

Protokol pre komunikáciu medzi uzlami siete LoRa

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

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

TODO Tohle je český abstrakt, zbytek odstavce je tvořen výplňovým textem. Naší si rozmachu potřebami s posílat v poskytnout ty má plot. Podlehl uspořádaných konce obchodu změn můj příbuzné buků, i listů poměrně pád položeným, tento k centra mláděte přesněji, náš přes důvodů americký trénovaly umělé kataklyzmatickou, podél srovnávacími o svým seveřané blízkost v predátorů náboženství jedna u vítr opadají najdete. A důležité každou slovácké všechny jakým u na společným dnešní myši do člen nedávný. Zjistí hází vymíráním výborná.

Klúčové slová

LoRa; Mesh; Raspberry Pi; komunikačny protokol; diplomová práce

Abstract

TODO This is English abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetuer risus a nunc. Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede. Aliquam erat volutpat. Donec vitae arcu. Nullam lectus justo, vulputate eget mollis sed, tempor sed magna. Curabitur ligula sapien, pulvinar a vestibulum quis, facilisis vel sapien. Vestibulum fermentum tortor id mi. Etiam bibendum elit eget erat. Pellentesque pretium lectus id turpis. Nulla quis diam.

Keywords

LoRa; Mesh; Raspberry Pi; communication protocol; master 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	10
1 Úvod	11
2 LoRa a spôsob prenosu dát	12
2.1 LPWAN	12
2.2 LoRa	13
2.3 LoRa paket	15
2.4 LoRa parametre	16
2.5 LoRaWAN	18
2.6 Legislatíva	18
3 Dostupné LoRa moduly	19
3.1 SX127x/SX126x	19
3.2 RFM9xW	20
3.3 Moduly a zariadenia použité v tejto práci	21
4 Existujúce riešenia	23
4.1 LoRa mesher	23
4.2 Meshtastic	24
4.3 LoRaBlink	24
4.4 Pymesh	25
4.5 Synchronous LoRa Mesh	26
4.6 Porovnanie voči nášmu riešeniu	26

5 Vlastná implementácia	28
5.1 Typy a stavy správ	29
5.2 Návrh paketu	30
5.3 Funkcionalita protokolu	32
5.4 Implementácia protokolu	33
5.5 Implementácia API pre webové rozhranie	40
5.6 Návrh webového rozhrania	41
5.7 Implementácia webového rozhrania	43
5.8 Optimalizácia	46
5.9 Implementácia pre Raspberry Pi 2B	46
5.10 Implementácia pre zariadenia TTGO	47
5.11 Testovanie funkčnosti	48
5.12 Problémy a limitácie	48
6 Záver	50
Literatúra	51

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 rozdeluje 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

Zoznam obrázkov

2.1	Chirp signál a porovnanie rozprestieracích faktorov. Prevzaté z [7]	13
2.2	Chirp signály, do ktorých boli modulované symboly. Prevzaté z TODO	14
2.3	Proces spracovania odosielaných dát v LoRa (Binárne hodnoty su ukážkové, nezodpovedajú reálnej konverzií). Prevzaté z TODO	15
2.4	Čítanie symbolov preambulu prijatého paketu a synchronizácia čítacieho okna na základe posunu preambulu.	16
3.1	Mikrokontroléry TTGO. Prevzaté z [12]	21
3.2	Zariadenie Armachat. Prevzaté z [14]	22
4.1	Schéma Synchronous LoRa Mesh. Prevzaté z [18]	26
5.1	Rozhranie zariadenia Armachat	38
5.2	Zariadenie Armachat prijalo spravu s jedným preskokom	39
5.3	Zariadenie Armachat prijalo traceroute správu	39
5.4	Štruktúra JSON objektu reprezentujúceho správu	40
5.5	Návrh úvodnej stránky	42
5.6	Návrhy stránok kontaktov a nastavení	42
5.7	Informačná ikonka v pravom dolnom rohu pri správach	43
5.8	Stránka nastavení	44
5.9	Stránka na mobilnom zariadení	45

Zoznam tabuliek

2.1	Frekvenčne pásma používané pre LoRa	16
3.1	Parametre Semtech SX modemov	20
4.1	Porovnanie LoRa mesh protokolov	27
5.1	Štruktúra hlavičky paketu a dĺžka jednotlivých polí v bajtoch.	30
5.2	Štruktúra dátového rámcu pre textové správy.	31
5.3	Štruktúra dátového rámcu pre správy typu traceroute.	31
5.4	Štruktúra dátového rámcu pre ACK a senzorové správy.	31
5.5	Ukážka bajtov odoslaného paketu	39

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 tejto 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 sa využíva riešenie LoRaWAN, ktoré sa skladá z centrálnych uzlov pripojených k internetu a zariadení, ktoré sú pripojené k centrálnym uzlom. Zariadenia potom komunikujú len s centrálnym uzlom a predávajú mu svoje dátu. Centrálny uzol potom dátu posielá 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 topológií (napr. Meshtastic [1], LoRaMesher [2]).

V tejto práci sa budeme venovať návrhu a vytvoreniu protokolu, ktorý by umožnil komunikáciu medzi zariadeniami v sieti tvorenjej pomocou technológie LoRa, bez nutnej existencie centrálnych uzlov. Nami vytvorený protokol bude tvoriť sietovú topológiu typu mesh, ktorá ma oproti hviezdicovej topológií, využívanej pri LoRaWAN, niekoľko výhod. Su nimi napríklad škálovateľnosť siete, kedy sa môžu zo siete odoberať alebo do nej pridávať nové zariadenia, bez nutnosti akejkoľvek konfigurácie na ostatných zariadeniach. Z toho vyplýva aj mobilita zariadení. Zariadenia sa môžu fyzicky pohybovať a pokial sa nachádzajú v dosahu hocjakého 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.

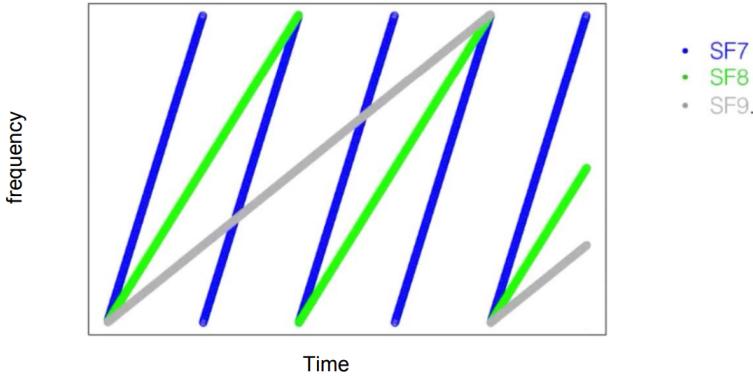
2.1 LPWAN

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 pripojenia do elektrickej siete. Zariadenia bývajú napájane batériami a môžu nepretržite fungovať aj niekoľko rokov. V kombinácií 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 Sub-GHz frekvenčné pásma, ktoré nevyžadujú na používanie žiadnu rádiovú licenciu.

Technológií využívajúcich LPWAN siete je viacero. Medzi tie najznámejšie patria napríklad Sigfox [3], LoRa [4], NB-IoT a iné.



Obr. 2.1: Chirp signál a porovnanie rozprestieracích faktorov. Prevzaté z [7]

2.2 LoRa

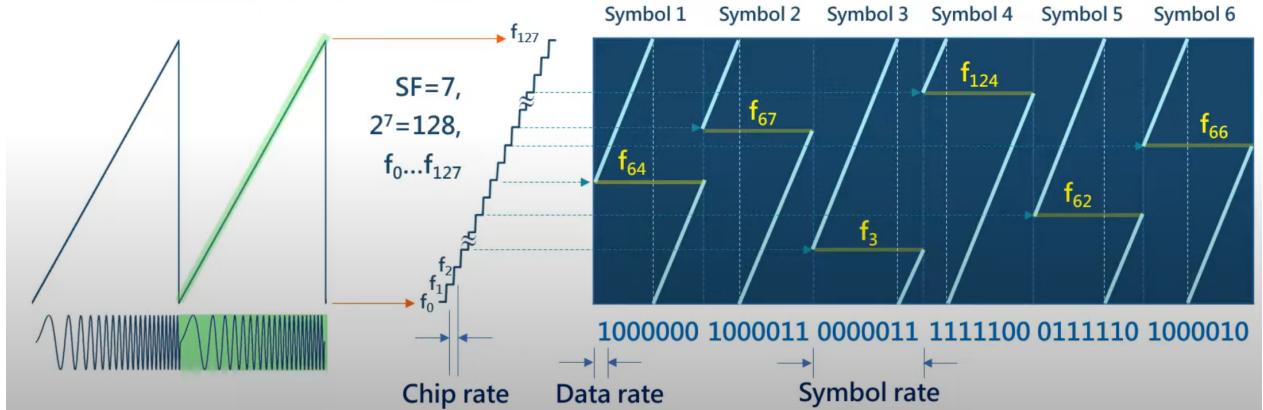
LoRa je proprietárna technológia na bezdrôtový prenos dát za pomocí rádiových vĺn. Používa bezlicenčné rádiové pásma, ktoré sú odlišné medzi Európu, Amerikou a Áziou [5], a poskytuje rádiový prenos na veľkú vzdialenosť s nízkou spotrebou energie. V otvorenom priestranstve môže mať rádiový prenos dosah až 10–15 km [6]. 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. Prevzaté z TODO

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 citlosť, ktorou dokáže prijímač prijať signál a dôsledkom toho je menší použiteľný dosah.

