Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-100241-79970

Matúš Zeleňák

MANAŽMENT BEZDRÔTOVÝCH SIETÍ V INTERNETE VECÍ

Bakalárska práca

Vedúci práce: Ing. Jaroslav Erdelyi

Máj 2019

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-100241-79970

Matúš Zeleňák

MANAŽMENT BEZDRÔTOVÝCH SIETÍ V INTERNETE VECÍ

Bakalárska práca

Študijný program: Informatika

Študijný odbor: Informatika

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava

Vedúci práce: Ing. Jaroslav Erdelyi

Máj 2019

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

ZADANIE BAKALÁRSKEHO PROJEKTU

Meno študenta:	**Zeleňák Matúš**
Študijný odbor:	Informatika
Študijný program:	Informatika
Názov projektu:	**Manažment bezdrôtových sietí v Internete vecí**

Zadanie:
V prostredí Internetu vecí existujú rôzne aplikácie. Tieto aplikácie získavajú údaje s pomerne veľkej škály zariadení. Tieto zariadenia potrebujú mať nejaký manažment. Preto je cieľom tohto projektu vytvoriť manažment pre siete, ktoré sa využívajú v Internet of Things. Internet of Things siete sú väčšinou tvorené bezdrôtovými zariadeniami, preto je nutné aby aj manažovanie v takýchto sieťach bolo bezpečné. V rámci riešenia je nutné spraviť analýzu aktuálnych riešení a v návrhu zohľadniť jednotlivé riešenia a následne navrhnúť bezpečný manažment pre bezdrôtové siete. Je nutne tento system aj implementovať a následne otestovať v simulátore alebo na reálnych zariadeniach a porovnať s existujúcimi riešeniami.

Práca musí obsahovať:

Anotáciu v slovenskom a anglickom jazyku
Analýzu problému
Opis riešenia
Zhodnotenie
Technickú dokumentáciu
Zoznam použitej literatúry
Elektronické médium obsahujúce vytvorený produkt spolu s dokumentáciou

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava
Vedúci projektu: Ing. Jaroslav Erdelyi

Termín odovzdania práce v zimnom semestri : 11. 12. 2018
Termín odovzdania práce v letnom semestri : 7. 5. 2019

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4
1

Bratislava 17. 9. 2018

prof. Ing. Pavol Návrat, PhD.
riaditeľ ÚISI

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave 6.5.2019 .....

Matúš Zeleňák

Poďakovanie

Týmto by som sa chcel poďakovať v prvom rade vedúcemu práce, Jarovi Erdelyimu, za odborné konzultácie a ochotu vždy pomôcť, bez ohľadu na to, v akej pokročilej nočnej hodine som sa naňho obrátil s problémom. Ďalej patrí moja veľká vďaka rodičom za psychickú podporu a otcovi za to, že mi vždy ochotne pomohol s akýmkoľvek problémom. Nakoniec patrí špeciálne ďakujem spolužiakovi, kolegovi, kamarátovi, Šimonovi Várošovi, s ktorým sme strávili nespočetne veľa prebdených nocí písaním svojich prác a navzájom sme sa vždy podporovali a motivovali.

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Autor: Matúš Zeleňák
Bakalárska práca: Manažment bezdrôtových sietí v internete vecí
Vedúci bakalárskeho projektu: Ing. Jaroslav Erdelyi
Máj 2019

V prostredí Internetu vecí existujú rôzne aplikácie. Tieto aplikácie získavajú údaje z pomerne veľkej škály zariadení. Tieto siete potrebujeme nejako manažovať. Vzhľadom na to, že väčšina týchto sietí je bezdrôtových, musíme tieto siete vedieť manažovať bezpečne. Účelom tejto práce je porovnať existujúce riešenia a navrhnúť a naimplementovať vhodné riešenie pre manažment takýchto bezdrôtových sietí v internete vecí. Ideálne by bolo naimplementovať viaceré riešenia a medzi sebou ich porovnať, vybrať to najvhodnejšie z nich pre manažovanie bezdrôtových sietí v IoT a to použiť vo finálnom riešení. Toto riešenie je následne potrebné otestovať na reálnych zariadeniach.

Annotation

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies
Degree Course: Informatics

Author: Matúš Zeleňák
Bachelor's Thesis: Wireless Network Management in the Internet of Things
Supervisor: Ing. Jaroslav Erdelyi
May 2019

There are different applications in the Internet of Things. These applications receive data from a relatively large range of devices. We need to manage these networks somehow. Since most of these networks are wireless, we must be able to manage these networks safely. The purpose of this work is to compare existing solutions and design and implement a suitable solution for managing such wireless networks in the Internet of Things. It would be ideal to implement several solutions and compare them, to choose the most suitable one for managing wireless networks in the IoT and to use it in the final solution. This solution then needs to be tested on real devices.

Obsah

Kapitola 1

Úvod

Nie je to tak dávno, čo bol pojem IoT väčšine populácie cudzí. Postupom času sa však dostáva do povedomia stále väčšieho okruhu ľudí. V dnešnej dobe sa na internete potrebujeme chrániť pred rôznymi typmi útokov a útočníkov. To isté platí pre IoT. Takisto aj IoT sieť môže byť napadnutá / zneužitá. Potrebujeme preto nejako overovať zariadenia, ktoré do siete pristupujú a posielajú do nej konfigurácie alebo update-y. Toto vieme efektívne zabezpečiť pomocou dobre navrhnutého blockchainu a jeho konsenzuálnych algoritmov.

V prvej kapitole si zanalyzujeme problémovú oblasť. Rozoberieme si existujúce riešenia a povieme si ich výhody a nevýhody. Začneme všeobecne o internete vecí a postupne sa dostaneme k zaujímavým riešeniam fungujúcim v praxi. Následne sa dostaneme k najhlavnejšej časti, ktorou je blockchain a konsenzuálne algoritmy. Na záver si povieme niečo o komunikačných a manažovacích protokoloch v sieťach.

V kapitole Návrh si spravíme predbežný návrh nášho riešenia. Tento sa ešte nebude musieť úplne zhodovať s naimplementovaným riešením, ale mal by mu z

väčšej časti odpovedať. Navrhneme si predovšetkým fungovanie algoritmov, ktoré budeme implementovať a štruktúru komunikačného protokolu.

V ďalšej kapitole sa bližšie pozrieme na priebeh implementácie. Takisto si v nej uvedieme základnú štruktúru implementovaného programu a fungovania algoritmov, ktorá bude podrobnejšie rozobraná v prílohe Technická dokumentácia.

Poslednou dôležitou kapitolou je overenie, v ktorom popíšeme overenie správnosti algoritmov, ich testovanie a zmeriame ich parametre, akými sú napríklad čas trvania vykonania alebo energetická náročnosť. V overení taktiež porovnáme tieto algoritmy a stanovíme záver, v ktorom určíme, ktorý algoritmus je pre naše riešenie najvhodnejší.

Kapitola 2

Analýza problémovej oblasti

2.1 IoT - Internet vecí

IoT, čiže Internet of Things, je v dnešnej dobe stále viac a viac skloňovanou témou. V tejto rýchlej dobe, kedy sa ľudstvo snaží všetko si uľahčovať, automatizovať, digitalizovať je IoT dokonalým prostriedkom na zjednodušenie života. V skratke by sa dalo povedať, ako už z názvu vyplýva, že IoT sú rôzne zariadenia, resp. veci dennej potreby (najrôznejšie domáce spotrebiče, merače, hlasoví asistenti..), ktoré sú navzájom poprepájané a komunikujú. Veľmi často bývajú tieto zariadenia pripojené aj do Internetu, kam môžu napríklad odosielať dáta, alebo odkiaľ môžu prijímať inštrukcie, konfiguráciu, atď [14].

S IoT nezadržateľne postupujú vpred aj kľúčové technológie pre jeho rozvoj a tými sú bezpochyby Wireless Sensor Networks (WSN), nanotechnológie a miniaturizácia.

2.1.1 Smart city

Jednou z veľkých oblastí, ktorých neodmysliteľnou súčasťou je IoT je oblasť Smart city. Neexistuje presná definícia toho, čo je smart city, no môže to byť chápané ako koncept, ktorý sa snaží znižovať spotrebu energií, využívať obnoviteľné zdroje, optimalizovať dopravu, či znížiť nepriaznivý dopad na životné prostredie. V skratke, je to koncept, ktorý má zlepšiť kvalitu života v mestách vo viacerých smeroch. A k tomuto z veľkej časti napomáha aj IoT.

2.1.2 Smart parking

Smart parking je začínajúcou témou v oblasti IoT. Ide v podstate o inteligentnú evidenciu parkovacích miest v mestských aglomeráciách. Je to koncept postavený na princípe senzora, ktorý je integrovaný priamo v parkovacom mieste a odosiela dáta zariadeniu, ktoré ich zhromažďuje a publikuje na nejakú webovú / mobilnú aplikáciu, kde si užívatelia vedia pozrieť obsadenosť parkovacích miest v reálnom čase [18].

Veľmi peknou realizáciou tohto konceptu je projekt "zaparkuj.to", ktorý vznikol pôvodne ako tímový projekt na Fakulte informatiky a informačných technológií STU v Bratislave v spolupráci s firmou Unicorn a Orange Slovensko [18].

"Na parkovacích miestach sa do asfaltu vŕtajú špeciálne senzory. Tie majú na vrchu magnetické čidlo, ktoré vie detegovať zmenu magnetického poľa, ktorú spôsobí auto pri príchode/odchode z parkovacieho miesta. Zmena magnetického poľa vyvolá odoslanie správy o zmene obsadenosti (s ďalšími metadatami ako stav batérie, sila signálu atď.) cez sieť LoRa. Na projekte sme spolupracovali s firmou Orange, ktorá práve zabezpečovala komunikáciu cez LoRa, pričom sme využili ich IoT infraštruktúru, ktorá sa volá Live Objects. Posielanie cez túto infraštruktúru

funguje na princípe MQTT protokolu. Z MQTT kanálov si potom už samostatná aplikácia číta prichádzajúce správy a spracováva ich tak, aby vedela všetko potrebné vizualizovať atď." povedal doktor Ivan Srba (FIIT), vedúci projektu Smart parking o tomto projekte.

2.1.3 Smart home

Je zvykom, že ľudia prispôsobujú domáce prostredie svojim požiadavkam. V praxi to znamená, že pokiaľ majú ľudia doma tmu, zažnú si svetlo, pokiaľ majú doma chladno, zakúria si, atď. Čo keby sme si tieto jednoduché a bežné procesy zautomatizovali m? Predstavte si, že by ste v lete prišli domov a v dome by bola ideálna teplota, pretože Váš inteligentný dom zdetegoval, že je vnútri príliš teplo, a tak sa rozhodol zapnúť klimatizáciu. Alebo, že sa vrátite domov a máte povysávané, pretože Váš inteligentný dom zdetegoval, že sa nikto nenachádza doma a teda vyhodnotil, že je ideálny čas povysávať, pretože teraz to nebude nikoho vyrušovať. Toto všetko môžeme zahrnúť pod pojem smart home, čiže inteligentná domácnosť [6].

Inteligentná domácnosť pozostáva v prvom rade zo senzorov. Tieto senzory môžu snímať a zhromažďovať údaje o činnosti obyvateľov alebo o stave okolia a v neposlednom rade o spotrebe energií. Tieto zhromaždené údaje softvér zanalyzuje a na ich základe sa môžu rôzne procesy v domácnosti automatizovať. Okrem automatizácie vieme na základe zanalyzovaných údajov urobiť štatistiky o spotrebe energií a nastaviť spotrebu tak, aby sme zbytočne nemíňali [6].

2.2 Blockchain

A práve to, že sú zariadenia pripojené do Internetu môže byť veľkým bezpečnostným problémom. Keďže Internet je sieť obrovského množstva navzájom poprepájaných zariadení a menších sietí, nikdy nevieme, čo nás tam čaká, na koho tam narazíme. Na Internete je veľa záškodníkov, hackerov... A preto treba mať svoju sieť dobre zabezpečenú. Čo sa môže stať napríklad v prípade napadnutia takej IoT siete? No jedným z najhlavnejších problémov je určite zaslanie zlej konfigurácie na niektoré zo zariadení, čo môže mať ďalšie nepríjemné následky.

Blockchain a jeho konsenzuálne algoritmy sú dobrým riešením pre zabezpečenie takejto siete. A práve Blockchainu a jeho konsenzuálnym algoritmom sa budeme ďalej v tejto práci venovať. Povieme si o tom, čo je to blok, čo je to reťaz, ako sa zabezpečí prepojenie blokov v reťazi.

Autorom blockchainu (a kryptomeny s názvom Bitcoin, pre ktorú bol tento algoritmus navrhnutý), je osoba alebo skupina osôb pod pseudonymom Satoshi Nakamoto, ktorý popísal tento svoj objav v dokumente [12]. V tejto kapitole uvádzame jeho základné myšlienky.

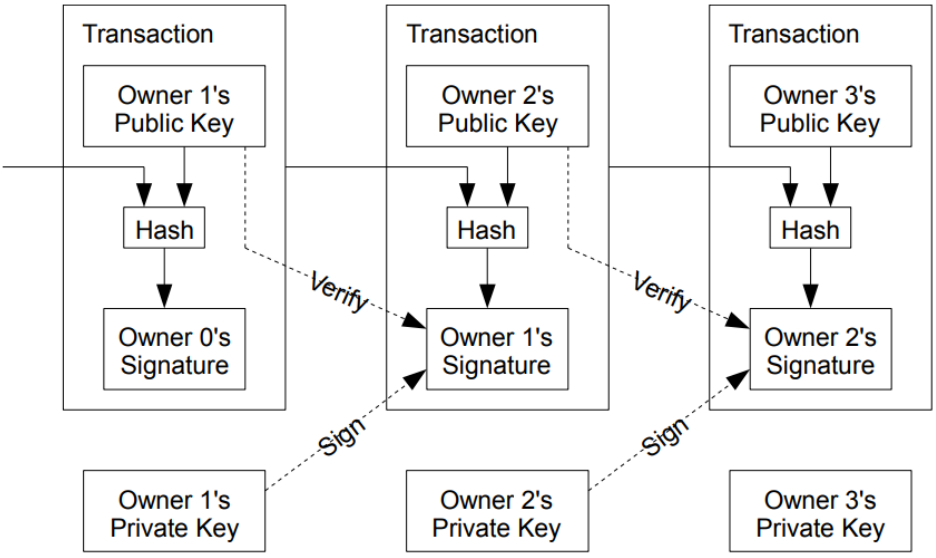
2.2.1 Transakcie a bloky

Pod transakciou rozumieme prenos informácie od odosielateľa ku prijímateľovi. V prípade kryptomien sa pod transakciou rozumie prevod peňazí od jednej osoby ku druhej.

Transakcie prevedené v sieti sa zapisujú do väčších celkov, nazývaných bloky. Blok predstavuje sadu záznamov o prevedených transakciách, ktoré navzájom nemusia súvisieť. V prípade kryptomien sú v bloku zapísané rôzne navzájom nesúvi-

siace prevody medzi rôznymi osobami. Každý takýto prevod, transakciu, popisujú informácie, ako dátum, čas, suma, odosielateľ, prijímateľ.

