VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

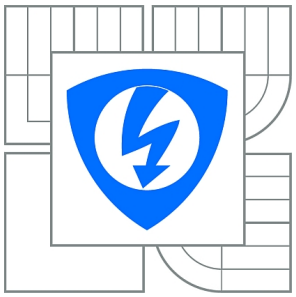
NÁVRH SYSTÉMU PRO VZDÁLENÝ UPGRADE FIRMWARU PRO UZLY BEZDRÁTOVÉ SENZOROVÉ SÍTĚ

DIPLOMOVÁ PRÁCA
MASTER'S THESIS

AUTOR PRÁCE Bc. TOMÁŠ MINÁR
AUTHOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH SYSTÉMU PRO VZDÁLENÝ UPGRADE FIRMWARU PRO UZLY BEZDRÁTOVÉ SENZOROVÉ SÍTĚ

OVER THE AIR FIRMWARE UPGRADE FOR WIRELESS SENSOR NETWORKS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

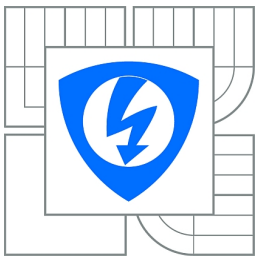
AUTOR PRÁCE — Bc. TOMÁŠ MINÁR

AUTHOR

VEDOUCÍ PRÁCE — Ing. ĽUBOMÍR MRÁZ

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Tomáš Minár ***ID:*** 125547
Ročník: 2 ***Akademický rok:*** 2013/2014

NÁZEV TÉMATU:

Návrh systému pro vzdálený upgrade firmwaru pro uzly bezdrátové senzorové sítě

POKYNY PRO VYPRACOVÁNÍ:

Student v práci prozkoumá možnosti vzdáleného upgradu firmwaru v rámci bezdrátové senzorové sítě. Cílem práce je implementace a vyhodnocení systému pro vzdálený upgrade firmwaru pro mesh sítě pomocí komunikační sady Light Weight Mesh. V práci bude dále srovnání upgradu firmwaru mezi LWM a Zigbee. Student bude mít k dispozici komunikační uzly deRFNode a software od společnosti Atmel.

DOPORUČENÁ LITERATURA:

[1] GISLASON, Drew. Zigbee Wireless Networking.: Newnes, 2008. 448 s. ISBN 0750685972.
[2] FARAHANI, Shahin. ZigBee Wireless Networks and Transceivers. : Newnes, 2008. 360 s. ISBN 0750683937

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 28.5.2014

Vedoucí práce: Ing. Ľubomír Mráz
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práca je venovaná jednej z problematík bezdrôtových senzorových sietí – preprogramovanie firmwaru uzlov rádiovým prenosom (OTAU). V prvej kapitole sú rozobrané požiadavky a formy prenosu firmwaru do jednotlivých uzlov. Druhá kapitola je venovaná najmä riešeniu problému rozoslania firmwaru do siete na základe analýzy dostupných protokolov. Praktická časť práce pozostáva v návrhu a realizácii systému na aktualizovanie uzlov siete v prostredí Lightweight Mesh od spoločnosti Atmel. Implementácia bola realizovaná na platforme deRFnode s modulmi deRFmega128. V závere je vyhodnotenie procesu aktualizácie navrhnutého systému na testovacej sieti. Praktické výsledky testov sú porovnané z riešením OTAU v BitCloud.

KĽÚČOVÉ SLOVÁ

Upgrade firmwaru, bezrôtové senzorové siete (WSN), Lightweight mesh, deRFnode platforma, Atmel AVR

ABSTRACT

This diploma thesis is dedicated to one of the problems in wireless sensor networks – over the air upgrade (OTAU) of nodes. The requirements for upgrade and possible ways how to transfer new firmware image to nodes are stated in the first chapter. The second chapter is focused on solving the problem of firmware dissemination to whole network based on analysis of known protocols. The practical part of this thesis deals with OTAU design and implementation in Lightweight Mesh software stack from Atmel. Proposed system was tested on deRFnode platform with plugged in deRFmega128 module. The upgrade process of designed system is evaluated on test network in last part. Practical test results are compared with OTAU solution for BitCloud.

KEYWORDS

Over the air firmware upgrade, wireless sensor networks (WSN), Lighweight mesh, deRFnode platform, Atmel AVR

MINÁR, Tomáš *Návrh systému pro vzdálený upgrade firmwaru pro uzly bezdrátové senzorové sítě*: diplomová práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 54 s. Vedúci práce bol Ing. Ľubomír Mráz,

PREHLÁSENIE

Prehlasujem, že som svoju diplomovoú prácu na tému „Návrh systému pro vzdálený upgrade firmwaru pro uzly bezdrátové senzorové sítě“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/nebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., o právu autorském, o právoch súvisejúcich s právom autorským a o zmeně niektorých zákonov (autorský zákon), vo znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhé, hlavy VI. diel 4 Trestného zákoníka č. 40/2009 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Ľubomírovi Mrázovi za odborné vedenie, konzultácie a bezproblémovú komunikáciu počas vypracovávania diplomovej práce.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Technicka 12, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

POĎAKOVANIE

Výzkum popísaný v tejto diplomovej práci bol realizovaný v laboratóriách podporených z projektu SIX; registračné číslo CZ.1.05/2.1.00/03.0072, operačný program Výzkum a vývoj pro inovace.

Brno ....
(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Bezdrôtová senzorová sieť – Wireless sensor network (WSN) je sieť, ktorá pozostáva z veľkého počtu uzlov s obmedzeným výpočtovým výkonom, ktoré medzi sebou spolupracujú s cieľom plnenia základnej funkcie danej aplikáciou. WSN musí často fungovať po dlhší čas bez fyzického prístupu v prostredí, ktoré sa vyvíja, čo môže viesť k požiadavkám na zmenu chovania celej siete.

Uzly bezdrôtovej senzorovej siete na svoju činnosť potrebujú základné programové vybavenie (firmware) tj. vhodný operačný systém a program. Požadovaný firmware sa naprogramuje do jednotlivých uzlov pred rozostavením siete. Keďže jedným z predpokladov pre bezdrôtovú senzorovú sieť je jej dlhodobé fungovanie, tak po istej dobe môže dôjsť k odhaleniu chyby v kóde, čo znamená preprogramovanie chybného uzlu alebo celej skupiny uzlov daného typu (koncové zariadenie alebo smerovač). Dôvodom pre preprogramovanie nemusí byť len chyba, ale taktiež pridanie funkcionality alebo zmena parametrov siete na základe zistených meraní parametrov siete.

Preprogramovanie uzlov fyzickým pripojením k počítaču je akceptovateľné v prípade jednoduchej domácej senzorovej siete, alebo siete slúžiacej na laboratórne účely. V prípade rozsiahlej siete o veľkom počte uzlov rozmiestnených na veľkej ploche nie je vhodný takýto prístup, a preto vývoj vo WSN sa taktiež zameriava na efektívne riešenie problematiky preprogramovania celej siete. Upgrade na diaľku vzduchom je nevyhnutný pre budúci úspech rozsiahlych WSN sietí, ich samotný vývoj a nasadenie.

1 PREPROGRAMOVANIE UZLOV SIETE

Jednou zo základných funkcií bezdrôtovej senzorovej siete by malo byť preprogramovanie uzlov, tj. možnosť zmeniť firmware uzlov siete za bežného fungovania siete. Zmeny firmwaru sú vo forme kompletného updatu operačného systému, opravy chyby alebo zmeny sieťových prípadne aplikačných parametrov jednotlivých uzlov siete. Hlavnými dôvodmi potreby nahrania nového firmwaru do uzlov sú:

- nemožný prístup k uzlom
- narastajúci počet uzlov v sieti
- zjednodušenie odladovania a testovania pri vývoji
- prispôsobovanie parametrov rozostavenej siete

Preprogramovanie senzorových uzlov pomocou rádiovej komunikácie OTAU (Over the Air Upgrade) je dosiahnuteľné pomocou šírenia nového obrazu kódu a jeho následného získania a uloženia uzlom. Schopnosť prenesenia nového firmwaru do zariadenia tým pádom závisí na dostatku pamäti uzlu. V prípade, že obraz operačného systému nepresahuje polovičku vnútornej operačnej pamäte mikrokontroléru nie je potrebná externá flash pamäť. Často však už senzorové uzly disponujú externou flash pripojenou pomocou SPI rozhrania.

1.1 Požiadavky na preprogramovanie

Základnými funkcionálnymi požiadavkami sú:

1. Spoľahlivý prenos updatu k cieľu
2. Doručenie updatu skupine alebo vybraným uzlom
3. Nahranie novej verzie do bežiaceho systému

Z pohľadu záťaže na samotnú sieť vyplývajú ďalšie požiadavky: nenarušenie bežného fungovanie siete, malé režijné náklady, minimalizovanie vplyvu na životnosť WSN, minimalizovanie rádiového prenosu z dôvody šetrenia energie, krátke prerušenie chodu aplikácie počas updatu, zabezpečenie rýchlej propagácie, škálovateľnosť pre rozsiahle a husté siete, vysporiadanie sa s asymetrickými spojmi a stratou konektivity.

Veľmi vplyvným požiadavkom je spôsob realizácie updatu z pohľadu objemu prenášaných dát sieťov, tj. nahranie kompletného obrazu alebo len časti kódu. Pri čiastkovom update je možné zmeniť napríklad len pár parametrov, pridať funkciu, bez nutnosti nahrania celého obrazu systému, z čoho vyplývajú menšie náklady na prenos sieťou a taktiež zavedenie týchto zmien bez nutnosti reštartovať uzol, ak to daný operačný systém dovolí. V ideálnom prípade je taktiež chcené nahrať firmware nezávisle na použitej platforme a hardvérovej špecifikácie použitých uzlov. Všetky požiadavky naraz nie je jednoduché dosiahnuť a preto sa implementácia zameriava na uspokojenie tých požiadavkov, ktoré sú nevyhnutné pre danú aplikáciu.

Ďalším dôležitým aspektom aktualizovania firmwaru je bezpečnosť. V tomto prípade chceme, aby update nemohol realizovať prípadný útočník, čím by kompromitoval a znefunkčnil našu sieť. Potrebné je zabezpečiť integritu prenesených dát a autentičnosť pôvodu.

Vzdialený update softvéru je jednou z kľúčových funkcií pre ďalší rozmach WSN, pretože uľahčuje rozostavenie a nastavenie siete v prvotnom štádiu rozmiestnenia a taktiež predstavuje dôležitý nástroj pre správu.[3]

1.2 Základná architektúra

Základnú architektúru tvorí OTAU server a OTAU klient. Úlohou servera je iniciovať preprogramovanie konkrétneho uzlu alebo viacerých uzlov v sieti. OTAU server je často pripojený k počítaču a komunikuje s obslužným softvérom pre preprogramovanie firmwaru uzlov senzorovej siete. Ďalšou možnou variantou je, že server je pripojený do Internetu, a je tak ovládaný na diaľku. Funkcia OTAU serveru nemusí byť plnená len samostatným zariadením, ale je možné ju zahrnúť do hlavného uzla siete – koordinátora.

Každé zariadenie schopné upgradu musí implementovať službu klienta, ktorá zabezpečuje komunikáciu so serverom, uloženie nového firmwaru a jeho iniciovanie. Pre zavedenie novej verzie a reštart zariadenia musí byť súčasťou zariadenia zavádzač obrazu operačného systému nazývaný Bootloader.

K preprogramovaniu senzorovej siete môžeme pristupovať dvoma spôsobmi. Prvým jednoduchým scenárom je, že obsluha do siete pripojí uzol s úlohou OTAU servera. Následne vykoná upgrade požadovaných uzlov alebo všetkých koncových zariadení. OTAU server obsluha odpojí a tak nezostáva súčasťou siete. V prípade druhého scenáru OTAU server je trvalou súčasťou siete a koncové zariadenia vedia o jeho prítomnosti v sieti. Tento scenár umožňuje periodickú kontrolu nového firmwaru zo strany koncového uzla, ktorý sa pýta servera, či nie potrebné vykonať zmenu v kóde. Zrejmou nevýhodou je pridanie režijných nákladov do siete.

1.3 Prístup ako preprogramovať uzly

V zásade sa naskytajú viaceré možnosti ako preprogramovať uzly od druhu zvolenej komunikácie. Prenos firmwaru je možné realizovať týmito spôsobmi:

- Singlehop
- Multihop
- Broadcast flooding

Pomocou single hop komunikácie sa firmware prenesie len do uzlov, ktoré sú v rádiovom dosahu OTAU servera, príkladom protokolu pracujúcom na tomto princípe je XNP [2], ktorý je implementovaný v TinyOS. Takýto protokol neovplyvňuje celú sieť, ale len zúčastnený uzol, ktorý počas upgradu dočasne prestáva plniť svoju funkciu.

V druhom prípade protokol pre OTAU dokáže preniesť firmware cez celú sieť pomocou viac-skokovej komunikácie využívajúcej smerovacieho protokolu.

