VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ ZAŘÍZENÍ NAVIGOVANÉ INTELIGENTNÍ SÍTÍ

MOBILE DEVICES NAVIGATED BY AN INTELLIGENT NETWORK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ANDREJ BARNA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Barna Andrej, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Mobilní zařízení navigované inteligentní sítí**
Mobile Devices Navigated by an Intelligent Network
Kategorie: Modelování a simulace
Zadání:

1. Seznamte se s problematikou inteligentních budov a také s využitím bezdrátových senzorových sítí v nich.
2. Nastudujte současné metody pro identifikaci a sledování pohyblivého uzlu v reálném prostředí. Zaměřte se na technologie bezdrátových senzorových sítí a použití indikace síly signálu při komunikaci (RSSI) v systémech s těmito technologiemi.
3. Navrhněte distribuovanou aplikaci pro sledování a navigaci pohybujících se uzlů. Předpokládajte, že účelem má být navigovat, například světelně, pohyb osoby k zadanému místu v budově.
4. Implementujte navržené řešení a ověřte jeho funkčnost při určování směru pohybu mobilního uzlu a schopnosti sítě uzel navigovat.
5. Diskutujte zjištěné nedostatky, problémy a jejich možná budoucí řešení. Řešení prezentujte formou plakátu A2.

Literatura:

- Levis, P., Gay, D.: TinyOS Programming, 2009 (online)
- Fischione, C.: An Introduction to Wireless Sensor Networks, 2014 (online)

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Práca sa zaoberá problematikou navigovania v budovách za využitia bezdrôtových senzorových sietí. Pri riešení sa aplikujú znalosti z multiagentných inteligentných systémov. Pre beh agentov v senzorových uzloch sa využíva platforma WSageNt, ktorá beží na operačnom systéme TinyOS. Tiež sa popisuje jazyk ALLL, ktorý sa využíva pri tvorbe agentov na platforme WSageNt. Za využitia týchto technológií je navrhnuté riešenie tohto problému formou rozšírenia platformy WSageNt a troch agentov, ktorí napĺňajú požadovanú funkcionalitu. Následne je popísaná implementácia tohto návrhu a jej testovanie, konštatujúce výhody a nedostatky vytvoreného systému.

Abstract

This work studies the problem of indoor navigation, using wireless sensor networks. The knowledge of multiagent systems is being used in the process of solving this problem. Platform WSageNt, running on the operating system TinyOS, is being used for the execution of the agents. Agents themselves on the WSageNt platform are created in the language ALLL, which is described in the process. With use of those technologies, a solution is proposed as an extension to the WSageNt plaform and three agents, which fulfill the desired functionality. Then the implementation of this solution is described, followed by its testing, stating advantages and disadvantages of the created system.

Kľúčové slová

bezdrôtová sieť, TinyOS, agent, ALLL, RSSI, lokalizácia, navigácia, IRIS, WSageNt

Keywords

wireless network, TinyOS, agent, ALLL, RSSI, localization, navigation, IRIS, WSageNt

Citácia

BARNA, Andrej. *Mobilní zařízení navigované inteligentní sítí.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Mobilní zařízení navigované inteligentní sítí

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána doc. Ing. Františka Zbořila, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Andrej Barna
20. mája 2019

Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce, doc. Ing. Františkovi Zbořilovi, Ph.D., za jeho ochotu pri konzultáciách práce a trpezlivosť.

Obsah

Kapitola 1

Úvod

Moderná doba tlačí na aplikovanie inteligentných systémov v rámci každodenného bežného života. Či už sa jedná o rôzne zariadenia na monitorovanie zdravotného stavu človeka (čo napríklad dokážu rôzne Smart hodinky), autonómne poľnohospodárske zariadenia, alebo inteligentné budovy, všetky tieto spomenuté technologicky pokročilé zariadenia sú dôkazmi o úspešnosti tohto začleňovania inteligentných systémov do moderného života.

Vezmime si do úvahy inteligentné budovy. Tieto objekty majú za účel čo najviac zjednodušiť život ľuďom, ktorí v týchto budovách bývajú, pracujú, či inak trávia čas. Za týmto účelom sa inteligentné budovy vybavujú rôznymi podpornými systémami, ktoré napríklad regulujú úroveň svetla v miestnostiach, čistotu a teplotu vzduchu, alebo iným spôsobom podporujú ľudí vo vykonávaní ich obvyklých činností.

Problémom, s ktorým sa už určite väčšina ľudí stretla, je navigovanie vo väčších budovách, najmä ak sa jedná o prostredie neznáme danej osobe. Jedná sa o netriviálny problém, ktorý vyžaduje ako zisťovanie aktuálnej pozície danej osoby, tak aj hľadanie čo najkratšej cesty k zvolenému bodu, kam sa táto osoba chce dostať.

Tento problém je možné riešiť s využitím bezdrôtových senzorových sietí, ktoré dokážu určiť polohu zariadenia (ktoré by mohla mať navigovaná osoba so sebou) v rámci skúmaného prostredia a následne by táto sieť dokázala navigovať túto osobu, napríklad pomocou vizuálnych signálov.

Ďalej v rámci riešenia tohto problému je možné aplikovať znalosti z multiagentných systémov, ktoré dokážu využívať spoluprácu viacerých agentov za účelom dosiahnutia cieľa. Pre tento problém by každý uzol senzorovej siete predstavoval agenta. Títo agenti by následne vzájomne komunikovali tak, aby danej osobe určili kam má ísť, aby sa dosiahlo požadovaného cieľa.

V niekoľkých prvých kapitolách budú rozobraté znalosti, potrebné k vyriešeniu tohto problému. Začínajúc prebratím senzorov ako prvkov bezdrôtových senzorových sietí, sa rozoberie ich štruktúra a účel. Následne sa rozoberú bezdrôtové senzorové siete ako celky a problém lokalizácie v nich. Ďalej bude spomenutý operačný systém TinyOS, určený práve k použitiu v bezdrôtových senzorových sieťach. V ďalšej časti sa rozoberú multiagentné systémy — princípy, na akých fungujú a bude rozobratý aj jazyk Agent Low Level Language (ALLL), použitý pri riešení tohto problému. Napokon bude v skratke rozobraná platforma WSageNt, ktorá je postavená nad TinyOS a umožňuje využitie agentného jazyka ALLL a teda aj agentných prístupov v bezdrôtových senzorových sieťach.

Potom budú nasledovať kapitoly o vlastnom návrhu riešenia tohoto problému, jeho realizácii a skúmaní vlastností vzniknutého systému. V kapitole 6 bude predstavený návrh možného riešenia tejto problematiky, za využitia znalostí získaných skúmaním vyššie uve-

dených technológií, ktorý bude založený na rozšírení platformy WSageNt a troch agentoch. Potom v kapitole 7 sa popíše spôsob, ktorým boli jednotlivé celky systému navrhnutého v kapitole 6 implementované, pričom sa následne v kapitole 8 popíšu určité problémy a nedostatky, na ktoré sa narazilo v priebehu implementácie a spôsob, akým sa tieto problémy riešili.

Kapitola 2

Senzory

V rámci vývoja vojenských technológií sa v minulosti vyvíjali prostriedky, ktoré by boli schopné detegovať napríklad pohyb v určitej oblasti, či už v exteriéri alebo interiéri. Tieto zariadenia, zvané senzory, majú celú sériu požiadaviek na ich funkčnosť a schopnosti, pričom sa eventuálne adoptovali aj pre bežné každodenné civilné využitie, ako napríklad monitorovanie budov či životného prostredia (pre zistenie znečistenia), monitorovanie zdravotného stavu, alebo poľnohospodárske využitie pre inteligentné poľnohospodárstvo. Znalosti prezentované v tejto kapitole boli čerpané z [4] a [11].

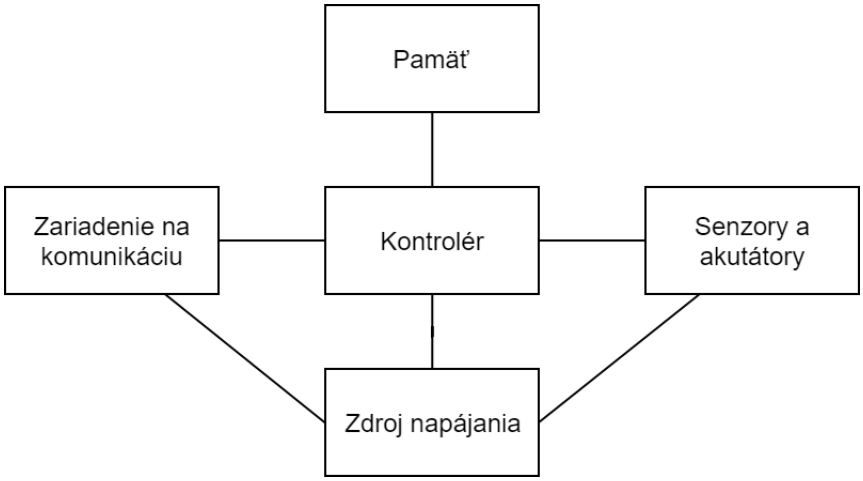
Primárnym účelom senzorov je snímanie javov z reálneho sveta — z ich prostredia, resp. okolia — a aplikovanie znalosti stavu snímaných javov (veličín) pri riešení nejakého problému, ku ktorému boli určené.

Ako bolo spomenuté, na tieto zariadenia sú určité nároky na funkčnosť, ktoré žiadali viaceré inovácie, aby boli efektívne použiteľné v praxi. Bežnými požiadavkami na senzory sú:

- veľmi nízka spotreba energie — senzory sú bežne napájané len batériami, pričom sa očakáva že vydržia bežať na jedno nabitie batérií dlhšiu dobu
- malé rozmery — pre praktickosť pri rozmiestnení v priestore
- nízka cena — aby ich bolo možné kúpiť dostatočné množstvo pre danú aplikáciu, keďže sa spájajú do sietí (popísané v nasledujúcej kapitole)
- výpočetná schopnosť — v závislosti od typu využitia senzorov
- schopnosť spracovávania a ukladania dát
- schopnosť komunikovať s inými zariadeniami — aby bolo možné z nich tvoriť siete

Kombinácia týchto požiadaviek spôsobuje dosť značné obmedzenia na senzory, aby boli prakticky použiteľné. Tieto obmedzenia však platia pre väčšinu bezdrôtových zariadení. To, čo od nich delí senzory je schopnosť snímať informácie z prostredia. Klasická schéma senzoru v bezdrôtových senzorových sieťach je znázornená na obrázku 2.1.

V doterajšom výklade sa pod pojmom „senzor“ myslelo celé zariadenie pozostávajúce z viacerých komponentov (uvedených vo vyššie spomenutom obrázku), ktoré bolo schopné fungovať v rámci bezdrôtovej siete. Ďalej budeme na takýto senzor referovať ako na uzol v bezdrôtovej senzorovej sieti (resp. senzorový uzol), ktorá bude popísaná v kapitole 3, a ako samotný senzor sa bude chápať len samotný komponent schopný snímať vnemy z prostredia, s prípadným analógovo-digitálnym prevodníkom.



Obr. 2.1: Schéma uzlu bezdrôtovej senzorovej siete[1]

2.1 Štruktúra uzlu bezdrôtovej siete

Jadrom takéhoto uzlu je kontrolér. Je to výpočetné jadro uzlu, ktoré má celú radu zodpovedností, ako napríklad spracovávanie dát zo senzorov, vykonávanie výpočtov nad získanými dátami a aplikovanie algoritmov, riadi ostatné komponenty uzlu, ale aj spolupracuje s inými uzlami, s ktorými je daný uzol spojený.

Ďalším komponentom uzlu je zariadenie na komunikáciu. Jedná sa o bezdrôtové rozhranie, ktorým dokáže uzol komunikovať s inými zariadeniami a ako posielať ostatným zariadeniam nejaké informácie, tak ich aj prijímať. Je to obvykle bežný vysielač/prijímač fungujúci na bázi rádiových vĺn. Okrem toho je to komponent systému s najvyšším odberom elektrickej energie.

Pamäť poskytuje úložisko dát, či už získaných zo senzorov, vypočítaných, prijatých bezdrôtovým rozhraním, alebo používaných k riadeniu behu uzlu (a teda mikrokontroléru). Dá sa využiť rôzne typy pamätí, či už RAM, ROM (pre program mikrokontroléru), alebo flash.

Ďalšou časťou uzlu je zdroj napájania. Obvykle sa používajú buď batérie (väčšinou nabíjateľné, ale nemusia nutne byť), alebo kondenzátory. Je možné využívať aj zdroje energie z okolitého prostredia, akými sú napríklad svetlo či kinetická energia a nimi dobíjať energiu uzlu napríklad fotovoltaickými článkami.

Poslednou uvedenou časťou sú senzory a akutátory. Senzory, ktoré už boli predtým spomenuté, sú súčiastky schopné snímať javy a/alebo veličiny z prostredia, akými sú napríklad vlhkosť, úroveň osvetlenia, tlak vzduchu a pod. Obvykle senzorové jednotky obsahujú v sebe aj analógovo-digitálny prevodník, ktorý poskytuje výstupy použiteľné mikrokontrolérom pri ďalšom spracovaní. Bežne bývajú uzly vybavené celou radou senzorov — v závislosti od toho, k čomu je daný uzol použitý. Akutátory sú elementy schopné manipulovať s prostredím, čo však od malých kompaktných uzlov nie je požadované často.

[1] Zdroj: [4], str. 17, obrázok 1.3

Vlastnosti procesoru	
Procesor	Atmel ATmega1281
Programová pamäť flash	128 KB
Sériová pamäť	512 KB
RAM	8 KB
Konfiguračná pamäť EEPROM	4 KB
Sériové rozhranie	UART
Číslicovo-digitálny prevodník (ADC)	10-bitový ADC
Ďalšie rozhrania	digitálne I/O, I^2C, SPI
Odber prúdu	8 mA (aktívny režim)
	8 μA (režim spánku)
Rádiový vysielač	
Frekvenčné pásmo	2405-2480 MHz
Prenosová rýchlosť	250 kbps
RF výkon	3 dBm
Citlivosť príjmu	-101 dBm
Odmietanie prenosu susedných kanálov	36 db (+5 MHz medzery medzi kanálmi)
	34 db (-5 MHz medzery medzi kanálmi)
Dosah	>300 m (v exteriéri)
	>50 m (v interiéri)
Odber prúdu	16 mA (režim príjmu)
	10 mA (-17 dBm vysielanie)
	13 mA (-3 dBm vysielanie)
	17 mA (3 dBm vysielanie)
Elektromechanické vlastnosti	
Batéria	2x AA batérie
Externé napájanie	2,7-3,3 V
Používateľské rozhranie	3 LED
Veľkosť (v mm)	58 x 32 x 7
Hmotnosť (v g)	18
Konektor pre rozšírenia	51-pinový konektor

Tabuľka 2.1: Špecifikácia Crossbow Iris XM2110[2]

Obr. 2.2: Crossbow Iris XM2110

2.2 Použité uzly

V rámci tejto technickej správy sa pri testovaní budú využívať uzly vyrobené spoločnosťou Crossbow, neskôr odkúpenou a v súčanosti už zvanou MEMSIC. Jedná sa o zariadenia Crossbow Iris XM2110, ktorých špecifikácie sú uvedené v tabuľke 2.1. Tieto uzly bezdrôtových sietí sa označujú ako motes (pl.). Kým samotný mote poskytuje výpočetné schopnosti a schopnosť si ukladať dáta a prenášať informácie, nemá sám o sebe senzory snímajúce informácie z prostredia. Z toho dôvodu je možné k motom pripojiť pomocou konektora dosky poskytujúce dodatočnú funkčnosť, akou je napríklad doska MTS420 (taktiež od spoločnosti MEMSIC), ktorá obsahuje akcelerometer, barometrický senzor, senzor úrovne svetla, či senzor vlhkosti alebo teplomer.

Ďalej sa bude využívať tzv. base station, ktorá je rozhraním medzi počítačmi a samotnými motami a umožňuje s motami komunikovať, alebo do nich nahrávať programy. V tejto práci sa bude využívať base station MIB520 od spoločnosti MEMSIC.

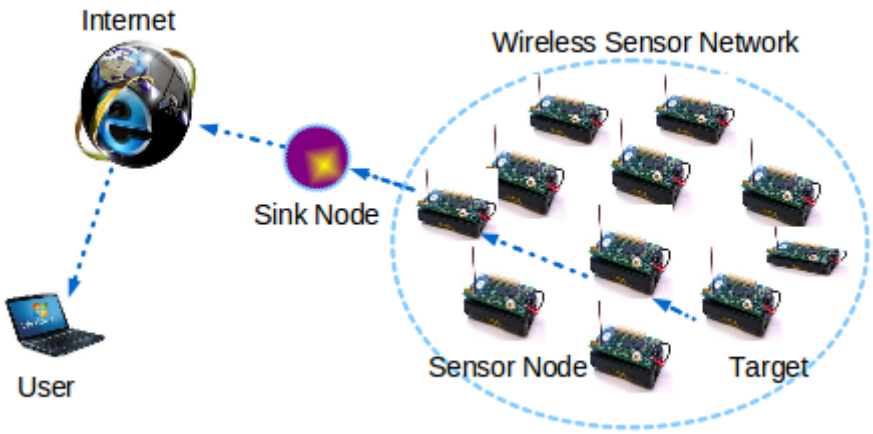
[2]Zdroj: *http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf*

Kapitola 3

Bezdrôtové senzorové siete (WSN)

3.1 Štruktúra WSN

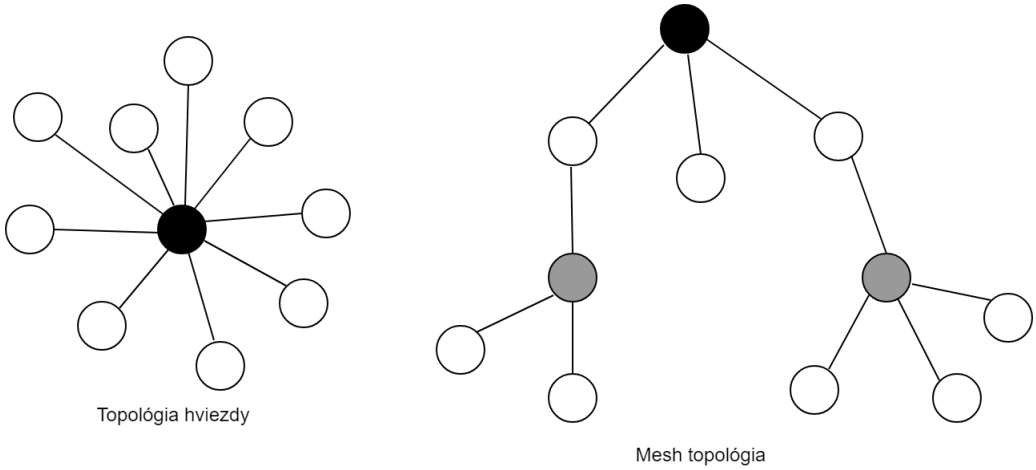
Bezdrôtová senzorová sieť (angl. *Wireless Sensor Network*, skr. WSN) je sieť vytvorená spájaním viacerých senzorových uzlov medzi sebou, za účelom vzájomnej komunikácie či distribuovaných výpočtov. Hlavnou motiváciou k tvorbe WSN je fakt, že jedna samostatná senzorová jednotka síce dokáže poskytnúť nejaké informácie z jej prostredia, ale jej pokrytie je nedostatočné pre väčšie aplikácie a dáta získané v danom bode môžu byť príliš zarušené, či z iného dôvodu nevalídne a preto je žiadané mať týchto senzorov viac, pričom tým vznikajú nové možnosti pre distribuované výpočty. Znalosti v tejto kapitole boli prebrané z [4], [1] a [11].



Obr. 3.1: Ilustrácia WSN[1]

WSN sú prepojením viacerých bezdrôtových senzorových uzlov, pričom jeden z nich sa nazýva *sink* (preklad *odtok*, v tejto práci sa však budeme držať anglického termínu). Tento uzol slúži ako prepojenie WSN s inými sieťami (teda je to brána), schopnými spracovávať informácie z WSN, prípadne ich tiež uzlom poskytovať, či uzlom určovať ich úlohu vo WSN.

[1] Zdroj:
http://microcontrollerslab.com/wp-content/uploads/2015/08/WIRELESS-SENSOR-NETWORKS.png



Obr. 3.2: Topológie bezdrôtových senzorových sietí

Sink môže byť aj nejaké samotné koncové zariadenie, ako napríklad osobný počítač, ktorý bude používaný na riadenie tejto siete.

3.2 Topológie WSN

Spôsob, akým sú vzájomne prepojené uzly vo WSN, tiež zvaný topológia, určuje akým štýlom komunikujú uzly medzi sebou navzájom a ako sa budú smerovať správy medzi uzlami. Spôsobov, akými môžu byť prepojené uzly WSN navzájom, je mnoho — prakticky každý spojitý graf, ktorý obsahuje ako vrcholy uzly WSN je valídnym spôsobom prepojenia. Nasledujúce dve popísané topológie sú uvedené na obrázku 3.2.

Jedným takýmto spôsobom prepojenia je hviezdicová topológia — jeden uzol slúži ako brána pre preposielanie správ ostatným uzlom (v obrázku je tento uzol vyfarbený na čierno). Ostatné koncové uzly (biele uzly v obrázku) nesmú vzájomne komunikovať a ak chcú si navzájom vymeniť informácie, tak ich musia poslať bráne a tá ich potom pošle ďalej. Výhodou tejto topológie je šetrenie energie, keďže je možné efektívne plánovať prebudenie uzlov zo spánku a nebránové uzly môžu byť dlhšie v režime spánku. Problémami sú nefunkčnosť siete v prípade výpadku brány a požiadavka na schopnosť prenášania informácií od uzlov k bráne, čo obmedzuje veľkosť siete, keďže uzly majú obmedzený vysielací výkon. Keďže je to dosť obmedzujúca požiadavka, tak sa táto topológia využíva len zriedka.

Obecnejším typom topológie WSN je *mesh* (anglický termín, budeme sa ho držať) topológia. V tejto topológii môžu byť uzly ľubovoľne prepojené. Mesh topológie, ktoré neobsahujú v sebe cyklické prepojenia uzlov, sa nazývajú stromovými (angl. *tree*) topológiami — hviezdicové topológie sú ich podmnožinou. Napríklad mesh topológia naznačená na obrázku 3.2 je takouto stromovou topológiou. Šedou farbou označené uzly sú tzv. agregačné uzly — sú prepojené s viac než jedným alebo dvoma uzlami. V týchto sieťach uzol, ktorému má doraziť správa, nemusí byť vo vysielacom dosahu odosielateľa a nie je jasné, ktorému uzlu sa má preposlať správa, aby dorazila čo najskôr príjemcovi. Správa sa teda posiela prostredníctvom ostatných, dostupných uzlov, čo sa nazýva multi-skok (z angl. *multi-hop*). Vzniká tým problém smerovania správ a z tohto dôvodu zohrávajú agregačné uzly dôležitú úlohu v týchto sieťach. Takéto siete môžu byť však na rozdiel od sietí s topológiou hviezdy rozložené vo väčšom priestore a ak v ich grafe obsahujú cykly tak sú aj robustnejšie voči vý-

padkom. Ďalším problémom je však to, že uzly musia byť pripravené prijímať od ostatných uzlov správy, čo zvyšuje čas so zapnutým RF vysielačom a teda aj odber energie.