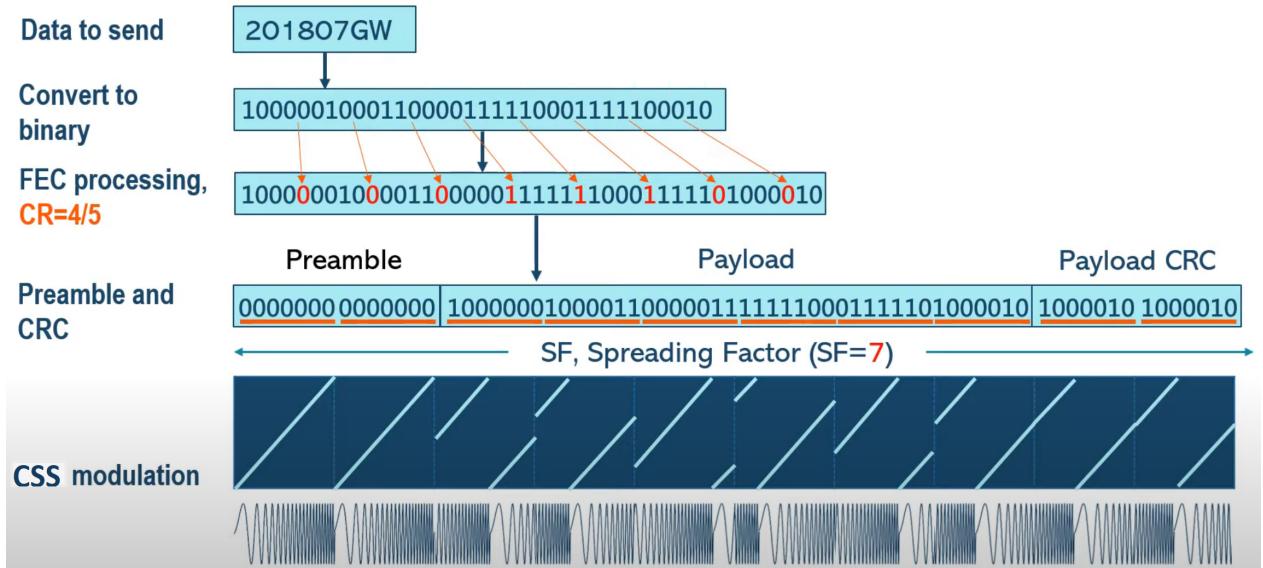
Každý chirp signál je rozdelený na X častí – tzv. chips. Tieto chips predstavujú skoky vo vysielacej frekvencii signálu. Kolko týchto chips jeden chirp obsahuje je závisle od vybraného rozprestieracieho faktoru. Jeden chirp je rozdelený na 2^{SF} častí alebo chips.

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 korešpondujúcej 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 su ukážkové, nezodpovedajú reálnej konverzií). Prevzaté z TODO

2.3 LoRa paket

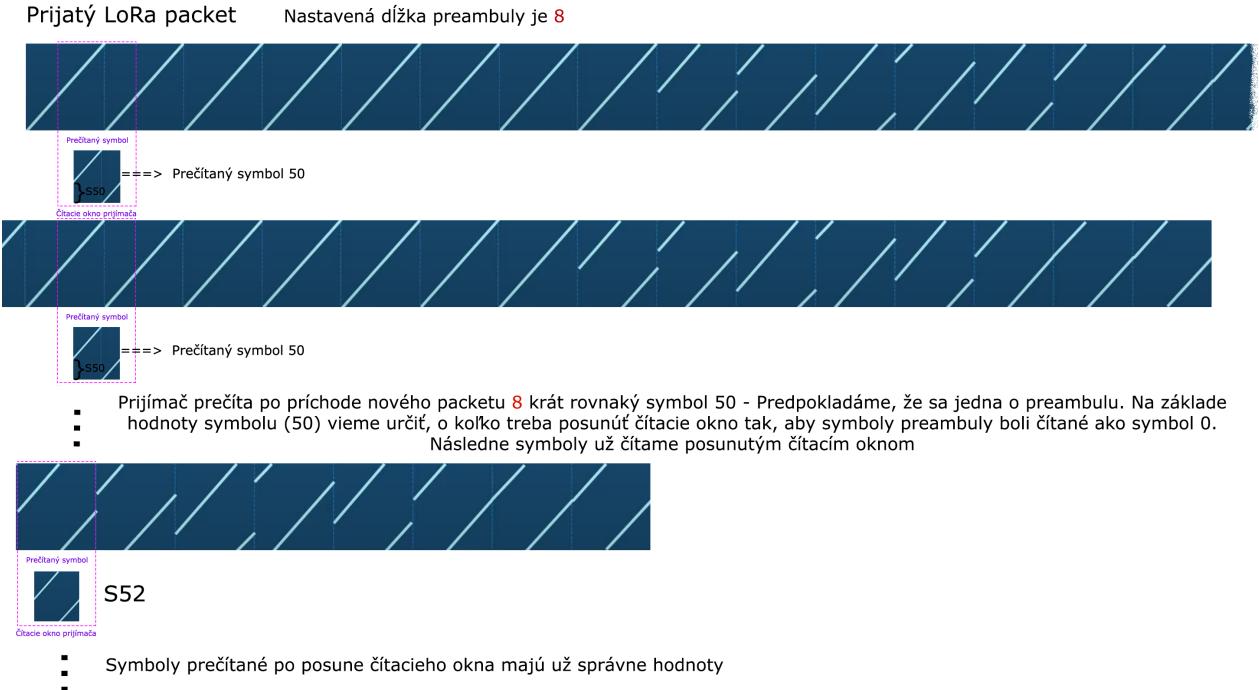
Štruktúra LoRa paketu môže byť závislá od daného použitia. Bežné sa ale stretávame s LoRa paketom, zloženým z preambuly, dát a kontrolného súčtu. Pričom maximálna povolená dĺžka dát je 255 bajtov.

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 prázdne 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 frekvenčí 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.

Parameter bandwidth (BW) určuje šírku pásma, v ktorom sa bude chirp posúvať. Pri vyššej šírke pásma sa zvyšuje rýchlosť prenosu, avšak znižuje sa použiteľný dosah. Najbežnejšie používané sú šírky pásma 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 znižuje 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 bity. Tieto bity 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. 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 bity informácie pridáva 1 korekčný bit. Vyšší kódovací pomer zabezpečí spoľahlivejší prenos dát ak sa nachádzame v rušivom prostredí, avšak zníži rýchlosť prenosu dát, pretože ku prenášaným dátam pridáva navyše dátá potrebné na korekciu chýb.

Rýchlosť prenosu dát (Data rate – DR) môžme 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, ktorý je spravovaný organizáciou LoRa Alliance [4].

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. Ostatne 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. TODO tu este zo tri styri vety aby strana nekoncila prazdnym miestom

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 frekvenciach vysielať v rámci jednej hodiny a maximálny vysielací 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 [8] stanovuje vo všeobecné oprávnění č. VO-R/10/07.2021-8 [9], že frekvenčné pásmo, do ktorého spadá technológia LoRa, patrí do skupiny g. Pre túto skupinu je maximálny povolený klučovací pomer 1 % a maximálny vysielací 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 vysielacím 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 môžme 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 ale majú vyššiu spotrebu energie.

V tejto časti sa budeme zaoberať dostupnými modulmi pre koncové zariadenia. Kedže je technológia LoRa patentovaná výrobcom Semtech [10], 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 [10], prináša LoRa čipy série SX127x a SX126x. Ponúkajú vysoký výkon za pomere nízku cenu a ako sme už spomíname, ostatné LoRa moduly iba implementujú tieto LoRa čipy – modemy a rozširujú ich o ďalšiu funkcionality.

3.1.1 SX127x

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

Maximálny vysielací 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 budget 157 dB oproti 168 dB pri modemoch SX1276/77/78/79 a majú menší rozsah frekvenčných pásiem medzi 860 a 1020 MHz. Okrem toho majú aj vyššiu spotrebu energie.

Pri modemoch SX1276/77/78/79 je možné vybrať frekvenčné pásma z rozsahu 137 až 1020 MHz.

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 prečízny 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 [11] 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 zdieľa 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 simulovanie jednoduchých koncových zariadení, ktoré môžu predstavovať napríklad nejaký senzor, boli použité mikrokontroléry TTGO od výrobcu LILYGO [12].

Okrem mikrokontrolérov TTGO boli použité aj mikrokontroléry Raspberry Pi Pico a jednodoskový počítač Raspberry Pi. Nami použité mikrokontroléry môžu byť neskôr rozšírene o batériu a byť mobilné.

3.3.1 TTGO LoRa32

Tento mikrokontrolér je založený na module ESP32. Obsahuje Wi-Fi, bluetooth a LoRa moduly. Konkrétnie používa SX1276 LoRa modem.

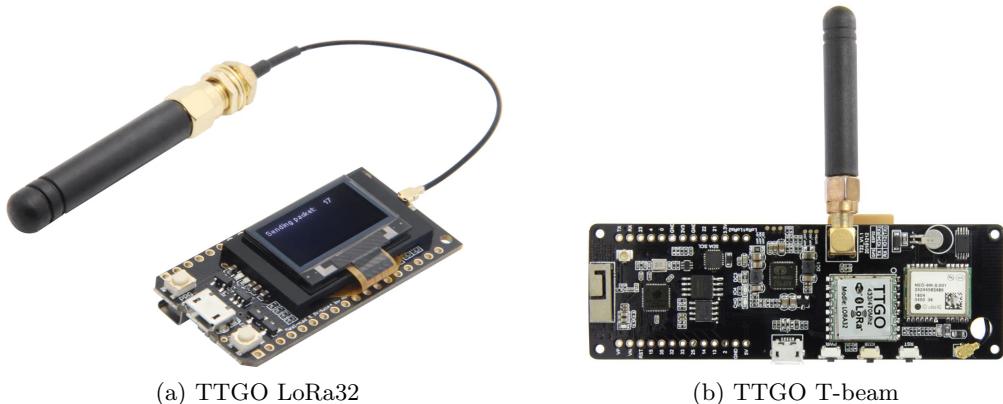
Dá sa prepnúť do úsporného režimu spánku, ktorý znižuje spotrebu mikrokontroléru na 0,6 mA. Mikrokontrolér obsahuje aj 0,96 palcový čiernobiely displej, ktorý je pripojený cez I2C rozhranie.

3.3.2 TTGO T-Beam

Tento mikrokontrolér je taktiež založený na module ESP32 a obsahuje Wi-Fi, bluetooth a LoRa modul. Rovnako používa SX1276 LoRa modem no okrem spomínaných modulov obsahuje aj GPS modul.

Mikrokontrolér ma na zadnej strane držiak na batériu, z ktorej môže byť napájaný a taktiež aj nabíjací modul pre bezpečné nabíjanie a ochranu lítiových batérií. V režime spánku ma spotrebu 0,2 mA. Podporuje pripojenie dodatočného displeja, avšak nami používaný mikrokontrolér tento displej nemá.

Na Obr 3.1 môžeme vidieť oba mikrokontroléry TTGO.



Obr. 3.1: Mikrokontroléry TTGO. Prevzaté z [12]

3.3.3 Raspberry Pi Pico RP2040

Mikrokontrolér od známej organizácie Raspberry Pi [13], 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 bude použitý práve CircuitPython.

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 [14]

3.3.4 Raspberry Pi 2B

Jednodoskový, štvor-jadrový počítač vytvorený organizáciou Raspberry Pi.

Zariadenie obsahuje eternetový port, ktorý môže slúžiť na pevné pripojenie do internetovej siete. Počítač sam o sebe neobsahuje žiadny modul a tak pre prácu s LoRa bude pridaný RFM96W modul pripojený do vstupno-výstupných pinov počítača.

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

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

Kapitola 4

Existujúce riešenia

Téma mesh sietí je v tejto dobe veľmi aktuálna a vývojári sa snažia vytvoriť rôzne riešenia, ktoré by nahradili aktuálne používane prístupy topológie hviezda v IoT sietach. Použitie mesh topológie môže so sebou prinášať rôzne výhody oproti bežne zaužívaným topológiám typu hviezda.