Najbežnejšou metódou distribuovania súboru firmwaru je pomocou záplavového rozosielania do celej siete. Táto metóda je nákladnou pretože každý uzol po obdržaní súboru rozosiela ďalej prijatý súbor a môže dochádzať k viacnásobnému prijatiu toho istého súboru, kvôli prekrývaniu vše-smerových zón.

2 KATEGORIZÁCIA PROBLÉMU UPDATU SOFTVÉRU

Úlohy potrebné na riešenie updatu softvéru je možné rozdeliť do troch všeobecných kategórií WSN: príprava súboru, rozoslanie sieťou a zavedenie zmeny v senzorovom uzle.

2.1 Príprava súboru pre update

Skompilovaný obraz použitého softvéru v senzorových uzloch ide poslať aj bez akejkoľvek úpravy. Vhodnejšie je však daný súbor upraviť, tak aby sme znížili potrebnú energiu na prenos, oneskorenie a tým aj celkový čas procesu updatu. Preto potrebujeme zredukovať veľkosť prenášaného súboru sieťou.

Naskytajú sa dve metódy kompresie. Prvá metóda aplikuje kompresných algoritmus na celý súbor. Druhá metóda sa zakladá na fakte, že pri update meníme len časť kódu, takže majoritná časť novej verzie oproti predošlej ostáva rovnaká. Z tohto dôvodu stačí preniesť len rozdiel medzi verziami, čo sa v anglickej literatúre nazýva inkrementálny update a používa rozdielových kódovacích algoritmov, ktorých výsledkom je tzv. delta kód. Na vykonanie tejto úlohy potrebujeme na súbor uplatniť rozdielový kódovací algoritmus a kompresiu, ktoré sú vykonané mimo senzorovej siete a tak nenavyšujú požiadavky na senzorovú sieť. Avšak daný súbor je treba v koncovom uzle dekomprimovať a uplatniť rozdielový algoritmus. Bohužiaľ väčšina týchto algoritmov nebola navrhnutá s cieľom použitia v zariadení s limitovanými výpočtovými a pamäťovými prostriedkami aké sa používajú v senzorových uzloch. [3, 4]

Ideálne kompresné a rozdielové kódovanie by malo spĺňať vlastnosti podľa [4]:

- Minimálne energetické nároky
- Minimálny čas na celý proces updatu
- Minimálne výpočetné požiadavky
- Zaplnenie malej časti pamäti ROM kódom na dekompresiu
- Malé využitie pamäte RAM

Aby celý proces redukcie veľkosti súboru malo vôbec význam realizovať, je potrebné splniť aspoň prvé dva požiadavky. Algoritmus, ktorý by spĺňal všetky požiadavky, nie je dostupný, a preto sa treba spokojiť buď s vyššou mierou kompresie a veľkými požiadavkami na výpočtové prostriedky (napríklad BZip2), alebo s menšou mierou kompresie, ale prijateľnými výpočtovými nárokmi (slovníkové kódy).

2.1.1 Odporúčané kompresné algoritmy

Použitie kompresných algoritmov je diskutované v publikácii[4], kde sa autori zamerali na použitie bezstratových algoritmov z dôvodu pracovania s binárnymi spustiteľnými súbormi.

V práci boli testované a vyhodnotené entropické kódovanie (Huffmanov kód), slovníkové kódy (LZ77, LZJB, FastLZ, RLE, Sensor-LZW) a ako referencia bolo použité BZip2. Pred uplatnení kompresie najprv bol uplatnený jeden z rozdielových algoritmov na získanie delta kódu pomocou RDIFF, VCDIFF a BSDIFF. Experimentálne otestovanie bolo realizované na operačnom systéme Contiki bežiacom na TelosB uzloch. V testovaní bola použitá kombinácia kompresných a rozdielových algoritmov, čiže obe techniky, čo malo veľký vplyv na dosiahnuté výsledky.

Z pohľadu mieri kompresie väčšina algoritmov dosiahlo podobných výsledkov až na RLE[1] kódovanie. Z použitým spomínaných algoritmov dosiahli mieru kompresie 37%–99% v závislosti na druhu updatu (update novej verzie OS, čiastkový update). Spomedzi rozdielových algoritmov je vo väčšine prípadov vhodnejší BSDIFF až na scenár zmeny parametru v kóde, kedy VCDIFF produkuje menšiu deltu, ale rozdiel v tomto prípade medzi BSDIFF a VCDIFF nie je taký markantný. Najvhodnejšie je teda použiť BSDIFF a na druhú stranu vôbec neodporúčajú použitie RDIFF algoritmu.

V danej práci[4] bola vykonaná séria testov z pohľadu výpočtových a pamäťových nárokov a oneskorenia. Výsledkom práce je poznatok, že preprogramovanie senzorových uzlov je vylepšiteľné z pohľadu energetickej a časovej náročnosti za použitia kombinácie dátovej kompresie a inkrementálnych updatov. Samotné pridanie dátovej kompresie nevedie k lepším výsledkom.

Výber použitých algoritmov závisí na konkrétnej implementácii. Hlavnými štyrmi faktormi pre výber algoritmu sú dostupné hardvérové prostriedky, druh updatu, topológia siete a druh optimalizácie (spotreba alebo oneskorenie). Odporúčaným riešením je použitie BSDIFF v kombinácii s LZ77 alebo FastLZ, a samotné použitie VCDIFF bez kompresného algoritmu.

2.1.2 Príklad inkrementálneho mechanizmu

Jedným z mechanizmov na preprogramovanie pomocou inkrementálnych updatov ktorý je nezávislý na použitom mikrokontroléry v danej platforme používa Rsync [5] algoritmus na vytvorenie rozdielového skriptu. Veľkou výhodou je nepotreba znalosti programového kódu samotným mechanizmom, z čoho vyplýva jeho univerzálnosť. Pôvodne bol Rsync navrhnutý pre výkonné výpočtové stroje, ktoré si posielajú binárne

[1] Run Length Encoding

updaty cez úzkopásmovú linku. Autori článku [5] upravili a použili daný algoritmus, aby bol použiteľný v TinyOS. Samotné updaty sa posielajú ako rozdielový skript, ktorý špecifikuje kam sa má nemodifikovaný blok prekopírovať v pamäti alebo obsahuje záznam o modifikácii pôvodné obrazu programu. Iniciátor updatu sa pýta koncového uzlu nech mu pošle kontrolný súčet jeho blokov, načo následne mu odpovie poslaním len zmenených blokov. Rozdielové skripty sa šíria sieťou pomocou XNP protokolu a ukladajú sa do externej pamäte. Obrazy operačného systému sú uložené v dvoch sekciách externej pamäte EEPROM, jedna pre starý obraz a druhá pre updatovaný obraz. Po prijatí rozdielových blokov senzorový uzol dekóduje rozdielový skript a zostaví nový obraz programu zo starej verzie pomocou rozdielového skriptu. Následne iniciátor updatu resp. OTAU server odošle príkaz s parametrom pre bootloader, ktorý obraz programu sa má zaviesť do pamäte pri ďalšom reštarte koncového uzla.

2.2 Rozosielanie updatu sieťou

Poslanie updatu sieťou je jedným z ďalších problémov. Jedným z prístupov ako bolo spomenuté v 1.3 je spoľahnúť sa na prostredie operačného systému a princípy, ktoré používa na odosielanie bežných správ. Pre rozsiahle WSN takýto prístup nie je úplne vhodným riešením, pretože kvôli nadbytočnému posielaniu správ s tým istým obsahom dochádza k navýšeniu energetickej náročnosti a taktiež k narušeniu bežného fungovania siete na istú dobu. Riešením tohto problému sú protokoly vhodné na šírenie dát tzv. diseminačné protokoly. Fungujú na následujúcom princípe: zdrojový uzol odošle oznam o novej verzii firmwaru; prijímajúce uzly si vyberú zdrojový uzol pre komunikáciu, keďže oznam v následujúcich krokoch okrem prvého môže prísť od viacerých uzlov; potom následuje spoľahlivý prenos do koncového uzla. Použitím diseminačných protokolov sa úspešne vyhneme niekoľko-skokovému prenosu pomerne veľkého súboru pre senzorovú sieť, pretože koncový uzol sa snaží vybrať ako zdroj blízkeho suseda.

V scenári, kedy obnova firmwaru uzlov nie je iniciovaná administrátorom, ale sieť si automaticky udržuje aktuálnu verziu si uzly medzi sebou vymieňajú meta správy o ich verzii používaného firmwaru. Pri takomto prístupe stačí, aby sa do jedného uzla nahral nový firmware a ten sa následne rozšíri postupne do celej siete.

Podľa [6] nielen samotné šírenie updatu je energeticky nákladné, ale taktiež celková réžia pozostávajúca z informačných a príkazových správ (meta správy). Efektívny algoritmus na zistenie, kedy sa majú posielať updaty by mal spĺňať tieto vlastnosti:

- **Nízke režijné náklady:** Ak je sieť v stabilnom stave, frekvencia režijných správ by mala byť čo najnižšia.

- **Rýchle rozšírenie:** Ak je v sieti uzol/-y so staršou verziou je nutné rýchle rozoslanie. Čas réžie pre jeden prenos by nemal presiahnuť čas prenosu updatu cez singlehop.
- **Škálovateľnosť:** Keďže treba predpokladať so zmenami prostredia, rozšírením stávajúcej siete a stratou uzlov, použitý protokol musí fungovať bez vopred známych parametrov siete.

2.2.1 Trickle

Protokol na šírenie kódu a údržbu vychádza z epidemického šírenia a škálovateľného multikastu. Svoje fungovanie zakladá na šírení meta správ nazývaných „zdvorilé klebety“, pomocou ktorých každý uzol ohlasuje svoju verziu kódu. Keď uzol prijme meta správu, ktorej obsah sa zhoduje s jeho rozosielanou správou, tak sa umlčí. V prípade obdržania starej „klebety“ začne posielať aktuálnu verziu kódu. Trickle si limituje počet rozoslaných meta správ počas jedného intervalu, tak že v prípade stabilného stavu komunikujú uzly, čo najmenej, čo ho robí energeticky efektívnym. Prínosom protokolu Trickle je práve potlačenie nadbytočnej komunikácie a dynamické nastavovanie rýchlosti prenosu. Protokol sa nestará o odozvu a vychádza z predpokladov šírenia malých binárnych súborov prenesiteľných v malom počte paketov. TinyOS a Mate obsahujú implementáciu tohto protokolu.[6, 7]

2.2.2 XNP

Spomínaný protokol XNP v predošlých kapitolách bol východiskovým vzorom pre viacskokové diseminačné protokoly, ktoré sa ho snažia vylepšiť.

Princíp spočíva v prenesení firmwaru z PC aplikácie pomocou uzla, na ktorom beží TOSBase[2] do jedného alebo celej skupiny uzlov v priamom rádiovom dosahu. Firmware sa posiela po častiach, vždy jedna na paket. V prípade unikastu sa kontroluje doručenie pre každú kapsulu. V opačnom prípade broadkastu si uzly posielajú kumulatívnu žiadosť o chýbajúce kapsule, ktoré sa nepodarilo doručiť. Stiahnutý obraz firmwaru sa ukladá do externej pamäte a bežiace aplikácie sú zatiaľ pozdržané. Po obdržaní príkazu na reštart, uzol zavedie nový obraz do operačnej pamäte. XNP nepodporuje inkrementálne updaty, takže vždy sa prenáša celý obraz a samozrejme vyžaduje bootloader v rezervovanej sekcii pamäte, ktorý nie je možné na diaľku obnoviť.[2, 1]

[2]aplikácia základňovej stanice v prostredí TinyOS

2.2.3 MOAP

MOAP (Multihop Over-the-Air Programming) je distribučný mechanizmus vyvinutý práve pre Mica-2 uzly pod operačným systémom TinyOS. Autori technickej správy [8] narvhli mechanizmus, tak aby bol energeticky a pamäťovo efektívny, na úkor navýšenia odozvy. V návrhu sa zamerali na tri oblasti: šírenie updatu sieťou, opakované posielanie chybných častí, pamäťová správa segmentov kódu. Iniciátor updatu rozosiela *publish* správy, ktorými oznamuje novú verziu obrazu. Prijímajúce uzly v prípade neaktuálnej verzie požiadajú o update správou *subscribe*. Mechanizmus zaznamenávajúci štatistiky prenosov medzi uzlami je použitý na vyhnutie sa nespoľahlivých spojov. Po uplynutí čakacieho intervalu a prijatí všetkých *subscribe* správ iniciátor začne prenos dát. Príjemca identifikuje stratené segmenty pomocou posuvného okna a žiada o znovuzaslanie pomocou unikastu, aby sa vyhol duplicitnému posielaniu od viacerých uzlov. Potom aktivuje časovač a v prípade, že sa mu do uplynutia nepodarí obdržať chýbajúce segmenty, tak posiela požiadavok broadkastom. Keď má uzol celý nový obraz, tak sa stáva rozosielateľom. Ak neobdrží žiadne *subscribe* správy od svojich susedov, tak uzol zavedie nový obraz do pamäte a reštartuje sa.