Technicky sa prevod peňazí realizuje tak, že pôvodný vlastník digitálne podpíše hash predchádzajúcej transakcie a verejného kľúča nového vlastníka. Táto transakcia po verifikácii na to určenými zariadeniami, tzv. timestamp servrami (viď. nasledujúca kapitola) je zapísaná do bloku ku ďalším tam zapísaným transakciám. Každý blok má svoju pevne určenú veľkosť. Keď sa blok naplní verifikovanými transakciami, je taktiež verifikovaný ako celok timestamp servrami (Obr. 2.1).



Obr. 2.1: Podpisovanie a overovanie transakcií

2.2.2 Timestamp server

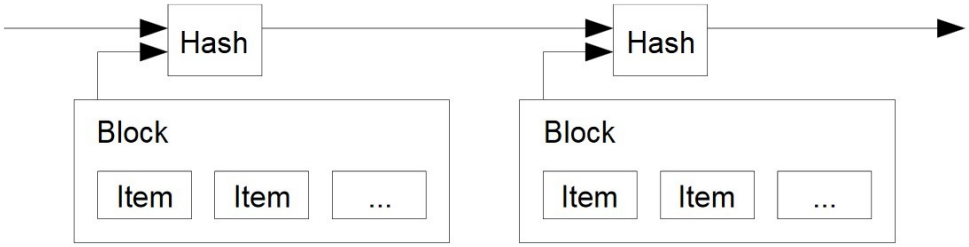
Timestamp server je uzol siete. Týchto uzlov môže byť vela, ich počet závisí od typu použitého konsenzuálneho algoritmu a od rozšírenia a mohutnosti siete. Timestamp server vydáva časové pečiatky. Vydaná časová pečiatka má zabrániť spochybneniu alebo zmene informácie, ktorá je časovou pečiatkou potvrdená. Úlohou timestamp servera je:

- potvrdzovať transakcie
- potvrdzovať bloky transakci
- zapisovať nové bloky do reťaze
- uchovávať celú reťaz kvôli overeniu histórie transakcií.

Timestamp server pracuje tak, že vytvára hash bloku transakcií a rozpošle ho do siete. Časová pečiatka (hash), ktorú takto vytvorí timestamp server, dokazuje, že dáta (transakcie) museli existovať v tom čase, keď sa dostali do vytvoreného hashu. Každá časová pečiatka zahŕňa do svojho hashu aj časovú pečiatku predchádzajúceho bloku, čím vytvára reťaz, v ktorej každá ďalšia časová pečiatka potvrdzuje predchádzajúce časové pečiatky. Toto môžme vidieť na obrázku 2.2.

2.2.3 Chain

Chain je reťaz blokov (tzv. blockchain), ktoré vzniknú chronologickým radením vznikajúcich blokov transakcií do radu za seba spôsobom popísaným v predchádzajúcej stati. Táto reťaz dokazuje to, že všetky transakcie, ktoré sa v nej



Obr. 2.2: Zreťazovanie blokov

nachádzajú, boli uskutočnené. Takisto dokazuje aj to, v akom poradí boli tieto transakcie uskutočnené.

2.2.4 Validácia a konsenzus

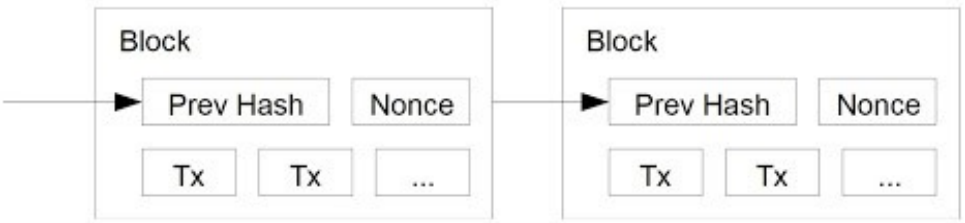
V dnešnej dobe existuje viacero spôsobov, ako môžu byť bloky v reťazi validované. Prvý a najznámejší z nich je „Proof of Work“ implementovaný v Satoshi Nakamotovej kryptomene Bitcoin. Ďalej sa zameriame naňho a o ďalších si povieme niečo v kapitole „Konsenzuálne algoritmy“.

Proof of work zahŕňa hľadanie hodnoty, ktorú keď zahashujeme napr. s hashovacím algoritmom SHA-256, hash bude začínať s niekoľkými nulovými bitmi. Priemerná práca potrebná na toto rastie exponenciálne s počtom požadovaných nulových bitov, môže však byť verifikovaná vypočítaním jediného hashu.

Do timestamp servra, kde sa toto hľadanie uskutočňuje, je Proof of Work implementovaný pridaním čísla „nonce“ do bloku a hľadaním hashu celého bloku. Hľadanie prebieha postupným inkrementovaním čísla „nonce“, až pokiaľ hash celého bloku neobsahuje požadovaný počet nulových bitov. Schému môžme vidieť na obrázku 2.3.

Keď raz bol vydaný potrebný CPU výkon na splnenie podmienky proof of

work na počet nulových bitov, nie je možné zmeniť tento blok bez opätovného vykonania takejto práce. Keďže za blok sa pridávajú ďalšie bloky, zmena bloku by vyžadovala vykonanie práce aj na ostatných pridaných blokoch.



Obr. 2.3: Zreťazovanie s číslom nonce

Na vyrovnanie rastúcej výpočtovej sily hardwaru a tiež narastajúceho záujmu o prevádzku timestamp servrov, je možné meniť obtiažnosť proof of work zmenou požadovaného počtu nulových bitov. S ich počtom narastá obtiažnosť proof of work exponenciálne. Nárast výpočtovej sily a záujmu sa odzrkadľuje na priemernej hodnote počtu vytvorených nových blokov za hodinu. Ak nárast tejto hodnoty je prudký, čo značí, že nové bloky vznikajú príliš rýchlo, obtiažnosť sa zvýši.

Blockchain protokol bráni existencii viacerých reťazí pomocou procesu zvaného „konsenzus“. V prípade, ak by existovali viaceré rôzne kópie reťaze blokov, konsenzus protokol si vyberie tú, ktorej dôveruje najväčší počet užívateľov (hostov siete). Najviac používateľov znamená, že do tejto reťaze sa pridávajú bloky najrýchlejšie, čo znamená že táto reťaz rastie najrýchlejšie, tým pádom je zo všetkých reťazí najdlhšia. Preto si konsenzus protokol vyberie najdlhšiu reťaz.

2.2.5 Sieť

Blockchain riešenia sú založené na peer-to-peer sieťach, kde užívatelia zdieľajú súbory medzi svojimi počítačmi napriamo, bez ďalšieho sprostredkovateľa.

Pre sieť, v ktorej je prevádzkovaný blockchain, musia platiť isté pravidlá, ktoré zaručia správnu funkčnosť blockchainu:

- Nové transakcie sú zasielané všetkým uzlom siete
- Každý uzol siete zbiera nové transakcie do bloku
- Určené uzly siete (v závislosti od použitého konsenzuálneho algoritmu) zabezpečujú dôkaz pre svoj blok
- Keď určený uzol siete podpíše blok, zašle ho do celej siete
- Uzly siete akceptujú blok iba, ak sú všetky transakcie v bloku validné
- Uzly vyjadria akceptáciu nového bloku tým, že pracujú na vytváraní ďalšieho bloku v reťazi použijúc hash akceptovaného bloku ako predchádzajúci hash.

Uzly siete vždy pokladajú za správny najdlhší chain a pracujú na jeho rozširovaní. Ak dva uzly šíria rôzne verzie ďalšieho bloku súčasne, niektoré uzly dostanú ako prvý jeden z týchto blokov, iné uzly druhý blok. Ak je blok, na ktorom pracujú, nesprávny, uložia si druhú vetvu, lebo sa stala dlhšou. Vtedy zrušia prepojenie na nesprávnu (kratšiu) vetvu reťaze (chainu).

Uzly môžu kedykoľvek opustiť sieť a opäť sa do nej pripojiť akceptovaním najdlhšieho chainu, ktorý vznikol, kým neboli súčasťou siete.

Existujú dva typy blockchain sietí:

- verejná - je to typ otvorenej siete, do ktorej môže pristúpiť a používať ju každý bez toho, aby musel požiadať a získať povolenie. Na prisúpenie do takejto siete postačuje nainštalovanie určitého softvéru, ktorý zabezpečuje komunikáciu a prácu v takejto sieti,
- privátna - je to proprietárna sieť, ktorú vytvorila nejaká spoločnosť, organizácia alebo skupina za účelom vykonávania transakcií. Na prístup do tejto siete je potrebné povolenie.

2.2.6 Ochrana súkromia (Privacy)

Jednou zo základných vlastností blockchainu je to, že všetky transakcie sú verejné. Súkromie pre účastníkov transakcie je možné zabezpečiť tým, že verejné kľúče budú anonymné, t.j. okolie môže vidieť, že niekto posiela transakciu niekomu inému, ale totožnosť odosielateľa a prijímateľa zostane utajená.

Ako ďalšia úroveň bezpečnosti a ochrany súkromia môže byť v blockchaine použité generovanie nových kľúčov pre každú transakciu, čím sa zabezpečí ochrana pred spojením viacerých transakcií ku ich spoločnému vlastníkovi.

2.3 Konsenzuálne algoritmy

Konsenzuálny algoritmus je algoritmus, pomocou ktorého sa zvolí ten, kto určí obsah ďalšieho bloku blockchainu. Vždy, keď je do siete vyslaná nová transakcia, majú uzly siete možnosť zapísať alebo nezapísať tento nový blok do svojej kópie blockchainu. Keď sa väčšina uzlov rozhodne rovnakým spôsobom, je dosiahnutý konsenzus. Dosiahnutie konsenzu v blockchain sieti, kde sa uzly navzájom nepoznajú a nedôverujú si, je kľúčové pre dosiahnutie bezpečnosti blockchainu

[16].

Ďalej uvedieme niektoré konsenzuálne algoritmy, ktoré sú dnes používané v rôznych implementáciách blockchainu.

2.3.1 Proof of Work (PoW) - dôkaz práce

Tento algoritmus sme popísali v kapitole 3.4. Tu uvedieme jeho výhody a nevýhody.

Výhody:

- je overený časom - je už niekoľko rokov implementovaný v blockchaine kryptomeny Bitcoin [17]
- je bezpečný - bezpečnosť a dôveru Bitcoinu sa zatiaľ nepodarilo narušiť napriek niekoľkým pokusom
- je odolný voči kompromitácii až (N/2)-1 uzlov siete, kde N je celkový počet uzlov v sieti
- s použitím tohto algoritmu je možné vybudovať decentralizovaný model blockchainu.

Nevýhody:

- proces ťažby (overovania validity bloku) je veľmi zdĺhavý, trvá cca 10 minút [17]
- v dôsledku toho je obmedzený počet transakcií, ktoré môžu byť validované za jednotku času

- je potrebná veľká výpočtová sila
- spotrebováva sa veľké množstvo energie - na overenie jednej transakcie Bitcoinu sa spotrebuje toľko energie, ako spotrebuje skoro 19 amerických domácností za deň [11]
- spotreba energie na overovanie transakcií stále rastie a bude ďalej narastať.

2.3.2 Proof of Stake (PoS) - dôkaz stávky

Proof of Stake eliminuje problémy Proof of Work algoritmu s potrebou veľkého výkonu a veľkej výpočtovej sily a nahrádza ich stávkou. Pod stávkou rozumieme množstvo meny (vklad), ktorú je jej vlastník (ťažiar) ochotný vsadiť, t.j. dať zadržať na určitú dobu do depozitu. Na oplátku dostane šancu úmernú výške jeho stávky, že bude zostavovať ďalší blok [7]. Sila ťažiara je tým vyššia, čím viac meny vlastní a vsadí. Tomu zodpovedá aj maximálny počet blokov, ktoré môže vyťažiť. Ten, kto vytvorí nový blok, získa odmenu - poplatky, ktoré užívatelia zaplatili pri transakciách (užívatelia musia pri transakciách platiť za ich overenie).

Proof of Stake algoritmus taktiež eliminuje riziko útoku 51%. Pri Proof of Stake algoritme by útočník potreboval vlastniť 51% kryptomeny, aby mohol previesť útok 51%. Tento typ algoritmu však znevýhodňuje majoritného vlastníka na prevedenie tohoto typu útoku. Jednak by bolo ťažké a drahé naakumulovať 51% kryptomeny, ak by sa to však niekomu podarilo, nemal by žiaden záujem útočiť na menu, ktorej väčšinu vlastní. Prevedením útoku na túto menu by oslabil jej hodnotu, a tým by klesla aj hodnota peňazí, ktoré vlastní. Takže väčšinový vlastník bude mať snahu udržiavať sieť bezpečnou [17].

Výhody:

- je bezpečný
- je odolný voči útoku 51%
- s použitím tohto algoritmu je možné vybudovať decentralizovaný model blockchainu
- proces overovania je oveľa rýchlejší, ako pri algoritme Proof of Work
- nie je potrebná veľká výpočtová sila na overovanie [17]
- je energeticky nenáročný, keďže sa nič nevypočítava, na rozdiel od PoW [17].

Nevýhody:

- hlavná nevýhoda PoS je tzv. Nothing-at-stake (nič na vsadenie) problém, kedy v prípade vzniku forku (rozdvojenia reťaze) ťažiari majú motiváciu vsadiť do oboch vetiev reťaze, čo ich nestojí o nič viac a môžu tým získať väčšiu pravdepodobnosť vyťaženia bloku [7]. Dôsledok toho je, že ak sú všetci ťažiari čisto ekonomicky zmýšľajúci, blockchain, aj keď v ňom nie sú žiadni útočníci, nikdy nemusí dosiahnuť konsenzus [3].

2.3.3 Delegated Proof of Stake (DPoS)

Delegated Proof of Stake (DPoS) je variant Proof of Stake algoritmu, ktorý poskytuje vysokú úroveň škálovateľnosti na úkor limitácie počtu producentov blokov v sieti. DPoS je systém, v ktorom pevne určený počet producentov blokov vytvára bloky systémom roundrobin [8].

Producenti blokov sú volení užívateľmi siete, z ktorých každý má počet hlasov proporcionálny počtu peňazí, ktoré v danej mene vlastní. Užívatelia siete môžu

svoje hlasy aj delegovať na iného užívateľa, ktorý bude hlasovať vo voľbách v ich mene. Užívatelia siete môžu overiť, či bloky vytvorené producentmi blokov, spĺňajú pravidlá dané konsenzuálnym algoritmom. Každý používateľ siete môže byť takýmto overovateľom – validátorom [8].

Počet producentov blokov je daný konsenzus pravidlami daného chainu. V známych implementáciách DPoS sa tento počet pohybuje v rozmedzí od 20 do cca 100 [8].

Jeden round-robin cyklus DPoS s N producentmi vyzerá nasledovne:

1. Z množiny kandidátov producentov blokov je zvolených N producentov blokov
2. V poradí i-ty producent podpíše i-ty blok, toto sa opakuje pre i=1 až N [8].