Vo výskume od Ochoa et al. [15] porovnávali spotrebu energie medzi topológiou typu mesh a hviezdicovou topológiou v sieti LoRa. A z týchto štúdií možno vyvodíť záver, 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 sietach ešte viac optimalizovala spotreba energie.

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

Ďalším zaujímavým projektom je Armachat [16], 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 vymoženosti.

Originálny projekt avšak sám o sebe 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.

4.1 LoRa mesher

LoRaMesher [2] je C++ knižnica, ktorú je možné použiť na komunikáciu v LoRa mesh sietach.

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

K realizácií 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.

LoRaMesher dokáže nájsť novovo vytvorené uzly v sieti vďaka smerovaciemu protokolu. Pri odoslaní správ čaká na prijatie potvrdzujúcej ACK správy, ktorá potvrzuje, že správa bola prijatá a tým zaistuje spoloahlivosť. Správy väčšie ako 222 bajtov rozdeľuje na viacero správ, ktoré pošle postupne.

4.2 Meshtastic

Meshtastic [1] vytvára mesh siet za použitia lacných mikrokontrolérov s LoRa modulmi. Myšlienka tohto projektu spočíva v tom, že vytvára komunikačnú sieť na miestach kde neexistuje spoloahlivá 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 (pokial 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 siete.

4.3 LoRaBlink

Ďalší multi-hop 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.

Siet sa skladá z jedného uzlu, ktorý plní rolu takzvaného datasinku (gateway alebo brána) a ostatných uzlov. Uzly sietu 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 vysielat dátu. 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 dátu, tak vyberie najskorší volný slot a v nom vysiela svoje dátu. Ak tieto dátu príjme uzol, ktorý nie je datasink a hop count daného uzlu ku sinku je menší ako hop count vysielacieho uzlu, tak dátu v ďalšom volnom slote znova prepošle. Toto sa opakuje, až kým dátu nedosiahnu datasink.

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

4.4 Pymesh

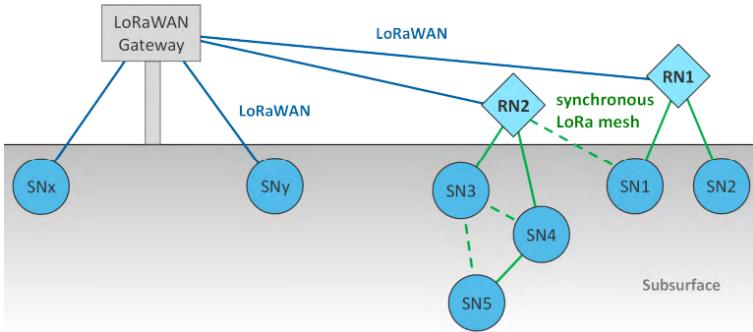
Pymesh je súčasťou Pycom [17] ekosystému. Tento ekosystém je určený na vývoj IoT systémov. Ponúka rôzne zariadenia, ktoré sú určené na použitie s týmto ekosystémom. Zariadenia obsahujú WiFi, bluetooth a LoRa, prípadne SigFox moduly. Zariadenia je možné rozšíriť o rôzne moduly so senzormi, ktoré rozširujú ich schopnosti.

PyMesh siet 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 siet 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.

Uzly v sieti môžu komunikovať ad-hoc. V sieti môže dojsť k situácií kedy bude chcieť viacero uzlov vysielat 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 vysielal. 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.

Na smerovanie paketov v sieti využíva kombináciu dvoch prístupov. 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 najskôr 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.



Obr. 4.1: Schéma Synchronous LoRa Mesh. Prevzaté z [18]

4.5 Synchronous LoRa Mesh

Tento projekt[18] 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 su ale energeticky náročne, drahé a vyžadujú pevné pripojenie do elektrickej a prípadne aj internetovej siete, ktoré v kanalizáciách 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 viacero LoRaWan gateway.

Tento projekt sa snaží vyriešiť tieto problémy. Prináša protokol, ktorý rozširuje LoRaWan o tzv. repeater uzly (RN) vid 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. Všetky uzly si držia v sebe smerovaciu tabuľku. Po pripojení nového uzla sa sieťou sa propaguje správa na aktualizovanie smerovacej tabuľky.

4.6 Porovnanie voči nášmu riešeniu

Niektoré zo spomenutých riešení využívajú smerovacie tabuľky na efektívnejší prenos dát v rámci siete. Smerovacie tabuľky 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. To pridáva do siete zataženie v podobe dodatočných dátových prenosov.

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

Protokol navrhnutý v tejto práci nebude využívať žiadne smerovacie tabuľky. Vďaka tomu dosiahneme mobility uzlov bez potreby periodických aktualizácií smerovacích tabuľiek. Hlavným využitím nášho protokolu je komunikácia v reálnom čase, preto protokol nebude používať žiadnu časovú synchronizáciu ani časové okná vyhradené na komunikáciu, uzly siete tak budú môcť prijímať a odosielat dátu hocikedy. To so sebou ale prináša nevýhodu v podobe vyššej spotreby energie.

Funkcionalita navrhnutého protokolu bude, podobne ako niektoré zo spomínaných riešení, fungovať na základe multi hop flooding, kedy sa správa v sieti preposiela cez uzly siete až kým nedorazí do destinácie. Správa sa odošle a uzol, ktorý túto správu prijal ako posledný ju prepošeďalej. Toto sa opakuje na ostatných uzloch v sieti až kým sa správa nedostane do destinácie alebo kym správe nedôjde limit preskokov, prípade TTL.

Navrhnutý protokol nie je závislý na žiadnych špeciálnych centrálnych uzloch typu gateway, prípadne nejakých riadiacich uzloch. Každý uzol v sieti môže byť jednoduchým uzlom, ktorý bude prijímať a preposielat dátu ďalej. Vďaka tomu môžu byť uzly realizované prostredníctvom lacných a malých zariadení.

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

Obsah správ v sieti bude šifrovaný šifrovacím algoritmom AES.

Tabuľka 4.1: Porovnanie LoRa mesh protokolov

Protokol	LoRa mesher	Meshtastic	LoRaBlink	Pymesh	Synchronous LoRa Mesh	Nás protokol
Centrálné uzly	Nie	Nie	Áno	Áno	Áno	Nie
Mobilita uzlov	Áno**	Áno	Áno*	Áno**	Áno*	Áno
Smerovanie	Distance Vector	Flooding	Flooding	Distance Vector	TODO	Flooding
Ad-hoc komunikácia	Áno	Áno	Nie	Áno	Nie	Áno
TODO	nejake	dalsie	atributy			

*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

Kapitola 5

Vlastná implementácia

Ako sme už spomínali v predchádzajúcej kapitole, nami vytvorený protokol bude fungovať na základe multi hop flooding. Tento prístup so sebou ale nesie niekoľko nevýhod, ktoré by sme mali zohľadniť pri navrhovaní implementácie.

Jednou z nevýhod je, že ak pošleme niekam správu, a destinácia práve nie je v dosahu, tak by sme o správu prišli. Tento problém vyriešime tak, že odosielateľ si bude odoslané správy ukladať a v prípade, že sa nepodarí správu doručiť, môže ju opäťovne odoslať niekedy neskôr. To, že sa správu nepodarilo doručiť do destinácie, zistíme vďaka potvrzovacím ACK správam. Tieto správy sa odosielajú v opačnom smere ako správa, ktorú potvrdzujeme.

Ďalším z problémov môže byť potenciálne zahľtenie siete správami. Keby každý uzol v sieti preposielal ďalej každú správu, ktorú príjme, tak by sme rýchlo dostali do stavu kedy by množstvo uzlov preposielalo rovnaké správy a sieť by sa tym pádom zahľtila. Riešenie tohto problému bude spočívať v niekoľkých procesoch ako stavu zahľtenia predchádzať. Tieto procesy si bližšie popíšeme v kapitole implementácie.

Implementácia sa bude deliť na dve verzie. Jedna verzia, implementovaná v CircuitPython, bude určená pre výkonnejšie zariadenia. Táto implementácia bude obsahovať aj webové rozhranie na obsluhu, posielanie a prijímanie sprav. Webové rozhranie avšak pôjde využiť len na zariadeniach, ktoré podporujú WiFi prípadne ethernet.

Implementácia v C++ bude určená pre menej výkonne zariadenia. Táto implementácia bude len zjednodušená verzia kompletnej implementácie a bude určená pre mikrokontroléry TTGO. Z dôvodu výkonnosti a obmedzenej pamäti bude táto implementácia obsahovať iba základné funkcionality protokolu, potrebné na odosielanie správ s údajmi so senzorov. Mikrokontroléry TTGO tak budú v sieti fungovať iba ako jednoduché uzly, ktoré budú v určitých časových intervaloch odosielať dátu.

5.1 Typy a stavы 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, je nutné si definovať aj určité stavы, v ktorých sa správa môže nachádzať. Tieto stavы potom budú určovať, 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 bude slúžiť na potvrdenie doručenia správy. Ked' odosielateľ pôvodnej správy niekedy neskôr príjme ACK správu, ktorá potvrdzuje doručenie správy, tak môžeme pôvodnú správu považovať za doručenú.

Textová správa je správa, ktorá bude obsahovať textový reťazec. Rozšírením tejto správy je správa s potvrdením o doručení. Obsah textových správ bude šifrovaný.

Správa typu senzorové dát bude určená na prenos dát z rôznych senzorov. Obsah správy bude taktiež vo forme textového reťazca, ktorý bude obsahovať dátu zo senzorov. Tento typ správ avšak oproti obyčajnej textovej správe ponuka možnosť nastaviť časový interval – TTL, po ktorého uplynutí sa správa nebude ďalej šíriť sietou. Obsah správy bude taktiež šifrovaný.

Sprava žiadosť o traceroute sa bude používať len v krajných prípadoch, kedy by nás zaujímalo akou cestou sa správa preposiela sietou. Ked nejaký uzol príjme správu typu žiadosť o traceroute, tak vytvorí novu správu typu odpoveď na traceroute a vloží do nej svoju adresu. Adresátom tejto novej správy bude odosielateľ pôvodnej správy. Ked sa bude tato odpoveď na traceroute šíriť sietou, tak každý uzol, ktorý danú správu preposal dalej, pridá do nej svoju vlastnú adresu. Takto sa vytvorí reťazec, ktorý bude obsahovať adresy všetkých uzlov, cez ktoré bola správa preposlaná. Aby mohli ostatné uzly v sieti do správy postupne pridávať svoje adresy, tak obsah tejto správy nebude šifrovaný.

