

Protokol pre komunikáciu medzi uzlami siete LoRa

LoRa-Based Protocol for Peer-to-Peer Long-Range Communication

Bc. Matúš Ozaniak

Diplomová práce

Vedoucí práce: Mgr. Ing. Michal Krumnikl, Ph.D.

Ostrava, 2022



Zadání diplomové práce

Student:

Bc. Matúš Ozaniak

Studijní program:

N2647 Informační a komunikační technologie

Studijní obor:

2612T025 Informatika a výpočetní technika

Téma:

Protokol pro komunikaci mezi uzly sítě LoRa

LoRa-Based Protocol for Peer-to-Peer Long-Range Communication

Jazyk vypracování:

slovenština

Zásady pro vypracování:

Navrhněte a implementujte komunikační protokol pro výměnu dat mezi stanicemi bez nutnosti existence centrálních uzlů. Protokol bude umožňovat zabezpečený přenos dat pomocí technologie LoRa. Součástí řešení bude realizace dvou bran s Ethernetovým nebo WiFi rozhraním demonstrující funkce navrženého řešení.

1. Proveďte rešerši v oblasti dostupných LoRa modulů a způsobu přenosu dat.
2. Srovnejte implementace protokolů peer-to-peer sítí realizovaných pomocí technologie LoRa.
3. Implementujte vlastní algoritmus na zvolené platformě (např. ESP32, Raspberry Pi).
4. Vytvořte vhodné rozhraní pro obsluhu a konfiguraci uzlů sítě (např. skrz webové rozhraní).
5. Navržené řešení otestujte a vyhodnotěte parametry sítě (propustnost, latence).

Seznam doporučené odborné literatury:

- [1] BERTO Riccardo, NAPOLETANO Paolo, SAVI Marco. A LoRa-based mesh network for peer-to-peer long-range communication. In: Sensors 21, no. 13 (2021): 4314.
- [2] SLABICKI Mariusz, PREMSANKAR Gopika, DI FRANCESCO Mario. Adaptive configuration of LoRa networks for dense IoT deployments. In: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, (2018): 1-9.
- [3] UMBER Noreen, BOUNCEUER Ahcène, CLAVIER Laurent. A study of LoRa low power and wide area network technology. In: International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). IEEE, (2017).
- [4] HANES, D. IOT fundamentals: networking technologies, protocols, and use cases for the internet of things. 3rd edition. Indianapolis, In: Cisco Press, (2017). ISBN 978-1-58714-456-1.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního oboru: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 07.11.2022 11:59:22

Abstrakt

Technologie LoRa je v současné době jednou z nejslibnějších pro internet věcí a získává si velkou pozornost. V současných aplikacích však existují omezení způsobená použitou specifikací LoRaWAN, která používá hvězdicovou topologii. Sítě založené na LoRaWAN mají obvykle omezenou škálovatelnost kvůli použití strategie centralizované správy. Nedávný pokrok v různých konstrukcích sítí LoRa mesh může představovat potenciální řešení těchto omezení. V této práci se zabýváme návrhem a implementací protokolu, který umožňuje komunikaci mezi uzly sítě LoRa s využitím topologie mesh. Protokol vyvinutý v této práci se zaměřuje na komunikaci v reálném čase prostřednictvím technologie LoRa a jako ukázka byla vyvinuta aplikace určená především pro chatovou komunikaci mezi uživateli v síti mesh. Součástí sítě však mohou být i senzorové uzly, které do sítě odesílají data ze senzorů.

Klúčové slová

LoRa; Mesh; Raspberry Pi; Komunikační protokol; IoT; Diplomová práce

Abstract

LoRa technology is currently one of the most promising for IoT and is gaining a lot of attention. However, there are limitations in current applications due to the LoRaWAN specification used, which uses a star topology. LoRaWAN-based networks typically have limited scalability due to the use of a centralized management strategy. Recent advances in various LoRa mesh network designs may present a potential solution to these limitations. In this work, we address the design and implementation of a protocol that enables communication between LoRa network nodes using a mesh topology. The protocol developed in this thesis focuses on real time communication through LoRa technology and as a demonstration an application has been developed primarily intended for chat communication between users in a mesh network. However, sensor nodes can also be part of the network, sending data from the sensors to the network.

Keywords

LoRa; Mesh; Raspberry Pi; Communication protocol; IoT; Master's thesis

Poděkovanie

Rád by som sa poděkoval svojmu vedúcemu diplomovej práce pánovi Mgr. Ing. Michalovi Krumniklovi, Ph.D. za odborné vedenie, konzultácie a podnety k diplomovej práci.

Obsah

Zoznam použitých symbolov a skratiek	8
Zoznam obrázkov	9
Zoznam tabuliek	11
1 Úvod	12
2 LoRa a spôsob prenosu dát	13
2.1 LPWAN	13
2.2 LoRa	14
2.3 LoRa paket	16
2.4 LoRa parametre	17
2.5 LoRaWAN	19
2.6 Legislatíva	19
3 Dostupné LoRa moduly	20
3.1 SX127x/SX126x	20
3.2 RFM9xW	21
3.3 Moduly a zariadenia použité v tejto práci	22
4 Existujúce riešenia	25
4.1 LoRa mesher	26
4.2 Meshtastic	26
4.3 LoRaBlink	27
4.4 Pymesh	27
4.5 Synchronous LoRa Mesh	28
4.6 Porovnanie a návrh nášho rozšírenia	28

5 Vlastná implementácia	31
5.1 Typy a stavy správ	32
5.2 Návrh paketu	34
5.3 Funkcionalita protokolu	36
5.4 Implementácia protokolu	36
5.5 Implementácia API pre webové rozhranie	43
5.6 Návrh webového rozhrania	44
5.7 Implementácia webového rozhrania	47
5.8 Optimalizácia	50
5.9 Implementácia pre Raspberry Pi 2B	52
5.10 Implementácia pre zariadenia TTGO	52
5.11 Testovanie funkčnosti	53
5.12 Problémy a limitácie	56
6 Záver	58
Literatúra	59
Prílohy	61

Zoznam použitých skratiek a symbolov

IoT	– Internet of Things - Internet vecí
SF	– Spreading factor - rozprestierací faktor
BW	– Bandwidth - šírka pásma
DR	– Data rate - rýchlosť prenosu
CR	– Coding rate - kódovací pomer
TDMA	– Time Division Multiple Access - Metóda zdielaného prístupu k prenosovemu médiu, ktorá rozdeľuje prístup na časové sloty
TTL	– Time To Live - Čas životnosti paketu
SNR	– Signal to Noise ratio - Pomer signálu k šumu
RSSI	– Received Signal Strength Indicator - Miera sily signálu
API	– Application Programming Interface - Rozhranie aplikácie
LPWAN	– Low Power Wide Area Network
FEC	– Forward Error Correction
LoRa	– Long Range
LoRaWAN	– Long Range Wide Area Network
SPI	– Serial Peripheral Interface
MQTT	– Message Queuing Telemetry Transport
AODV	– Ad hoc On-Demand Distance Vector

Zoznam obrázkov

2.1	Chirp signál a porovnanie rozprestieracích faktorov. Prevzaté z [6]	14
2.2	Chirp signály, do ktorých boli modulované symboly. [7, Prevzaté z video prezentácie výrobcu GW Instek]	15
2.3	Proces spracovania odosielaných dát v LoRa (Binárne hodnoty su ukážkové, nezodpovedajú reálnej konverzií). [7, Prevzaté z video prezentácie výrobcu GW Instek] . .	16
2.4	Čítanie symbolov preambuly prijatého paketu a synchronizácia čítacieho okna na základe posunu preambuly.	17
3.1	Mikrokontroléry TTGO. Prevzaté z [15]	23
3.2	Zariadenie Armachat. Prevzaté z [17]	23
3.3	Zariadenie Raspberry Pi 2B s LoRa modulom	24
4.1	Schéma Synchronous LoRa Mesh. Prevzaté z [25]	29
5.1	Časť stavového diagramu	33
5.2	Výpočet časového oneskorenia	38
5.3	Rozhranie zariadenia Armachat	41
5.4	Zariadenie Armachat prijalo spravu s jedným preskokom	42
5.5	Zariadenie Armachat prijalo traceroute správu	43
5.6	Štruktúra JSON objektu reprezentujúceho správu	44
5.7	Návrh úvodnej stránky	45
5.8	Návrhy stránok kontaktov a nastavení	46
5.9	Volanie cez fetch API	47
5.10	Informačná ikonka v pravom dolnom rohu pri správach	48
5.11	Stránka nastavení - časť 1	48
5.12	Stránka nastavení - časť 2	49
5.13	Stránky na mobilnom zariadení	50
5.14	Graf závislosti veľkosti pamäte na dĺžke správy	51
5.15	Rozmiestnenie zariadení použité pri testovaní	54

5.16	Meshtasticator simulátor	55
5.17	Priebeh šírenia správy s náhodnou trasou	55
1	Zjednodušený stavový diagram	62

Zoznam tabuliek

2.1	Frekvenčne pásma používané pre LoRa	17
3.1	Parametre Semtech SX modemov	21
4.1	Porovnanie LoRa mesh protokolov	30
5.1	Štruktúra hlavičky paketu a dĺžka jednotlivých polí v bajtoch.	34
5.2	Štruktúra dátového rámcu pre textové správy.	34
5.3	Štruktúra dátového rámcu pre správy typu traceroute.	35
5.4	Štruktúra dátového rámcu pre ACK a senzorové správy.	35
5.5	Ukážka bajtov odoslaného paketu	43

Kapitola 1

Úvod

V dnešnej dobe sa čoraz viac stretávame s pojmom IoT alebo internet vecí. Jedna sa o lokálne siete, zložené z fyzických zariadení, ktoré tvoria uzly siete. Zariadenia môžu byť jednoduché senzory na monitorovanie nejakej fyzikálnej veličiny, domáce spotrebiče, vozidla, prípadne zariadenia, ktoré je možné ovládať na diaľku. Zariadenia tvoria sieť, v ktorej si môžu medzi sebou posielat dátu.

K realizácii takejto siete je potrebné mať niečo, čo by zariadenia spájalo a umožňovalo im komunikáciu. Veľmi používanou technológiou v tejto oblasti je práve technológia LoRa, ktorá umožňuje bezdrôtovú komunikáciu na veľmi veľké vzdialenosťi.

Často je využívané riešenie LoRaWAN, ktoré sa skladá z centrálnych uzlov pripojených k internetu a zariadení, ktoré sú pripojené k centrálnym uzlom pomocou LoRa. Zariadenia potom komunikujú len s centrálnym uzlom a predávajú mu svoje dátu. Centrálny uzol dátu môže posielat cez internet na nejakú službu kde k ním môžu užívateľia pristupovať z internetu.

Pri LoRaWAN je potrebné mať nejaký centrálny uzol a ak chceme nejaké zariadenie pripojiť do siete, musí mať dosah na daný centrálny uzol. Takto sme limitovaní existenciou a dosahom centrálnych uzlov, a hviezdicovou topológiou, čož nie je v niektorých prípadoch užitia vhodné. Neustále vznikajú nové protokoly, ktoré by tieto problémy riešili, napríklad za použitia mesh topológie [1].

V tejto práci sa venujeme návrhu a vytvoreniu protokolu, ktorý umožňuje komunikáciu medzi zariadeniami v sieti tvorennej pomocou technológie LoRa, bez nutnej existencie centrálnych uzlov. Nami vytvorený protokol vytvára sietovú topológiu typu mesh, ktorá ma oproti hviezdicovej topológií, využívanej pri LoRaWAN, niekoľko výhod. Su nimi napríklad škálovatelnosť siete, kedy sa môžu zo siete odoberať alebo do nej pridávať nové zariadenia, bez nutnosti akejkoľvek konfigurácie na ostatných zariadeniach. Z toho vyplýva aj vyššia mobilita zariadení. Zariadenia sa môžu fyzicky pohybovať a pokiaľ sa nachádzajú v dosahu hocijakého iného uzla, majú prístup do siete.

Kapitola 2

LoRa a spôsob prenosu dát

V dnešnej dobe sa na trhu nachádza veľa rôznych technológií, ktoré umožňujú bezdrôtovú komunikáciu. Niektoré technológie, ako napríklad Bluetooth, sú vhodné pre komunikáciu na malé vzdialenosť, napríklad komunikácia medzi mobilným telefónom a smart hodinkami. Zatiaľ čo iné technológie ako napríklad WiFi nám ponúkajú možnosť komunikácie na väčšie vzdialenosť. Nie všetky technológie sú však vhodné pre použitie v IoT sieťach. V IoT sieťach sa často dbá na predpoklad nízkej spotreby energie. Okrem nízkej spotreby energie je taktiež vhodné aby mal bezdrôtový prenos veľký dosah. Z týchto dôvodov sa pri IoT sieťach využívajú takzvané LPWAN siete.

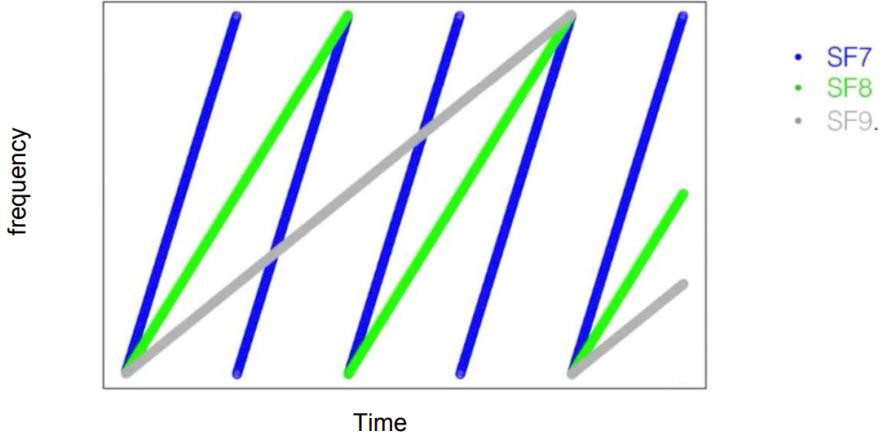
2.1 LPWAN

LPWAN (Low Power Wide Area Network) je kategória sieti s veľkou rozlohou a nízkou spotrebou energie. Tieto siete sa vyznačujú nízkymi obstarávacími nákladmi a dlhodobou prevádzkou. Siete sú tvorené jednoduchými a často pomerné lacnými zariadeniami, ktoré vďaka nízkej spotrebe energie dokážu pracovať dlhú dobu bez nutnosti stáleho pripojenia do elektrickej siete. Zariadenia bývajú napájane batériami a môžu nepretržite fungovať aj niekoľko rokov. V kombinácii so solárnymi panelmi môže byť ich prevádzka ešte predĺžená.

Tieto siete sú teda vhodné pre aplikácie, kde je potrebná dlhodobá prevádzka bez nutných servisných zásahov.

LPWAN siete majú veľké pokrytie, v niektorých prípadoch to môžu byť až desiatky kilometrov v otvorenom priestranstve. Zväčša využívajú na prenos ultrakrátke (sub-GHz) frekvenčné pásma, ktoré nevyžadujú na používanie žiadnu rádiovú licenciu.

Medzi tie najznámejšie technológie na realizáciu LPWAN sieti patria napríklad Sigfox [2], LoRa [3], NB-IoT (NarrowBand Internet of Things) a iné.



Obr. 2.1: Chirp signál a porovnanie rozprestieracích faktorov. Prevzaté z [6]

2.2 LoRa

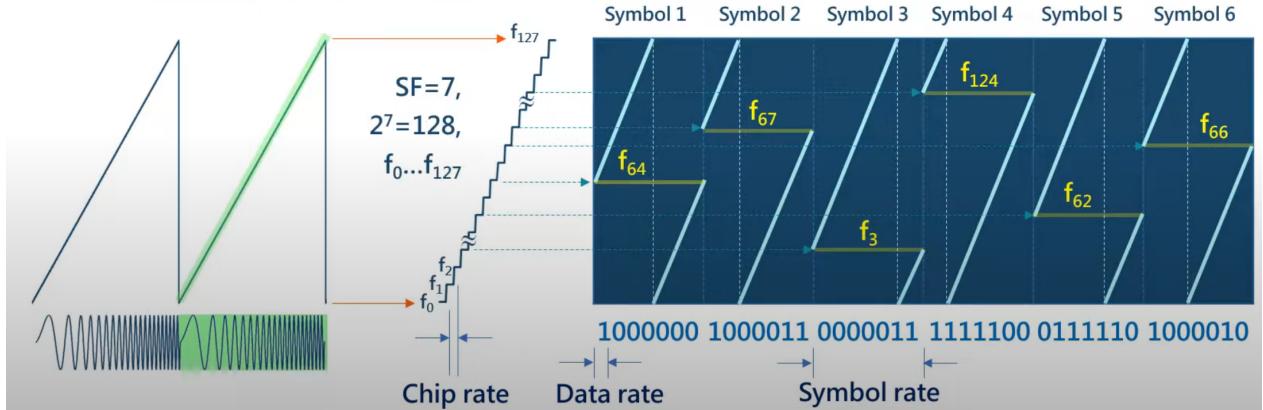
LoRa je proprietárna technológia na bezdrôtový prenos dát za pomocí rádiových vln. Používa bezlicenčné rádiové pásma, ktoré sú odlišné medzi Európou, Amerikou a Áziou [4], a poskytuje rádiový prenos na veľkú vzdialenosť s nízkou spotrebou energie. V otvorenom priestranstve môže mať rádiový prenos dosah až 10–15 km [5]. LoRa má však veľkú limitáciu v podobe nízkej rýchlosťi prenosu dát. Rýchlosť prenosu sa pohybujú medzi 0,3 až 37,5 kbps v závislosti na vybraných parametroch.

Vďaka týmto aspektom je vhodná pre použitie v IoT senzorových sietach, kde sa často vyskytujú senzory poháňané batériami a je potrebné aby vydržali dlhú dobu bez výmeny batérií. Okrem toho senzory väčšinou odosielajú veľmi malý obsah dát a dáta posielajú iba v určitých časových intervaloch (napr. raz za hodinu), takže nízka prenosová rýchlosť v tomto prípade nie je až tak veľkou limitáciou.

Na prenos dát v LoRa, je použitá proprietárna frequency hopping chirp spread-spectrum modulácia – modulácia rozprestreného spektra s preskokom frekvencií, pri ktorej sú prenášané dátá kódovane do symbolov a každý vysielaný symbol je prenášaný takzvaným chirp signálom, do ktorého sa daný symbol moduluje.

Chirp signál ma konštantnú amplitúdu ale mení svoju frekvenciu lineárne s časom. Frekvencia sa mení v rozmedzí od spodnej hranice frekvenčného pásma, po hornú hranicu frekvenčného pásma. Po dosiahnutí hraničnej frekvencie sa frekvencia vráti na opačnú hranicu a proces sa opakuje. Frekvenčné pásma, v ktorom sa chirp signál prenáša je určené vybranou šírkou pásma. Graf frekvenčnej charakteristiky a postupné zvyšovanie frekvencie chirp signálu môžeme vidieť na Obr. 2.2.

Existujú dva druhy chirp signálov, sú nimi up-chirp signál a down-chirp signál. Pri up-chirp signále sa prechádza zo spodnej hranice frekvenčného pásma do hornej hranice a pri down-chirp zase naopak.



Obr. 2.2: Chirp signály, do ktorých boli modulované symboly. [7, Prevzaté z video prezentácie výrobcu GW Instek]

Ako rýchlo sa chirp posúva po frekvenčnom pásme - tzn. ako rýchlo chirp signál mení svoju frekvenciu, je určené parametrom rozprestierací faktor – spreading factor (SF). Rozprestierací faktor zároveň vyjadruje, kolko bitov informácie je v každom symbole prenesených. Pri nižšom rozprestieracom faktore sa chirp signál posúva po frekvenčnom pásme rýchlejšie (viď Obr. 2.1) a tým sa zvyšuje rýchlosť dátového prenosu, avšak zhoršuje sa citlivosť, ktorou dokáže prijímač prijať signál a dôsledkom toho je menší použiteľný dosah. Citlivosť sa zhoršuje kvôli tomu, že je vysielaný signál menej rozšírený v čase a signál môže byť v porovnaní so šumom, ktorý je prítomný v prostredí, relatívne slabší. Hodnota šumu voči signálu je vyššia a môže to viesť k zhoršeniu citlivosti prijímača.