Ďalej pre WSN sú charakteristické isté črty, ktoré spôsobujú znateľné prekážky pri využívaní týchto sietí. Prvým takým črtom je rozsiahlosť. Ako bolo spomenuté, senzory sú rozložené v priestore, kde sa má skúmaný jav nachádzať. Tento priestor môže byť často veľmi rozsiahly a nie len, že uzly nemusia mať všetky na seba vzájomne dosah, ale ani dva uzly nemusia mať oba dosah na nejaký jeden spoločný uzol — môžu mať nulový prienik uzlov, s ktorými sú schopné komunikovať. V tom prípade je topológia hviezdy nepoužiteľná, keďže nie je možné zvoliť žiadny uzol ako bránu, ktorá by v nej preposielala správy. Zväčšovanie siete vyžaduje buď zvyšovanie vysielacieho výkonu uzlov (nepraktické), alebo pridanie viacerých uzlov do priestoru, kam sa má sieť rozšíriť.

Druhým črtom sú dynamické zmeny topológie siete. V tomto prípade je problémom smerovanie správ — smerovanie v týchto sieťach môže byť značne komplikované, keďže uzlu nemusí byť aktuálna topológia známa (uzol má dosah len na niektoré uzly siete a nemá informácie o zvyšku). Táto neznalosť topológie je spôsobená možnými výpadkami uzlov — uzol môže byť zarušený (a teda bezdrôtovo nedostupný), či môže mať vybité batérie alebo iný typ poruchy.

Uzly WSN preto majú dve funkcie — sú súčasne aj zdrojmi informáciami, ale ich aj smerujú medzi ostatnými uzlami v sieti:

- zdroj — zber dát zo senzorových jednotiek, čo je aj ich primárnym účelom; tieto dáta sa spracujú a môžu sa napr. poslať ďalej na sink
- smerovač — uzol musí byť schopný prijímať informácie od ostatných senzorov a posielať ich ďalej iným senzorom v závislosti od ich destinácie

3.3 Lokalizácia uzlu

Jedným z riešených problémoch vo WSN je lokalizovanie uzlov alebo javov v priestore. Lokalizácia v rámci WSN je silným nástrojom, ktorý značne rozširuje použiteľnosť týchto sietí a viacero aplikácií WSN je závislých na nej. Obsah tejto sekcie bol založený na [4] a doplnený znalosťami z [5].

Jedná sa o určenie polohy objektu (či osoby, javu) relatívne voči uzlom siete vo fyzickom priestore, alebo aj absolútne určenie polohy ak sú známe referenčné body, z ktorých bola táto pozícia určená. Lokalizačné techniky sa delia do dvoch väčších skupín:

1. založené na meraní vzdialenosti
2. bez merania vzdialenosti

Prvá skupina techník, využíva určitého spôsobu merania vzdialenosti sledovaného objektu od uzlov siete, akým je napríklad sila prijímaného signálu (RSSI), uhol príchodu signálu, či čas príchodu signálu. Na základe zmeranej vzdialenosti objektu od aspoň troch uzlov siete sa dá vypočítať presná pozícia objektu pomocou techník, akými sú napríklad triangulácia, trilaterácia, či iteratívna a kolaboratívna multilaterácia. Pre výpočet pozície objektu je požadované, aby spomenuté uzly siete, z ktorých sa merala vzdialenosť objektu, mali známu pozíciu v priestore.

Druhá spomenutá skupina, techniky nevyužívajúce merania vzdialenosti, má princíp fungovania založený na znalostiach o konektivite siete. Tieto techniky zisťujú polohu objektu

na základe toho, z ktorých uzlov je možná komunikácia s týmto objektom, teda u ktorých uzlov je vo vysielacom dosahu. Takýmto technikami sú napríklad ad-hoc pozičný systém, technika približného bodu v triangulácii či lokalizácia založená na viacrozmernom škálovaní. V rámci tejto práce sa však budeme zaoberať prvou spomenutou skupinou techník, teda sa táto skupina nebude ďalej rozoberať.

3.3.1 Techniky merania vzdialenosti

Ako bolo spomenuté v predošlej sekcii, pre určenie polohy uzlu sa bude používať vypočítaná vzdialenosť uzlu od nejakých referenčných uzlov v sieti. Existuje viacero takýchto techník, ktoré sa dajú využívať a ktoré budú spomenuté v tejto sekcii.

Čas príchodu (ToA)

Čas príchodu je jedným spôsobom merania vzdialenosti (angl. *Time of Arrival*, skr. ToA). Meranie vzdialenosti založené na čase príchodu využíva znalosti, že sa signál šíri v priestore určitou rýchlosťou a teda so znalosťou oneskorenia príchodu signálu na prijímacie zariadenie sa dá vypočítať vzdialenosť medzi komunikujúcimi stanicami. Problémom tohto prístupu je fakt, že sa vyžaduje presná časová synchronizácia, inak v prípade desynchronizácie bude vypočítaná vzdialenosť nepresná.

Existuje aj upravený obojsmerný variant tohto prístupu, ktorý využíva aj spätného odosielania správy od príjemcu odosielateľovi po tom, ako príjemca prijal správu určenú pre zmeranie doby prenosu signálu. Tento prístup už nevyžaduje synchronizáciu (keďže obe komunikujúce strany ku prenášanej správe prikladajú svoje vlastné časové informácie), avšak sa zvyšuje počet prenášaných správ, potrebných pre určenie vzdialenosti.

Čas rozdielu príchodu (TDoA)

Táto technika, anglicky zvaná *Time Difference of Arrival* (skr. TDoA), funguje na základe merania doby prenosu dvoch signálov s odišnými rýchlosťami šírenia. Tieto dva signály, oba vyslané jednou stranou komunikácie (s nejakým prípadným oneskorením medzi vyslaním jednotlivých signálov) musia mať známu rýchlosť šírenia. Ako prvý sa vysiela signál s vyššou rýchlosťou šírenia. Prijímacie zariadenie najprv spracuje teda signál, ktorý sa šíri rýchlejšie, potom pomalšie sa šíriaci signál a na základe znalosti ich rýchlostí, oneskorenia medzi odoslaním jednotlivých signálov a oneskorenia príchodu jednotlivých signálov vypočíta vzdialenosť. Podobne ako u obojsmerného ToA, nie je vyžadovaná synchronizácia medzi komunikujúcimi stranami. Je však zvýšená presnosť voči ToA, ale môže byť potrebný dodatočný hardware.

Uhol príchodu signálov (AoA)

Uhol príchodu (angl. *Angle of Arrival*, skr. AoA) signálu je použiteľný na určenie pozície vzhľadom na referenčné body, pokiaľ sú známe ich pozície. Využíva sa viacero smerových antén na uzle, ktorého pozíciu chceme zistiť a sleduje sa, ktorá anténa prijala signál ktorého referenčného bodu a pod akým uhlom, pričom na základe znalosti týchto veličín je možné určiť pozíciu tohto uzlu v priestore s využitím triangulácie. Síce táto technika môže poskytnúť veľmi precízne vypočítanú lokáciu uzlu, ale pre uzly vo WSN je nepraktická vzhľadom na požiadavku na viacero smerových antén, ktoré zvyšujú cenu uzlov a ich presnosť určuje aj presnosť tejto metódy. Taktiež je táto metóda náchylná na odrazy v prostredí.

Meranie sily signálu (RSSI)

Metóda merania vzdialenosti založená na meraní sily prijatého signálu (angl. *Received Signal Strength Indication*, skr. RSSI) funguje na základe zoslabovania sily signálu so vzdialenosťou. Táto metóda je použiteľná na všetkých zariadeniach, ktoré dokážu merať silu prijatého signálu. RSSI je mapovanie výkonu na kvantizované úrovne a je závislé od použitého prijímača a teda môže byť odlišné pre rôznych výrobcov prijímačov. Friisova prenosová rovnica, z ktorej sa dá vypočítať vzdialenosť, ignoruje niektoré aspekty ktoré môžu tiež ovplyvňovať silu prijatého signálu:

$$\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi d}\right)^2$$

kde P_r značí výkon dostupný pre prijímaciu anténu, P_t značí výkon dodávaný do vysielacej antény (keby to bol izotropný žiarič — idealizovaná anténa), G značí zisk antény v smere k druhému účastníkovi (spodný index odlišuje, či sa jedná o odosielateľa t, alebo príjemcu r). Vzdialenosť je označená d a λ značí dĺžku vĺn, na ktorých sa prenášajú informácie. Aby táto rovnica platila, musí byť splnených viacero podmienok — vzdialenosť medzi anténami musí byť o dosť väčšia než je vlnová dĺžka, antény musia mať medzi sebou nezastavaný priestor pre komunikáciu (teda neberie do úvahy viaccestné šírenie signálu) a prenosové pásmo musí byť dostatočne úzke vzhľadom na vlnovú dĺžku.

3.3.2 Určenie polohy uzlu

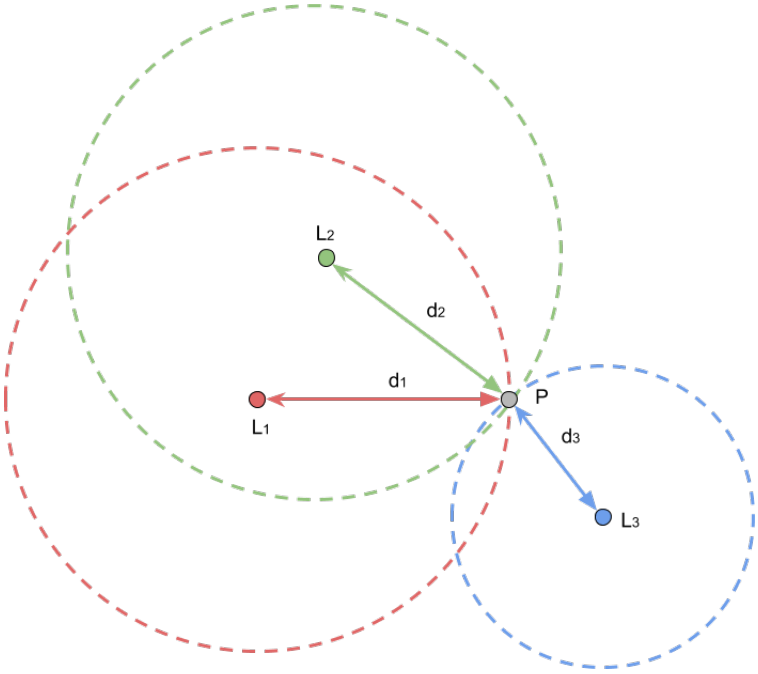
V sekcii 3.3 sme spomenuli dve skupiny metód určovania polohy uzlov, pričom bola vyradená skupina nevyužívajúca merania vzdialenosti uzlu od referenčných uzlov. V tejto sekcii budú podrobnejšie rozpísané metódy určenia polohy.

Triangulácia

Triangulácia využíva metódu merania uhlu príchodu signálov (AoA) z niekoľkých (troch) referenčných bodov. Pre uzol, ktorého pozícia sa určuje, sa vypočítajú uhly z ktorých boli prijaté signály od referenčných bodov, a teda sa predpokladá umiestnenie jednotlivých referenčných bodov v daných smeroch od lokalizovaného uzlu. Ďalej sa smery umiestnenia týchto referenčných bodov využívajú k prekladaniu lokalizovaného bodu s pármi zvolených referenčných bodov pomocou trojuholníkov, ktoré je možné tvoriť vzhľadom na to že rozdiel uhlov príchodu signálov jednotlivých ref. uzlov je možné vypočítať a pozície daných ref. uzlov sú známe. Nakoniec sa na základe vypočítanej pozície tretieho vrcholu (lokalizovaného uzlu) v trojuholníkoch určí pozícia lokalizovaného uzlu. Ako však bolo spomenuté v predošlej sekcii, metóda uhlu príchodu signálov nie je praktická pre využitie vo WSN. Taktiež táto metóda je veľmi chybová v prostredí, kde medzi používanými uzlami sú nejaké prekážky.

Trilaterácia

Na rozdiel od triangulácie využívajúcej uhlov referenčných bodov, trilaterácia využíva vzdialenosť referenčných uzlov od lokalizovaného uzlu. Táto vzdialenosť mohla byť vypočítaná napríklad pomocou ToA, TDoA, alebo RSSI. Zmeraná vzdialenosť od referenčných uzlov sa využije ako polomer pri tvorbe kružníc (v 2D priestore, v 3D priestore sú to gule) so stredmi v referenčných uzloch. To je znázornené na obrázku 3.3, kde body L_i sú referenčné



Obr. 3.3: Trilaterácia[2]

body, d_i sú zmerané vzdialenosti a P je hľadaný bod. Následne sa hľadaná pozícia uzlu nachádza v prieniku vytvorených kružníc (resp. gulí). Pre ukážku je za využitia Pytagorovej vety a preloženia pozícií referenčných uzlov rovinou vypočítaná pozícia uzlu v 3D priestore získaná pomocou rovníc:

$$r_1^2 = x^2 + y^2 + z^2$$
$$r_2^2 = (N_x - x)^2 + y^2 + z^2$$
$$r_3^2 = (O_x - x)^2 + (O_y - y)^2 + z^2$$

kde $P(x, y, z)$ je hľadaná pozícia uzlu v trojrozmernom priestore a $M(0, 0, 0)$, $N(N_x, 0, 0)$ a $O(O_x, O_y, 0)$ sú pozície jednotlivých referenčných bodov v ich spoločnej rovine. Úpravami týchto rovníc získame súradnice hľadaného bodu:

$$x = \frac{r_1^2 - r_2^2 + N_x^2}{2N_x}$$
$$y = \frac{r_1^2 - r_3^2 + O_y^2 + O_x(O_x - 2x)}{2O_y}$$
$$z = \pm\sqrt{r_1^2 - x^2 - y^2}$$

Z rovníc je možné vidieť, že vznikajú dve súradnice na osi z. Jedno riešenie bude správnou pozíciou uzlu a druhé riešenie je falošné. Toto falošné riešenie je možné eliminovať pomocou využitia štvrtého referenčného bodu, ktorý nebude ležať v rovine so zvyšnými troma, alebo elimináciou dimenzie z a lokalizovaním v 2D priestore. Každopádne, takto

[2]Zdroj: *https://www.alanzucconi.com/wp-content/uploads/2017/03/Trilateration3.png*

vypočítaná pozícia je platná pre zvolenú rovinu referenčných uzlov a je ešte potrebné ju transformovať naspäť do bežne využívaného súradnicového systému.

Tento spôsob riešenia však zanedbáva hocijakú nepresnosť, čo sa v praxi môže prejaviť ako problematické — môže napríklad nastať situácia, kedy je nulový prienik kružníc. Existuje však mnoho riešení problému trilaterácie, s odlišnými spôsobmi prístupu — uvedený bol geometrický prístup, avšak využívajú sa v praxi aj riešenia založené na strojovom učení a pod. (napríklad [14]). Vzhľadom na relatívne nízke výpočetné schopnosti bežných uzlov WSN je však vhodné preferovať menej náročný spôsob výpočtu, aj za cenu možného mierneho zvýšenia chybovosti vypočítanej lokácie.

3.3.3 Problémy lokalizácie

Lokalizovanie objektov je komplexná činnosť, ktorá sa spolieha na viaceré dostupné nástroje, ktoré môžu do výpočtov prinášať chyby, alebo spôsobovať problémy iným spôosobom. V tejto sekcii budú spomenuté niektoré z nich.

Prvým problémom je presnosť merania. Presnosť merania je problematická z toho dôvodu, že sa chyba merania prenáša do výpočtov a tam má zvýšený efekt na vypočítanú vzdialenosť. Zdrojov chýb merania môže byť viacero. Bežným problémom sú chyby merania spojené so šírením signálu — v ceste šírenia signálu priamo od zdroja k cieľu (vzdušnou čiarou) je nejaká prekážka, a teda sa šíri signál odrazmi a rôzne materiály môžu spôsobovať rôzny útlm signálu. To spôsobuje, že nameraná sila signálu nemusí správne reprezentovať vzdialenosť dvoch účastníkov komunikácie.

Signál sa teda môže šíriť viaccestne, čiže môže dojsť prenášaná správa viackrát (rozličnými trasami), s inou dobou trvania prenosu správy. To značí, že prostredie (resp. jeho komplexnosť), v ktorom je WSN, má značný vplyv na negatívne vplyvy prostredia na meranie vzdialenosti. Ďalším zdrojom chýb merania sú napríklad lacné komponenty (používané pre zníženie ceny uzlov), ktoré jednak merajú silu prijatého signálu s menšou presnosťou, ale majú aj väčšiu náchylnosť na výpadky a iné poruchy.

S chybovosťou merania je spojený problém *Dilution of Precision* (angl., skr. DOP), ktorá spôsobuje že aj pri menších chybách merania vzdialenosti sa táto chyba prejaví násobne vo vypočítanom výsledku, čo môže spôsobiť nulový prienik kružníc, alebo naopak prienikom bude celý útvar v dvojrozmernom či trojrozmernom priestore, v ktorom sa môže hľadaný uzol nachádzať. Pre malé merané rozsahy však DOP nespôsobuje veľké problémy, prejavuje sa v značne vyššej miere pri meraní veľkých vzdialeností (napr. GPS).

Ďalším problémom je obmedzená výpočetná sila uzlov. Síce sa dajú niektoré algoritmy aplikovať v distribuovanej podobe, kde sa znižujú pamäťové a výpočetné nároky pre jednotlivé uzly, ale tým aj stúpa využitie sieťovej komunikácie, keďže uzly vykonávajúce tieto výpočty musia komunikovať navzájom. Taktiež sa pri meraní vzdialenosti viacerými uzlami síce spresňuje vypočítaná pozícia objektu, ale aj sa tým zvyšuje výpočetná náročnosť.

Kapitola 4

TinyOS

TinyOS (skr. TOS) je operačný systém (skr. OS) určený špecificky pre bezdrôtové senzory. Vzhľadom na vysoké nároky na šetrenie elektrickej energie v bezdrôtových sieťach nie sú bežné OS pre osobné počítače či mobilné telefóny praktické a teda bolo potrebné vytvoriť OS, ktorý bude stvorený s dôrazom na mimoriadne efektívne manipulovanie so značne obmedzenými zdrojmi. TinyOS sa teda snaží šetriť elektrickú energiu najviac ako to je len možné a väčšinu času strávia senzorové uzly v režime spánku pre maximalizáciu šetrenia energie. Obsah tejto kapitoly je prebraný z [9].

TinyOS je napísaný v jazyku nesC, ktorý bude popísaný v nasledujúcej sekcii. Taktiež sa v nesC píšu aj aplikácie pre TinyOS. TInyOS poskytuje na vyššej úrovni tri nástroje, ktorými sa snaží zjednodušiť tvorbu aplikácií:

- model komponentov — definuje ako písať malé znovupoužiteľné kusy kódu (tj. komponenty), ktoré sa následne dá skladať dokopy pri riešení väčších problémov
- konkurentný model vykonávania — definuje ako sa prekladajú výpočty komponentov a interakciu kódu prerušení s bežným kódom
- aplikačné rozhranie (API), služby, knižnice komponentov

Model komponentov je založený na princípe definovania závislostí komponentov, pričom sa už ďalej nerieši konkrétna implementácia danej závislosti. To umožňuje znovupoužiteľnosť komponentov. Tento spôsob definovania komponentov pochádza z jazyka nesC.

TinyOS neobsahuje vlákna. Taktiež TinyOS využíva jediný zásobník, pretože sa využívajú callbacky namiesto blokovania pri I/O operáciách (hneď po I/O volaní sa vracia späť požiadavka, callback sa zavolá po ukončení). Tento spôsob fungovania sa nazýva *split-phase*. V TinyOS existujú úlohy (angl. *tasks*), umožňujúce odložené volanie procedúr, ktoré môže hocijaká komponenta vyvolať. Tieto úlohy majú tendenciu byť vykonané dostatočne skoro po ich naplánovaní, vzhľadom na nízke obvyklé využitie CPU u takýchto zariadení s nízkym výkonom. Keďže sa tieto úlohy nemôžu preemptívne plánovať, je vylúčený problém *race condition* ktorý môže vzniknúť pri súčasnom viacnásobnom prístupe k nejakým zdrojom v pamäti. Výnimkou je kód prerušení, u ktorého tieto problémy aj tak môžu nastať (kompilátor v takom prípade vydá varovanie pri preklade). Všetky tieto vlastnosti TinyOS umožňujú súčasné vykonávanie mnohých úloh rôznych komponentov, s nízkou náročnosťou na RAM — to je možné vďaka konkurentnému modelu vykonávania.

Tretím poskytovaným nástrojom, poskytovaným prakticky každým operačným systémom, sú služby a aplikačné rozhranie OS. Jedná sa o celé spektrum poskytovaných služieb,

prítomných na senzorových uzloch, ktoré môžu byť potrebné pri vykonávaní požadovanej činnosti. Takýmito službami sú napríklad sieťová komunikácia, spracovanie dát zo senzorov či reagovanie na udalosti vyvolané komponentmi.

4.1 nesC

nesC je rozšírenie jazyka C, ktoré vzniklo primárne pre tvorbu TinyOS a programovanie senzorových uzlov, ktoré TinyOS využívajú. Spôsob programovania v nesC je však výrazne odlišný v porovnaní s programami vykonávajúcimi ekvivalentnú činnosť v C. Kým C využíva premenné, funkcie a dátové typy, nesC skôr modeluje prepojenie jednotlivých komponentov — využívané rozhrania, udalosti s nimi spojené a reakcie na ne a volania rozličných úloh, ktoré je potrebné vykonať pre splnenie funkcionality komponentu.

Pri programovaní v nesC sa využívajú termíny ako príkazy (angl. *commands*), rozhrania (angl. *interface*), udalosti (angl. *events*) používateľ rozhrania (angl. *user*), či poskytovateľ rozhrania (angl. *provider*), kde:

- komponenty implementujú konkrétne služby (napr. zapnutie LED) a sú základným stavebným kameňom TinyOS aplikácií
- rozhrania definujú funkcionalitu, ktorú musí niektorý komponent poskytovať
- udalosti nastávajú v komponente po zavolaní príkazov niektorého z implementovaných rozhraní
- príkazy sú volané napr. udalosťami pri ich obsluhe a odosielané na rozhranie iných komponentov, kde vyvolávajú udalosti
- používateľ rozhrania volá príkazy u poskytovateľa rozhrania, formou požiadaviek
- poskytovateľ rozhrania vykonáva callbacky používateľovi rozhrania, teda signalizuje udalosti

V praxi sa tieto spomenuté rozhrania chovajú obdobne, ako aj rozhrania v jazyku Java. Pomocou nich sa definuje komunikácia komponentov (na rozdiel od C, kde sa funkcie volajú priamo). V konečnom dôsledku sú však príkazy a udalosti prakticky ako bežné funkcie (dokonca môžu obsahovať kód písaný v C) — signalizovanie udalosti či volanie príkazu sú opäť len volaniami funkcií.

Kapitola 5

Agenti a multiagentné systémy

V tejto kapitole bude stručne popísaný princíp, na základe ktorého fungujú agentné systémy a taktiež bude popísaný agentný jazyk ALLL a platforma WSageNt, ktorá ho využíva. Hlavným zdrojom informácií v tejto kapitole je [13].

5.1 Agent

Agent (odvodené z latinského slova *agentum* = konať) je entita, ktorá úmyselne vykonáva nejakú činnosť za účelom dosiahnutia určitého cieľu v prospech svojho majiteľa, respektíve klienta. Táto definícia, ktorá je dosť obecná, však nevystihuje úplne také podstatné črty, ktoré by mali umelého agenta (tj. neživého agenta, programovo vytvoreného) definovať. Umelý agent by mal byť racionálny — v daný okamih by mal voliť také akcie, ktoré podľa jeho aktuálnych znalostí o prostredí povedú k výsledkom, ktoré budú čo najviac viesť k cieľom, ktoré by mal splniť.