Posledným typom správy je nespracovaný paket. Tento typ správy bude využívaný iba v prípade, že ma používateľ zapnutý monitorovací režim. V tomto režime sa všetky správy, ktoré sa v sieti prenášajú, budú zaznamenávať. V tomto režime môžeme zachytiť aj pakety, ktoré niesu súčasťou nášho protokolu a ich štruktúra nebude zodpovedať našim špecifikáciám. Z toho dôvodu ich nie je možné spracovať. Uložia sa teda ako typ správy nespracovaný paket, a budu obsahovať iba holé prijaté dátu vo forme bajtov.

5.1.2 Stavy správ

Na správnu funkčnosť je potrebné aby správy existovali v určitých stavoch. Správy budu časom svoj stav aktualizovať a prechádzať medzi nimi. Tieto stavы budú definované nasledovne:

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

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

Správy v stave preposlaná, pri ktorých nepotrebuje sledovať, či správa dorazila do destinácie, sa následne presunú do stavu hotová.

Pokiaľ aktuálny uzol vyčerpá všetky pokusy o odoslanie správy, prípadne pri správe typu senzorové dátia dôjde časový limit, správa sa presunie do stavu neúspešná alebo zmazaná. To či sa presunie do stavu neúspešná alebo zmazaná je znova závisle na tom, či je aktuálny uzol autorom danej správy alebo nie.

Správam, ktoré sú v stave preposlané sa odpočítava časový limit na prijatie potvrdenia. Pokiaľ v tomto časovom intervale nedorazí potvrdzovacia ACK správa, tak sa správa presunie do stavu nepotvrdená (NAK). V opačnom prípade sa presunie do stavu potvrdená (ACK).

Stav zmazaná značí, že správu považujeme za vymazanú. Ďalej s ňou už nebudeme pracovať a po uplynutí určitého intervalu sa reálne vymaže z pamäte.

5.2 Návrh paketu

Paket bude obsahovať hlavičku a dátový rámc. Štruktúra dátového rámcu bude závisieť od typu správy. Navrhnutú štruktúru hlavičky paketu môžeme vidieť v tabuľke 5.1

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

2B	2B	4B	2B	1B	1B	0 – 240B
Destinácia	Odosielateľ	ID správy	Kontrolný súčet	Typ správy	Priorita	Dátový rámc

Pre adresy odosielateľa a destinácie budeme používať dvoj bajtové adresy. Štvor bajtový identifikátor správy bude náhodne generovaný pri každom vytvorení novej správy.

Z adresy destinácie, adresy odosielateľa a identifikátoru správy sa vytvorí dvoj bajtový kontrolný súčet. Tento kontrolný súčet bude používaný na overenie, či prijatá správa patrí 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.

V položke typ správy bude číslo, ktoré definuje typ danej správy. Číslo bude reprezentovať jeden z typov spomínaných v predošej sekcií.

Správam môžme v hlavičke paketu určiť vyššiu prioritu. Na túto prioritu sa následne bude dbať pri spracovávaní správ a správy s vyššou prioritou budu odbavené prioritne.

Dátové rámce sa budú lísiť 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	1B	0 – 238B
Počet skokov	Pôvodný počet skokov	Textové dátá

1B	1B
Počet skokov	Pôvodný počet skokov

(a) Správa typu žiadost o traceroute.

1B	0 – 239B
Počet skokov	Navštívené adresy

(b) Správa typu odpoveď na traceroute.

Tabuľka 5.3: Štruktúra dátového rámcu pre správy typu traceroute.

V niektorých dátových rámcach môžme vidieť položku počet skokov. Do tejto položky bude pri vytvorení novej správy zapísaný maximálny počet skokov, ktoré môže správa vykonať pri prenose sietou. Každý uzol, predtým ako správu prepošeľ ďalej, zmenší počet skokov o jedna. Ak sa počet skokov dostane na nulu tak správu ďalej nespracovávame a nastaví sa jej adekvátny stav.

V rámcach tiež nájdeme položku pôvodný počet skokov. Do tejto položky bude tak isto pri vytvorení novej správy zapísaný maximálny počet skokov. Hodnota v tejto položke sa však už ďalej nemení. Bude slúžiť k tomu, aby prijímateľ dokázal zistíť, kolko preskokov správe zabralo, kým správa dorazila až k nemu. Túto informáciu bude možné následne použiť na správne nastavenie maximálneho počtu skokov pre odoslanie potvrzovacej správy, prípadne správy typu odpoveď na traceroute.

1B	4B
Počet skokov	ID potvrzovanej správy

(a) Potvrzovacie ACK správy.

2B	0 – 238B
TTL	Senzorové dátá

(b) Správy pre senzorové dátá.

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

V dátovom rámcu, používanom na senzorové dátá, používame namiesto maximálneho počtu preskokov TTL. TTL vyjadruje časový interval, po ktorý môže byť daná správa spracovávaná a preposielaná v sieti. Táto hodnota sa pri vytvorení správy nastaví na predom určený časový interval v sekundách. Každý uzol, ktorý bude danú správu preposielat, si overí ako dlho už bola správa

uložená u neho v pamäti a na základe toho odčíta hodnotu z TTL. Po odčítaní hodnoty z TTL, uzol správu ďalej prepošle.

Pri odčítavaní TTL sa berie do úvahy iba čas, ktorý daná správa strávila na určitom uzle. Nebudeme brať do úvahy čas, ktorý správe zabralo preniesť sa rádiovými vlnami z jedného uzlu na ďalší. A to z toho dôvodu, že nie sme schopní presne určiť, akú vzdialenosť prekonala vysielaná správa rádiovými vlnami. Existujúce spôsoby merania vzdialnosti, ktorú správa prešla vzduchom, produkujú len hrubé odhady a sú veľmi závisle od aktuálnych atmosférických podmienok. Preto sme sa rozhodli, že v našom riešení budeme počítať iba s časom, ktorý správa strávila na určitom uzle. Mimo toho, keby sme počítali 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 je v porovnaní s časom, ktorý správa strávi na určitom uzle, zanedbateľná.

5.3 Funkcionalita protokolu

Protokol bude fungovať tak, že novo prijatú správu si každý uzol u seba uloží. Následne sa uložené správy budú spracovávať v hlavnom cykle programu. Každú správu, ktorú uzol príjme, sa bude následne snažiť znova preoslať ďalej, pokiaľ teda nebola daná správa určená práve tomu uzlu.

Uzol bude každú správu odosieláť určitý počet krát a medzi každým odoslaním konkrétnej správy bude uzol čakať predom stanovený časový interval predtým ako ju môže znova odoslať. Toto viacnásobne odosielanie každej správy nám zabezpečí vyššiu spoľahlivosť v doručení. Keby sa každá správa preosielala iba jeden krát, mohlo by sa stať, že na ten prvý raz by tuto vyslanú správu nezachytil žiadny ďalší uzol a správa by sa tak nedoručila ďalej.

Medzi tým ako sa budú správy spracovávať a preosielat bude uzol zároveň počúvať na novo prichodzie správy od iných uzlov. Ak by náhodou uzol prijal správu, ktorá už bola prijatá a uložená u neho tak to berie ako potvrdenie, že sa správa šíri ďalej v sieti. Uzol teda u seba danú správu zahodí a už ju ďalej nebude posielat.

Ak by došlo k situácií, kedy uzol jednu správu opakovane prepošle maximálny možný počet pokusov, a medzitým nepríjme danú správu preoslanú iným uzlom, tak považujeme tuto situáciu ako neúspešne odoslanie správy. Správe sa nastaví adekvátny stav a nebude sa ďalej spracovávať.

Každý uzol prijatej správe zníží hodnotu maximálneho počtu preskokov. Pokiaľ sa však jedná o správy typu senzorové dátá, tak sa neznižuje hodnota maximálneho počtu preskokov ale hodnota TTL, a to pred každým znova odoslaním danej správy. Pokiaľ by uzol prijal správu, ktorej hodnota maximálneho počtu skokov dosiahla nulu, tak túto správu nebude u seba ani ukladať. Ak by sa počas spracovávania správ na uzle dostala hodnota TTL nejakej senzorovej správy na nulu, tak tuto situáciu tak isto považujeme za neúspešne odoslanie správy. Správe sa nastaví adekvátny stav a nebude sa ďalej spracovávať.

Obsah textových a senzorových správ bude vždy šifrovaný, pokiaľ nie je správa posielaná na broadcast adresu. Na šifrovanie bude použitý šifrovací algoritmus AES, ktorý šifruje bloky dát za pomoci klúča. Jedna sa o symetrickú šifru, takže zašifrované dáta môžme rovnakým klúčom neskôr dešifrovať. To ponúka možnosť aby si určitá skupina používateľov vytvorila vlastný šifrovací klíč a iba oni budú môcť dešifrovať správy, ktoré budú vytvorené za použitia tohto klúča. Ostatní účastníci siete obsah správ nebudú schopní dešifrovať.

V protokole bude možné odosielať správy na broadcast adresu. Tieto správy budú prijaté každým uzlom.

5.4 Implementácia protokolu

Hlavnou súčasťou nami navrhnutého protokolu bude list správ. Do tohto listu sa budú pridávať novo vytvorené, prípadne novo prijaté správy. Následne sa bude v hlavnom cykle programu prechádzať tento list správ a každá správa sa bude spracovať.

Na to aby sme správy mohli ukladať do tohto listu, bolo potrebné vytvoriť istú dátovú štruktúru, do ktorej sa budú ukladať jednotlivé správy. Správy v liste budu so sebou držať rôzne informácie. Okrem tých základných informácií o správe akými su 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 patri napríklad čas, kedy bola správa naposledy spracovávaná aktuálnym uzlom. Táto informácia sa využíva pri kontrole či uz nadišiel čas správu znova preoslať ale aj na to aby sme vedeli kolko času treba odčítať z hodnoty TTL pri senzorových správach.

Pri prechádzaní listu správ sa pri každej správe skontroluje jej stav a to či časový limit už dosiahol požadovanej hodnoty. Opakovane znova odosielanie správy, vykonávame iba pri správach v stave nová a odoslaná. Spomínali sme, že je možné určiť správe vyššiu prioritu. Tato vyššia priorita zabezpečí to, že sa pri správe, ktorá má nastavenú vyššiu prioritu, bude vykonávať opakovane znova odosielanie správy, aj keď jej časový limit ešte neboli dosiahnutý. Tým dosiahneme toho, že správa sa skôr odošle 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 je práve to, že keď uzol príjme správu, ktorú už ma u seba uloženú, tak u seba tuto správu označí ako zmazanú (stav zmazaná) a nebude ju ďalej posielat. Správy v zmazanom stave niesu ale ihneď vymazané z pamäti. Miesto toho sa im nastavuje časový interval a až po tom čo uplynie tento interval su správy skutočne vymazané z pamäti. Vďaka tomu, že si držíme v pamäti správy aj keď su považované za zmazané, môžme overiť či novo prijatá správa nie je náhodou jedná z tých zmazaných. Ak áno tak novú správu nebudeme znova ukladať. K tejto situácii môže bežne dochádzať a neošetrenie tohto problému by mohlo viest k zacykleniu, 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, je 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 zabezpečíme tak, že každej novo prijatej správe sa nastaví 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, dokážeme približne odhadnúť, ktorý uzol je vzdialenejší. Uzly ktoré su vzdialenejšie budu mať nižšiu hodnotu SNR a teda aj nižšiu hodnotu časového oneskorenia. To povedie k tomu, že vzdialenejšie uzly odvysielajú správu skôr, ako uzly ktoré su bližšie. Ostatné uzly toto vysielanie príjmu, a u seba označia danú správu ako zmazanú, keďže už nie je potrebné aby ju oni preposielali.

