tak podobně. *MyNode* umožňuje správu všech aplikací a nastavení LN uzlu prostřednictvím jednoduchého webového rozhraní, které je zobrazeno na Obrázek 25.



Obrázek 25: Webové rozhraní MyNode [autor]

6.2.2 Hardware

Vybraná distribuce operačního systému podporuje několik typů zařízení, a to *Raspberry Pi 4*, *RockPro64*, *Rock Pi 4* a virtuální PC. Doporučené požadavky na hardware jsou SSD disk o minimální velikosti 1 TB a paměť RAM s minimální kapacitou 2 GB.

V rámci vyvíjeného řešení je použito zařízení *Raspberry Pi 4* model *B*. Jedná se o malý jednodeskový počítač s čtyř-jádrovým procesorem o výkonu 1.5 GHz a RAM pamětí s kapacitou 4 GB. K počítači je přes konektor typu USB 3.0 připojen SSD disk s kapacitou 1 TB, který by měl zaručit dostatečnou rychlost, a především kapacitu k uložení celého Bitcoinového blockchainu. Operační systém sám o sobě je uložen na SD kartě o velikosti 32 GB. Nakonec je počítač vybaven síťovým konektorem RJ-45 pro stabilní připojení k internetové síti.

LN uzel obsahuje citlivá data jako privátní klíče, které mohou být v případě odcizení použita k získání prostředků uzamčených v Lightning network síti. Proto je důležité uchovávat zařízení neveřejné, a to jak fyzicky, tak i z pohledu sítě. Zařízení je tedy součástí zabezpečené privátní sítě a je fyzicky umístěno v neveřejných prostorách. Kromě těchto opatření je privátní sít i zařízení samotné zabezpečeno silnými hesly.

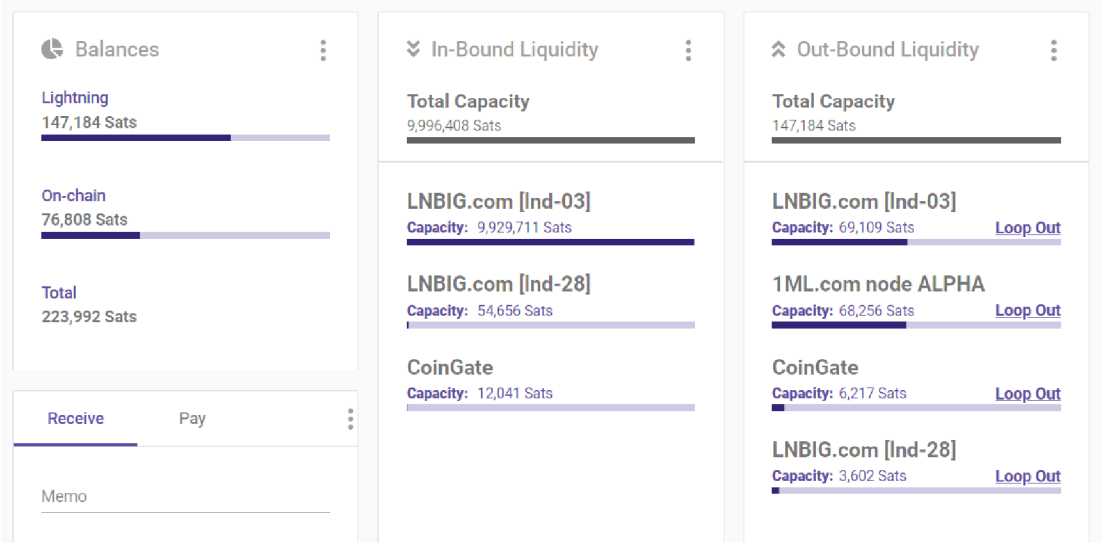
6.2.3 Lightning network kanály

Po inicializaci uzlu je potřeba otevřít LN kanály, aby bylo možné provádět LN platby. Ke správě kanálů obsahuje uzel nástroj RTL, který za tímto účelem poskytuje jednoduché webové rozhraní. Před otevřením kanálu je nutné převézt libovolné množství bitcoinů na Bitcoinovou peněženku, která byla vytvořena při inicializaci LN uzlu. Převedené prostředky je pak možné uzamknout do LN kanálu.

Před otevřením kanálu samotného je potřeba nalézt vhodný uzel, se kterým bude kanál otevřen. Takový uzel by měl mít již otevřené kanály s jinými uzly, aby mohl

být využit k routování LN platby, pokud možno do celé sítě. Dalším aspektem je likvidita kanálů uzlu, tedy množství uzamčené hodnoty, kterou je možné skrz kanály poslat. K výběru vhodného uzlu pro otevření kanálu se využívají Lightning network prohlížeče jako například *1ML*, *Amboss* a *ACINQ explorer*. Tyto prohlížeče umožňují získat informace o všech veřejných LN uzlech v síti, a také u každého uzlu sledují data jako otevřené kanály, likvidita, stáří, dostupnost, růst a tak podobně.

Prostřednictvím nástroje RTL, který je zobrazen na Obrázek 266, byly postupně zřízeny celkem čtyři kanály s různými uzly v různé hodnotě. Problém u těchto nových kanálu je ten, že mají všechny prostředky na straně implementovaného uzlu. Tyto kanály je tedy možné využít pouze k placení ostatním účastníkům sítě. Kvůli tomu uzel není schopen plnit hlavní účel, kterým je přijímání příspěvků od dárců.



Obrázek 26: Nástroj RTL pro správu LN uzlu [autor]

Problém nedostatku příchozí likvidity je možné řešit několika způsoby. Tím nejjednodušším je provádět LN platby přes otevřené kanály, čímž se prostředky v kanálech přesunou na druhou stranu. Příjemcem těchto plateb může být například i další z vlastních LN peněženek. Další možností je využití služby sloužící k převodu prostředků z Lightning network sítě na Bitcoinovou síť. Tyto služby jsou zpoplatněné, jelikož musí hradit transakční poplatek těžařům za provedení

Bitcoinové transakce. V rámci těchto služeb je možné odeslat platbu na cizí uzel, který po jejím přijetí odešle Bitcoinovou transakci ve stejné hodnotě zpět. Tímto způsobem se převáží prostředky v kanálu na druhou stranu a zároveň jsou prostředky vyplaceny zpět na Bitcoinovou peněženku, tedy mimo LN síť. Všechny předchozí řešení mají tu nevýhodu, že k získání určité příchozí likvidity je nutné vlastnit danou sumu prostředků. Pokud například uzel potřebuje přijmout LN platby v celkové hodnotě 1 BTC, musí nejprve 1 BTC sám vlastnit a uzamknout ho do kanálu. Proto zde existuje možnost si příchozí kapacitu koupit. Tuto možnost nabízí celá řada uzlů disponující velkou likviditou.