Podľa Jenningsovej klasifikácie by mal každý racionálny agent spĺňať niekoľko vlastností. Tieto vlastnosti by agenta mali odlišovať od iných aktívnych prvkov systému. Jedná sa o:

- autonómnosť — agent by mal byť schopný samostatne a nezávisle jednať na základe svojich vlastných rozhodnutí, bez nutnosti vstupu iných entít ktoré by mu dávali pokyny
- reaktivita — agent by mal byť schopný pohotovo a adekvátne reagovať na možné zmeny prostredia a prípadnú voľbu jeho akcií meniť v závislosti od aktuálneho stavu prostredia
- proaktivita — agent sa chová iniciatívne so zámerom splniť svoj cieľ, ku ktorému chce dospieť, teda aj mení prostredie v súlade s plánom vedúcim k splneniu jeho cieľov
- sociálne schopnosti — agent je schopný komunikovať s inými agentmi, ktorí sa môžu vyskytovať v danom systéme a poprípade je schopný riešiť konflikty s nimi, alebo s nimi spolupracovať

Od agentov sa očakáva, že budú rozhodovať prakticky a nie teoreticky. Rozdiel spočíva v tom, že teoretické rozhodovanie spočíva v rozhodovaní podľa toho čo je pravda a čo nie je pravda, kým praktické rozhodovanie je založené na tom aké treba vykonať akcie, ktoré by zaistili aby sa niečo stalo pravdou. Praktické rozhodovanie má dve časti:

- zvažovanie (angl. *deliberation*) — volí sa ním cieľ, ktorý sa má dosiahnuť a vzniká ním zámer
- plánovanie (angl. *planning*) — volí sa akcia, alebo sa zostavuje plán s vedomím, že je možné dosiahnuť zámeru, ktorý vznikol zvažovaním

Spomenutý zámer je mentálnym stavom agenta a modeluje jeho proaktívnu snahu dosiahnuť nejaký dlhodobý cieľ aj napriek tomu, že aktuálny stav systému to neumožňuje (napríklad konanie agenta to mohlo znemožniť, alebo sa dynamicky zmenilo prostredie). Zámer je teda zadaním problému, ktorý chce vyriešiť — verí, že je možné zámer splniť (a teda aj že sa niekedy priblíži k jeho dosiahnutiu), taktiež neverí že nie je možné sa priblížiť k jeho dosiahnutiu, aj keď nemusí mať v úmysle všetky vedľajšie účinky, ktoré jeho snaha a akcie zvolené k dosiahnutiu zámeru mohla spôsobiť.

Pre dosiahnutie zvoleného zámeru môže byť potrebný plán. Plán je usporiadaná postupnosť akcií, ktorá povedie k splneniu zámeru. Plány je možné vytvoriť plánovačmi, alebo je možné použiť preddefinované plány — plány sa ukladajú do knižnice plánov a nemusia byť použité hneď po ich vytvorení, je možné ich odložiť do knižnice a zvoliť daný plán v prípade potreby. Plány je možné tvoriť aj pre celú skupinu agentov, kedy sa už berú do úvahy aj zmeny prostredia spôsobené všetkými agentmi, s ktorými tento plán počíta a môže sa teda využívať spolupráce agentov.

BDI agenti sú agenti s určitými znalosťami a mentálnymi stavmi, na základe ktorých sa riadia. Skratka BDI znamená v angličtine *Belief-Desire-Intention*. *Belief* (viera) určuje, že agent má nejakú množinu znalostí, o ktorých verí, že sú pravdivé (napríklad znalosti o prostredí). *Desire* (túžba) vyjadruje nejaké priania, ktoré by chcel agent naplniť (teda jeho ciele). *Intention* (zámer) je už skôr spomenutý zámer k dosiahnutiu nejakého cieľa.

5.2 Agentný a multiagentný systém (MAS)

Agentný systém je systém obsahujúci agenta, ktorý je v ňom schopný jednať. Takýto systém okrem agenta má definované prostredie, v ktorom agent vystupuje a ktoré dokáže meniť — v takomto systéme sú definované akcie agenta, ich vplyv na prostredie, vnemy agenta získané z prostredia, či funkcia určujúca chovanie agenta na základe jeho vnemov. Prostredie systému sa dá rozdeliť podľa určitých vlastností:

- spojité/diskrétne — v závislosti od časovej množiny
- dynamické/statické — ak sa môže prostredie zmeniť samovoľne aj bez akcie agenta tak sa jedná o dynamické prostredie
- deterministické/nedeterministické — prostredie je deterministické, ak by za určitého stavu malo vykonanie nejakej akcie stále rovnaký výsledný stav

Multiagentný systém je variantom agentného systému, kde sa nenachádza len jeden agent, ale viacero agentov. Takýto systém poskytuje značne rozšírené možnosti využitia agentného prístupu k riešeniu problémov, avšak prináša aj nové problémy spojené so súčasnou prítomnosťou viacerých agentov v rovnakom prostredí. Agenti v takomto systéme môžu vzájomne komunikovať, spolupracovať, zdieľať informácie o prostredí, vykonávať spoločné plánovanie, či dávať si záväzky. Vzniká však aj problém konfliktov medzi agentmi (agenti vykonávajú činnosti, ktorými si navzájom znemožňujú činnosť či inak škodia), potreby skupinového plánovania, nekonzistencie databází znalostí a iné.

5.3 Agent Low Level Language (ALLL)

Agent Low Level Language je agentný jazyk, ktorý bol vytvorený doc. Ing. Františkom V. Zbořilom, Ph.D. a doc. Ing. Františkom Zbořilom, Ph.D. v rámci článku o simuláciách vo WSN. Tento jazyk vznikol s úmyslom skombinovania schopností jazykov určených pre distribuované systémy a spravovania procesov, prítomných v už predtým existujúcich riešeniach ako napríklad Chi systém. ALLL by mal ešte poskytovať ďalšiu funkcionalitu v porovnaní s už existujúciami systémami, ako napríklad spracovanie výnimiek, či klonovanie a mal by fungovať s ohľadom na vysokú dynamickosť prostredia. Obsah tejto sekcie je kompiláciou informácií poskytnutých v [6], [8] a [12].

Samotná štruktúra programov písaných v ALLL je podobná iným programovacím jazykom — program je postupnosť tzv. viet, pričom každá veta sa skladá z určitej postupnosti akcií. Vety predstavujú možné plány agenta — ak zlyhá nejaká akcia plánu, tak zlyhal aj celý plán. Plány môžu tvoriť hierarchickú štruktúru — plán môže volať podplány a môže byť podplánom iných plánov. Po ukončení vykonávania plánu sa vracia vykonávanie plánu vyššej úrovne — ak bol splnený plán na najvyššej úrovni, tak sa považuje cieľ za splnený.

Obecný formát akcie je `<operátor>(<parametre>)`. Operátor určuje typ akcie. Parametrami akcie môžu byť rôzne položky, závislé od typu danej akcie, avšak aj pre jednu akciu môže byť viacero legálnych kombinácií parametrov. Vzhľadom na to, že sa v rámci tejto práce bude používať neskôr popísaná platforma WSageNt, ktorá využíva upravenú syntax jazyka ALLL, bude tu uvedená táto pozmenená syntax namiesto pôvodnej syntaxe. Pôvodnú syntax je možné vidieť v [8].

Agent v ALLL, tak ako bol navhnutý v [12], má 7 častí: báza znalostí (inak zvaná *Belief Base*), knižnica plánov (angl. *Plan Base*), zámer (angl. *Plan*), báza vstupov (angl. *Input Base*, obsahujúca správy obdržané od platformy) a 3 univerzálne registre.

Operátor	Parametre	Význam
+	n-tica | register	Pridanie do bázy znalostí
-	n-tica | register	Odobranie položiek z bázy znalostí
!	číslo | register n-tica | register	Odoslanie správy zadané n-ticou alebo registrom na mote so zadanou adresou
?	číslo | register	Test bázy vstupov na správu od mote (resp. senzoru) so zadanou adresou
@	zoznam akcií	Priame spustenie, akcia sa vloží na zásobník so zarážkou
^	meno | register	Nepriame spustenie, hľadá sa plán v knižnici plánov s rovnakým menom
&	číslo	Zmena aktívneho registra
*	n-tica | register	Test bázy znalostí na zadanú n-ticu alebo register, výsledok sa uloží do aktívneho registra
$	písmeno {n-tica | register}	Volanie služieb platformy, popísané neskôr
#	žiadne	Zarážka za plánom, sémanticky bez významu

Tabuľka 5.1: Tabuľka akcií v ALLL

V rámci parametrov sa môžu v akciách vyskytovať dva špeciálne symboly: `&` a `_` (podčiarkovník). Symbol `&`, nasledovaný číslom od 1 do 3 sa využíva k nahradeniu tejto dvojice znakov obsahom registra s číslom, ktoré je uvedené za `&`. Toto sa nazýva substitúcia registrov. Symbol `_`, zvaný tiež anonymná premenná, sa využíva pri unifikácii — to znamená,

že sa hľadajú všetky možné legálne hodnoty, ktoré sa môžu vyskytovať na danom mieste, a aplikuje sa na ne zvolená akcia.

5.3.1 Sémantika akcií

V tejto sekcií bude popísaná sémantika akcií, ktoré sú uvedené v tabuľke 5.1. Popis významu týchto akcií je prebraný z bakalárskej práce p. Spáčila [12], pričom tu sú uvedené v stručnejšej forme.

- `+(abc, def)` — pridanie znalosti (abc, def) do báze znalostí (ak tam už bola, tak sa nevykoná nič), využitie unifikátoru nemá zmysel ale je možné použiť registre
- `-(abc)` — odstránenie znalosti (abc) z báze znalostí, je možné využiť unifikáciu aj registre
- `!(10, (hello))` — odošle správu „hello“ na zariadenie s označením 10, je možné použiť registre (ak sa má príjemca správy určiť registrom, tak to musí byť jednoprvková n-tica) ale nie unifikáciu
- `?(1)` — prečíta správu (sú vo forme `(adresa, obsah)`) od odosielateľa s označením 1 a uloží jej obsah do aktívneho registra, dá sa použiť aj registre pre určenie odosielateľa a anonymnú premennú na unifikáciu (tá však vyberie len prvú dostupnú správu)
- `@(+xyz)` — priame vykonanie plánu, na zásobník sa pridá zarážka ak tam nebola a vykoná sa tento plán; je to priamo určená postupnosť akcií
- `^(plan)` — nepriame vykonanie plánu, vykoná sa plán z knižnice plánov s uvedeným menom; je možné použiť registre
- `&(1)` — zmena aktívneho registra na register 1, pri zmene sa aj zmaže jeho obsah
- `*(xyz)` — test bázy znalostí pomocou unifikácie, v prípade že sa nepodarila unifikácia tak akcia končí chybou, inak sa do registra uloží n-tica nájdených unifikovaných zhôd; je možné použiť register aj anonymnú premennú (tá zastupuje ľubovoľný prvok v hľadanej n-tici); pri nájdení viacerých zhôd bude ich poradie v registri opačné
- `$(<písmeno>,(parametre))` — volanie služieb platformy, ktoré sa odlišujú použitým písmenom:
  - `$(a)` — aktivovanie sledovania prichádzajúcich správ pri zapnutom interprete
  - `$(f, (zoznam | register))` — vloží do aktívneho registra prvý prvok zoznamu, resp. prvý zoznam (ak je ich viac)
  - `$(k)` — zastavenie činnosti interpretu
  - `$(l, (zoznam | register))` — ovládanie LED diód, obsahuje znak farby (r/g/y) a voliteľne či má sa zapnúť alebo vypnúť (bez neho sa stav prepne)
  - `$(m, (zoznam | register))` — skopíruje kód agenta na platformu určenú parametrom, vykonáva sa následne paralelne ak nebol za označenie cieľového uzlu pridaný parameter `s`
  - `$(r, (zoznam | register))` — vloží do aktívneho registra zvyšok zoznamu daného parametrom, bez prvého prvku

  - `$(s)` — zastaví prevádzanie kódu kým nepríde cez RF rozhranie správa
  - `$(w, (zoznam | register))` — zastaví prevádzanie kódu na dobu danú parametrom
  - `$(d, (žiadne | zoznam | register))` — meranie teploty, bez parametru sa zmeria aktuálna teplota, inak sa zadáva typ hodnoty (a = priemer, m = minimum, M = maximum) a počet hodnôt, z koľkých sa výsledok má vypočítať

- `#` — zarážka značí koniec vnoreného plánu, nevykonáva sama o sebe žiadnu činnosť a v prípade zlyhania tohto plánu sa zmažú všetky akcie plánu až po túto zarážku (vrátane nej)

5.4 WSageNt

WSageNt je platformou, postavenou nad TinyOS, ktorá je schopná interpretovať jazyk ALLL. Bola vytvorená Ing. Janom Horáčkom v jeho diplomovej práci [6]. Táto platforma využíva interpretu jazyka ALLL napísaného v bakalárskej práci Ing. Pavla Spáčila a poskytuje spravovanie zdrojov a riešenie ďalších aspektov, ktoré je nutné riešiť u bežných aplikácií bežiacich na TinyOS. Jedná sa napríklad o synchronizáciu, spracovanie vstupov, či zasielanie správ. Znalosti o platforme WSageNt pochádzajá z [6].

WSageNt bola vytvorená s dôrazom na niekoľko vlastností, ktoré mala spĺňovať:

- Nahrávať, alebo mazať agentov z/do motov (tj. mobilita agentov)
- Posielať a prijímať správy (komunikácia agentov)
- Interpretovať kód agentov napísaný v jazyku ALLL
- Byť implementovaná v TinyOS v2 pre budúcu kompatibilitu
- Byť open-source a spustiteľná v simulátore TOSSIM

Takéto riešenie vlastnej platformy nad TinyOS bolo zvolené z toho dôvodu, že samotné motes neumožňujú nahrávanie programu cez bezdrôtovú sieť priamo a bolo nutné používanie basestation pre to — na rozdiel od toho je však už interpret schopný kedykoľvek meniť agentný kód, ktorý by mal vykonávať.

Samotná platforma spravuje komunikáciu so systémovými modulmi, interpretu ďalej umožňuje fungovať na vyššej úrovni abstrakcie a riadi jeho chod pomocou rady signálov, ktoré vydáva keď nastanú určité udalosti — inicializácia, prevádzanie jedného kroku a informovanie o zmene informácií o okolí.

V konečnom dôsledku však je podstatná samotná agentná platforma. Platforma by mala byť izolovaná od samotného interpretu, aby medzi nimi neboli príliš úzke väzby. Ako bolo spomenuté, táto platforma poskytuje väčšinu funkcií interpretu v abstraktnejšej forme a riadi beh interpretu, pričom berie do úvahy aj možnosť, že sa interpret môže uspať. Agentná platforma by mala ponúkať:

- zasielanie správ — je nutná dĺžka viac než 28 bajtov, ktorá bola bez nej značným obmedzením
- zasielanie kódu agenta — prakticky tiež zasielanie správ, len s iným typom dát
- ukladanie správ do bázy vstupov (a aj uloženie kódu agenta)

- správu systémových prostriedkov a prideľovanie pamäti
- rozličné algoritmy pre prácu so zoznamami (napríklad z jazyka LISP)
- rozhranie pre prácu so senzormi
- sadu dodatočných služieb — získanie zoznamu susedných uzlov v sieti a podobne

Okrem toho všetkého ešte WSageNt obsahuje aj Control Panel. Je to webová aplikácia, ktorá umožňuje manipulovať s agentmi vo WSN, ako napríklad pridávať či odoberať motes do tejto siete, posielať správy a podobne.

Kapitola 6

Návrh riešenia

Pri navrhovaní spôsobu riešenia tohoto problému sme sa rozhodli, že sa zameriame špecificky na navigáciu v statických bezdrôtových sieťach, keďže riešenie tohoto problému pre dynamické siete by bolo rozsahom nad rámec jednej diplomovej práce. Vzhľadom na značne obmedzenú výpočtovú schopnosť uzlov WSN je taktiež aj dosť obmedzená množina algoritmov použiteľných pri riešení tohoto problému. Hlavnými obmedzeniami návrhu sú neprítomnosť FPU[1] a malá pamäťová kapacita uzlov, najmä z hľadiska pamäte RAM, v ktorej sa ukladá interpretovaný agent. Navrhli sme teda riešenie, ktoré bude pracovať na základe viacerých agentov (jednotliví agenti budú tým pádom menší) a je založené na smerovacej tabuľke, ktorú budú obsahovať statické uzly siete, v ktorej sa bude zvolený pohyblivý uzol navigovať.

6.1 Smerovacia tabuľka

Samotná smerovacia tabuľka bude obsahovať záznamy pre každý uzol WSN, ktorý je v danej sieti statický a teda ku ktorému je možné navigovať nejaký uzol. Pre každý statický uzol budú teda uložené v smerovacej tabuľke dva údaje:

- **next-hop adresa** — adresa prvého uzlu, ktorý je na najkratšej ceste k cieľovému uzlu
- **metrika** — súhrnné ohodnotenie ceny cesty k cieľovému uzlu

Takáto smerovacia tabuľka teda obsahuje len lokálne informácie o cestách k jednotlivým uzlom siete — ku ktorému uzlu treba ísť, aby sme šli najkratšou cestou k zvolenému cieľu. Vzhľadom na to, že táto tabuľka by mala byť uložená aj naprieč vypnutiami a zapnutiami uzlu siete, nebude dostatočné ukladať tieto informácie do pamäte RAM a je potrebné ukladať tieto informácie do perzistentného úložiska dát daného uzlu.

Dáta smerovacej tabuľky sa teda z tohto dôvodu budú ukladať do pamäte flash. Z tejto pamäte je možné čítať dáta po blokoch. Pre jednoduchosť implementácie a rýchlosť získavania záznamov smerovacej tabuľky z pamäte bude smerovacia tabuľka obsahovať len jediný záznam — záznam o najkratšej ceste. Zjednodušenie získavania záznamov o ceste k nejakému uzlu bude spočívať vo využití identifikačného označenia (adresy) cieľového uzlu ako offsetu do pamäte vyhradenej pre ukladanie dát smerovacej tabuľky.

[1] FPU = Floating Point Unit

Takto navrhnutá smerovacia tabuľka má viacero vlastností, ktoré je potrebné brať do úvahy pri použití v nejakej bezdrôtovej senzorovej sieti. Konkrétne sú to nasledujúce vlastnosti:

- nie je nutné sekvenčne prechádzať všetky záznamy smerovacej tabuľky – vyplýva to z využitia adresy destinácie ako offsetu do pamäti, avšak to prináša niekoľko ďalších vlastností
- nie je potrebné ukladať adresu destinácie do smerovacej tabuľky
- najväčšia možná adresa uzlu, ku ktorému je možné uložiť smerovacie informácie, je daná veľkosťou vyhradenej pamäte pre smerovaciu tabuľku – adresa uzlu nemôže byť väčšia než je počet blokov, ktoré je možné uložiť do pamäti
- využitie pamäti smerovacej tabuľky je značne závislé od adries uzlov v sieti – najsúvislejšie využitie pamäti nastane v prípade, ak sú uzly adresované sekvenčne, čo však je aj nevýhodou vzhľadom na obmedzený počet zápisov do pamäti flash
- nie je možné uložiť viac než jeden záznam o ceste k nejakému uzlu – narušilo by to využívanie adresy uzlu ako offsetu
- uložená je len znalosť o najkratšej ceste k cieľovému uzlu – v prípade výpadku niektorého uzlu na najkratšej ceste k cieľu nebude fungovať navigácia správne, keďže je známa len adresa ďalšieho suseda a cena danej cesty

6.1.1 Vyhodnocovanie cesty

Ako bolo spomenuté v sekcii 3.3.1, RSSI vyjadruje silu prijatého signálu od nejakého zdroja (napríklad odosielateľa správy). Jedná sa o číselnú hodnotu, ktorej vyššia hodnota znamená vyššiu silu signálu. Čím je teda vyššia hodnota RSSI, tým sú dva uzly k sebe bližšie. To však prináša problémy pri použití nemodifikovaného RSSI ako metriky dĺžky cesty. Ak by sa za najkratšiu cestu napríklad považovala cesta s najvyššou sumou RSSI po ceste, tak algoritmus by mal tendenciu robiť cykly, aby sa čo najviac navýšila metrika cesty, lebo by to bolo vyhodnotené ako kratšia cesta. Alebo ak by sa za najkratšiu cestu považovala cesta s najvyšším priemerom RSSI, tak algoritmus by sa snažil robiť ľubovoľný počet čo najkratších skokov, aby bolo čo najvyššie priemerné RSSI skokov.

Preto sa bude využívať pre ohodnotenie dĺžky cesty invertované RSSI, ku ktorému bude pričítaná nejaká konštanta. Touto konštantou by mohla byť napríklad hodnota 255, ktorá predstavuje maximálnu možnú hodnotu RSSI. Ak dĺžka cesty bude rovná sume takto modifikovaných hodnôt RSSI, tak sa eliminuje problém cyklov pri používaní sumy samotných hodnôt RSSI po ceste a čím bude suma invertovaného RSSI nižšia, tým bude cesta kratšia (ako to býva u väčšiny algoritmov pre hľadanie najkratšej cesty v grafe).

6.2 Návrh agentov

Samotné riešenie uvedeného problému agentmi bolo rozdelené na dve časti — konfigurácia siete bezdrôtových uzlov a navigácia v takto nakonfigurovanej sieti. Tieto dve časti budú reprezentované tromi osobitnými agentmi, ktorí budú plniť svoju predurčenú rolu. Konfiguráciu siete bude vykonávať agent, ktorého ďalej budeme nazývať **konfigurátor**. Navigovanie

v sieti budú vykonávať dvaja agenti – jeden bude pracovať na statických uzloch siete, z ktorých bude poskytovať na požiadanie informácie o smere k cieľu. Tento agent bude zvaný **smerovač**. Požiadavky na smerovača bude posielať uzol, ktorý bude navigovaný k zvolenému cieľu – bude to teda pohyblivý uzol, ktorý (pre správnu funkčnosť) sa nepodieľal pri konfigurácii siete – tento agent sa bude nazývať **navigátor**. Pôvodne boli síce smerovač a navigátor jedným agentom, avšak vzhľadom na takmer disjunktnú množinu využívaných plánov boli rozdelení pre zmenšenie pamäťovej náročnosti agentov.

Pri komunikácii si agenti budú posielať správy v jednotnom formáte, obsahujúcom typ akcie, ktorý požadujú aby príjemca vykonal a k tomu prípadne nejaké ďalšie informácie špecifické pre danú akciu. Taktiež bude súčasťou správy aj identifikácia odosielateľa, keďže sa síce pri prijatí správy uloží prijatá správa do bázy vstupov s identifikáciou odosielateľa v podobe dvojíc *(adresa odosielateľa, správa)*, ale pri čítaní správy z bázy vstupov sa táto informácia stráca a do aktívneho registra sa vkladá len samotný obsah správy. Výsledný obsah správ bude štrukturovaný teda takto:

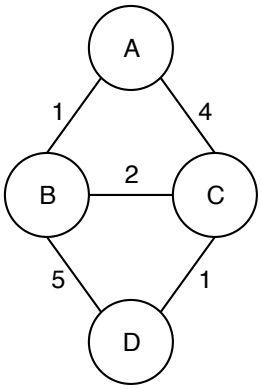
(	Typ akcie (označený znakom)	,	Adresa odosielateľa	,	Parametre akcie	)
(	`t`	,	`(1)`	,	`(5)`	)

Tabuľka 6.1: Štruktúra správy od uzlu 1, s typom `t` a parametrom 5

6.2.1 Návrh konfigurátora

K hľadaniu najkratších ciest v sieti sa bude využívať algoritmus podobný Uniform Cost Search (UCS, popísaný a porovnaný s Dijkstrovým algoritmom v [3]). Tento algoritmus sa bude pri hľadaní ciest využívať tak, že každý statický uzol v sieti začne šírenie cesty k sebe samému — určí cenu cesty k sebe rovnú nule. Následne každému susednému uzlu oznámi cenu cesty k cieľu (v tomto prípade sebe samému) a cenu cesty k danému susedovi. Každý sused si potom sčíta cenu dvoch prijatých ciest a porovná vypočítanú cestu s uloženou — ak je vypočítaná cesta kratšia, tak si ju uloží a ďalej šíri túto cestu ostatným susedom, v opačnom prípade nevykoná nič. Rozdiel voči UCS spočíva v tom, že UCS používa spoločnú pamäť v ktorej je zoznam spracovávaných ciest k cieľu, avšak spoločná pamäť nie je prítomná vo WSN, len zasielanie správ. Susedom sa teda posielajú aj cesty, ktoré sú dlhšie než je aktuálne známa najkratšia cesta u daného suseda a až príjemca rozhoduje, či sa bude táto cesta ďalej expandovať alebo nie (UCS expandovalo vždy práve tú najkratšiu známu cestu zo všetkých). Vďaka distribuovanému expandovaniu uzlov je možné súčasne šíriť v sieti cesty k viac než len jednému cieľovému uzlu.