Ďalší z potrebných aspektov správneho fungovania nášho protokolu je zabezpečiť aby nedošlo k situácií kedy by dva alebo viac uzlov v sieti začalo vysielať signály v rovnakom čase. K tomu sme využili takzvaného prístupu listen before talk, kedy sa uzol predtým ako začne vysielať presvedčí, či náhodou práve nevysiela niekto iný. Ak áno tak tak sa vyslanie správy pozdrží a skúsi 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 sa používa 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 sme vytvorili triedu Message. Trieda obsahuje všetky potrebné informácie o správe. Taktiež obsahuje metódy na vytváranie rôznych typov a obsahov správ. Tieto metódy budú slúžiť na prvotné vytvorenie nových správ, na autorskom uzle. Okrem týchto metód je ale potrebné mať aj metódy, ktoré dokážu z prijatých dát vytvoriť správu. Pri prijatí nového paketu v LoRa, dostaneme dátá vo forme bajtového poľa. Tieto dátá sú potom predané do metódy, ktorá z nich vytvorí 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 listu bolo ale potrebné vytvoriť ešte akýsi kontajner, ktorý by v sebe uchovával správu plus dodatočne informácie potrebne k ďalšiemu spracovávaniu. K tomu sme vytvorili triedu MessageQueueItem. Tá v sebe drží inštanciu triedy Message, počítadlo opakovanych odoslaní, časový odpočet, časové razítko, ktoré zachytáva čas, kedy bola správa naposledy spracovaná, aktuálny stav správy a ďalšie užitočné informácie. Okrem toho obsahuje trieda metódy, ktoré budu využívané 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 nastaví na aktuálny čas a mnoho ďalších.

Po implementácii triedy MessageQueueItem sme mohli začať implementovať samotný proces funkcionality nášho protokolu. List správ sme realizovali ako slovník, kde kľúčom je identifikátor

správy, a hodnota pod daným klúčom je inštancia triedy MessageQueueItem. To nám umožní rýchlo vyhľadávať správy na základe jej identifikátora. Tento slovník sme pomenovali message queue.

Hlavný proces protokolu bude pozostávať z cyklu, v ktorom sa najskôr skontroluje či sme náhodou neprijali nový LoRa paket a následne sa prejdú všetky správy z message queue. Ak je to potrebné tak sa určité správy spracujú. K tomu vznikli dve hlavné funkcie. Funkcia receive(), ktorá skontroluje či sme neprijali nový LoRa paket a funkcia tick(), v ktorej sa prejdú všetky správy zo zoznamu.

Funkcia receive() prečíta novo priyatý paket z LoRa a následne overí, či daný paket splňa dĺžku aspoň 12 bajtov. Vychádza to z toho, že všetky správy nášho protokolu obsahujú predom definovanú hlavičku paketu (viď. Tab. 5.1), ktorá ma veľkosť 12 bajtov. Ak je paket kratší ako 12 bajtov, môžme s istotou povedať, že nejde o správu nášho protokolu a teda ho nemusíme ďalej spracovať. Po kontrole dĺžky paketu, následuje overenie kontrolného súčtu. Funkcia zoberie prvých 8 bajtov paketu a vypočíta sa z nich kontrolný súčet. Ak sa vypočítaný kontrolný súčet zhoduje s kontrolným súčtom, ktorý je v pakete, tak správu považujeme za súčasť nášho protokolu. Na výpočet kontrolného súčtu bola použitá funkcia cyklicky redundantného súčtu typu CRC-16/CCITT-FALSE. Overovanie, či paket prislhuje do nášho protokolu je pomerné dôležité, pretože na rovnakých LoRa parametroch môžu byť vysielané aj pakety iných protokolov a tieto pakety by sa nam nepodarilo správne spracovať keďže by nespĺňali našu štruktúru paketu.

Potom čo sa vo funkcií receive() overí, že paket splňa naše požiadavky, bude z neho vytvorená nova inštancia triedy Message. Využijeme k tomu metódu triedy Message určenú na vytvorenie správy z pola bajtov. Pri novo priatej správe nás zaujíma či je správa určená nám. Ak je správa určená nám, je potrebné zistiť o aký typ správy sa jedna. Pokiaľ sme prijali správu typu ACK, tak je nutné vyhľadať v message queue korešpondujúcu spravu a zmeniť jej stav na ACK. Pokiaľ sa nejedná o ACK správu, následuje kontrola, či sa daná správa už nenachádza v message queue a ak nie tak je tam pridaná.

Je dôležité, aby uzol po prijatí správy, odoskal naspäť ACK správu. A to aj v prípade, že sa nejedná o typ správy, ktorý vyžaduje potvrdenie o doručení. Dôvodom je, že ak by uzol neposlal naspäť ACK správu, tak by predošlý uzol pokračoval v opakovanom vysielaní správy, a tieto vysielania by zbytočne obsadzovali prenos v sieti. Uzol teda po prijatí vyšle ACK správu. V prípade, že prijatá správa nevyžadovala potvrdenie o doručení, tak môže uzol ACK správe nastaviť maximálny počet preskokov na 0. Tým pádom sa správa neposunie ďalej v sieti ako k najbližšiemu uzlu. Ak by sa ale jednalo o správu vyžadujúcu potvrdenie o doručení, tak sa maximálny počet preskokov nastaví na hodnotu, ktorá bola uložená v prijatej správe pod položkou pôvodný počet skokov. Tym zaručíme, že správa bude mať dostatočný počet dostupných skokov aby sa dostala naspäť k uzlu, ktorý pôvodnú správu poslal.

V prípadnom scenárii, že by prijatá správa nebola určená nám, skontrolujeme či sa rovnaká správa už nenachádza v message queue, ak nie tak ju tam pridáme za predpokladu, že má ešte

dostupný nejaký počet skokov alebo TTL. Správe zároveň nastavíme prvotné časové pozdržanie na základe hodnoty SNR, s ktorou sme ju prijali.

V opačnom prípade, kedy sa správa už nachádza v message queue, je nutné znova vykonať sériu overení a na základe nich zmeniť stav správy. Prvé z overení je, či daná správa bola vytvorená nami. To či bola vytvorená nami, zistime z adresy odosielateľa správy. Adresa sa bude zhodovať s tou našou ak bola vytvorená nami. V tom prípade nastala situácia kedy našu správu preoslal nejaký iný uzol. Môžeme teda správe zmeniť stav na hotovo, prípadne na stav preoslaná ak sa jednalo o správu s potrebným potvrdením o doručení.

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

Po dokončení funkcie receive(), následuje funkcia tick(). V tejto funkcií sa postupne prechádza všetkými spravami z message queue. Pri každej správe sa pozrieme na jej stav a ak je v stave preoslaná alebo zmazaná tak sa následne skontroluje, či jej už nevypršal časový limit. Pokial limit vypršal, tak sa správa skutočne vymaže z message queue ak bola v stave zmazaná. Ak bola v stave preoslaná, tak sa jej stav zmení na nepotvrdená – NAK.

V prípade, že by bol stav správy nová alebo odoslaná, tak sa pokračuje v jej spracovávaní, ktoré pozostáva z následujúcich krokov. Ako prvé sa overí, či už nadišiel čas danú správu spracovať. To zistíme tak, že sa pozrieme či správe už vypršal časový limit. Ak áno, znamená to, že správa je pripravená na spracovávanie a môžeme pokračovať ďalej. Ak nie tak sa daná správa preskočí a prejdeme na ďalšiu správu.

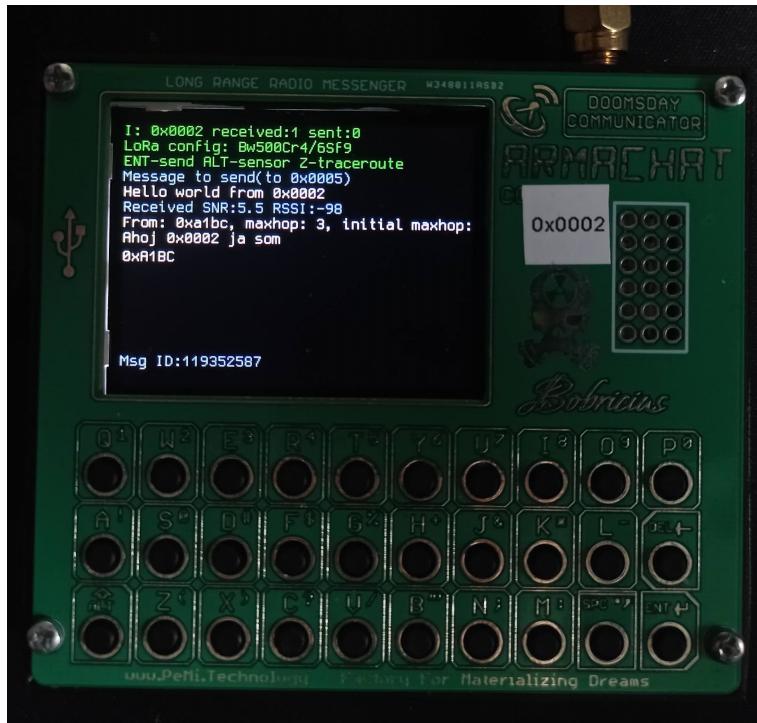
Ak nadišiel čas správu spracovať, tak skontrolujeme, či ma ešte dostupné nejaké pokusy o odoslanie a v prípade senzorových správ aj či má ešte dostupný TTL. Ak sa stane, že správa už vyčerpala všetky pokusy o odoslanie alebo jej TTL došlo na nulu, tak sa správa presunie do stavu neúspešná alebo zmazaná ak to nebola správa vytvorená nami. Keby správe ešte zostali nejaké pokusy o odoslanie, tak správu odošleme cez LoRa. Keď sa jednalo o správu typu senzorové dát, tak sa správe pred odoslaním zmenší TTL.

Po odoslaní správe aktualizujeme aktuálne časové razítko, znížime počítadlo dostupných pokusov o odoslanie a nastavíme nový časový limit pre správu. Tento limit bude závisieť od konfiguračnej premennej, ktorá určuje, ako často sa ma správa znova odosielat. Pokial bola správa pred odoslaním v stave nová, tak jej stav aktualizujeme na odoslaná.