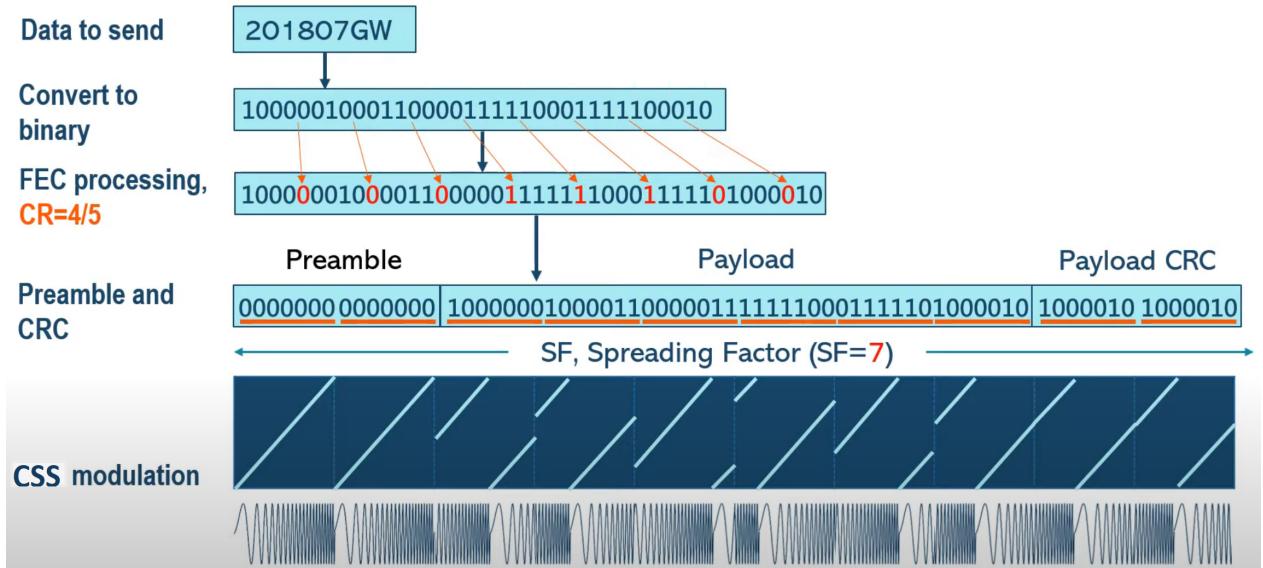
Každý chirp signál je rozdelený na X častí – tzv. chips. Tieto chips predstavujú skoky vo vysielacej frekvencii signálu. Kolko týchto chips jeden chirp obsahuje je závisle od vybraného rozprestieracieho faktoru. Jeden chirp je rozdelený na 2^{SF} častí alebo chips.

Vysielaný symbol sa potom skladá z cyklicky posunutého chirp signálu, kde posun definuje hodnotu daného symbolu. To znamená, že vysielaný chirp nebude začínať na spodnej hranici frekvenčného pásma, ale na určitej frekvencii korespondujúcej so symbolom, ktorý je modulovaný do daného chirp signálu. Viď Obr. 2.2.

Celý proces vyslania správy cez LoRa sa teda skladá z nasledujúcich krokov:

1. Konverzia správy do binárneho kódu
2. Pridanie korekčných bitov slúžiacich na opravu chýb
3. Pridanie preambuly a kontrolného súčtu, a poskladanie do LoRa paketu
4. Modulácia bitov do chirp signálov
5. Odvysielanie chirp signálov

Ukážku môžeme vidieť na Obr. 2.3.



Obr. 2.3: Proces spracovania odosielaných dát v LoRa (Binárne hodnoty sú ukážkové, nezodpovedajú reálnej konverzií). [7, Prevzaté z video prezentácie výrobcu GW Instek]

2.3 LoRa paket

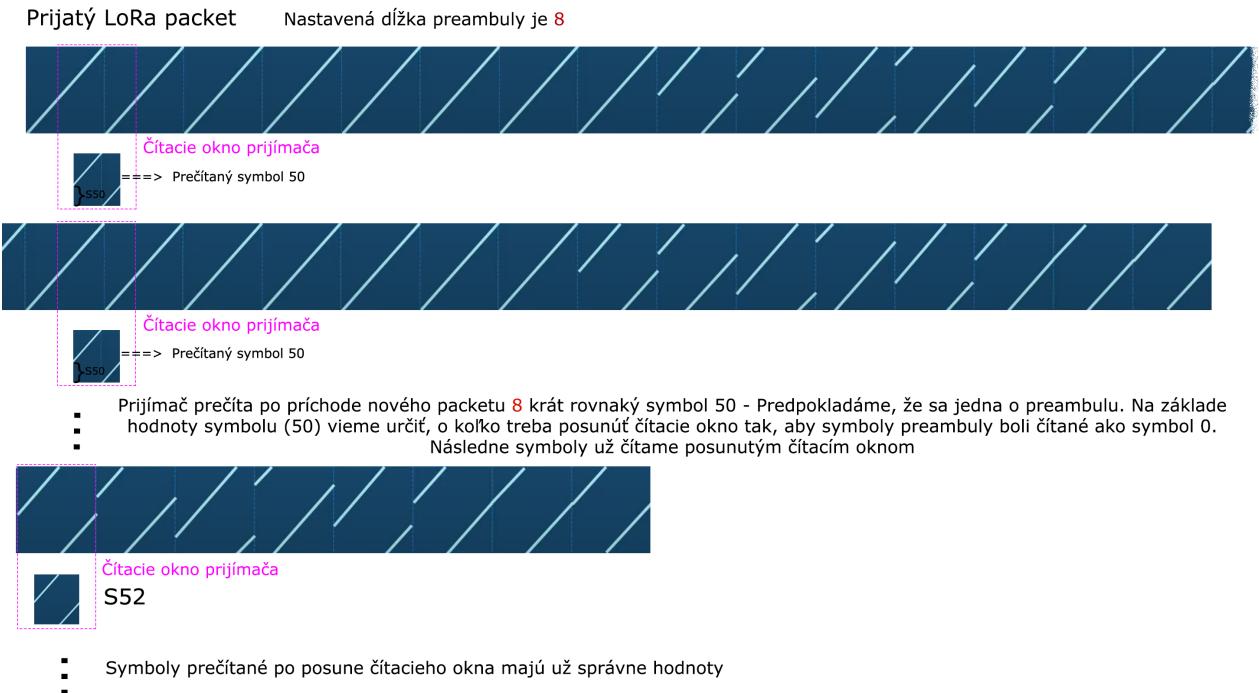
Štruktúra LoRa paketu môže byť závislá od daného použitia. Bežné sa ale stretávame s LoRa paketom, zloženým z preambuly, dát a kontrolného súčtu. Pričom maximálna povolená dĺžka dát je 255 bajtov.

Preambula slúži na synchronizáciu prijímacieho zariadenia. Je tvorená niekoľkými opakovaniami prázdnego chirp signálu, ktorý predstavuje symbol s hodnotou 0. Dĺžka preambuly, tzn. počet opakovania prázdnego chirp signálu, je stanovená konfiguráciou zariadenia. Bežne sa stretávame s dĺžkou preambuly 8. Na Obr. 2.3 môžeme vidieť, že na preambulu boli použité iba 2 prázdné chirp signály – teda dĺžka preambuly je 2.

Prijímacie zariadenie číta prijaté symboly v určitom časovom intervale – okne. Tento interval sa ale nemusí zhodovať s intervalom, ktorý bol použitý pri vysielaní daných symbolov. Je preto potreba synchronizovať prijímacie zariadenie aby hodnoty symbolov boli správne interpretované.

Zariadenie po prijatí nového paketu očakáva, že na začiatku paketu bude preambula definovanej dĺžky. Ak z paketu prečíta počet symbolov rovnakej hodnoty, zhodný s definovanou dĺžkou preambuly, predpokladá že sa jedná o preambulu a synchronizuje čítacie okno tak aby bolo zarovnané so symbolmi preambuly. Tzn. tak, aby symboly preambuly boli prečítané ako symbol 0.

Zjednodušené znázornenie procesu čítania symbolov a posun čítacieho okna môžme vidieť na Obr. 2.4.



Obr. 2.4: Čítanie symbolov preambuly prijatého paketu a synchronizácia čítacieho okna na základe posunu preambuly.

2.4 LoRa parametre

Pri používaní technológie LoRa je nutné správne zvoliť parametre na bezdrôtový prenos. Medzi nastaviteľné parametre patrí frekvencia, šírka pásma, rozprestierací factor a kódovací pomer. Použitá frekvencia je závislá od regiónu, v ktorom chceme technológiu LoRa využívať, viď Tabuľka 2.1.

V Európe je možné mimo 866 MHz pásma používať na LoRa aj 433 MHz pásmo. Okrem toho existuje ešte globálne používaná frekvencia 2,4 GHz. Tu sme ale limitovaní podporou daných frekvencií v nami použitom LoRa modeme. Bežné používané moduly, zväčša nemajú podporu pre 2,4 GHz frekvenčné pásma.

Tabuľka 2.1: Frekvenčne pásma používane pre LoRa

Región	Frekvenčné pásma (MHz)
Ázia	433
Európa, Rusko, India, Afrika	863–870
Severná Amerika	902–928
Austrália	915–928
Kanada	779–787
Čína	779–787, 470–510

Ostatné LoRa parametre sú vyberané na základe toho ako daleko, ako spoľahlivo a ako rýchlo je potrebné dátá prenášať. Je nutné zvoliť vhodný kompromis medzi rýchlosťou prenosu a dosahom prenosu. Pri výbere parametrov je ale taktiež potrebné dbať na to, aby neboli prekročený povolený klučovací pomer. O tom si povieme viac v sekcií legislatívnej.

Parameter bandwidth (BW) určuje šírku pásma, v ktorom sa bude chirp posúvať. Pri vyššej šírke pásma sa zvyšuje rýchlosť prenosu, avšak znížuje sa použiteľný dosah. Najbežnejšie používané šírky pásma sú 125 kHz, 250 kHz a 500 kHz.

Rozprestierací faktor – spreading factor (SF) určuje kolko bitov dát bude prenesených v každom vysielanom symbolu. To určuje ako rýchlo sa chirp posúva po frekvenčnom pásme a tym pádom zvyšuje alebo znížuje rýchlosť prenosu na úkor zníženia alebo zvýšenia dosahu prenosu. Spreading factor je vo väčšine prípadov možné zvoliť z intervalu 6–12. Avšak niektoré LoRa modemy umožňujú nastaviť aj nižšie hodnoty rozprestieracieho faktoru.

Používanie rozprestieracích faktorov prináša výhodu v podobe ortogonality správ vysielaných s rôznymi rozprestieracími faktormi. To znamená, že prijímač dokáže správne prijať a dekódovať správu poslanú s rozprestieracím faktorom X aj keď sa vysielaná správa časovo prekrýva s inou vysielanou správou s iným rozprestieracím faktorom Y.

LoRa obsahuje korekciu chýb spôsobených rušením. Využíva k tomu samo-opravný kód – forward error correction (FEC), pri ktorom sa ku prenášaným dátam pridávajú korekčné byty. Tieto byty sú potom na prijímacej strane použité na detekciu a prípadnú opravu chyby ak k nejakej došlo vplyvom rušenia. K nastaveniu korekcie chýb slúži parameter kódovací pomer – coding rate (CR).

V technológií LoRa máme na výber zo štyroch možností pre parameter coding rate: 4/5, 4/6, 4/7 a 4/8. Tieto možnosti sú niekedy označované aj ako čísla x = 1 až 4, kde kódovací pomer je 4/4+x. Označenie vyjadruje pomer bitov, ktoré nesú informáciu, ku bitom, ktoré budú reálne použité na prenos informácie. Napríklad pri kódovacom pomere 4/5 sa na každé 4 byty informácie pridáva 1 korekčný bit. Vyšší kódovací pomer zabezpečí spoľahlivejší prenos dát ak sa nachádzame v rušivom prostredí, avšak zníží rýchlosť prenosu dát, pretože ku prenášaným dátam pridáva navyše dátá potrebné na korekciu chýb.

Rýchlosť prenosu dát (Data rate – DR) môžeme vyjadriť vzťahom:

$$DR = SF * \frac{\frac{4}{4+CR}}{\frac{2^SF}{BW}} * 1000$$

DR	rýchlosť prenosu dát v bitoch za sekundu
BW	šírka pásma v kHz
SF	rozprestierací faktor (hodnoty 6–12)
CR	kódovací pomer (hodnoty 1–4)

2.5 LoRaWAN

Technológia LoRa je definovaná len na fyzickej vrstve. Na používanie LoRa v IoT sieťach sú však potrebné aj vyššie vrstvy sieťového modelu. K tomu vznikol protokol LoRaWAN [8], ktorý je spravovaný organizáciou LoRa Alliance [3].

LoRaWAN definuje komunikačný protokol a architektúru IoT sieti. Siete používajú hviezdicovú topológiu, prípadne topológiu hviezdu hviezd, kde centrálnym uzlom je LoRaWAN brána – gateway, ktorá je pripojená k internetu a pevnému napájaniu. Ostatné uzly siete posielajú dátu do tejto brány, ktorá ich potom preposiela na internet. Tam sú už dátu dostupné odkiaľkoľvek.

LoRaWAN definuje tri základne triedy zariadení, ktoré sa v sieti vyskytujú. Triedy špecifikujú funkciu zariadenia a jeho vlastnosti. Sú nimi triedy A, B a C, kde do triedy A spadajú zariadenia väčšinou poháňané batériami, ktoré po odvysielaní svojich dát otvárajú dve prijímacie okna, v ktorých čakajú na príjem dát z brány. Trieda B je rozšírením triedy A. Zariadenia v tejto triede môžu otvárať dodatočné prijímacie okna v naplánovaných časových intervaloch. Zariadenia z triedy C môžu prijímať neustále. Z tohto dôvodu majú vyššiu spotrebú energie a zvyčajne sú pripojené k stálemu napájaniu.

Zariadenia používané na realizáciu gateway bývajú zväčša výkonnejšie oproti koncovým zariadeniam. Okrem toho sú schopné prijímať na viacerých frekvenciach a LoRa parametroch zároveň. Gateway môže byť pripojená nejakú clouдовú službu, cez ktorú je možné spravovať zariadenia v sieti. Medzi najznámejšie služby patrí The Things Network (TTN) [9]. Je však možné spojiť LoRaWAN aj s inými populárnymi clouдовými službami ako je napríklad Microsoft Azure [10] alebo Google Cloud [11].

2.6 Legislatíva

V Európe sa k frekvenčnému pásmu používaného pri technológií LoRa viažu určité obmedzenia. Obmedzenia sa týkajú používania fyzického média. Regulácia určuje maximálnu povolenú dobu, v ktorú môže vysielač na daných frekvenciách vysielať v rámci jednej hodiny a maximálny vysielačí výkon, akým môže signal vysielať.

Určuje sa takzvaný klučovací pomer, ktorý hovorí kolko percent času z nejakého celkového času môže vysielač vysielať. Ak by zariadenie vysielalo signál po dobu jednej jednotky času z celkových 10 jednotiek času, tak klučovací pomer by bol 10 %.

Český Telekomunikačný úrad stanovuje vo všeobecné oprávnení č. VO-R/10/07.2021-8 [12], že frekvenčné pásmo, do ktorého spadá technológia LoRa, patrí do skupiny g. Pre túto skupinu je maximálny povolený klučovací pomer 1 % a maximálny vysielačí výkon 25 mW (14 dBm). Znamená to teda, že zariadenia môžu každú hodinu vysielať maximálne po dobu 36 sekúnd. Pre pásmo 869,40–869,65 MHz je ale udelená výnimka, ktorá umožňuje vysielať s klučovacím pomerom 10 % a maximálnym vysielačím výkonom 500 mW (27 dBm).

Kapitola 3

Dostupné LoRa moduly

Pri práci s technológiou LoRa máme na výber z rôznych modulov od rôznych výrobcov. Moduly môžeme rozdeliť podľa použitia na moduly pre koncové zariadenia a moduly pre gateway. Moduly pre koncové zariadenia obvykle dokážu prijímať a odosielat iba na jednom kanále (kombinácia LoRa parametrov – BW, SF, CR a frekvencia) súčasne a majú nízku spotrebu energie. Moduly pre gateway dokážu prijímať a odosielat na viacerých kanáloch súčasne.

V tejto časti sa budeme zaoberať dostupnými modulmi pre koncové zariadenia. Kedže je technológia LoRa patentovaná výrobcom Semtech [13], tak všetky dostupné LoRa čipy sú práve od tohto výrobcu a moduly od iných výrobcov sú založené na používaní týchto čipov. Kompletné porovnanie parametrov najpoužívanejších modulov je možné vidieť v tabuľke 3.1.

3.1 SX127x/SX126x

Výrobca Semtech [13], prináša LoRa čipy – modemy série SX127x a SX126x. Ponúkajú vysoký výkon za pomerne nízku cenu a ako sme už spomínali, ostatné LoRa moduly iba implementujú tieto LoRa modemy a rozširujú ich o ďalšiu funkcionalitu.

3.1.1 SX127x

SX127x LoRa modemy používajú LoRa modulačnú techniku frequency hopping spread-spectrum, patentovanú firmou Semtech.

Maximálny vysielační výkon týchto modemov je 100 mW. Vďaka použitej modulačnej technike je možné dosiahnuť vysokú citlivosť modemov. Výrobca uvádza citlosť cez -137 dBm pri modemoch SX1272/73 a -148 dBm pri modemoch SX1276/78/79.

Modemy SX1272 a SX1273 ponúkajú menší link budget¹ 157 dB oproti 168 dB pri modemoch SX1276/77/78/79.

¹Súčet zisku a strát signálu medzi vysielačom a prijímačom v bezdrôtovom prenose.

Majú menší rozsah frekvenčných pásiem medzi 860 a 1020 MHz oproti modemom SX1276/77/78/79, kde je možné vybrať frekvenčné pásma z rozsahu 137 až 1020 MHz.

Okrem toho majú aj vyššiu spotrebu energie.

3.1.2 SX126x

Modemy zo série SX126x - SX1261/62/68 sú nasledovníkmi predošlých modemov SX127x. Majú väčší vysielačí výkon vďaka integrovanému zosilňovaču a menšiu spotrebu energie. Obsahujú precízny TCXO oscilátor, ktorý zabezpečuje presnejšie a stabilnejšie riadenie počas prevádzky modemu. Okrem LoRa modulácie obsahujú aj G(FSK) moduláciu, ktorá je vhodná pre staršie prípady použitia.

Modemy obsahujú +22/+15 dBm zosilňovač, vďaka ktorému majú vyšší link budget oproti modemom zo série SX127x. Ten je pri modemoch série SX126x výrobcom udávaný na 170 dBm, takže sú optimálne pre aplikácie vyžadujúce väčší dosah alebo robustnosť.

Tabuľka 3.1: Parametre Semtech SX modemov

Modem	Frekvencia	SF	BW (kHz)	Citlivosť	Spotreba ¹	Rozhranie	Výkon ²	Cena ³
SX1272	860–1020 MHz	6–12	125–500	-137 dBm	11,2 mA	SPI	20 dbm	9€
SX1273	860–1020 MHz	6–9	125–500	-130 dBm	11,2 mA	SPI	20 dbm	7€
SX1276	137–1020 MHz	6–12	7,8–500	-148 dBm	12 mA	SPI	20 dbm	10€
SX1277	137–1020 MHz	6–9	7,8–500	-139 dBm	12 mA	SPI	20 dbm	7€
SX1278	137–525 MHz	6–12	7,8–500	-148 dBm	12 mA	SPI	20 dbm	8€
SX1279	137–960 MHz	6–12	7,8–500	-148 dBm	12 mA	SPI, UART	20 dbm	11€
SX1261	150–690 MHz	5–12	7,80–500	-125 dBm	8 mA	SPI	15 dbm	7€
SX1262	150–690 MHz	5–12	7,80–500	-125 dBm	8 mA	SPI	22 dbm	8€
SX1268	410–810 MHz	5–12	7,80–500	-148 dBm	4,6 mA	SPI	22 dbm	7€

3.2 RFM9xW

Moduly RFM95W/96W/98W od výrobcu HopeRF [14] implementujú SX LoRa modemy od výrobcu Semtech. Jedná sa o jednoduchý modul, ktorý obsahuje všetku riadiacu elektroniku potrebnú pre riadenie Semtech LoRa modemu. Okrem riadiacej elektroniky obsahuje modul aj zosilňovač signálu (+14 dBm), ktorý zvyšuje dosah bezdrôtového prenosu.