Na obrázku 6.1 je znázornená ukážková sieť, na ktorej je demonštrované šírenie ciest k uzlu A. Priebeh šírenia je naznačený v tabuľke 6.2. V danej ukážke sa v jednom kroku spracuje prvá prijatá správa, ktorá je vo vstupoch a ak je cesta v nej obsiahnutá kratšia, než tá uložená tak sa aktualizuje uložená cesta a táto cesta sa ďalej zdieľa susedným uzlom, ktoré nie sú cieľom danej cesty (teda uzlom A) a ani uzlom, ktorý nám túto cestu poslal (optimalizácia pre redukovanie počtu prenášaných správ). Ceny ciest v tabuľke sú dopredu sčítané pre kompaktnosť a ukladajú sa len informácie o next-hop uzloch. Uvedený formát správ v tabuľke je teda zjednodušený (pre kompaktnosť), v plnej podobe by tieto správy obsahovali aj adresu cieľového uzlu šírenej cesty a cena cesty by bola rozdelená na cenu cesty od cieľa k next hop uzlu a cenu cesty od next-hop uzlu k uzlu, na ktorom sa práve daná správa spracováva. Taktiež sa v danej ukážke predpokladá, že odosielanie správy všetkým susedom prebehne v tom istom kroku, ako sa spracováva daná prijatá správa.



Obr. 6.1: Ukážková sieť

Krok	1.	2.	3.	4.	5.	6.
A – vstup	–	–	–	–	–	–
A – cesta	A/0	A/0	A/0	A/0	A/0	A/0
B – vstup	A/1	C/6	–	D/10	D/9	–
B – cesta	–	A/1	A/1	A/1	A/1	A/1
C – vstup	A/4	B/3	D/7	–	–	–
C – cesta	–	A/4	B/3	B/3	B/3	B/3
D – vstup	–	B/6,C/5	C/5,C/4	C/4	–	–
D – cesta	–	–	B/6	C/5	C/4	C/4

Obr. 6.2: Postup šírenia cesty z uzlu A; písmená v bunkách značia next-hop uzly a čísla značia cenu cesty cez daný uzol

Konfigurátor bude počiatočne odoslaný na niektorý z uzlov siete, ktorý by mal byť statický. Jeho následná činnosť bude prebiehať v troch fázach:

1. šírenie agenta po sieti
2. šírenie informácie o počte uzlov v sieti
3. zdieľanie informácií o cenách ciest k jednotlivým uzlom

Prvá fáza – šírenie agenta v sieti – by mohla byť vykonávaná za pomoci agenta presúvajúceho sa po uzloch v sieti, ktorého vytvoril v jeho diplomovej práci Marek Houšť. Tohto agenta určeného k prechodu sieťou bude potrebné modifikovať takým spôsobom, aby po ukončení bežal tento agent na všetkých statických uzloch v sieti. To je žiadané z toho dôvodu, že očakávame od uzlov ich následnú vzájomnú komunikáciu a v prípade samotného spomenutého agenta nebolo niečo také vyžadované.

Počas šírenia sa agenta po uzloch sa budú ukladať do bázy znalostí adresy navštívených uzlov, aby bolo následne možné určiť počet uzlov v sieti. Táto informácia bude počas hľadania ciest určená k rozhodnutiu, či už daný uzol dostal informácie o cestách ku všetkým uzlom v sieti (budeme predpokladať, že počas konfigurácie nebude prítomná tretia strana, ktorá by posielala zavádzajúce správy ohlasujúce cestu k neexistujúcim uzlom — teda počet známych ciest k nejakým uzlom nebude vyšší, než je reálny počet uzlov v sieti). Samotné adresy uzlov v sieti sa nebudú šíriť, vzhľadom na obmedzenú veľkosť prenášaných správ a problémy s prenosom veľkého množstva adries uzlov v rozsiahlych sieťach.

Spočítanie uzlov v sieti a následné ohlásenie bude vykonávať uzol, na ktorý bol počiatočne nahraný agent a z ktorého sa následne začal šíriť po sieti. To je žiadané z toho dôvodu, že sa pri presune agenta neaktualizujú bázy znalostí v uzloch, ktoré už boli navštívené (teda nebudú navštívené opäť, ak sa nenachádzajú po ceste navracania agenta) a teda skôr navštívené uzly budú obsahovať v bázi znalostí menej informácií o existujúcich uzloch v sieti, než neskôr navštívené uzly. Tento prechod agenta sieťou sa napokon skončí práve v predtým spomenutom uzle, na ktorý bol počiatočne nahraný agent a ktorý teda bude mať v bázi znalostí uložené adresy všetkých uzlov prítomných v sieti.

Ďalšiu fázu, šírenie informácie o počte uzlov v sieti, začne tiež práve uzol, na ktorý bol najprv nahraný agent. Najprv sa spočíta počet uzlov v sieti. Následne sa tento počet uzlov v sieti rozošle susedným uzlom. Ak tieto susedné uzly túto informáciu ešte nemali, tak si ju uložia a oznámia to všetkým svojim susedom. V prípade, že túto informáciu

už mali nevykonajú žiadnu akciu. Takýmto spôsobom sa táto informácia rozšíri po celej sieti, pričom každý uzol túto informáciu pošle len raz, čiže sa tým zabráni nekonečnému cyklickému odosielaniu tej istej správy a teda aj záplave siete touto správou. Každý uzol po poslaní tejto správy svojim susedom počká nejakú dobu (aby sa táto správa mohla dostatočne rozšíriť po sieti pred odosielaním ďalších správ) a následne každý uzol začne ďalšiu fázu – šírenie informácií o cestách.

Fáza šírenia informácií o cestách začína tým, že si každý uzol na jej začiatku uloží do smerovacej tabuľky cestu k sebe, so svojou vlastnou adresou ako next-hop adresou a nulovou cenou cesty. Následne každému susedovi oznámi, že cez neho existuje cesta k danému cieľu (v tomto prípade sebe samému) o nulovej dĺžke a priloží k nej zmeranú hodnotu RSSI suseda, s ktorým túto informáciu ide zdieľať. Po zdieľaní tejto cesty susedom uzol následne čaká na prichádzajúce správy.

Pri obdržaní správy o ceste si uzol skontroluje, či už má uloženú cestu k cieľu obsiahnutému v správe. Ak cestu k danému uzlu už má uloženú, tak si porovná dĺžku uloženej cesty s dĺžkou prijatej cesty, ku ktorej pričíta RSSI od odosielateľa, ktoré bolo taktiež obsiahnuté v prijatej správe. V prípade, že cestu k danému uzlu ešte nemal uloženú, tak si pridá do bázy znalostí informáciu o tom, že už pozná cestu k tomuto cieľovému uzlu a inkrementuje počítadlo známych ciest. Ak cestu k uzlu zo správy už poznal a prijatá cesta je dlhšia než uložená cesta, tak sa tým končí spracovanie tejto správy. Inak si uloží do flash pamäte informácie o dĺžke a ako next-hop adresu určí odosielateľa prijatej správy. Následne informuje susedov o ceste k cieľovému uzlu zo správy, rovnakou správou akou na začiatku bolo začaté šírenie (neinformuje odosielateľa správy a ani cieľ tejto cesty, ak je v zozname susedov).

Aktuálny priebeh sa bude počas konfigurovania vizuálne vyznačovať pomocou LED diód na uzloch. Konkrétne sa bude vyznačovať dopredné šírenie agenta po uzloch v prvej fázi, a v zvyšných dvoch fázach sa bude vyznačovať či uzol práve pracuje, teda spracováva nejakú správu, alebo či už má agent informácie o toľkých cieľoch, koľko je uzlov v sieti (čo je predpokladom, že má uzol znalosti o cestách ku všetkým statickým uzlom v sieti — známe cesty ale nemusia byť ešte v tom prípade najkratšie).

6.2.2 Návrh smerovača

Smerovač bude mať za úlohu odpovedať navigátorovi na správy, v ktorých bude žiadaný smer k určitému uzlu. Tento agent by sa mohol buď šíriť sieťou podobne ako agent konfigurátor, alebo by sa mohol nahrávať na jednotlivé statické uzly siete ručne. Keďže tento agent bude prítomný na statických uzloch siete, rovnako ako aj agent konfigurátor, by bolo možné spojiť týchto dvoch agentov do jedného agenta. To sa však nebude robiť z viacerých dôvodov:

- zvýšila by sa veľkosť agenta
- síce by potenciálne bolo možné navigovať už počas konfigurovania siete, ale v tej dobe známe smerovacie údaje môžu buď navádzať na nie najkratšiu cestu, alebo nemusia byť prítomné pre niektorý cieľ vôbec a kým by sa presunul navigovaný uzol medzi dvoma statickými uzlami, by sa smerovacie informácie mohli aktualizovať — čo by mohlo zmeniť trasu voči pôvodne navrhnutej ceste a spôsobovalo by to chaos
- pre účely navigovania v statickej sieti je žiadané, aby bolo možné spustiť agentov-smerovačov bez konfiguračnej časti

- keďže smerovači majú navádzať navigátora vizuálnymi indikáciami (tj. pomocou LED), tak by konfigurátorove indikovanie priebehu konfigurovania siete prekážalo s vizuálnymi indikáciami pri navigovaní

Z týchto dôvodov sa teda oddeľuje smerovač od konfigurátora. Ako bolo predtým spomenuté, až na zopár rovnakých plánov je množina akcií vykonávaných smerovačom odlišná od tých akcií, ktoré vykonáva navigátor. Teoreticky by to mohol byť jeden a ten istý agent a nie dvaja odlišní agenti. To by umožnilo napríklad presun jediného agenta v sieti namiesto viacerých prítomných vzájomne komunikujúcich agentov, ale znamenalo by to niekoľko vecí:

- väčšia veľkosť agenta
- ak by sa agent presúval po uzloch tak by sa spojením zvýšila doba potrebná pre prenos agenta
- ak by sa agent nepresúval po uzloch tak by bola časť plánov nevyužívaná

Ako bolo naznačené, tak ak by boli smerovač s navigátorom spojení, tak by bolo možné mať v sieti jedného agenta, ktorý by sa presúval po uzloch. Tým pádom by sa pri navigácii nemuseli vôbec posielať správy a len by sa presúval agent z navigovaného uzlu na statické uzly. Takýto spôsob riešenia tohoto problému však prináša viacero komplikácií:

- namiesto prenosu jednej krátkej správy by sa musel prenášať celý agent — ak by sa navigovaný uzol chcel dozvedieť ktorým smerom by mal ísť k určitému cieľu, tak by sa musel presunúť na nejaký statický uzol, na ňom si z pamäte flash prečítať požadované informácie a následne by sa musel presunúť späť, čo by bolo časovo náročné, najmä pri opakovaní týchto presunov
- mohla by nastať situácia, kedy by sa agent presunul na statický uzol za účelom vykonania nejakej akcie, ale následne by sa navigovaný uzol presunul mimo dosahu statického uzlu, na ktorom by bol aktuálne agent (počas vykonávania požadovanej akcie) a tým pádom by nebolo možné pre agenta sa vrátiť späť na navigovaný uzol

Preto teda smerovač bude osobitný agent, ktorý bude mať za úlohu odpovedať na žiadosti navigátora. Okrem vyhnutia sa viacerým úskaliam, ktoré by vznikli u spomenutých alternatív, tento spôsob riadenia navigácie navyše prináša aj ďalšie výhody:

- nižšia veľkosť jednotlivých agentov
- menej prenášaných dát
- možnosť prítomnosti viacerých navigovaných uzlov v sieti — ak by smerovač bol spojený s navigátorom, tak by mohla nastať situácia kedy by na statickom uzle bol agent jedného navigovaného uzlu prepísaný druhým agentom a ten prvý by teda zanikol

Samotný beh navigátora by bol začatý šírením sa po sieti identickým spôsobom, ako to bolo u konfigurátora (ak by sa teda mal sám rozšíriť po sieti po statických uzloch). Následne by už len cyklicky čakal na správy, ktoré by spracoval a na ktoré by v prípade potreby odpovedal odosielateľovi.

Pri takto navrhnutom spôsobe navigovania by smerovač mal poskytovať aspoň dve základné funkcie pre navigátora. Prvou funkciou je získanie smerovacích informácií zo smerovacej tabuľky a oznámenie smeru navigátorovi. V prípade, že v smerovacej tabuľke nie

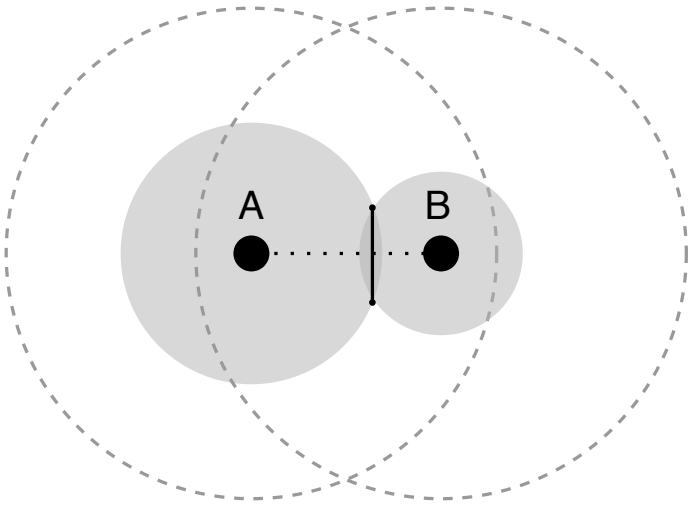
sú uložené smerovacie informácie k cieľu, ktorý požadoval navigátor, sa táto skutočnosť oznámi navigátorovi osobitným typom odpovede — ak však prebehla konfigurácia úplne a v poriadku tak by takáto situácia mohla nastať len v prípade, ak by navigátor obdržal ako cieľ adresu uzlu, ktorý sa v tejto sieti nenachádza. Druhou požadovanou akciou, ktorú má byť smerovač schopný vykonať, by malo byť rozsvietenie LED diód pre indikáciu, že sa má navigátor pohybovať tým smerom — ak by bolo v sieti viacero navigovaných uzlov a dva by sa súčasne chceli dostať k rovnakému statickému uzlu, tak by si mohli vzájomne meniť svetelnú indikáciu smerovača. To by mohlo spôsobovať zmätok v prípade, že by sa svetelne indikovala aj sila RSSI k danému uzlu.

Mimo spracovávania správ môže byť tento agent uspaný, teda sa tým potenciálne zníži spotreba energie pri nečinnosti. V tomto prípade by teda mali byť aj úlohy vykonávané smerovačom dokončené v čo najkratšom čase, a to nie len z dôvodu úspory energie, ale aj z toho dôvodu, že sa tým zvyšuje počet navigátorov, ktorých je smerovač schopný súčasne obsluhovať.

6.2.3 Návrh navigátora

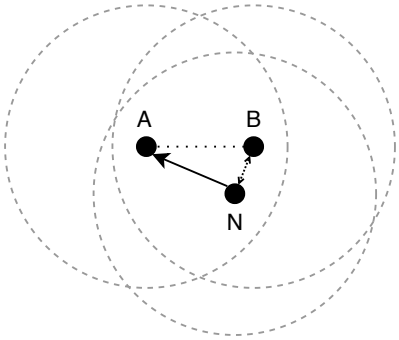
Navigátor bude zodpovedný za aktívne riadenie procesu navigácie k cieľu, ktorý obdržal napríklad prostredníctvom správy od base station. Kým konfigurátor a smerovač sú navrhnutí tak, že im stačí čakať na prichádzajúce správy, je tento spôsob riadenia vykonávania plánu nevhodný pre navigátora. Dôvodom je to, že v prípade ak by sa v blízkosti nachádzal uzol, ktorý by neobsahoval smerovača a navigátor by mu poslal správu s tým, že sa uspí do doby príchodu odpovede, tak by sa tejto odpovede (s najvyššou pravdepodobnosťou) nikdy nedočkal a teda by proces navigácie skončil neúspešne uviaznutím pri čakaní na odpoveď. Takáto situácia by mohla nastať aj v prípade, že sa smerovačovi nepodarilo odoslať správu späť navigátorovi (napríklad sa navigátor príliš vzdialil).

Z toho dôvodu bude navigátor počas procesu navigácie riadený s pevne danými krokmi čakania, aby bolo možné počítať dobu, po ktorú ešte bude navigátor čakať na odpoveď. Ak vyprší doba čakania na odpoveď, tak sa znovu vykoná plán pre vyžiadanie si smeru k cieľu. Inak ak navigátor nemá určený cieľ, tak môže byť uspaný a čakať na prichádzajúcu správu, napr. obsahujúcu cieľ ku ktorému sa bude navádzať.

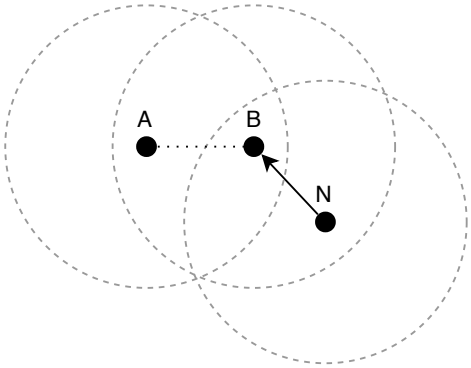


Obr. 6.3: Odhad pozície podľa RSSI

Navigátor svoju pozíciu stanovuje na základe znalostí z trilaterácie. Predpokláme, že medzi susednými uzlami v sieti sú priamočiaré spojenia bez prekážok a pozíciu navigátora aproximujeme na základe vzdialenosti medzi dvoma uzlami — najbližším susedným uzlom a pozíciou next-hop uzlu. Ďalej využívame znalosť, že vzdialenosť medzi uzlami je symetrická, teda navigátor nemusí žiadať susedov aby určili jeho pozíciu, stačí mu RSSI susedov ktoré on sám zmeria pri ich hľadaní. Navigátor trivializuje svoju pozíciu na bod na spojnici medzi týmito dvoma uzlami. Tento bod je vyznačený na obr. 6.3 zvislou čiarou (resp. prienikom tej čiary so spojnicou) — **A** a **B** sú spomenuté susedné uzly, prerušované kružnice reprezentujú oblasť pokrytia ich signálu, šedé kruhy reprezentujú vzdialenosť uzlu od nich podľa RSSI a bodkovaná čiara je spojnicou medzi týmito dvoma uzlami. Keby sa však navigátor nachádzal skutočne priamo na spojnici medzi tými dvoma uzlami, tak by stačilo k určeniu jeho pozície medzi týmito dvoma uzlami získať len hodnotu RSSI jedného z týchto dvoch uzlov — vzhľadom na to, že by prienikom gulí vzdialeností od týchto uzlov (predstavujúcich všetky možné umiestnenia navigovaného uzlu) bol jediný bod, ktorý by ležal práve na tejto spojnici. Práve preto teda idealizujeme pozíciu navigovaného uzlu na jeden bod a budeme používať RSSI len jedného uzlu z týchto dvoch uzlov — RSSI next-hop uzlu, ku ktorému sa chceme dostať. V prípade, že sa navigátor nenachádza v dosahu next-hop uzlu (nie je v zozname susedov, resp. má nulové RSSI), sa budeme snažiť dostať k druhému uzlu z tejto dvojice, keďže v jeho dosahu už musí byť next-hop uzol — to, že nejaký uzol má v dosahu next-hop uzol na ceste k nejakému uzlu musí byť splnené už pri konfigurácii siete, keďže inak by daný uzol nemohol dostať od next-hop uzlu správu o ceste k cieľu.



Obr. 6.4: Next-hop uzol **A** je v dosahu navigátora **N**



Obr. 6.5: Next-hop uzol **A** nie je v dosahu navigátora **N**

Priebeh typického navigovania začína príchodom správy, obsahujúcej cieľ. Následne navigátor (označený ako **N** na obrázkoch 6.4 a 6.5) získa zoznam susedov, ktorý spracuje a z neho vyberie najbližšieho suseda, na obrázkoch vyššie označeného ako **B**. Tomuto susedovi sa následne pošle správa, ktorá žiada o smer k zvolenému cieľu. Po prijatí odpovede sa skontroluje, či sa nachádzal next-hop uzol **A** v zozname susedov (tj. či je v dosahu **N**, dosah je na obrázkoch vyznačený prerušovanou kružnicou okolo uzlov) — v prípade že áno (obr. 6.4), tak sa navigátor bude snažiť dostať k tomuto next-hop uzlu, keďže predpokladáme že je v tomto prípade v dohľade a teda presun k tomuto uzlu bude bezproblémový. V prípade, že sa next-hop uzol nenachádzal v zozname susedov (obr. 6.5), sa navigátor bude snažiť dostať k uzlu, od ktorého sme žiadali smer, keďže sa predpokladá, že od tohto uzlu už bude next-hop uzol v dosahu.

Následne navigátor bude periodicky zisťovať RSSI od uzlu, ku ktorému sa snaží aktuálne dostať a posiela hodnotu RSSI tomuto statickému uzlu, ktorý následne podľa hodnoty RSSI bude vyznačovať vzdialenosť navigovaného uzlu od neho. Toto informovanie uzlu, ku ktorému sa snažíme navigovať sa skončí v dvoch prípadoch — buď je dosiahnutá určitá prahová sila signálu, alebo už nie je tento uzol už v dosahu. Ak bola dosiahnutá prahová sila signálu a uzol, ku ktorému sa snažil navigátor dostať bol cieľom jeho cesty, tak sa tým ukončí navigácia a to sa vyznačí vizuálne na navigátorovi. Ak sa nejednalo o cieľový uzol, tak požiadame tento next-hop uzol o ďalší smer cesty. Inak ak už uzol nie je v dosahu, tak pýtame o smer cesty susedný uzol s najvyššou hodnotou RSSI. Následne sa tento proces od poslania žiadosti o smer opakuje dovtedy, kým sa navigátor nedostane k požadovanému cieľu.

Navigátor teda má byť schopný spracovať aspoň tri typy správ — správu oznamujúcu cieľ jeho trasy a správy, ktoré mu môžu prísť ako odpoveď na žiadosť o smer. Ako bolo uvedené v predošlom odstavci, v prípade príchodu správy určujúcej cieľ navigovania sa inicializuje proces navigácie a žiada najbližšieho suseda o smer. V prípade príchodu správy s adresou next-hop uzlu prebehne navigovanie buď k next-hop uzlu, alebo k danému susedovi — v závislosti od dosiahnuteľnosti next-hop uzlu, ako to bolo popísané vyššie. Ak navigátor po žiadosti o smer dostal správu od najbližšieho suseda, že nie je známa cesta k danému uzlu, tak sa následne pokúsi požiadať o smer niektorého z ostatných susedov — v prípade, že ani jeden sused nepoznal cestu k danému cieľu sa táto skutočnosť bude vizuálne reprezentovať a navigátor sa po určitej časovej dobe znovu pokúsi žiadať susedov o smer (pre prípad, ak by sa medzitým navigátor samovoľne presunul k nejakým iným uzlom, alebo ak predtým pri žiadaní o smer nebol detegovaný niektorý sused). Taktiež navigátor bude vizuálne naznačovať, aj keď v jeho dosahu nie je žiadny sused.