K navýšení příchozí likvidity byla využita možnost nákupu kanálu od společnosti LNBIG. Tato společnost vlastní desítky LN uzlů, které mají otevřených tisíce kanálů napříč celou sítí. Nově otevřený kanál má příchozí likviditu v hodnotě 0.1 BTC a díky tomu lze pohodlně přijímat velké množství drobných LN plateb.

6.3 Implementace webová aplikace

Webová aplikace je hlavní komponentou celé darovací platformy. Je tvořena ze dvou vrstev, a to klientskou (front-end) a serverovou (back-end).

6.3.1 Použité technologie

Vue.js

Vue.js je front-end framework sloužící ke tvorbě webového uživatelského rozhraní a jednostránkových webových aplikací a to pomocí jazyka JavaScript. Framework se zaměřuje především na zobrazovací vrstvu aplikace, také podporuje různá rozšíření sloužících k směrování, správu dat nebo podpůrné nástroje k vývoji.

Node.js

Node.Js je multiplatformní běhové prostředí umožňující používat programovací jazyk JavaScript na serverové vrstvě řešení. Díky tomu je možné při vývoji webových aplikací používat pouze jeden programovací jazyk, a to jak na front-end, tak na

back-end. Node.js je v implementaci využíván pro tvorbu serverové vrstvy poskytující veřejné rozhraní (API) s architekturou REST.

NPM

NPM je software pro správu balíčků neboli knihoven pro JavaScript. Ve vyvíjené aplikaci je použit jak v klientské vrstvě, tak serverové.

REST API

REST je datově orientovaná architektura veřejného rozhraní (API), která vymezuje několik principů a omezení jako jednotnost rozhraní, identifikaci zdrojů, separace klienta a serveru, úplnost dat a další. REST API není vázán na konkrétní komunikační protokol, ale ve většině případů se používá protokol HTTP.

gRPC

Google Remote Procedure Call neboli gRPC je open-source framework pro vzdálené volání procedur. gRPC je zaměřený především na vysokou rychlost spolu s nezávislostí na běhovém prostředí a podporou většiny známých programovacích jazyků. Tato technologie se hojně využívá k efektivnímu propojování takzvaných mikroslužeb. Ve vyvíjené platformě se gRPC používá ke komunikaci mezi LN uzlem (LND) a serverovou vrstvou webové aplikace.

MySQL

MySQL je populární open-source systém pro správu relačních databází. Hlavním účelem MySQL je efektivní ukládání a spravování dat v strukturovaných a provázaných tabulkách. Tato technologie je využita na serverové vrstvě webové aplikace k ukládání všech potřebných dat o uživatelích, příspěvcích, platbách a tak podobně.

SSH tunel

SSH tunel nebo také SSH Port Forwarding je mechanismus k směrování portů mezi klientem a serverem. Jedná se o bezpečný komunikační kanál využívající definovaný

port. SSH tunel je ve vyvíjené platformě použit k šifrované komunikaci mezi LN uzlem a webovým serverem.

6.3.2 Webové uživatelské rozhraní

Webové uživatelské rozhraní (dále jen web) sloužící k uživatelsky přívětivé komunikaci se serverem je implementováno za použití frameworku Vue.js. Základními kameny Vue.js webů jsou pohledy představující jednotlivé stránky. Jednotlivé pohledy se pak skládají z komponent, které definují libovolně velké části stránek. Pohledy i komponenty obsahují vizuální část definovanou pomocí HTML a CSS a funkční část definovanou pomocí jazyka JavaScript. Komponenty často slouží k zobrazení vybraných dat, a proto jsou také takzvaně reaktivní, což znamená, že při změně dat se automaticky změna projeví i vizuálně v komponentě.

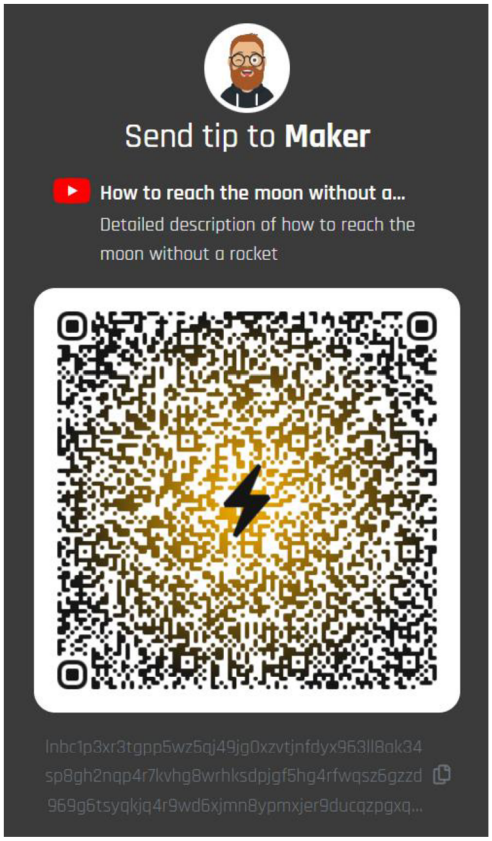
Web je pomyslně rozdělen na dvě části, a to na veřejně přístupnou a dashboard pro přihlášeného uživatele. Veřejně přístupná část se skládá ze stránek:

- *Home* – úvodní stránka se základními informacemi,
- *Register* – stránka sloužící k registraci nového tvůrce,
- *Login* – stránka umožňující přihlášení se do dashboardu, a to jak prostřednictvím emailu a hesla, tak pomocí autorizace přes platformy YouTube nebo StackExchange,
- *Tip* – stránka sloužící k přispívání tvůrcům.