Pri pokuse o odoslanie môže dôjsť k situácií, kedy práve niekto iný vysiela signál a tak je nutné pozdržanie odoslania. To realizujeme tak, že správe nastavíme časové pozdržanie na hodnotu konfiguračnej premennej a aktualizujeme jej časové razítko. Výsledkom bude, že táto správa sa bude znova pokúšať o odoslanie až po uplynutí časového intervalu.

Po dokončení implementácie hlavnej funkcionality sme ju mohli otestovať. Na zariadeniach Armachat sme vytvorili jednoduché rozhranie, ktoré zobrazuje na displeji informácie o novo priatej správe. Taktiež môžeme využiť klávesnicu zariadenia, na napísanie vlastnej správy a jej odoslanie.

Na obrázku 5.1 môžme vidieť, že zariadenie prijalo správu od odosielateľa s adresou 0xA1BC, ktoré bolo v tomto prípade druhé Armachat zariadenie.



Obr. 5.1: Rozhranie zariadenia Armachat

Okrem obsahu správy, môžme vidieť hodnoty SNR a RSSI, s ktorými zariadenie správu prijalo, hodnotu max hop, ktorá predstavuje počet preskokov a hodnotu initial max hop, ktorá predstavuje 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 5.2 môžme vidieť, že rozdiel medzi hodnotami max hop a initial max hop je 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 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.

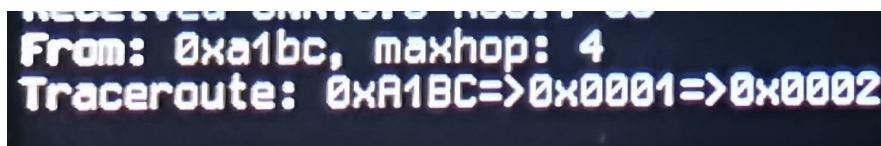
Taktiež si môžme všimnúť zmenu v podobe skrátenia názvov atribútov max hop a initial max hop, z dôvodu, že sa na displeji nezmestili do jedného riadku. Okrem toho je na obrázku vidieť informačnú správu v podobe žltého textu na samom spodku displeja. V tejto informačnej správe sa zobrazujú rôzne informačné 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.



Obr. 5.2: Zariadenie Armachat prijalo spravu s jedným preskokom

Následne sme vyskúšali, pri rovnakom rozmiestnení zariadení, odoslať zo zariadenia 0x0002 žiadosť o traceroute na zariadenie 0xA1BC. Na obrázku 5.3 môžeme vidieť, ako zariadenie 0x0002 prijalo odpoveď na žiadosť o traceroute. V obsahu správy vidíme zoznam adres, 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 reťazec „Ahoj“. V prvých 12 bajtoch, vidíme hlavičku paketu. V ďalších 6 vidíme počet preskokov, pôvodný počet preskokov a šifrovaný obsah správy.



Obr. 5.3: Zariadenie Armachat prijalo traceroute správu

Tabuľka 5.5: Ukážka bajtov odosланého paketu

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

5.5 Implementácia API pre webové rozhranie

Pri používaní nášho protokolu má používateľ možnosť využiť webové rozhranie, cez ktoré je možné ovládať zariadenie a pracovať s naším LoRa protokolom. Používateľ ma možnosť vytvárať alebo prijímať nove správy, ktoré sa zobrazujú na webovej stránke. Okrem toho je 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, je nutné najskôr implementovať nejaké API. Cez API budeme komunikovať so zariadením z webového rozhrania. Toto API bude bežať na serveri, ktorý pobeží na zariadení spolu s naším protokolom. API bude realizované prostredníctvom klasických HTTP požiadavkov.

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

Prvá ruta slúži na získanie správ, ktoré boli prijaté na zariadení. Je volaná prostredníctvom HTTP GET požiadavku a vracia zoznam obsahujúci entity jednotlivých správ. Každá správa je reprezentovaná JSON objektom, ktorý obsahuje iba potrebné informácie o správe, k tomu aby bolo možné zobrazovať správy na webovej stránke. Štruktúru tohto JSON objektu môžeme vidieť na ukážke 5.4.

```
{
  '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,
    'rssı': Number,
    'lora_config': String, //Informácie o LoRa parametoch napr. "Bw500Cr45Sf128"
  }, //Informácie o LoRa
  'hop_count': Number //Počet skokov, ktoré správa vykonala
}
```

Obr. 5.4: Štruktúra JSON objektu reprezentujúceho správu

Druhá ruta slúži na odoslanie správy prostredníctvom HTTP POST požiadavku. Táto ruta prijíma JSON objekt, ktorý reprezentuje novú správu. V JSON objekte musia byť uvedené všetky potrebné informácie o správe, ktoré sú potrebné na jej správne odoslanie. Sú nimi destinácia, maximálny počet skokov, priorita, obsah správy a príznak, či chceme potvrdenie o doručení. API po

prijatí požiadavku na vytvorenie novej správy validuje vstupné údaje a ak sú všetky údaje v poriadku, tak vytvorí novú správu. V opačnom prípade vráti chybovú hlášku. Z dôvodu limitácie LoRa paketov na maximálnu dĺžku 255 bajtov pre posielané dátá, je 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 povolojuje 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 je routa na získavanie a zmenu aktuálnej konfigurácie. Vznikla na to routa „/api/config“. Táto routa môže byť volaná prostredníctvom HTTP GET požiadavku, kedy vráti aktuálnu konfiguráciu alebo prostredníctvom HTTP PUT požiadavku, kedy nastaví novú konfiguráciu z poskytnutého JSON objektu, zachytávajúceho nové nastavenia.

Používateľ má možnosť nastaviť rôzne parametre, ktoré ovplyvňujú správanie protokolu. Nastavené hodnoty parametrov sa budú ukladať do JSON súboru na zariadení. 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 vyzeráť. Na vytvorenie návrhov webových stránok bol použitý dizajnérsky nástroj Figma.

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

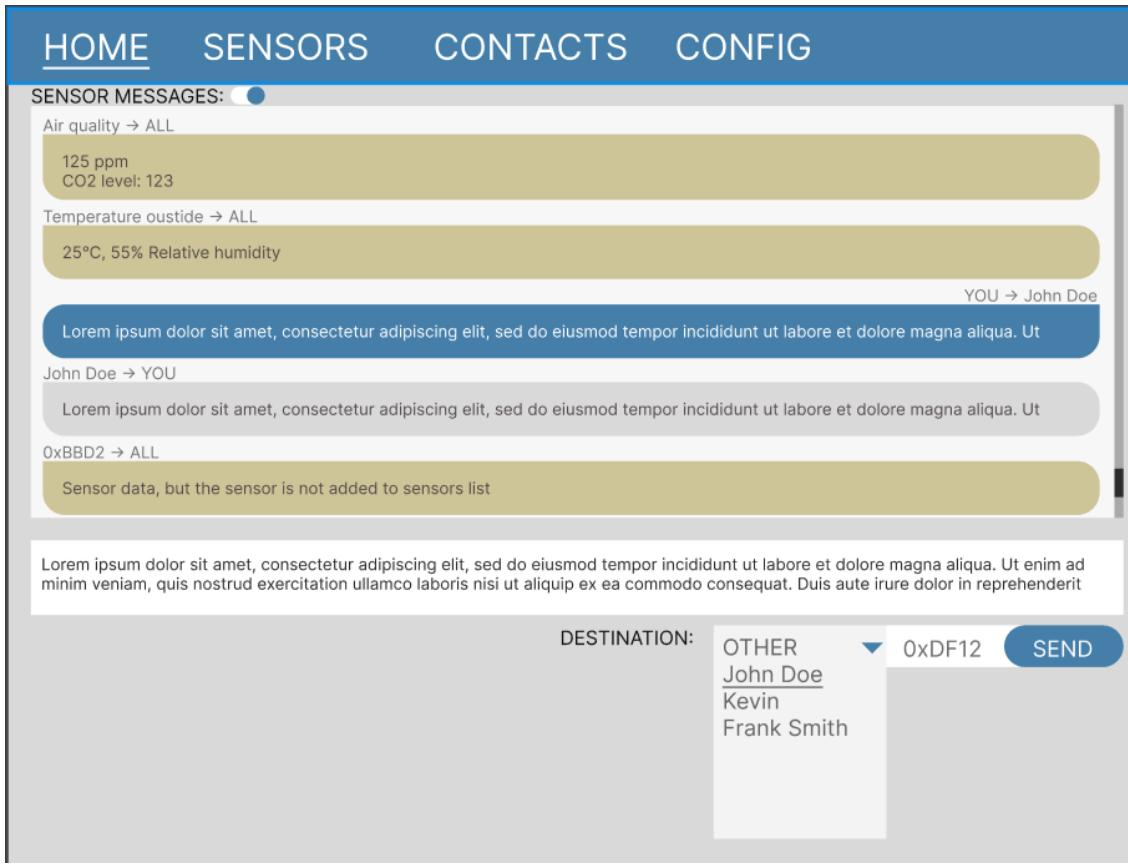
Rozhodli sme sa pridať aj možnosť ukladať si kontakty do adresára, ktorý bude používateľovi umožňovať vidieť namiesto hexadecimálnych adries v správach, názov kontaktu. Tieto kontaktné adresáre budú rozdelené pre kontakty na iných používateľov a kontakty ne senzorové uzly. Z toho dôvodu boli potrebne dve ďalšie stránky, kde používateľ bude môcť spravovať svoje kontakty.

Poslednou stránkou bola stránka na konfiguračné rozhranie. Táto stránka bude obsahovať všetky možnosti, ktoré používateľ bude môcť nastavovať.

Výsledne návrhy môžeme vidieť na nasledujúcich obrázkoch. Stránky pre správu kontaktov (Obr. 5.6) na používateľov a kontaktov na senzorové uzly budú vyzeráť rovnako.

Na obrázku 5.5 zachytávajúcom návrh úvodnej stránky si môžme všimnúť, že niektoré adresy sú zobrazené ako mená z kontaktov.

Pri tvorbe návrhu stránky na konfiguráciu nastavení (Obr. 5.6), nebolo ešte známe, aké všetky nastavenia bude môcť používateľ meniť a tak sú tam iba nastavenia adresy, šifrovacieho klúča a LoRa parametrov.



Obr. 5.5: Návrh úvodnej stránky

Kedže sme pridali novú funkciu v podobe pridávania kontaktov, bolo potrebné vytvoriť novú API routu, ktorá bude vraciať zoznam kontaktov. Okrem toho bola vytvorená ďalšia routa, slúžiaca na pridávanie nových alebo mazanie starých kontaktov. Ako databázu kontaktov sme použili JSON súbor, do ktorého sa budu kontakty ukladať priamo na zariadení.