Existuje niekoľko verzií modulov RFM9xW, kde každá verzia používa iný semtech LoRa modem a zdiela parametre daného modemu.

¹Spotreba počas prijímania (mA)

²Maximálny vysielačí výkon

³Cena platná ku Q4 2022

3.3 Moduly a zariadenia použité v tejto práci

Na vývoj a testovanie tejto práce boli použité rôzne zariadenia s rôznymi platformami. Na simu-lovanie jednoduchých koncových zariadení, ktoré môžu predstavovať napríklad nejaký senzor, boli použité mikrokontroléry TTGO od výrobcu LILYGO [15].

Okrem mikrokontrolérov TTGO boli použité aj mikrokontroléry Raspberry Pi Pico a jednodos-kový počítač Raspberry Pi. Nami použité mikrokontroléry môžu byť neskôr rozšírene o batériu a stať sa tak mobilnými.

3.3.1 TTGO LoRa32

TTGO LoRa32 je multifunkčný mikrokontrolér postavený na module ESP32, ktorý poskytuje mo-duly pre bezdrôtovú komunikáciu, vrátane Wi-Fi, Bluetooth a SX1276 LoRa modulu.

Tento mikrokontrolér je navrhnutý pre projektovanie internetu vecí (IoT), senzorov a ďalších podobných aplikácií, ktoré vyžadujú spoľahlivú a energeticky úspornú bezdrôtovú komunikáciu.

Jednou z výhod tohto mikrokontroléra je jeho úsporný režim spánku, ktorý umožňuje znížiť spotrebu až na 0,6 mA. Tento režim je ideálny pre aplikácie, ktoré potrebujú byť v prevádzke čo najdlhšie, bez nutnosti pravidelného nabíjania batérie.

Okrem toho TTGO LoRa32 obsahuje aj 0,96 palcový čiernobiely displej pripojený cez I2C rozhranie. Tento displej môže slúžiť ako zobrazovací panel pre aplikácie, ktoré potrebujú zobrazovať textové a grafické informácie.

3.3.2 TTGO T-Beam

TTGO T-Beam je mikrokontrolér taktiež založený na module ESP32 s integrovaným Wi-Fi, Bluetooh a LoRa modulom. V porovnaní s predošlým mikrokontrolérom je však tento vybavený aj GPS modulom, ktorý umožňuje určovanie polohy zariadenia.

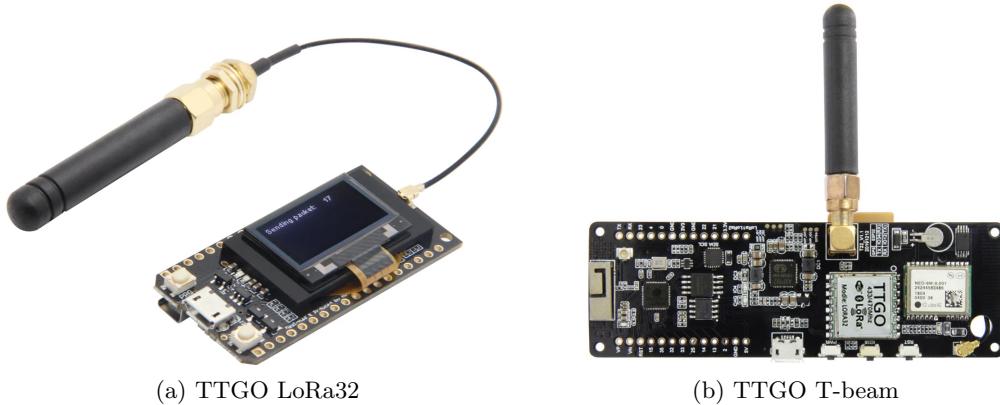
Mikrokontrolér má na zadnej strane držiak na batériu, z ktorej môže byť napájaný. Taktiež je vybavený nabíjacím modulom, ktorý umožňuje bezpečné nabíjanie a ochranu lítiových batérií. TTGO T-Beam je v režime spánku schopný spotreby iba 0,2 mA.

Okrem toho, TTGO T-Beam je tiež schopný pripojenia dodatočného displeja, ktorý môže byť použitý na zobrazovanie informácií. Nami používaný mikrokontrolér však tento displej nemal.

Na Obr 3.1 môžeme vidieť oba mikrokontroléry TTGO.

3.3.3 Raspberry Pi Pico RP2040

Mikrokontrolér od známej organizácie Raspberry Pi [16], založený na dvoj-jadrovom ARM procesore. Existuje verzia s Wi-Fi modulom aj bez. Na programovanie sa využíva jazyk C alebo MicroPython, prípadne derivát MicroPythonu – CircuitPython. V tejto práci bol použitý práve CircuitPython.



Obr. 3.1: Mikrokontroléry TTGO. Prevzaté z [15]

Avšak aby bolo možné využívať tieto mikrokontroléry s technológiou LoRa, bolo potrebné pridať k nim nejaký LoRa modul.

Na implementáciu v tejto práci boli preto zvolené zariadenia Armachat, ktoré kombinujú mikrokontrolér Raspberry Pi Pico s LoRa modulom RFM96W. Okrem toho pridávajú dvoj palcový farebný displej a klávesnicu.

Ako toto zariadenie vyzerá môžeme vidieť na Obr 3.2.



Obr. 3.2: Zariadenie Armachat. Prevzaté z [17]

3.3.4 Raspberry Pi 2B

Jednodoskový, štvor-jadrový počítač vytvorený organizáciou Raspberry Pi. Toto zariadenie obsahuje eternetový port, ktorý môže slúžiť na pevné pripojenie do internetovej siete. Počítač sam o sebe neobsahuje žiadnen modul a tak bol pre prácu s LoRa pridaný RFM96W modul, pripojený do vstupno-výstupných pinov počítača.

Na pripojenie bola použitá SPI zberonica. Raspberry Pi ponúka dva SPI porty, v našom prípade bol použitý ten prvý na pinoch 11, 10, 9 a 8. Výsledné zostavenie môžeme vidieť na Obr 3.3.

Na rozdiel od predchádzajúcich mikrokontrolérov disponuje tento jednodoskový počítač oveľa väčším výkonom a pamäťou, avšak ma oveľa vyššiu spotrebu energie. Z tohto dôvodu tento počítač plnil rolu nemobilného uzlu v sieti, ktorý bol pripojený cez ethernet do internetovej siete.

Vďaka tomu, že CircuitPython je možné spojazdniť aj na Raspberry Pi, bolo možné časť implementácie zdieľať medzi Raspberry Pi Pico a Raspberry Pi 2B.



Obr. 3.3: Zariadenie Raspberry Pi 2B s LoRa modulom

Kapitola 4

Existujúce riešenia

Téma mesh sietí je v tejto dobe veľmi aktuálne a vývojári sa snažia vytvoriť rôzne riešenia, ktoré by nahradili aktuálne používane prístupy topológie hviezda v IoT sieťach.

V rámci sietového prenosu je klúčové zabezpečiť správne smerovanie, aby správy mohli byť úspešne doručené od odosielateľa k príjemcovi. Je k dispozícii celé množstvo rôznych smerovacích algoritmov, ktoré môžu byť využité k dosiahnutiu tohto cieľu, pričom každý z nich prináša svoje výhody a nevýhody [18]. Preto je dôležité zvážiť konkrétnu požiadavku a okolnosti siete a zvoliť ten najlepší algoritmus pre danú situáciu.

Použitie mesh topológie môže so sebou prinášať rôzne výhody oproti bežne zaužívaným topológiám typu hviezda. Vo výskume od Ochoa et al. [19] porovnávali spotrebu energie medzi topológiou typu mesh a hviezdicovou topológiou v sieti LoRa. A z týchto štúdií možno vyvodíť, že pre rozsiahlejšie siete je viac efektívnejšie, z hľadiska spotreby energie, použiť topológiu mesh.

Okrem toho je možné optimalizovať výber rádiových nastavení v súlade s hustotou siete tak, aby sa v mesh sieťach ešte viac optimalizovala spotreba energie.

Existujú rozvinuté projekty ako napríklad Meshtastic [20], ktorý sa snaží vytvoriť rozsiahlu decentralizovanú mesh sieť na miestach bez inej dostupnej konektivity, za použitia lacných zariadení.

Ďalším zaujímavým projektom je Armachat [21], ktorý ponúka možnosť komunikácie v prípade nedostupnosti ostatných sieti, napríklad po nejakej prírodnej alebo inej katastrofe. Súčasťou projektu Armachat sú plošné spoje, z ktorých si používateľ poskladá finálne zariadenie. Zariadenia Armachat sú poháňané mikrokontrolérmi Raspberry Pi Pico, obsahujú LoRa moduly, displej, klávesnicu a ďalšie vymožnosti.

Originálny projekt avšak sám o sebe zatiaľ nepodporuje využívanie mesh topológie. Používa ale rovnakú štruktúru správ ako projekt Meshtastic a vďaka tomu je možné v projekte Armachat využívať mesh sieť projektu Meshtastic.

Teoretických návrhov ako realizovať smerovanie v mesh sieťach je mnoho, avšak v praxi sa najčastejšie stretávame s distance vector smerovaním alebo flooding prístupom. Najzaujímavejšie riešenia si bližšie popíšeme v následujúcich sekciách.

4.1 LoRa mesher

LoRaMesher [22] [23] je C++ knižnica, ktorú je možné použiť na komunikáciu v LoRa mesh sietach.

Na smerovanie v sieti používa proaktívny¹ distance vector routing protokol. Tento protokol vyberá cestu, kadiaľ bude správa v sieti preposlaná od odosielateľa k príjemcovi, na základe najlepšej cesty. Najlepšiu cestu definuje ako cestu s najmenším počtom hopov – preskokov medzi uzlami v mesh sieti.

K realizácii distance vector smerovania je potrebné udržiavať smerovaciu tabuľku, ktorá obsahuje informácie o ID uzlov, cez ktoré susedné uzly sa dané uzly dajú dosiahnuť a koľko preskokov bude potrebných na dosiahnutie týchto uzlov. Smerovacia tabuľka je periodicky aktualizovaná cez špeciálny typ správ, ktoré sú odosielané všetkými uzlami v sieti. Túto smerovaciu tabuľku si drží a priebežne aktualizuje každý uzol v sieti.

LoRaMesher používa FreeRTOS, čo je operačný systém reálneho času. Takéto operačné systémy garantujú dokončenie úloh v určitom čase. FreeRTOS je použitý na zabezpečenie plánovania úloh. Rozličné úlohy sa starajú o prijatie a odoslanie paketov, iné úlohy sa starajú o samotné spracovanie prijatých paketov.

Pri odoslaní správ čaká na prijatie potvrdzujúcej ACK správy, ktorá potvrzuje, že správa bola prijatá a tým zaistuje spoľahlivosť. Správy väčšie ako 222 bajtov sú rozdeľované na viacero správ, ktoré sa pošú postupne.

4.2 Meshtastic

Meshtastic vytvára mesh siet za použitia lacných mikrokontrolérov s LoRa modulmi. Myšlienka tohto projektu spočíva v tom, že vytvára komunikačnú sieť na miestach kde neexistuje spoľahlivá infraštruktúra na bezdrôtovú komunikáciu (napr. v horách).

Posielanie správ v sieti je založené na jednoduchom multi-hop flooding. Každý uzol znova odvysiela paket, ktorý prijal (pokiaľ nedošiel počet preskokov na 0), až kým sa paket nedostane do určenej destinácie naprieč mesh sietou. Prenášane správy sú šifrované za pomoci šifrovacieho algoritmu AES.

Zariadenia používané v projekte Meshtastic majú okrem LoRa modulu zabudovaný aj bluetooth modul, vďaka ktorému je možné sa k zariadeniu pripojiť cez mobilný telefón, ktorý slúži ako rozhranie pre užívateľa. Cez aplikáciu v mobilnom telefóne môže používateľ vytvárať a prijímať správy. Správy sa cez bluetooth prenášajú z telefónu do zariadenia kde sa následne odošlú cez LoRa do siete.

Meshtastic poskytuje možnosť pripojenia sa k oficiálnemu meshtastic MQTT brokerovi. Toto umožňuje prepojiť malé lokálne mesh siete do väčšej globálnej siete a tak rozšíriť dosah sieti.

¹Smerovacia tabuľka sa pravidelne aktualizuje

4.3 LoRaBlink

Ďalší multi-hop flooding protokol, ktorý používa časovú synchronizáciu medzi uzlami. Časová synchronizácia definuje sloty, v ktorých môže uzol pristupovať ku prenosovému médiu a vysielať svoje dátá. Správy sa sietou šíria pomocou multi-hop flooding.

Sieť sa skladá z jedného uzlu, ktorý plní rolu takzvaného datasinku (gateway alebo brána) a ostatných uzlov. Uzly siete posielajú dátá do datasinku alebo dátá z neho prijímajú.

V určitých intervaloch datasink vyšle tzv. beacon signál. Tento signál slúži na časovú synchronizáciu medzi uzlami a značí začiatok novej epochy. Každá epocha obsahuje N slotov, v ktorých môžu uzly vysielať dátá. Beacon správa obsahuje hop count, ktorá udáva vzdialenosť ku datasinku. Ked nejaký uzol príjme beacon signál, vyšle svoj vlastný beacon signál v ďalšom volnom slote, ktorý vyberá na základe vzdialenosť od datasinku (hop count).

Ked uzol potrebuje poslať nejaké dátá, tak vyberie najskorší volný slot a v nom vysiela svoje dátá. Ak tieto dátá príjme uzol, ktorý nie je datasink a hop count daného uzlu ku sinku je menší ako hop count vysielacieho uzlu, tak dátá v ďalšom volnom slote znova prepošle. Toto sa opakuje, až kým dátá nedosiahnu datasink.

Takto tvorená sieť avšak vyžaduje existenciu nejakého hlavného uzlu (datasinku), ktorý je potrebný na riadenie siete prostredníctvom časovej synchronizácie.

4.4 Pymesh

Pymesh je súčasťou Pycom [24] ekosystému. Tento ekosystém je určený na vývoj IoT systémov. Ponúka rôzne zariadenia, ktoré sú určené na použitie s týmto ekosystémom. Zariadenia obsahujú WiFi, bluetooth a LoRa, prípadne SigFox moduly. Zariadenia je možné rozšíriť o rôzne moduly so senzormi, ktoré rozširujú ich schopnosti.

PyMesh sieť sa skladá z uzlov, ktoré môžu vystupovať v rôznych roliach. Sú nimi Border Router, Leader, Router a Child. Uzly typu gateway sú označované ako Border Routers a prepájajú LoRa sieť s internetom. Uzly typu router slúžia len na preposielanie paketov ďalej. Leader alebo koordinátorske uzly sú zodpovedné za vytváranie a správu siete a Child uzly sú koncové zariadenia. Na smerovanie paketov v sieti je využívaný reaktívny¹ distance vector smerovanie (AODV).

Uzly v sieti môžu komunikovať ad-hoc. V sieti môže dojsť k situácií kedy bude chcieť viacero uzlov vysielať v rovnakom čase a došlo by tak ku kolízii. Aby sa zabránilo takýmto situáciám, je použitá metóda Listen Before Talk, pri ktorej sa pred vysielaním signálu overí, či nie je v sieti už niekto iný, kto by práve vysielał. Pokiaľ áno, tak sa posielaná správa vyšle neskôr.

PyMesh je primárne určený na použitie s Pycom zariadeniami a použitie na inom zariadení by vyžadovalo väčšie úpravy zdrojového kódu.

¹Smerovacie informácie sú zasielané iba na vyžiadanie

Ak chce uzol vyslať nejaký paket a destinácia toho paketu je v dosahu daného uzla, tak sa paket priamo odošle. Pokiaľ ale destinácia nie je v dosahu, uzol najsíkôr vyšle požiadavok na vytvorenie trasy ku destinácii všetkým svojím susedom. Títo susedia to prepošlú ďalej svojím susedom. Takto sa bude paket preposielat, až kým sa nedostane ku uzlu, ktorého susedom je hľadaná destinácia prípadne destinácia samotná. Následne sa nájde najkratšia cesta ku destinácii a táto cesta sa uloží do smerovacích tabuľiek uzlov. Následne môže uzol vyslať pôvodný paket do destinácie.

4.5 Synchronous LoRa Mesh

Projekt Synchronous LoRa Mesh [25] vznikol z potreby získavania real-time dát z podzemných kanalizácií. Tieto dáta sú potrebné k monitorovaniu a predikcii kritických situácií akými sú napríklad záplavy.

LoRaWAN však nemá moc veľký dosah do podzemia. Z toho dôvodu by bolo potrebné v podzemných priestoroch implementovať LoRaWAN gateway-e, ktoré sú ale energeticky náročne, drahé a vyžadujú pevné pripojenie do elektrickej a prípadne aj internetovej siete, ktoré v kanalizačiach niesu dostupné. Okrem toho, aby museli byť všetky ostatné uzly v podzemnej sieti v dosahu danej gateway a pri väčšej podzemnej sieti by teda bolo potrebné implementovať viacerlo LoRaWAN gateway.

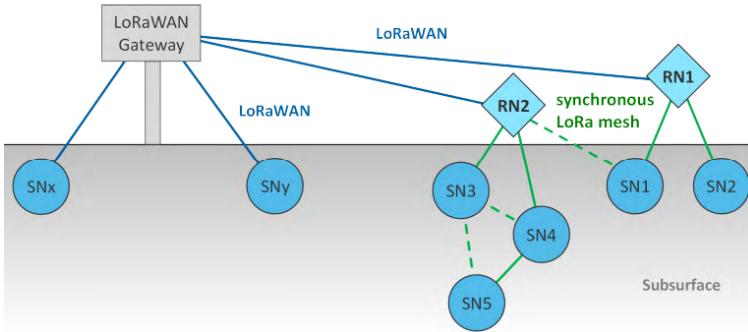
Tento projekt sa snaží vyriešiť tieto problémy. Prináša protokol, ktorý rozširuje LoRaWAN o tzv. repeater uzly (RN) viď Obr. 4.1. Tieto uzly sa vyskytujú na povrchu a preposielajú dátu z podzemných monitorovacích uzlov (sensor node - SN) do LoRaWAN siete. Monitorovacie uzly pod zemou tvoria mesh siet a RN plní funkciu riadiaceho uzlu pre podzemnú mesh siet. Komunikácia medzi RN a SN je synchronizovaná pomocou presného časovania, čo umožňuje koordináciu zmeny stavov SN z režimu spánku do normálneho režimu v čase, kedy potrebuje SN prijímať a odosielat dátu. Komunikácia sa cez uzly šíri multi-hop prístupom, za použitia smerovacej tabuľky.

Protokol používa na riadenie komunikácie metódu TDMA. RN priradí každej SN časový slot, v ktorom SN môže vyslať alebo prijať dátu. Monitorovacie uzly sú väčšinu času v režime spánku, zobudia sa len v ich určenom časovom slote a vďaka tomu majú tieto uzly nízku spotrebú energie.

Novo pripojený uzol do siete musí čakať na periodický beacon, vysielaný RN uzlom. Z beaconu sa zistí, aký časový slot je priradený danému uzlu a až po priradení slotu, môže uzol začať komunikovať v sieti.

4.6 Porovnanie a návrh nášho rozšírenia

Niektoré zo spomenutých riešení využívajú smerovacie tabuľky na efektívnejší prenos dát v rámci siete. Tie môžu byť však limitáciou pokial chceme dosiahnuť vyššej mobility uzlov. V prípade, že by sme chceli spolu so smerovacími tabuľkami podporovať mobilitu uzlov, je potrebné zabezpečiť dostatočné časté aktualizácie smerovacích tabuľiek.



Obr. 4.1: Schéma Synchronous LoRa Mesh. Prevzaté z [25]

Niektoré spomennuté riešenia limitujú komunikáciu medzi uzlami na vyhradené časové okná, mimo ktoré sú uzly v úspornom režime. To môže byť veľkou limitáciou pri niektorých aplikáciách, kde je potrebná komunikácia v reálnom čase (napr. chat).