Stěžejní stránkou v této části webu je stránka *Tip*, která slouží přispívání a je přístupná veřejně bez nutnosti přihlášení. Tato stránka obsahuje dva povinné parametry, které slouží k identifikaci tvůrce a obsahu, na který chce dárce přispět. První parametr je *platform,* který určuje, o jakou platformu se jedná. Druhým parametrem je *ID* jednoznačně identifikující obsah dané platformy. Podle hodnoty parametru *ID* je také identifikován autor obsahu. Pokud je autor zaregistrovaný v aplikaci, je vygenerována a zobrazena LN platba k přispění tvůrci. URL stránky určené k přispění tvůrci YouTube videa by mohla vypadat například takto:

```
/tip?id=HSdQD-fpDOs&platform=youtube
```

Generování URL se správně vyplněnými parametry má na starosti rozšíření webového prohlížeče. Ukázka stránky *Tip* k přispění tvůrci za YouTube video je na Obrázek 277. Stránka mimo LN platbu obsahuje také jméno, obrázek tvůrce, ikonu platformy a název s popisem obsahu, v tomto případě videa.



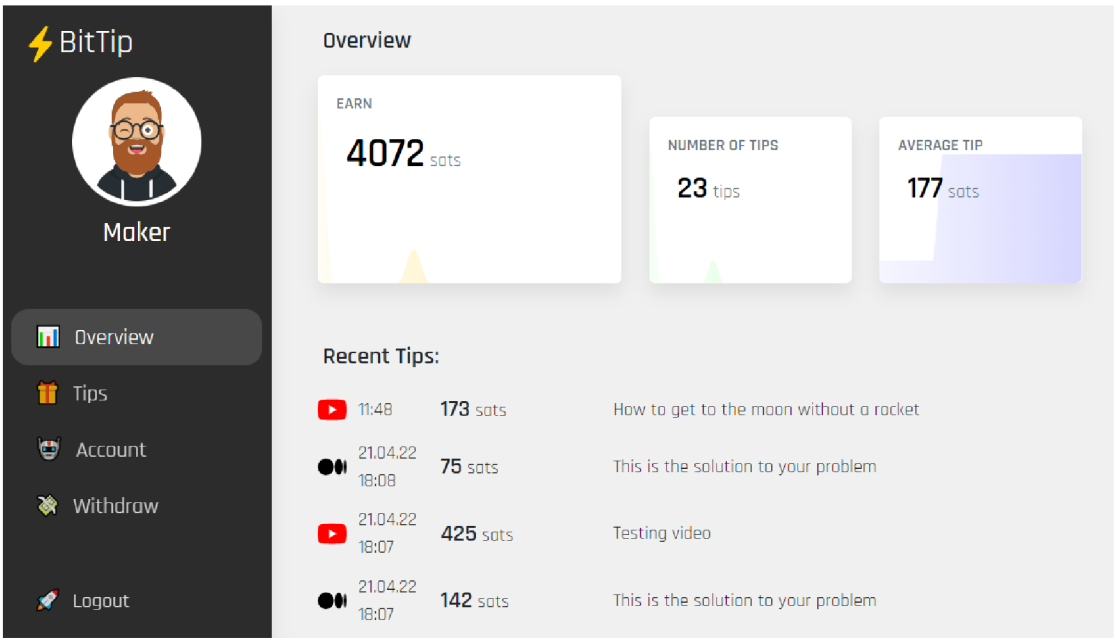
Obrázek 27: Ukázka stránky Tip s příspěvkem za YouTube video [autor]

Druhou část webu tvoří stránky, které obsahují informace určené pro přihlášeného tvůrce:

- *Overview* – úvodní stránka dashboardu, která obsahuje celkový součet hodnoty přijatých příspěvků spolu se seznamem nedávných příspěvků,
- *Tips* – stránka obsahující seznam všech přijatých příspěvků,
- *Account* – stránka s nastavením účtu s možností připojení, či odpojení od účtu platformy třetí strany,

- *Withdraw* – stránka umožňující výběr prostředků z virtuální peněženky.

Na Obrázek 28 je zobrazena ukázka stránky *Overview*, která především ukazuje celkovou hodnotu přijatých příspěvků, dále jejich celkový počet a průměrnou výši. V dolní části stránky se nachází seznam s několika nejnovějšími příspěvky. Stránka také obsahuje boční menu, které slouží k přepínání mezi jednotlivými stránkami dashboardu a v neposlední řadě zde najdeme také tlačítko k odhlášení z účtu.



Obrázek 28: Ukázka stránky *Overview* [autor]

Pro docílení jednoduššího směrování mezi jednotlivými stránkami využívá web rozšiřující modul *Vue Router*, který umožňuje jednotlivým pohledům přidělit URL, na kterých budou přístupné. Dalším rozšiřujícím modulem, který byl při vývoji použit, je *Vuex*. Ten poskytuje nástroje ke sdílení dat napříč komponentami a pohledy, popřípadě celým webem. Následkem toho je, že v případě úpravy dat jednou komponentou se změna projeví i v ostatních komponentách obsahujících tyto data.

Komunikace mezi webovým rozhraním a serverovou částí aplikace probíhá skrz REST API, které server implementuje. Web pomocí API zasílá serveru požadavky na procesy, které chce provést či k získání potřebných dat. K vytváření požadavků

směřujících na server využívá web knihovnu *axios*. Tato knihovna umožňuje jednoduché volání API s podporou vypořádání se s chybami a možností asynchronního volání. Příklad asynchronní funkce vytvářející požadavek pro přihlášení uživatele prostřednictvím emailu a hesla je ve Výpis 4. Tato funkce provádí POST požadavek na URL adresu „`/auth/login`“ obsahující email a heslo. Po zpracování požadavku serverem funkce vrací jako návratovou hodnotu data přijatá ze serveru, která obsahují v případě úspěšného přihlášení autorizační token.

Výpis 4: Ukázka funkce s REST API požadavkem k přihlášení [autor]

```
1.  async function login(email, password) {
2.      var res = await axios.post('/auth/login', {
3.        email: email,
4.        password: password
5.      });
6.      return res.data;
7.  }
```

6.3.3 Back-end server

Back-end server (dále jen server) je hlavní součástí celé platformy. Stará se o přijímání a provádění požadavků od uživatele, ukládání a správu dat, komunikaci s API platforem třetích stran a v neposlední řadě o komunikaci s LN uzlem k přijímání a odesílání LN plateb. Server běží v prostředí Node.js a díky tomu je možné používat programovací jazyk JavaScript stejně jako na front-endu.

