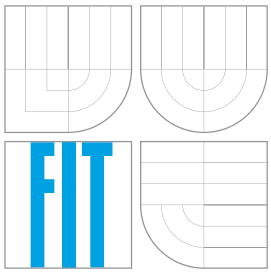
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ALOKACE POZICE SENZOROVÉHO UZLU MOBILNÍM SYSTÉMEM

ALOCATION OF SENSOR NODE POSITION BY A MOBILE SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. JAKUB HYRÁK

VEDOUCÍ PRÁCE
SUPERVISOR
Doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.

BRNO 2016

Zadání diplomové práce/14551/2015/xhyrak00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Hyrák Jakub, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Alokace pozice senzorového uzlu mobilním systémem**
Alocation of Sensor Node Position by a Mobile System
Kategorie: Počítačové sítě

Pokyny:

1. Nastudujte problematiku komunikace v sítích se senzorovými uzly.
2. Navrhněte metody, kterou by bylo možné alokovat umístění senzorového uzlu z mobilního zařízení, jako je například robot.
3. Implementujte příslušné algoritmy jak pro senzorový uzel, tak i pro detekující zařízení.
4. Proveďte experimenty, které ukáží vhodnost navrženého řešení a přesnost alokace uzlů.
5. Diskutujte výsledky a vytyčte další možné směry řešení tohoto problému

Literatura:

- Senzorové sítě
- TinyOS

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zbořil František, doc. Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je nastudovat problematiku bezdrátových senzorových sítí. Popsat prvky senzorové sítě a probrat, jak jednotlivé senzorové uzly mezi sebou komunikují. Zjistit, jak by bylo možné určit pozici senzorového uzlu nově přidaného či mobilního senzorového uzlu do dané senzorové sítě. Z navržených algoritmů určování pozice senzorového uzlu bude pak vybrán jeden, který pak bude v rámci diplomové práce implementován. Jednotlivé algoritmy pro určování pozic senzorových uzlů jsou rozděleny do skupin podle používajících metod. Zvolený algoritmus bude odzkoušen v simulačním prostředí na dáném zařízení.

Abstract

The goal of this diploma work is to study the problems of wireless sensor networks. Describe elements of sensor network and discuss how the individual sensor nodes communicate with each other. Find the way how it would be possible to determine the possition of the new added sensor node or mobile sensor node in the sensor network. The selected one algorithm of determining the possition of the sensor node will be implemented in diploma thesis. Algorithms for determining the positions of sensor nodes are divided into groups by using the methods. Selected algorithm will be tested in simulation on chosen platform.

Klíčová slova

Bezdrátová senzorová síť, senzorový uzel, kotevní uzel, triangulace, RSSI, RF signál, Contiki OS, μIP, Cooja.

Keywords

Wireless sensor network, sensor node, static node, triangulation, RSSI, RF signal, Contiki OS, μIP, Cooja.

Citace

HYRÁK, Jakub. *Alokace pozice senzorového uzlu mobilním systémem.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zbořil František.

Alokace pozice senzorového uzlu mobilním systémem

Prohlášení

Prohlašuji, že jsem tento diplomový projekt vypracoval samostatně za pomoci a vedení doc. Ing. Františka Zbořila Ph.D.

V seznamu použitých zdrojů jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Hyrák
25.5.2016

Poděkování

Tímto bych rád poděkoval vedoucímu mé diplomové práce panu doc. Ing. Františku Zbořilovi Ph.D. za věnovaný čas a poskytnutí odborné pomoci. Dále také všem, kteří se přičinili k dokončení mého diplomového projektu.

Obsah

Kapitola 1

Úvod

Pro určení pozice nového senzorového uzlu v síti či mobilního senzorového uzlu je nutné se nejprve seznámit s jednotlivými prvky a složením senzorové sítě. V dnešní době dochází k prudkému rozvoji a velkým změnám v této oblasti. Od dob, kdy bylo potřeba „mobilní“ senzor převážet nákladním automobilem, jsme se v dnešní době dostali k miniaturnímu zařízení o velikosti 1 cm^3. Takto malé zařízení pak obsahuje několik senzorů, kdy je možné snímat hned několik fyzikálních veličin, jako je například tlak, vlhkost, teplota, hladina, změny v magnetickém poli, vibrace,...