2.2.4 Deluge

Jedným z významných diseminačných protokolov na šírenie veľkých súborov z jedných alebo viacerých zdrojových uzlov do veľa ostatných pomocou viacskokovej komunikácie je Deluge. Tento protokol nie je implementovaný len v prostredí TinyOS, ale aj napríklad v Contiki. Deluge stavia na predchodzom vývoji diseminačných protokol a mechanizmov správy používajúcich epidemického rozosielania. Na rozdiel od Trickle 2.2.1 nerieši len šírenie jedného paketu, ale podporu šírenia veľkých dátových objektov ako sú spomínané celé binárne obrazy operačného systému používaného v danej sieti. Aj keď Trickle v podstate rieši hlavne otázku kedy uzly majú rozosielať nový kód, tak Deluge vychádza priamo z tohto protokolu a rozširuje ho.

V prípade spoľahlivosti doručovania sa inšpiroval protokolmi ako Pomalé pretláčanie sieťou, Rýchle doručovanie – Pump Slowly, Fetch Quickly (PSFQ) a Spoľahlivý viacsegmentový prenos – Reliable Multi-Segment Transport (RMST) používajúcich selektívnych NACK správ v prípade chyby prenosu. Tieto protokoly zdôrazňujú limitovanie opravy chýb na čo najmenej skokov, keďže náklady na opravu koncovej komunikácie závisia od dĺžky cesty sieťou. Deluge si stanovuje limit opravy chyby práve na jeden skok. Ďalej s protokolom MOAP zdiela použitie spomínaných NACK správ, unikástových žiadostí, broadkastu na prenos dát a posuvného okna na kontrolu prenesených segmentov. Zatiaľ čo MOAP nerozdeľuje prenášané dáta na menšie časti a je potrebné preniesť celý súbor pred preposlaním do ďalšieho uzlu, tak Deluge rozdeľuje dátové objekty na zostavu stránok pevnej veľkosti. Toto rozdeľovanie dovoľuje

priestorový multiplex a efektívny inkrementálny update. Deluge charakterizujeme ako protokol s epidemickým šírením, ktorý funguje ako stavový stroj, kde každý uzol prechádza medzi stavmi na základe pevne daných pravidiel, aby sa dosiahlo rýchleho, spoľahlivého rozoslania veľkých dátových objektov veľa uzlom.

Reprezentácia dát Použitie delenia dátových objektov na stránky prináša tieto výhody: malý počet stavov, ktoré musí držať príjemca; efektívne inkrementálne updaty, priestorový multiplex. Každá stránka sa skladá z pevného počtu paketov. Ako pakety tak aj stránky sú zabezpečené 16-bitovým CRC. Takáto redundantná ochrana na dvoch úrovniach navyšuje spoľahlivosť prenosu a predchádza šíreniu chybných dát ďalším uzlom.

Inkrementálne updaty Aby bolo možné realizovať rozdielové updaty Deluge vedie objektové záznamy s informáciou o verzi (`v`) a vekovom vektore (`a`), ktoré slúžia na určenie potrebných stránok pre update. Objekt je kompletne popísaný objektovým profilom skladajúcim sa z dvojice (`v,a`) Vekový vektor použije prijímajúci uzol na určenie potrebných stránok na prenos. Uzly pozadu o 16 verzií ,bez ohľadu na ich vek, vyžadujú prenos všetkých stránok objektu.

Každý uzol používa sadu pravidiel a nachádza sa v jednom z troch stavov: `MAINTAIN`, `RX`, `TX`. Lokálne pravidlá určujú, ktoré príkazy sa realizujú a do akého stavu sa prejde na základe udalostí.

MAINTAIN stav Uzol v tomto stave zabezpečuje, že všetky uzly v jeho vysielacom dosahu majú najnovšiu verziu objektového profil a všetky potrebné dáta pre novú verziu kódu. Nepravidelne na základe náhodného parametru oznamuje uzol sumár reprezentujúci aktuálnu verziu objektového profilu $\{\texttt{v},\gamma\}$[3] a súbor stránok objektu, ktoré sú pripravené na prenos. Deluge používa Trickle na kontrolu nadbytočných správ. Trickle v tejto implementácii používa delenie času na kolá (i), počas ktorých uzly vysielajú oznamy alebo niektoré kolá mlčia. Trvanie kola je špecifikované $\tau_{m,i}$ a je ohraničené τ_l a τ_h. V každom kole uzol používa náhodnú hodnotu r_i z rozsah: $[\frac{\tau_{m,i}}{2}, \tau_{m,i}]$. Lokálne pravidlá pre uzly v tomto stave:

- V jednom kole sú vysielané oznamy o objektovom profile v čase $t_i + r_i$, kde t_i je začiatok kola. Oznamy sa posielajú iba ak bol prijatý menší počet oznamov ako k s rovnakým $\{\texttt{v},\gamma\}$ ako uzol posiela.
- Začne sa nové kolo, po zistení rozdielov v objektovom profile medzi susednými uzlami a $\tau_{m,i}$ sa nastaví na τ_l
- Ak neboli zistené rozdielne verzie medzi susedmi, tak pre ďalšie kolo sa zvýši interval jeho trvania na $min(2 \times \tau_{m,i-1}, \tau_h)$

[3] γ je najvyššie číslo stránky pripravenej na odoslanie

- Pre verziu v v čase $t_i + r_i$ uzol posiela objektový profil ak prijal oznamy so staršou verziu v čase $t \geq t_i$ a zároveň ak zaznamenal menej ako k pokusov o update objektu.

Hraničné k limituje počet poslaných oznamov v danej bunke alebo oblasti, čím sa znižuje počet nadbytočných oznamov. Táto metóda obnovovania objektového profilu je formou kontrolovaného záplavového rozosielania, ktorá je spoľahlivá a vedomá hustoty rozmiestnených uzlov. Uzol následne, ak neobdržal žiadosť o dostupné stránky od susedov, tak po prijatí oznamu s $\gamma' > \gamma$ prechádza do stavu žiadostí (RX).

RX stav Uzol aktívne posiela žiadosti o zostávajúce pakety na skompletovanie stránky pomocou selektívnych NACK správ. Pre minimalizovanie kolízií so žiadosťami ostatných uzlov sa použije náhodný časovač pre odstup od média. Ak počas tohto intervalu zachytí žiadosti iného uzlu o rovnaké pakety, tak čaká na ich poslanie bez odoslania vlastnej žiadosti. Pokiaľ dôjde ku strate paketov, tak rozosiela znovu žiadosti, v čase ticha. Lokálne pravidlá pre RX stav sú:

- Pošli žiadosť oznamujúcemu uzlu, ak neboli prijaté žiadosti alebo dáta v špecifikovanom časovom intervale
- Prejdi do stavu MAINTAIN, ak bolo dosiahnuté limitnej hodnoty k žiadostí, aj napriek tomu že nebola obdržaná celá stránka
- Prejdi do stavu MAINTAIN po spoľahlivom prijatí všetkých paketov pre danú stránku

Deluge využíva výhody broadkastového média tým spôsobom, že ak zachytí chýbajúce pakety v akomkoľvek stave, tak ich uloží. Z toho vyplýva, že uzol nemusí byť nevyhnutne v RX stave na prijímanie paketov.

TX stav V tomto stave je uzol zodpovedný za rozoslanie vyžiadaných paketov pre danú stránku. Deluge z prijatých žiadostí tvorí prienik nových žiadostí a tých predošlých, ktoré ešte neboli obslúžené, aby sa rovnaké pakety neposielali viackrát. Pakety na odoslanie sú mapované round-robin algoritmov pre dosiahnutie rovnoprávnosti medzi žiadateľmi. Lokálne pravidlá pre TX stav sú:

- Tvorenie prieniku žiadostí paketov pre stránku p
- Pokiaľ všetky vyžiadané pakety nie sú rozoslané, nie je možné prejsť do stavu MAINTAIN

Deluge kladie dôraz na priestorový multiplex, kedy uzol oznamuje dostupnosť stránky ešte pred obdržaním všetkých stránok objektového profilu, čím je navýšená celková priepustnosť. Pre efektívny priestorový multiplex je potrebná trojskoková medzera, pretože simultálne vysielanie v dvojskokovej sieti môže spôsobiť kolíziu. Aby sa predišlo súperenie uzlov o rozdielne stránky v jednej oblasti, tak Deluge

obmedzuje uzly na žiadosti stránok v poradí, kde prenos starších stránok má väčšiu prioritu. [9]

V následujúcej tabuľke je uvedené súhrnné porovnanie rozoberaných protokolov na základe troch vlastností.

Tab. 2.1: Vlastnosti protokolov

Protokol	**Režijné náklady**	**Šírenie do uzlov**	**Škálovateľnosť**
Trickle	Nízke	Multihop	Áno
XNP	Nízke	Singlehop	Nie
MOAP	Stredné	Multihop	Áno
Deluge	Vysoké	Multihop	Áno
Vlastný návrh	Nízke	Singlehop [4]	Áno

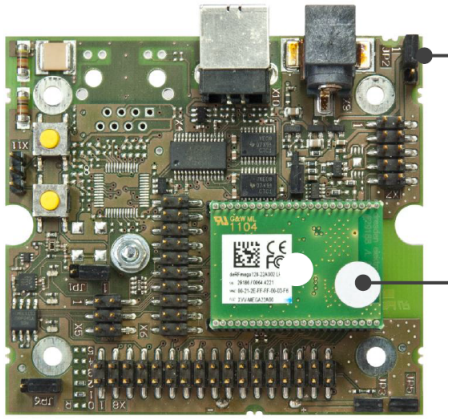
[4]Singlehop v zmysle prenosu od uzla s úlohou servera k susedným klientom v priamom rádiovom dosahu. Uzly siete s dostupným firmwarom sa stávajú serverom.

3 PREPROGRAMOVANIE UZLOV V LIGHTWEIGHT MESH

Táto kapitola je venovaná praktickej realizácii preprogramovania uzlu vo WSN na disponibilnom hardvéri deRFnode s rádiovým modulom deRFmega128. Zameriava sa na použitie Lightweight Mesh protokolu pre mesh senzorové siete na danom hardvéri a realizáciu updatu tohto systému.

3.1 Platforma deRFnode

Platforma navrhnutá a vyrobená dresden elektronik je určená na vývoj a použitie vo WSN. Predstavuje základovú dosku pre rádiové moduly s AVR mikroprocesormi alebo ARM procesormi. Doska bola navrhnutá pre nízku spotrebu. Vlastnosti a periférie platformy sú zhrnuté v tabuľke 3.1.[10]



Obr. 3.1: deRFnode so zasunutým deRFmega128 [10]

Tab. 3.1: Vlastnosti deRFnode

deRFnode	
Napájanie	5V USB/DC alebo 3 x AA
Rozhrania	JTAG, UART, USB, I2C, SPI, ADC
Senzory	Teploty, Akcelerácie, Osvetlenia
Prídavná pamäť	4 Mbit Serial Flash
Zasuvné rádiové moduly	deRFarm7, deRFmega128
Periférie	3 x LED; 2 x tlačítko

Rádiové moduly použité s deRFnode sú deRFmega128, ktorý je vidieť na obrázku 3.1, postavené na ATmega128RFA1 mikrokontroléri, ktorý je podporovaný v prostredí LightWeight Mesh. Daný modul v sebe zahŕňa mikrokontrolér, chipovú keramickú anténu s vysielacím výkonom 2,4 dBm a pridanú pamäť FLASH o veľkosti 128 kB. [11] Samotný ATmega128RFA1 má k dispozícii 128 kB programovej pamäte a 4kB EEPROM.

3.2 Atmel LightWeight Mesh (LWM)

LightWeight Mesh softvérový stack je vyvíjaný pod záštitou Atmelu a prvá verzia 1.0.0 bola publikovaná v septembri 2012, čiže sa jedná o veľmi mladý stack pre WSN, ktorý podlieha intenzívnemu vývoju. Prvotné testovanie bolo realizované vo verzii 1.1.0 z mája 2013. Po odhalení bugu v buffri pre odosielané správy, kde v prípade zaplnenia bufru ukazovateľ poslednej správy ukazoval mimo buffer boli vyvíjané aplikácie premigrované do verzie 1.1.1 bez tohto problému.