Server využívá řadu knihoven neboli balíčků, které jsou spravovány pomocí softwaru NPM. Mezi knihovny, které je třeba zmínit, patří:

- *express* – webový framework pro node.js umožňující vytvářet REST API,
- *axios* – http klient k volání API třetích stran,
- *mysql* – MySQL driver pro node.js sloužící ke komunikaci s databází,
- *grpc-js* – gRPC klient pro JavaScript, který se používá pro vzdálené volání procedur LN uzlu,

- *lnurl* – implementace LNURL poskytující nástroje k lepší uživatelské přívětivosti LN plateb,
- *jsonwebtoken* – implementace standardu JSON Web Token (JWT), který je využíván k autentizaci uživatele,
- *bcryptjs* – knihovna k hašování a porovnávání hesel,
- *passport* –balíček obsahující sadu strategií pro autentizaci účtů platforem třetích stran.

K přijímání požadavků z webového rozhraní server implementuje REST API. To je implementováno pomocí knihovny *express*, která umožňuje odstínění od složitého HTTP protokolu. Použití knihovny *express* je v projektu převážně v souborech *.routes.js, které slouží ke směrování HTTP požadavků na odpovídající funkce. Výpis 5 obsahuje ukázku kódu ze souboru *users.routes.js*, který definuje koncový bod pro požadavky typu POST na adresu „`/me`“. Ta slouží k vrácení informací o přihlášeném uživateli. Obdobně je definována celá řada koncových bodů pro zpracování všech potřebných požadavků z webového rozhraní.

Výpis 5: Ukázka definování POST metody pomocí knihovny *express* [autor]

```
1.  const router = require('express').Router();
2.  const userService = require('./users.service');
3.  const { auth } = require('../auth/auth.jwt');
4.
5.  router.post('/me', auth, asyncHandler(async (req, res) => {
6.    var me = await userService.findById(req.user.id, true);
7.    res.send(me);
8.  }));
```

Některé API požadavky, jako například ve výpisu výše, obsahují autorizaci uživatele tak, aby každý uživatel mohl přistupovat pouze ke svým informacím a aby nebylo ohroženo soukromí ostatních uživatelů. Autorizace probíhá pomocí JWT tokenu, který se předává a uchovává v podobě webových cookies. JWT token je uživateli uložen do cookies po úspěšném přihlášení. Součástí JWT tokenu jsou zašifrované informace o uživateli, které jsou opatřené digitálním podpisem a je tedy možné je ověřit.

Přihlášení, přesněji autentizaci, může uživatel provést celkem třemi způsoby a to pomocí emailu a hesla, YouTube účtu nebo StackExchange účtu. Server k přihlášení uživatelů využívá knihovnu *passport*, která obsahuje řadu rozšiřujících modulů neboli strategií k jednodušší integraci různých způsobů přihlášení. V případě přihlašování pomocí účtů platforem třetích stran komunikuje server s jejich API prostřednictvím protokolu OAuth 2.0. Při přihlašování uživatele pomocí emailu a hesla se používá knihovna *bcryptjs* pro hašování a porovnávání hesla.

Všechny důležitá data jsou v rámci webového serveru ukládána do MySQL databáze. Při komunikaci s touto databází server využívá knihovny *mysql*, která umožňuje jednoduché volání SQL dotazů. Funkce pro přístup k datům v databázi jsou definovány v souborech **.model.js*. Tyto soubory obsahují funkce k vytváření, upravování, smazání a získání dat jednotlivých objektů. Na Výpis 6 je ukázka funkce, která volá SQL příkaz k získání databázových dat o uživateli podle jeho ID. Obdobně jsou definovány potřebné funkce pro všechny objekty.

Výpis 6: Ukázka funkce k volání SQL příkazu pro získání uživatelských dat z databáze [autor]

```
1.  getById: async (id, toView, onlyActive = true) => {
2.    let sql = `
3.      SELECT
4.      ${toView ? User.columnsToView : '*'}
5.      FROM users WHERE id = ?`;
6.    if (onlyActive) sql += ' AND active = 1';
7.    return await db.query(sql, id, { firstOrNull: true });
8.  }
```

Důležitou rolí serveru je komunikace s LN uzlem. Ta probíhá za účelem generování nových LN faktur, kontroly stavu probíhajících plateb a vyplácení tvůrců. LN uzel používá implementaci LND, která podporuje komunikaci prostřednictvím REST API nebo gRPC. Z důvodu větší rychlosti a přívětivějšího použití, server využívá rozhraní gRPC. Přes toto rozhraní je možné ze serveru jednoduše volat vzdálené procedury na LN uzlu. Za tímto účelem server využívá knihovnu *grpc-js*, která umožňuje volání vzdálených gRPC procedur. Stěžejním souborem pro integraci s gRPC rozhraním jsou **.proto* soubory, které definují všechny procedury a zprávy, které dané rozhraní

podporuje. Pro zajištění zabezpečení rozhraní je také vyžadována autentizace pomocí takzvaných *macaroon* souborů. Tyto soubory jsou generovány na straně LN uzlu a jsou vyžadována při používání rozhraní. Server prostřednictvím gRPC rozhraní volá následující procedury:

- *addInvoice* – generuje novou LN fakturu,
- *decodePayReq* – dekóduje informace z LN faktury,
- *sendPaymentSync* – provádí platbu přiložené LN faktury,
- *subscribeInvoices* – slouží k odchytávání změn všech aktivních LN faktur, které LN uzel zpracovává.

Na Výpis 7 je znázorněna funkce *addInvoce*, která volá stejnojmennou vzdálenou proceduru prostřednictvím gRPC rozhraní. Tato funkce generuje nové LN faktury s definovanou hodnotou, popisem a časem expirace. Tento výpis také poukazuje na to, že samotné volání vzdálených procedur se nijak neliší od volání těch lokálních a jedná se tak o pohodlný způsob integrace vzdálených systémů.