Ako bolo však vyššie spomenuté, existuje aj možnosť, že by sa v jednej bezdrôtovej sieti mohli nachádzať aj viacerí navigátori súčasne. Keďže navigátor žiada o smer najbližšieho suseda, tak by mohla nastať situácia, kedy by jeden navigátor žiadal druhého o smer. Pre tento prípad by mal navigátor byť schopný reagovať aj na žiadosti o smer, a to aspoň správou že nemá informácie o ceste k danému cieľu — čím následne navigátor, ktorý žiadal o smer toho druhého, požiada ďalších susedov. Takéto riešenie pomocou zápornej odpovede by malo byť dostatočné vzhľadom na to, že sa nepredpokladá že mobilné uzly budú mať hocijaké smerovacie informácie, keďže sa nepodieľali (aspoň sa teda nemali podieľať) pri konfigurovaní siete.

Kapitola 7

Implementácia

7.1 Smerovacia tabuľka

V predošlej kapitole bolo spomenuté, že smerovacia tabuľka bude vyžadovať možnosť ukladať dáta perzistentne. Preto bude potrebné implementovať nový modul pre platformu WSageNt, ktorý by bol schopný takto ukladať tieto dáta a tiež bude následne potrebné vytvoriť pre interpret rozhranie pre volanie služby platformy, ktorá by následne dokázala pracovať s týmto novým modulom.

7.1.1 Modul pre prácu s pamäťou

Platforma TinyOS od verzie 2 poskytuje niekoľko rôznych abstrakcií pracujúcich s perzistentným úložiskom dát. Konkrétne sú poskytované tri rôzne typy úložiska (a teda k nim rozhrania a komponenty): `ConfigStorage`, `BlockStorage` a `LogStorage`. Každé z týchto úložísk má svoje výhody a nevýhody [9]:

- `LogStorage` — umožňuje atomické zapisovanie menších záznamov, pričom sa ale vždy zapisuje na aktuálny koniec doposiaľ zapísaných dát. Týmto spôsobom je možné ukladať veľké objemy dát. Dopĺňajú sa teda záznamy (definované napríklad nejakou štruktúrou) buď ako v lineárnom úložisku (ak sa vyčerpá kapacita tak sa už nezapisuje) alebo kruhovom úložisku (po vyčerpaní kapacity sa začnú prepisovať najstaršie záznamy, tj. záznamy na začiatku úložiska). Pre čítanie sa musí vyhľadávať zapísaná pozícia.

- `BlockStorage` — využíva sa pre ukladanie veľkých objemov dát. Umožňuje ako náhodné zápisy, tak aj náhodné čítania, ale negarantuje žiadnu spoľahlivosť týchto operácií. Navyše, medzi dvoma vymazaniami celej jednotky môže byť vykonaný zápis každého bajtu len raz.

- `ConfigStorage` — používa sa pre uchovanie menších objemov dát. Jednotku je potrebné pred použitím pripojiť (angl. `Mount`). Tento typ úložiska je navrhnutý za účelom prevencie straty dát. Preto sa uchovávajú dve kópie dát uložených v tejto jednotke — jedna kópia, do ktorej sa pri zápise uložia zmeny (príkaz `write`) a druhá kópia, do ktorej sa ukladajú už potvrdené zmeny (príkaz `commit`). To však znamená, že maximálne množstvo uložiteľných dát bude menšie alebo rovné polovici celkového objemu jednotky. Čítanie prebieha z kópie, ktorá obsahuje už potvrdené zmeny.

Keďže pri tvorbe smerovacej tabuľky je potrebná možnosť viacnásobného zapisovania a aj náhodného prístupu pre čítanie dát, je najvhodnejšou voľbou typu úložiska pre túto aplikáciu `ConfigStorage`. `ConfigStorage` využíva dve rozhrania: `Mount` a `ConfigStorage` — `Mount` poskytuje rozhranie pre pripojenie jednotky a `ConfigStorage` následne poskytuje rozhranie pre bežné operácie s úložiskom.

Preto teda v súbore `volumes-at45db.xml` bola pridaná jedna nová jednotka s názvom `ROUTETABLE` o veľkosti 4096 B, ktorá bude používaná pre uchovávanie týchto dát:

```
<volume_table>
  <volume size="16384" name="SENSELOGTEMP" />
  <volume size="16384" name="AGFOOTPRINT" />
  <volume size="4096" name="ROUTETABLE" />
</volume_table>
```

Kvôli duplicitnému uchovávaniu dát je však reálne využiteľná kapacita tejto jednotky menšia. Konkrétne príkaz `getSize` vracia dostupnú veľkosť 2022 B. Táto dostupná kapacita by teda mala postačiť pre sieť o veľkosti až do 503 uzlov.

Ďalej bol vytvorený modul `ConfigMemoryM`, ktorý používa rozhrania poskytované od `ConfigStorage`. Tento modul poskytuje dve rozhrania — rozhranie `iConfigMemoryI` a rozhranie `ConfigMemoryI`. Rozhranie `ConfigMemoryI` poskytuje päť asynchrónnych príkazov (teda sa pri ich vykonávaní pozastaví beh interpretu, kým sa nedokončia): `eraseMemory`, `readValue`, `readPair`, `writeValue` a `writePair`. Rozhranie `iConfigMemoryI` poskytuje príkaz `ableToAdd`, ktorý používa riadiaca slučka interpretu. Tento modul taktiež implementuje spracovanie udalostí, ktoré vyvoláva `ConfigStorage`:

- Ak bolo ukončené čítanie z pamäti tak nastaví značku, že je pripravený vložiť prečítané dáta do aktívneho registra a spustí opäť beh interpretu. Pokiaľ je nastavená táto značka, tak sa pri ďalšom kroku interpretu vložia prečítané dáta do aktívneho registra.
- Ak sa dokončil zápis dát a jedná sa o príkaz `eraseMemory`, tak sa vymazáva ďalšia časť pamäti (ak sa nemazal posledný blok). V opačnom prípade sa vyvolá príkaz `commit`, ktorý tieto dáta uloží na trvalo.
- Ak bola dokončená operácia `commit`, tak sa opätovne spustí beh interpretu a dokončuje sa tým naposledy spustená operácia (tj. `eraseMemory`, `readValue` alebo `readPair`).

Príkazy poskytované týmto modulom sú:

- `bool eraseMemory()` — prepíše celý obsah pamäte hodnotami `-1` (pôvodná hodnota po inicializácii). Návratová hodnota je vždy `TRUE`.
- `bool readValue(int16_t offset)` — prečíta 16-bitovú hodnotu so znamienkom a vloží ju do aktívneho registra, uzatvorenú v zátvorkách (napríklad `(123)`). Za zmienku stojí, že `offset` neudáva absolútny posun v bajtoch od začiatku jednotky, ale slúži skôr ako index do poľa dvojbajtových premenných — určuje, ktorá hodnota sa má prečítať. Absolútny posun v bajtoch je teda dvojnásobkom hodnoty `offset`. Vracia `FALSE` ak je jednotka nevalídna, ak sa ešte nedokončila iná operácia alebo ak je `offset` mimo legálnych hodnôt; inak vracia `TRUE`.

- `bool readPair(int16_t offset)` — prečíta dve 16-bajtové hodnoty so znamienkom a vloží ich uzatvorené v zátvorkách do aktívneho registra ako pár, teda v ďalších zátvorkách — napríklad sa z prečítaných hodnôt 123 a -987 vloží do aktívneho registra `((123),(-987))`. Paramater `offset`, podobne ako u `readValue` neudáva absolútny posun, tentokrát je ale posun štvornásobkom tejto hodnoty, keďže sa čítajú dve dvojbajtové hodnoty súčasne. Návratová hodnota je identická s `readValue`.
- `bool writeValue(int16_t offset, int16_t value)` — zapíše 16-bitovú hodnotu na daný `offset`. Hodnota `offset` je identická s `readValue`, návratová hodnota tiež.
- `bool writePair(int16_t offset, int16_t first, int16_t second)` — zapíše dve 16-bitové hodnoty `first` a `second` na daný `offset`. Hodnota `offset` je identická s `readPair`, návratová hodnota tiež.
- `bool ableToAdd()` — ak sú k dispozícii prečítané dáta, tak ich vloží do aktívneho registra. Tento príkaz musí byť zavolaný pre dokončenie operácií čítania.

7.1.2 Rozhranie pre volanie služby platformy

Po vytvorení samotného TinyOS modulu, ktorý bude schopný manipulovať s perzistentným úložiskom je ešte potrebné umožniť využívanie tohto úložiska v ALLL. Preto bolo pridané volanie služby platformy, ktoré pracuje s týmto modulom.

Pridaná služba platformy do ALLL je označená znakom `u`. Prvým parametrom volania tejto služby je opäť znak, ktorý určuje operáciu ktorá sa má vykonať. Volanie tejto služby zlyhá, ak je chyba v parametroch volania operácie, alebo ak zlyhá samotné volanie príkazov modulu `ConfigMemoryM`. Dostupné operácie sú rovnaké, ako príkazy z rozhrania `ConfigMemoryI`.

Numerické parametre je možné určiť rôznymi spôsobmi. Je možné využiť substitúciu, teda môžu byť uložené v registroch, alebo môžu byť zapísané priamo. Či už sú parametre čítané z registrov, alebo sú zadané priamo v akcii hodnotou, môžu byť ako v zátvorkách, tak aj mimo nich. Taktiež to môžu byť ako záporné, tak aj kladné čísla — to je možné použiť pri zápise čísel do pamäti.

Prvá operácia je vymazanie pamäti — kód operácie je `e`, bez parametrov. Táto operácia prepíše všetky bajty dostupnej pamäti hodnotou -1. Ukážka volania: `$(u,(e))`.

Ďalej sú k dispozícii operácie so zápisom/čítaním jednej dvojbajtovej hodnoty. Obe tieto operácie majú aspoň jeden parameter — tým je offset. Parametre týchto operácií a ich sémantika sú identické k tým, ktoré majú príkazy `readValue` a `writeValue`.

Čítanie jedného dvojbajtového čísla: kód operácie je `r`, ako parameter je len offset. Prečítaná hodnota sa vloží do aktívneho registra. Príklady volania:

- `$(u,(r,12))` — prečíta hodnotu z offsetu 12
- `$(u,(r,&2))` — prečíta hodnotu z offsetu uloženého v druhom registri
- `$(u,(r,(13)))` — prečíta hodnotu z offsetu 13

Zápis jedného dvojbajtového čísla má kód operácie `w`, má dva parametre — offset a zapisovanú hodnotu. Príklady volania:

- `$(u,(w,12,-125))` — na offset 12 zapíše hodnotu -125
- `$(u,(w,&2,(13)))` — na offset uložený v registri 2 zapíše hodnotu 13

- `$(u,(w,(12),&3))` — na offset 12 zapíše hodnotu uloženú v registri 3
- `$(u,(w,&1,&3))` — na offset uložený v registri 1 zapíše hodnotu uloženú v registri 3

Zvyšné dve operácie sú čítanie a zápis párov hodnôt, ktoré sa budú využívať pri ukladaní informácií o smerovaní. Taktiež ako predošlé dve operácie majú obe tieto operácie ako parameter offset, z ktorého sa má buď hodnota čítať, resp. na ktorý sa má hodnota zapísať. Tieto operácie využívajú páry hodnôt, zapísané v zátvorkách — napríklad `((15),(4532))`. Stojí teda za pripomenutie, že offset tejto operácie funguje ako to bolo popísané u implementácie modulu pracujúceho s pamäťou, teda u týchto dvoch operácií sa pri offsete x manipuluje s hodnotami, s ktorými by sa manipulovalo, ak by sme pracovali s operáciami čítajúcimi/zapisujúcimi jedinú hodnotu na offsetoch $2x$ a $2x + 1$.

Čítanie páru: kód operácie je `q`, ako parameter je len offset. Prečítaný pár sa vloží do aktívneho registra. Príklady volania:

- `$(u,(q,12))` — prečíta pár z offsetu 12
- `$(u,(q,&2))` — prečíta pár z offsetu uloženého v druhom registri

Zápis páru: kód operácie je `p`, ako parameter je offset a pár, ktorý sa má zapísať. Tento pár musí byť zapísaný v zátvorkách, či už je uložený v registri alebo je zapísaný priamo v akcii. Substitúciou je možné nahradiť jednu alebo aj obe z hodnôt páru, alebo rovno aj celý pár. V prípade, že sa substituuje celý pár sa nesmie v ňom nachádzať ďalšie adresovanie registra, keďže sa už táto substitúcia nevykoná a vyhodnotí sa to ako chyba. Inak samotné numerické hodnoty v pároch nasledujú vyššie spomenuté možnosti zápisu číselných hodnôt. Príklady volania — predpokladajme že v prvom registri je hodnota 15, v druhom registri je hodnota 175 a v treťom registri je uložený pár ((-123),456):

- `$(u,(p,12,((11),256)))` — na offset 12 zapíše pár hodnôt (11,256)
- `$(u,(p,1,(&1,256)))` — na offset 1 zapíše pár hodnôt (15,256)
- `$(u,(p,&1,(-4,(2))))` — na offset 15 zapíše pár hodnôt (-4,2)
- `$(u,(p,7,(&2,&1)))` — na offset 7 zapíše pár hodnôt (175,15)
- `$(u,(p,&1,&3))` — na offset 15 zapíše pár hodnôt (-123,456)

7.2 Konfigurátor

Konfigurátor, implementovaný podľa fáz ako to bolo uvedené v sekcii 6.2.1, bude popísaný v tejto sekcii. Pre pripomenutie jednotlivé fázy sú: šírenie agenta, šírenie informácie o počte uzlov a šírenie ciest. Najprv bude popísaný počiatočný plán a následne bude popísaná knižnica plánov a význam jednotlivých plánov. Ako bolo spomenuté, stav sa bude indikovať vizuálne — ak sa na uzle ešte šíri agent dopredne, tak sa rozsvieti červená dióda a ak sa bude z tohto uzlu vracať spätne, tak sa červená dióda vypne. Ďalej ak práve agent pracuje počas fáz 2 a 3 (napríklad spracovávanie správy), tak rozsvieti žltú diódu. Ak počas fázy 3 bude mať informácie aspoň o toľkých uzloch, koľko ich je v sieti, tak rozsvieti zelenú diódu.

Konfigurátor používa dva typy správ:

- `c` (Count) — oznamuje počet uzlov v sieti, ako parameter má číselnú hodnotu udávajúcu počet uzlov v sieti (používaný v druhej fázi)

- `p` (Path) — oznamuje cestu k nejakému cieľu, prvý parameter udáva identifikáciu tohto cieľa a druhý parameter je dvojica (cena, RSSI), kde cena udáva cenu danej cesty od cieľa k odosielateľovi správy a RSSI je hodnota RSSI tohoto uzlu zmeraná odosielateľom správy

Počiatočný plán konfigurátora je

```
$(l,(g,1))$(w,(250))$(l,(g,0))+(SA)^(nb)
$(u,(e))&(1)$(t,(i))+(DC,(1))+(D,&1)$(u,(p,&1,(&1,(0))))
*(RC)-(RC)$(l,(y,1))+(NC,(0))^(Cn)-(NB,_,_)$(n)$(w,(250))^(Tn)
$(l,(y,0))$(w,(5000))$(l,(y,1))^(Ss)$(l,(y,0))
#+(AI)^(Mh)
```

Najprv blikne zelená dióda ako indikácia, že sa úspešne nahral agent na uzol. Pridanie znalosti `SA` (Spread Agent) sa používa pri šírení agenta v sieti — to vykonáva plán `nb`, ktorý bol vytvoril v jeho diplomovej práci Marek Houšť a ktorý bol následne modifikovaný aby zostal agent bežať na uzloch pri navracaní agenta; práve pri tom sa používa znalosť `SA`. Konfigurátor po ukončení plánu šírenia (tj. už na každom statickom uzle siete osobitne) vymaže konfiguračnú pamäť a nastaví cestu k sebe samému s cenou nula a inicializuje znalosti `DC` (tj. počet uzlov, ku ktorým je známa cesta) a pridá znalosť `D` o sebe samom, ktorá hovorí že k tomuto uzlu je už známa cesta v smerovacej tabuľke. Následne znalosť `RC` bude mať v bázi znalostí po šírení len ten uzol, na ktorý bol počiatočne tento agent nahraný. Pre ostatné uzly sa vykoná len posledný riadok tohto počiatočného plánu za zarážkou, teda sa zavolá plán `Mh` (Message Handler). Pre počiatočný uzol sa nastaví znalosť `NC` (Node Count) na hodnotu nula a následne sa zavolá plán `Cn` (Count Nodes), ktorý spočíta uzly v sieti. Následne sa počet uzlov oznámi susedom plánom `Tn` a po piatich sekundách sa plánom `Ss` začne šíriť cesta k tomuto uzlu. Potom sa spustí plán `Mh` ako pre ostatné uzly.

Ďalej popíšeme knižnicu plánov. Prvých zopár plánov, ktoré sú väčšinou rovnaké pre všetkých troch agentov, sú plány spracovania správ a dva pomocné plány:

```
(Mh,(^(Pm)$(l,(r))$(w,(75))$(l,(r))$(s)^(Mh)))
(Pm,(&(1)?(_)&(2)$(f,&1)-(MT,_)+(MT,&2)&(3)$(r,&1)&(2)$(f,&3)-(MS,_)
    +(MS,&2)&(1)$(r,&3)-(MC,_)+(MC,&1)^(Pa)^(Pm)))
(Rs1,(&(1)$(f,&2)&(2)$(r,&1)&(1)))
(Rs2,(&(2)$(f,&1)&(1)$(r,&2)&(2)$(f,&1)))
```

Plán `Mh` (Message Handling) len zavolá plán `Pm` (Process Message) a po ukončení spracovávania správ blikne červenou diódov a uspí sa do príchodu ďalšej správy, po čom sa opäť spustí. Plán `Pm` prečíta správu z bázy vstupov a spracuje jej jednotlivé položky do znalostí `MT` (Message Type), `MS` (Message Sender) a `MC` (Message Content) a zavolá sa plán `Pa` (Perform Action) pred tým, než sa pokúsi spracovať ďalšie správy z bázy vstupov pred uspaním. Pomocné plány `Rs1` a `Rs2` sa používajú väčšinou po teste bázy znalostí pre získanie hodnoty z párov uložených do bázy znalostí (ako napríklad práve `MS`).

Plán `Pa` (Perform Action) vykoná plán na základe typu správy, ktorý bol prijatý — buď `Sc` (Share Count) pre typ správy `c` (Count) obsahujúcej počet uzlov v sieti, alebo `Sr` (Spreading Route) pre typ správy `p` (Path) oznamujúcej nejakú cestu k cieľu.

```
(Pa,($(l,(y,1))&(3)*(MT,(c))^(Sc)#*(MT,(p))^(Sr)#$(l,(y,0))))
```

Ďalej nasledujú plány šírenia agenta. Ako bolo spomenuté vyššie, jedná sa o mierne upravené plány agenta z [7]. Dve väčšie zmeny v porovnaní s originálnymi plánmi sú ukladanie

identifikátorov navštívených plánov do báze znalostí ako znalosť `(V,<id>)` a v pokračovaní behu agenta pri navracaní:

```
(nb,($(l,(r,1))&(1)$(t,(i))+(V,&1)$(n)$(w,(250))^(cn)))
(cn,(^(gn)*(SA)^(gi)^(tm)#*(SA)^(cn)))
(gi,(&(1)$(r,&2)&(2)$(f,&1)))
(gn,(+(bk)&(1)*(NB,_,_)&(2)$(f,&1)-&2-(bk)#&(1)*(bk)^(bk)))
(tm,($(t,(q,&2))-(NB,_,_)$(m,(&2,s))^(nb)))
(bk,(&(3)$(t,(b,f))+(b,&3)&(2)*(b,(1))-(SA)+(RC)$(l,(r,0))#&(2)*(SA)-(PN,_)
    +(PN,&3)$(m,&3)&(1)$(t,(i))+(C)&(2)@(*(PN,&1)-(C)^(nb))&(2)*(C)-(SA)
    $(l,(r,0))))
```

Ďalej sú prítomné plány relevantné k získavaniu počtu uzlov v sieti a šíreniu tejto informácie. Prvým je `Cn` (Count Nodes), ktorý spočíta uzly v sieti podľa identifikátorov uložených uzlov v bázi znalosti. Ďalej už skôr spomínaný plán `Sc` uloží do bázy znalostí počet uzlov v sieti, ak ešte nebol známy a šíri ďalej túto informáciu susedom pomocou plánu `Tn`, pričom následne po piatich sekundách zavolá plán `Ss`. Ak už počet uzlov bol uložený v bázi znalostí, tak sa nič nevykoná. Plán `Tn` (Tell Neighbors) pošle každému susedovi uloženému v bázi znalostí (napr. po volaní služby `n`) informáciu o počte uzlov v sieti.

```
(Cn,(&(1)*(V,_)&(2)$(f,&1)-&2&(1)*(NC,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())
    &(3)$(f,&1)-(NC,_)+(NC,&3)^(Cn)))
(Sc,(&(3)+(CU)*(NC,_)-(CU)#*(CU)&(3)$(f,&1)+(NC,&3)-(NB,_,_)$(n)$(w,(250))
    &(1)$(e,(c,&3))&(2)@(*(DC,&1)$(l,(g,1)))^(Tn)$(l,(y,0))$(w,(5000))
    $(l,(y,1))^(Ss)))
(Tn,(&(2)*(NB,_,_)&(1)$(f,&2)-&1&(2)$(r,&1)&(1)$(f,&2)+(SN)&(2)*(MS,&1)
    -(SN)#*(SN)-(SN)&(2)$(t,(i))!(&1,(c,&2,&3))^(Tn)))
```

Plán `Ss` (Start Spreading) pripraví tak bázu znalostí, aby bol použiteľný plán `In` (Inform Neighbors) pre posielanie informácií o ceste susedom, ktorý sa práve pre tento účel ďalej využíva.

```
(Ss,(-(TR,_)+(TR,(0))&(1)$(t,(i))-(MC,_)+(MC,(&1))-(MS,_)+(MS,&1)-(NB,_,_)
    $(n)$(w,(250))&(2)*(NB,_,_)^(In)))
```