Blok je sfinalizovaný, keď zaňho zahlasuje viac ako dve tretiny producentov blokov. V opačnom prípade sa použije pravidlo najdlhšieho chainu [8].

Výhody:

- DPoS je schopný obhospodáriť priepustnosť transakcií niekoľkonásobne vyššiu ako dnešné Proof of Work algoritmy
- je škálovateľný [8].

Nevýhody:

- je len do istej miery decentralizovaný, čo môže viesť ku snahám formovať kartely alebo uplácať voličov [8].

2.3.4 Practical Byzantine Fault Tolerance (pBFT)

Byzantine Fault Tolerance (BFT) je algoritmus, ktorý distribuovanej počítačovej sieti umožňuje pracovať, ako je požadované, a umožňuje jej dosiahnuť dostatočný konsenzus aj v prípade existencie škodlivých uzlov šíriacich nesprávne informácie ostatným uzlom siete. Jeho úlohou je ochrana proti katastrofickým chybám znižovaním vplyvu, ktorý majú škodlivé uzly na správnu funkčnosť siete a konsenzus, ktorý je dosahovaný ostatnými, tzv. poctivými uzlami v sieti [4].

Practical Byzantine Fault Tolerance je jedna z mnohých optimalizácií BFT algoritmu. Je založená na predpoklade, že počet škodlivých uzlov v sieti nie je väčší alebo rovný tretine všetkých uzlov siete. Čím je viac uzlov v sieti, tým je menšia pravdepodobnosť, aby počet škodlivých uzlov siete dosiahol jednu tretinu, t.j. tým je sieť bezpečnejšia. Pokiaľ počet škodlivých alebo pokazených uzlov v jednom čase nedosiahne jednu tretinu všetkých uzlov, algoritmus efektívne poskytuje životnosť a bezpečnosť siete [4].

Uzly v pBFT sieti sú sekvenčne usporiadané, jeden uzol je líder a ostatné sú označené ako záložné uzly. Všetky uzly komunikujú so všetkými za tým účelom, aby poctivé uzly dosiahli dohodu o stave systému na základe pravidla väčšiny [9].

Pri komunikácii uzly musia overiť, že správy prišli z daného uzla a takisto, že správa nebola zmenená počas prenosu [9].

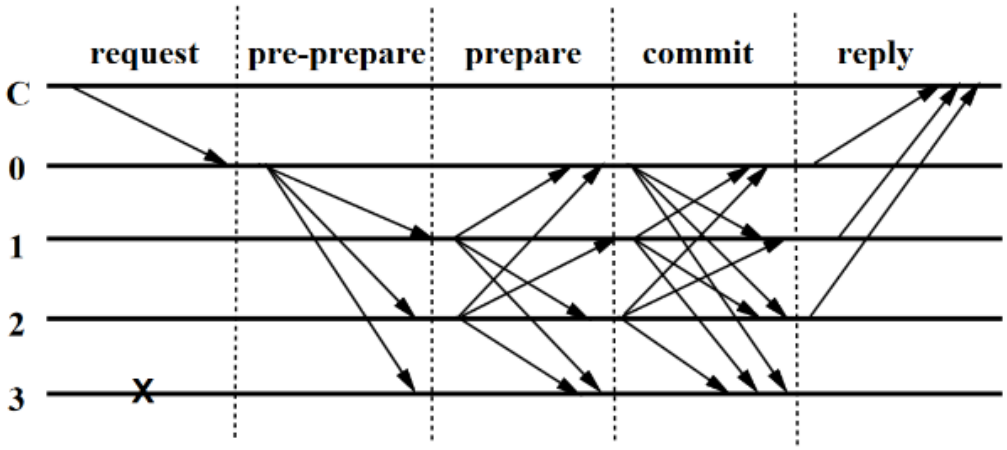
Cyklus v pBFT (nazýva sa pohľad) je rozdelený na 4 fázy:

1. Klient pošle dopyt lídrovi na vyvolanie servisnej činnosti.
2. Líder pošle dopyt na záložné uzly
3. Záložné uzly vykonajú požiadavku a pošlú odpoveď klientovi.

4. Klient čaká f+1 rovnakých odpovedí od rôznych uzlov, kde f je maximálny možný počet škodlivých uzlov [2].

Líder sa mení počas každého cyklu systémom round-robin, ale môže byť zmenený aj protokolom zvaným view change, ak prejde určitý čas bez toho, aby líder poslal dopyt, t.j. ak je neaktívny. Líder môže byť taktiež zmenený, keď sa drvivá väčšina poctivých uzlov zhodne na tom, že líder je škodlivý [2].

Cyklus pBFT je schématicky znázornený na Obrázku č.2.4. Fáze 3 pritom v tomto schématickom znázornení zodpovedajú časti prepare a commit.



Obr. 2.4: Flow diagram algoritmu pBFT

Výhody:

- Definitívnosť transakcií – podstatou pBFT je to, že transakcie sú dohadované počas cyklov a sfinalizované bez potreby mnohonásobného potvrdzovania [9].

- Energetická efektívnosť – na rozdiel od proof-of-work v pBFT k dosiahnutiu konsenzu nie sú potrebné intenzívne výpočty [9].

Nevýhody:

- Nedostatočná škálovateľnosť – pBFT sa stáva neefektívnym pre veľké siete, pretože pri tomto algoritme každý uzol musí komunikovať s každým, aby sieť bola bezpečná. S rastúcim počtom uzlov v sieti tým pádom enormne narastá komunikácia medzi uzlami [9].
- Útoky Sybil – pBFT je náchylné na útok Sybil, kde jeden nepriateľ vytvorí alebo ovláda veľké množstvo uzlov predstierajúc, že je to viacero nezávislých strán. Nebezpečenstvo tejto hrozby eliminujú siete s väčším počtom uzlov, čo môže mať za následok problémy so škálovateľnosťou. Preto je zvyčajne potrebné pBFT kombinovať s iným konsenzus mechanizmom, ako napr. proof-of-work, ktorý sa používa až po určitom počte blokov, napr. 100 [9].

2.3.5 Proof of Authority (PoA) - dôkaz autority

Pri tomto algoritme sú transakcie validované tzv. validátormi, čo sú schválené účty v tomto systéme - môžeme ich považovať za niečo ako “administrátorov”. Validátori sú autority, od ktorých ostatné uzly v sieti dostávajú svoju “pravdu”. Uzly siete (užívatelia) získavajú právo stať sa validátormi získavaním pozitívnej reputácie k ich identite. Taktiež užívatelia môžu získavať negatívnu reputáciu. Je snahou jednotlivých uzlov, aby získali čo najviac pozitívnej reputácie a udržali si post validátora transakcií.

PoA nevyužíva žiaden spôsob "minovania". Používa sa napríklad pri kryptomene Ethereum. PoA je primárne určený a optimalizovaný pre privátne siete, no je takisto vhodný aj pre verejné siete.

Výhody:

- tento algoritmus má vysokú priepustnosť
- je energeticky nenáročný
- overovanie transakcií je veľmi lacné

Nevýhody:

- je centralizovaný - vo forme autorít, čo v podstate narúša základnú myšlienku kryptomien a blockchainu

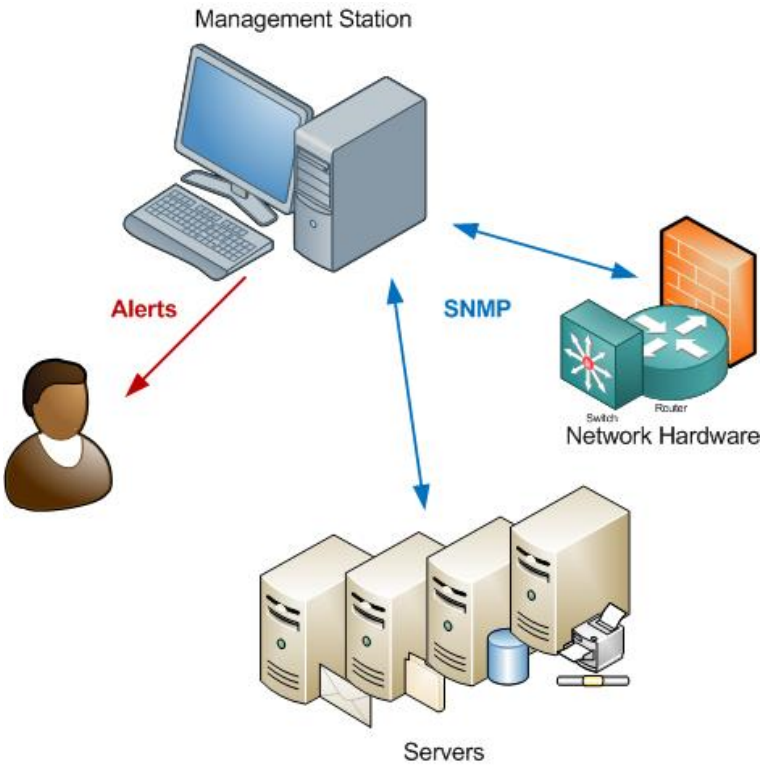
2.4 Komunikačné a manažovacie protokoly

V nasledujúcej kapitole si opíšeme základné najčastejšie používané komunikačné a manažovacie protokoly.

2.4.1 Simple network management protocol (SNMP)

Simple network management protokol (SNMP) je protokol určený na manažovanie a monitorovanie zariadení a softvéru. Drvivá väčšina výrobcov podporuje tento protokol, takže je možné jeho široké použitie. SNMP k svojej funkčnosti vyžaduje iba dve komponenty: manažovaciu stanicu a agenta, resp.agentov.

Manažovacia stanica je sofvér, ktorý zbiera informácie zo siete, t.j. od agentov. Agent je softvér, ktorý beží na hardvéri alebo softvéri, ktorý chceme monitorovať a manažovať. Agenti sú väčšinou zabudovaní v monitorovanom hardvéri alebo softvéri priamo od výrobcu a je ich potrebné iba aktivovať. Agent môže pracovať v dvoch režimoch. V prvom zbiera informácie a keď si ich manažovacia stanica vyžiada, pošle jej ich. V druhom režime agent posiela informácie manažovacej stanici bez toho, aby si ich vyžiadala, toto sa deje napr. v prípade vzniku chybového stavu [10]. Manažovacia stanica môže posielať alerty užívateľovi o mimoriadnych stavoch monitorovaných komponent formou e-mail, SMS-notifikácie, atď. Viď. Obrázok 2.5 [10].



Obr. 2.5: Komunikácia v SNMP protokole s pomocou manažovacej stanice.

SNMP je asymetrický (request-reply) protocol zabezpečujúci komunikáciu medzi management stanicou a agentom. SNMP štandardne beží nad protokolom UDP na portoch 161 a 162, ale je možná aj prevádzka nad protokolom TCP. Táto komunikácia má formu Protokol Data Unit (PDU) [5].

Základné operácie, t.j. dopyty a odpovede, používané v komunikácii, sú:

- GET: zasielaný SNMP managerom agentovi na získanie hodnoty parametra identifikovaného jeho OID identifikátorom v MIB
- RESPONSE: zasielaný agentom SNMP managerovi ako odpoveď na GET dopyt. Obsahuje hodnotu požadovaného parametra
- GETBULK: zasielaný SNMP managerom agentovi na získanie veľkých tabuliek dát vykonaním viacerých GETNEXT príkazov
- GETNEXT: zasielaný SNMP managerom agentovi na získanie hodnoty ďalšej OID v MIB hierarchii
- SET: zasielaný SNMP managerom agentovi na vykonanie konfiguračných zmien alebo príkazov
- TRAP: asynchrónny alert zasielaný agentom SNMP managerovi na indikáciu významného stavu, ako je chyba alebo zlyhanie, ktoré nastalo [13].

Hlavným účelom SNMP správ je nastavovať (set) alebo monitorovať (get) parametre na SNMP agentovi. V SNMP agentovi sú parametre zoradené do stromu, na identifikáciu konkrétneho parametra v strome SNMP používa Object Identifier (OID). OID je reprezentovaný číslami oddelenými bodkou, napr. „1.3.6.1.4.1.2680.1.2.7.3.2.0“. OID sa skladá z dvoch častí. Prvá (1.3.6.1.4.1.2680.1.2.7.3.2) reprezentuje samotný

objekt, napr. sieťový adaptér servra. Druhá (0) reprezentuje konkrétnu inštanciu daného objektu v agentovi, t.j. ak má daný server viac sieťových adaptérov, budú postupne označené poradovými číslami od 0. V tomto prípade má server len jeden sieťový adaptér, označený ako „0“ [1].

Každý SNMP agent má zoznam všetkých svojich objektov, ktorý sa nazýva Management information database (MIB). Pre každý objekt SNMP agenta MIB poskytuje meno, OID, dátový typ, read-write povolenia a krátky popis. Majúc informácie z MIB databázy môže SNMP manager posielať SNMP správy za účelom nastavenia alebo zistenia hodnoty hociktorého z parametrov SNMP agenta [1].

SNMP protokol prešiel značným vývojom, vďaka čomu existujú už 3 jeho štandardizované verzie (1, 2c, 3), z ktorých každá má svoje výhody a nevýhody a tým pádom aj oblasti svojho využitia. Väčšina dnešných zariadení a softvérov už podporuje všetky 3 verzie tohto protokolu.

Verzia 1 je najjednoduchšia a najzákladnejšia zo všetkých troch verzií. Jej použitie v dnešnej dobe je obmedzené na prípady, kedy potrebujeme zabezpečiť podporu starších zariadení, ktoré nemajú implementované novšie verzie protokolu SNMP [10].

Verzia 2c pridala ku základnej verzii 1 rôzne vylepšenia, ako napr. podporu pre „Informs“. Z tohto dôvodu sa táto verzia stala najrozšírenejšou. Avšak obe verzie (1 aj 2c) majú rovnakú slabosť v zabezpečení protokolu. Tá spočíva v tom, že Community stringy, čo je ekvivalent hesiel, sú posielané v čitateľnej forme (cleartext), t.j. nie je použité šifrovanie, a taktiež tieto verie nemajú podporu autentifikácie. To vytvára riziko možnosti kompromitácie použitých community stringov. Toto je veľká slabina oboch verzií SNMP s ohľadom na možnosti, ktoré SNMP má na zmenu konfigurácie manažovaných zariadení a softvéru [10].

Posledná verzia SNMP, verzia 3, pridala bezpečnostné možnosti, ktoré prekonávajú slabiny jeho predchádzajúcich verzií. Táto verzia by sa z tohto dôvodu mala používať, ak je to možné, najmä v prípade, ak je v pláne prenášať informácie cez nezabezpečené prepojenia. Avšak zvýšená bezpečnosť prináša aj zložitejšiu konfiguráciu tejto verzie protokolu [10].

2.4.2 MQTT a CoAP

Message Queue Telemetry Transport (MQTT) protokol je protokol aplikačnej vrstvy, určený primárne pre zariadenia, ktoré majú obmedzené zdroje, t.j. malú batériu, málo pamäte, nízky výpočtový výkon. Používa topic-based publish-subscribe architektúru, čo znamená, že keď niekto publikuje správu v nejakej téme, všetci, ktorí túto tému sledujú, obdržia danú správu. Podobne ako HTTP protokol aj MQTT využíva TCP a IP protokoly ako podvrstvy. Avšak narozdiel od HTTP je MQTT navrhnutý tak, aby mal menšiu hlavičku a samozrejme aj menšie režijné náklady s ním spojené [15].