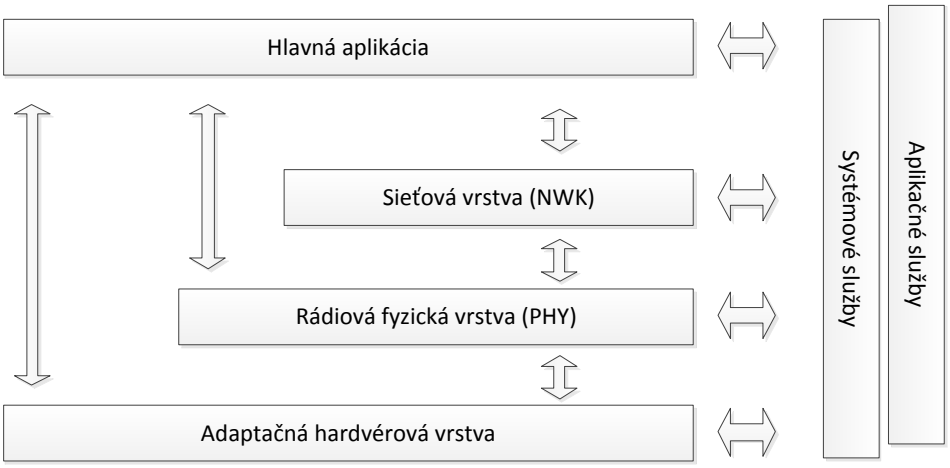
LWM bol navrhnutý pre IEEE 802.15.4 kompatibilné Atmel rádiové čipy a beží na AVR mikrokontroléroch, ale je ho možné prispôsobiť pre akýkoľvek MCU od Atmelu. Veľmi dôležitým aspektom je, že tento operačný systém/protokolová sada bola vyvinutá s cieľom nízkych požiadavkov na hardvérové zdroje a jednoduchosť implementácie bez zbytočných režijných nákladov na vytvorenie WSN a komunikácie medzi uzlami. Nízku energetickú náročnosť a efektívne využívanie zdrojov napájanie dosahuje pomocou nízkej réžie v správach a efektívneho prístupu k zdielanému médiu. Typické pamäťové nároky sa pohybujú okolo 8 kB pre flash a 4 kB pre RAM ako uvádza dokumentácia [12]. Základné vlastnosti a odlišnosti oproti štandardizovanej implementácii ZigBee Pro staku v BitCloud:

- Na zriadenie siete nie je potrebný dedikovaný uzol.
- Žiadne periodické správy údržby alebo manažmentu medzi uzlami
- Dva druhy uzlov: so smerovaním a bez smerovanie. Odlišujú sa na základe sieťovej adresy menšej ako 0x8000 pre smerovacie uzly a väčšej alebo rovnej ako 0x8000 pre uzly bez funkcie smerovania.
- LWM nepoužívaj žiadnu procedúru pripojenia do existujúcej PAN, čo znamená, že uzly hneď po štarte posielajú a prijímajú správy. Z tohto vyplýva rýchle naštartovanie siete.
- Neexistujú vzťahy otec a syn medzi uzlami
- Uzly bez smerovania posielajú a prijímajú správy od všetkých ostatných uzlov, nie len od smerovačov
- Ak uzol nepozná cestu k cieľu, tak automaticky vykonáva objavovanie cesty

- Smerovacia tabuľka sa obnovuje automaticky na základe adries v prijatých alebo preposielaných rámcoch
- LWM aktuálne podporuje Native smerovanie – originálny algoritmus v LWM a AODV štandardizované smerovanie

3.2.1 Architektúra LWM

LightWeight Mesh sadu je možné rozdeliť do logických vrstiev na základe služieb, ktoré je potrebné zabezpečiť na jednotlivých úrovniach. Horizontálne delenie taktiež zodpovedá rozdeleniu jednotlivých zdrojových kódov a knižníc v rámci LWM. Architektúra LWM je navrhnutá, tak aby zabezpečovala základné funkcie potrebné pre bezdrôtovú komunikáciu. Predpokladá vytvorenie špecifických služieb a funkcií treťou stranou. Z pohľadu OTAU nás hlavne zaujímajú systémové a aplikačné služby.



Obr. 3.2: Logické vrstvy LWM architektúry

Systémové služby poskytujú všetkým vrstvám základné funkcie potrebné pre chod systému, ako sú napríklad základné dátové typy, softvérové časovače, konfigurácia parametrov siete, prístup ku šifrovaciemu modulu a iným.

Na druhú stranu aplikačné služby zahŕňajú rozširovacie moduly, ktoré nie sú požadované na chod systému. Momentálne jedinou službou pracujúcou na tejto úrovni je práve OTAU, zabezpečujúca preprogramovanie. [1]

[1] V poslednej verzii 1.2.0 z marca z roku 2014 vývoj LWM už nie je pod správou Alexeja Taradova a pripravovaná OTAU služba nebola zahrnutá v tejto verzii

3.2.2 Bootloader

Bootloader predstavuje najzákladnejší softvér pre mikrokontrolér, ktorého úloha je zavedenie hlavného programu, v našom prípade softvérový stack LWM. Pre samotný bootloader sa na začiatku pamäte vyčlenia 2 kB. Zvyšok pamäte sa rozdelí na dve časti, kde do prvej sa nahrá prvotný LWM obraz pomocou sériovej linky z počítača a druhý blok pamäte je rezervovaný pre uloženie novej verzie LWM pri realizácii updatu. Keďže LWM vyžaduje v základe bez rozšírení okolo 8 kB pamäte, tak pamäťový priestor ATmega128RFA1 plne postačuje pre OTAU bez použitia externej prídavnej pamäte. Rozdelenie pamäťových blokov je znázornené na obrázku 3.3 .

Obr. 3.3: Flash pamäť 128kB ATmega128RFA1

V tejto časti kódu sú hlavne funkcie na zápis a pracovanie s pamäťou flash[2], ktoré sú pri update volané z hlavnej bežiacej aplikácie. Počas preberania nového obrazu systému sa zapisujú dáta do druhej polovice flash pamäte po stránkach veľkosti 256 bajtov. Po prijatý pod kontrolou bežiacej aplikácie je obraz presunutý do bloku pamäte hlavnej aplikácie a po reštarte je následne zavedená nová verzia softvéru.

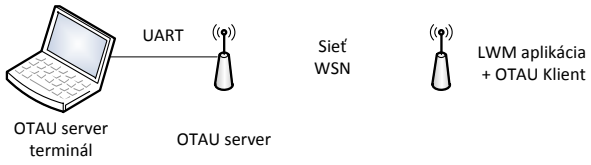
Bootloader bol upravený tak, aby bol použiteľný s deRFnode platformou pripojenou k počítaču cez sériovú linku pripojenú na port `UART0` použitého mikrokontroléru s nastavenými parametrami: 38400 Baud 8N2 (8 dátových bitov, bez parity, 2 stop bity). Pri nahraní bootloader do zariadenia pomocou JTAG programátoru je treba

[2]Zamykacie bity pre sekcie pamäte dovoľujú zápis do programovej flash časti len inštrukciám z boot sekcie

nastaviť fuses na tieto hodnoty: `EXTENDED=0xFE`; `HIGH=0x14`; `LOW=0xE2`. Prvotný skompilovaný binárny obraz systému je potrebné nahrať za použitia Xmodem protokolu napríklad pomocou aplikácie teraterm. Dôležité je ešte spomenúť, že bootloader aplikácia nie je po štarte v režime prijímania dát cez sériovú linku, ale treba ju prepnúť do bootloader režimu pomocou stlačenia tlačítka na deRFnode doske označený ako SW1. Po úspešnom nahraní sa naštartuje LWM s hlavnou aplikáciou.

3.3 Architektúra OTAU v LWM

Základná architektúra OTAU pozostáva z troch komponent: OTAU server, Uzol s OTAU klientom a obslužná aplikácia v počítači. Prepojenie jednotlivých komponent je znázornené na obrázku 3.4



Obr. 3.4: OTAU komponenty

3.3.1 OTAU terminál

Jedna z komponent architektúry, ktorá zohráva dôležitú úlohu v iniciovaní celého procesu prenosu nového firmwaru do uzlov. OTAU terminál zabezpečuje komunikáciu cez UART s OTAU serverom, prenos nového binárneho súboru rozdeleného na bloky a overenie úspešného prenosu. Aplikácia resp. skript je napísaný v jazyku Python využívajúca knižnicu pre komunikáciu cez sériový port.

Pred samotnou realizáciou aktualizácie firmwaru potrebujeme: skompilovaný celý LWM stack spolu s aplikáciou v binárnom formáte, úspešnú komunikáciu s OTAU serverom pripojeným do fungujúcej WSN siete. Použitie pomocou príkazovej riadky je uvedené vo výpise 3.1.

Príklad úspešnej aktualizácie a výmeny správ medzi OTAU serverom a OTAU terminálom je v následujúcom výpise 3.2. Pre prichádzajúce správy z OTAU serveru je stanovená doba čakania na odpoveď. Ak terminál neobdrží žiadnu správ do tejto doby, tak sa skript ukončí s chybovou hláškou „response timeout“. Odladovacie správy slúžia na odhalenie chyby, napríklad v prípade, že OTAU server stratil spojenie s koncovým uzlom alebo uzol neodpovedá na opakované výzvy do stanovenej doby.

```
python otaServerDemo.py [nastavenia] firmware.bin

-d, --zapnutie odladovacích správ
-p PORT, --zvolenie sériového portu [východzí COM1]
-b BAUD, --nastavenie modulačnej rýchlosti sériovej linky [38400]
-m, --overenie komunikácie s OTAU serverom bez updatu
-i PANID, --nastavenie PAN ID identifikátora siete [0x1234]
-c CHANNEL, --použitý rádiový kanál [15]
-a ADDR, --ADDR adresa serveru [0x8555]
-t CLIENT, --sieťová adresa koncového uzlu [0x0010]
```

Výpis 3.1: Voľby z OTAU terminálu

Počas aktualizácie, taktiež môže dôjsť k prerušenie prenosu blokov a skript sa ukončí. Toto má za dôsledok nutnosti opakovania procesu aktualizácie od začiatku bez ohľadu na počet úspešne prenesených blokov nového obrazu.

```
python otaServerDemo.py -d -p COM4 -a 0x8555 -t 0x0010 WSNupdate.bin

Port : COM4
Baudrate : 38400
Server : 0x8555
Client : 0x0010
PAN ID : 0x1234
Channel : 15 (0x0f)
Firmware : WSNupdate.bin
Firmware size: 9442 bytes
-> UART_COMMAND_COMM_CHECK
<- APP_UART_STATUS_CONFIRMATION
<- APP_UART_STATUS_PASSED
* Comm check passed
-> UART_COMMAND_START_REQUEST
<- APP_UART_STATUS_CONFIRMATION
<- OTA_CLIENT_READY_STATUS
* Upgrade started
-> UART_COMMAND_BLOCK_REQUEST
<- APP_UART_STATUS_CONFIRMATION
<- OTA_CLIENT_READY_STATUS
* Block sent
... PRENOS BLOKOV ...
-> UART_COMMAND_BLOCK_REQUEST
<- APP_UART_STATUS_CONFIRMATION
<- OTA_UPGRADE_COMPLETED_STATUS
* Block sent
* File sent
Time 10.5 sec
```

Výpis 3.2: Indikácia výmeny správ

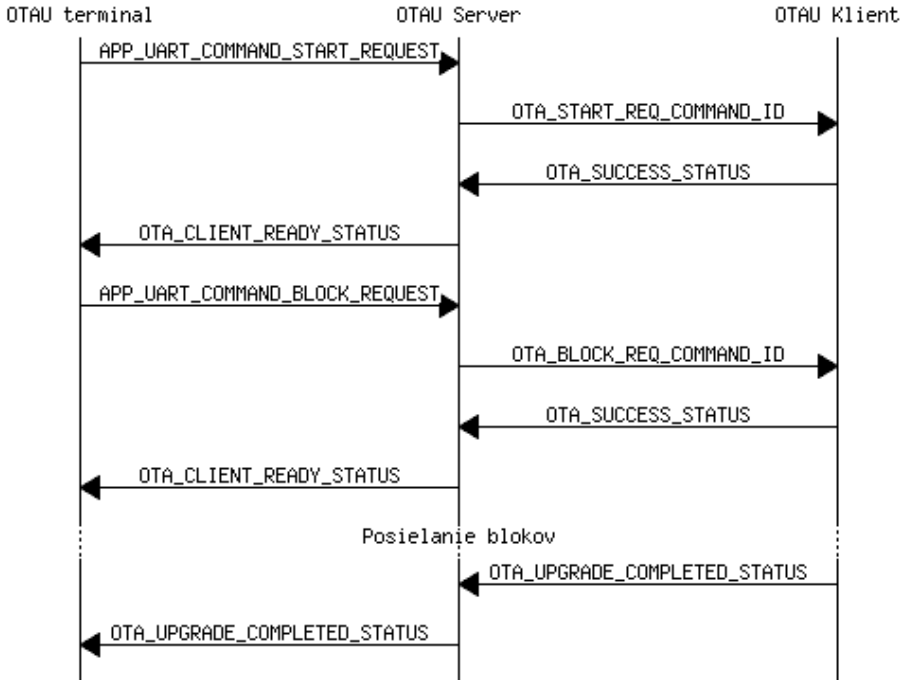
Terminál prenáša binárny obraz do OTAU serveru po blokoch pevne stanovenej maximálnej veľkosti, ktoré sú následne posielané do koncového uzla. Takže v prípade veľkej chybovosti pri prenose medzi OTAU serverom a aktualizovaným uzlom je jedným riešením problému nastavenie menšej veľkosti prenášaných blokov. Každý jeden

prijatý dátový blok OTAU klientom musí byť skontrolovaný voči chybám pri prenose pomocou CRC a potvrdený ACK správou. Prípadné zmenšenie veľkosti prenášaných blokov vedie k celkovému nárastu prenášaných správ do a z aktualizovaného uzla.