V tejto práci sme navrhli protokol, ktorý nevyužíval žiadne smerovacie tabuľky. Vďaka tomu bola dosiahnutá vysoká mobilita uzlov a dynamickosť siete. Hlavným využitím nášho protokolu bola ad-hoc komunikácia, preto protokol nepoužíval žiadnu časovú synchronizáciu a časové okná vyhradené na komunikáciu, uzly siete tak mohli prijímať a odosielat dátu kedykoľvek to bolo potrebné. To so sebou ale prinášalo nevýhodu v podobe vyššej spotreby energie.

Funkcionalita navrhnutého protokolu, podobne ako niektoré zo spomínaných riešení, fungovala na základe multi hop flooding, kedy sa správa v sieti preposielala cez uzly siete až kým nedorazila do destinácie. Správa sa odoslala a uzol, ktorý túto správu prijal ako posledný ju preposal dalej. Toto sa opakovalo na ostatných uzloch v sieti až kým sa správa nedostala do destinácie alebo kym správe nedošiel limit preskokov, prípade TTL.

Navrhnutý protokol neboli závislý na žiadnych špeciálnych centrálnych uzloch typu gateway, prípadne nejakých riadiacich uzloch. Každý uzol v sieti mohol byť jednoduchým uzlom, ktorý prijímal a preposielal dátu dalej. Vďaka tomu mohli byť uzly realizované prostredníctvom lacných a malých zariadení.

Protokol bolo možné používať na rôznych platformách. V tejto práci vznikla implementácia protokolu pre mikrokontroléry používajúce programovací jazyk C++ a mikrokontroléry alebo jednodoskové počítače podporujúce CircuitPython.

Na šifrovanie obsahu správ bol použitý šifrovací algoritmus AES.

*Po zmene polohy mobilného uzla, je potrebné čakať pokým nastane ďalšia epocha

**Po zmene polohy mobilného uzla, je potrebné nájsť a aktualizovať cestu k nemu v smerovacích tabuľkách

Tabuľka 4.1: Porovnanie LoRa mesh protokolov

Protokol	LoRa mesher	Meshtastic	LoRaBlink	Pymesh	Synchronous LoRa Mesh	Náš protokol
Centrálne uzly	Nie	Nie	Áno	Áno	Áno	Nie
Mobilita uzlov	Áno**	Áno	Áno*	Áno**	Áno*	Áno
Smerovanie	Distance Vector	Flooding	Flooding	Distance Vector	Flooding	Flooding
Ad-hoc komunikácia	Áno	Áno	Nie	Áno	Nie	Áno

Kapitola 5

Vlastná implementácia

Ako sme už spomínali v predchádzajúcej kapitole, nami vytvorený protokol fungoval na základe multi hop flooding. Tento prístup so sebou ale niesol niekoľko nevýhod, ktoré sme museli zohľadniť pri navrhovaní implementácie.

Jednou z nevýhod bolo, že ak sme poslali niekam správu, a destinácia práve nebola v dosahu, tak sme o správu prišli. Tento problém bol vyriešený tak, že odosielateľ si odoslané správy ukladá a v prípade, že sa nepodarí správu doručiť, môže ju opäťovne odoslať niekedy neskôr. To, že sa správu nepodarilo doručiť do destinácie, sme zistovali vďaka potvrzovacím ACK správam.

Ďalším z problémov mohlo byť potenciálne zahľtenie siete správami. Keby každý uzol v sieti preposielal ďalej každú správu, ktorú prijal, tak by sme sa rýchlo dostali do stavu kedy by množstvo uzlov preposielalo rovnaké správy a siet by sa tym pádom zahltala. Riešenie tohto problému spočívalo v niekoľkých procesoch ako stavu zahľtenia predchádzajúceho. Tieto procesy si bližšie popíšeme v kapitole implementácie.

V tejto práci vznikli dve verzie implementácie. Jedna verzia, implementovaná v CircuitPython, bola určená pre výkonnejšie zariadenia. Táto implementácia obsahovala aj webové rozhranie na obsluhu, posielanie a prijímanie sprav. Webové rozhranie avšak šlo využiť len na zariadeniach, ktoré podporujú WiFi prípadne ethernet.

Implementácia v C++ bola určená pre menej výkonné zariadenia. Táto implementácia bola len zjednodušená verzia kompletnej implementácie a bola primárne určená pre mikrokontroléry TTGO. Z dôvodu výkonnosti a obmedzenej pamäti táto implementácia obsahovala iba základné funkcionality protokolu, potrebné na odosielanie správ s údajmi so senzorov. Mikrokontroléry TTGO tak v sieti fungovali iba ako jednoduché uzly, ktoré v určitých časových intervaloch odosielali dátu zo senzorov.

5.1 Typy a stavy správ

Predom sme si stanovili, že v našom protokole bude možné odosielat rôzne typy správ. Na základe typov správ, sa potom správy spracovávajú odlišne.

Okrem typov správ, bolo nutné si definovať aj určité stavy, v ktorých sa správa môže nachádzať. Tieto stavy potom určovali, ako sa správa spracováva.

5.1.1 Typy správ

Rozhodli sme sa v našom protokole implementovať dokopy sedem typov správ. Každý typ správy má svoju úlohu a využitie. Sú nasledovné:

ACK správa slúžila na potvrdenie doručenia správy. Keď odosielateľ pôvodnej správy niekedy ne-skôr prijal ACK správu, ktorá potvrdzovala doručenie správy, tak mohol pôvodnú správu považovať za doručenú.

Textová správa (TEXT_MSG) je správa, ktorá obsahovala textový reťazec. Rozšírením tejto správy je správa s potvrdením o doručení (TEXT_MSG_W_ACK). Obsah textových správ je šifrovaný.

Správa typu senzorové dát (SENSOR_DATA) bola určená na prenos dát z rôznych senzorov. Obsah správy bol taktiež vo forme textového reťazca, ktorý obsahoval dátu zo senzorov. Tento typ správ avšak oproti obyčajnej textovej správe ponúkal možnosť nastaviť časový interval – TTL, po ktorom uplynutí sa správa nebude ďalej šíriť sietou. Obsah správy bol taktiež šifrovaný.

Správa žiadosť o traceroute (TRACEROUTE_REQUEST) bola používaná len v krajiných prípadoch, kedy nás zaujímalо akou cestou sa správa preposiela sietou. Keď nejaký uzol prijal správu typu žiadosť o traceroute, tak vytvoril novu správu typu odpoveď na traceroute (TRACEROUTE) a vložil do nej svoju adresu. Adresátom tejto novej správy bol odosielateľ pôvodnej správy. Keď sa táto odpoveď na traceroute šírila sietou, tak každý uzol, ktorý danú správu preposlal ďalej, pridal do nej svoju vlastnú adresu. Takto sa vytvoril reťazec, ktorý obsahoval adresy všetkých uzlov, cez ktoré bola správa preposlaná. Aby mohli ostatné uzly v sieti do správy postupne pridávať svoje adresy, tak obsah tejto správy nemohol byť šifrovaný.

Posledným typom správy je nespracovaný paket (RAW_PACKET). Tento typ správy bol využívaný iba v prípade, že ma používateľ zapnutý monitorovací režim. V tomto režime sa zaznamenávali všetky správy, ktoré sa v sieti prenášali. V tomto režime bolo možné zachytiť aj pakety, ktoré neboli súčasťou nášho protokolu a ich štruktúra nezodpovedala našim špecifikáciám. Z toho dôvodu ich nebolo možné spracovať a uložili sa teda ako typ správy nespracovaný paket. Ich obsah tvorili iba holé prijaté dátá vo forme bajtov.

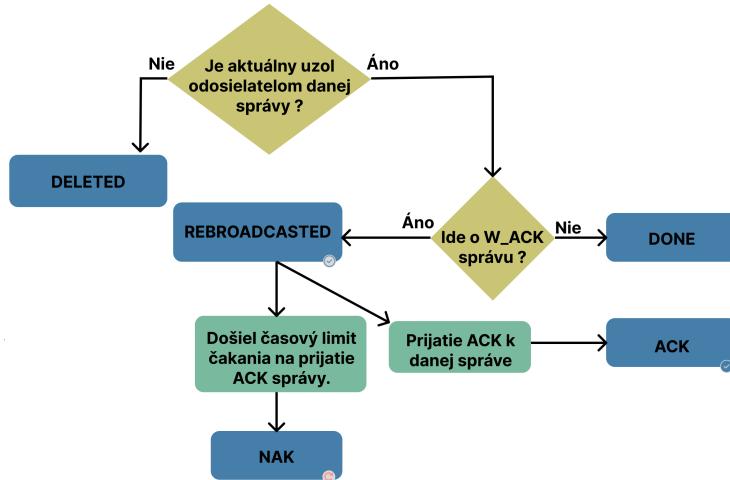
5.1.2 Stavy správ

Na správnu funkčnosť bolo potrebné aby správy existovali v určitých stavoch. Správy časom svoj stav aktualizovali a prechádzali medzi nimi. Tieto stavy boli definované nasledovne:

Stav nová správa (NEW). Tento stav bol určený pre správu, ktorá bola práve vytvorená. Nebola ešte ani raz odoslaná aktuálnym uzlom, a čaká na svoje prvé odoslanie. Po prvom odoslaní sa správa presunula do stavu odoslaná (SENT).

Zo stavu odoslaná bolo možné správu presunúť do stavu neúspešná (FAILED), hotová (DONE), preposlaná (REBROADCASTED) alebo zmazaná (DELETED). Ak aktuálny uzol prijal tu istú správu, odoslanú iným uzlom, znamená to, že dana správa sa šíri ďalej v sieti. Aktuálny uzol teda správu presunul do stavu preposlaná alebo do stavu zmazaná. To či sa správa presunula do stavu zmazaná alebo preposlaná bolo závisle od toho, či bol aktuálny uzol zároveň aj autorom danej správy. V prípade, že bol autorom, správa sa presunula do stavu preposlaná.

Správy v stave preposlaná, pri ktorých nepotrebujueme sledovať, či správa dorazila do destinácie, sa následne presunuli do stavu hotová. Prechody medzi týmito stavmi su znázornené v časti stavového diagramu na obrázku 5.1. Kompletnejší stavový diagram prechodov medzi stavmi správ je vyobrazený na obrázku v prílohe 1.



Obr. 5.1: Časť stavového diagramu

Pokiaľ aktuálny uzol vyčerpal všetky pokusy o odoslanie správy, prípadne pri správe typu senzorové dát došiel časový limit, správa sa presunula do stavu neúspešná alebo zmazaná. To či sa presunula do stavu neúspešná alebo zmazaná bolo znova závisle na tom, či bol aktuálny uzol autorom danej správy alebo nie.

Správam, ktoré boli v stave preposlané sa odpočítaval časový limit na prijatie potvrdenia. Pokiaľ v tomto časovom intervale nedorazila potvrzovacia ACK správa, tak sa správa presunula do stavu nepotvrdená (NAK). V opačnom prípade sa presunula do stavu potvrdená (ACK).

Stav zmazaná značil, že správu považujeme za vymazanú. Ďalej s ňou už nebudem pracovať a po uplynutí určitého intervalu sa reálne vymazala z pamäti.

5.2 Návrh paketu

Paket obsahoval hlavičku a dátový rámec. Štruktúra dátového rámcu bola závislá od typu správy. Navrhnutú štruktúru hlavičky paketu môžme vidieť v tabuľke 5.1

Tabuľka 5.1: Štruktúra hlavičky paketu a dĺžka jednotlivých polí v bajtoch.

2B <i>uint</i>	2B <i>uint</i>	4B <i>uint</i>	2B <i>uint</i>	1B <i>uint</i>	1B <i>uint</i>	0 – 240B
Destinácia	Odosielateľ	ID správy	Kontrolný súčet	Typ správy	Priorita	Dátový rámec

Pre adresy odosielatela a destinácie boli využité dvoj bajtové adresy. Pre broadcast adresu bola rezervovaná hodnota 0xFFFF. Štvor bajtový identifikátor správy bol náhodne generovaný pri každom vytvorení novej správy.

Z adresy destinácie, adresy odosielatela a identifikátoru správy bol vytvorený dvoj bajtový kontrolný súčet. Tento kontrolný súčet bol používaný na overenie, či prijatá správa patrila do nášho protokolu. Overovať integritu dát na základe kontrolného súčtu nie je za potreby, pretože táto kontrola je už zahrnutá na fyzickej vrstve LoRa technológie.

Položka typ správy obsahovala číslo, ktoré definovalo typ danej správy. Číslo reprezentovalo jeden z typov spomínaných v predošej sekcií.

Správam bolo možné v hlavičke paketu určiť vyššiu prioritu. Na túto prioritu sa následne bral ohľad pri spracovávaní správ a správy s vyššou prioritou boli odbavené prioritne. Rozhodli sme sa poskytovať iba dva stupne priority správ a to normálna priorita a vysoká priorita.

Dátové rámce sa líšili na základe typu správy. V tabuľkách 5.2, 5.3 a 5.4 sú uvedené štruktúry dátových rámcov pre jednotlivé typy správ.

Tabuľka 5.2: Štruktúra dátového rámcu pre textové správy.

1B <i>uint</i>	1B <i>uint</i>	0 – 238B <i>String</i>
Počet skokov	Pôvodný počet skokov	Textové dátá

V niektorých dátových rámcach môžme vidieť položku počet skokov. Do tejto položky bol pri vytvorení novej správy zapísaný maximálny počet skokov, ktoré môže správa vykonať pri prenose sietou. Každý uzol, predtým ako správu preposal dalej, zmenšíl počet skokov o jedna. Ak sa počet skokov dostal na nulu tak sa daná správa dalej nespracovávala a nastavil sa jej adekvátny stav.

V rámcach tiež nájdeme položku pôvodný počet skokov. Do tejto položky bol tak isto pri vytvorení novej správy zapísaný maximálny počet skokov. Hodnota v tejto položke sa však už dalej

1B <i>uint</i>	1B <i>uint</i>	1B <i>uint</i>	0 – 239B <i>String</i>
Počet skokov	Pôvodný počet skokov	Počet skokov	Navštívené adresy

(a) Správa typu žiadosť o traceroute.

(b) Správa typu odpoved na traceroute.

Tabuľka 5.3: Štruktúra dátového rámcu pre správy typu traceroute.

nemenila. Slúžila k tomu, aby prijímateľ dokázal zistieť, kolko preskokov správe zabralo, kým správa dorazila až k nemu. Túto informáciu bolo možné následne použiť na správne nastavenie maximálneho počtu skokov pre odoslanie potvrdzovacej správy, prípadne správy typu odpoveď na traceroute.

1B <i>uint</i>	4B <i>uint</i>	2B <i>uint</i>	0 – 238B <i>String</i>
Počet skokov	ID potvrdzovanej správy	TTL	Senzorové dátá

(a) Potvrdzovacie ACK správy.

(b) Správy pre senzorové dátá.

Tabuľka 5.4: Štruktúra dátového rámcu pre ACK a senzorové správy.

V dátovom rámci, používanom na senzorové dátá, bol použitý, namiesto maximálneho počtu preskokov, TTL. TTL vyjadruje časový interval, po ktorom mohla byť daná správa spracovaná a preposielaná v sieti. Táto hodnota sa pri vytvorení správy nastavila na predom určený časový interval v sekundách. Každý uzol, ktorý danú správu preposielal, si overil ako dlho už bola správa uložená u neho v pamäti a na základe toho odčítal hodnotu z TTL. Po odčítaní hodnoty z TTL, uzol správu ďalej preposal.

Pri odčítavaní TTL bol braný do úvahy iba čas, ktorý daná správa strávila na určitom uzle. Nebral sa do úvahy čas, ktorý správe zabralo preniest sa rádiovými vlnami z jedného uzlu na ďalší. A to z toho dôvodu, že sme neboli schopní presne určiť, akú vzdialenosť prekonala vysielaná správa rádiovými vlnami. Existujúce spôsoby merania vzdialnosti, ktorú správa prešla vzduchom, produkovali len hrubé odhady a boli veľmi závisle od aktuálnych atmosférických podmienok [26]. Preto sme sa rozhodli, že v našom riešení budeme počítať iba s časom, ktorý správa strávila na určitom uzle. Mimo toho, keby sme počíタli s maximálnou možnou vzdialenosťou medzi dvoma uzlami, ktorá sa pri LoRa udáva okolo 15 kilometrov v otvorenom priestranstve, a rýchlosťou ktorou sa šíria rádiové vlny, ktorá je vo vákuu rovná rýchlosťi svetla, tak by sme zistili, že prenos správy medzi týmito dvoma uzlami by v ideálnom prípade trval okolo 50 mikrosekúnd. Táto hodnota bola v porovnaní s časom, ktorý správa strávi na určitom uzle, zanedbateľná.

5.3 Funkcionalita protokolu

Navrhnutý protokol fungoval tak, že každý uzol si u seba uložil novo prijatú správu. Následne boli uložené správy spracovávané v hlavnom cykle programu. Každú správu, ktorú uzol prijal, sa následne pokúšal znova preposlať ďalej, pokiaľ teda nebola daná správa určená práve tomu uzlu.

Uzol každú správu odosielal iba určitý počet krát a medzi každým odoslaním konkrétnej správy uzol čakal predom stanovený časový interval predtým ako ju mohol znova odoslať. Toto viacnásobne odosielanie každej správy nám zabezpečilo vyššiu spoľahlivosť v doručení. Keby sa každá správa preposielala iba jeden krát, mohlo by sa staf, že na ten prvý raz by tuto vyslanú správu nezachytil žiadnený ďalší uzol a správa by sa tak nedoručila ďalej.

Medzi tým ako boli správy spracovávané a preposielané, uzol zároveň počúval na novo prichodzie správy od iných uzlov. Ak náhodou uzol prijal správu, ktorá už bola prijatá a uložená u neho tak to považoval ako potvrdenie, že sa správa šírila ďalej v sieti. Uzol teda u seba danú správu zahodil a už ju ďalej nepreposielal.

Ak došlo k situácií, kedy uzol jednu správu opakovane preposlal maximálny možný počet pokusov a medzitým neprijal danú správu preposlanú iným uzlom, tak bola považovaná táto situácia ako neúspešne odoslanie správy. Správe sa nastavil adekvátny stav a nebola ďalej spracovávaná.

Každý uzol prijatej správe znížil hodnotu maximálneho počtu preskokov. Pokiaľ sa však jednalo o správy typu senzorové dát, tak sa neznižovala hodnota maximálneho počtu preskokov ale hodnota TTL, a to pred každým znova odoslaním danej správy. Pokiaľ však uzol prijal správu, ktorej hodnota maximálneho počtu skokov dosiahla nulu, tak táto správa nebola u daného uzla ani ukladaná. Ak sa počas spracovávania správ na uzele dostala hodnota TTL nejakéj senzorovej správy na nulu, tak táto situácia bola tak isto považovaná za neúspešne odoslanie správy. Správe sa nastavil adekvátny stav a nebola ďalej spracovávaná.

Obsah textových a senzorových správ bol vždy šifrovaný, pokiaľ nebola správa posielaná na broadcast adresu. Na šifrovanie bol použitý šifrovací algoritmus AES, ktorý šifruje bloky dát za pomoci klíča. Jedna sa o symetrickú šifru, takže zašifrované dátu bolo možné rovnakým klíčom neskôr dešifrovať. To ponúkalo možnosť aby si určitá skupina používateľov vytvorila vlastný šifrovací klíč a iba oni boli schopní dešifrovať svoje správy, ktoré boli vytvorené za použitia tohto klíča. Ostatní účastníci siete obsah správ neboli schopní dešifrovať.

V protokole bolo možné odosielať správy aj na broadcast adresu. Tieto správy boli potom prijaté každým uzlom.

5.4 Implementácia protokolu

Hlavnou súčasťou nami navrhнутej implementácie protokolu bol dynamický zoznam správ. Do tohto zoznamu boli pridávané novo vytvorené, prípadne novo prijaté správy. Následne bol tento zoznam správ v hlavnom cykle programu prechádzaný a každá správa bola spracovávaná.