Tri Quality of Service (QoS) levely sa starajú o spoľahlivosť doručenia správy. Nultý level zabezpečuje, že správa je prijatá najviac raz a nie je potrebné žiadne potvrdenie o prijatí. Pri prvom leveli je potvrdenie o prijatí potrebné a zabezpečuje, že správa je doručená aspoň raz. Pri druhom leveli sa uzatvára tzv. four-way handshake, ktorý zabezpečí, že správa je doručená práve raz [15].

Constrained Application Protocol (CoAP) je takisto protokol aplikačnej vrstvy na komunikáciu zariadení s obmedzenými zdrojmi. Využíva architektúru REST s podporou request-response modelu ako pri HTTP. Rovnako ako protokol MQTT podporuje publish-subscribe architektúru, ktorá však už nie je topic-based ale využíva Universal Resource Identifier (URI). URI funguje podobne ako publish-

subscribe architektúra s tým rozdielom, že keď niekto publikuje správu na URI, ostatní, ktorí sú prihlásení na odber URI dostanú o tomto notifikáciu [15].

Rozdiely:

- CoAP:
  - síce využíva nespoľahlivý UDP protokol, avšak má implementovaný vlastný mechanizmus potvrdzovania správ. Má dva typy správ - tie ktoré vyžadujú potvrdenie a tie ktoré nevyžadujú potvrdenie [15].
  - neposkytuje diferencovateľné QoS [15]
- MQTT:
  - je spoľahlivý, pretože využíva TCP [15]
  - poskytuje tri úrovne QoS [15]

Rozdiely sú pekne prehľadné v tabuľke na obrázku č. 2.6 [15].

	MQTT	CoAP
Application Layer	Single Layered completely	Single Layered with 2 conceptual sub layers (Messages Layer and Request Response Layer)
Transport Layer	Runs on TCP	Runs on UDP
Reliability Mechanism	3 Quality of Service levels	Confirmable messages, Non-confirmable messages, Acknowledgements and retransmissions
Supported Architectures	Publish-Subscribe	Request-Response, Resource observe/Publish-Subscribe

Obr. 2.6: Rozdiely medzi MQTT a CoAP

2.5 Zhodnotenie analýzy

V analýze sme sa venovali témam ako sú IoT, Smart home, komunikačné protokoly, no najmä blockchainu a konsenzuálnym algoritmom. Z analýzy vyplýva, že najvhodnejším riešením na zabezpečenie IoT siete s viacerými zariadeniami je blockchain.

Blockchain je vhodný hneď z viacerých dôvodov. Týmito sú napríklad decentralizácia – neexistuje centrálny uzol, ktorý by všetko riadil, resp. distribuovanosť –

blockchain je rozdistribuovaný medzi všetky uzly v sieti, každý uzol má jeho kópiu. Z tohto vyplýva ďalší dôvod, prečo je blockchain vhodný a tým je bezpečnosť – tým, že je distribuovaný, je veľmi ťažké urobiť v ňom nejaké zmeny. Na spravenie zmeny v blockchaine by útočník musel upraviť všetky inštancie tohto ledger na všetkých zariadeniach.

Z analýzy nám vyplýva, že blockchain musí byť hlavne rýchly a nenáročný na zdroje, akými sú batéria a pamäť. Preto sa k takejto implementácii blockchainu a konsenzuálneho algoritmu budeme snažiť priblížiť.

Kapitola 3

Špecifikácia požiadaviek

Z predchádzajúcej analýzy sme zistili, že naše riešenie bude musieť spĺňať určité kritériá. Tie si v stručnosti zhrnieme v tejto kapitole. Vzhľadom na to, že naša sieť bude pripojená do internetu, bude potreba túto sieť patrične zabezpečiť. Vhodným zabezpečením pre takéto riešenie je blockchain s implementáciou konsenzuálneho algoritmu.

Požiadavky na blockchain s konsenzuálnym algoritmom a komunikažný, resp. manažovací protokol:

- musí byť rýchly, keďže ho bude využívať veľa zariadení a je možné, že sa doňho bude zapisovať často
- musí byť energeticky nenáročný, keďže IoT zariadenia majú obmedzené zdroje energie
- nesmie zaberať veľa pamäti, kedže malé IoT zariadenia majú obmedzenú pamäť - to znamená, že bude treba zvážiť, ktoré dáta je a ktoré dáta nie je vhodné doňho ukladať

Kapitola 4

Návrh riešenia

4.1 Blockchain

V našom projekte sme sa rozhodli použit blockchain ako prostriedok, s ktorého použitím dosiahneme zvýšenie bezpečnosti siete a to vďaka jeho vlastnostiam ako sú distribuovanosť, transparentnosť, použitie šifrovania, či možnosť dohľadania histórie každej transakcie. S použitím blockchainu je možné v sieti presne identifikovať škodlivé zariadenie.

Rolu uzlov blockchainu, na ktorých bude uchovávaný blockchain záznam s prevedenými transakciami, budú plniť všetky zariadenia v sieti. Ich úlohami v súvislosti s blockchainom bude:

- potvrdzovať transakcie
- zapisovať transakcie do blockchainu
- uchovávať blockchain kvôli overeniu histórie transakcií
- poskytovať informácie z blockchainu o prevedených transakciách.

4.2 Konsenzuálny algoritmus

Súčasťou blockchainu je vždy konsenzuálny algoritmus, ktorý umožňuje uzlom siete dosiahnuť konsenzus. Pre náš projekt sme sa rozhodli implementovať dva konsenzuálne algoritmy:

- Practical byzantine fault tolerance
- Proof of Stake.

Základnou vlastnosťou oboch týchto algoritmov je energetická efektívnosť, čo je jednou z požiadaviek na riešenie nášho projektu. Algoritmy sme sa rozhodli upraviť tak, aby spĺňali aj ďalšie požiadavky, ktorými sú:

- rýchlosť algoritmu
- nenáročnosť na pamäť
- nenáročnosť na úložný priestor

Spôsob úprav oboch algoritmov je popísaný v ďalších kapitolách.

4.2.1 Upravený konsenzuálny algoritmus pBFT

Požiadavky na konsenzuálny algoritmus sa v prípade pBFT algoritmu budeme snažiť splniť úpravou jeho cyklu. Upravený cyklus vyzerá nasledovne:

1. Klient pošle dopyt na vyvolanie servisnej činnosti priamo na všetky uzly siete
2. Uzly vykonajú požiadavku a pošlú odpoveď klientovi
3. Klient čaká f rovnakých odpovedí od rôznych uzlov, kde f nami stanovená hranica akceptovateľnosti.

Ďalšie úpravy algoritmu spočívali v tom, že:

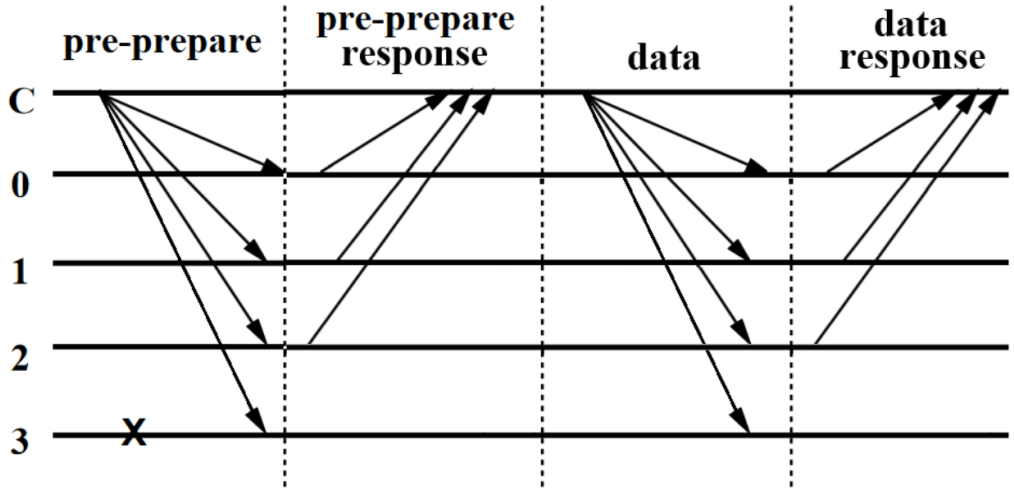
- použijeme kontrolný súčet (checksum) namiesto podpisovania transakcií, na overenie integrity dát

- v blockchaine nebudeme uchovávať celé zasielané správy, ale len informácie súvisiace s odoslaním správy

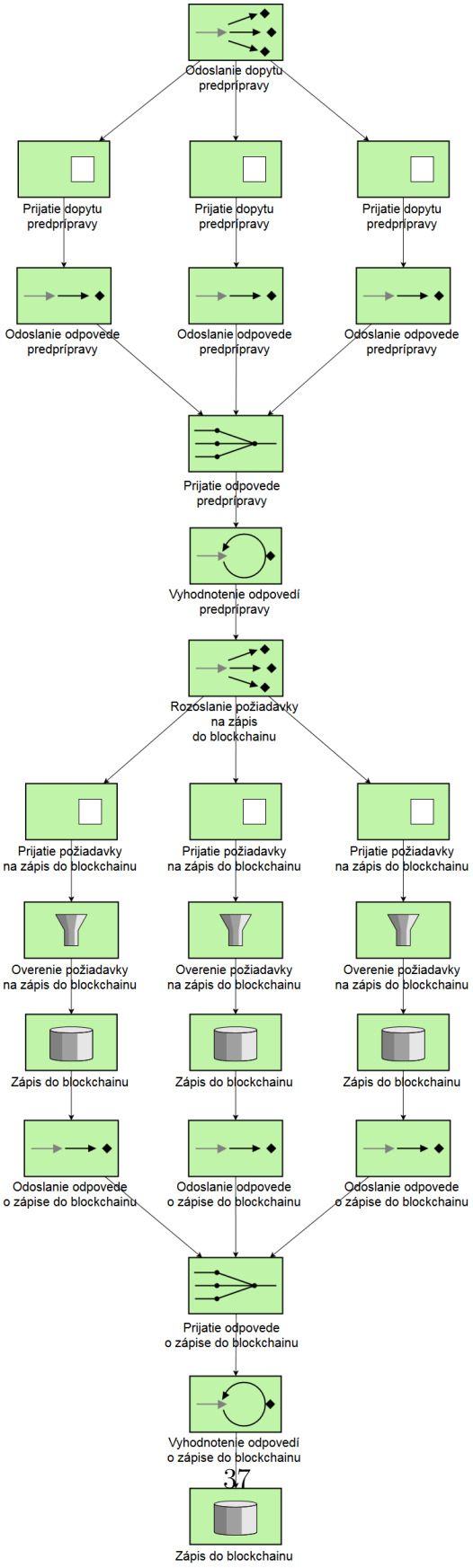
- zjednodušíme vykonanie požiadavky klienta uzlami siete. Toto zjednodušenie spočíva v tom, že na rozdiel od pBFT algoritmu, kde uzol siete overuje prijatú požiadavku voči ostatným uzlom siete jej opätovným rozoslaním do celej siete, v nami upravenom algoritme uzol siete overí správnosť prijatých dát vypočítaním kontrolného súčtu (checksum) a jeho porovnaním s kontrolným súčtom, ktorý mu zaslal klient pri zaslaní transakcie

- transakcie nebudeme zapisovať do blokov.

Nami upravený algoritmus pBFT nazveme pre účely nášho projektu Upravený pBFT (UpBFT). Jeho flow diagram je zobrazený na Obrázku 4.1. Vývojový diagram je zobrazený na obrázku 4.2.



Obr. 4.1: Diagram cyklu upraveného UpBFT algoritmu



Obr. 4.2: Vývojový diagram upraveného UpBFT algoritmu

4.2.2 Upravený konsenzuálny algoritmus PoS

Proof of Stake algoritmus je ďalší algoritmus, ktorý sme sa rozhodli v našej práci implementovať. V oblasti kryptomien, kde bol zavedený, je založený na vlastníctve danej kryptomeny. Čím viac kryptomeny daný uzol siete vlastní, tým viac je schopný vsadiť ako dôkaz na validáciu bloku a tým mať väčšiu šancu, že bude validovať daný blok. Týmto uzol zvyšuje celkový počet blokov, ktoré zvalidoval, a samozrejme tým pádom aj sumu, ktorú za danú činnosť zarobí. V oblasti IoT nefunguje žiadna kryptomena, ktorú by bolo možné použiť na dôkaz stávky. Preto sme sa rozhodli, že výšku stávky budeme u každého uzla generovať generátorom náhodných čísel.

Vybraný uzol, označme ho Decider (rozhodovač), bude vygenerované hodnoty, ktoré mu jednotlivé uzly v pravidelných intervaloch zasielajú, vyhodnocovať a určovať validátora transakcií.

Prácu nami upraveného PoS algoritmu možno zapísať do nasledovných krokov:

1. Decider pošle požiadavku na zaslanie stávok (stake) na všetky uzly siete
2. Uzly siete vygenerujú náhodné číslo - svoju stávku do volieb Validátora transakcií a pošlú ich Deciderovi
3. Decider určí Validátora transakcií a pošle túto informáciu na všetky uzly siete
4. Uzol siete pošle požiadavku na zápis do blockchainu Validátorovi
5. Validátor požiadavku na zápis overí, sformuluje transakciu a zapíše ju do blockchainu, následne pošle túto transakciu na všetky uzly siete

6. Uzly overia transakciu a zapíšu ju do svojich blockchainov

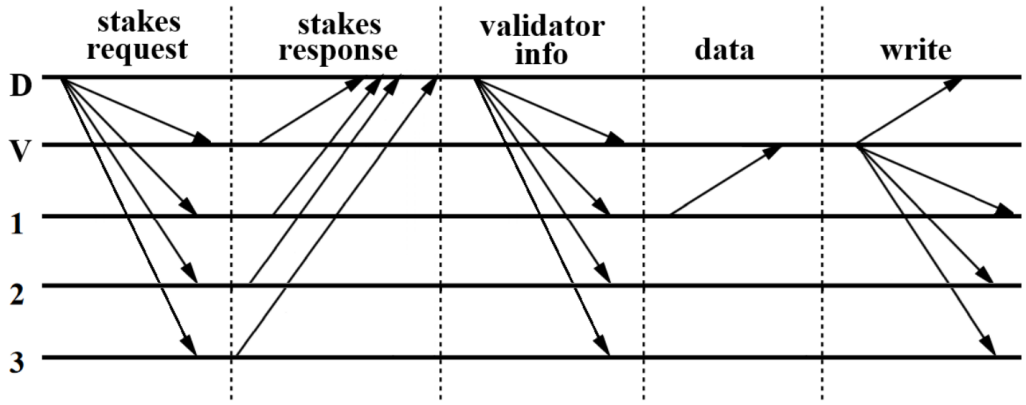
Voľby Validátora, t.j. prvé tri kroky uvedené vyššie, by sa mali vykonávať pred validáciou každého bloku. V našom prípade, keďže z transakcií nevytvárame bloky, resp. za jeden blok považujeme jednu transakciu, budeme voľby prevádzať raz za nejakú časovú jednotku, napr. za 60 sekúnd.