Výpis 7: Ukázka funkce ro generování nové LN faktury [autor]

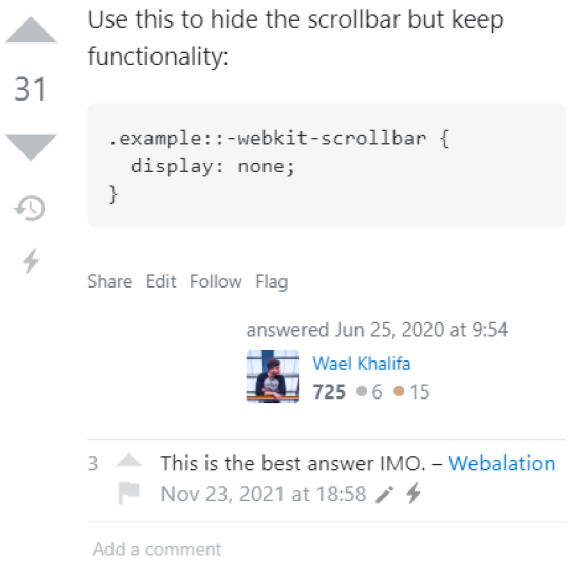
```
1.  addInvoice: (value, memo, expiry) => {
2.    return new Promise((resolve, reject) => {
3.      let request = {
4.        value: value,
5.        memo: memo,
6.        expiry: expiry
7.      };
8.      lnd.addInvoice(request, function (err, response) {
9.        if (err) reject(err);
10.       resolve(response);
11.     });
12.   });
13. }
```

Vzhledem k tomu webová aplikace není součástí stejné lokální sítě jako LN uzel tak komunikace probíhá prostřednictvím internetu. Z tohoto důvodu je k propojení webového serveru a LN uzlu využíván takzvaný *SSH tunel*. Ten umožňuje provádět zašifrovanou komunikaci mezi vzdálenými zažíráními. Navíc díky SSH tunelu může webový server provádět směrování požadavků z internetu přímo na LN uzel. Díky tomu je LN uzel dosažitelný prostřednictvím statické IP adresy webového serveru a nevystavuje svou opravdovou IP adresu, díky tomu ho není možné lokalizovat.

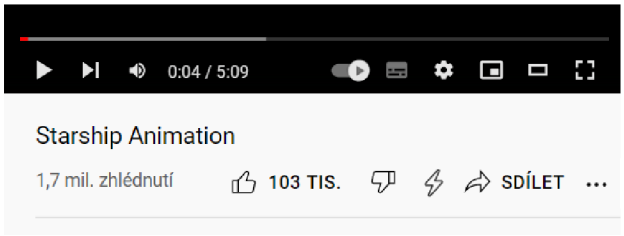
Server dále implementuje uživatelsky přívětivější výběr prostředků z virtuální peněženky pomocí knihovny *lnurl*. Ta podporuje generování speciálních QR kódů, které při načtení uživatelskou LN peněženkou umožňují provézt LN platbu směrem k uživateli a vybrat tím tak nashromážděné prostředky. Bez této funkce by uživatel při každém výběru musel nejprve vygenerovat novou LN fakturu ve své LN peněžence a následně ji zadat do webové aplikace. To je však uživatelsky nepřívětivé vzhledem k tomu, že LN faktura má většinou podobu QR kódu, který se velice špatně sdílí v rámci jednoho zařízení, a také je špatně čitelný prostřednictvím počítače či notebooku.

7 Výsledky

Tato kapitola popisuje výslednou darovací platformu, a to především z pohledu uživatele. Nejprve z pohledu dárce, který má na svém webovém prohlížeči nainstalované vyvinuté rozšíření. Toto rozšíření generuje uživateli na podporovaných platformách tlačítka pro přispívání, aby bylo přispívání co nejvíce jednoduché. Na obrázku Obrázek 29 a Obrázek 30 jsou příklady tlačítek v podobě ikony blesku integrované do platformy YouTube a StackExchange. Tlačítka jsou záměrně nevýrazná tak, aby splynula s ostatními komponenty webu a nerušila



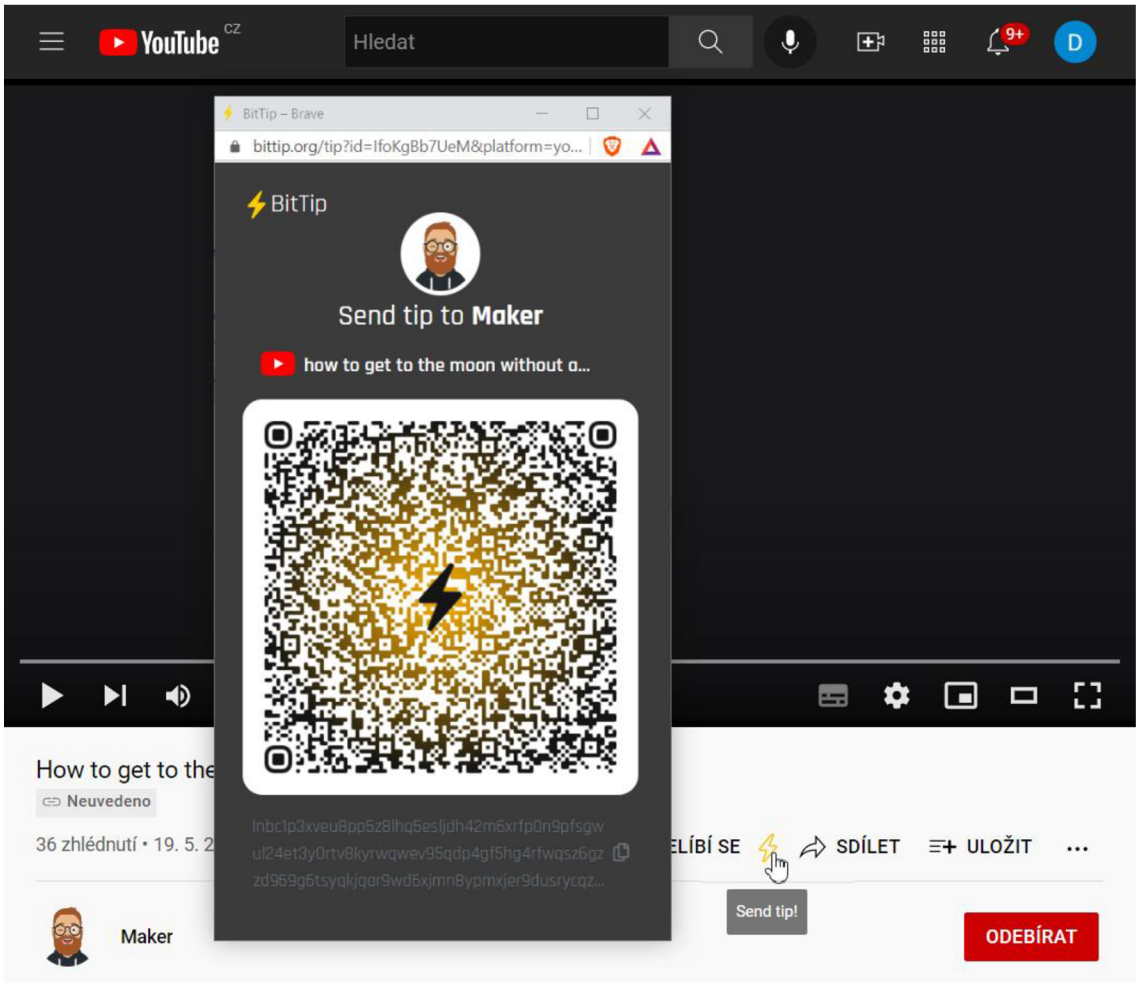
Obrázek 30: Tlačítko pro přispívání na platformě StackExchange [autor]