Zostáva blok plánov relevantných k spracovávaniu a šíreniu ciest v sieti:

```
(Ir,(&(2)$(o,sub,(125),&3,(),(),())&(3)$(f,&2)&(2)$(o,les,&3,(1),(),(),())
    -(IU,_)+(IU,&2)&(1)*(IU,((1)))&(3)$(f,&2)))
(Pl,(&(2)$(r,&1)&(1)$(f,&2)&(2)$(f,&1)&(3)$(r,&1)&(1)$(f,&3)&(3)
    $(o,add,&1,&2,(),(),())&(2)$(f,&3)&(3)$(e,(c,&2))-(TR,_)+(TR,&3)))
(Mp,(-(MP,_)+(MP,(&2,&3))&(2)*(MP,_)&(3)$(f,&2)&(2)$(r,&3)&(3)$(f,&2)))
(Ic,(&(1)*(DC,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())&(2)$(f,&1)-(DC,_)
    +(DC,&2)&(1)$(e,(c,&2))&(2)*(NC,&1)$(l,(g,1))&(2)*(AI)-(AI)+(RS)))
(Gt,(&(1)*(MC,_)^(Rs2)&(1)$(f,&2)))
(Si,(+(S)^(Gt)&(2)*(ON,&1)-(S)#&(2)*(MS,_)^(Rs1)$(f,&2)&(2)*(ON,&1)-(S)))
(In,(&(1)*(NB,_,_)&(2)$(f,&1)-&2&(3)$(r,&2)&(2)$(f,&3)-(ON,_)+(ON,&2)^(Si)
    ^(Sp)))
(Sp,(&(2)*(S)-(S)&(1)$(r,&3)&(3)$(f,&1)^(Ir)&(1)*(TR,_)^(Rs2)^(Mp)^(Gt)&(2)
    $(t,(i))-(MSG,_)+(MSG,(p,&2,&1,&3))&(2)*(ON,_)&(3)$(f,&2)&(2)$(r,&3)
```

```
    &(3)$(f,&2)&(1)*(MSG,_)^(Rs2)!(&3,&2)#^(In)))
(Ur,(&(1)*(MS,_)^(Rs2)^(Mp)^(Gt)$(u,(p,&1,&3))+(I)&(2)*(D,&1)-(I)#&(2)*(I)
    +(D,&1)^(Ic)#-(NB,_,_)$(n)$(w,(250))^(In)&(2)*(RS)^(Ss)))
(Sr,(^(Pl)+(NP)^(Gt)&(2)$(u,(q,&1))&(1)$(r,&2)&(2)$(f,&1)&(1)
    $(o,les,&2,(0),(),(),())-(LP,_)+(LP,&1)*(LP,((0)))&(1)
    $(o,leq,&2,&3,(),(),())-(LP,_)+(LP,&1)*(LP,((1)))-(NP)#*(NP)^(Ur)))
```

- `Ir` (Invert RSSI) upraví hodnotu RSSI tak, ako to bolo spomenuté v sekcii 6.1.1, pričom sa ako hodnota posunu používa konštanta 125. Dôvod bude podrobnejšie popísaný v sekcii 8.1.
- `Pl` (Path Length) vypočíta dĺžku cesty k tomuto uzlu podľa informácií obsiahnutých v správe.
- `Mp` (Make Pair) vytvorí dvojicu, ktorá sa bude odosielať susedom alebo ktorou sa bude volať služba platformy `u`.
- `Ic` (Increment Counter) navýši počítadlo uzlov, ku ktorým sú známe cesty a ak je známych aspoň toľko ciest, koľko je uzlov v sieti tak rozsvieti zelenú diódu.
- `Gt` (Get Target) uloží do registra 1 cieľ cesty zo správy.
- `Si` (Should Inform) pridá do bázy znalostí znalosť `S`, ak sa má daný uzol informovať — pri šírení znalosti o nejakej ceste sa nebude informovať cieľ a ten uzol, ktorý správu obsahujúcu túto cestu odoslal.
- `In` (Inform Neighbors) spracuje prvého suseda uloženého v bázi znalostí a zavolá pre neho plány `Si` a následne `Sp`.
- `Sp` (Share Path) — ak po vykonaní plánu `Si` je pre daného suseda v bázi znalostí znalosť `S`, tak vytvorí pre neho správu o tejto ceste a pošle mu ju. Či sa odosielala správa alebo nie, pokračuje sa vykonaním plánu `In`.
- `Ur` (Update Route) uloží do pamäte flash nové smerovacie údaje, vykoná plán `Ic` ak cesta k danému cieľu ešte nebola známa a oznámi susedom túto novú cestu pomocou plánu `In`. Taktiež ak sa po zavolaní plánu `Ic` rozsvietila zelená dióda tak vykoná plán `Ss` (len raz).
- `Sr` (Spreading Route) vypočíta dĺžku cesty k tomuto uzlu plánom `Pl` a ak cesta k cieľu zo správy ešte nie je známa, alebo je kratšia než uložená cesta, tak vykoná plán `Ur`.

Konfigurátor teda zelenou LED vizuálne indikuje, že má informácie o cestách ku všetkým uzlom v sieti. Ak však ešte na nejakom uzle svieti žltá LED tak to znamená, že sa ešte stále spracovávajú cesty — môže sa nájsť kratšia cesta. V prípade, že žiadny uzol nemá zasvietenú žltú LED a nejaký uzol nemá zasvietenú zelenú LED, tak zrejme nastala nejaká chyba pri konfigurácii a je možné, že bude potrebné konfiguráciu siete opakovať. Výskyt tohoto javu bude popísaný v kapitole o testovaní.

7.3 Smerovač

Smerovač má podľa návrhu čakať na žiadosti od navigátora, teda väčšinu času strávi v režime spánku až do príchodu správy. Očakávame od neho, že bude schopný reagovať na správy žiadajúce smer k určitému cieľu a na správy žiadajúce o vizuálnu signalizáciu vzdialenosti navigovaného uzlu.

Prijíma teda dva typy správ:

- `d` (Direction) — predstavuje žiadosť o smer k cieľovému uzlu, zadaného parametrom.
- `l` (Light) — žiadosť o rozsvietenie diód, ako parameter má hodnotu RSSI.

Ďalej smerovač odosiela dva typy správ, ktoré predstavujú možné odpovede na správu `d`, žiadajúcu o smer:

- `n` (Navigate) — v parametri pošle identifikátor next-hop uzlu, ktorý bol prečítaný zo smerovacej tabuľky.
- `f` (Failure) — ak sa v smerovacej tabuľke nenachádza záznam o cieli, o ktorý žiadal navigátor, tak sa pošle táto správa, signalizujúca že daná destinácia nie je známa. Nemá parametre.

Počiatočný plán smerovača blikne zelenou diódou pre signalizáciu že sa spustil agent, začne šírenie po ostatných uzloch a následne pokračuje slučkou spracovania správ:

```
$(l,(g,1))$(w,(250))$(l,(g,0))+(SA)^(nb)^(Mh)
```

Za zmienku stojí, že plány `Mh`, `Pm`, `Rs1`, `Rs2` a plány pre šírenie agenta v sieti sú identické s tými, ktoré používal konfigurátor. Nebudeme ich tu uvádzať duplicitne, celý agent bude uvedený v prílohe.

Plán `Pa` (Perform Action), tak ako aj u konfigurátora, vykoná nejaký plán na základe prijatého typu správy. Na začiatku sa blikne žltou LED, aby sa naznačilo začatie vykonávania požadovanej operácie po prijatí správy. Po správe `d` sa vykoná plán `Gd`, po správe `l` sa vykoná plán `Lg`.

```
(Pa,($(l,(y))$(w,(75))$(l,(y))&(3)*(MT,(d))^(Gd)#*(MT,(l))^(Lg)))
```

Plán `Gd` (Give Direction), okrem toho že vypne na začiatku všetky LED (ak je next-hop uzol v dosahu navigátora a by boli zapnuté LED tak by to mohlo byť mätúce, nebolo by jasné ku ktorému uzlu sa má navigátor dostať), prečíta z pamäte flash záznam o požadovanom cieli. Ak je adresa next-hop uzlu záporná, tak odpovie na žiadosť o smer správou `f`, keďže nemôže byť záporná adresa uzlu (a práve po vymazaní pamäte sú vo flash zapísané hodnoty -1). V opačnom prípade pošle odosielateľovi správy správu `n`, obsahujúcu adresu next-hop uzlu.

```
(Gd,($(l,(g,0))$(l,(y,0))$(l,(r,0))&(3)$(f,&1)&(1)$(u,(q,&3))&(3)$(f,&1)
    &(1)$(o,leq,&3,(0),(),(),())-(C,_)+(C,&1)&(1)*(C,((1)))&(1)$(t,(i))
    !(&2,(f,&1))#*(C,((0)))&(1)$(t,(i))!(&2,(n,&1,&3))))
```

Napokon pár plánov `Lg` (Light) a `Lc` (Light — Comparison) slúžia pre zasvietenie LED ako navádzanie navigátora na jeho žiadosť na tento uzol. Plán `Lg` riadi rozsvietenie LED podľa hodnoty udanej v správe. K tomu sa využívajú prahové hodnoty z bázy znalostí, a to označené ako `LRT` (Light — Red Threshold), `LYT` (Light — Yellow Threshold) a `LGT` (Light

— Green Threshold). Podľa vzdialenosti od najväčšej po najmenšiu sa rozsvecujú diódy červená-žltá-zelená (čo je aj zoradené vzostupne podľa RSSI) — nanajvýš však len jedna z nich. Plán `Lc` vykonáva porovnanie s jednotlivými prahovými hodnotami.

```
(Lc,(^(Rs2)&(1)$(o,meq,&3,&2,(),(),())-(L,_)+(L,&1)))
(Lg,($(l,(r,0))$(l,(y,0))$(l,(g,0))&(3)$(f,&1)&(1)*(LRT,_)^(Lc)*(L,((1)))
    $(l,(r,1))&(1)*(LYT,_)^(Lc)*(L,((1)))$(l,(y,1))$(l,(r,0))&(1)*(LGT,_)
    ^(Lc)*(L,((1)))$(l,(g,1))$(l,(y,0))))
```

V bázi znalostí sa nachádzajú tri znalosti, ktoré slúžia pre rozsvecovanie LED a bez nich by plán `Lg` len zhasol LED, ktoré svietili pred jeho vykonaním. Sú to:

```
(LRT,(1))(LYT,(50))(LGT,(100))
```

7.4 Navigátor

Navigátor na rozdiel od predošlých dvoch agentov nebude používať cyklický plán, ktorý len spracováva správy po ich príchode (pričom dovtedy pasívne čaká), ale bude mať upravené plány `Mh` a `Pm` — z toho dôvodu, že je vyžadovaná manuálna slučka schopná kontrolovať, či už neuplynul čas na odpoveď. Preto budú tieto plány rozšírené o kontrolu počtu cyklov čakania bez nejakej prijatej správy, ak sa má tento uzol navigovať k nejakému cieľu. V prípade, že sa prekročí nejaká hranica pre počet cyklov bez odpovede, sa bude volať opätovne žiadosť o smer (len na ten typ správy čaká navigátor odpoveď).

Navigátor musí byť schopný spracovať aspoň nasledujúce typy správ:

- `t` (Target) — udáva v parametri cieľ, ku ktorému sa má navigátor dostať; začne navigovanie k tomuto uzlu.
- `n` (Navigate) — je to odpoveď od smerovača, obsahujúca adresu uzlu ku ktorému by sa mal presunúť navigátor.
- `f` (Failure) — odpoveď od smerovača, signalizujúca že smerovač nepozná cestu k požadovanému cieľu.
- `d` (Direction) — ak by bolo viac navigátorov v sieti, tak môže dojsť táto správa od iného navigátora. Navigátor na tento typ správy bude odpovedať vždy správou typu `f`.

Typy správ odosielaných navigátorom sú:

- `d` (Direction) — žiada o adresu next-hop uzlu na ceste k cieľu (ktorý je daný parametrom správy) od nejakého uzlu (o ktorom predpokladáme že je smerovač).
- `l` (Light) — oznamuje smerovačovi, že má vizuálne indikovať vzdialenosť tohto uzlu od neho, pomocou RSSI ktoré je obsiahnuté v parametri správy.
- `f` (Failure) — odpoveď v prípade, ak došla správa `d` žiadajúca o smer k nejakému cieľu.

Upravený plán `Mh` je teda

```
(Mh,(^(Pm)$(l,(r))$(w,(75))$(l,(r))+(W)*(NN)-(W)^(Pc)#*(W)$(s)#^(Mh)))
(Pc,(&(1)*(CW,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())&(2)$(f,&1)-(CW,_)
    +(CW,&2)&(1)$(o,meq,&2,(50),(),(),())-(NQ,_)+(NQ,&1)*(NQ,((1)))
    ^(Rd)#$(w,(100))))
```

kde plán `Pc` (Perform Cycle) riadi počítanie cyklov uspatia, počas ktorých nedošla nová správa (tj. očakávaná odpoveď) a ak počet týchto cyklov prekročí určitú hodnotu (tj. 50), tak vyvolá plán `Rd` (Request Direction) žiadajúci o smer najbližšieho suseda. Ďalej tento plán uspí agenta na pevnú dobu (100 ms). Plán `Pc` sa vykoná namiesto uspania do príchodu ďalšej správy len ak nie je v bázi znalostí znalosť `NN` (Navigated Node), ktorá určuje že sa práve tento uzol niekam naviguje.

```
(Pm,(&(1)?(_)&(2)$(f,&1)-(MT,_)+(MT,&2)&(3)$(r,&1)&(2)$(f,&3)-(MS,_)
    +(MS,&2)&(1)$(r,&3)-(MC,_)+(MC,&1)^(Pa)^(Pm)*(NN)-(CW,_)+(CW,(0))))
```

Zmena v pláne `Pm` je len jedna a to tá, že ak je v bázi znalostí `NN` tak sa nastaví počítadlo cyklov bez prijatej správy na nulu po vykonaní akcie.

```
(Pa,($(l,(y))$(w,(75))$(l,(y))&(3)*(MT,(d))&(3)$(t,(i))!(&2,(f,&3))#
    *(MT,(t))^(St)#*(MT,(n))^(Nv)#*(MT,(f))^(Fr)))
```

Plán `Pa` blikne žltou diódou ak začne vykonávať akciu a následne podľa typu správy vykoná patričný plán. Pri príchode správy `d` rovno posiela späť správu typu `f`, pri príchode správy oznamujúcej cieľ cesty vyvolá plán `St`, pri príchode správy `n` započne navigáciu plánom `Nv` a ak príde správa `f`, tak vykoná plán `Fr`.

```
(St,(+(NN)$(l,(g,0))$(l,(y,1))$(l,(r,0))&(3)$(f,&1)-(DST,_)+(DST,&3)^(Rd)))
(Rd,(*(DST,_)-(NB,_,_)$(n)$(w,(250))+(D)&(1)*(NB,_,_)-(D)-(CD,_)-(CN,_)
    -(DN,_)^(Fcn)&(1)*(DST,_)^(Rs2)&(3)$(e,(c,&2))&(1)*(CN,_)^(Rs2)&(1)
    $(t,(i))!(&2,(d,&1,&3))&(1)#*(D)$(l,(r,1))$(w,(5000))$(l,(r,0))^(Rd)))
(Fcn,(&(2)$(f,&1)-&2&(1)$(r,&2)&(2)$(f,&1)-(TN,_)+(TN,&2)+(DN,&2)&(3)
     $(r,&1)+(NCN)&(1)*(CD,_)-(NCN)^(Rs2)&(1)$(f,&3)&(3)
     $(o,mor,&1,&2,(),(),())-(N,_)+(N,&3)&(3)*(N,((1)))-(CD,_)+(CD,&1)&(1)
     *(TN,_)^(Rs2)-(CN,_)+(CN,&2)#*(NCN)&(1)$(f,&3)+(CD,&1)+(CN,&2)#&(1)
     *(NB,_,_)^(Fcn)))
```

Plán `St` pridá do bázy znalostí znalosť `NN`, rozsvietením žltej diódy naznačí že je navigovanie v priebehu, uloží cieľ cesty zo správy do bázy znalostí ako `DST` a volá plán `Rd`. `Rd` (Request Direction) nájde najbližšieho suseda pomocou plánu `Fcn` (Find Closest Neighbor) a pošle mu správu typu `d` žiadajúcu o smer. Ak nie je prítomný žiadny sused, tak sa rozsvieti červená dióda, po piatich sekundách sa vypne a opakuje sa pokus. Plán `Fcn` prejde všetkých susedov v bázi znalostí a nájde suseda s najvyššou hodnotou RSSI, tj. najbližšieho suseda.

```
(Fr,(-(DN,&2)&(1)*(DST,_)^(Rs2)+(Q)&(1)*(DN,_)-(Q)&(3)$(f,&1)-&3&(1)$(r,&3)
    &(3)$(f,&1)&(1)$(t,(i))!(&3,(d,&1,&2))#*(Q)$(l,(r,1))$(w,(5000))
    $(l,(r,0))^(Rd)))
```

Príchod správy o zlyhaní získavania smerovacích informácií spracováva plán `Fr`. Ak je v bázi znalostí uložený ešte nejaký susedný uzol, ktorý sme nežiadali o smer (tj. `DN`, ktoré sa do bázy znalostí pridávali v pláne `Fcn`), tak ho oň požiadame. V opačnom prípade rozsvietime červenú diódu a po piatich sekundách vykonáme plán `Rd`, čím sa opakuje celý proces žiadania smeru od susedov.

```
(Nv,(&(3)$(f,&1)&(1)$(e,(c,&3))+(NHU)-(NH,_)&(3)*(DN,&1)-(NHU)+(NH,&1)#
    *(NHU)+(NH,&2)#+(HU)^(Un)))
(Un,(*(NH,_)*(DST,_)-(NB,_,_)$(n)$(w,(250))^(Gn)*(HU)-(HU)-(NH,_)^(Rd)))
(Gn,(^(Pn)&(2)*(NH,&1)^(Hf)))
(Pn,(-(Y)&(3)*(NB,_,_)&(1)$(f,&3)-&1&(3)$(r,&1)&(1)$(f,&3)+(Y)&(2)*(NH,&1)
    -(Y)#*(Y)^(Pn)))
(Hf,(&(1)*(TH,_)^(Rs2)&(1)$(r,&3)&(3)$(f,&1)&(1)$(o,meq,&3,&2,(),(),())
    -(TM,_)+(TM,&1)&(1)*(TM,((1)))^(Rh)#*(TM,((0)))&(1)*(NH,_)^(Rs2)&(1)
    $(t,(i))!(&2,(l,&1,&3))^(Un)))
(Rh,(+(E)-(HU)&(2)*(NH,_)^(Rs1)$(f,&2)-(NH,_)&(3)*(DST,&1)-(E)-(DST,_)-(NN)
    $(l,(y,0))$(l,(g,1))&(2)$(t,(i))!(&1,(l,&2,(0)))#*(E)-(E)&(2)*(DST,_)
    &(3)$(f,&2)&(2)$(r,&3)&(3)$(f,&2)&(2)$(t,(i))!(&1,(d,&2,&3))))
```

Posledná skupina plánov je zodpovedná za samotnú funkcionalitu navigovania. Po prijatí správy `n` sa vykoná plán `Nv`, ktorý skontroluje či next-hop adresa bola medzi detegovanými susedmi pri žiadaní o smer a ak áno, tak nastaví túto adresu ako `NH` (Next Hop), inak nastaví odosielateľa správy ako `NH`. Ďalej sa zavolá plán `Un`.

Plán `Un` (Update Neighbor) získa zoznam susedov, zavolá plán `Gn` a ak po ukončení plánu `Gn` zostane prítomná znalosť `HU` (Hop Unreachable), tak požiada znovu o smer plánom `Rd` (keďže sa navigátor zrejme príliš vzdialil od uzlu `NH`). V pláne `Gn` sa najprv spracuje zoznam susedov plánom `Pn` (Process Neighbors) a to tak, že sa nájde sused, ktorého adresa je rovnaká ako `NH` — ak sa takýto sused našiel plánom `Pn`, tak sa vykoná plán `Hf`.

Samotný plán `Hf` (Hop Found) následne porovnáva RSSI hodnotu next-hop uzlu s prahovou hodnotou `TH`, uloženou v bázi znalostí. Táto hodnota rozhoduje, pri akej sile signálu navigátor je presvedčený, že dorazil k uzlu ku ktorému mal dojsť. Ak táto prahová hodnota nebola dosiahnutá, tak sa pošle správa `l` obsahujúca RSSI uzlu `NH` a opakuje sa plán `Un` odznova. V opačnom prípade sa vykoná plán `Rh`.

Plán `Rh` (Reached Hop) sa teda vykonáva, ak sa navigátor dostal dostatočne blízko k ďalšiemu uzlu na ceste k cieľu. Ak bola dosiahnutá prahová hodnota a uzol, ku ktorému došiel navigátor bol cieľom navigovania, tak sa rozsvieti na uzle s navigátorom zelená dióda, zhasne sa žltá dióda a odstránia sa z báze znalostí tie znalosti, ktoré špecifikovali že sa jedná o navigovaný uzol, next-hop adresu či celkový cieľ cesty. Ak to bol iný uzol, tak ho navigátor požiada o ďalší smer k cieľu.

Báza znalostí kvôli plánu `Hf` musí obsahovať teda aspoň jednu znalosť — znalosť prahovej hodnoty pre rozhodnutie o dosiahnutí cieľa. Na začiatku teda báza znalostí vyzerá takto:

```
(TH,(150))
```

Keďže sa očakáva, že po spustení dostane navigátor cieľ jeho cesty oznámený prostredníctvom správy, tak má jednoduchý počiatočný plán:

```
$(l,(g,1))$(w,(250))$(l,(g,0))^(Mh)
```

Potom teda po nahraní agenta na uzol je očakávané, že navigátor dostane správu špecifikujúcu cieľ jeho cesty (napríklad od base station). Príkladom takej správy je `(t,(1),(2))` pre navigovanie k uzlu s označením 2.

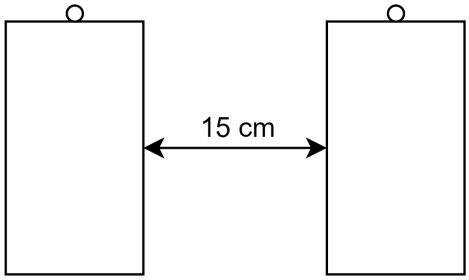
Kapitola 8

Testovanie

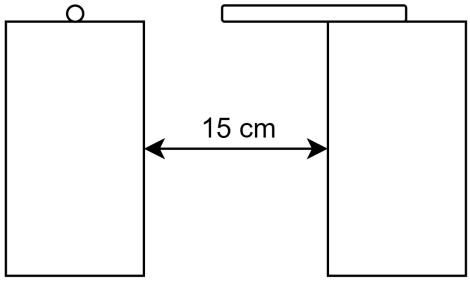
V tejto kapitole popíšeme priebeh experimentovania s jednotlivými agentmi a problémy, ktoré boli odhalené pri experimentovaní a prípadné riešenia k týmto problémom, ktoré boli zvolené.

8.1 Použitie RSSI ako metriky vzdialenosti

Prvým problémom, spoločným pre všetkých agentov je diskrétny počet úrovní RSSI. Síce je RSSI mapované platformou na hodnoty v intervale <0;255>, reálne ale samotné uzly IRIS dokážu rozlíšiť len 29 úrovní, teda celé čísla 0-28 ako silu signálu. Preto sa platformou tieto hodnoty násobia koeficientom 9, aby boli zmerané hodnoty rovnomerne rozprestrené naprieč množinou hodnôt, ktoré je možné uložiť do jednobajtovej premennej. Takéto spracovávanie týchto hodnôt však vzbudzuje prvý dojem, že presnosť merania RSSI je podstatne vyššia, než reálne je. Tento problém však nanešťastie nie je možné reálne riešiť inak, než výmenou uzlov IRIS za iný typ uzlov, ktorý by dokázal merať presnejšie hodnoty RSSI, keďže tento problém je hardvérového charakteru — takéto riešenie by ale potrebovalo, aby sa platforma WSageNt upravila pre podporu nových typov motes, keďže aktuálne podporovanými typmi motes sú len MicaZ a IRIS.