Rovnako, ako v prípade pBFT, aj pri tomto algoritme upravíme niektoré atribúty na naše podmienky:

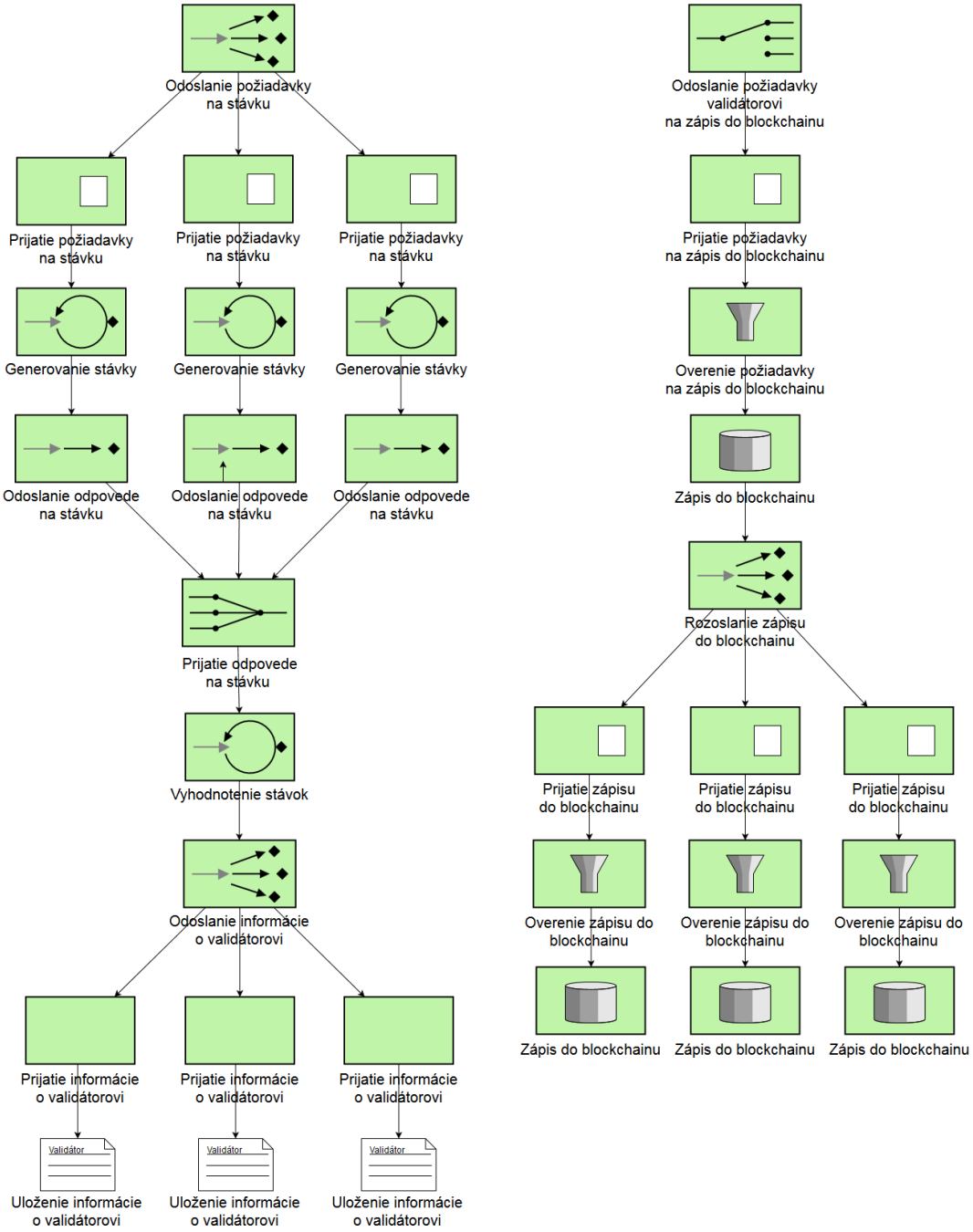
- použijeme kontrolný súčet (checksum) namiesto podpisovania transakcií, na overenie integrity dát
- v blockchaine nebudeme uchovávať celé zasielané správy, ale len informácie súvisiace s odoslaním správy
- transakcie nebudeme zapisovať do blokov.

Ako blockchain úložisko opäť použijeme súbor na filesystéme uzla. Blockchain sa bude nachádzať na všetkých uzloch, aby každý z uzlov mohol validovať a zapisovať transakcie do blockchainu v prípade, že vyhrá stávku.

Nami upravený algoritmus PoS nazveme pre účely nášho projektu Upravený PoS (UPoS). Flow diagram algoritmu UPoS algoritmu je zobrazený na Obrázku č. 4.3. Vývojový diagram je zobrazený na obrázku 4.4.



Obr. 4.3: Diagram cyklu upraveného UPoS algoritmu



Obr. 4.4: Vývojový diagram upraveného UPoS algoritmu

4.3 Transakcie

Blockchain používaný pri kryptomenách prenáša a uchováva transakcie, ktorými sú prevody kryptomeny od jedného vlastníka ku druhému. Pod transakciami blockchainu v IoT sieťach rozumieme výmenu dát, ktoré posiela jeden uzol siete druhému, pričom sa môže jednať o akékoľvek dáta, ktoré musí IoT zariadenie odoslať alebo prijať. V našom projekte sa jedná o dva druhy dát:

- dáta namerané senzormi IoT zariadenia, ktoré toto zariadenie odosiela serveru na spracovanie
- konfiguračný súbor, ktorý zasiela server IoT zariadeniu.

Tieto dáta sa líšia smerom prenosu, veľkosťou i frekvenciou zasielania. Kým senzormi namerané hodnoty sú z IoT zariadenia zasielané v pravidelných intervaloch aj niekoľkokrát za minútu, konfiguračný súbor je zasielaný nepravidelne, raz za deň, týždeň, či ešte dlhšie obdobie. Veľkosť i frekvencia zasielania dát má vplyv ako na stav siete, tak i na stav systémov zúčastnených na komunikácii.

4.3.1 Ukladanie transkacií do blockchainu

Blockchain, respektíve samotné ledgre, navrhujeme v našom projekte realizovať ako textový súbor na filesystéme uzla siete. V súbore bude každá transakcia reprezentovaná samostatným riadkom, v ktorom sa budú uchovávať údaje o transakcii. Každú transakciu budú v blockchaine popisovať nasledovné údaje:

- druh zasielaných dát (viď Tabuľka 4.1)
- IP adresa a MAC adresa odosielateľa

- čas zaslania
- kontrolný súčet zaslanej správy.

Štruktúra jednej transakcie v blockchaine je uvedená v Tabuľke 4.2. Oddeľovačom medzi jednotlivými údajmi transakcie bude znak „;“.

Druh dát	**Označenie v blockchaine**	**Popis**
Konfigurácia	C	Konfiguračný súbor posielaný zo servera na zariadenia Arduino
Dáta	D	Dáta s nameranými hodnotami zasielané z Arduino zariadenia serveru

Tabuľka 4.1: Druhy dát zasielaných v sieti

Poradie položky	**Položka**	**Dĺžka položky [Byte]**	**Popis**
1	Druh dát	1	Druh dát zasielaných v danej transakcii (viď Tabuľka 4.1)
2	IP odosielateľa	7 - 15	IP adresa odosielateľa
3	MAC odosielateľa	12	MAC adresa odosielateľa - každá zo 6 častí je v hexadecimálnom tvare, medzi časťami nie sú dvojbodky
4	Timestamp	10	Čas zaslania správy vo formáte timestamp
5	CRC	8	Kontrolný súčet - checksum

Tabuľka 4.2: Štruktúra transakcie

4.4 Komunikačný protokol

Na použitie posielania správ v sieti sme sa po analýze rozhodli vyvinúť vlastný komunikačný protokol. Hlavným dôvodom pre to bol fakt, že sme dospeli k názoru, že komunikačný protokol vyvinutý presne pre účely nášho projektu bude najlepšie spĺňať všetky identifikované požiadavky:

- rýchlosť
- nenáročnosť na spotrebu batérie
- nenáročnosť na pamäť
- decentralizovanosť

Nami vyvinutý protokol bude založený na nespojovanom protokole transportnej vrstvy UDP, bližšie je tento protokol popísaný v ďalších kapitolách.

4.4.1 Komunikačný protokol pre UpBFT algoritmus

Rozhodli sme sa vytvoriť veľmi jednoduchý sieťový protokol, ktorý je založený na UDP komunikácii a je úzko spätý práve s nami implementovaným algoritmom UpBFT.

Hlavička protokolu pozostáva prakticky z dvoch bytov. Prvý byte značí, o akú správu sa jedná (viď. Tabuľka 4.3). Druhý byte je len oddeľovač (znak ";"), ako tomu bolo aj v prípade dát.

Čo sa týka päty, nemôžme povedať, že protokol obsahuje samostatnú pätu, no nemôžme však povedať, že žiadnu neobsahuje. Ide o pole CRC32, ktoré evidujeme v podstate vrámci dát, no využíva sa aj na úrovni protokolu. Za iných okolností by

bolo korektné, keby sa CRC, ktoré chceme v zapisovať v blockchaine vypočítalo z dát a prilepilo sa k nim a následne sa z celého tohto stringu vypočítalo CRC na úrovni protokolu. V tomto prípade, keďže chceme dosiahnuť, aby bol protokol čo najjednoduchší a najúspornejší, budeme využívať jedno pole CRC pre oba účely.

Toto pole budeme zapisovať aj do blockchainu a budeme ho využívať aj na úrovni protokolu na zistenie neporušenosti dát počas prenosu a na prípadné znovuvyžiadanie správy.

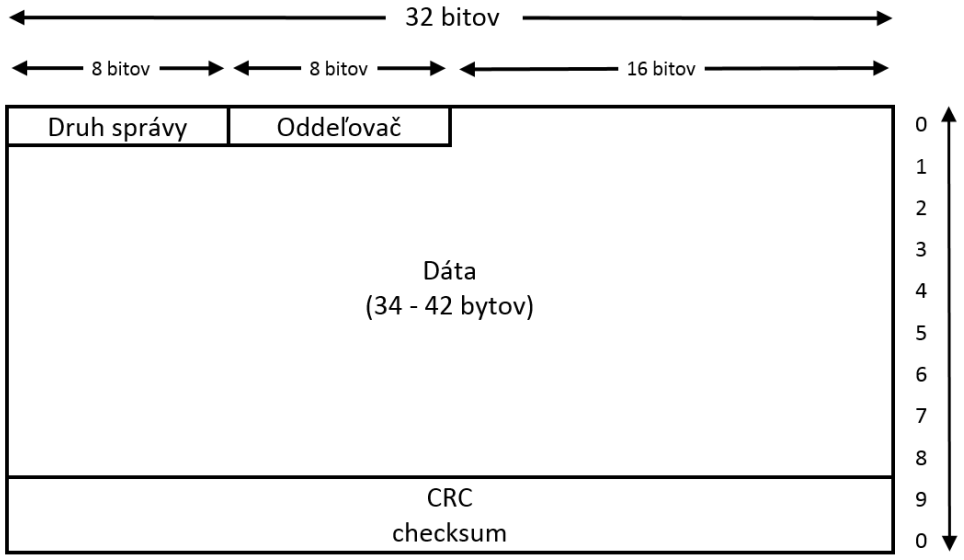
Znovuvyžiadanie, resp. znovuodoslanie správy funguje v našom protokole v dvoch prípadoch:

- pri pred-prípravnom requeste - keď sa klientovi nevrátia aspoň 2 / 3 potvrdzujúcich odpovedí, klient rozošle pred-prípravnú požiadavku znova, ak sa mu nevráti požadovaný počet odpovedí ani na druhý krát, požiadavka sa zahodí

- pri požiadavke na zápis do blockchainu, v prípade, že klient odošle zariadeniu požiadavku a táto dorazí na zariadenie porušená, t.j. nesedí CRC, zariadenie si od klienta znovuvyžiada správu, toto sa tiež opakuje iba raz, rovnako, ako vo vyššie popisovanom prípade, po druhom neúspešnom pokuse sa požiadavka zahodí.

Druh správy	**Označenie v protokole**	**Popis**
pred-príprava	1	Požiadavka odoslaná do siete, na zistenie, či sú zariadenia pripravené na vykonanie zápisu do blockchainu.
pred-príprava odpoveď	2	Potvrdenie od zariadení na správu č. 1.
požiadavka	3	Požiadavka na zápis do blockchainu.
odpoveď	4	Potvrdenie od zariadení na správu č. 3.

Tabuľka 4.3: Druhy správ v protokole UpBFT



Obr. 4.5: Formát správy nášho protokolu

4.4.2 Komunikačný protokol pre UPoS algoritmus

Pri implementácii UPoS algoritmu použijeme obdobný komunikačný protokol ako pri UpBFT algoritme. Bude rovnako založený na UDP komunikácii a takisto bude úzko spätý s algoritmom UPoS.

Hlavička protokolu bude pozostávať z dvoch bytov. V prvom byte bude uvedený druh zasielanej správy podľa Tabuľky 4.4. Druhý byte bude oddeľovač (znak ";"), ako tomu je aj v prípade dát.

Päta protokolu je rovnaká ako v prípade UpBFT algoritmu, preto ju ďalej nepopisujeme.

Druh správy	**Označenie v protokole**	**Popis**
stávka požiadavka	1	Požiadavka na zaslanie stávok Deciderovi odoslaná do celej siete.
stávka odpoveď	2	Zaslanie stávky Deciderovi, odpoveď na správu č.1
info o validátorovi	3	Zaslanie informácie o tom, kto je validátor, do celej siete.
dáta	4	Požiadavka na zápis dát do blockchainu zaslaná validátorovi.
zápis do blockchainu	5	Validátor po overení požiadavky zasiela do celej siete na zápis do blockchainu.

Tabuľka 4.4: Druhy správ v protokole UPoS

4.5 Používateľské rozhranie

Program nebude obsahovať žiadne používateľské rozhranie a nebude s používateľom nijako interagovať, ani graficky, ani cez konzolu. Pôjde o samostatnú aplikáciu bežiacu na pozadí bez používateľských vstupov alebo výstupov smerovaných používateľovi.

Po pripojení zariadenia k počítaču cez seriálový port je možné sledovať kontrolné / pomocné výpisy, akými sú hlavne nastavenie siete, úspešné pripojenie zariadenie k sieti, inicializácia SD karty a podobne.

Kapitola 5

Implementácia

Riešenie nášho projektu sme implementovali na open-source elektronických prototypových zariadeniach Arduino, konkrétne Arduino MKR WiFi 1010. Implementácia riešenia bola realizovaná použitím programovacieho jazyka Arduino (APL), ktorý umožňuje ovládanie Arduino základnej dosky a pripojených rozhraní, senzorov a modulov. Na vývoj zdrojového kódu v APL bolo použité Arduino integrované vývojové prostredie (IDE) vo verzii 1.8.8. Programy boli písané v jazyku C/C++, pričom sme použili funkcie obsiahnuté v knižniciach funkcií Arduino IDE. Použité knižnice uvádzame v nasledujúcej kapitole.