3.3.2 OTAU server

Tento prvok OTAU architektúry v LWM predstavuje pasívneho člena/uzol danej WSN siete identifikovanej pomocou PAN ID v čase bežného fungovania siete. Počas procesu aktualizácie sa stáva aktívnym prvkom, ktorý funguje ako prostredník alebo brána medzi OTAU terminálom a koncovým uzlom danej siete na prenos nového obrazu operačného systému.

OTAU server komunikuje s terminálom cez sériovú linku a prijíma od neho príkazy na začatie aktualizácie (`UART_COMMAND_START_REQUEST`), na prenos dátového bloku (`UART_COMMAND_BLOCK_REQUEST`) a samotný binárny obraz rozdelený na časti. Na druhú stranu do terminálu posiela stavové správy o priebehu aktualizácie. Súslednosť výmeny správ ,medzi tromi zúčastnenými komponentmi - fáza iniciovania a úspešného prenosu všetkých blokov sú znázornené v Grafe 3.5. Komunikácia medzi serverom



Obr. 3.5: Prenášané správy

a OTAU klientom a terminálom je postavená na báze žiadostí a odpovedí. Žiadosť o začatie aktualizácie (`START`) a prenos bloku (`BLOCK`) sú identifikované v hlavičke správ na mieste hviezdičky celého identifikátora tvaru `OTA_*_REQ_COMMAND_ID`.

Identifikátory a stavy sa prenášajú ako príslušné priradené hexadecimálne hodnoty. Na základe tohto delenia žiadostí na fázu iniciovania aktualizácie (`START`) a samotného prenosu blokov (`BLOCK`) sú rozdelené aj metódy, ktoré majú na starosti príslušnú fázu. Potvrdzovacie resp. stavové správy od klienta sú identifikované rovnakým spôsobom v identifikátore odpovede `OTA_*_RESP_COMMAND_ID`, ktoré naviac nesú informáciu o stave. Napríklad kladná potvrdzovacia správa nesie stav `OTA_SUCCESS_STATUS`.

V konfiguračnom súbore servera je možné nastaviť dobu čakania na odpoveď, východziu veľkosť blokov a počet opakovaní správ. Ak klient neodpovie do stanovenej čakacej doby, tak vo východzom nastavení sa pokúsi server poslať správu maximálne trikrát. Po neúspešnom poslaní žiadosti server informuje terminál správou `OTA_NO_RESPONSE_STATUS` a prechádza do východzieho stavu `IDLE`.

3.3.3 OTAU klient

Poslednú časť celej architektúry resp. systému pre aktualizáciu firmwaru tvorí práve služba bežiaca na strane rozostavených uzlov v senzorovej sieti. Koncový uzol bežne prechádza stavmi na plnenie funkcie hlavnej bežiacej aplikácie. Zároveň pri každom prechode hlavného cyklu aplikácie sa kontroluje stav OTA služby. Pri obdržaní správy od OTAU servera o aktualizáciu prechádza uzol do režimu preberania firmwaru, počas ktorého prestáva plniť svoju hlavnú funkciu v rámci senzorovej siete. Po dokončení aktualizácie sa uzol znovu stáva aktívnym plnohodnotným uzlom danej siete.

Komunikácia s OTAU serverom prebieha v dvoch fázach iniciovania aktualizácie a prenosu blokov firmwaru, tak ako bolo spomenuté v časti o serveri. Pre klienta je dôležité, aby všetky bloky boli doručené bez chyby, preto pre každý blok počíta CRC z prijatých dát a porovnáva s CRC obsiahnutým v správe. V prípade nezhody posiela správu typu `OTA_BLOCK_RESP_COMMAND_ID` s obsiahnutým statusom `OTA_CRC_ERROR_STATUS`, čím zároveň žiada opätovné zaslanie. V opačnom prípade dochádza k zápisu blok po bloku do druhej polovice rozdelenej pamäte a zaslaniu správy rovnakého typu so statusom `OTA_SUCCESS_STATUS`. Chybný prenos server rieši opakovaným poslaním po obdržaní chybného statusu. Opakovanie vykoná server maximálne toľkokrát ako má nastavenú hodnotu `OTA_MAX_RETRIES`. Po dosiahnutí tejto hodnoty sa celý proces aktualizácie ukončí a terminál je informovaný správou `OTA_NO_RESPONSE_STATUS` – žiadnej odozvy od OTAU klienta. Na konci úspešného prijatia všetkých blokov okrem poslania správy `OTA_UPGRADE_COMPLETED_STATUS` sa následné zavolá metóda appSwitch na prekopírovanie nového firmwaru z druhého bloku pamäte na začiatok pamäte za pomoci metódy `iap_switch_handler` z Bootloaderu.

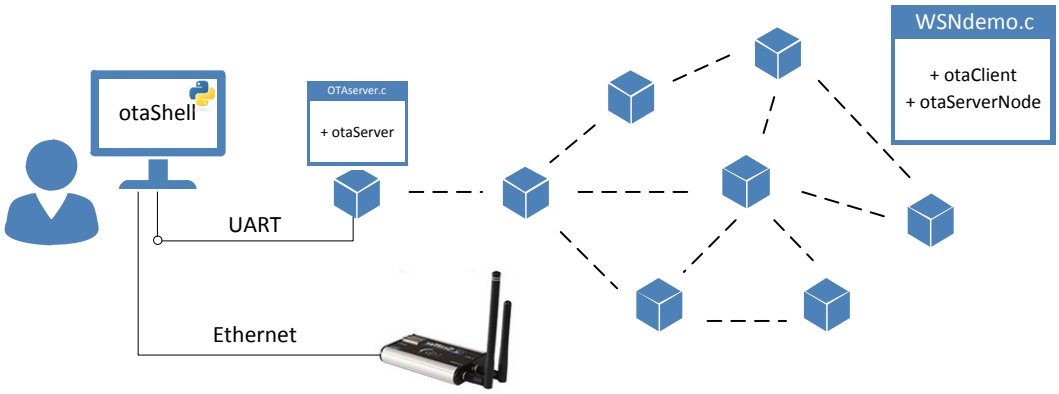
Konkrétna zrealizovaná aktualizácia uvedená vo výpise 3.2 bola analyzovaná pomocou sieťového a protokolového analyzátoru Perytons pre štandard IEEE 802.15.4 za použitia Perytons deRFusb-23E00, ktorý je súčasťou vývojového balíka deRF ZigBee 2,4 GHz. Celkový čas na prenesenie firmwaru o veľkosti 9442 bajtov bol 10,5 sekundy. Tento čas bol odčítaný z údajov prvej a poslednej správy procesu aktualizácie v spomínanom analyzátore. Okrem rozoberaných správ sa každá správa poslaná uzlami potvrdzuje 5 bajtovým ACK na úrovni sieťovej vrstvy LWM na Obrázku 3.2, keďže sa jedná o unikastové správy.

4 NÁVRH ARCHITEKTÚRY

Táto kapitola sa zaoberá návrhom systému prenosu a zavedenia nového firmwaru do celej siete používajúcej Lightweight Mesh vo verzii 1.1.1. Cieľom návrhu je zaktualizovať sieť v čo najkratšom čase a s nízkou réžiou medzi uzlami, tak aby nebola vyťažovaná sieť po celú dobu aktualizácie, ale len zainteresované uzly, u ktorých práve dochádza k prenosu. Jednotlivé podkapitoly sú venované popisu kľúčových funkcií potrebných OTAU komponentov a nie celých aplikácii z dôvodu úspory. Zdrojové kódy všetkých aplikácii a použitého LWM sú priložené k práci v digitálnej forme.

4.1 Architektúra

Skladá sa z troch aplikácií: otaShell[1], OTAU server, koncová aplikácia senozorového uzlu doplnená o služby otaServerNode a otaClient. Všebecné súvislosti medzi jednotlivými aplikáciami a na akom zariadení bežia sú znázornené na následujúcom diagrame 4.1. V tomto diagrame je zobrazený aj Open Sniffer pre IEEE 802.15.4 [13], ktorý bol použití pri vývoji a analýze komunikácie v prostredí Wireshark. Súčasťou senzorovej siete zobrazenej pomocou kociek je aj koordinátor siete, ktorého funkcionalita týmto návrhom nie je ovplyvňovaná. V podstate ide o hardvérovo totožný uzol ako ostatné v sieti s aplikáciou WSNdemo, ktorá na základe adresy 0x0000 vykonáva iné funkcie a nevyužíva služieb otaClient a otaServerNode.



Obr. 4.1: Vizualizácia systému

[1]konzolová aplikácia v pythone bežiaca na obslužnom PC

4.1.1 Princíp systému

V prvotnej fáze otaShell na základe pozbieraných tabuliek susedov si zostaví graf siete. Z tohto grafu vie, ktoré uzly sú v priamom dosahu OTAU server uzla a začne do nich postupne prenášať firmware. Tento prenos je realizovaný postupne do susedných uzlov formou unikastovej komunikácie a nie naraz.

Akonáhle skončí prenos k prvému susedovi, obdrží tento sused príkaz na prenos do svojich susedných uzlov. Od tohto okamihu dochádza k simultánnemu prenosu medzi viacerými susednými uzlami. Ak sú v dosahu OTAU servera neaktualizované uzly, tak prebieha ďalší prenos medzi OTAU serverom a uzlom. Dôležité je poznamenať, že firmware sa vždy prenáša po blokoch medzi dvoma uzlami v priamom dosahu bez použitia smerovania.

Susedné uzly k OTAU serveru aktualizujú svojich susedov na základe príkazu na iniciovanie obdržaného z otaShell aplikácie. Uzly, ktoré sa nachádzajú na vzdialenosť 3 skokov od OTAU servera sú aktualizované na základe komunikácie so svojimi susednými uzlami bez kontroly otaShell aplikácie.

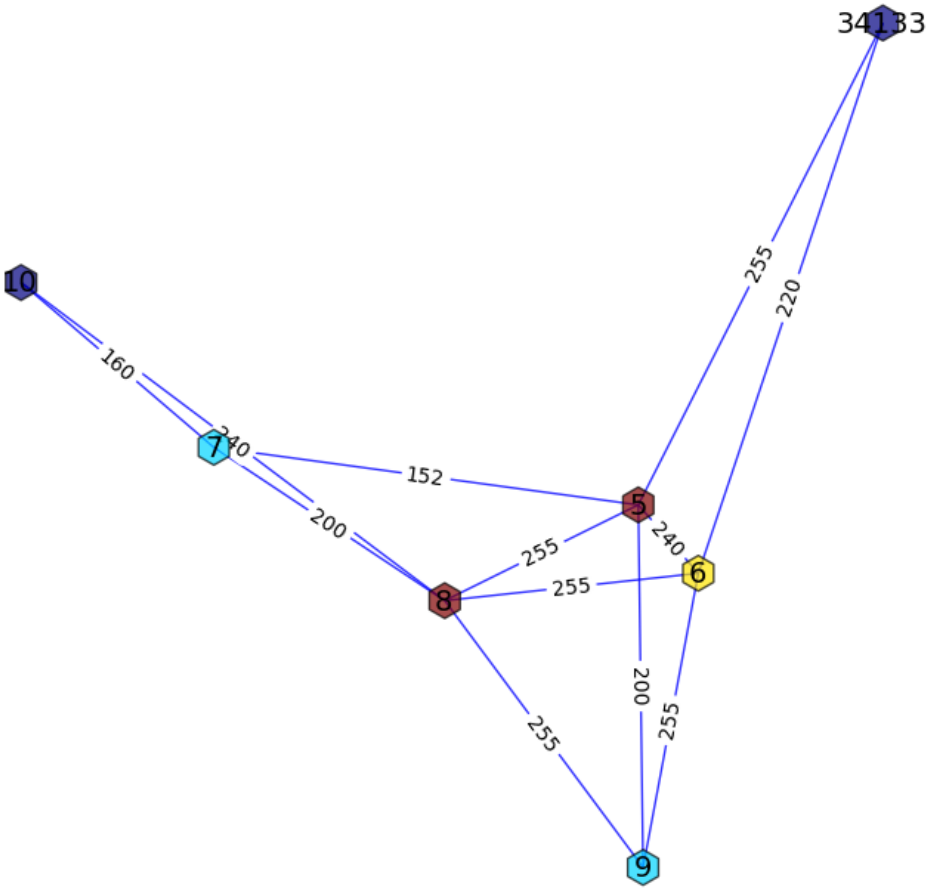
Aktualizovaný uzol po prijatí a zavedení novej verzie na základe tabuľky susedov vie, ktoré jeho susedné uzly zostávajú neaktualizované. Po prijatí správy „Hello“ uzol môže poslať neaktualizovanému uzlu žiadosť o začatie prenosu. V prípade kladnej odpovede dochádza k ďalšiemu prenosu hlbšie do siete. Týmto spôsobom je možné preniesť firmware až do najvzdialenejšieho uzla siete od OTAU servera, bez potreby prenosu blokov firmwaru cez viaceré uzly.