Obrázek 29: Tlačítko pro přispívání pod YouTube videem [autor]

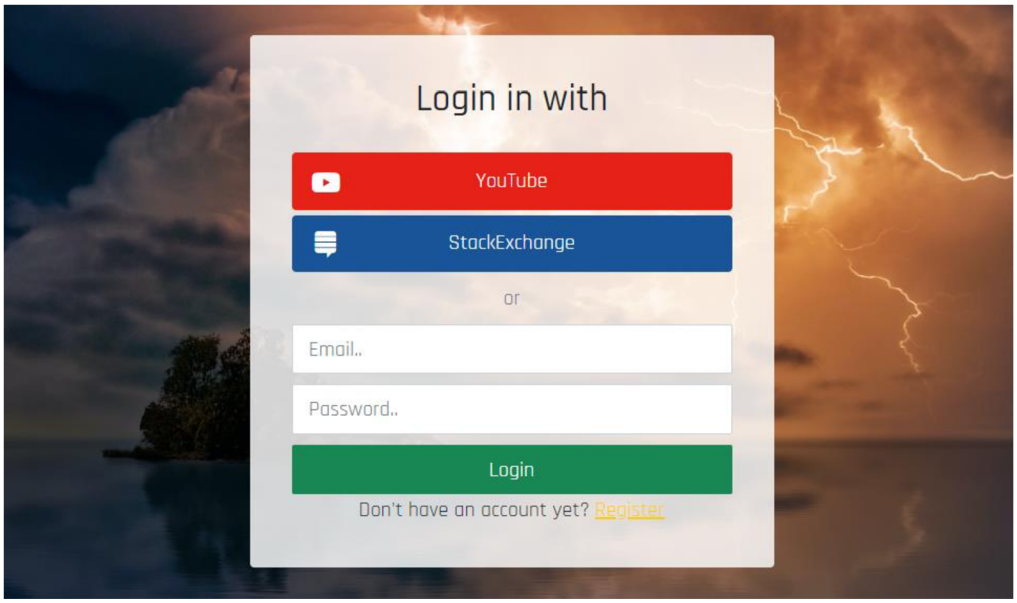
uživatele.

Po kliknutí na tlačítko se otevře nové okno s vygenerovanou LN fakturou v podobě QR kódu a informacemi o tvůrci a obsahu. V případě, že tvůrce daného obsahuje není zaregistrován do darovací platformy, dárci se zobrazí hláška, že danému tvůrci není možné přispět. Na Obrázek 31 je zobrazena situace po kliknutí na tlačítko pro přispění na video platformy YouTube.



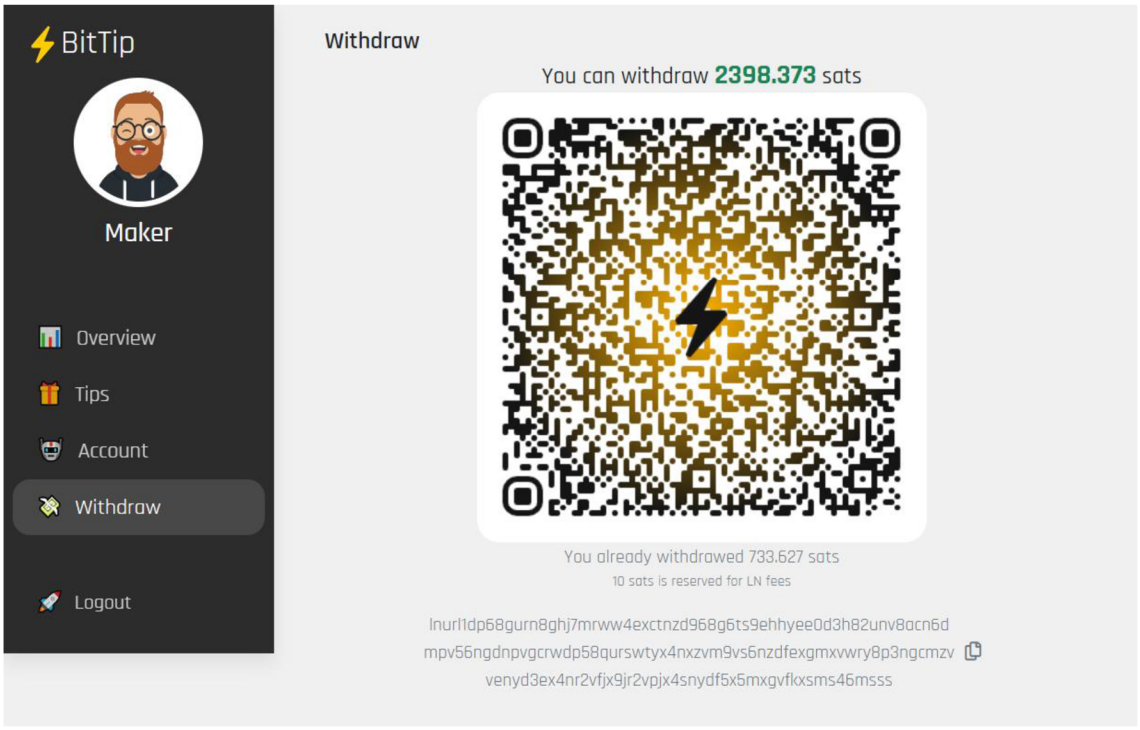
Obrázek 31: Ukázka vyskakovacího okna pro přispívání tvůrci YouTube videa [autor]

Po zaplacení LN faktury prostřednictvím libovolné LN peněženky je tvůrci přičtena přispěná částka na virtuální peněženku vedenou v rámci platformy. Tvůrce může sledovat přijaté příspěvky na stránce *Overview* nebo *Tips*. Nejdřív se však musí autentizovat na stránce *Login*, která je zobrazena na Obrázek 32. K přihlášení může využít jak email a heslo, tak YouTube účet či StackExchange účet. Po přihlášení má uživatel přístup mimo jiné na stránku *Account*, kde může připojovat nebo odpojovat účty třetích stran.