Nami vyvinutý softvér (v prípade oboch algoritmov) zaberal približne 15 - 20 % z celkovej dostupnej pamäte zariadenia, zvyšnú časť využíval kolega na beh ním vyvinutého kódu na spracovanie dát zo senzorov v rámci jeho projektu.

Arduino zariadenia sme mali zapojené v sieti popísanej v kapitole 5.2.

5.1 Použité knižnice

V tejto kapitole uvádzame knižnice Arduino IDE vývojového prostredia použité pri implementácii riešenia nášho projektu.

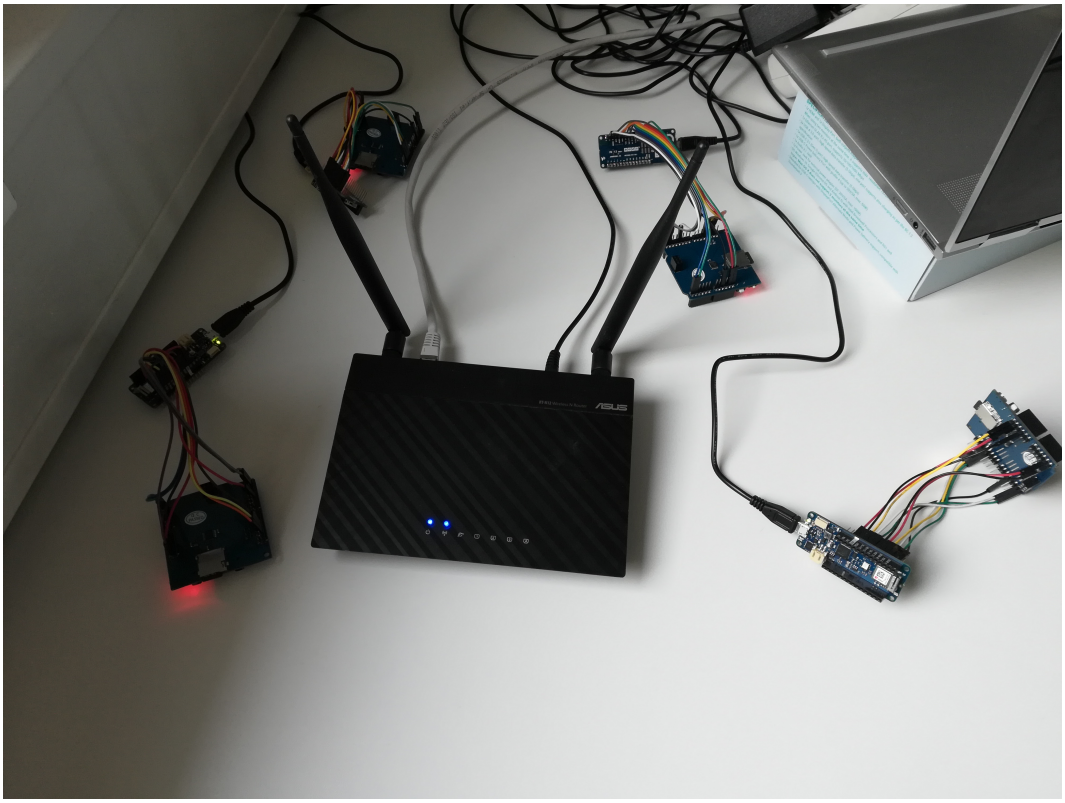
- SPI.h
- WiFiUdp.h
- SD.h
- WiFiNINA.h
- CRC32.h
- ArduinoSTL.h

5.2 Testovacia sieť

Pre náš projekt sme vytvorili testovaciu sieť zobrazenú na obrázku č. 5.1. V sieti sa nachádzali 4 zariadenia Arduino v konfigurácii uvedenej v tabuľke č. 5.1. Po spojení prácí s kolegami bude v sieti 7 Arduino zariadení a ku každému Arduino zariadeniu budú pripojené senzory, ktoré budú merať vybrané parametre prostredia. V sieti bude taktiež zapojený server, určený na spracovanie nameraných hodnôt senzormi Arduino zariadení, ktoré mu za týmto účelom v pravidelných intervaloch tieto hodny budú zasielať. Zariadenia a server komunikujú prostredníctvom WiFi siete.

Softvér blockchainu vyvinutý v našom projekte bol nasadený na všetky zariadenia Arduino, aby bola dosiahnutá distribuovanosť, ako základný princíp blockchain technológie. Zariadenia navzájom komunikovali a dosahovali konsenzus po-

užitím jedného z implementovaných algoritmov UpBFT alebo UPoS.



Obr. 5.1: Fyzické rozloženie zariadení v testovacej sieti

Označenie zariadenia	**Popis zariadenia**	**Počet kusov v sieti**
ARDUINO MKR WIFI 1010	mikrokontrolér s WiFi modulom	4
SD CARD SHIELD V3.0	modul na pripojenie SD karty	4
SD CARD SHIELD V3.0	modul na pripojenie SD karty	4
MICROUSB Adapter 5V/2A	zdroj napätia	4
Kingston microSDHC 16GB class 4	mikro-SD karta	4
Asus RT-N12vD	Bezdrôtový router / prístupový bod / repeater	1

Tabuľka 5.1: Konfigurácia zariadení v sieti

Rovnako, ako tomu je pri pBFT, aj UpBFT sme založili na predpoklade, že počet škodlivých uzlov v sieti nie je väčší alebo rovný jednej tretine všetkých uzlov siete. Keďže celkový počet uzlov našej siete bol 4 a z toho jedno zariadenie sa správalo ako klient, predpokladaný maximálny počet škodlivých uzlov siete bol 0. Z tohto dôvodu sme akceptovanú hranicu posunuli o niečo nižšie a predpokladali sme, že počet škodlivých uzlov v sieti nie je väčší ako jedna tretina. To znamená, že v prípade našej siete to mohol byť 1 chybný uzol. Takže pri overovaní transakcií klient čakal na 2 rovnaké odpovede od uzlov siete, keď ich dostal, považoval transakciu za overenú.

5.3 Moduly systému

5.3.1 Moduly systému pri UpBFT algoritme

Náš systém je rozdelený do dvoch súborov / modulov s názvami pBFT a wifi.

V module wifi sú funkcie, ktoré inicializujú WiFi adaptéry Arduino zariadení a taktiež ich pripoja ku predvolenej WiFi sieti. Ďalšie funkcie slúžia na zobrazenie parametrov WiFi siete a sieťových nastavení WiFi adaptérov Arduino zariadení.

Modul pBFT obsahuje hlavnú logiku celého vykonávania algoritmu. Nachádzajú sa v ňom funkcie na odosielanie dát, prijímanie paketov a ich spracovávanie a vyhodnocovanie a takisto funkcia na samotné zapísanie do blockchainu.

Na Obrázku 5.2 sú znázornené moduly systému použité pri implementovanom UpBFT algoritme a dáta, ktoré si medzi sebou vymieňajú.

5.3.2 Moduly systému pri UPoS algoritme

Aj pri UPoS algoritme je systém rozdelený do dvoch súborov / modulov s názvami PoS a wifi.

Modul wifi bol popísaný pri algoritme UpBFT, preto ho ďalej nepopisujeme.

Modul PoS, podobne ako pri algoritme UpBFT, pozostáva z funkcií, ktoré vykonávajú hlavnú logiku tohto algoritmu. Sú to opäť funkcie na odosielanie a spracovávanie dát a zápis do blockchainu, ďalej sú to funkcie na rozosielanie informácií o validátorovi transkacií a na vyhodnocovanie týchto informácií.

Na Obrázku 5.3 sú znázornené moduly systému použité pri implementovanom UPoS algoritme a dáta, ktoré si medzi sebou vymieňajú.

5.3.3 Funkcie použité pri UpBFT algoritme

Uvádzame zoznam a krátky popis vyvinutých funkcií, ktoré sú súčasťou implementácie algoritmu UpBFT. Podrobnejšie si funkcie, ako aj fungovanie algoritmu, rozpíšeme v prílohe Technická dokumentácia.

- void setup() - funkcia, ktorá sa spúšťa na začiatku programu a používa sa na nastavenie parametrov
- void loop() - funkcia, ktorá je periodicky spúšťaná, pokiaľ je mikrokontrolér pripojený ku zdroju elektrickej energie
- void sendPrepRequest() - pošle požiadavku na zariadenia, či sú pripravené na zápis do blockchainu
- void sendPrepResponse(IPAddress remoteIp) - odpovie klientovi, že zariadenie je pripravené na zápis
- void sendData(String data) - pošle samotné dáta na zápis do blockchainu

- void sendDataResponse(IPAddress remoteIp) - potvrdí klientovi prijatie a zapísanie dát
- int packetReceived(int packetSize) - najdôležitejšia funkcia, v ktorej sa spracúva väčšina prijatých správ
- void writeToLedger(char *data) - zapíše prijaté dáta do ledgera
- void setupWiFi() - inicializuje WiFi modul daného Arduino zariadenia a taktiež pripojí ku určenej WiFi sieti
- void printWiFiData() - vypíše nastavenia siete prináležiace ku WiFi sieťovému adaptéru daného Arduino zariadenia
- void printCurrentNet() - vypíše informácie o WiFi sieti, ku ktorej sa WiFi adaptér daného Arduino zariadenia pripojil
- void printMacAddress(byte mac[]) - vypíše MAC adresu WiFi adaptéra daného Arduino zariadenia

5.3.4 Funkcie použité pri PoS algoritme

Uvádzame zoznam a krátky popis vyvinutých funkcií, ktoré sú súčasťou implementácie algoritmu UPoS. Podrobnejšie si funkcie, ako aj fungovanie algoritmu, rozpíšeme v prílohe Technická dokumentácia.

- void setup() - funkcia, ktorá sa spúšťa na začiatku programu a používa sa na nastavenie parametrov
- void loop() - funkcia, ktorá je periodicky spúšťaná, pokiaľ je mikrokontrolér pripojený ku zdroju elektrickej energie

- void sendStakeRequest() - rozpošle do siete požiadavku na vykonanie stávok
- void sendStakeResponse(IPAddress remoteIp) - vygeneruje stávku a odošle ju Deciderovi
- void sendValidatorInfo(IPAddress val) - sformuluje paket a pošle informácie o validátorovi
- void sendDataForValidation(String data) - pošle validátorovi dáta na overenie
- void sendValidatedData(char *data) - validátor rozpošle zvalidované dáta na zápis do blockchainu
- void packetReceived(int packetSize) - najdôležitejšia funkcia, v ktorej sa vyhodnocuje väčšina prijatých správ
- void writeToLedger(char *data) - zapíše prijaté dáta do blockchainu
- char *ipToHexa(int addr) - prevedie ip adresu z dekadického tvaru do hexadecimálneho tvaru
- static char *ipToDec(const char *in) - prevedie ip adresu z hexadecimálneho tvaru do dekadického tvaru
- void swap(char *a, char *b) - pomocná funkcia funkcie ipToHexa
- void reverse(char* str) - pomocná funkcia funkcie ipToHexa
- void setupWiFi() - inicializuje WiFi modul daného Arduino zariadenia a taktiež pripojí ku určenej WiFi sieti
- void printWiFiData() - vypíše nastavenia siete prináležiace ku WiFi sieťovému adaptéru daného Arduino zariadenia

- void printCurrentNet() - vypíše informácie o WiFi sieti, ku ktorej sa WiFi adaptér daného Arduino zariadenia pripojil
- void printMacAddress(byte mac[]) - vypíše MAC adresu WiFi adaptéra daného Arduino zariadenia

5.4 Organizácia programu

5.4.1 Organizácia programu pri UpBFT algoritme

Náš program je rozdelený do niekoľkých logických častí:

- inicializácia zariadení a siete
- komunikácia s užívateľom
- odosielanie správ
- prijímanie správ
- zapisovanie do blockchainu.

Náš program sme nasadili na 4 Arduino zariadenia, z ktorých jedno slúžilo ako klient, ostatné ako bežné uzly siete. Úlohou klienta bolo posielať požiadavky na zapisovanie do blockchainu, bežné uzly siete mali za úlohu tieto požiadavky validovať a zapisovať do blockchainu. Na klienta aj na bežné uzly bol nahratý rovnaký program, pričom na klientovi bola aktívna časť kódu, ktorá zabezpečovala vysielanie požiadaviek na zápis do blockchainu.

Tento kód sa nachádza vo funkcii loop, ktorá je v pravidelných intervaloch opakovane vyvolávaná systémom. Po spustení tohoto kódu klient prostredníctvom funkcie sendPrepRequest pošle dopyt do siete, na ktorý očakáva počet odpovedí

aspoň dve tretiny počtu všetkých uzlov siete. Účelom tohto dopytu je overiť dostupnosť siete a potrebného počtu uzlov na validáciu požiadavky. Uzly siete pošlú klientovi odpoveď prostredníctvom funkcie sendPrepResponse. Po obdržaní dostatočného počtu odpovedí od uzlov siete klient pošle samotný dopyt na zapísanie do blockchainu prostredníctvom funkcie sendData. Uzly po prijatí dopytu tento spracujú a overia porovnaním kontrolného súčtu, ktorý sami vypočítajú s tým, ktorý dostanú v požiadavke. Ak je kontrolný súčet správny, dáta z požiadavky zapíšu do blockchainu pomocou funkcie writeToLedger a pošlú odpoveď klientovi. Ak súčet správny nie je, odpoveď nepošlú.

Všetky správy sú na všetkých uzloch prijímané vo funkcii loop, odkiaľ sú bezprostredne poslané do funkcie packetReceived na spracovanie.

5.4.2 Organizácia programu pri UPoS algoritme

Aj program s implementovaným algoritmom PoS sme nasadili na 4 Arduino zariadenia. Z nich jedno slúžilo ako Decider, ostatné ako klienti posielajúci transakcie na zápis do blockchainu i ako validátori, ktorí overia a sformujú transakciu. Na Decidera aj na ostatné uzly bol nahratý rovnaký program, pričom program na základe manuálne odoslaného paketu, ktorý obsahoval znak ï"(inicializácia), rozoznal, či sa jedná o Decidera alebo o bežný uzol siete. Podľa toho potom prispôsobil akcie, ktoré boli na danom uzle v činnosti.

Po aktivácii Arduino zariadení začína činnosť programu na uzle v roli Decidera, kde sa prostredníctvom funkcie sendStakeRequest rozošle na ostatné zariadenia požiadavka na doručenie stávok Deciderovi. Túto požiadavku uzly prijmú prostredníctvom funkcie loop, odkiaľ sa prijatá správa pošle na spracovanie do funkcie packetReceived. Vo funkcii loop sa prijímajú všetky správy a to na všetkých

uzloch a tiež sa všetky prijaté správy na všetkých uzloch spracúvajú vo funkcii packetReceived.