4.2 Získanie topológie siete

LWM pre svoje základné fungovanie si nezostavuje tabuľku susedných uzlov. Táto funkcionalita bola doplnená z dôvodu vytvorenia topológie celej siete na strane kontrolnej aplikácie otaShell a prehľadu o verzii firmwaru bežiacom na uzloch.

Každý uzol okrem aplikačnej správy `appMsg` s dátami zo senzorov zasiela správu `Hello` zo zdrojového a cieľového `APP_OTA_ENDPOINT` susedným uzlom. Správa sa posiela ako broadcast s nastaveným bitom `NWK_OPT_LINK_LOCAL` v poli kontroly rámca, aby prijímajúce uzly danú správu nepreposielali ďalej. Štruktúra tejto správy pozostáva z identifikátoru správy a verzie firmwaru zasielajúceho uzla. Ak sa v konfiguračnom súbore nedefinuje `APP_ENABLE_OTA`, tak sa daná správa neodosiela.

Služba otaClient tieto správy prijíma od susedných uzlov na základe čoho si vytvára tabuľku susedov. Tabuľka obsahuje adresu uzla, RSSI, LQI a verziu získané z prijatej správy typu `NWK_DataInd_t`. Veľkosť tabuľky je nastaviteľná pomocou konštanty `NWK_NEIGBOURS` v config.h. Aktuálne informácie sa udržujú pomocou periodickej obnovy tabuľky, ktorej doba je nastaviteľná konštantou `NWK_NEIGBOURS_CHECK_INT`.



Obr. 4.2: Príklad vytvoreného grafu

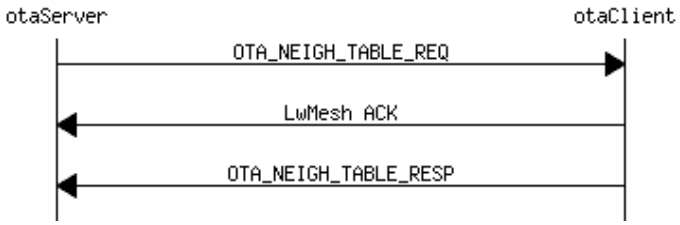
Súčasťou mechanizmu získanie topológie je aj odoslanie tabuľky susedov na vyžiadanie správou `OTA_NEIGH_TABLE_REQ` OTAU serverom a jej spracovanie v otaShell. Žiadosť sa zasiela najprv uzlom v priamom dosahu OTAU servera a na základe získaných informácii sa zasielajú ďalšie žiadosti postupne hlbšie do siete. Týmto postupom sa zároveň získa informácia o počte skokov od OTAU servera ku konkrétnemu uzlu.

Z prijatej tabuľky od každého uzla sa vytvára graf siete, kde spojnice medzi vrcholmi nesú informáciu o parametre LQI. Parameter LQI bol zvolený z dôvodu jeho výpovednej hodnoty o kvalite daného bezdrôtového spoja, ktorá sa určuje na základe úspešnosti prijatých rámcov kontrolou integrity pomocou kontrolného súčtu. Výsledný graf je vykreslený pomocou pythonovského knižnice Matplotlib [14] a uložený do súboru vo formáte PNG a taktiež ako pythonovský objekt vo formáte

pickle. Objekt graf je možné znovu načítať zo súboru, bez nutnosti opätovného získavania tabuliek z uzlov. Príklad vytvoreného grafu je na obrázku 4.2. Farba jednotlivých uzlov je volená na základe stupňa daného uzla v grafe, to znamená závisí od počtu hrán k susedným uzlom. Toto farebné odlíšenie môže napríklad slúžiť k rýchlejšiemu nájdeniu uzlov s rovnakým počtom potomkov v rozsiahlejšom grafe. Konkrétne na grafe napríklad vidíme dva krajné uzly 34133 a 10 fialovej farby, ktoré nesprostredkujú spojenie s ďalšími uzlami, ktoré by boli mimo dosah, pomocou viacskokovej (multihop) komunikácie.

4.3 Služba otaServer

Beží na uzle OTAU server. Hlavnými funkciami je iniciácia a prenos blokov aktualizácie na úrovni komunikácie so službou otaClient. Služba bola doplnená o zasielanie požiadavku pre tabuľku susedov a jej následné prijatie znázornené na 4.3. Štruktúra správy nesúcej tabuľku je v prílohe C. Na úrovni komunikácie so službou

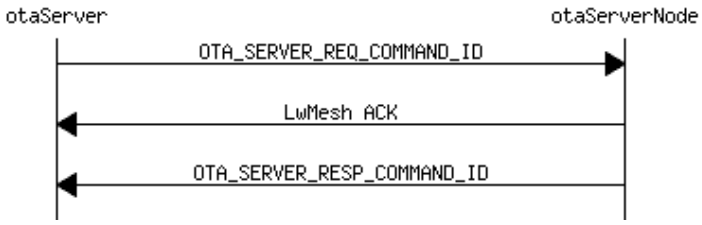


Obr. 4.3: Prenos tabuľky susedov

otaServerNode posiela príkaz uzlom na začatie upgradu uzlu špecifikovaného adresou. Štruktúru správ z obrázka 4.4 možno vidieť v prílohe C v sekcii INIT. V odpovedi na tento požiadavok sa zasiela hlavne status koncového uzla a v prípade kladnej odpovedi `OTA_SUCCESS_STATUS` začína úspešne prenos firmwaru. Ostatné statusy sú preposielané do OTAU serveru. Chybne prenesené bloky alebo nedoručenie je riešené opakovaným zaslaním. Prenos medzi OTAU serverom a službou otaClient nepodporuje obnovu relácie, tak ako bude ďalej popisované pre službu otaServerNode.

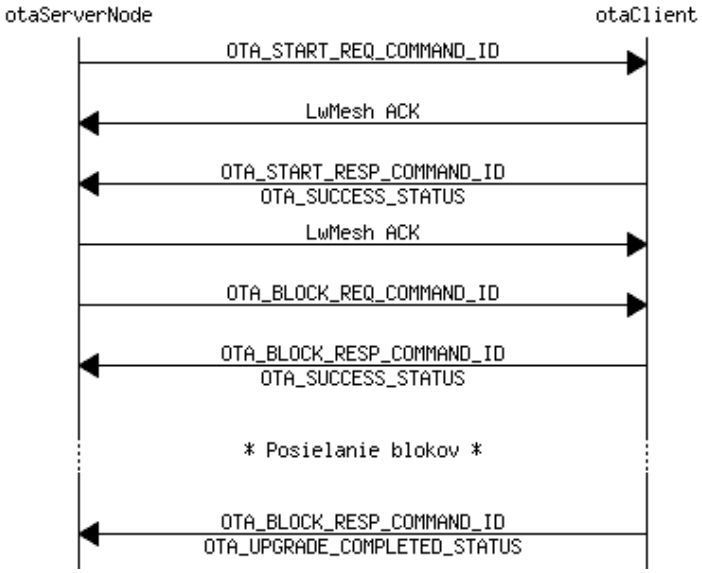
4.4 Služba otaServerNode

Umožňuje hlavne vyčítanie firmwaru z pamäte flash a jeho prenos medzi uzlami, ktorý by bez tejto služby nebol možný. Obsluhuje správy posielané na `APP_OTA_SERVERPOINT` od OTAU servera v podobe žiadostí ako napríklad začatie prenosu do uzla.



Obr. 4.4: Inicializovanie prenosu z uzla do uzla

Prenos firmwaru klientovi sa začína žiadosťou o štart, v ktorej je klient informovaný o verzii ponúkaného firmwaru. Ak je ponúkaná verzia väčšia, tak sa začne prenos blok po bloku. Medzi prenosom jednotlivých blokov je definovaná časová medzera na prijatie a spracovanie na strane klienta pomocou časovača. Veľkosť medzere je možné nastaviť konštantou `OTA_FRAME_SPACING` v *otaCommon.h* hlavičkovom súbore spoločnom pre obe OTA služby. Spolu s bajtami bloku sa prenáša jeho veľkosť a kontrolný súčet vypočítaným algoritmom CRC na kontrolu voči chybám po prijatí klientom. Náhľad do štruktúry je v sekcii „block“ prílohy C. Celý záznam komunikácie je dostupný v digitálnej prílohe v súbore *n2nNoCrypt.pcapng*



Obr. 4.5: Komunikácia otaServerNode a otaCLient

Chyba prenosu bloku je zabezpečená dvoma spôsobmi: opätovným zaslaním a opakovaním relácie. V prípade prenosu bloku s chybou klient odpovedá statusom `OTA_CRC_ERROR_STATUS` a server blok posiela znovu. Taktiež ak klient neodpovedá vôbec, dochádza k opätovnému poslaniu. Toto opakovanie je limitované konštantou

`OTA_MAX_RETRIES` predvolenou na tri opakovania. Druhý spôsob rieši problém, keď klient neodpovedá do doby vypršania relácie, ktorej časovač je nastavený na 2 sekundy konštantou `OTA_RESPONSE_TIMEOUT`. Obnovenie relácie rieši problém úplnej straty signálu s klientským uzlom a hlavne umožňuje pokračovať v prenose blokov od posledného úspešného preneseného bloku. Týmto prístupom sa prenos nemusí začínať úplne od znova a dosiahne sa tak menšieho energetického vyťaženia uzla a časovej úspory hlavne pri prenose veľkých súborov aktualizácie. V odladovacom výstupe D môžeme vidieť ako prebieha zápis po stránkach do pamäte flash. V tomto výstupe je vidieť prerušenie prenosu, ktoré bolo simulované vypnutím a zapnutím uzla. Pri obnove relácie sa v správe „start“ C prenáša hodnota stránky a bajtový posun vrámci tejto stránky kam sa má pokračovať so zápisom. Celý záznam odchytenej komunikácie je dostupný v digitálnych prílohách ako *interrupt_n2nA.pcapng*. Pri tomto teste bol prenášaný firmware o veľkosti 15210 bajtov rozdelený do 169 blokov. Zo záznamu bol odčítaný celkový čas 25,6 sekúnd z čoho reinicializácia spolu s opakovaním trvala 1,2 sekundy.

4.5 Služba otaClient

Funkcie a úlohy tejto služby vyplývajú z predošlých popisov. Treba však doplniť, že pri prijatí „Hello“ správy sa kontroluje verzia firmwaru susedného uzlu a dochádza k zaslaniu žiadosti o prenos v prípade nižšej verzie. Toto správanie je možné jednoducho vypnúť, závisí na požiadavkách pre realizáciu aktualizácie celej siete. Veľmi dôležitou súčasťou je indikácia prebiehajúcej aktualizácie do hlavnej aplikácie bežne bežiacej na senzorových uzloch. Táto indikácia slúži hlavne vyhnutiu sa uspávania uzlov typu Koncové zariadenie – End Device (ED), aby bolo možné dosiahnuť aktualizácie aj takýchto uzlov. Taktiež je možné pozastaviť iné funkcionality ako napríklad posielanie aplikačných správ koordinátorovi siete.

4.6 otaShell aplikácia

Úlohou tejto aplikácie je zasielanie príkazov do senzorovej siete prostredníctvom OTAU server uzla a taktiež vyhodnocovanie prijatých výsledkov. Pomocou tejto aplikácie je možné zrealizovať prenos firmwaru z počítača do senzorového uzla za použitia prepínača `-U`, zaslanie voliteľných správ resp. príkazov prepínačom `-s` a hlavne vytvorenie a zobrazenie grafu siete, ktoré následuje aktualizovaním celej siete prepínačom `-g`. Voliteľné správy slúžia na zaslanie príkazu uzlu pre inicializovanie prenosu aktualizácie `0x05: 'UART_COMMAND_SERVER_INIT'` a na vyžiadanie tabuľky

susedov `0x06: 'UART_COMMAND_REQUEST_NTABLE'` . Túto funkciu je možné jednoducho rozšíriť o ďalšie príkazy pridaním do slovníka `d_command`. Pre úspešné spustenie sú potrebné knižnice *networkx*[14] a *matplotlib*[15]. Výpis všetkých nastaviteľných parametrov pomocou možnosti help je zobrazený na výpise 4.1.

```
C:\Python27\python.exe W:/AADIP/#python/otaShell.py -help WSNDemo.bin 3
Usage: otaShell.py [options] file

Options:
  -h, --help show this help message and exit
  -d, --debug enable debug output
  -p PORT, --port=PORT communication port [default COM1]
  -b BAUD, --baud=BAUD communication baudrate [default 38400]
  -s MSG, --msg=MSG send custom message 0x05 || 0x06 [default empty]
  -U, --upgrade perform only server to node upgrade
  -g, --scan build network graph
  -i PANID, --panid=PANID
                      PAN ID [default 0x1234]
  -c CHANNEL, --channel=CHANNEL
                      channel [default 15]
  -a ADDR, --addr=ADDR server network address [default 0x8555]
  -n NODE, --node=NODE intermediary node [default 0x0010]
  -t CLIENT, --client=CLIENT
                      client network address [default 0x8001]
```