Na to aby bolo možné správy do tohoto zoznamu ukladať, bolo potrebné vytvoriť istú dátovú štruktúru, do ktorej by sa ukladali jednotlivé správy. Správy v zozname so sebou držali rôzne informácie. Okrem tých základných informácií o správe akými boli napríklad jej identifikátor, odosielateľ a prijímateľ, obsah správy atď., bolo potrebné držať spolu so správou aj dodatočné informácie, ktoré boli potrebné na jej následne spracovávanie. Do týchto dodatočných informácií patrili napríklad čas, kedy bola správa naposledy spracovávaná aktuálnym uzlom. Táto informácia bola využitá pri kontrole či uz nadišiel čas správu znova preposlať ale aj na to aby bolo možné zistíť kolko času treba odčítať z hodnoty TTL pri senzorových správach.

Pri prechádzaní zoznamu správ sa pri každej správe skontroloval jej stav a to či časový limit už dosiahol požadovanej hodnoty. Opakované znova odosielanie správy, bolo vykonávané iba pri správach v stave nová a odoslaná. Spomínali sme, že je bolo možné určiť správe vyššiu prioritu. Táto vyššia priorita zabezpečila to, že sa pri správe, ktorá má nastavenú vyššiu prioritu, vykonávalo opakované znova odosielanie správy, aj keď jej časový limit ešte neboli dosiahnutý. Tým bolo dosiahnuté toho, že správa sa skôr odoslala a do destinácie by mala v ideálnom prípade doraziť skôr ako správa s nižšou prioritou.

Spomínali sme tiež, že môže dôjsť k problému zahľtenia siete, keby každý uzol v sieti preposielal všetky správy, ktoré príjme. Spôsob akým sme vzniku tohto problému zabránili spočíval práve v tom, že keď uzol prijal správu, ktorú už mal u seba uloženú, tak u seba tuto správu označil ako zmazanú (stav zmazaná) a správa nebola ďalej preposielaná. Správy v zmazanom stave neboli ale ihneď vymazané z pamäti. Miesto toho sa im nastavoval časový interval a až po tom čo uplynul tento interval boli správy skutočne vymazané z pamäti. Vďaka tomu, že sme si držali v pamäti správy aj keď boli považované za zmazané, bolo možné overiť či novo prijatá správa nebola náhodou jedná z tých zmazaných. Ak áno tak nová správa nebola znova ukladaná. K tejto situácii môže bežne dochádzať a neošetrenie tohto problému by mohlo viest k zackyleniu, kedy by sa jedna a ta istá správa neustále preposielala medzi dvoma uzlami až kým by jej nedošiel limit skokov prípadne TTL.

Aby bolo v mesh sieti dosiahnuté čo najlepšieho dosahu, bolo nutné nejak zabezpečiť, aby sa odosielané správy dostali k čo najvzdialenejším uzlom a aby tieto najvzdialenejšie uzly boli tie ktoré propagujú prenášanú správu ďalej. To bolo zabezpečené tak, že každej novo prijatej správe bolo nastavené prvotné časové oneskorenie na základe toho s akou hodnotou SNR bola daná správa prijatá.

Hodnota SNR vyjadruje pomer signálu k šumu. Čím je hodnota SNR vyššia, tým je signál silnejší oproti okolitému šumu. Vďaka tomu, bolo možné približne odhadnúť, ktorý uzol bol vzdialenejší. Uzly ktoré boli vzdialenejšie mali nižšiu hodnotu SNR a teda aj nižšiu hodnotu časového oneskorenia. To poviedlo k tomu, že vzdialenejšie uzly odvysielali správu skôr, ako uzly ktoré boli bližšie. Ostatné uzly toto vysielanie prijali, a u seba označili danú správu ako zmazanú, keďže už nebolo potrebné aby ju oni preposielali. Na určenie časového oneskorenia bol použitý následovný výpočet vyobrazený na Obr 5.2. Výpočet transformuje hodnotu SNR prijatého paketu na časové oneskorenie v intervale od 1 do 6 sekúnd, pričom ráta s minimálnou hodnotou SNR -20 dB a maximálnou hodnotou 20 dB.

```

def get_timeout(self, snr):
    if self.config.RANDOMIZE_PATH:
        return 150*random.randint(5, 15)

    SNR_min = -20
    SNR_max = 20
    timeout = int(((snr - SNR_min) / (SNR_max - SNR_min)) * 5000)+1000
    return timeout if timeout > 0 else 0
#Output min: 1000, max: 6000 (with SNR range of -20 to 20)

```

Obr. 5.2: Výpočet časového oneskorenia

Na obrázku 5.2 môžme taktiež vidieť, že ak je v konfigurácii zapnutá možnosť náhodnej cesty (*RANDOMIZE_PATH*), tak sa hodnota časového oneskorenia nastavuje náhodne. Jedná sa o experimentálne nastavenie a jeho funkčnosť si bližšie popíšeme v kapitole testovania funkčnosti.

Ďalší z potrebných aspektov správneho fungovania nášho protokolu bolo zabezpečiť aby nedošlo k situácií kedy by dva alebo viac uzlov v sieti začalo vysielat signály v rovnakom čase. K tomu bol využitý takzvaný prístup listen before talk, kedy sa uzol predtým ako začal vysielat presvedčil, či náhodou práve nevysielal niekto iný v jeho dosahu. Ak áno tak tak sa vyslanie správy pozdržalo a skúšilo sa odoslať neskôr.

Rozhodli sme sa implementáciu protokolu začať na zariadeniach Armachat. Keďže sme zatiaľ nemali vytvorené žiadne užívateľské rozhranie, tak veľké farebné displeje na zariadeniach Armachat boli vhodné na zobrazovanie informácií o prebiehajúcich procesoch a komunikácií v sieti. Na programovanie zariadení Armachat bol použitý CircuitPython, ktorý ma ale rovnakú syntax ako programovací jazyk Python.

Ako prvé bolo potrebné vytvoriť vhodnú dátovú štruktúru, ktorá by uchovávala v sebe správu. K tomu bola vytvorená trieda *Message*. Trieda obsahovala všetky potrebné informácie o správe. Taktiež obsahovala metódy na vytváranie rôznych typov a obsahov správ. Tieto metódy slúžili na prvotné vytvorenie nových správ, na autorskom uzle. Okrem týchto metód bolo ale potrebné mať aj metódy, ktoré dokázali z prijatých dát vytvoriť správu. Pri prijatí nového paketu v LoRa, sme dostali dátá vo forme bajtového poľa. Tieto dátá boli potom predané do metódy, ktorá z nich vytvorila správu.

Po implementácii triedy *Message*, sme teda boli schopní vytvoriť novu správu alebo prijať bajtové pole z LoRa, z ktorého sa následne poskladala správa. Na to aby sme mohli správy ukladať do zoznamu správ bolo ale potrebné vytvoriť ešte akýsi kontajner, ktorý by v sebe uchovával správu plus dodatočne informácie potrebne k ďalšiemu spracovávaniu. K tomu vznikla trieda *MessageQueueItem*. Tá v sebe držala inštanciu triedy *Message*, počítadlo opakovanych odoslaní, časový odpočet, časové razítko, ktoré zachytávalo čas, kedy bola správa naposledy spracovaná, aktuálny stav správy a ďalšie užitočné informácie. Okrem toho obsahovala táto trieda metódy, ktoré boli vy-

užívané pri spracovávaní správ. Sem patria metódy na zníženie maximálneho počtu preskokov alebo TTL, metóda na zmenu stavu správy, metóda na aktualizovanie časového razítka, ktorá časové razítko nastavila na aktuálny čas a mnoho ďalších.

Po implementácii triedy *MessageQueueItem* sme mohli začať implementovať samotný proces funkcionality nášho protokolu. Zoznam správ bol realizovaný ako dynamický slovník, kde klúčom bol identifikátor správy, a hodnota pod daným klúčom bola inštanciou triedy *MessageQueueItem*. To nám umožnilo rýchlo vyhľadávať správy na základe jej identifikátora. Tento slovník sme pomenovali *message queue* a tak ku nemu budeme referovať aj vo zvyšku tejto práce.

Hlavný proces protokolu pozostával z cyklu, v ktorom bolo najskôr skontrolované či náhodou neboli prijatý nový LoRa paket a následne boli prejdené všetky správy z *message queue*. Ak to bolo potrebné tak sa určité správy spracovali. K tomu vznikli dve hlavné funkcie. Funkcia *receive()*, ktorá slúžila na skontrolovanie či neboli prijatý nový LoRa paket a funkcia *tick()*, v ktorej boli prejdené všetky správy z *message queue*.

Funkcia *receive()* prečítala novo prijatý paket z LoRa a následne overila, či daný paket spĺňa dĺžku aspoň 12 bajtov. Vychádza to z toho, že všetky správy nášho protokolu obsahujú predom definovanú hlavičku paketu (viď. Tab. 5.1), ktorá mala veľkosť 12 bajtov. Ak bol paket kratší ako 12 bajtov, bolo možné s istotou povedať, že sa nejednalo o správu nášho protokolu a teda ho nebolo nutné ďalej spracovať. Po kontrole dĺžky paketu, nasledovalo overenie kontrolného súčtu. Funkcia zobražovala prvých 8 bajtov paketu a vypočítala sa z nich kontrolný súčet. Ak sa vypočítaný kontrolný súčet zhodoval s kontrolným súčtom, ktorý bol v pakete, tak bolo možné správu považovať za súčasť nášho protokolu. Na výpočet kontrolného súčtu bola použitá funkcia cyklicky redundantného súčtu typu CRC-16/CCITT-False. Overovanie, či paket prísluší do nášho protokolu bolo pomerné dôležité, pretože na rovnakých LoRa parametroch mohli byť vysielané aj pakety iných protokolov a tieto pakety by sa nam nepodarilo správne spracovať keďže by nesplňali našu štruktúru paketu.

Potom čo sa vo funkcií *receive()* overilo, že paket spĺňa naše požiadavky, bola z neho vytvorená nova inštancia triedy *Message*. Bola k tomu využitá metóda triedy *Message* určená na vytvorenie správy z pola bajtov. Pri novo prijatej správe nás zaujímalо či je správa určená nám. Ak bola správa určená nám, bolo potrebné zistiť o aký typ správy sa jednalo. Pokiaľ bola prijatá správa typu ACK, tak bolo nutné vyhľadať v *message queue* korešpondujúcu správu a zmeniť jej stav na ACK. Pokiaľ sa nejednalo o ACK správu, nasledovala kontrola, či sa daná správa už nenachádzala v *message queue* a ak nie tak tam bola pridaná.

Bolo dôležité, aby uzol po prijatí správy, odoskal naspäť ACK správu. A to aj v prípade, že sa nejednalo o typ správy, ktorý vyžadoval potvrdenie o doručení. Dôvodom bolo to, že ak by uzol neposlal naspäť ACK správu, tak by predošlý uzol pokračoval v opakovacom vysielaní správy, a tieto vysielania by zbytočne obsadzovali prenos v sieti. Uzol teda po prijatí vyslal ACK správu. V prípade, že prijatá správa nevyžadovala potvrdenie o doručení, tak mohol uzol ACK správe nastaviť maximálny počet preskokov na 0. Tým pádom sa správa neposunula ďalej v sieti ako k najbližšiemu uzlu. Ak sa ale jednalo o správu vyžadujúcu potvrdenie o doručení, tak sa maximálny

počet preskokov nastavil na hodnotu, ktorá bola uložená v prijatej správe pod položkou pôvodný počet skokov. Tym bolo zaručené, že správa bude mať dostatočný počet dostupných skokov aby sa dostala naspäť k uzlu, ktorý pôvodnú správu poslal.

V prípadnom scenári, že prijatá správa nebola určená nám, bolo skontrolované či sa rovnaká správa už nenachádzala v *message queue* a ak nie tak tam bola pridaná za predpokladu, že mala ešte dostupný nejaký počet skokov alebo TTL. Správe bolo zároveň nastavené prvotné časové pozdržanie na základe hodnoty SNR, s ktorou bola prijatá.

V opačnom prípade, kedy sa správa už nachádzala v *message queue*, bolo nutné znova vykonať sériu overení a na základe nich zmeniť stav správy. Prvé z overení bolo, či daná správa bola vytvorená nami. To či bola vytvorená nami, sme zistili na základe toho, či sa adresa odosielateľa správy zhodovala s tou našou. V tom prípade nastala situácia kedy našu správu preposlal nejaký iný uzol. Bolo možné teda správe zmeniť stav na hotovo, prípadne na stav preposlaná ak sa jednalo o správu s potrebným potvrdením o doručení.

Ak sa ukázalo, že správa nebola vytvorená nami ale niekým iným, tak bola správa presunutá do stavu zmazaná. Do tohto stavu bola ale presunutá iba v prípade overenia, že hodnota počtu preskokov alebo TTL z prijatej správy bola nižšia ako hodnota správy v *message queue*.

Po dokončení funkcie *receive()*, nasledovala funkcia *tick()*. V tejto funkcií bolo postupne prechádzane všetkými spravami z *message queue*. Pri každej správe bol overený jej stav a ak bola v stave preposlaná alebo zmazaná tak sa následne skontrolovalo, či jej už nevypršal časový limit. Pokial limit vypršal, tak sa správa skutočne vymazala z *message queue* ak bola v stave zmazaná. Ak bola v stave preposlaná, tak sa jej stav zmenil na nepotvrdená – NAK.

V prípade, že bol stav správy nová alebo odoslaná, tak sa pokračovalo v jej spracovávaní, ktoré pozostávalo z následujúcich krokov. Ako prvé bolo overené, či už nadišiel čas danú správu spracovať. To bolo zistené tak, že sme sa pozreli či správe už vypršal časový limit. Ak áno, znamenalo to, že správa bola pripravená na spracovávanie a bolo možné pokračovať ďalej. Ak nie tak sa daná správa preskočila a prešlo sa na ďalšiu správu.

Ak nadišiel čas správu spracovať, tak bolo skontrolované, či má ešte dostupné nejaké pokusy o odoslanie a v prípade senzorových správ aj to, či bol ešte dostupný TTL. Ak došlo k situácii, že správa už vyčerpala všetky pokusy o odoslanie alebo jej TTL došlo na nulu, tak bola správa presunutá do stavu neúspešná alebo zmazaná ak to nebola správa vytvorená nami. Ak správe ešte zostali nejaké pokusy o odoslanie, tak bola správa odoslaná cez LoRa. Ked sa jednalo o správu typu senzorové dátá, tak sa správe pred odoslaním zmenšila hodnota TTL.

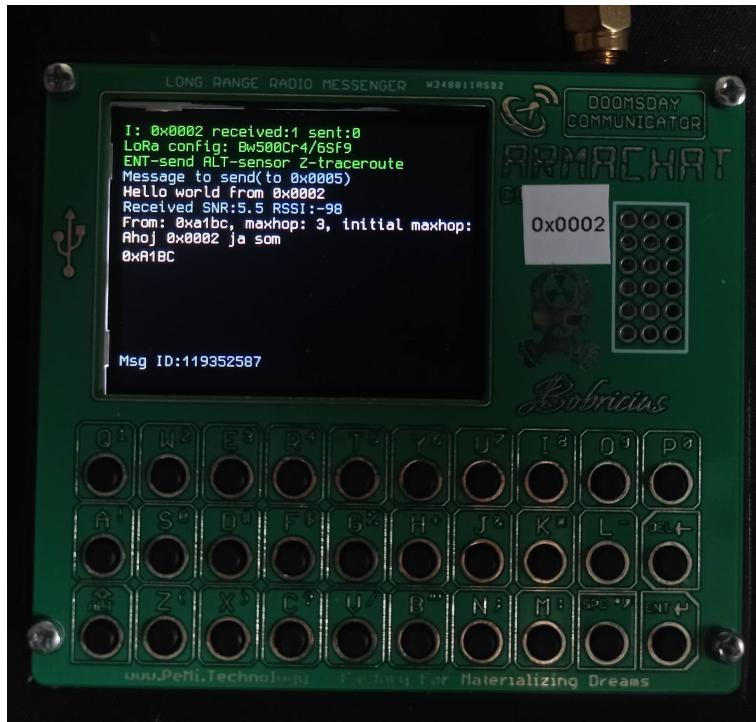
Po odoslaní bolo správe aktualizované aktuálne časové razítko, znížilo sa počítadlo dostupných pokusov o odoslanie a bol nastavený nový časový limit pre správu. Tento limit bol závislý od konfiguračnej premennej, ktorá určovala, ako často sa ma správa znova odosielať. Pokial bola správa pred odoslaním v stave nová, tak jej stav bol aktualizovaný na odoslaná.

Pri pokuse o odoslanie mohlo dôjsť k situácií, kedy práve niekto iný vysielal signál a tak bolo nutné pozdržanie odoslania. To bolo realizované tak, že správe bolo nastavené časové pozdržanie na

hodnotu konfiguračnej premennej a bolo aktualizované jej časové razítko. Výsledkom bolo, že táto správa sa pokúšala znova odoslať až po uplynutí časového intervalu.

5.4.1 Overenie jednoduchej funkcionality

Po dokončení implementácie hlavnej funkcionality sme ju mohli otestovať. Na zariadeniach Armachat sme vytvorili jednoduché rozhranie, ktoré zobrazovalo na displeji informácie o novo prijatej správe. Taktiež bolo možné využiť klávesnicu zariadenia, na napísanie vlastnej správy a jej odoslanie. Na obrázku 5.3 môžeme vidieť, že zariadenie prijalo správu od odosielateľa s adresou 0xA1BC, ktoré bolo v tomto prípade druhé Armachat zariadenie.



Obr. 5.3: Rozhranie zariadenia Armachat

Okrem obsahu správy, môžme vidieť hodnoty SNR a RSSI, s ktorými zariadenie správu prijalo, hodnotu max hop, ktorá predstavovala počet preskokov a hodnotu initial max hop, ktorá predstavovala pôvodný maximálny počet preskokov. Na základe rovnosti týchto dvoch hodnôt, môžme usúdiť, že správa prešla medzi dvoma zariadeniami bez toho aby prešla cez nejaké iné tretie zariadenie. Na spodku displeja môžme vidieť identifikátor prijatej správy.

Na ďalšom obrázku 5.4 môžme vidieť, že rozdiel medzi hodnotami max hop a initial max hop bol 1. Indikuje to, že správa spravila preskok cez ďalšie zariadenie predtým ako bola doručená na cieľové zariadenie. Tento test sme vykonali tak, že sme tri zariadenia rozmiestnili s určitou vzdialenosťou



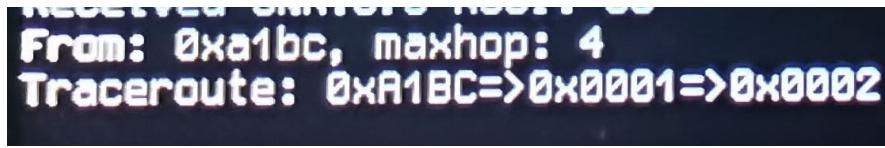
Obr. 5.4: Zariadenie Armachat prijalo spravu s jedným preskokom

medzi nimi, tak aby sme docielili preskok cez jedno zariadenie. Zariadenia mali nastavené adresy 0x0001, 0x0002 a 0xA1BC. Správa bola odoslaná zo zariadenia 0xA1BC na zariadenie 0x0002.

Taktiež si môžeme všimnúť zmenu v podobe skrátenia názvov atribútov max hop a initial max hop, z dôvodu, že sa na displeji nezmestili do jedného riadku. Okrem toho je na obrázku vidieť informačnú správu v podobe žltého textu na samom spodku displeja. V tejto informačnej správe sa zobrazujú rôzne informačné hlášky počas behu programu. Zariadenie 0x0002 prijalo správu od zariadenia 0xA1BC a následne vytvorilo ACK správu, ktorej ID bolo 1787352650. V informačnej správe môžme vidieť, že v dobe vyhotovenia fotografie sa akurát táto ACK správa pridávala do *message queue*.

Následne sme vyskúšali, pri rovnakom rozmiestnení zariadení, odoslať zo zariadenia 0x0002 žiadosť o traceroute na zariadenie 0xA1BC. Na obrázku 5.5 môžeme vidieť, ako zariadenie 0x0002 prijalo odpoveď na žiadosť o traceroute. V obsahu správy vidíme zoznam adries, cez ktoré správa putovala.

V tabuľke 5.5 môžeme vidieť vygenerovaný paket vo forme bajtov, ktorý bol odoslaný zo zariadenia 0xA1BC na zariadenie 0x0002. Obsah správy bol v tomto prípade, textový refazec „Ahoj“. V prvých 12 bajtoch, vidíme hlavičku paketu. V ďalších 6 vidíme počet preskokov, pôvodný počet preskokov a šifrovaný obsah správy.