Po spracovaní požiadavky na doručenie stávok sa následne vo funkcii sendStakeResponse vygeneruje stávka a z tejto funkcie sa aj zašle Deciderovi. Ten požiadavku stávky od ostatných uzlov spracuje prostredníctvom funkcie packetReceived, kde jednotlivé odpovede ukladá do príslušných polí. Po uplynutí určeného času Decider vyhodnotí stávky, ktoré mu boli doručené. Toto sa udeje vo funkcii loop, odkiaľ je následne zavolaná funkcia sendValidatorInfo, prostredníctvom ktorej sa odošle informácia na všetky uzly siete o tom, kto bude validátorom.

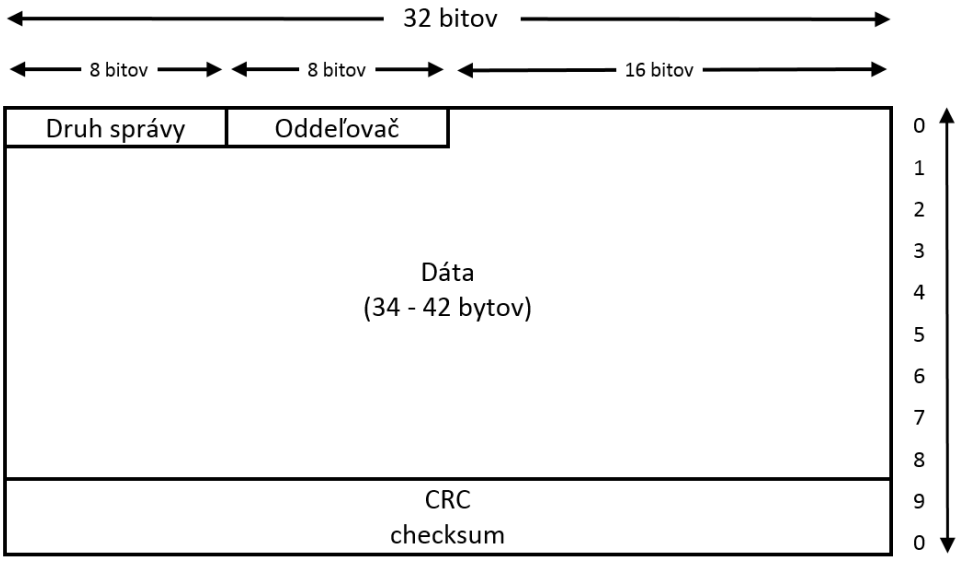
Uzly siete po prijatí informácie o validátorovi si jeho identifikáciu uložia prostredníctvom funkcie packetReceived, a začnú posielať požiadavky na validáciu transakcií a ich zápis do blockchainu. Toto robia prostredníctvom funkcie sendDataForValidation. Validátor, na ktorého tieto požiadavky uzly posielajú, ich opäť spracúva prostredníctvom funkcie packetReceived, odkiaľ sú prijaté dáta zvalidované a poslané do funkcie writeToLedger na zápis do blockchainu a zároveň pomocou funkcie sendValidatedData rozdistribuované do siete, aby si ich aj ostatné uzly mohli zapísať do blockchainu.

5.5 Zmeny oproti návrhu

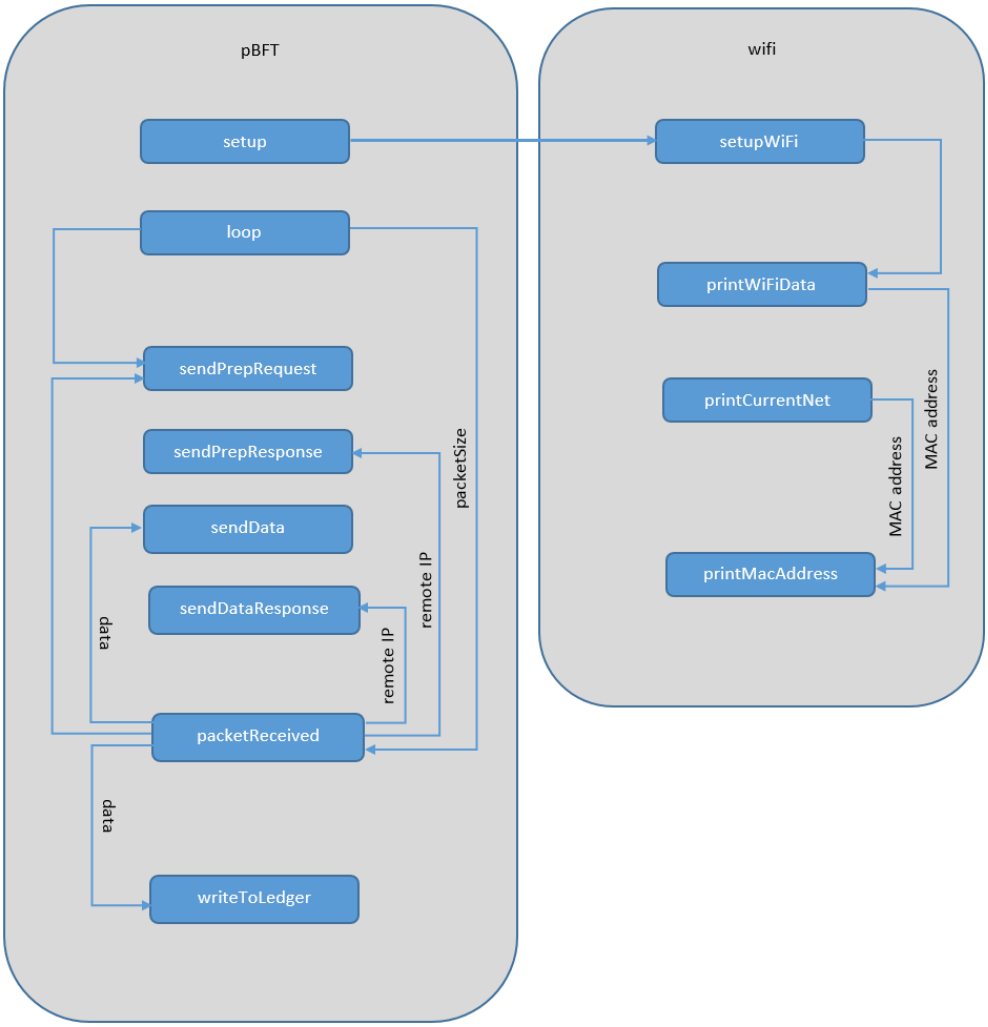
Medzi informáciami o transakciách ukladanými do blockchainu sa nachádza aj IP adresa odosielateľa správy. Túto IP adresu sme navrhovali ukladať do blockchainu v tvare reťazca o dĺžke 7-15 znakov (4 čísla v desiatkovej sústave oddelené bodkami). Pri implementácii som tento návrh zmenil na ukladanie IP adresy v hexadecimálnom tvare a bez bodiek. Tým sme získali jednak úsporu miesta v blockchaine, nakoľko hexadecimálny tvar IP adresy má dĺžku 8 bytov, na rozdiel

od desiatkového tvaru, ktorý môže mať dĺžku 7 až 15 bytov. V prípade privátnych IP adries, na ktorých sme prácu testovali, je táto dĺžka minimálne 11 bytov. Ďalší benefit, ktorý sme touto zmenou získali, je to, že dĺžka uloženej IP adresy bude vždy rovnaká, t.j. 8 bytov. Nový formát správy je zobrazený na obrázku č. 5.4.

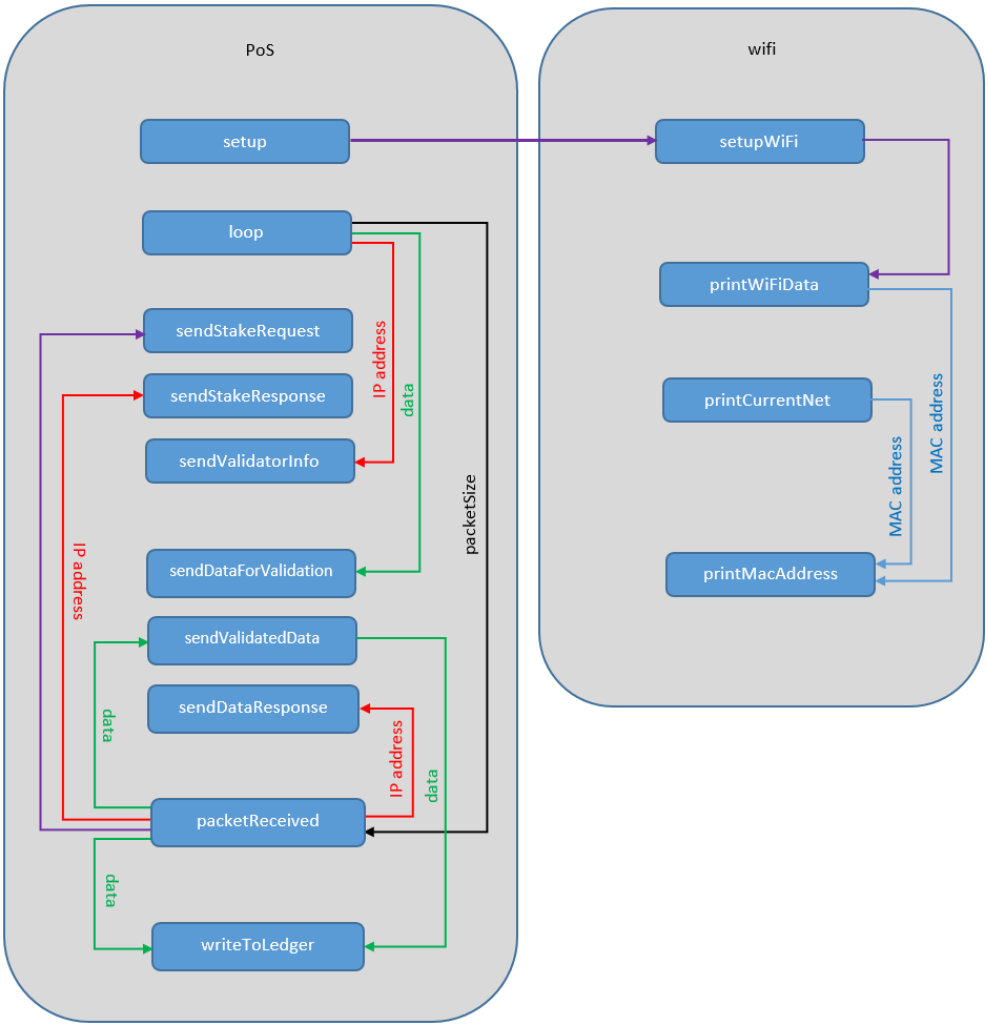
Ďalšou zmenou oproti návrhu je, že sme pri algoritme nenaimplementovali obe znovuzaslanie a znovuvyžiadanie správy. Je naimplementované znovuzaslanie požiadavky na prípravu na zápis do blockchainu, v prípade, že sa klientovi nedostaví do určitého časového limitu požadovaný počet odpovedí. Nie je naimplementované znovuvyžiadanie správy s dátami záložnými uzlami v prípade, že im príde správa porušená. Toto sme nenaimplementovali prevažne z dôvodu časovej tiesne. Túto funkcionalitu môžme považovať ako návrh na vylepšenie algoritmu do budúcosti.



Obr. 5.4: Nový formát správy nášho protokolu



Obr. 5.2: Moduly systému pri UpBFT algoritme a dáta, ktoré si medzi sebou vymieňajú



Obr. 5.3: Moduly systému pri UPoS algoritme a dáta, ktoré si medzi sebou vymieňajú

Kapitola 6

Overenie

Počas implementácie riešenia sme previedli viaceré testy na overenie funkčnosti nášho riešenia. Taktiež sme previedli merania spotreby energie a času trvania cyklu konsenzuálneho algoritmu. Prevedené testy a merania sú popísané v ďalších častiach tejto kapitoly.

6.1 Testovanie sieťového protokolu

Základným predpokladom pre komunikáciu uzlov siete je funkčný sieťový protokol, preto jedna z prvých vecí, ktorú sme v rámci implementácie vyvinuli a testovali bol sieťový protokol. Jeho testovanie sme previedli medzi dvoma vybranými Arduino zariadeniami. Z jedného sme posielali testovací rámec adresovaný priamo druhému testovaciemu Arduino zariadeniu. Zasielaný testovací rámec je uvedený na obrázku 6.1.

```
"3;D;c0a8c807;807d3a86899c;1553595857;2ba3b112"
```

Obr. 6.1: Testovací rámec posielaný po sieti

Keď sa nám podarilo úspešne prijať na druhom Arduino zariadení testovací rámec, mohli sme pristúpiť k ďalšej fáze testovania a síce ku zasielaniu rámcov prostredníctvom broadcastu na všetky uzly siete. Opäť sme prijatie rámca overovali na jednom vybratom Arduino zariadení.

6.2 Testovanie zápisu do blockchainu

Ďalšie testovanie, ktoré sme v rámci implementačnej časti práce prevádzali, bolo testovanie zápisu do blockchainu. Prevádzali sme ho takým spôsobom, že sme odstavili v rámci vyvinutého softvéru časť potvrdzovania transakcie a taktiež časť overovania transakcie a ponechali sme len samotný zápis do blockchainu s využitím už overeného sieťového protokolu. Na zápis do blockchainu sme si pripravili testovacie dáta jednej transakcie prevedenej na vybranom Arduino zariadení. Toto zariadenie poslalo broadcast rámcom požiadavku na zápis do blockchainu. Následne sme na ostatných zariadeniach overili zápis do blockchainu prezretím súboru reprezentujúceho blockchain. Skonštatovali sme, že zápis prebehol bez problémov. Následne sme z iného Arduino zariadenia poslali druhú testovaciu vzorku dát na zápis do blockchainu a overili sme pridanie ďalšieho záznamu v súbore a to opäť na viacerých uzloch testovacej siete. Podstatné pritom bolo, aby sa prvý testovací záznam v blockchainoch na jednotlivých zariadeniach nachádzal taktiež a to pred

druhým testovacím záznamom. Aj v tomto prípade bol test úspešný.

6.3 Testovanie konsenzuálnych algoritmov

Po otestovaní zápisu do blockchainu sme pristúpili ku testovaniu samotných algoritmov.

6.3.1 Testovanie UpBFT

Ako sme si spomenuli už vyššie pri návrhu, tento algoritmus spočíval v 4 krokoch. Skladal sa teda z výmeny týchto 4 paketov medzi klientom a zariadeniami:

- predpríprava - request
- predpríprava - response
- dáta - request
- dáta - response.

Pri každom z týchto paketov sme si dali vypísať pre testovacie účely do ledgeru a na seriálový monitor aj informáciu o odoslaní a prijatí týchto paketov. V tejto fáze sme komunikáciu testovali už aj s CRC32 checksumou, ktorá sa vypočítala z dát a pridala sa do päty paketu.

Rovnako ako pri predchádzajúcich testoch a overeniach, ani pri tomto sme nezistili žiadne väčšie problémy s funkcionalitou. Všetky výpisy sa nám zobrazili ako na seriálovom monitore, tak aj v ledgeri.