Výpis 4.1: Prepínače otaShell aplikácie

5 VYHODNOTENIE

V tejto časti práce boli vyhodnotené prenosy firmwaru na týchto scenároch:

- Prenos medzi OTAU serverom a susedným uzlom
- Prenos medzi dvoma uzlami
- Prenos do celej siete

Východzie nastavenia aplikácie a LWM bežiacej na uzloch sú v tabuľke 5.1

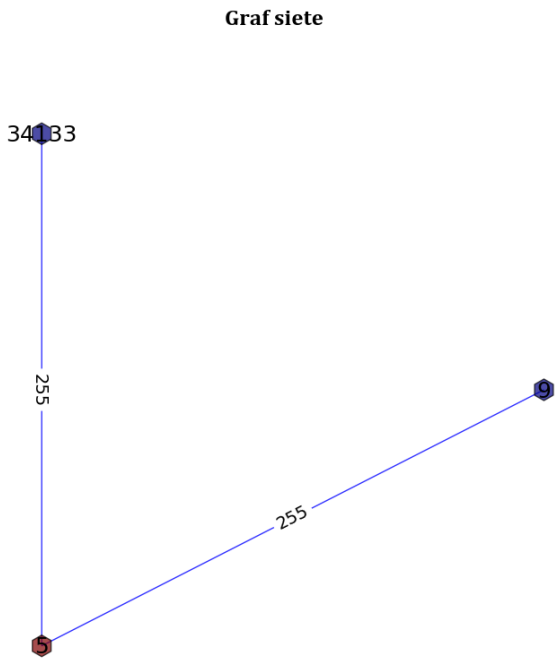
Tab. 5.1: Relevantné parametre z config.h

Parameter	**Hodnota**
APP_CHANNEL	0x0f
APP_PANID	0x1234
APP_SENDING_INTERVAL	8000
APP_ENDPOINT	1
APP_OTA_ENDPOINT	2
APP_OTA_SERVERPOINT	3
APP_SECURITY_KEY	„TestSecurityKey0“
APP_POWER	0x00 [1]
NWK_NEIGHBOURS	8
NWK_NEIGBOURS_CHECK_INT	20000

5.1 Prenos medzi dvoma uzlami

V tomto prípade išlo o prenos v následujúcej topológii na Obr. 5.1, kde dochádza k počiatočnému prenosu z operačného systému Windows cez sériovú linku prostredníctvom uzla 0x8555 (dekadicky 34133) k susedovi so sieťovou adresou 0x0005. Následne po uložení a zavedení novej verzie v tomto uzle dochádza k prenosu do uzla 0x0009. Táto schéma prenosu bola realizovaná so šifrovaním a bez šifrovania všetkých prenášaných správ. Pre poslanie šifrovanej správy je nutné nastaviť možnosť `NWK_OPT_ENABLE_SECURITY` pre dátový požiadavok typu `NWK_DataReq_t`. Toto nastavenie sa zároveň premietne nastavením bitu zapnutia zabezpečenia (Security Enabled) v 8 bitovom Frame Control Field LWM rámca. Následne je potreba zadefinovať kľúč (`APP_SECURITY_KEY "keystring"`) a režim šifrovania (`SYS_SECURITY_MODE 0`) a povoliť šifrovanie definovaním `NWK_ENABLE_SECURITY` v *config.h*. Zvolený režim nula znamená, že sa použije hardvérovo akcelerované AES-128 šifrovanie.

[1] 0x00 = 3.5 dBm; 0x0f = −16.5 dBm



Obr. 5.1: Topológia s jedným skokom

V LWM je možné ešte zvoliť režim jedna, v ktorom sa správy šifrujú softvérovo algoritmom eXtended Tiny Encryption Algorithm (XTEA). Podľa [12] v zariadení, ktoré majú pripojenú rádiovú časť pomocou zbernice SPI a nie sú súčasťou čipu sa odporúča použitie XTEA. Dôvodom je možné odchytenie šifrovacieho kľúča, ktorý sa pri použití AES prenáša v čitelnej forme cez SPI zbernicu do rádiového čipu. Takže ak sú uzly fyzicky prístupné nepovereným osobám je vhodnejšie použitie XTEA. Pri použití ATmega128RFA1 toto riziko práve nehrozí.

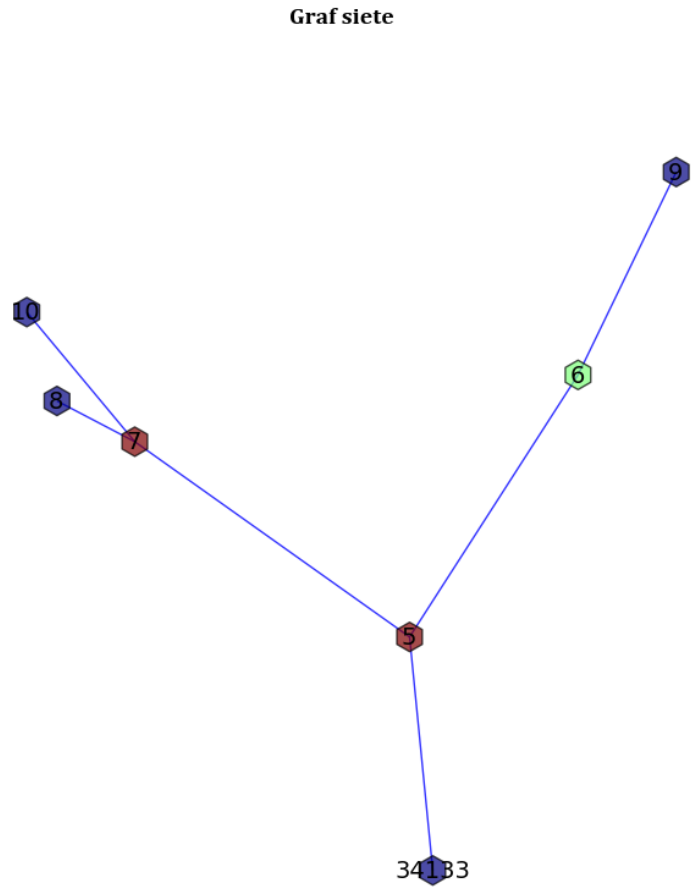
Tab. 5.2: Jednoskokový prenos

Spojenie so šifrovaním	**obraz (B)**	**Blokov**	**Čas prenosu(s)**	**Normalizovaný čas na kB (s)**
PC – 0x0005	13428	150	26,55	1,98
0x0005 – 0x0006	13428	150	13,17	0,98
Spojenie bez šifrovania				
PC – 0x0005	12456	139	17,50	1,40
0x0005 – 0x0006	12456	139	12,00	0,96

Časy testovania prenosu boli odčítané zo záznamu komunikácie vo Wiresharku od okamihu výzvy „start“ po odoslanie prvej správy klientským uzlom. Celé záznamy komunikácie sú dostupné v *n2nCrypt.pcapng* a *n2nNoCrypt.pcapng*. Pri použití

šifrovaného prenosu 90 B bloku je zabalený do správy o celkovej veľkosti 118 B a pri nešifrovaní do 114 B. Z výsledkov je zrejmé, že nie je vhodné prenášať obraz zakaždým z počítača po jednom do jednotlivých uzlov, ale hneď ako je už obraz dostupný v jednom uzle je výhodné pokračovať prenos práve od tohto uzla k susedným. Keďže obrazy bez a so šifrovaním sa veľkostne líšia z dôvodu počtu potrebných riadkov kódu, tak bol vypočítaný parameter Normalizovaný čas na kB, ktorý vyjadruje čas potrebný na prenos jedného kB, aby bolo možné tieto výsledky porovnať. Čas prenosu jedného kB so šifrovaním je dlhší preto, že do každej správy sú vložené ďalšie 4 B v podobe Kód integrity správy – Message Integrity Code (MIC). MIC sa vkladá za dátovú časť aplikačné správy LWM a slúži na overenie autenticity správy.

5.2 Aktualizácia siete



Obr. 5.2: Topológia testovacej siete

Otestovanie prenosu cez viac skokov a na sieti o viacerých uzloch bolo realizované na topológii na obrázku 5.2, ktorý je vytváraný aplikáciou otaShell. Z tejto topológie

vidíme, že dochádza k prenosu maximálne cez tri skoky. Uzol sedem musí aktualizovať postupne dva uzly. Táto topológia bola nastavená v uzloch fixne pomocou zapísania adries susedných uzlov do pamäti EEPROM, na základe ktorých si každý uzol vytvoril tabuľku susedov. Dôvodom takéhoto postupu bolo docielenie testovania na rovnakej sieti pre zapnuté a vypnuté šifrovanie, a taktiež aby nedochádzal k dynamickým zmenám topológie kvôli odrazom signálov v jednej miestnosti a premenlivosti celého rádiové prostredia v čase. Na obrázku 5.3 je rozostavenie uzlov počas testovania.

Celý scenár prebiehal následovne: OTAU server 0x8555 aktualizoval uzol 5, ktorý realizoval prenos do uzla 7. V tomto okamihu sú uzly 5 a 7 pripravené k ďalšiemu prenosu. Takže dochádza k simultánnemu prenosu medzi uzlami 7 a 8 a taktiež medzi 5 a 6. Pri tomto simultánnom prenose môže dochádzať k vzájomnému rušeniu, ale výrazné komplikácie komunikácie v analyzovanom provoze vo Wiresharku neboli spozorované až na dve zlé FCS u dvoch blokov. Ďalej pri tomto simultánnom prenose môže dochádzať k navýšeniu čakacích intervalov pre prístup k médiu z dôvodov vyťaženia rádiového média susednými uzlami. Následne v ďalšom kroku sú obnovené aj posledné uzly s adresami 9 a 10 ich najbližšími susedmi 6 a 7.

Tab. 5.3: Multihop v sieti

Spojenie so šifrovaním	**obraz (B)**	**Blokov**	**Čas prenosu(s)**
Sieť 6 uzlov	13450	150	88,90
Spojenie bez šifrovania			
Sieť 6 uzlov	12336	138	70,00

5.3 OTAU v BitCloud

Jednou z najpoužívanejších implementácií protokolu ZigBee je BitCloud od Atmelu.

Fungovanie OTAU v LWM a BitCloud bolo porovnané principiálne a na základe praktických výsledkov. BitCloud stack umožňuje aktualizáciu uzlov rovnakou formou ako LWM popisovanom v Podkapitole 3.3 bez navrhnutého riešenia s rozdielnou implementáciou a štruktúrou správ, keďže sa jedná o komplexný a ucelený stack v porovnaní s LWM, ktorý je v podstate na začiatku svojho vývoja. V BitCloud je potrebné udržovať obrazy firmwaru zvlášť pre jednotlivé typy zariadení, čo vo výsledku znamená, že je možné prenášať napríklad firmvér len pre smerovače v sieti, alebo len pre koncové zariadenia. Na druhú stranu v LWM typ zariadenia a jeho funkcia sa odlišuje na úrovni samotnej aplikácie prostredníctvom zvolenej adresy. Keďže

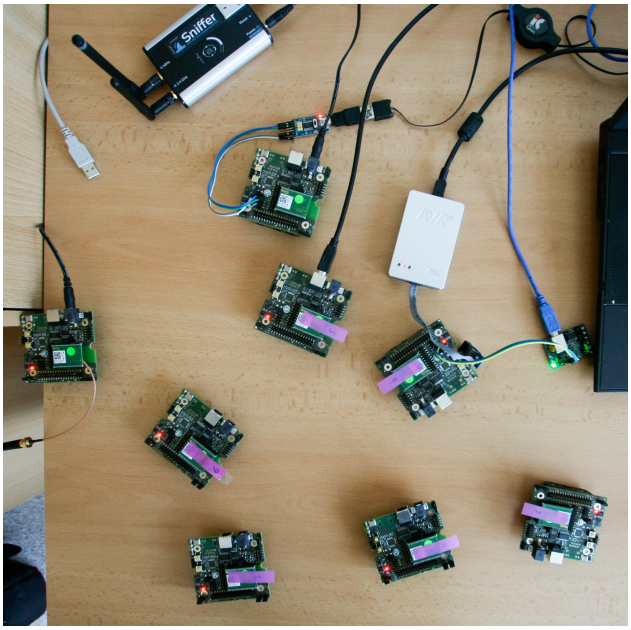
práca bola zameraná na LightWeight Mesh, tak informácie k OTAU v BitCloud sú dostupné v dokumentácii [17].

Výsledky porovania na základe potrebného času na prenos v BitCloude a LWM s navrhnutým riešením je uvedené v tabuľke 5.4. V prostredí BitCloud bol prenesený obraz určený pre zariadenie typu smerovač.

Aktualizáciu celej siete v oboch stakoch by bolo možné realizovať aj postupným prenosom z PC do každého uzla zvlášť. V prípade tohto scenára sa čas prenosu navyšuje počtom skokov potrebných na dosiahnutie cieľového uzla.