Obr. 8.1: Antény oboch uzlov sú pri meraní natočené rovnako

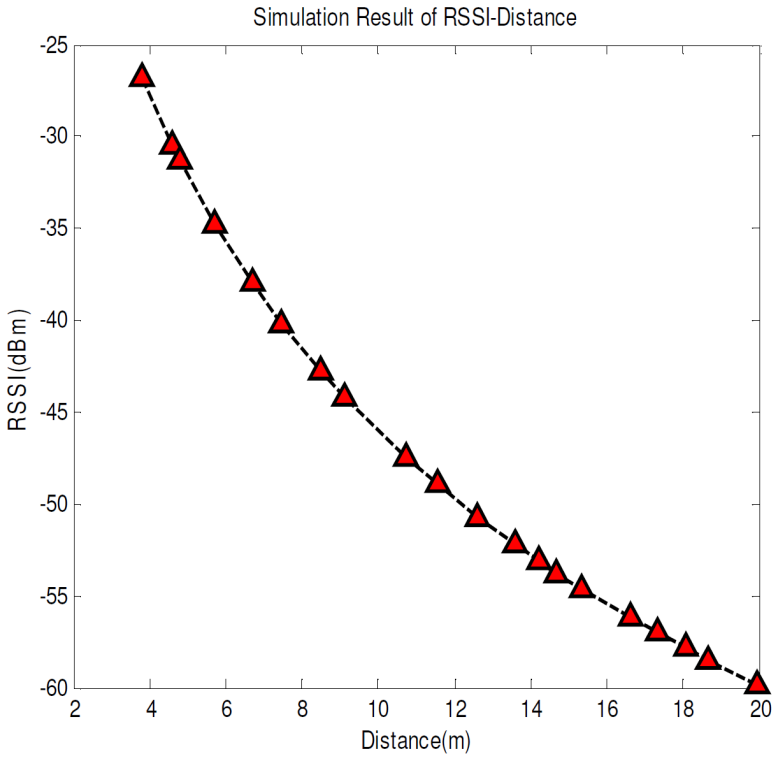


Obr. 8.2: Anténa druhého uzlu je natočená o 90°

Meranie samotného RSSI sa preukázalo ako dosť nespoľahlivá metóda určovania vzdialenosti. Pri pohybe uzlu v sieti často namerané hodnoty RSSI neodpovedali verne skutočnej vzdialenosti, k čomu prispievalo viacero faktorov, napríklad aj natočenie uzlu pri prenose, ale hlavne smerovanie antén uzlov (ktoré teda menila aj rotácia samotných uzlov) — keďže sa pochopiteľne nejedná o izotropné žiariče, tak má to na efektívnosť príjmu a odosielania správ značný vplyv. Vplyv tohto problému je možné sledovať jednoduchým plánom:

```
(EchoRSSI,(-(NB,_,_)$(n)$(w,(100))&(1)*(NB,_,_)$(l,(r))
          !(1,RSSI,&1)$(w,(1000))^(EchoRSSI)))
```

ktorý cyklicky každú sekundu vyhľadá dostupných susedov, prepne červenú LED a pošle zoznam susedov na base station. Po umiestnení dvoch uzlov (rovnako natočené, v jednej výške vzdialené od seba približne 15 cm, znázornené na obrázkoch 8.1 a 8.2) sme s týmto plánom kontrolovali silu RSSI podľa natočenia antény druhého uzlu (tj. výbežok na vrchu obrázkov vyššie). V prípade rovnobežne natočených antén (obr. 8.1) sme namerali hodnotu RSSI 153 alebo 162. Ak sme však natočili jednu z antén o 90° (obr. 8.2), tak RSSI zmerané uzlom vľavo na obrázku bolo 2-3 rozlíšiteľné úrovne merania RSSI nižšie, teda 126/135. Keďže nie je možné zistiť v rámci programovej časti rotáciu antény a bez IMU senzorovej dosky ani rotáciu samotného uzlu, tento problém taktiež nie je korigovateľný (inak než dbaním na to, že budú mať všetky uzly správne natočené antény). Nepresnosť zmeraného RSSI, spôsobenú nesprávne otočenými anténami je teda potrebné brať do úvahy pri voľbe parametrov navigácie — a to konkrétne prahové hodnoty, ako napríklad `TH` či prahy požadovaných hodnôt RSSI pre signalizovanie vzdialenosti, keďže sa prahové hodnoty pri nesprávnej voľbe nemusia vôbec dosiahnuť, čo môže mať za následok napríklad zlú funkčnosť navigátora, alebo dokonca priamo jeho nefunkčnosť.



Obr. 8.3: Pokles RSSI v závislosti od vzdialenosti[1]

Ďalším problémom súvisiacim s RSSI bolo vyhodnocovanie cesty. Metóda zvolená v sekcii 6.1.1 má tendenciu nastavovať ako next-hop adresu cesty k nejakému cieľu samotný ten cieľ, ak je možné prijať hoci len najslabšiu možnú silu signálu tohoto uzlu. Tento jav sa vyskytuje pri pričítaní hodnoty 255 (max. možná hodnota RSSI) k hodnote invertovaného

[1] Prebrané z [2]

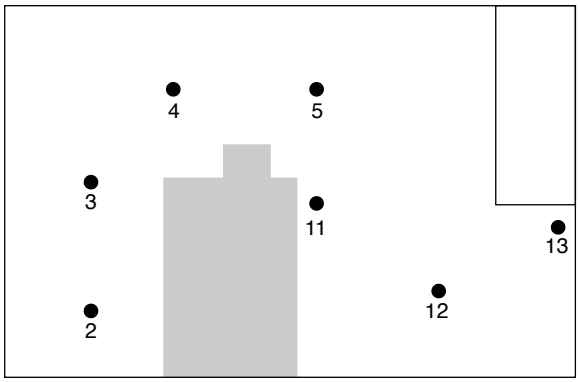
RSSI. Je to spôsobené tým, že sila signálu neklesá lineárne so vzdialenosťou (klesá s druhou mocninou vzdialenosti) — teda pokles zmeraného RSSI je tým rýchlejší na jednotku vzdialenosti, čím sú bližšie tieto uzly k sebe navzájom. Túto skutočnosť vyobrazuje simulácia zmeny RSSI v závislosti od vzdialenosti, vyobrazená v grafe na obrázku 8.3.

Na danom obrázku, pochádzajúcom z [2], je možné badať znateľne nižší pokles RSSI napríklad medzi vzdialenosťami 15 a 16 metrov, než medzi vzdialenosťami 4 a 5 metrov. To v praxi znamenalo, že veľmi vzdialené uzly sa mohli podľa RSSI zdať byť bližšie než reálne boli a naopak, blízke uzly sa mohli zdať byť ďalej. Preto sa suma niekoľkých kratších skokov na ceste (aj keď bola zmeraná hodnota RSSI na každý skok hoci len o 80 menšia od najvyššej možnej hodnoty) vyhodnotila ako dlhšia než jeden skok na cieľový uzol, ktorý mal veľmi nízku zmeranú hodnotu RSSI (napr. 18, čo je o 237 menej než najvyššia možná hodnota) — a ktorý teda mohol byť reálne dosť ďaleko.

	2	3	4	5	11	12	13
2	2/0	3/129	4/138	5/174	11/183	12/228	13/201
3	2/129	3/0	4/129	5/129	11/156	12/183	13/210
4	2/138	3/129	4/0	5/120	11/165	12/183	13/174
5	2/183	3/129	4/120	5/0	11/174	12/165	13/165
11	2/183	3/156	4/165	5/174	11/0	12/147	13/129
12	2/219	3/183	4/174	5/156	11/138	12/0	13/156
13	2/201	3/210	4/174	5/156	11/129	12/156	13/0

Tabuľka 8.1: Výsledná konfigurácia siete po behu konfigurátora s posunom 255

Tento jav je možné pozorovať na smerovacej tabuľke, ktorá vznikla behom konfigurátora s posunom o hodnotu 255. V tabuľke 8.1 riadky predstavujú jednotlivé uzly, na ktorých bežal konfigurátor a stĺpce predstavujú jednotlivé cieľové uzly v danej sieti. Bunky potom obsahujú dve hodnoty: prvá (pred lomkou) je adresa next-hop uzlu na ceste z uzlu daného riadkom k cieľu danému stĺpcom a druhá ja cena danej uloženej cesty. Z tejto tabuľky je možné vyčítať, že každý uzol mal ako next-hop adresu na ceste k nejakému cieľu vždy priamo samotnú adresu daného cieľa, aj keď cena daného skoku bola vysoká — napríklad z uzlu 12 je cena priamej cesty na uzol 2 ohodnotená 219, teda zmerané RSSI bolo rovné 36, čo je štvrtá najnižšia úroveň sily signálu. Sieť, v ktorej bol tento konfigurátor testovaný, je znázornená na obrázku 8.4. Všetky uzly boli umiestnené vo výške približne 50 cm a šedá zóna znázorňuje prekážku, vysokú približne 120 cm.



Obr. 8.4: Rozmiestnenie uzlov v miestnosti pri testovaní

Jedným možným riešením by bolo ignorovanie nízkeho RSSI pre konfiguráciu, čo by však kládlo obmedzenie na maximálnu možnú vzdialenosť medzi uzlami. Druhým navrhovaným riešením je zmena hodnoty posunu z 255 na nižšie číslo — ideálne také, kde sa už dá pokles RSSI podľa vzdialenosti dostatočne linearizovať. To by však mohlo generovať nekonečné cykly za účelom znižovania ceny cesty, keďže by pre veľmi blízke uzly s vysokým RSSI bolo možné mať zápornú hodnotu ceny cesty medzi dvoma uzlami. Preto by bolo nutné tento prípad explicitne ošetriť, napríklad nastavením ceny cesty medzi dvoma rôznymi uzlami na hodnotu 1, ak bola po inverzii a posune RSSI nižšia alebo rovná nule. Zápornou stránkou tohoto riešenia je možná voľba mierne dlhších skokov na veľmi krátke vzdialenosti, práve tam kde sa aplikuje ošetrenie voči záporným cenám ciest medzi dvoma skokmi. Tento zápor je však akceptovateľný a toto riešenie sa bude používať pre riešenie tohoto problému.

Tento prístup spôsobuje, že sa pre konfiguráciu siete susedné uzly s vyššou hodnotou RSSI javia, akoby boli všetky rovnako vzdialené od daného uzlu. Samozrejme, zvolená hodnota posunu má na to značný efekt — avšak aj pri hodnote posunu menšej než je polovica maximálneho RSSI (teda 125) je u sietí s menšou silou signálu medzi jednotlivými uzlami stále badateľné, že uzly preferujú priame spojenia k cieľom. Na druhú stranu je však potrebné konštatovať, že vhodná voľba tohoto posunu je dosť závislá od prostredia, v ktorom bude práve konfigurátor spustený.

	2	3	4	5	11	12	13
2	2/0	3/1	3/3	4/17	3/28	12/71	4/45
3	2/1	3/0	4/1	5/1	4/27	5/25	5/45
4	2/8	3/1	4/0	5/1	11/26	5/18	5/19
5	3/9	3/1	4/1	5/0	12/34	12/17	12/18
11	3/34	3/26	4/35	13/26	11/0	12/17	13/1
12	5/26	5/43	4/44	5/17	11/8	12/0	13/1
13	12/34	11/27	4/44	12/25	11/1	12/8	13/0

Tabuľka 8.2: Výsledná konfigurácia siete po behu konfigurátora s posunom 125

V tabuľke 8.2 je možné vidieť výslednú smerovaciu tabuľku po behu konfigurátora, ktorý mal nastavenú hodnotu posunu na 125 a pre ceny skokov, ktoré boli záporné po pripočítaní tejto konštanty, nastavoval cenu skoku 1. Tento konfigurátor bol spustený v rovnakej sieti, ako aj predošlý konfigurátor s hodnotou posunu 255, teda v sieti znázornenej na obrázku 8.4. Je badateľné, že u niektorých ciest boli nahradené veľmi drahé priame skoky pomocou nepriamych ciest okolo prekážok. Taktiež je aj možné sledovať, že sú ceny ciest značne nižšie, čo spôsobuje hodnota posunu, keďže najhoršia možná cena jedného skoku je rovná hodnote tohto posunu a pri použitej modifikácii sú krátke skoky (ktoré by inak mali zápornú cenu) nahradené hodnotou 1.

Samozrejme, sieť sa dá konfigurovať aj manuálne pomocou jednoduchých agentov, ktorí ručne nastavia next-hop adresy ciest pre daný uzol. Alternatívne je možné použiť takéhoto jednoduchého agenta pre možnú úpravu niektorých vybraných ciest po ukončení behu konfigurátora. Ukážkou takéhoto agenta je

```
$(u,(p,2,(2,0)))$(u,(p,3,(5,0)))$(u,(p,4,(5,0)))$(u,(p,5,(5,0)))
```

ktorý konkrétne nastavuje priamu cestu k uzlu 2 a cestu k uzlom 3, 4, a 5 cez uzol 5. Samotnú cenu cesty netreba nastavovať, keďže navigátor ani smerovač nesledujú cenu; tá je používaná len pri konfigurovaní siete.

8.2 Prenos správ — spoľahlivosť

Pri testovaní behu konfigurátora nastávali situácie, kedy sa výpočet náhle zastavil — bolo to možné pozorovať tým, že zostala počas spracovávania správy svietiť žltá LED a tento stav sa nemenil. Následným skúmaním možných zdrojov tohoto problému sa odhalilo, že takáto situácia môže nastať vo viacerých prípadoch, ale vždy tento problém spôsobil modul `SendM`. Ako bude popísané ďalej v tejto sekcii, všetky tieto problémy sú spôsobené neprítomnosťou odblokovania agenta pokúšajúceho sa poslať správu, ak túto správu nemôže úspešne odoslať a teda odblokovať interpret po asynchrónnej operácii. Počas experimentov sa ale preukázalo, že niektoré problémy boli prítomné na strane prijímateľa, teda v moduli `ReceiveM`.

Už pozorovanie chovania uzlov počas konfigurovania siete naznačovalo, že sa mohlo jednať o problém pri súčasnom posielaní správ. Jednoduchý experiment s uzlami **2**, **3** a **4** (na uzol **2** je počiatočne nahraný agent) a plánom

```
$(m,(3))$(l,(y,1))!(4,Hello)$(l,(y,0))
```

preukázal na staršej verzii WSageNt (na ktorej sa začala táto práca vyvíjať), že jeden z dvoch súčasne odosielajúcich uzlov nedokončil činnosť — zostala svietiť žltá LED. Táto situácia nastala, ak uzol **4** bol nečinný — teda interpret nevykonával žiadnu činnosť, napríklad nemal žiadny plán, alebo bol pozastavený z dôvodu čakania na dokončenie asynchrónnej operácie (čo mohlo byť práve takto uviaznuté odosielanie). Problém bol na oboch stranách, ako odosielateľa tak aj prijímateľa.

Problém na strane prijímateľa spočíval v spôsobe spracovávania prijatých správ — ich vloženie do bázy vstupov sa vykonáva až na začiatku kroku interpretu a aby sa mohla prijímať ďalšia správa musí byť posledná prijatá správa vložená do bázy vstupov. Keďže sa v spomenutých prípadoch krok interpretu nevykonal buď vôbec (agent bez plánu), alebo sa vykonal až po dokončení asynchrónnej operácie (čo sa nemuselo stať vôbec), tak sa síce prvú správu podarilo úspešne prijať a jeden agent odosielajúci správu mohol pokračovať ďalej, ale ten druhý agent sa snažil neúspešne poslať jeho správu v nekonečnej slučke. Riešenie tohto problému spočívalo v povolení pridávania správ do bázy vstupov, ak aktuálne nebeží agentný kód a ani interpret — obmedzenie vkladania správy do bázy vstupov slúžilo ako prevencia situácie, kedy by sa menili indexy jednotlivých tabuliek v pamäti uzlu počas vykonávania kódu a tým pádom by sa mohlo znehodnotiť vykonávanie agentného programu — čo pochopiteľne nemôže nastať ak sa nevykonáva kód, teda toto riešenie by nemalo spôsobovať problémy.

Ako bolo spomenuté, hlavným problémom odosielania bola neprítomnosť odblokovania, ak nie je možné odoslať správu. To spôsobovalo napríklad to, že pri pokuse poslať správu na adresu neexistujúceho uzlu alebo na uzol, ktorý je mimo dosahu, odosielateľ uviazol a zostal v nekonečnej slučke. Preto bol pridaný do modulu `SendM` mechanizmus počítania pokusov pri odosielaní paketov. Maximálny povolený počet opakovaných odoslaní paketu je stanovený konštantou `SEND_RETRY_LIMIT` v hlavičkovom súbore `AMAgent.h`, pričom je aktuálne daný na hodnotu 10. Ak sa niektorý paket z odosielanej správy nepodarilo odoslať toľkokrát (alebo viackrát), ako je daná hranica, tak sa ukončí odosielanie správy, tým pádom aj blokovanie asynchrónnou operáciou a opätovne sa spustí interpret. Tento limit na opakované odosielanie paketov sa aplikuje len na bežné správy, neaplikuje sa na prenos agenta.

Keďže bolo implementované rušenie odosielania správy pri prekročení určitej hranice, bolo potrebné implementovať aj ukončenie prijímania správy na strane príjemcu. Dôvodom je to, že príjemca blokuje prijímanie ďalších správ kým sa neprijme jedna celá správa a ak by

teda odosielateľ prestal prenášať danú správu, tak by prijímateľ zostal zablokovaný a nemohol by prijať už žiadnu ďalšiu správu. Na strane prijímateľa v module `ReceiveM` bolo teda implementované obdobné počítanie ako na strane odosielateľa — cyklicky sa volá jednorázový časovač, ktorý inkrementuje počítadlo sekúnd, kedy nebol prijatý paket. Ak hodnota v počítadle prekročí hranicu danú konštantou `SEND_RETRY_LIMIT`, tak sa ukončí prijímanie tejto správy. Na strane príjemcu bola použitá ostrá nerovnosť, aby bola poskytnutá väčšia voľnosť pri omeškaní prenosu.

8.3 Vyhľadávanie susedných uzlov

Ďalší problém relevantný k správam spočíva v spôsobe hľadania susedných uzlov. K tomu sa využíva všesmerové vysielanie (angl. *broadcast*), na ktoré odpovedajú prijímatelia správy späť odosielateľovi. Toto vysielanie však nezaručuje doručenie správy prijímateľom. Problém spočíva v tom, že pri súčasnom všesmerovom vysielaní viacerými uzlami nastane kolízia — uzol teda nemusí detegovať všetky susedné uzly, keďže niektoré uzly aj napriek tomu, že sú v dosahu odosielateľa nemuseli toto vysielanie prijať (tj. problém skrytých a vzdialených terminálov). Extrémnym prípadom sú situácie, kedy ani jeden zo susedných uzlov v dosahu vysielania nepošle späť odpoveď, ktorou by sa odosielateľ dozvedel že daný uzol je v dosahu a aká je sila signálu tohoto uzlu. Táto situácia by v prípade navigátora vyvolala len opätovné vyhľadávanie susedných uzlov po menšom čakaní, ale v prípade konfigurátora by mohla spôsobiť značne väčšie komplikácie.

Práve u konfigurátorov sa občas vyskytol tento problém, že nejaký uzol pri vykonaní plánu `Ss` (tj. agent pošle všetkým susedom správu o ceste k sebe samému) nenašiel žiadnych susedov a teda sa nemohli šíriť informácie o ceste k tomu uzlu, ktorý ten plán vykonal. Ak nastala táto situácia, tak bola vizuálne indikovaná tým, že práve jeden (a žiadny iný) uzol v sieti svietil na zeleno, pričom už žiadny uzol v sieti nespracovával žiadne správy (nesvietila žltá LED). Pri testovaní znalostí ostatných uzlov v sieti o cestách im chýbala práve cesta k tomu uzlu, ktorý mal zapnutú zelenú LED. Frekvencia, s ktorou nastávala táto situácia pri konfigurovaní sa zvyšovala s počtom uzlov, ktorým sa prekrývali dosahy ich signálu a s počtom uzlov, ktoré súčasne chceli vysielať — teda ak bolo veľa uzlov vedľa seba v tesnej blízkosti (napr. na stole) a práve začínala konfigurácia (kedy je veľmi početné vyhľadávanie susedných uzlov), tak bola vysoká pravdepodobnosť, že na nejakom uzle nastane tento problém.

Tento problém bol vyriešený programovo a to tak, že sa v pláne `Ur` vykoná znovu plán `Ss`, ak bol prijatím poslednej správy dosiahnutý počet spracovaných ciest k uzlom zhodný s počtom uzlov v sieti (teda ak sa práve rozsvietila zelená dióda). Toto riešenie (aj keď dosť triviálne) rieši tento problém vďaka tomu, že v čase získania poslednej cesty, ktorá ešte nebola známa, je už podstatne menej frekventované všesmerové vysielanie a tiež je výrazne nižšia šanca, že nastane tomu istému uzlu úplné prekrytie signálu až dvakrát.

Napokon sa ešte pri testovaní prejavilo ako občasne problémové šírenie agentov po sieti. Kód prevzatý z práce p. Houšťě [7] sa presúva na prvého suseda obsiahnutého v bázi znalostí, na ktorom tento agent ešte nebol. Keďže títo susedia však nie sú nijak usporiadaní, môže sa jednať aj o veľmi vzdialený uzol — tým sa teda agent môže snažiť presunúť na uzol, s ktorým nie je možná spoľahlivá komunikácia a presun môže trvať veľmi dlho, kým sa neprenesie úspešne celý agent, alebo v horšom prípade sa presun nedokončí vôbec. Alternatívny prístup by vyžadoval hľadanie najbližšieho suseda, čo je však pomalé — najmä ak by zostal nenavštívený len veľmi vzdialený sused a musel by sa zoznam susedov prechádzať viackrát.

Vyhľadávanie susedov tiež občas nenájde všetkých susedov bez dodatočného čakania. Samotné vyhľadávanie susedov funguje tak, že sa pošle všesmerová správa na objavenie susedov a následne každý uzol, ktorý ju prijal, pošle späť špeciálnu odpoveď. Automaticky po prijatí tejto odpovede uzlom, ktorý vyhľadával susedov, sa vkladajú informácie do interného bufferu, z ktorého sa následne vložia do bázy znalostí na začiatku ďalšieho kroku interpretu. Táto operácia však čaká len pevnú dobu 100 ms pred tým, než sa ukončí táto asynchrónna operácia a tým sa opätovne spustí vykonávanie kódu.

To v praxi však znamená, že ak sa priamo po ukončení tejto operácia prehľadáva báza znalostí na nájdených susedov, tak sa v nej nemusia nachádzať všetky susedné uzly, ktoré nakoniec poslali odpoveď — niektoré uzly sa mohli oneskoriť v posielaní odpovede apod. Dokonca ani dodatočné čakanie 100 ms po ukončení tejto operácie občas nestačilo na to, aby boli prijaté odpovede od všetkých susedných uzlov — to bolo badateľné najmä u navigátora, ktorý začal spontánne uprostred navigácie (k ďalšiemu uzlu na ceste) opätovne žiadať o smer najbližšieho nájdeného suseda, aj keď daný uzol bol v dosahu na vzdialenosť zopár metrov nezastavanou vzdušnou čiarou. Riešenie tohto problému spočívalo v zvýšení doby čakania po vykonaní akcie vyhľadávania susedov zo 100 ms na 250 ms. Znamenalo to značné zvýšenie spoľahlivosti, mierne na úkor rýchlosti — vzhľadom na dobu vykonávania zložitejších plánov sa však nejedná o badateľné spomalenie.

8.4 Beh agentov a interpret

V tejto sekcii budú spomenuté niektoré problémy, ktoré boli sa prejavili či už pri tvorbe agentov, alebo pri testovaní behu a nespadajú do predošlých dvoch sekcií.