Obrázek 32: Stránka pro přihlášení uživatele [autor]

Výběr nashromážděných prostředků může přihlášený tvůrce provést na stránce *Withdraw*, ta je zobrazena na Obrázek 33. Tato stránka obsahuje informace o tom, kolik prostředků již tvůrce v minulosti vybral a kolik jich je ještě k dispozici k výběru. Hlavní součástí stránky *Withdraw* je však speciální LNURL QR kód umožňující výběr do libovolné LN peněženky. Tento QR kód může tvůrce naskenovat pomocí své LN peněženky a vybrat tím z účtu prostředky.



Obrázek 33: Ukázka stránky pro výběr prostředků z virtuální peněženky [autor]

8 Závěr

Hlavním cílem této práce bylo vyvinout darovací platformu pro zpřístupnění možnosti přijímání darů pro tvůrce obsahu na internetu, a to ve virtuální měně bitcoin. K přesunu bitcoinů v rámci darovací platformy byla využita platební síť Lightning network. Technologiím Bitcoin a Lightning network byla věnována teoretická část práce, která se zaměřila především na jejich technické principy. Z výsledků technické analýzy bylo zjištěno, že i přes relativní náročnost použití sítě Lightning network se jeví jako vhodná k účelům darovací platformy. To především z důvodu podpory plateb s velice nízkou částkou. Provedené platby v rámci Lightning network sítě jsou navíc rychlé a s nízkými poplatky na zpracování.

V rámci praktické části práce byly vymezeny požadavky na vyvíjenou platformu, které by mělo nové řešení splňovat. Na základě výsledků z technické analýzy Bitcoinové technologie a platební sítě Lightning network bylo navrženo řešení splňující stanovené požadavky na darovací platformu. Z principu fungování Lightning network sítě bylo v rámci řešení nutné při převodu hodnoty mezi dárcem a tvůrcem použít prostředníka. Při provádění příspěvků není převedena hodnota daru přímo tvůrci, ale je odeslána právě tomuto prostředníkovi. Tvůrce pak může prostředky, které mu náleží od prostředníka kdykoliv vybrat. Bez použití prostředníka by nebylo možné docílit žádoucí uživatelské přívětivosti platformy.

Poslední část práce pak byla věnována samotné implementaci navrhované darovací platformy. Tato implementace sestává ze tří částí, které jsou navzájem propojené a umožňují uživateli pohodlně provádět přispívání na třech vybraných platformách a to YouTube, StackExchange a Medium. První část implementace je rozšíření webového prohlížeče umožňující integraci přispívání přímo do zmíněných platforem. Druhá část je Lightning network uzel, který přijímá platby od dárců a zprostředkovává následný výběr prostředků tvůrcem. Poslední částí implementace je webová aplikace, která se stará o chod celé platformy propojováním všech součástí a uživatele.

Výsledné řešení by mohlo být v budoucnu vylepšeno o možnost zasílání příspěvků napřímo mezi dárcem a tvůrcem. Této funkcionality zatím nebylo možné v rámci této práce dosáhnout bez značného snížení použitelnosti darovací platformy.

9 Použitá literatura

[1] ANDRESEN, Gavin. *BIP 11 - M-of-N Standard Transactions* [online]. 18. říjen 2011 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0011.mediawiki

[2] ANDRESEN, Gavin. *BIP 13 - Address Format for pay-to-script-hash* [online]. 18. říjen 2011 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0013.mediawiki

[3] ANDRESEN, Gavin. *BIP 16 - Pay to Script Hash* [online]. 3. leden 2012 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0016.mediawiki

[4] ANTONOPOULOS, Andreas M. *Mastering Bitcoin: Programming the Open Blockchain*. Second edition. Sebastopol, CA: O'Reilly, 2017. ISBN 978-1-4919-5438-6.

[5] ANTONOPOULOS, Andreas M, Olaoluwa OSUNTOKUN a René PICKHARDT. *Mastering the Lightning Network: A Second Layer Blockchain Protocol for Instant Bitcoin Payments*. 2021. ISBN 978-1-4920-5486-3.

[6] BISTARELLI, Stefano, Ivan MERCANTI a Francesco SANTINI. An Analysis of Non-standard Transactions. *Frontiers in Blockchain* [online]. 2019, **2**. Dostupné z: doi:10.3389/fbloc.2019.00007

[7] BITCOIN PROJECT. Bitcoin Core version 0.13.1 released. *Bitcoin.org* [online]. 27. říjen 2016 [vid. 2022-02-17]. Dostupné z: https://bitcoin.org/en/release/v0.13.1

[8] BITCOIN PROJECT. Block Chain - Bitcoin. *Bitcoin.org* [online]. 12. květen 2019 [vid. 2021-12-17]. Dostupné z: https://developer.bitcoin.org/reference/block_chain.html

[9] BITCOIN PROJECT. Transactions - Bitcoin (Developer Guide). *Bitcoin.org* [online]. 19. leden 2021 [vid. 2022-01-28]. Dostupné z: https://developer.bitcoin.org/devguide/transactions.html

[10] BITCOIN PROJECT. Transactions - Bitcoin (Reference). *Bitcoin.org* [online]. 6. listopad 2020 [vid. 2021-12-17]. Dostupné z: https://developer.bitcoin.org/reference/transactions.html

[11] BITCOINCORE.ORG. Segregated Witness Benefits. *Bitcoin Core* [online]. 26. leden 2016 [vid. 2022-03-04]. Dostupné z: https://bitcoincore.org/en/2016/01/26/segwit-benefits/

[12] BRAKMIĆ, Harris. *Bitcoin and Lightning Network on Raspberry Pi: running nodes on Pi3, Pi4 and Pi Zero* [online]. 2019 [vid. 2021-04-27]. ISBN 978-1-4842-5522-3. Dostupné z: https://www.overdrive.com/search?q=41DEC669-F317-4F2A-A32C-F0EA074B61D1

[13] BTCDRAK, Mark FRIEDENBACH a Eric LOMBROZO. BIP 112 - CHECKSEQUENCEVERIFY. *Bitcoin Improvement Proposal* [online]. 1. říjen 2014 [vid. 2021-10-22]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0112.mediawiki

[14] DRYJA, Thaddeus. *13. Payment Channels and Lightning Network* [online]. 12. červenec 2019 [vid. 2022-04-02]. MIT MAS.S62 Cryptocurrency Engineering and Design. Dostupné z: https://www.youtube.com/watch?v=Hzv9WuqIzA0