Obr. 5.5: Zariadenie Armachat prijalo traceroute správu

Tabuľka 5.5: Ukážka bajtov odoslaného paketu

Hlavička										Dátový rámec								Initial max hop				Šifrovaný obsah správy					
Destinácia		Odosielateľ		ID správy				Kontrolný súčet		Typ správy		Priorita správy		Max hop		Initial max hop				Šifrovaný obsah správy							
0x00 0x02 0xA1 0xBC 0xEF 0x42 0x5D 0xC2 0xF2 0x64 0x01 0x00 0x02 0x03 0xA4 0x4A 0x33 0x56																											

5.5 Implementácia API pre webové rozhranie

Pri používaní nášho protokolu mal používateľ možnosť využiť webové rozhranie, cez ktoré bolo možné ovládať zariadenie a pracovať s naším LoRa protokolom. Používateľ mal možnosť vytvárať alebo prijímať nove správy, ktoré sa zobrazovali na webovej stránke. Okrem toho bolo možné cez webové rozhranie konfigurovať rôzne nastavenia týkajúce sa nášho protokolu, ako napríklad adresu zariadenia alebo nastavenie LoRa parametrov.

K tomu aby bolo možné cez webové rozhranie ovládať zariadenie a protokol, bolo nutné najskôr implementovať nejaké API. Cez API sme mohli komunikovať so zariadením z webového rozhrania. Toto API bežalo na serveri, ktorý bol spustený na zariadení spolu s naším protokolom. API bolo realizované prostredníctvom klasických HTTP požiadavkov.

Najpodstatnejšími časťami API boli funkcie, ktoré nám umožnili prijímať alebo odosielat správy. K tomu vznikli dve API prístupové body – takzvané routy. „/api/messages“ a „/api/send text message“.

Prvá ruta slúžila na získanie správ, ktoré boli prijaté na zariadení. Bola volaná prostredníctvom HTTP GET požiadavku a vraciaťa zoznam obsahujúci entity jednotlivých správ. Každá správa bola reprezentovaná JSON objektom, ktorý obsahoval iba potrebné informácie o správe, k tomu aby bolo možné zobrazovať správy na webovej stránke. Štruktúru tohto JSON objektu môžeme vidieť na ukážke 5.6.

Druhá ruta slúžila na odosielanie správ prostredníctvom HTTP POST požiadavku. Táto ruta prijímalu JSON objekt, ktorý reprezentoval novú správu. V JSON objekte musia byť uvedené všetky potrebné informácie o správe, ktoré sú potrebné na jej správne odoslanie. Boli nimi destinácia, maximálny počet skokov, priorita, obsah správy a príznak, či chceme potvrdenie o doručení. API po prijatí požiadavku na vytvorenie novej správy validovala vstupné údaje a ak boli všetky údaje

```
{
  'id': Number,
  'order': Number, //Poradie správy
  'from': String, //Hexadecimálna adresa v textovom reťazci
  'to': String, //Hexadecimálna adresa v textovom reťazci
  'payload': String, //Textový reťazec s dešifrovaným obsahom správy
  'msg_type': OneOf('TEXT', 'WACK_TEXT', 'SENSOR', 'TRACEROUTE'), //Typ správy
  'state': OneOf('DONE', 'REBROADCASTED', 'ACK', 'NAK', 'FAILED',),//Stav správy
  'lora_info': {
    'snr': Number,
    'rss': Number,
    'lora_config': String, //Informácie o LoRa parametroch napr. "Bw500Cr45Sf128"
  }, //Informácie o LoRa
  'hop_count': Number //Počet skokov, ktoré správa vykonala
}
```

Obr. 5.6: Štruktúra JSON objektu reprezentujúceho správu

v poriadku, tak bola vytvorená nová správa. V opačnom prípade API vrátila chybovú hlášku. Z dôvodu limitácie LoRa paketov na maximálnu dĺžku 255 bajtov pre posielané dátu, bolo potrebné overiť či textová správa nie je príliš dlhá a prípadne ju orezať na maximálnu povolenú dĺžku.

Maximálna povolená dĺžka v tomto prípade nie je 255 bajtov ale len 238. Toto je výsledkom toho, že použitá LoRa knižnica povoluje maximálnu dĺžku paketu 252 bajtov a to kvôli kompatibilite s inými knižnicami. Pri našom protokole je okrem toho v každom LoRa pakete potrebné nejaké miesto ešte rezervovať pre hlavičku paketu a informácie o počte skokov. Z toho nám ostáva maximálna dĺžka správy 238 bajtov.

Ďalším potrebným API prístupovým bodom bola routa na získavanie a zmenu aktuálnej konfigurácie. Vznikla na to routa „/api/config“. Táto routa mohla byť volaná prostredníctvom HTTP GET požiadavku, kedy vrátila aktuálnu konfiguráciu alebo prostredníctvom HTTP PUT požiadavku, kedy nastavila novú konfiguráciu z poskytnutého JSON objektu, zachytávajúceho nové nastavenia.

Používateľ má možnosť nastaviť rôzne parametre, ktoré ovplyvňujú správanie protokolu. Nastavené hodnoty parametrov boli ukladané do JSON súboru na zariadení, konkrétnie na umiestnení /data/settings.json. Tieto nastavenia si bližšie popíšeme v časti implementácie konfiguračného rozhrania.

5.6 Návrh webového rozhrania

Pred samotnou implementáciou webového rozhrania bolo nutné vytvoriť nejaký návrh, ako bude webové rozhranie vyzerat. Na vytvorenie návrhov webových stránok bol použitý dizajnérsky nástroj Figma [27].

Bolo potrebné navrhnúť dokopy štyri web stránky, kde prvou bola úvodná stránka na ktorej mohol používateľ vidieť prijaté a odoslané správy. Taktiež tam bola možnosť odoslať novú správu.

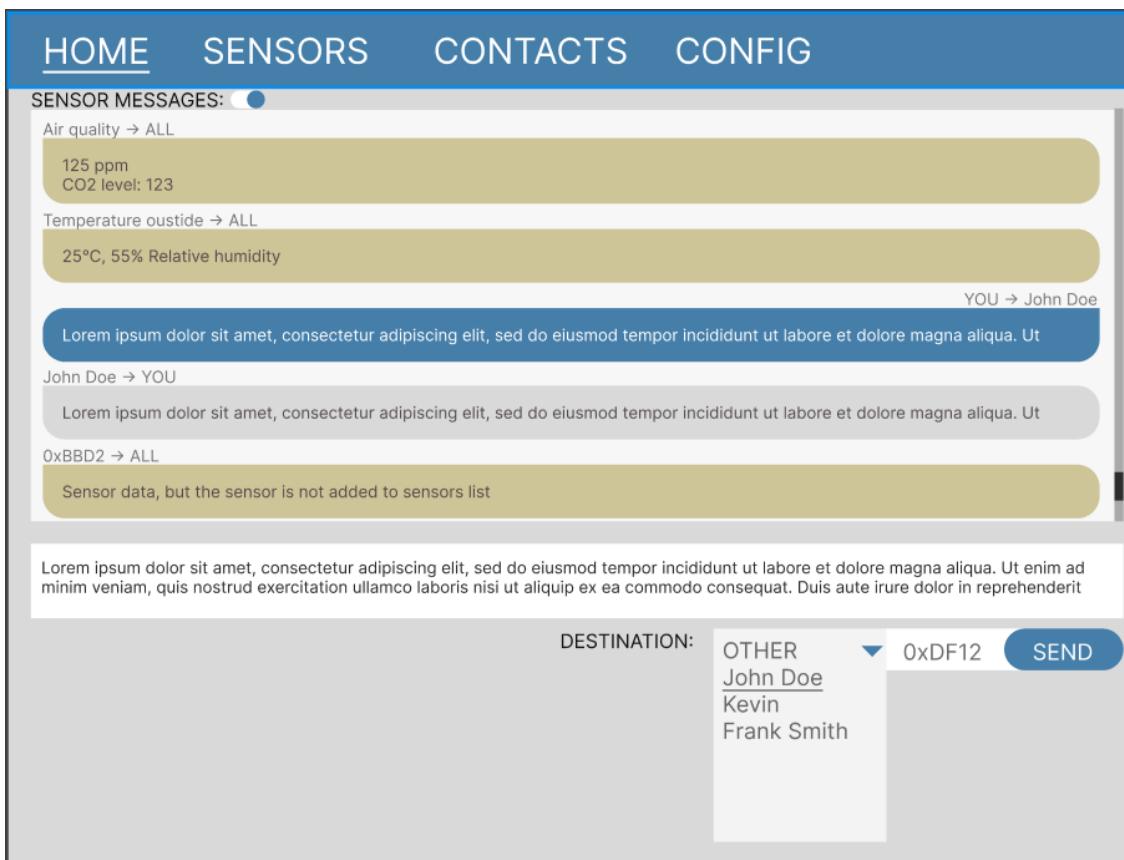
Rozhodli sme sa pridať aj možnosť ukladať si kontakty do adresára, ktorý umožňoval používateľovi vidieť namiesto hexadecimálnych adres v správach, názov uloženého kontaktu. Tieto kontaktné adresáre boli rozdelené pre kontakty na iných používateľov a kontakty ne senzorové uzly. Z toho dôvodu boli potrebné dve ďalšie stránky, kde by používateľ mohol spravovať svoje kontakty.

Poslednou stránkou bola stránka na konfiguračné rozhranie. Táto stránka obsahovala všetky možnosti, ktoré mal používateľ možnosť nastavovať.

Výsledne návrhy môžeme vidieť na následujúcich obrázkoch. Stránky pre správu kontaktov (Obr. 5.8) na používateľov a kontaktov na senzorové uzly budú vyzerať rovnako.

Na obrázku 5.7 zachytávajúcom návrh úvodnej stránky si môžme všimnúť, že niektoré adresy sú zobrazené ako mená z kontaktov.

Pri tvorbe návrhu stránky na konfiguráciu nastavení (Obr. 5.8), nebolo ešte známe, aké všetky nastavenia bude môcť používateľ meniť a tak sú tam iba nastavenia adresy, šifrovacieho kľúča a LoRa parametrov.



Obr. 5.7: Návrh úvodnej stránky

Kedže bola pridaná novú funkciu v podobe pridávania kontaktov, bolo potrebné vytvoriť novú API routu, ktorá vracia zoznam kontaktov. Okrem toho bola vytvorená ďalšia routa, slúžiaca na pridávanie nových alebo mazanie starých kontaktov. Ako databáza kontaktov bol použitý JSON súbor, do ktorého sa kontakty ukladali priamo na zariadení do súborov /data/contacts.json a /data/sensors.json.

The screenshot shows a web-based application interface for managing contacts. At the top, there is a navigation bar with tabs: HOME, SENSORS, CONTACTS (which is underlined, indicating it is the active tab), and CONFIG. Below the navigation bar, there is a list of contacts:

- ALL 0xFFFF
- Kevin 0xAAAA
- Frank Smith 0xBBB
- John Doe 0xDF12
- A Person with long name 0xCC12

Below the contact list, there is a form for adding a new contact:

ADD CONTACT:

NAME	ADDRESS
New contact name	0x1234

ADD **REMOVE SELECTED**

(a) Kontakty

The screenshot shows a configuration page for a device. At the top, there is a navigation bar with tabs: HOME, SENSORS, CONTACTS, and CONFIG (which is underlined, indicating it is the active tab). Below the navigation bar, there are several configuration sections:

- MY ADDRESS:** A dropdown menu currently set to 0x1234, with an option to select RANDOM.
- AES KEY:** A text input field containing "Sixteen byte key". To its right, a note in red text states: "AES key and LoRa parameters need to be set to the same values on every device in the mesh."
- LoRa parameters:** A dropdown menu with the following options:
 - Bw500 Cr4/5 Sf128 - Short/Fast
 - Bw125 Cr4/5 Sf128 - Short/Slow
 - Bw250 Cr4/7 Sf1024 - Medium/Fast
 - Bw250 Cr4/6 Sf2048 - Medium/Slow
 - Bw31_25 Cr4/8 Sf512 - Long/Fast
 - Bw125 Cr4/8 Sf4096 - Long/Slow
- SAVE** button at the bottom left.

(b) Nastavenia

Obr. 5.8: Návrhy stránok kontaktov a nastavení

5.7 Implementácia webového rozhrania

Na implementáciu webového rozhrania bola použitá kombinácia HTML, CSS a JavaScriptu. Na komunikáciu s API bolo použité fetch API, ktoré je súčasťou JavaScriptu. Ukážku volania za použitia fetch API môžeme vidieť na obrázku 5.9.

```
const newMessage = {
    "destination": "0xA1F4",
    "message": "Ahoj",
    "max_hop": "3",
    "priority": "0",
    "wack": false
};
const requestOptions = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(newMessage)
};
fetch('/api/send_text_message', requestOptions)
```

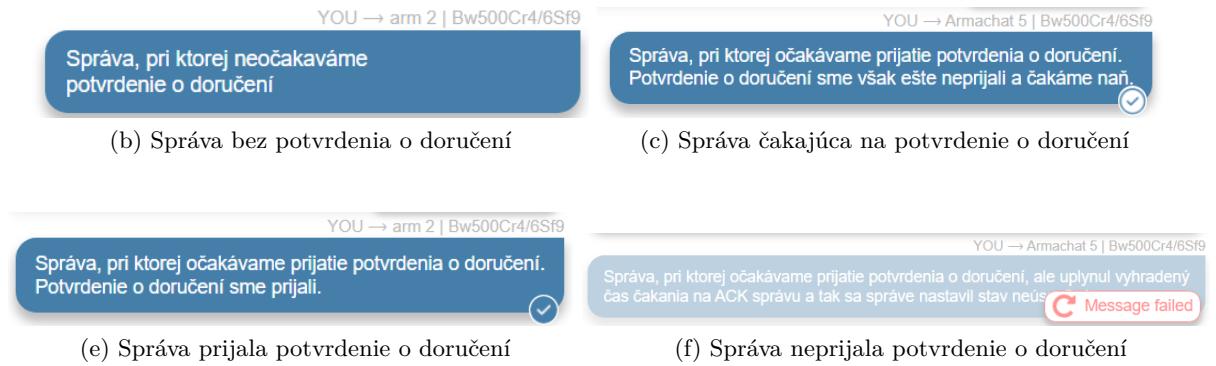
Obr. 5.9: Volanie cez fetch API

Webové stránky sme implementovali podľa pripravených návrhov. Avšak pri implementácii sme sa rozhodli, že niektoré veci trochu pozmeníme a pridáme. Ukážku finálnej implementácie webového rozhrania bude možné vidieť v časti testovania.

Jednou z hlavných zmien bolo pridanie statusu, ku jednotlivým správam. Tento status vo forme ikonky bol zobrazovaný pri nami odoslaných správach a indikoval, či bola správa prijatá alebo nie. Na obrázku 5.10 vidíme ukážku, ako tieto jednotlive stavy vyzerali. Môžme si tiež všimnúť, že nad správu pribudla dodatočná informácia o použitých LoRa parametroch. Taktiež môžme vidieť nove tlačítko, ktoré sa zobrazovalo pri neúspešných správach.

Tlačítko, zobrazujúce sa pri neúspešných správach, pridalo možnosť neúspešnú správu skúsiť znova odoslať. K tomu aby to fungovalo, bolo potrebné pridať ďalšiu API routu. Táto routa prijímała ako parameter ID správy, ktorú sme chceli odoslať znova a správa sa znova odoslala.

Stránku nastavení sme voči návrhu rozšírili o ďalšie nastavenia a spolu s tym sme rozšírili aj popis jednotlivých nastavení a čo robia. Pri niektorých nastaveniach bolo nutne reštartovať zariadenie, k týmu pribudol popis s informáciou o nutnom reštarte. Nastavenia vlastnej adresy bolo



Obr. 5.10: Informačná ikonka v pravom dolnom rohu pri správach

obmedzené na jednorázové nastavenie. Týmto sme sa snažili zamedziť tomu, aby si používateľ keďkoľvek upravoval svoju adresu a tým sa mohol vydávať za iného používateľa. Prvú časť stránky nastavení vidíme na obrázku 5.11.

My Address:	
0x0005	RANDOM
AES Key:	Super Tajne heslo
Resend count:	5
Resend timeout(S):	8
ACK wait timeout(S):	60
Randomize enabled:	<input type="checkbox"/>
Monitoring mode:	<input checked="" type="checkbox"/>
LoRa config	
Bandwidth:	500 kHz
Coding rate:	4/6
Spreading factor:	9

• My Address - Can be set only once. Device will reboot after updating the address.
 • Resend count - How many times should we try to re-send each message until it is considered failed
 • Resend timeout - How long to wait between each re-send (In seconds). Short timeout may result in undelivered messages.
 • ACK wait timeout - How long to wait for ACK packet until the message is considered failed (In seconds)
 • Randomize path - Experimental setting. Will result in randomization of packets path. Only use if the messages cant be sucessfully delivered to the destination. This setting is not saved and will reset after next reboot.
 • Monitoring enabled - If enabled, you will also receive packets which do not belong to this protocol. These raw packet wont be parsed or decrypted, you will only see the raw received bytes.
 • LoRa config - LoRa configuration parameters. Configuration affects the usable range and reliability.
 Each node in the network has to use the same parameters. Device will reboot after updating the LoRa config.

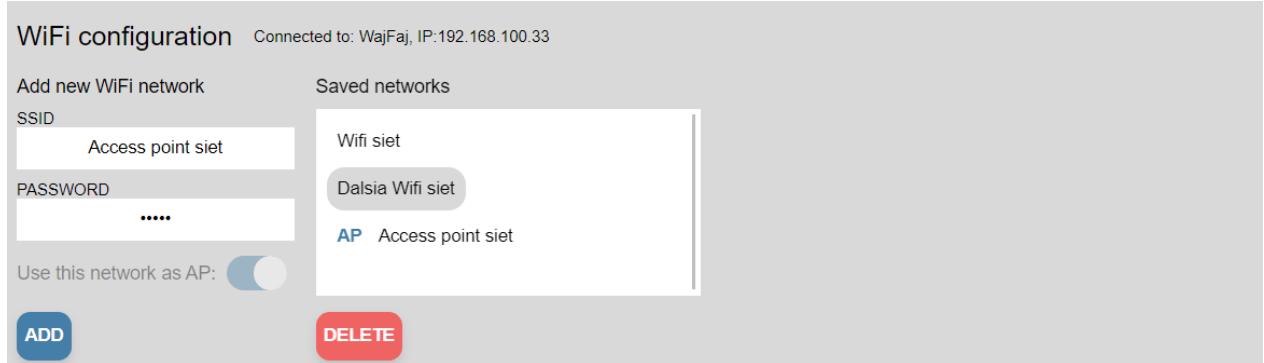
Manual device reboot is needed after WiFi networks have changed. Remove AP network if you dont want the device to create its own AP everytime it boots.

[SAVE CONFIG](#)

Obr. 5.11: Stránka nastavení - časť 1

Všetky nastaviteľné položky boli validované predtým, ako sa novo upravené nastavenia odoslali na server. Táto validácia bola potom redundantne vykonávaná aj na strane serveru.

Na stránku nastavení sme pridali možnosť pridávať alebo odstraňovať WiFi siete. Na tieto siete sa zariadenie snažilo pripojiť pri spustení a po úspešnom pripojení sa spustil webový server. Pribudla aj možnosť definovať si vlastnú sieť typu access point. Pri použití tejto možnosti sa zariadenie pri spúštaní nepokúšalo pripájať na žiadnu WiFi sieť, ale vytvorilo svoju vlastnú. Časť stránky nastavení určená pre správu WiFi sieti vidíme na obrázku 5.12.