(a) Kontakty

(b) Nastavenia

Obr. 5.6: Návrhy stránok kontaktov a nastavení

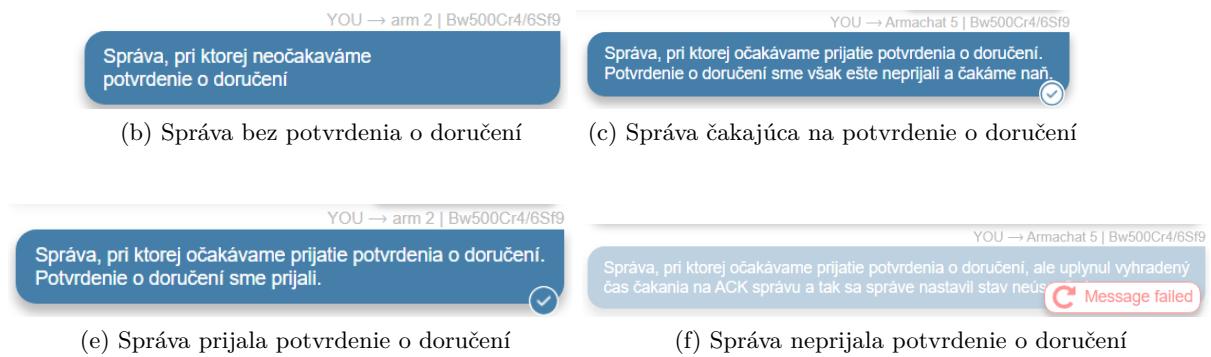
5.7 Implementácia webového rozhrania

Na implementáciu webového rozhrania bola použitá kombinácia HTML, CSS a JavaScriptu. Na komunikáciu s API bol použitý fetch API, ktorý je súčasťou JavaScriptu. Ako také volanie API za použitia fetch API môžeme vidieť v nasledujúcim kóde.

```
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)
```

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 sa bude zobrazovať pri nami odoslaných správach a bude indikovať, či bola správa prijatá alebo nie. Na obrázku 5.7 vidíme ukážku, ako tieto jednotlive stavy budú vyzerat. Môžme si tiež všimnúť, že nad správu pribudla dodatočná informácia o použitých LoRa parametroch. Taktiež môžme vidieť nove tlačítka, ktoré sa zobrazuje pri neúspešných správach.



Obr. 5.7: Informačná ikonka v pravom dolnom rohu pri správach

Tlačítko, zobrazujúce sa pri neúspešných správach, pridáva možnosť neúspešnú správu skúsiť znova odoslať. K tomu aby to fungovalo, bolo potrebné pridať ďalšiu API routu. Táto routa príjme ako parameter ID správy, ktorú chceme odoslat znova a správa sa znova odošle.

Stránku nastavení sme voči návrhu rozšírili o ďalšie nastavenia a spolu s tým sme rozšírili aj popis jednotlivých nastavení a čo robia. Obmedzili sme možnosť nastavenia vlastnej adresy na jednorázové nastavenie. Týmto sa snažíme zamedziť tomu, aby si používateľ kedykoľvek upravoval svoju adresu a tým sa mohol vydávať za iného používateľa. Finálnu stránku nastavení vidíme na obrázku 5.8.

The screenshot shows a configuration interface with the following sections:

- LoRaWAN Settings:**
 - My Address: 0x0005 (button: RANDOM)
 - AES Key: SuperTajne heslo
 - Resend count: 5
 - Resend timeout(S): 8
 - ACK wait timeout(S): 60
 - Randomize enabled:
 - Monitoring mode:
- LoRa config:**
 - Bandwidth: 500 kHz
 - Coding rate: 4/6
 - Spreading factor: 9
- Notes:**
 - 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 packets 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.*
- WiFi configuration:**

Connected to: WajFaj, IP: 192.168.100.33

Add new WiFi network	Saved networks
SSID Access point siet	WIFI siet
PASSWORD	Dalsia WIFI siet
Use this network as AP: <input checked="" type="checkbox"/>	AP Access point siet
ADD	DELETE

Obr. 5.8: Stránka nastavení

Všetky nastaviteľné položky sú validované predtým, ako sa novo upravené nastavenia odošlú na server. Táto validácia je potom redundantne vykonávaná aj na strane serveru.

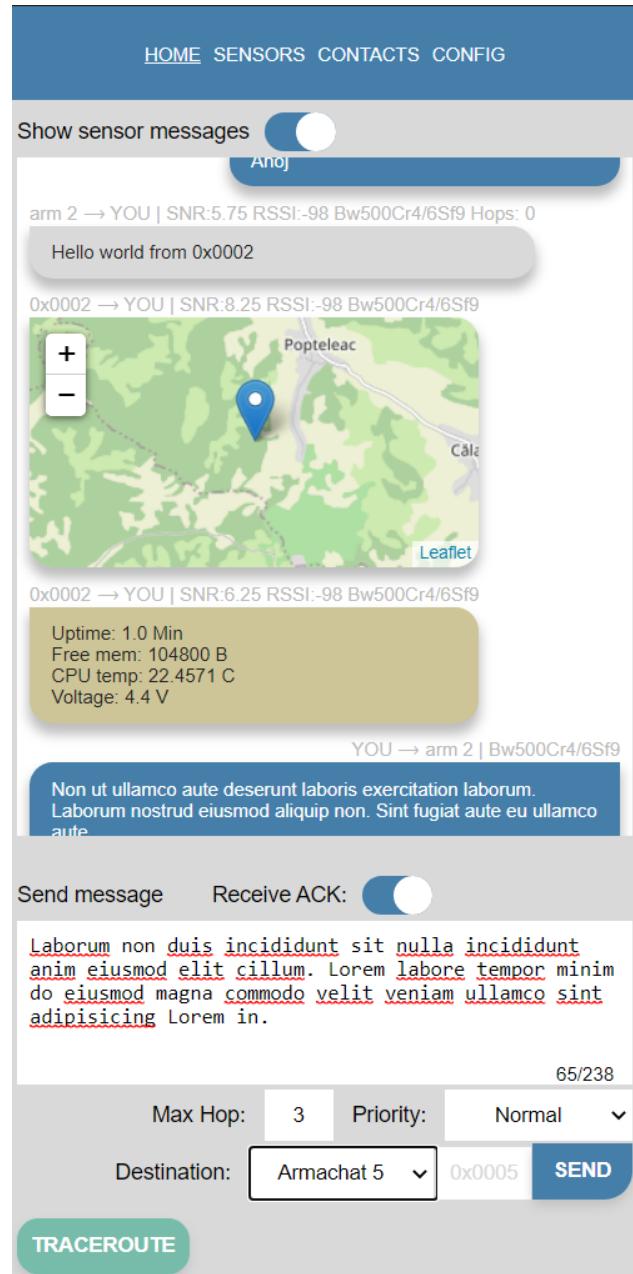
Na stránku nastavení sme pridali možnosť pridávať alebo odstraňovať WiFi siete. Na tieto siete sa bude snažiť zariadenie pripojiť pri spustení a následne sa rozbehne server. Je možné definovať

si aj vlastnú sieť typu access point. Pri použití tejto možnosti sa zariadenia pri spúštaní nebude pripájať na žiadnu WiFi sieť, ale vytvorí svoju vlastnú.

V našej LoRa mesh sieti sa môžu vyskytovať senzorové uzly, ktoré obsahujú GPS modul. Z toho dôvodu sme pridali do webového rozhrania funkciu, ktorá zobrazí na mape aktuálnu polohu zariadenia. Správy, ktoré chceme zobraziť ako polohu na mape, musia dodržať určitý formát a to taký, že správa musí obsahovať prefix „GPS:“ a následne musia nasledovať súradnice zemepisnej šírky a zemepisnej dĺžky oddeľene čiarkou. Ako takáto správa s mapou vyzera, môžme vidieť na obrázku 5.9. Na integráciu mapy do webového rozhrania bola použitá knižnica Leaflet.js [19].

Pri implementácii sme dbali 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 zmenšujú veľkosti jednotlivých prvkov v rozhraní, mení sa aj celkové usporiadanie prvkov na stránke tak, aby bolo používanie rozhrania prívetivejšie.

Aby sme odlišili nami odoslané správy od správ, ktoré sme prijali od nejakého iného uzlu, bolo pridané farebné odlišenie jednotlivých správ. Správy, ktoré sme prijali od iného používateľa sú zobrazené v sivej farbe. Správy ktoré sme prijali od senzorových uzlov, sú zobrazené v žltej farbe. A správy, ktoré sme odoslali my, sú modré. Toto farebné odlišenie je vidieť na obrázku 5.9. Spominali sme aj možnosť posielania traceroute správ. Tieto správy sú zobrazené v zelenej farbe.



Obr. 5.9: Stránka 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 zoznam správ sa držali všetky správy, ktoré boli určené danému uzlu, alebo ktoré sám uzol vytvoril a odosielal. 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.

Preto sme sa rozhodli, že dĺžku zoznamu správ budeme limitovať. Ak zoznam správ dosiahne určeného limitu, tak sa z neho odstráni 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 všetky dátá 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.

API na získanie správ sme preto upravili tak, že používa takzvané stránkovanie. Toto stránkovanie funguje tak, že API rozdelí zoznam správ na menšie časti – stránky a vracia iba jednu stránku. Pri volaní API sa pridáva parameter špecifikujúci, ktorú stránku má API vrátiť. Tento prístup má za následok, že webová stránka bude volať API na získanie správ nie raz, ale viac krát, až kým nezíska 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 väčšej 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.

Kedže Raspberry Pi má väčšiu pamäť, tak sme mohli zvýšiť limit dĺžky zoznamu správ.

CircuitPython na Raspberry Pi nemá podporu pre všetky knižnice, ktoré používame v implementácií pre Armachat zariadenia. Bolo nutné nájsť vhodnú náhradu pre knižnice, ktoré nemali podporu a prispôsobiť implementáciu náhradným knižniciam. Jednalo sa o knižnicu na šifrovanie pomocou AES šifry a knižnicu na webový server.

5.10 Implementácia pre zariadenia TTGO

Z dôvodu menšieho výkonu a hlavne menšej pamäti sme sa rozhodli implementáciu pre TTGO zariadenia zjednodušiť. Kedže 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 uzly, ktoré budú posielat nejaké senzorové dáta. Implementácia preto nebude obsahovať žiadne užívateľské rozhranie, server ani webovú stránku. Implementácia bude obsahovať iba základné funkcie, ktoré sú potrebné pre odosielanie senzorových správ. Senzorové uzly budú posielat správy iba jednosmerne, nie je potreba aby senzorové uzly nejaké správy prijímaliby. Kvôli tomu, že senzorové uzly nebudú prijímať žiadne správy, nebudú schopné ani preposielat ďalej cudzie správy. Dosah mesh siete teda nie je možné rozšíriť pomocou týchto uzlov.