[15] FRIEDENBACH, Mark, BTCDRAK, Nicolas DORIER a KINOSHITAJONA. BIP 68 - Relative lock-time using consensus-enforced sequence numbers. *Bitcoin Improvement Proposal* [online]. 28. květen 2015 [vid. 2021-10-22]. Dostupné z: https://github.com/bitcoin/bips/blob/a79eb556f37fdac96364db546864cbb9ba0cc634/bip-0068.mediawiki

[16] KERIN, Thomas a Mark FRIEDENBACH. BIP 113 - Median time-past as endpoint for lock-time calculations. *Bitcoin Improvement Proposal* [online]. 10. srpen 2015 [vid. 2021-12-04]. Dostupné z: https://github.com/bitcoin/bips/blob/edffe529056f6dfd33d8f716fb871467c3c09263/bip-0113.mediawiki

[17] LAU, Johnson. *BIP 147 - Dealing with dummy stack element malleability* [online]. 2. září 2016 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0147.mediawiki

[18] LOMBROZO, Eric, Johnson LAU a Pieter WUILLE. *BIP 141 - Segregated Witness (Consensus layer)* [online]. 21. prosinec 2015 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/97e02b2223b21753acefa813a4e59dbb6e849e77/bip-0141.mediawiki

[19] NAKAMOTO, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008, 2008, 9.

[20] POON, Joseph a Thaddeus DRYJA. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016, 59.

[21] PRITZKER, Yan a Tereza WONGOVÁ. *Vynález jménem bitcoin*. 2020. ISBN 978-80-907975-0-5.

[22] RUSSELL, Rusty, Christian DECKER, Landon MUTCH a Fabrice DROUIN. *BOLT #4: Onion Routing Protocol* [online]. 14. listopad 2016 [vid. 2022-04-05]. Dostupné z: https://github.com/lightning/bolts/blob/32a76e80c737155db196ec96a27395fd27586260/04-onion-routing.md

[23] RUSSELL, Rusty, Landon MUTCH a Bastien TEINTURIER. *BOLT #11: Invoice Protocol for Lightning Payments* [online]. 27. červen 2017 [vid. 2022-04-05]. Dostupné z: https://github.com/lightning/bolts/blob/93909f67f6a48ee3f155a6224c182e612dd5f187/07-routing-gossip.md

[24] RUSSELL, Rusty, Pierre-Marie PADIOU a Landon MUTCH. *BOLT #2: Peer Protocol for Channel Management* [online]. 15. listopad 2016 [vid. 2022-04-05]. Dostupné z: https://github.com/lightning/bolts/blob/32a76e80c737155db196ec96a27395fd27586260/02-peer-protocol.md

[25] RUSSELL, Rusty, Pierre-Marie PADIOU a Landon MUTCH. *BOLT #3: Bitcoin Transaction and Script Formats* [online]. 15. listopad 2016 [vid. 2022-04-05]. Dostupné z: https://github.com/lightning/bolts/blob/32a76e80c737155db196ec96a27395fd27586260/03-transactions.md

[26] RUSSELL, Rusty, Neil SAITUG a Christian DECKER. *BOLT #7: P2P Node and Channel Discovery* [online]. 21. listopad 2016 [vid. 2022-04-05]. Dostupné z: https://github.com/lightning/bolts/blob/93909f67f6a48ee3f155a6224c182e612dd5f187/07-routing-gossip.md

[27] SONG, Jimmy. *Programming bitcoin: learn how to program bitcoin from scratch*. First edition. Beijing: O'Reilly, 2019. ISBN 978-1-4920-3149-9.

[28] TODD, Peter. BIP 65 - OP_CHECKLOCKTIMEVERIFY. *Bitcoin Improvement Proposal* [online]. 1. říjen 2014 [vid. 2021-10-22]. Dostupné z: https://github.com/bitcoin/bips/blob/02de475efc528058bd04a0c4ad31b6422aed5f5f/bip-0065.mediawiki

[29] WALKER, Greg. TXID - The hash of a transaction's data. *learn me a bitcoin* [online]. 2015 [vid. 2021-10-28]. Dostupné z: https://learnmeabitcoin.com/technical/txid#footnote-unique-txids

[30] WUILLE, Pieter a Greg MAXWELL. *BIP 173 - Base32 address format for native v0-16 witness outputs* [online]. 20. březen 2017 [vid. 2022-01-23]. Dostupné z: https://github.com/bitcoin/bips/blob/97e02b2223b21753acefa813a4e59dbb6e849e77/bip-0173.mediawiki

10 Přílohy

A. Přiložený adresář

Součástí práce je přiložený adresář obsahující zdrojové kódy implementace. Spuštěná implementace darovací platformy je pro testovací účely dostupná na webu https://bittip.org/. Přiložený podadresář obsahuje:

- *web* – adresář obsahující webovou aplikaci rozdělenou na dva podadresáře:
  - *client* – zdrojové kódy webového rozhraní,
  - *server* – zdrojový kód pro back-end server,
- *chrome_extension* – adresář obsahující rozšíření pro webový prohlížeč Google Chrome,
- *db.sql* – soubor obsahující strukturu MySQL databáze.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. David Nekolný**

Studium: I1800757

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Vývoj darovací platformy využívající Lightning network**

Název diplomové práce AJ: Development of a donation platform using the Lightning network

Cíl, metody, literatura, předpoklady:

Cíl práce

Cílem práce je vyvinout platformu umožňující internetovým tvůrcům přijímat dary od uživatelů prostřednictvím Lightning network.

Osnova

1. Úvod
2. Cíl práce, metodika
3. Bitcoin a Blockchain
4. Lightning network
5. Analýza a návrh řešení
6. Implementace
7. Výsledky
8. Shrnutí výsledků
9. Závěr a doporučení
10. Použitá literatura

NAKAMOTO, Satoshi, nedatováno. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008, 9.

POON, Joseph a Thaddeus DRYJA, 2016. The Bitcoin Lightning Network: 59.

ANTONOPOULOS, Andreas M., 2015. Mastering bitcoin. First edition. Sebastopol CA: O'Reilly. ISBN 978-1-4493-7404-4.

BRAKMIĆ, Harris, 2019. Bitcoin and Lightning Network on Raspberry Pi: running nodes on Pi3, Pi4 and Pi Zero [online] [vid. 2021-04-27]. ISBN 978-1-4842-5522-3.

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Datum zadání závěrečné práce: 25.12.2020