Prvý problém je spracovávanie zoznamov po jednom. Tento problém je prítomný ako u konfigurátora, tak aj u navigátora. Vplyv tohto problému je čiste z výkonnostného hľadiska — na funkčnosť to nemá efekt. Kým u konfigurátora príliš neprekáža dlhší beh vykonávania akcií, u navigátora je to inak. Ideálne by navigátor mal byť čo najresponzívnejší, aby sa čo najčastejšie aktualizovalo vizuálne indikovanie vzdialenosti navigátora od daného uzlu a aby nájdený susedný uzol s najvyšším RSSI bol stále uzlom s najvyšším RSSI v dobe, kedy mu pošleme správu. Keďže však pre také úlohy, ako vyhľadávanie najbližšieho suseda je potrebné prejsť celý zoznam všetkých susedov a vykonávať porovnávanie u každého s nich s doterajším najbližším susedom, tak časová náročnosť vykonávania takéhoto plánu stúpa lineárne s počtom susedných uzlov. Preto klesá v tomto ohľade výkonnosť v komplikovanejších sieťach, kde má navigátor dosah na väčší počet susedov súčasne a kvôli tomu sa môže zdať byť málo responzívny, alebo dokonca až úplne zamrznutý.

Ďalším problémom bola malá maximálna veľkosť agentov. Keď ešte boli niektorí agenti spojení v priebehu vývoja, tak nastávala situácia kedy začali zlyhávať plány pretože nebol dostatok miesta v pamäti. Vzhľadom na to, že bola platforma WSageNt počiatočne vyvíjaná na platformu MicaZ, ktorá mala polovičnú kapacitu pamäti RAM, bolo možné navýšiť maximálnu veľkosť agenta na 5000 B, čo by už malo byť dostatočné pre normálny beh agentov.

Dosť špecifickým problémom interpretu je zvláštne fungujúca substitúcia registrov pri testovaní bázy znalostí. Substitúcia registrov u tejto operácie funguje správne len pre prvý register, pri pokuse o použitie zvyšných dvoch registrov na substitúciu u tejto operácie spôsobí vždy zlyhanie operácie, aj ak bola táto znalosť priamo predtým pridaná do bázy znalostí. Tento problém však bolo našťastie možné celkom jednoducho obísť, teda v konečnom dôsledku nemá vplyv na funkčnosť ani výkonnosť agentov (až na zopár možných akcií navyše).

Posledným problémom, na ktorý sa natrafilo pri vývoji, sú miestami neúplné alebo nepresné informácie o službách a akciách poskytovaných platformou. Napríklad v práci p. Lichého [10] je uvedené v sekcii 6.1.8, že akcia odosielania správy zlyhá, ak sa nepodarilo odoslať správu — to nanešťastie nie je pravda, ako bolo skôr spomenuté; ak by to bola pravda tak by to umožnilo podstatne lepšie reagovanie na neúspech odosielania správy. Dokonca vzhľadom na to, že sa jedná o asynchrónnu operáciu nie je ani implementovateľné, aby priamo akcia odosielania mohla zlyhať pri neúspechu odosielania (bolo by možné implementovať ďalšiu službu, ktorá by zistila či naposledy odosielaná správa bola odoslaná úspešne).

Nedokumentovanou akciou, ktorá sa dá vykonať je posielanie všesmerovej správy pomocou akcie odosielania správ — je to možné pomocou anonymnej premennej (`_`), ak sa využije namiesto adresy cieľového uzlu. V tom prípade sa doručí cieľovému uzlu okrem samotnej správy aj RSSI — tieto dva údaje sú uložené v dvojici. Takéto správy by však bolo problematické rozpoznávať od bežných správ a taktiež by bolo problematické ich spracovávanie. Keďže by využitie týchto správ vyžadovalo väčšie zásahy do štruktúry agentov a všesmerové vysielanie nezaručuje, že sa správy doručia (a navyše k tomu trpia viacerými problémami, ktoré sa rozoberali v sekcii o prenose správ), tak sa tieto správy nepoužívajú (posielajú sa len tie všesmerové správy, ktoré sú nutné pre detekciu susedných uzlov a sú posielané v rámci služby `n`, nie `!`).

Kapitola 9

Záver

Táto diplomová práca úspešne implementovala riešenie pre navigáciu uzlu v budovách, čo bolo jej hlavným cieľom. K tomu boli využité bezdrôtové senzorové siete a multiagentné systémy — využívajúc platformu WSageNt pre interpretáciu agentov, ktorá beží na operačnom systéme TinyOS.

Najprv boli rozobrané znalosti o bezdrôtových senzoroch a taktiež boli uvedené informácie o bezdrôtových senzorových sieťach, ktoré sú potrebné pri ďalšom riešení problému navigácie s využitím týchto technológií. Ďalej boli prebrané metódy lokalizácie objektu v priestore, ktoré sú použiteľné v senzorových sieťach — spôsoby, akými je možné zmerať vzdialenosť hľadaného objektu a ako túto zmeranú vzdialenosť použiť k určeniu presnej lokácie. Tiež boli uvedené aj základné informácie o operačnom systéme TinyOS, ktorý je použiteľný na tvorbu aplikácií vo WSN.

Potom boli uvedené informácie o umelých agentoch a multiagentných systémoch, ktoré sú aplikovateľné v bezdrôtových senzorových sieťach, vzhľadom na určité charakteristické črty ktoré majú tieto systémy spoločné. Nakoniec bol uvedený agentný jazyk ALLL, ktorý je interpretovateľný platformou WSageNt na zariadeniach s operačným systémom TinyOS.

Na základe týchto nadobudnutých znalostí bolo možné navrhnúť riešenie za využitia viacerých agentov, ktoré tiež vyžadovalo tvorbu rozšírenia platformy WSageNt, aby bolo možné dosiahnuť požadovanej funkcionality. Navrhnuté riešenie sa podarilo úspešne naimplementovať a vzniklo tým rozšírenie platformy WSageNt, ktoré umožňuje do pamäte flash ukladať numerické dáta, ako napríklad smerovacie informácie. Ďalej boli vytvorení traja agenti, ktorí realizujú požadovanú funkčnosť v sieti: konfigurátor, smerovač a navigátor.

RSSI sa preukázalo ako použiteľná technika pre meranie vzdialenosti, ale s určitými problémami s presnosťou, ako to bolo konštatované v kapitole o testovaní. Jeho nelineárny pokles so vzdialenosťou, relatívne nízky počet rozpoznávaných úrovní na bezdrôtových senzorových uzloch a premenlivosť pri pohybe v priestore spôsobuje komplikácie pri navigovaní. Aj napriek týmto problémom je však implementované riešenie funkčné.

Práca by sa dala v budúcnosti ďalej rozšíriť pre funkčnosť v dynamických sieťach, keďže sme sa v tejto práci sústredili na statické siete. Ďalej by bolo možné rozšíriť prácu o možnosť ukladania viacerých ciest k cieľom do smerovacej tabuľky pre vyššiu odolnosť v prípade výpadkov alebo o pridanie ochrany voči možným úmyselne škodiacim agentom tretích strán. Implementáciou ukladania číselných hodnôt do stáleho úložiska dát taktiež vznikla možnosť tvorby mnohých ďalších aplikácií v bezdrôtových sieťach, ktoré by túto funkcionalitu mohli využiť.

Literatúra

[1] *Wireless Sensor Networks and their Applications*. [Online; navštíveno 15.01.2019]. URL https://www.elprocus.com/introduction-to-wireless-sensor-networks-types-and-applications/

[2] Adewumi, O. G.; Djouani, K.; Kurien, A. M.: RSSI based indoor and outdoor distance estimation for localization in WSN. In *2013 IEEE International Conference on Industrial Technology (ICIT)*, Feb 2013, s. 1534–1539, doi:10.1109/ICIT.2013.6505900.

[3] Felner, A.: Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm. 01 2011.

[4] Fischione, C.: *An Introduction to Wireless Sensor Networks*. Sep 2014, [Online; navštívené 15.01.2019].
URL https://www.kth.se/social/files/5431a388f276540a05ad2514/An_Introduction_WSNS_V1.8.pdf

[5] Hanáček, P.: *Bezdrátové a mobilní systémy – materiály k prednáškam*. [Online; dostupné 15.01.2019].
URL https://www.fit.vutbr.cz/study/courses/BMS/public/bms.htm

[6] Horáček, J.: *Platforma pro mobilní agenty v bezdrátových senzorových sítích*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2009.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=8922

[7] Houšť, M.: *Monitorování stavu bezdrátových senzorových sítích agenty*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2011.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=12684

[8] Jr., F. Z.; Zboril, F.: Simulation for Wireless Sensor Networks with Intelligent Nodes. IEEE, 2008, ISBN 9780769531144, s. 746–751.

[9] Levis, P.: *TinyOS programming*. Cambridge: Cambridge University Press, prvé vydanie, 2009, ISBN 978-0-521-89606-1.

[10] Lichý, S.: *Agentní platforma pro bezdrátové senzorové sítě*. Diplomová práca, Vysoké učení technické v Brně, Fakulta informačních technologií, 2013.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=13891

[11] Martincic, F.; Schwiebert, L.: Introduction to Wireless Sensor Networking. 09 2005: s. 1 – 40, doi:10.1002/047174414X.ch1.

[12] Spáčil, P.: *Mobilní agenti v bezdrátových senzorových sítích*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2009.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=7959

[13] Zbořil, F.: *Agentní a multiagentní systémy – Studijní opora*. Okt 2006, [Online vo WIS FIT VUT; dostupné 15.01.2019].
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FAGS-IT%2Ftexts%2FAGS_opora_m1_ESF.pdf

[14] Zhou, Y.: A closed-form algorithm for the least-squares trilateration problem. *Robotica*, ročník 29, č. 3, 2011: s. 375–389, ISSN 0263-5747.

Príloha A

Obsah priloženého pamäťového média

Na priloženom pamäťovom médiu sa nachádza:

1. priečinok `wsagent` — kód upravenej platformy WSageNt
2. `configurator.txt` — kód konfigurátora v textovej podobe
3. `router.txt` — kód smerovača v textovej podobe
4. `navigator.txt` — kód navigátora v textovej podobe
5. priečinok `thesis` — zdrojové kódy pre preloženie textu tejto práce
6. `thesis.pdf` — elektronická podoba tejto práce, odovzdaná do WIS FIT VUT
7. `thesis_print.pdf` — podoba tejto práce určená pre tlač
8. `poster.pptx` — vytvorený plagát v upraviteľnej podobe
9. `poster.pdf` — vytvorený plagát vo formáte PDF
10. `readme.txt` — súbor popisujúci obsah média, štruktúru súborov s kódmi agentov a ďalšie dodatočné informácie

Príloha B

Kód agentov

B.1 Konfigurátor

Počiatočný plán

```
$(l,(g,1))$(w,(250))$(l,(g,0))+(SA)^(nb)
$(u,(e))&(1)$(t,(i))+(DC,(1))+(D,&1)$(u,(p,&1,(&1,(0))))
*(RC)-(RC)$(l,(y,1))+(NC,(0))^(Cn)-(NB,_,_)$(n)$(w,(250))^(Tn)
$(l,(y,0))$(w,(5000))$(l,(y,1))^(Ss)$(l,(y,0))
#+(AI)^(Mh)
```

Knižnica plánov

```
(Mh,(^(Pm)$(l,(r))$(w,(75))$(l,(r))$(s)^(Mh)))
(Pm,(&(1)?(_)&(2)$(f,&1)-(MT,_)+(MT,&2)&(3)$(r,&1)&(2)$(f,&3)-(MS,_)
    +(MS,&2)&(1)$(r,&3)-(MC,_)+(MC,&1)^(Pa)^(Pm)))
(Rs1,(&(1)$(f,&2)&(2)$(r,&1)&(1)))
(Rs2,(&(2)$(f,&1)&(1)$(r,&2)&(2)$(f,&1)))
(Pa,($(l,(y,1))&(3)*(MT,(c))^(Sc)#*(MT,(p))^(Sr)#$(l,(y,0))))
(nb,($(l,(r,1))&(1)$(t,(i))+(V,&1)$(n)$(w,(250))^(cn)))
(cn,(^(gn)*(SA)^(gi)^(tm)#*(SA)^(cn)))
(gi,(&(1)$(r,&2)&(2)$(f,&1)))
(gn,(+(bk)&(1)*(NB,_,_)&(2)$(f,&1)-&2-(bk)#&(1)*(bk)^(bk)))
(tm,($(t,(q,&2))-(NB,_,_)$(m,(&2,s))^(nb)))
(bk,(&(3)$(t,(b,f))+(b,&3)&(2)*(b,(1))-(SA)+(RC)$(l,(r,0))#&(2)*(SA)-(PN,_)
    +(PN,&3)$(m,&3)&(1)$(t,(i))+(C)&(2)@(*(PN,&1)-(C)^(nb))&(2)*(C)-(SA)
    $(l,(r,0))))
(Cn,(&(1)*(V,_)&(2)$(f,&1)-&2&(1)*(NC,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())
    &(3)$(f,&1)-(NC,_)+(NC,&3)^(Cn)))
(Sc,(&(3)+(CU)*(NC,_)-(CU)#*(CU)&(3)$(f,&1)+(NC,&3)-(NB,_,_)$(n)$(w,(250))
    &(1)$(e,(c,&3))&(2)@(*(DC,&1)$(l,(g,1)))^(Tn)$(l,(y,0))$(w,(5000))
    $(l,(y,1))^(Ss)))
(Tn,(&(2)*(NB,_,_)&(1)$(f,&2)-&1&(2)$(r,&1)&(1)$(f,&2)+(SN)&(2)*(MS,&1)
    -(SN)#*(SN)-(SN)&(2)$(t,(i))!(&1,(c,&2,&3))^(Tn)))
(Ss,(-(TR,_)+(TR,(0))&(1)$(t,(i))-(MC,_)+(MC,(&1))-(MS,_)+(MS,&1)-(NB,_,_)
```

```
    $(n)$(w,(250))&(2)*(NB,_,_)^(In)))
(Ir,(&(2)$(o,sub,(125),&3,(),(),())&(3)$(f,&2)&(2)$(o,les,&3,(1),(),(),())
    -(IU,_)+(IU,&2)&(1)*(IU,((1)))&(3)$(f,&2)))
(Pl,(&(2)$(r,&1)&(1)$(f,&2)&(2)$(f,&1)&(3)$(r,&1)&(1)$(f,&3)&(3)
    $(o,add,&1,&2,(),(),())&(2)$(f,&3)&(3)$(e,(c,&2))-(TR,_)+(TR,&3)))
(Mp,(-(MP,_)+(MP,(&2,&3))&(2)*(MP,_)&(3)$(f,&2)&(2)$(r,&3)&(3)$(f,&2)))
(Ic,(&(1)*(DC,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())&(2)$(f,&1)-(DC,_)
    +(DC,&2)&(1)$(e,(c,&2))&(2)*(NC,&1)$(l,(g,1))&(2)*(AI)-(AI)+(RS)))
(Gt,(&(1)*(MC,_)^(Rs2)&(1)$(f,&2)))
(Si,(+(S)^(Gt)&(2)*(ON,&1)-(S)#&(2)*(MS,_)^(Rs1)$(f,&2)&(2)*(ON,&1)-(S)))
(In,(&(1)*(NB,_,_)&(2)$(f,&1)-&2&(3)$(r,&2)&(2)$(f,&3)-(ON,_)+(ON,&2)^(Si)
    ^(Sp)))
(Sp,(&(2)*(S)-(S)&(1)$(r,&3)&(3)$(f,&1)^(Ir)&(1)*(TR,_)^(Rs2)^(Mp)^(Gt)&(2)
    $(t,(i))-(MSG,_)+(MSG,(p,&2,&1,&3))&(2)*(ON,_)&(3)$(f,&2)&(2)$(r,&3)
    &(3)$(f,&2)&(1)*(MSG,_)^(Rs2)!(&3,&2)#^(In)))
(Ur,(&(1)*(MS,_)^(Rs2)^(Mp)^(Gt)$(u,(p,&1,&3))+(I)&(2)*(D,&1)-(I)#&(2)*(I)
    +(D,&1)^(Ic)#-(NB,_,_)$(n)$(w,(250))^(In)&(2)*(RS)^(Ss)))
(Sr,(^(Pl)+(NP)^(Gt)&(2)$(u,(q,&1))&(1)$(r,&2)&(2)$(f,&1)&(1)
    $(o,les,&2,(0),(),(),())-(LP,_)+(LP,&1)*(LP,((0)))&(1)
    $(o,leq,&2,&3,(),(),())-(LP,_)+(LP,&1)*(LP,((1)))-(NP)#*(NP)^(Ur)))
```

B.2 Smerovač

Počiatočný plán

```
$(l,(g,1))$(w,(250))$(l,(g,0))+(SA)^(nb)^(Mh)
```

Knižnica plánov

```
(Mh,(^(Pm)$(l,(r))$(w,(75))$(l,(r))$(s)^(Mh)))
(Pm,(&(1)?(_)&(2)$(f,&1)-(MT,_)+(MT,&2)&(3)$(r,&1)&(2)$(f,&3)-(MS,_)
    +(MS,&2)&(1)$(r,&3)-(MC,_)+(MC,&1)^(Pa)^(Pm)))
(Rs1,(&(1)$(f,&2)&(2)$(r,&1)&(1)))
(Rs2,(&(2)$(f,&1)&(1)$(r,&2)&(2)$(f,&1)))
(nb,($(l,(r,1))&(1)$(t,(i))+(V,&1)$(n)$(w,(250))^(cn)))
(cn,(^(gn)*(SA)^(gi)^(tm)#*(SA)^(cn)))
(gi,(&(1)$(r,&2)&(2)$(f,&1)))
(gn,(+(bk)&(1)*(NB,_,_)&(2)$(f,&1)-&2-(bk)#&(1)*(bk)^(bk)))
(tm,($(t,(q,&2))-(NB,_,_)$(m,(&2,s))^(nb)))
(bk,(&(3)$(t,(b,f))+(b,&3)&(2)*(b,(1))-(SA)+(RC)$(l,(r,0))#&(2)*(SA)-(PN,_)
    +(PN,&3)$(m,&3)&(1)$(t,(i))+(C)&(2)@(*(PN,&1)-(C)^(nb))&(2)*(C)-(SA)
    $(l,(r,0))))
(Pa,($(l,(y))$(w,(75))$(l,(y))&(3)*(MT,(d))^(Gd)#*(MT,(l))^(Lg)))
(Gd,($(l,(g,0))$(l,(y,0))$(l,(r,0))&(3)$(f,&1)&(1)$(u,(q,&3))&(3)$(f,&1)
    &(1)$(o,leq,&3,(0),(),(),())-(C,_)+(C,&1)&(1)*(C,((1)))&(1)$(t,(i))
    !(&2,(f,&1))#*(C,((0)))&(1)$(t,(i))!(&2,(n,&1,&3))))
```

```
(Lc,(^(Rs2)&(1)$(o,meq,&3,&2,(),(),())-(L,_)+(L,&1)))
(Lg,($(l,(r,0))$(l,(y,0))$(l,(g,0))&(3)$(f,&1)&(1)*(LRT,_)^(Lc)*(L,((1)))
    $(l,(r,1))&(1)*(LYT,_)^(Lc)*(L,((1)))$(l,(y,1))$(l,(r,0))&(1)*(LGT,_)
    ^(Lc)*(L,((1)))$(l,(g,1))$(l,(y,0))))
```

Báza znalostí

```
(LRT,(1))(LYT,(50))(LGT,(100))
```

B.3 Navigátor

Počiatočný plán

```
$(l,(g,1))$(w,(250))$(l,(g,0))^(Mh)
```

Knižnica plánov

```
(Mh,(^(Pm)$(l,(r))$(w,(75))$(l,(r))+(W)*(NN)-(W)^(Pc)#*(W)$(s)#^(Mh)))
(Pc,(&(1)*(CW,_)^(Rs2)&(1)$(o,add,&2,(1),(),(),())&(2)$(f,&1)-(CW,_)
    +(CW,&2)&(1)$(o,meq,&2,(50),(),(),())-(NQ,_)+(NQ,&1)*(NQ,((1)))
    ^(Rd)#$(w,(100))))
(Pm,(&(1)?(_)&(2)$(f,&1)-(MT,_)+(MT,&2)&(3)$(r,&1)&(2)$(f,&3)-(MS,_)
    +(MS,&2)&(1)$(r,&3)-(MC,_)+(MC,&1)^(Pa)^(Pm)*(NN)-(CW,_)+(CW,(0))))
(Rs1,(&(1)$(f,&2)&(2)$(r,&1)&(1)))
(Rs2,(&(2)$(f,&1)&(1)$(r,&2)&(2)$(f,&1)))
(Pa,($(l,(y))$(w,(75))$(l,(y))&(3)*(MT,(d))&(3)$(t,(i))!(&2,(f,&3))#
    *(MT,(t))^(St)#*(MT,(n))^(Nv)#*(MT,(f))^(Fr)))
(St,(+(NN)$(l,(g,0))$(l,(y,1))$(l,(r,0))&(3)$(f,&1)-(DST,_)+(DST,&3)^(Rd)))
(Rd,(*(DST,_)-(NB,_,_)$(n)$(w,(250))+(D)&(1)*(NB,_,_)-(D)-(CD,_)-(CN,_)
    -(DN,_)^(Fcn)&(1)*(DST,_)^(Rs2)&(3)$(e,(c,&2))&(1)*(CN,_)^(Rs2)&(1)
    $(t,(i))!(&2,(d,&1,&3))&(1)#*(D)$(l,(r,1))$(w,(5000))$(l,(r,0))^(Rd)))
(Fcn,(&(2)$(f,&1)-&2&(1)$(r,&2)&(2)$(f,&1)-(TN,_)+(TN,&2)+(DN,&2)&(3)
     $(r,&1)+(NCN)&(1)*(CD,_)-(NCN)^(Rs2)&(1)$(f,&3)&(3)
     $(o,mor,&1,&2,(),(),())-(N,_)+(N,&3)&(3)*(N,((1)))-(CD,_)+(CD,&1)&(1)
     *(TN,_)^(Rs2)-(CN,_)+(CN,&2)#*(NCN)&(1)$(f,&3)+(CD,&1)+(CN,&2)#&(1)
     *(NB,_,_)^(Fcn)))
(Fr,(-(DN,&2)&(1)*(DST,_)^(Rs2)+(Q)&(1)*(DN,_)-(Q)&(3)$(f,&1)-&3&(1)$(r,&3)
    &(3)$(f,&1)&(1)$(t,(i))!(&3,(d,&1,&2))#*(Q)$(l,(r,1))$(w,(5000))
    $(l,(r,0))^(Rd)))
(Nv,(&(3)$(f,&1)&(1)$(e,(c,&3))+(NHU)-(NH,_)&(3)*(DN,&1)-(NHU)+(NH,&1)#
    *(NHU)+(NH,&2)#+(HU)^(Un)))
(Un,(*(NH,_)*(DST,_)-(NB,_,_)$(n)$(w,(250))^(Gn)*(HU)-(HU)-(NH,_)^(Rd)))
(Gn,(^(Pn)&(2)*(NH,&1)^(Hf)))
(Pn,(-(Y)&(3)*(NB,_,_)&(1)$(f,&3)-&1&(3)$(r,&1)&(1)$(f,&3)+(Y)&(2)*(NH,&1)
    -(Y)#*(Y)^(Pn)))
```

```
(Hf,(&(1)*(TH,_)^(Rs2)&(1)$(r,&3)&(3)$(f,&1)&(1)$(o,meq,&3,&2,(),(),())
    -(TM,_)+(TM,&1)&(1)*(TM,((1)))^(Rh)#*(TM,((0)))&(1)*(NH,_)^(Rs2)&(1)
    $(t,(i))!(&2,(l,&1,&3))^(Un)))
(Rh,(+(E)-(HU)&(2)*(NH,_)^(Rs1)$(f,&2)-(NH,_)&(3)*(DST,&1)-(E)-(DST,_)-(NN)
    $(l,(y,0))$(l,(g,1))&(2)$(t,(i))!(&1,(l,&2,(0)))#*(E)-(E)&(2)*(DST,_)
    &(3)$(f,&2)&(2)$(r,&3)&(3)$(f,&2)&(2)$(t,(i))!(&1,(d,&2,&3))))
```

Báza znalostí

```
(TH,(150))
```