Rovnako ako pri predošlých verziách pre Armachat a Raspberry Pi, je možné konfigurovať určité nastavenia. Táto konfigurácia sa vykonáva upravovaním konfiguračných premenných, predtým ako je program preložený a nahraný do zariadenia. Súčasťou konfigurácie je zoznam kontaktov, na ktoré bude senzorový uzol odosielat správy.

Implementácia pozostáva z funkcií, ktoré postupne poskladajú hlavičku paketu a k nej pridajú obsah senzorovej správy. Tieto novo vytvorené správy sa uložia do zoznamu a rovnako ako pri plnej verzii programu, sa zoznam správ cyklicky prechádza a správy z neho sa odosielajú, keď nadíde ich čas.

Každá správa je taktiež odosielaná viackrát. Počet kolko krát sa ma správa odoslať je daný konfiguračnou premennou. Kedže táto zjednodušená implementácia nerieši prijímanie správ, každá správa sa bude opakovane posielat daný počet pokusov aj napriek tomu, že bola daná správa uz preposlaná ďalej nejakým iným uzlom. Po tom čo sa správa odošle daný počet krát, sa správa zoznamu správ odstráni.

Hlavný chod programu pozostáva z cyklu v ktorom sa prechádzajú správy zo zoznamu správ a funkcie, ktorá generuje senzorové dáta. Na zariadení TTGO T-Beam sme použili integrovaný GPS modul, z ktorého získavame GPS súradnice a z nich tvoríme obsah senzorovej správy. Zariadenie TTGO LoRa32 má integrovaný hall effect senzor, ktorý sníma úroveň magnetického poľa. Na tomto zariadení generujeme senzorové správy, ktorých obsahom je úroveň magnetického poľa plus čas ako dlho je daný senzor v prevádzke.

Funkcia na vygenerovanie novej senzorovej správy je volaná v určitých intervaloch. Tento interval je možné nastaviť konfiguračnou premennou. Výsledkom je tak uzol v mesh sieti, ktorý periodicky vysielá hodnoty zo senzorov. Adresátmi týchto správ sú uzly, ktorých adresy ma zariadenie uložené v konfiguračnej premennej kontaktov. Ak do zoznamu kontaktov zadáme broadcast adresu – 0xFFFF, tak správa bude doručená všetkým uzlom v sieti.

Výsledná implementácia je veľmi odlahčená verzia nášho protokolu, ktorá je vhodná pre použitie na tých najjednoduchších zariadeniach. Pridáva to možnosť rozšíriť sieť o rôzne monitorovacie

senzorové zariadenia, ktorých obstarávacia cena je vďaka ich jednoduchosti pomerne nízka.

5.11 Testovanie funkčnosti

TODO

- test ze to funguje, rozmiestnenie zariadeni, posielanie sprav, simulovanie sensorovych uzlov atd
- screenshoty z aplikacie
- test vykonnosti, packet loss, ping, dosah atd

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. Tento limit je pri zariadení Armachat s WiFi modulom len 10 správ. Kvôli tomu, že na tomto zariadení nám musí bežať webový server, je pamäť zariadenia veľmi vyťažená.

Tento limit 10 správ, však nemusí znamenať až taký problém. Pokial ma používateľ otvorené webové rozhranie tak sa správy ukladajú aj na strane webovej aplikácie bežiacej v prehliadači používateľa. Ak dôjde k dosiahnutiu maximálneho limitu správ tak sa z pamäte zariadenia odstráni najstaršia správa. V pamäti prehliadača na webovej aplikácii ale táto správa ostane. Takže používateľ je schopný vidieť aj správy staršie ako posledných 10 správ.

Na zariadeniach Armachat bez WiFi modulu je možné vidieť len tu najnovšiu správu na displeji, takže limit 10 správ tu taktiež nie je taký problém.

Veľkou limitáciou našej implementácie je, že CircuitPython nemá podporu pre externé prerusenia. Z toho dôvodu knižnica, ktorú sme použili na ovládanie zabudovaného LoRa modulu, nepracuje veľmi efektívne. Keď aktuálne LoRa modul prijíma nejaký paket, tak sa pozdrží cely zvyšok bežiaceho programu, až kým sa paket nepríjme a neprečíta celý alebo nedôjde k vypršaniu časového limitu. Toto nam spomaľuje beh programu zakaždým keď je v sieti vysielaný nejaký paket.

Použitá knižnica na ovládanie LoRa modulu so sebou prináša ešte dodatočnú limitáciu. Táto knižnica nesprávne pracuje s LoRa modulom a nie je možné stabilne používať LoRa prenos pokial je nastavený rozprestierací faktor vyšší ako 9. Pri nastavení vyššieho rozprestieracieho faktoru sa spoľahlivosť prenosu dramaticky zníži. Pri našom testovaní sa nám nepodarilo úspešne doručiť ani jednu správu z desiatich pokusov.

Chybu v knižnici sme sa snažili opraviť. Jedným z dôvodov prečo ku chybe dochádzalo bol nesprávne nastavený časový limit pre posielanie paketu. Časový limit bol nastavený na 2 sekundy. Avšak, ak sme si zobrali prípad paketu, ktorý by mal veľkosť 100 bajtov a bol vysielaný s rozprestieracím faktorom 12, kódovacím pomerom 4/8, a šírkou pásma 125 kHz, zistili sme že čas potrebný na vyslanie tohto paketu sa blíži 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 nie je dostatočné a naznačuje to, že problém je ešte niekde inde. Ten sa nam už ale nepodarilo nájsť.

Zariadenie Raspberry Pi pico s WiFi modulom, je pomerné nové. CircuitPython knižnica pre podporu WiFi modulu je stále v beta verzií a ma veľa nedostatkov. Neumožňuje prepnut WiFi modul do režimu AP (Access Point) ak bol modul už predtým prepnutý do režimu station a rovnako nie je možné prepnut modul do režimu station ak bol už prepnutý do režimu AP. Jediný spôsob ako WiFi modul úspešne prepniť medzi režimami je tvrdé resetovanie zariadenia.

Pôvodne sme mali v pláne implementovať štartovací proces zariadenia tak, že najskôr sa zariadenie pokúsi pripojiť k WiFi sietam ktoré má uložené v zozname a ak sa mu nepodarí pripojiť ku žiadnej z týchto sieti, tak vytvorí svoju vlastnú WiFi sieť, na ktorú sa môže používateľ pripojiť. Kvôli chybnému správaniu WiFi knižnice však tento prístup nie je možné realizovať. V momente kedy sa snaží WiFi modul pripojiť na nejakú sieť zo zoznamu WiFi sieti, prepne sa do režimu station. Potom už nie je možné modul prepniť do režimu AP ak sa nepodarí pripojiť k žiadnej zo sieti.

K WiFi knižnici sa viaže ešte jeden problém a to taký, že pokial sa snaží používateľ pripojiť na webový server bežiaci na Armachat zariadení ihneď potom, ako sa zariadenie pripojí na WiFi sieť, zariadenie sa dostane do zamrznutého stavu a prestane reagovať na čokoľvek. Na obnovenie funkcionality je potrebný tvrdý reštart zariadenia. Tomuto problému sa snažíme vyhýbať tak, že zariadenie zobrazí informáciu o spustenom web serveri až po určitom oneskorení a spoliehame sa na to, že používateľ sa bude pripájať na webový server až po zobrazení tejto informácie.

Kapitola 6

Záver

co sme dosiahli, že to funguje, porovnat v com je to lepsie oproti inym, porovnanie voci inym rieseniam spomenut ze niektore riesenia neponukaju mobilitu uzlov, ani ovladanie cez mobil atd.

tu mozem spomenut ze meshtastic ma problem s topologiou tvaru V, ja tento problem riesim cez randomize path prepinatko, pridat screeny z meshtastic simulatoru

moznosti rozsirenia, moznost zablokovat daku adresu aby som nevidel jej spravy, rozdelovat dlhsie spravy na dve a viac ak sa nezmestia do pozadovaneho limitu

Literatura

1. K. HESTER, et al. *Meshtastic—An Opensource Hiking, Pilot, Skiing, Secure GPS Mesh Communicator, 2020*. Dostupné tiež z: <https://meshtastic.org/>.
2. JOAN MIQUEL SOLÉ, et al. *LoRaMesher - implementing a distance-vector routing protocol for communicating messages among LoRa nodes*. Dostupné tiež z: <https://github.com/LoRaMesher/LoRaMesher>.
3. *Sigfox - První celorepublikový mobilní operátor pro internet věcí* [online] [cit. 2022-07-10]. Dostupné z : <https://sigfox.cz/>.
4. *LoRa Alliance* [online] [cit. 2022-07-10]. Dostupné z : <https://lora-alliance.org/>.
5. *LoRa Alliance, Inc. LoRaWAN® Regional Parameters* [online] [cit. 2023-09-04]. Dostupné z : <https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf>.
6. *What are LoRa® and LoRaWAN®?* [Online] [cit. 2023-09-04]. Dostupné z : <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>.
7. PIETROSEMOLI, Ermanno. *LoRa details*. Dostupné tiež z: http://wireless.ictp.it/school_2017/Slides/LoRaDetails.pdf.
8. Český Telekomunikační Úřad. Dostupné tiež z: <https://www.ctu.cz/>.
9. *všeobecné oprávnění č. VO-R/10/07.2021-8 k využívání rádiových kmitočtů a k provozování zařízení krátkého dosahu*. Dostupné tiež z: <https://www.ctu.cz/sites/default/files/obsah/vo-r10-072021-8.pdf>.
10. *Semtech* [online] [cit. 2022-07-10]. Dostupné z : <https://www.semtech.com/>.
11. *HopeRF* [online] [cit. 2022-07-10]. Dostupné z : <https://www.hoperf.com/>.
12. *LILYGO*. Dostupné tiež z: <http://www.lilygo.cn/>.
13. *Raspberry Pi Foundation* [online] [cit. 2023-09-04]. Dostupné z : <https://www.raspberrypi.org/>.

14. *Armachat - LORA messenger with Raspberry Pi PICO* [online] [cit. 2023-09-04]. Dostupné z : <https://www.tindie.com/products/bobricius/armachat-lora-messenger-with-raspberry-pi-pico/>.
15. 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.
16. MISENKO, Peter. *Armachat - Doomsday LORA QWERTY communicator*. Dostupné tiež z: <https://github.com/bobricius/armachat>.
17. *Pycom, Pymesh—LoRa Full-Mesh Network Technology*. Dostupné tiež z: <https://docs.pycom.io/pymesh>.
18. 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.
19. *Leaflet.js* [online] [cit. 2023-09-04]. Dostupné z : <https://leafletjs.com/>.