Tab. 5.4: Multihop v LWM a BitCloud

Platforma	**Počet skokov**			
	1		2	
	Čas prenosu(s)	**Normalizovaný čas na kB (s)**	**Čas prenosu(s)**	**Normalizovaný čas na kB (s)**
BitCloud (107 kB) bez zabezpečenia	255	2,38	390	3,64
BitCloud (121,5 kB) štandardné zabezpečenie	400	3,29	615	4,50
BitCloud (123,2 kB) štandardné linkové zabezpečenie	540	3,90	754	6,08
LWM (12,45 kB) bez zabezpečenia	17,50	1,40	20,80	1,67
LWM (13,43 kB) zabezpečenie	26,55	1,98	36,20	2,70



Obr. 5.3: Rozloženie testovacej siete

6 ZÁVER

Diplomová práca v teoretickom úvode rozoberá dôvody preprogramovania uzlov senzorovej siete. Z pohľadu typu prenosu existujú tri hlavné spôsoby ako preniesť nový firmware do senzorových jednotiek: singlehop, multihop a broadcast flooding.

V teórii boli rozobrané problémy, ktoré sa riešia pri realizácii updatu softvéru. Jedným z hlavných problémov je rozoslanie nového firmwaru sieťou do skupiny alebo všetkých uzlov. Existujúce protokoly riešiace rozosielanie (Trickle, XNP, MOAP, Deluge) boli naštudované a popísané na akom princípe pracujú. Na základe týchto protokolov boli stanovené základné požiadavky riešenia tohoto problému.

Jadro práce pozostávalo v navrhnutí a implementácii systému updatu firmwaru senzorových jednotiek rádiovým prenosom v prostredí LightWeight Mesh, ktorý spĺňa štandard IEEE 802.15.4. Systém bol navrhnutý tak, že je možné realizovať prenos do uzlov tromi spôsobmi. V prvom prípade ide o prenos z PC prostredníctvom uzla OTAU server do uzlov v jeho rádiovom dosahu. Druhý spôsob pozostáva v zaslaní príkazu uzlu s disponibilným firmwarom, aby zrealizoval prenos do adresou špecifikovaného uzlu. Tretí spôsob je postavený na výmene režijných „Hello“ správach medzi susednými uzlami. Uzol s novšou verziou po obdržaní požiadavku pre aktualizovanie realizuje prenos. Navrhnutý systém prenáša celý binárny obraz firmwaru o veľkosti 11–15 kB. Na obsluhu aktualizácie bola napísaná konzolová aplikácia otaShell, ktorá zároveň vytvára súbory s grafom siete vo formáte .gpickle a .png.

Navrhnutý systém bol prakticky vyhodnotený na senzorových uzloch deRFnode. Vyhodnocovanie prebiehalo podľa troch scenárov prenosu: medzi OTAU serverom a susedným uzlom, medzi dvoma uzlami a prenos do celej testovacej siete. Taktiež bol porovnaný prenos so šifrovaním a bez šifrovania správ. Z výsledkov vyplýva, že priamy prenos medzi dvoma susednými uzlami je podstatne rýchlejší ako prenos medzi PC a uzlom. Dôvodom je nižšia réžia a priama komunikácia medzi uzlami. Uplatnenie šifrovania vyžaduje prenos väčšieho binárneho súboru a použitie dlhších správ. Pri realizácii aktualizácie siete o šiestich uzloch to znamenalo predĺženie celkového času o 8,9 sekundy pri zapnutom šifrovaní.

Protokol ZigBee v prostredí BitCloud bol porovnaný na základe praktických výsledkov. V zásade umožňuje poslanie updatu akémukoľvek uzlu v sieti použitím dátového prenosu a smerovania. Keďže LWM a BitCloud sa veľkostne výrazne líšia, tak boli porovnané na základe normalizovaného času potrebného na prenos jedného kilobajtu v procese aktualizácie. Bez použitia zabezpečenia potrebuje BitCloud pre singlehop prenos 2,38 s a LWM 1,98 s. Pri prenose cez dva skoky to je 3,64 s a 2,70 s. Nižšie časy pre LWM boli dosiahnuté nižšou réžiou a hlavne tým, že nový firmware nie je preposielaný uzlami na celej trase medzi OTAU serverom a klientským uzlom ako u BitCloudu. Z toho vyplýva aj lepšia škálovateľnosť systému v LWM.

Navrhnutý systém je možné rozšíriť o analyzovanie preložených binárnych obrazov pred prenosom. V prípade malých zmien zdrojového kódu, by sa prenášali len tie časti pamäte, ktoré sú rozdielne. Pri prenose a zápise do pamäte by bolo možné použiť navrhnuté funkie a správy, ktoré nesú informáciu o adrese stránky a bajtovom ofsete, kde sa do pamäte má zapisovať. Konzolovú aplikáciu je možné premietnuť do aplikácie s grafickým rozhraním a rozšíriť o monitorovanie prebiehajúcej aktualizácie celej siete.

LITERATÚRA

[1] JEONG, J., S. KIM a A. BROAD. *Network Reprogramming. TinyOS documentation.* [online]. Berkeley, 2003 [cit. 29.10.2013]. Dostupné z: http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/NetworkReprogramming.pdf

[2] CROSSBOW TECHNOLOGY INC. *Mote In-Network Programming User Reference* [online]. 2003 [cit. 29.10.2013]. Dostupné z: http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/Xnp.pdf

[3] BROWN, S., C.J. SREENAN. *Updating Software in Wireless Sensor Networks: A Survey.* 2006. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.4510&rep=rep1&type=pdf

[4] STOLIKJ, Milosh, Pieter J. L. CUIJPERS a Johan J. LUKKIEN. Efficient reprogramming of wireless sensor networks using incremental updates. In: 2013 *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops).* San Diego, CA: IEEE, 2013, s. 584-589. ISBN 978-1-4673-5077-8. DOI: 10.1109/PerComW.2013.6529563. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6529563

[5] JAEIN JEONG a D. CULLER. Incremental network programming for wireless sensors. In: 2004 *First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004.* IEEE SECON 2004. IEEE, 2004, s. 25-33. ISBN 0-7803-8796-1. DOI: 10.1109/SAHCN.2004.1381899. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1381899

[6] LEVIS, Philip, Neil PATEL, David CULLER a Scott SHENKER. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI).* 2004, s. 15-28. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.5116&rep=rep1&type=pdf

[7] MILLER, Christopher a Christian POELLABAUER. PALER: A Reliable Transport Protocol for Code Distribution in Large Sensor Networks. In: *2008 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks.* San Francisco, CA: IEEE, 2008, s. 206-214. ISBN 978-1-4244-1777-3. DOI: 10.1109/SAHCN.2008.34. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4557757

[8] STATHOPOULOS, Thanos, John HEIDEMANN a Deborah ESTRIN. *A Remote Code Update Mechanism for Wireless Sensor Networks* [online]. 2003 [cit. 29.10.2013]. Dostupné z: `http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.5.4326&rep=rep1&type=pdf`

[9] HUI, Jonathan W. a David CULLER. The dynamic behavior of a data dissemination protocol for network programming at scale. In: *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*. New York, NY, USA: ACM Press, 2004, s. 81-94. ISBN 1-58113-879-2. DOI: 10.1145/1031495.1031506. Dostupné z: `http://portal.acm.org/citation.cfm?doid=1031495.1031506`

[10] DRESDEN ELEKTRONIK. *User Manual deRFnode / deRFgateway* [deRFkitCD_v1.3]. Version 1.2. 2012 [cit. 3.12.2013].

[11] DRESDEN ELEKTRONIK. *User Manual Radio Modules: deRFmega128-22A00* [deRFkitCD_v1.3]. Version 1.4. 2011 [cit. 3.12.2013].

[12] ATMEL CORPORATION. *Atmel AVR2130: Lightweight Mesh Developer Guide*. 2013.

[13] *Open Sniffer for 802.15.4, Zigbee, 6LoWPAN* [online]. 2013 [cit. 2014-05-13]. Dostupné z: `http://www.sewio.net/open-sniffer/`

[14] Matplotlib. HUNTER, John, Darren DALE, Eric FIRING a Michael DROETTBOOM. *matplotlib* [online]. 2013 [cit. 2014-05-16]. Dostupné z: `http://matplotlib.org/`

[15] NetworkX. *NetworkX Documentation* [online]. 2013 [cit. 2014-05-16]. Dostupné z: `https://networkx.github.io/documentation.html`

[16] CAMERA, Dean. *Using the EEPROM memory in AVR-GCC*. 2014, 12 s. Dostupné z: `https://github.com/abcminiuser/avr-tutorials/blob/master/EEPROM/Output/EEPROM.pdf`

[17] ATMEL CORPORATION. *Atmel AVR2058: BitCloud OTAU User Guide*. 2012.

ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

ACK	Potvrdenie – Acknowledgement
AODV	Smerovanie na základe dĺžky vektoru v Ad hoc sieťach – Ad hoc On-Demand Distance Vector
CRC	Cyclic redundancy check
ED	Koncové zariadenie – End Device
EEPROM	Elektricky mazateľná pamäť ROM – Electrically Erasable Programmable Read-Only Memory
LQI	Indikátor kvality spoja – Link Quality Indicator
LWM	LightWeight Mesh
NACK	Záporné potvrdzovanie – Negative Acknowledgement
MIC	Kód integrity správy – Message Integrity Code
OTAU	Upgrade pomocou rádiovej komunikácie – Over the Air Upgrade
OTA	Cez rádiové prostredie – Over the Air
PAN	Veľmi malá osobná sieť – Personal Area Network
PSFQ	Pomalé pretláčanie sieťou, Rýchle doručovanie – Pump Slowly, Fetch Quickly
RMST	Spoľahlivý viacsegmentový prenos – Reliable Multi-Segment Transport
RSSI	Indikátor úrovne prijatého signálu – Received Signal Strength Indicator
WSN	Bezdrôtová senzorová sieť – Wireless sensor network
XTEA	eXtended Tiny Encryption Algorithm

ZOZNAM PRÍLOH

A OBSAH PRILOŽENÉHO DVD

- /
  - `práca` ..... elektronická verzia tejto práce
  - `prílohy`
    - `LWM_stack` ..... Lightweight Mesh Software stack
    - `otaShell` ..... konzolová aplikácia spolu s grafmi
    - `python_moduly` ..... použité rozširovacie moduly
    - `zaznamy_provozu` ..... pcapng z programu Wireshark

B OBSAH LIGHTWEIGHT MESH SOFTWARE STACK

Priložených archív k práci obsahuje LWM stack, súčasťou ktorého je zložka apps s použitými a upravenými aplikáciami pre jednotlivé komponenty potrebné pre vykonanie updatu firmwaru. Konkrétne sú to tieto aplikácie:

- Bootloader
- OTAServerDemo
- WSNDemo

C ŠTRUKTÚRY PRENÁŠANÝCH SPRÁV

```
/*** HEADER ***/
typedef struct PACK OtaCommandHeader_t
{
  uint8_t commandId;
  uint16_t sessionId;
} OtaCommandHeader_t;
/*** NEIGHBOUR TABLE ***/
typedef struct PACK OtaNeighTableCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  Neighbour_t recvTable[NWK_NEIGHBOURS];
} OtaNeighTableCommand_t;
/*** START ***/
typedef struct PACK OtaStartReqCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint32_t size;
  uint8_t version;
  uint16_t page;
  uint8_t byteoffset;
} OtaStartReqCommand_t;

typedef struct PACK OtaStartRespCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint8_t status;
} OtaStartRespCommand_t;

/*** INIT ***/
typedef struct OtaInitReqCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint16_t targetNode;
  uint32_t size;
  uint8_t version;
} OtaInitReqCommand_t;
```

```
typedef struct OtaInitRespCommand_t
{
  uint8_t commandId;
  uint8_t status;
  uint16_t sessionId;
  uint8_t node;

} OtaInitRespCommand_t;
////////////////////////////////////////
/*** BLOCK ***/
typedef struct PACK OtaBlockReqHeader_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint16_t crc;
  uint8_t size;
} OtaBlockReqHeader_t;

typedef struct PACK OtaBlockReqCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint16_t crc;
  uint8_t size;
  uint8_t data[OTA_MAX_BLOCK_SIZE];
} OtaBlockReqCommand_t;

typedef struct PACK OtaBlockRespCommand_t
{
  uint8_t commandId;
  uint16_t sessionId;
  uint8_t status;
} OtaBlockRespCommand_t;
```

D ODLAĎOVACÍ VÝSTUP SO ZÁPISOM BLOKOV

```
INITCCNode start
size=15120
Client response
size=15030
Client response
size=14940
Page address 65536 wr 0
Client response
size=14850
Client response
size=14760
Client response
size=14670
Page address 65792 wr 1
Client response
size=14580
Client response
size=14490
Client response
size=14400
Page address 66048 wr 2
.........................
Page address 69376 wr 15
Client response
size=10980
Client response
size=10890
Client response
/* Prerušenie vypnutím */
Node start
size=10800
byteoffset=224, page=16
Page address 69632 wr 16
.........................
/* Zápis poslednej stránky */
Page address 80384 wr 58
Client response
size=0
Client response
/* Štart uzla */
Node start
```