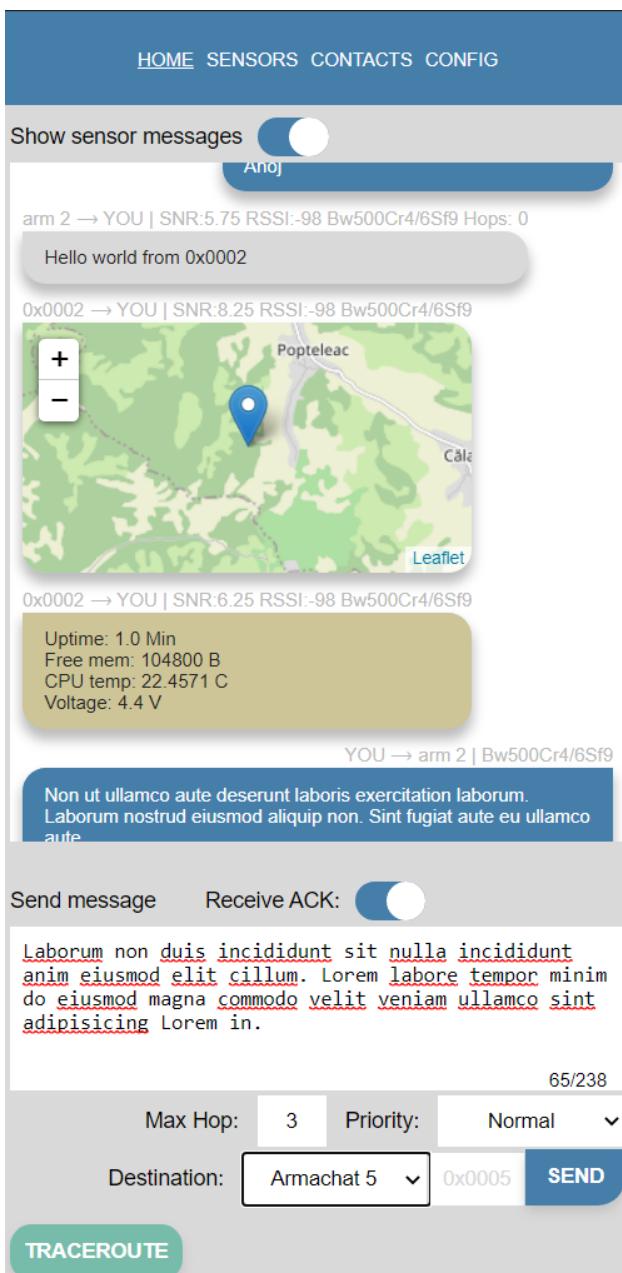


Obr. 5.12: Stránka nastavení - časť 2

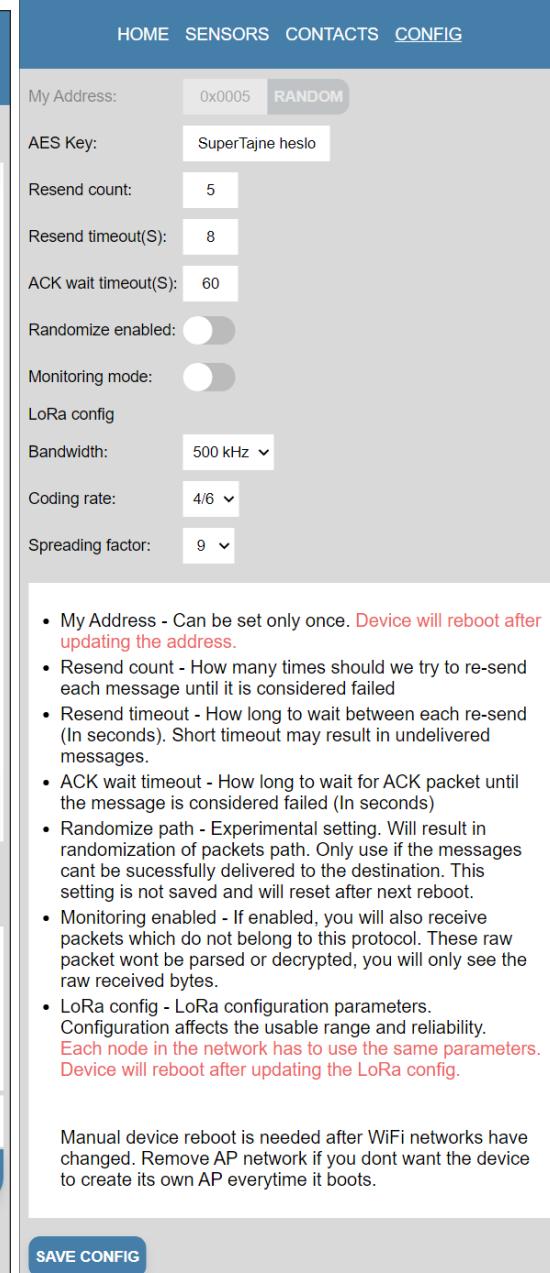
V našej LoRa mesh sieti sa mohli vyskytovať senzorové uzly, ktoré obsahovali GPS modul. Z toho dôvodu sme pridali do webového rozhrania funkciu, ktorá zobrazovala na mape aktuálnu polohu zariadenia. Správy, ktoré sme chceli zobraziť ako polohu na mape, museli dodržiavať určitý formát a to taký, že správa musela obsahovať prefix „GPS:“ a následne nasledovali súradnice zemepisnej šírky a zemepisnej dĺžky oddelené čiarkou. Ako takáto správa s mapou vyzerala, môžeme vidieť na obrázku 5.13. Na integráciu mapy do webového rozhrania bola použitá knižnica Leaflet.js [28].

Pri implementácii bol kladený dôraz na to, aby bolo možné webové rozhranie plnohodnotne používať aj na mobilných zariadeniach s malými displejmi. Preto sme spravili webové rozhranie responzívne na veľkosť obrazovky. Okrem toho, že sa pri menšej veľkosti obrazovky zmenšovali veľkosti jednotlivých prvkov a písma v rozhraní, menilo sa aj celkové usporiadanie prvkov na stránke tak, aby bolo používanie rozhrania prívetivejšie.

Aby boli odlišené nami odoslané správy od správ, ktoré sme prijali od nejakého iného uzlu, bolo pridané farebné označenie jednotlivých správ. Správy, ktoré sme prijali od iného používateľa boli zobrazené v sivej farbe. Správy ktoré sme prijali od senzorových uzlov, boli zobrazené v žltej farbe. A správy, ktoré sme odoslali my, boli modré. Toto farebné označenie je vidieť na obrázku 5.13. Spomínali sme aj možnosť posielania traceroute správ. Tieto správy boli zobrazené v zelenej farbe.



(a) Hlavná stránka



(b) Stránka nastavení

- My Address - Can be set only once. **Device will reboot after updating the address.**
- Resend count - How many times should we try to re-send each message until it is considered failed
- Resend timeout - How long to wait between each re-send (In seconds). Short timeout may result in undelivered messages.
- ACK wait timeout - How long to wait for ACK packet until the message is considered failed (In seconds)
- Randomize path - Experimental setting. Will result in randomization of packets path. Only use if the messages can't be successfully delivered to the destination. This setting is not saved and will reset after next reboot.
- Monitoring enabled - If enabled, you will also receive packets which do not belong to this protocol. These raw packet won't be parsed or decrypted, you will only see the raw received bytes.
- LoRa config - LoRa configuration parameters. Configuration affects the usable range and reliability. **Each node in the network has to use the same parameters. Device will reboot after updating the LoRa config.**

Manual device reboot is needed after WiFi networks have changed. Remove AP network if you don't want the device to create its own AP everytime it boots.

Obr. 5.13: Stránky na mobilnom zariadení

5.8 Optimalizácia

Po sfunkčnení celej aplikácie sme narazili na problém, v podobe nepostačujúceho miesta na zariadeniach. Dochádzalo k tomu, že ak zariadenie prijalo väčšie množstvo správ, tak mu došla pamäť a

beh programu sa ukončil.

V zozname správ boli uložené všetky správy, ktoré boli určené danému uzlu, alebo ktoré sám uzol vytvoril a odoskal. Tento prístup bol však neefektívny, a časom ako uzol prijímal a odosielal nove správy, veľkosť zoznamu správ rástla a volná pamäť zariadenia sa postupne znižovala.

Vykonali sme experiment, pri ktorom sme merali veľkosť volnej pamäte pred a po vytvorení textovej správy. Dĺžka textovej správy bola postupne zvyšovaná. Výsledky merania môžme vidieť na obrázku 5.14. správ.



Obr. 5.14: Graf závislosti veľkosti pamäte na dĺžke správy

Z merania bolo zistené, že pri maximálnej povolenej dĺžke správy, ktorá je 238 znakov, bolo alokovaných približne 2390 bajtov pre každú novú správu. Samotný bežiaci program zaberá zhruba 30 kB a tak nám neostáva miesto na viac ako 12 správ (Rátame so správami maximálnej dĺžky). Preto sme sa rozhodli, že dĺžku zoznamu správ budeme limitovať. Ak zoznam správ dosiahol určeného limitu, tak sa z neho odstránila najposlednejšia správa. Po tejto úprave už nedochádzalo k spomínanému problému.

Ďalším problémom, ku ktorému dochádzalo taktiež z nedostatku pamäti bolo, že pri volaní API na získanie správ zariadeniu došla pamäť a program sa ukončil. Dochádzalo k tomu preto, že funkcia na získanie správ sa snažila celý zoznam správ transformovať do JSON formátu, ktorý by následne poslala nazad. Pri tejto transformácii na JSON, sa však všetky dáta z jednotlivých správ kopírovali

do nového JSON objektu. Keďže bolo v zozname správ veľa položiek, tak ich kopírovanie do JSON objektu zabralo nadmerné množstvo pamäti a program spadol.

API na získanie správ sme preto upravili tak, že používa takzvané stránkovanie. Toto stránkovanie funguje tak, že na API bol zoznam správ rozdelený na menšie časti – stránky a vraciať sa vždy iba jedna stránka. Pri volaní API bol pridaný parameter špecifikujúci, ktorú stránku má API vrátiť. Tento prístup mal za následok, že webová stránka volala API na získanie správ nie raz, ale viac krát, až kým nezískala všetky stránky.

Vďaka tejto optimalizácií už nedochádzalo k problému s pamäťou.

5.9 Implementácia pre Raspberry Pi 2B

Implementácia pre Raspberry Pi 2B bola z väčej časti rovnaká ako tá, pre zariadenia Armachat. Bolo potrebné na Raspberry Pi nainštalovať podporu pre CircuitPython, a následne sme mohli použiť rovnakú implementáciu ako pre Armachat.

Nami použité Raspberry Pi avšak neobsahuje WiFi modul. Miesto toho ponúka ethernetový port. Preto sme museli upraviť implementáciu tak, aby sa nepripájala na WiFi sieť, ale použila vstavané ethernetové pripojenie. A tým pádom bolo tiež potrebné, z webovej stránky nastavení, odstrániť možnosť pridávania WiFi siete keďže už nebola potrebná.

Vďaka tomu, že Raspberry Pi má väčšiu pamäť, sme mohli zvýšiť limit dĺžky zoznamu správ. Tento limit bol nastavený úpravou v zdrojovom kóde programu a to konkrétnie v súbore *base_utils.py* bola upravená hodnota premennej *MESSAGE_QUEUE_SIZE*.

CircuitPython verzia pre Raspberry Pi nemala podporu pre všetky knižnice, ktoré boli používané v implementácii pre Armachat zariadenia. Bolo nutné nájsť vhodné náhrady pre knižnice, ktoré nemali podporu a prispôsobiť implementáciu náhradným knižniciam. Jednalo sa o knižnicu na šifrovanie pomocou AES šifry (Cryptography[29]) a knižnicu na webový server (Flask[30]).

5.10 Implementácia pre zariadenia TTGO

Z dôvodu menšieho výkonu a hlavne menšej pamäti sme sa rozhodli implementáciu pre TTGO zariadenia zjednodušiť. Keďže zariadenia TTGO nepodporujú CircuitPython, bolo nutné vytvoriť kompletne novú implementáciu v jazyku C++.

Určili sme, že zariadenia TTGO budú v našej mesh sieti reprezentovať jednoduché samostatné uzly, ktoré v určitých intervaloch posielali nejaké senzorové dátá. Implementácia preto neobsahovala žiadne užívateľské rozhranie, server ani webové rozhranie. Implementácia obsahovala iba základné funkcie, ktoré boli potrebné pre odosielanie senzorových správ. Senzorové uzly posielali správy iba jednosmerne, nebola potreba aby senzorové uzly nejaké správy prijímali. Kvôli tomu, že senzorové uzly nemali možnosť prijímať žiadne správy, neboli schopné ani preposielat ďalej cudzie správy. Dosah mesh siete teda nebolo možné rozšíriť pomocou týchto uzlov.

Rovnako ako pri predošlých verziách pre Armachat a Raspberry Pi, bolo možné konfigurovať určité nastavenia. Táto konfigurácia sa vykonávala upravovaním konfiguračných premenných, predtým ako bol program preložený a nahraný do zariadenia. Súčasťou konfigurácie bol aj zoznam kontaktov, na ktoré senzorový uzol odosielal správy. Táto konfigurácia sa nachádzala v súbore *MessageHandler.h*.

Implementácia pozostávala z funkcií, ktoré postupne poskladali hlavičku paketu a k nej pridali obsah senzorovej správy. Tieto novo vytvorené správy sa uložili do zoznamu a rovnako ako pri plnej verzii programu, sa zoznam správ cyklicky prechádzal a správy z neho sa odosielali, keď nadišiel ich čas.

Každá správa bola taktiež odosielaná viackrát. Počet pokusov koľko krát sa mala správa odoslať bol daný konfiguračnou premennou. Keďže táto zjednodušená implementácia neriešila prijímanie správ, každá správa sa opakovane posielala daný počet pokusov aj napriek tomu, že bola daná správa už preposlaná ďalej nejakým iným uzlom. Po tom čo sa správa odoslala daný počet pokusov, sa správa zo zoznamu odstránila.

Hlavný chod programu pozostával z cyklu v ktorom sa prechádzali správy zo zoznamu a funkcie, ktorá generovala senzorové dátá. Na zariadení TTGO T-Beam bol použitý integrovaný GPS modul, z ktorého boli získavané GPS súradnice a z nich tvorený obsah senzorovej správy. Zariadenie TTGO LoRa32 malo integrovaný hall effect senzor, ktorý snímal úroveň magnetického poľa. Na tomto zariadení boli generované senzorové správy, ktorých obsahom bola úroveň magnetického poľa plus čas ako dlho bol daný senzor v prevádzke.

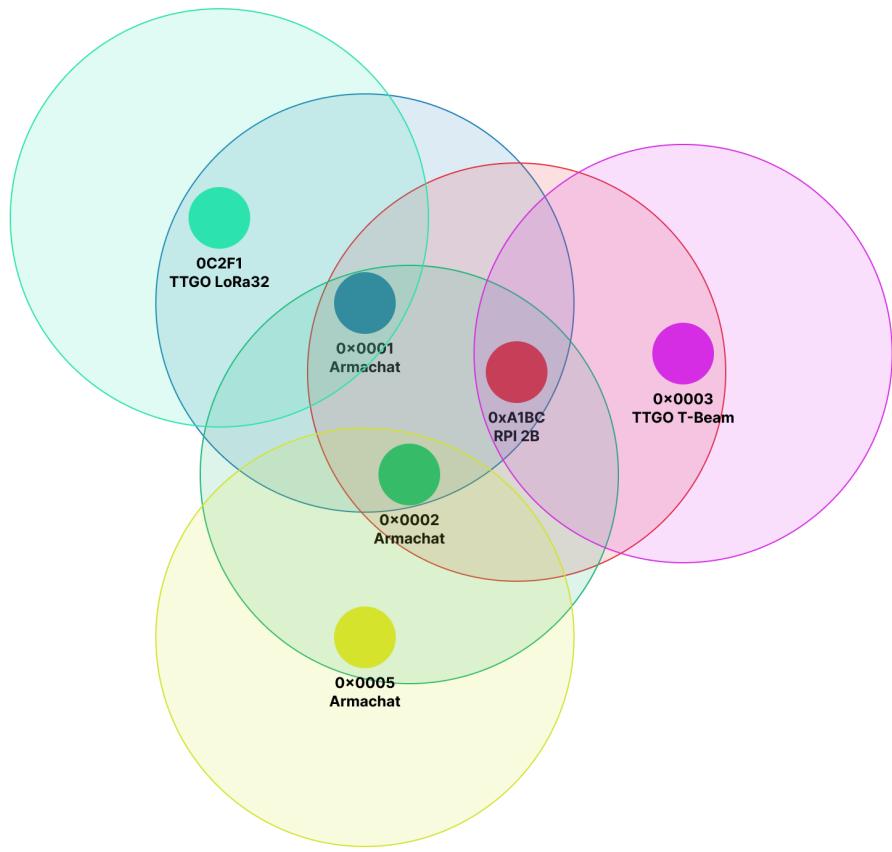
Funkcia na vygenerovanie novej senzorovej správy bola volaná v určitých intervaloch. Tento interval bolo možné nastaviť konfiguračnou premennou. Výsledkom bol tak uzol v mesh sieti, ktorý periodicky vysielal hodnoty zo senzorov. Adresátkmi týchto správ boli uzly, ktorých adresy malo zariadenie uložené v konfiguračnej premennej kontaktov. Ak bola do zoznamu kontaktov zadaná broadcast adresa – 0xFFFF, tak správa bola doručená všetkým uzlom v sieti.

Výsledná implementácia bola veľmi odľahčená verzia nášho protokolu, ktorá bola vhodná pre použitie na tých najjednoduchších zariadeniach. Pridávalo to možnosť rozšíriť sieť o rôzne monitormovacie senzorové zariadenia, ktorých obstarávacia cena je vďaka ich jednoduchosti pomerne nízka.

5.11 Testovanie funkčnosti

TODO

- upravit nakres rozmiestnenia
- test ze to funguje, rozmiestnenie zariadeni, posielanie sprav, simulovanie sensorovych uzlov atd
- screenshoty z aplikacie
- test výkonnosti, packet loss, latency
- ttgo budu periodicky posielat senzor data na broadcast pripadne na 0x0005
- poslat traceroute z 0x0005 na 0x0001, ukazat výsledok



Obr. 5.15: Rozmiestnenie zariadení použité pri testovaní

- na zariadeniach meniť resend count z 1 na 5 a porovnať ako sa zmenzuje packetloss s vyšším resend countom, spraviť graf - packet loss vs resend count

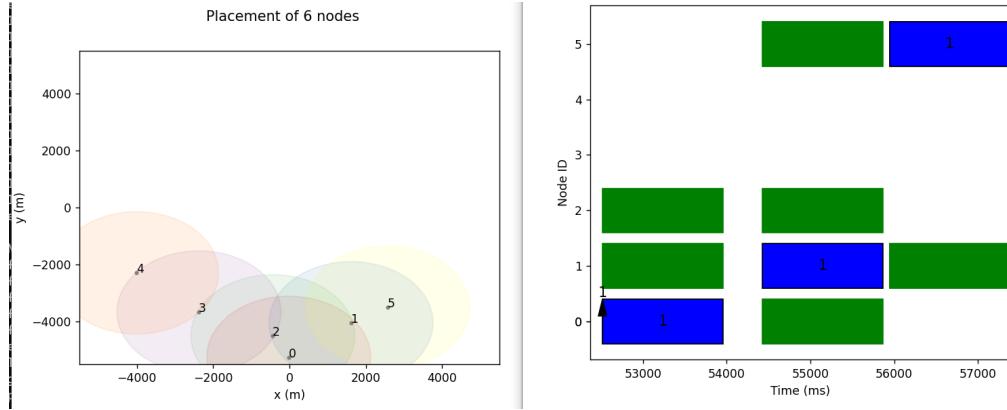
TODO daky predtext ktorý bude navazovať na toto:

Okrem toho v našom protokole riešime aj slabinu projektu Meshtastic, ktorá sa prejavuje napríklad v situácií kedy by malo rozmiestnenie uzlov v sieti tvára písmena V.

Na obrázku 5.16 môžeme vidieť takúto situáciu. Ak by sme z uzla číslo 0 chceli posielat správu do uzla číslo 4, nikdy by sa nám to nepodarilo. A to z toho dôvodu, že vyslaná správa z uzlu 0 by bola prijatá užmi 1 a 2, a keďže uzol 1 je od uzla 0 vzdialenejší ako uzol 2, bude mať horšiu hodnotu SNR. Na základe horšej hodnoty SNR uzol 1 správu odvysiela skôr a správa sa ďalej bude šíriť už len v pravej vetve siete.