6.3.2 Testovanie UPoS

Testovanie tohto algoritmu prebiehalo obdobne ako pri predchádzajúcom algoritme, preto si ho podrobnejšie nebudeme rozpisovať.

Pri testovaní tohto algoritmu sme sa však veľmi dlho trápili s konverziou IP adresy. Pri overení transakcie validátorom nám IP adresa prišla zabalená v dátach správy ako hexadecimálny reťazec. Takisto sme si z UDP triedy WiFi knižnice získali IP adresu, z ktorej táto správa prišla. Tieto dve adresy sme chceli medzi sebou porovnať, či sa zhodujú. Problém bol, že programovací jazyk C/C++, v ktorom bol algoritmus písaný nepodporuje implicitné konverzie medzi väčšinou dátových typov a jedna IP adresa bola typu IPAddress v dekadickom tvare a druhá typu char* v hexadecimálnoom tvare. Nakoniec sa nám však podarilo IP adresy dostať do rovnakého tvaru a úspešne ich porovnať.

6.4 Meranie spotreby energie a času

6.4.1 Meranie spotreby energie a času algoritmu UpBFT

V tejto kapitole si opíšeme, ako sme testovali spotrebu energie a dĺžku trvania vykonania jedného cyklu UpBFT.

Najprv sme potrebovali vymyslieť spôsob, akým dĺžku trvania a spotrebu energie pri jednom cykle odmerať. Keďže sa v prípade času jedná o veľmi malé časové úseky, rozhodli sme sa, pre zvýšenie presnosti, že zmeriame čas pre vykonanie viacerých cyklov a následne ho vydelíme počtom vykonaných cyklov. Počet cyklov, vykonaných za sebou, sme sa, pre toto testovanie, rozhodli nastaviť na dvesto.

Takisto sme čakali, že všetko prebehne bez problémov, no opak bol pravdou. Hneď po spustení algoritmu sa začali vynárať prvé problémy. Samozrejme, kontrolné výpisy na seriálový monitor sme mali stále zapnuté. Po pár vymenených paketoch sa celá komunikácia zastavila. Vykonali sme zopár opakovaných pokusov, no situácia sa nezlepšila. Po konzultácii s vedúcim práce, dôkladnejšej analýze a vylúčení ostatných možností sme sa zhodli na tom, že v sieti dochádza ku strate paketov. Konkrétnejšie sme mali väčšinou problém pri strate paketov, ktoré potvrdzovali zapísanie dát do blockchainu. To znamená, že dáta sa reálne zapísali, ale potvrdenie sa už naspäť ku klientovi nedostalo. Z tohto dôvodu sme znížili hranicu akceptovateľnosti z "viac ako dve tretiny"na äspoň dve tretinyödpovedí od zariadení klientovi. V našej testovacej sieti to spravilo obrovskú zmenu, pretože odpoveď od "viac ako dvoch tretín"znamenala odpoveď od všetkých troch zariadení a ödpoveď aspoň od dvoch tretín"znamená odpoveď od dvoch alebo troch zariadení.

Po tejto drobnej úprave sa ešte sem tam stane, že dôjde k takej strate paketov, že sa test zastaví, no je to skôr výnimka ako pravidlo. Dali sme teda otestovať algoritmus podľa vyššie opísaného postupu.

Prvých pár spustení bežalo 200 cyklov algoritmu cca 26 sekúnd, čo vychádzalo niečo cez 130 milisekúnd na jeden cyklus. Po týchto pár spusteniach sme si všimli, že jeden kábel medzi Arduino zariadením a modulom na pripojenie SD karty je odpojený.

Keď sme znova prepojili Arduino zariadenie s SD modulom, spravili sme všetky merania nanovo. Tieto už vykazovali úplne iné hodnoty, ktoré môžeme vidieť v tabuľke č. 6.1 a 6.2.

Číslo merania	**Čas [ms]**	**Čas / cyklus [ms]**
1	10299	51,495
2	10218	51,09
3	10300	51,5
4	9851	49,255
5	10049	50,245
6	10094	50,47
7	9975	49,875
8	10304	51,52
9	11579	57,895
10	10651	53,255
Priemer	**10332**	**51,66**

Tabuľka 6.1: Namerané časy trvania 200 cyklov

Typ zariadenia	**Príkon v pokoji [W]**	**Príkon za behu [W]**
Klient	0,56	0,61
Záložný uzol	0,45	0,60

Tabuľka 6.2: Nameraná spotreba pri vykonaní 200 cyklov

6.4.2 Meranie spotreby energie a času algoritmu UPoS

Meranie tohto algoritmu bolo prakticky identické s meraním predchádzajúceho algoritmu, preto si ho nebudeme podrobnejšie popisovať.

Opäť sme merania previedli pri vykonaní 200 cyklov, kde sa na začiatku zinicializovala sieť, určil sa validátor a následne jeden uzol posielal 200 požiadaviek za sebou na validátora a ten do celej siete. Pri meraní 200 cyklov sme nebrali do

úvahy voľbu validátora, kde sa čaká 2 sekundy na odpoveď ostatných zariadení, pretože táto voľba sa vykoná len raz. Prípadne raz za časovú jednotku, ktorá je ale konštantná a nijako sa neodvíja od počtu poslaných requestov.

Toto meranie prebehlo bez väčších problémov. Namerané hodnoty môžeme vidieť v tabuľkách 6.3 a 6.4.

Číslo merania	**Čas [ms]**	**Čas / cyklus [ms]**
1	6110	30,55
2	5783	28,915
3	6628	33,14
4	6501	32,505
5	7424	37,12
6	5935	29,675
7	6008	30,04
8	6410	32,05
9	5891	29,455
10	7236	36,18
Priemer	**6392,6**	**31,963**

Tabuľka 6.3: Namerané časy trvania 200 cyklov

Typ zariadenia	**Príkon v pokoji [W]**	**Príkon za behu [W]**
Klient	0,45	0,60
Validátor	0,55	0,60
Ostatné uzly	0,45	0,60

Tabuľka 6.4: Nameraná spotreba pri vykonaní 200 cyklov

6.5 Zhodnotenie a porovnanie algoritmov

V projekte sme naimplementovali 2 algoritmy - upravený Practical Byzantine Fault Tolerance (UpBFT) a upravený Proof of Stake (UPoS). V tejto kapitole si ich porovnáme aj s tretím algoritmom - Proof of Work (PoW).

Z testovania nám vyplynulo, že časovo najlepší algoritmus je UPoS, za ktorým nie oveľa zaostáva UpBFT. S PoW sme v tomto ohľade ani nepočítali, pretože jedno overenie transkacie, resp. bloku, pri tomto algoritme trvá 10 minút. Môžme povedať, že čo sa času týka, UpBFT a UPoS sú skoro vyrovnané algoritmy.

Čo sa týka výpočtovej náročnosti, resp. spotreby energie, opäť sú algoritmy UPoS a UpBFT približne vyrovnané a naozaj veľmi málo náročné, vzhľadom na to, že sme sa ich tak snažili koncipovať. PoW sa s týmito algoritmami opäť nemôže porovnávať, keďže pri overení jednej transkacie, resp. bloku, sa hľadá vhodný hash a tým pádom je potrebná veľká výpočtová sila a spotrebuje sa veľa energie.

Čo do škálovateľnosti, víťazom je pre nás algoritmus UpBFT, napriek tomu, že v časti analýzy uvádzame pBFT ako neškálovateľný. Toto však znamená, že nie je použiteľný pre veľké siete. V našom prípade sa škálovateľnosťou myslí to, že si vieme upravovať hranicu akceptovateľnosti, t.j. že vieme nastaviť, koľko chybných uzlov v siete budeme akceptovať.

Práve výborná škálovateľnosť je dôvod, prečo sme sa rozhodli vo finálnom riešení použiť algoritmus UpBFT. Vieme pri ňom určiť, koľko uzlov musí odpovedať na požiadavku, aby sme ju považovali za potvrdenú. Do tejto hranice sa nepočítajú iba záškodnícke uzly, ale sú v nej zahrnuté aj chybné uzly prípadne straty paketov.

Kapitola 7

Záver

Pri písaní a implementovaní tejto práce sme sa veľa naučili. Či už to bolo z oblasti komunikačných algoritmov, IoT ale najmä z oblasti blockchainu a konsenzuálnych algoritmov. Týchto algoritmov sme zanalyzovali hneď niekoľko. Zistili sme, že blockchain je výbornou technológiou na zabezpečenie nielen bezdrôtových IoT sietí, ale prakticky akejkoľvek siete alebo systému, kde prebieha veľa komunikácie medzi rôznymi zariadeniami.

Naimplementovať sa nám podarilo dva z analyzovaných konsenzuálnych algoritmov, ktoré sme si upravili podľa našich potrieb. Zistili sme, že implementácia týchto algoritmov nie je až taká jednoduchá, ako sa môže na prvý pohľad zdať. No napriek tomu sa nám obidva algoritmy podarilo úspešne doimplementovať.

Naše riešenie sme overovali po častiach. Použili sme takzvanú iteratívno-inkrementálnu metódu vývoja, čo v skratke znamená, že sme naimplementovali nejakú menšiu časť celku, ktorú sme poriadne otestovali a vyladili a až potom sme prešli k implementácii ďalšej časti. Algoritmy sme porovnali medzi sebou na základe ich časovej náročnosti, pamäťovej náročnosti alebo ich spotreby energie.

Ako najlepší nám vyšiel UpBFT algoritmus, ktorý sme sa rozhodli použiť aj vo finálnom riešení.

Myslíme si, že je táto práca dobre rozpracovaná a má veľký potenciál rozvoja do budúcna. Táto časť bola pre nás prospešná najmä na dôkladné zoznámenie sa s touto problematikou a na osvojenie si princípov blockchainu a konsenzuálnych algoritmov. Do budúcna by mohlo byť naše riešenie rozšírené napríklad o šifrovanie správ alebo použitie spoľahlivejšej komunikácie. Možné návrhy vylepšenia uvádzame v nasledujúcej kapitole.

7.1 Návrhy na vylepšenie

Prvým návrhom na vylepšenie tohto riešenia je zrefaktorovať kód. Toto sme opäť nedotiahli do dokonalosti najmä z dôvodu časovej tiesne. V kóde vidíme tieto dva najhlavnejšie problémy:

- veľa podobných funkcií na odosielanie správ
- dlhá funkcia na spracovanie prijatých správ (packetReceived)

V kóde máme viacero funkcií na odosielanie správ (sendDataForValidation, sendValidatorInfo, ...), ktoré majú veľkú časť rovnakú. Túto časť navrhujem extrahovať do samostatnej funkcie, ktorú by sme mohli nazvať "sendä táto by dostala ako argument ČO má poslať a KOMU to má poslať.

Takisto aj pri probléme s dlhou a relatívne neprehľadnou funkciou packetReceived navrhujem extrahovať kusy kódu do samostatných funkcií, ktoré by boli volané z hlavnej funkcie packetReceived. V tejto funkcii je totižto niekoľko IF podmienok, z ktorých je vždy vykonávaná jedna, na základe toho, aký druh správy na

zariadenie prišiel. Kusy kódu z týchto podmienok by teda mohli byť extrahované do samostatných funkcií, ktoré by sa v týchto podmienkach volali.

Ďalším vylepšením do budúcnosti by mohla byť implementácia blockchainu na výkonnejších zariadeniach, ktoré budú mať viac systémových prostriedkov, vďaka čomu by bolo možné implementovať zvolený konsenzuálny algoritmus v plnom rozsahu a zároveň by sa nám otvorila možnosť použitia spoľahlivejšieho typu komunikácie (TCP). Ak by bol blockchain implementovaný na výkonnejších zariadeniach, mohli by sme uvažovať aj nad implementáciou iného typu konsenzuálneho algoritmu, napr. niektorého z vylepšených pBFT algoritmov.

Literatúra

[1] Douglas Bruey. *SNMP: Simple? Network Management Protocol*. 2005. URL: https://www.rane.com/note161.html.

[2] Miguel Castro a Barbara Liskov. *Practical Byzantine Fault Tolerance*. URL: http://pmg.csail.mit.edu/papers/osdi99.pdf.

[3] Many contributors. *Proof of Stake FAQs*. URL: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ#what-is-the-nothing-at-stake-problem-and-how-can-it-be-fixed.

[4] Brian Curran. *What is Practical Byzantine Fault Tolerance? Complete Beginner's Guide*. 2018. URL: https://blockonomi.com/practical-byzantine-fault-tolerance/.

[5] Freesoft. *SNMP protocol overview*. URL: https://www.freesoft.org/CIE/Topics/108.htm.

[6] Sean Dieter Tebje Kelly, Nagender Kumar Suryadevara a Subhas Chandra Mukhopadhyay. „Towards the Implementation of IoT for Environmental Condition Monitoring in Homes“. In: *IEEE Sensors Journal* (2013). ISSN: 1558-1748. DOI: 10.1109/JSEN.2013.2263379.

[7] Georgios Konstantopoulos. *Understanding Blockchain Fundamentals, Part 2: Proof of Work & Proof of Stake*. 2017. URL: https://medium.com/loom-

network/understanding-blockchain-fundamentals-part-2-proof-of-work-proof-of-stake-b6ae907c7edb.

[8] Georgios Konstantopoulos. *Understanding Blockchain Fundamentals, Part 3: Delegated Proof of Stake*. 2018. URL: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef.

[9] Victor Lai a Kieran O'Day. *What is Practical Byzantine Fault Tolerance? Complete Beginner's Guide*. 2018. URL: https://blockonomi.com/practical-byzantine-fault-tolerance/.

[10] Aaron Leskiw. *SNMP Basics: What is SNMP & How Do I Use It?* URL: https://www.networkmanagementsoftware.com/snmp-tutorial/.

[11] Mix. *Bitcoin mining consumes more electricity than 20+ European countries*. 2017. URL: https://thenextweb.com/hardfork/2017/11/23/bitcoin-mining-electricity-africa/.

[12] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf.

[13] Margaret Rouse et al. *Simple Network Management Protocol (SNMP)*. URL: https://searchnetworking.techtarget.com/definition/SNMP.

[14] Magdalena Schafferová. *Jak se vyznat v záplavě sítí pro internet věcí*. 2017. URL: https://www.zooco.io/blog/jak-se-vyznat-v-zaplave-siti-pro-internet-veci/.

[15] Dinesh Thangavel et al. „Performance evaluation of MQTT and CoAP via a common middleware". In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)* (2014). DOI: 10.1109/ISSNIP.2014.6827678.

[16] Dalibor Valko. *Základný prehľad algoritmov, na ktorých stoja kryptomeny*. 2018. URL: https://kryptomagazin.sk/zakladny-prehlad-algoritmov-na-ktorych-stoja-kryptomeny/.

[17] Zane Witherspoon. *A Hitchhiker's Guide to Consensus Algorithms*. 2018. URL: https://hackernoon.com/a-hitchhikers-guide-to-consensus-algorithms-d81aae3eb0e3.

[18] Tím Zaparkuj.to. *Zaparkuj.to - Príručka používateľa*. 2018. URL: https://zaparkujto.sk/public/assets/man/sp_user_manual_sk.pdf.