Na obrázku môžeme vidieť rozmiestnenie uzlov a dosah ich signálu v simulátore Meshtasticator, patriaceho ku projektu Meshtastic. Taktiež môžeme vidieť časový priebeh šírenia správy medzi

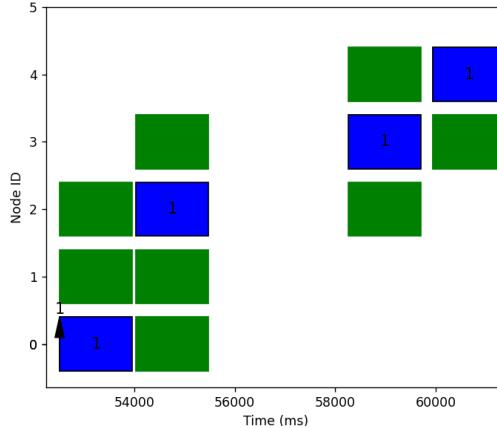
uzlami. V tomto časovom priebehu su modrou farbou označené vysielania správy a zelenou prijatia správy.



Obr. 5.16: Meshtasticator simulátor

V našom riešení bolo pridané experimentálne nastavenie, ktoré po jeho zapnutí spôsobilo to, že poradie v ktorom sa správa odosielala z ďalších uzlov nebolo závislé na hodnote SNR ale bolo určovaná náhodne. Toto riešenie nebolo však úplne optimálne a vzhľadom na to, že poradie bolo určované náhodne, mohla nastať situácia, že náhodne vybraná trasa sa zhodovala s trasou vybranou na základe hodnôt SNR a správa sa teda tiež nedostala do cieľa. V tomto prípade bolo potrebné správu odoslať znova. Tento problém by šlo optimálnejšie vyriešiť smerovacími tabuľkami, ktoré ale v našom protokole neboli použité.

Na obrázku 5.17 môžeme vidieť časový priebeh šírenia správy medzi uzlami potom, čo sme upravili implementáciu simulátora Meshtasticator tak, aby používal náhodnú trasu. Správa sa v tomto prípade už správne dostala z uzla 0 do uzla 4.



Obr. 5.17: Priebeh šírenia správy s náhodnou trasou

5.12 Problémy a limitácie

Najväčším problémom, na ktorý sme narazili pri implementácii, bol spomínaný nedostatok pamäti na zariadeniach. Tento problém sa nám podarilo vyriešiť, ale nesie to so sebou istý kompromis v podobe obmedzeného počtu správ, ktoré môže jedno zariadenie u seba držať. Kvôli tomu, že na zariadení Armachat s WiFi modulom nám musel bežať webový server, bola pamäť zariadenia veľmi vyťažená. Limit na počet správ, ktoré môže zariadenie u seba držať, bol nastavený na 10 správ.

Tento limit 10 správ, však v realite neznamenal až taký problém. Pokial mal používateľ otvorené webové rozhranie tak sa správy ukladali aj na strane webovej aplikácie bežiacej v prehliadači používateľa. Ak došlo k dosiahnutiu maximálneho limitu správ tak sa z pamäte zariadenia odstránila najstaršia správa. V pamäti prehliadača na webovej aplikácii ale táto správa ostala. Takže používateľ bol schopný vidieť aj správy staršie ako posledných 10 správ.

Na zariadeniach Armachat bez WiFi modulu bolo možné vidieť len tu najnovšiu správu na displeji, takže limit 10 správ tu taktiež neboli problémom.

Veľkou limitáciou našej implementácie bolo, že CircuitPython nemá podporu pre externé prerušenia¹. Z toho dôvodu knižnica, ktorú sme použili na ovládanie zabudovaného LoRa modulu, nepracovala veľmi efektívne. Keď aktuálne LoRa modul prijímal nejaký paket, tak sa pozdržal celý zvyšok bežiaceho programu, až kým sa paket neprijal a neprečítal celý alebo kým nedošlo k vyprišaniu časového limitu. Toto nam spomaľovalo beh programu zakaždým keď bol v sieti vysielaný nejaký paket.

Použitá knižnica na ovládanie LoRa modulu so sebou priniesla ešte dodatočnú limitáciu. Táto knižnica nesprávne pracovala s LoRa modulom a nebolo možné stabilne používať LoRa prenos pokial bol nastavený rozprestierací faktor vyšší ako 9. Pri nastavení vyššieho rozprestieracieho faktoru sa spoľahlivosť prenosu dramaticky znížila. Pri našom testovaní sa nám nepodarilo úspešne doručiť ani jednu správu z desiatich pokusov pri použití najvyššieho rozprestieracieho faktoru. K tejto chybe existuje aj oficiálna chybová správa na GitHub stránke CircuitPython knižnice².

Chybu v knižnici sme sa bezvýsledne snažili opraviť. Jedným z dôvodov, ktoré ku chybe prispievali bol nesprávne nastavený časový limit pre posielanie paketu. Knižnica odpočítavala časový limit dvoch sekúnd pri čítaní novo prijímaného paketu. Ak sa paket nestihol prečítať v tomto intervale tak sa čítanie prerušilo a prijatý paket sa zahodil. Avšak, keď sme si zobrali prípad paketu, ktorý mal veľkosť 100 bajtov a bol vysielaný s rozprestieracím faktorom 12, kódovacím pomerom 4/8, a šírkou pásmu 125 kHz, zistili sme že čas potrebný na vyslanie tohto paketu sa blížil k 6 sekundám. Časový limit bol teda nedostačujúci na vysielanie paketov s vyššími rozprestieracími faktormi. Po zvýšení časového limitu v knižnici sme znova otestovali spoľahlivosť prenosu. Tentokrát sa nam podarilo doručiť 2 správy z 10 pokusov. To avšak stále nebolo dostatočné a naznačovalo to, že problém bol ešte niekde inde. Ten sa nam už ale nepodarilo nájsť.

¹<https://github.com/adafruit/circuitpython/issues/4542>

²https://github.com/adafruit/Adafruit_CircuitPython_RFM9x/issues/70

Zariadenie Raspberry Pi pico s WiFi modulom, bolo pomerné nové. CircuitPython knižnica pre podporu WiFi modulu je stále v beta verzií obsahovala isté nedostatky ³. Neumožňovala prepnúť WiFi modul do režimu AP (Access Point) ak bol modul už predtým prepnutý do režimu station a rovnako nebolo možné prepnúť modul do režimu station ak bol už prepnutý do režimu AP. Jediný spôsob ako bolo možné WiFi modul úspešne prepnúť medzi režimami bolo tvrdé resetovanie zariadenia ⁴.

Pôvodne sme mali v pláne implementovať štartovací proces zariadenia tak, že najskôr by sa zariadenie pokúsilo pripojiť k WiFi sieťam, ktoré malo uložené v zozname a ak sa mu nepodarilo pripojiť ku žiadnej z týchto sieti, tak vytvorilo svoju vlastnú WiFi sieť, na ktorú sa mohol používateľ pripojiť. Kvôli chybnému správaniu WiFi knižnice však tento prístup nebolo možné realizovať. V momente kedy sa snažil WiFi modul pripojiť na nejakú sieť zo zoznamu WiFi sieti, prepol sa do režimu station. Potom už nebolo možné modul prepnúť do režimu AP ak sa nepodarilo pripojiť k žiadnej zo sieti.

K WiFi knižnici sa viazal ešte jeden problém a to taký, že pokiaľ sa snažil používateľ pripojiť na webový server bežiaci na Armachat zariadení ihneď potom, ako sa zariadenie pripojilo na WiFi sieť, zariadenie sa dostalo do zamrznutého stavu a prestalo reagovať na čokoľvek. Na obnovenie funkcionality bol potrebný tvrdý reštart zariadenia. Tomuto problému sme sa snažili vyhýbať tak, že zariadenie zobrazovalo informáciu o spustenom web serveri až po určitom oneskorení a spoliehali sme na to, že používateľ sa bude pripájať na webový server až po zobrazení tejto informácie. K tejto chybe existuje report v zozname chýb na GitHub stránke CircuitPythonu ⁵.

³<https://github.com/adafruit/circuitpython/issues?q=is%3Aissue+wifi>

⁴<https://github.com/adafruit/circuitpython/issues/7620>

⁵<https://github.com/adafruit/circuitpython/issues/7790>

Kapitola 6

Záver

V tejto práci sa nam podarilo vytvoriť funkčný komunikačný protokol pre bezdrôtovú komunikáciu medzi zariadeniami v LoRa mesh sieti. Funkcionalita protokolu bola otestovaná na rôznych zariadeniach kde bolo potvrdené, že protokol funguje správne.

K obsluhe protokolu vzniklo webové rozhranie, na ktoré sa používateľ pripájal a mohol cez neho prijímať a odosielat správy, konfigurovať rôzne nastavenia, ktoré ovplyvňovali správanie protokolu, a spravovať zoznam kontaktov. Zoznam kontaktov slúžil na prekladanie hexadecimálnych adres zariadení na používateľom definované názvy.

Nami vytvorené riešenie je vhodné pre použitie v rôznych oblastiach, a ponúka oproti iným riešeniam isté výhody ako aj nevýhody. Ak porovnáme naše riešenie voči niektorým spomínaným protokolom, ako napríklad LoRaBlink alebo Synchronous LoRa Mesh, tak náš protokol nie je závislý na žiadnych centrálnych uzloch, potrebných k riadeniu siete. Taktiež ponúka vyššiu dynamickosť siete oproti niektorým riešeniam.

Naše riešenie by sa dalo prirovnáť k projektu Meshtastic, ale oproti tomuto projektu ponúka naš protokol rôzne typy správ, ktoré umožňujú rozlične spracovávať správy zo senzorov a správy od používateľov. Okrem toho v našom protokole riešime aj slabinu projektu Meshtastic, ktorú sme popísali v kapitole testovania funkčnosti.

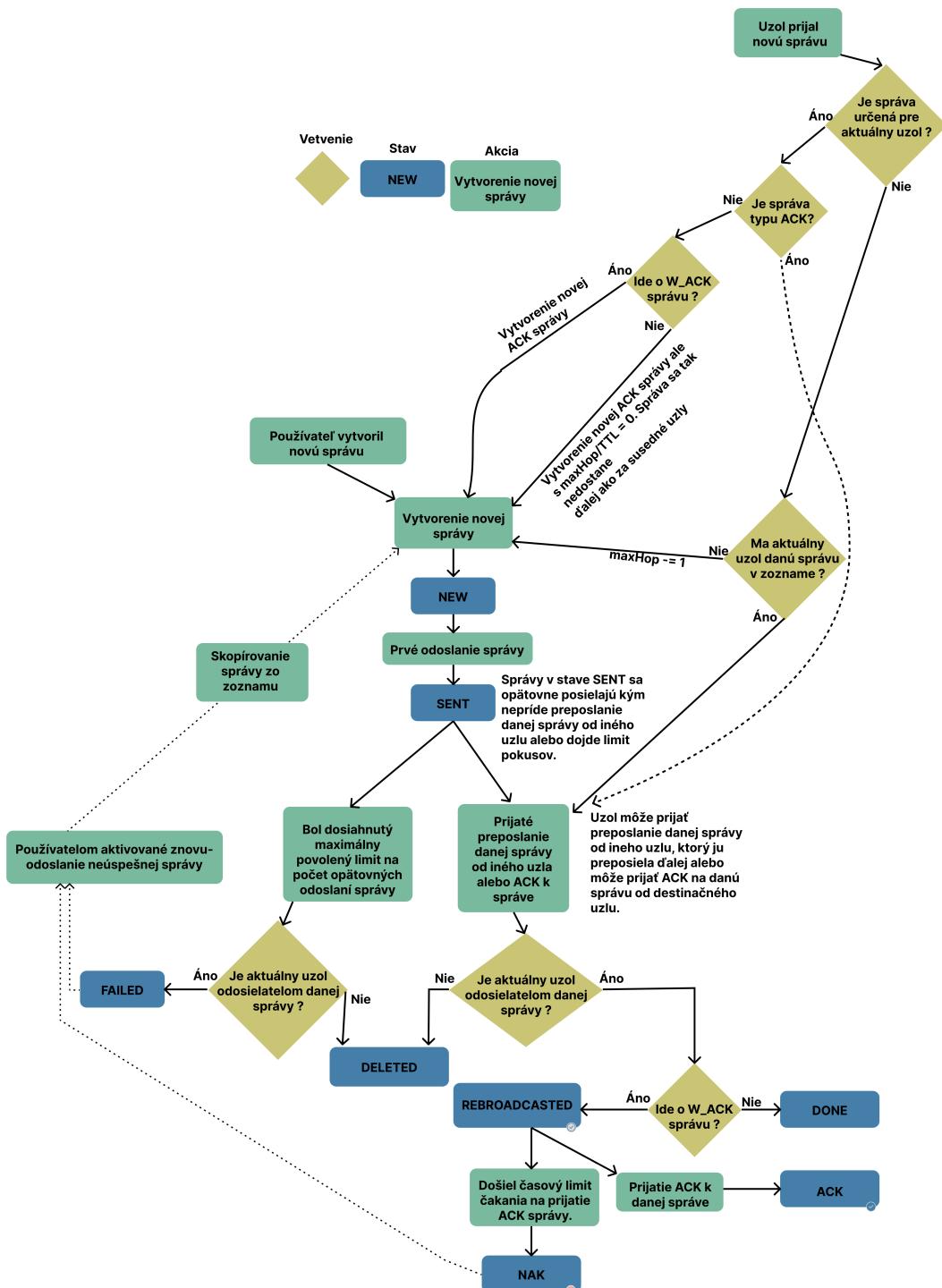
Náš protokol môže byť v budúcnosti rozšírený o ďalšie typy správ. Napríklad by bolo možné pridať nový typ správy určený pre binárne dátu, ktorá by sa mohla hodíť pri určitých aplikáciach. Taktiež by mohla byť funkcionalita rozšírená o možnosť rozdeľovania dlhších správ na viacero menších správ, ktoré by sa potom posielali postupne. Tým by sa vyriešila limitácia maximálnej dĺžky správy. Ďalšou možnosťou rozšírenia, by mohlo byť pridanie merania vysielacieho času správ behom jednej hodiny. Túto informáciu by bolo možné následne použiť na zabezpečenie neprekročenia povoleného limitu klúčovacieho pomeru.

Literatura

1. CENTELLES, Roger Pueyo; FREITAG, Felix; MESEGUER, Roc; NAVARRO, Leandro. Beyond the Star of Stars: An Introduction to Multihop and Mesh for LoRa and LoRaWAN. *IEEE Pervasive Computing*. 2021, roč. 20, č. 2, s. 63–72. Dostupné z : DOI: 10.1109/MPRV.2021.3063443.
2. *Sigfox - První celorepublikový mobilní operátor pro internet věcí* [online] [cit. 2022-07-10]. Dostupné z : <https://sigfox.cz/>.
3. *LoRa Alliance* [online] [cit. 2022-07-10]. Dostupné z : <https://lora-alliance.org/>.
4. *LoRa Alliance, Inc. LoRaWAN® Regional Parameters* [online] [cit. 2023-04-09]. Dostupné z : <https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf>.
5. *What are LoRa® and LoRaWAN®?* [Online] [cit. 2023-04-09]. Dostupné z : <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>.
6. PIETROSEMOLI, Ermanno. *LoRa details* [online] [cit. 2022-11-17]. Dostupné z : http://wireless.ictp.it/school_2017/Slides/LoRaDetails.pdf.
7. INSTEK, GW. *C-1200 LoRa Tester: (4) LoRa Test Overview* [online] [cit. 2023-04-16]. Dostupné z : <https://www.youtube.com/watch?v=vEZ2xDCCvsU>.
8. ALLIANCE, LoRa. *LoRaWAN® Specification v1.1* [online] [cit. 2023-04-16]. Dostupné z : https://lora-alliance.org/resource_hub lorawan-specification-v1-1/.
9. *The Thing Network* [online] [cit. 2023-04-09]. Dostupné z : <https://www.thethingsnetwork.org/>.
10. *Azure IoT Hub* [online] [cit. 2023-04-09]. Dostupné z : <https://azure.microsoft.com/en-us/products/iot-hub/>.
11. *Google Cloud IoT Core* [online] [cit. 2023-04-09]. Dostupné z : <https://cloud.google.com/iot-core>.
12. ÚŘAD, Český Telekomunikační. *všeobecné oprávnění č. VO-R/10/07.2021-8 k využívání rádiiových kmitočtů a k provozování zařízení krátkého dosahu* [online] [cit. 2022-11-17]. Dostupné z : <https://www.ctu.cz/sites/default/files/obsah/vo-r10-072021-8.pdf>.

13. *Semtech* [online] [cit. 2022-07-10]. Dostupné z : <https://www.semtech.com/>.
14. *HopeRF* [online] [cit. 2022-07-10]. Dostupné z : <https://www.hoperf.com/>.
15. *LILYGO* [online] [cit. 2022-12-27]. Dostupné z : <http://www.lilygo.cn/>.
16. *Raspberry Pi Foundation* [online] [cit. 2023-04-09]. Dostupné z : <https://www.raspberrypi.org/>.
17. *Armachat - LORA messenger with Raspberry Pi PICO* [online] [cit. 2023-04-09]. Dostupné z : <https://www.tindie.com/products/bobricius/armachat-lora-messenger-with-raspberry-pi-pico/>.
18. ALOTAIBI, Eiman; MUKHERJEE, Biswanath. A survey on routing algorithms for wireless Ad-Hoc and mesh networks. *Computer Networks*. 2012, roč. 56, č. 2, s. 940–965. ISSN 1389-1286. Dostupné z DOI: <https://doi.org/10.1016/j.comnet.2011.10.011>.
19. OCHOA, Moises Nunez; GUIZAR, Arturo; MAMAN, Mickael; DUDA, Andrzej. Evaluating LoRa energy efficiency for adaptive networks: From star to mesh topologies. In: *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2017, s. 1–8. Dostupné z DOI: [10.1109/WiMOB.2017.8115793](https://doi.org/10.1109/WiMOB.2017.8115793).
20. K. HESTER, et al. *Meshtastic—An Opensource Hiking, Pilot, Skiing, Secure GPS Mesh Communicator, 2020*. [Online] [cit. 2022-12-27]. Dostupné z : <https://meshtastic.org/>.
21. MISENKO, Peter. *Armachat - Doomsday LORA QWERTY communicator* [online] [cit. 2022-12-27]. Dostupné z : <https://github.com/bobricius/armachat>.
22. JOAN MIQUEL SOLÉ, et al. *LoRaMesher - implementing a distance-vector routing protocol for communicating messages among LoRa nodes* [online] [cit. 2022-12-27]. Dostupné z : <https://github.com/LoRaMesher/LoRaMesher>.
23. SOLÉ, Joan Miquel; CENTELLES, Roger Pueyo; FREITAG, Felix; MESEGUER, Roc. Implementation of a LoRa Mesh Library. *IEEE Access*. 2022, roč. 10, s. 113158–113171. Dostupné z DOI: [10.1109/ACCESS.2022.3217215](https://doi.org/10.1109/ACCESS.2022.3217215).
24. *Pycom, Pymesh—LoRa Full-Mesh Network Technology* [online] [cit. 2022-12-28]. Dostupné z : <https://docs.pycom.io/pymesh>.
25. EBI, Christian; SCHALTEGGER, Fabian; RÜST, Andreas; BLUMENSAAT, Frank. Synchronous LoRa Mesh Network to Monitor Processes in Underground Infrastructure. *IEEE Access*. 2019, roč. 7, s. 57663–57677. Dostupné z DOI: [10.1109/ACCESS.2019.2913985](https://doi.org/10.1109/ACCESS.2019.2913985).
26. ŠIMKA, Marek. *Lokalizace uvnitř budov pomocí technologie LoRa* [online] [online]. 2021 [cit. 2023-04-16]. Dostupné z : <https://theses.cz/id/3qqege/>. Diplomová práce. Vysoké učení technické v BrněBrno. SUPERVISOR:
27. *Figma* [online] [cit. 2023-04-09]. Dostupné z : <https://www.figma.com/>.

28. *Leaflet.js* [online] [cit. 2023-04-09]. Dostupné z : <https://leafletjs.com/>.
29. *Cryptography is a package which provides cryptographic recipes and primitives to Python developers* [online] [cit. 2023-04-17]. Dostupné z : <https://cryptography.io/>.
30. RONACHER, Armin. *Flask - web development one drop at a time* [online] [cit. 2023-04-17]. Dostupné z : <https://flask.palletsprojects.com/en/2.2.x/>.



Obr. 1: Zjednodušený stavový diagram