U bezdrátových senzorových sítí (WSN) je pak důležitým aspektem při výběru jednotlivého senzoru především jeho cena a energetická náročnost neboť každý uzel musí mít svoje vlastní napájení. Dále pak dosah komunikace a velikost datového uložiště každého senzoru pro zpracování dat.

Typicky jsou WSN určeny k plošnému monitorování fyzikálních veličin, kdy je nutné znát polohy jednotlivých senzorů. WSN je pak většinou homogenního typu, to znamená, že každý senzor, kterým je WSN tvořena má stejný bezdrátový komunikační dosah, stejné přenosové rychlosti a využívají stejné rozhraní a protokoly. Komunikace mezi uzly je pak směrována k centrálnímu uzlu, kde probíhá shromažďování a vyhodnocení dat.

Je tedy důležité vědět, kam daná data zařadit a znát tedy pozici senzorových uzlů. Někdy se může stát, že pro danou WSN je potřeba více lokalizačních algoritmů, proto jich existuje celá řada. Nastavením parametrů lokalizačních algoritmů je pak možné se dané WSN přizpůsobit. Proto vznikají různá simulační prostředí, kde je pak možné zvolit nejvhodnější lokalizační algoritmus.

Jedním z využívaných prostředí je Contiki OS ze Švédského institutu počítačových věd, v současné době je dostupná verze 3.0. Jedná se o volně dostupný operační systém. Je vysoce přenosný a specializovaný pro efektivní práci s pamětí síťových vestavěných systémů a bezdrátových senzorových sítí. Contiki OS je navržen pro mikroprocesory s malou pamětí. Contiki je multi-platformní systém navržený pro mikrokontrolery jako jsou MSP430 a AVR pro starší domácí počítače. Contiki je napsáno pomocí programovacího jazyka C. Implementaci v prostředí Contiki je poté možné simulovat v přidruženém simulátoru Cooja.

Pomocí simulátoru můžeme ověřit správnost navrženého řešení a provádět určitá měření. To vše před samotnou investicí do senzorových zařízení.

Kapitola 2

Bezdrátové senzorové sítě

Při tvorbě této kapitoly jsem čerpal ze zdrojů uvedených v použité literatuře [10] a [9].

Bezdrátové senzorové sítě (WSN) se obvykle používají k měření jednoho či více fyzikálních jevů v široce distribuovaném prostředí. Ve většině případů se naměřená hodnota zaznamená spolu s časovým měřítkem a i s údajem udávajícím polohu daného senzorového uzlu. Z toho důvodu musí být známy pozice jednotlivých senzorových uzlů.

V současné době jsou WSN založeny na technologii MEMS (micro electro mechanical systems), která se snaží rozměr zařízení miniaturizovat. Vznikají nové typy senzorových uzlů a potřebného hardwaru. Z počátku byly WSN vyvíjeny hlavně ve vojenském průmyslu, v dnešní době už vzniklo mnoho firem zabývajících se touto problematikou.

2.1 Definice pojmu senzor

Senzor je obecně zdroj informací pro daný řídicí systém. V technickém slova smyslu by se za senzor dalo považovat zařízení, které měří či snímá určitou fyzikální nebo biologickou veličinu, kterou převádí nejčastěji na elektrický signál. Signál pak lze dále zpracovávat a dálkově přenášet do určeného řídicího systému.

V elektrotechnickém světě máme více druhů senzorů, které pak dále můžeme dělit například na senzory multifunkční nebo inteligentní. WSN mají oproti tradičním sítím velmi omezené zdroje.

2.2 Komponenty senzorových sítí

WSN jsou obvykle homogenního typu, kdy jsou senzorové uzly složeny ze stejných komponent se stejnými vlastnostmi – například stejný dosah komunikace. Jednotlivé senzorové uzly pak ve velkém množství slouží k plošnému snímání zvolených fyzikálních jevů. Senzorové uzly jsou tak většinou složeny z levných a energeticky nenáročných komponent. Hlavními požadavky na jednotlivé senzorové uzly jsou v dnešní době tedy: rozměr, nízká cena, nízká energetická náročnost, přesnost. Každý senzorový uzel je obvykle vybaven hlavně nízkonákladovým CPU a radiofrekvenčním (RF) vysílačem. Komunikační protokoly musí být taky navrženy s ohledem na nízkou spotřebu energie.

2.2.1 Senzorový uzel

Každá senzorová síť se skládá z jednotlivých senzorových uzlů. Uzly mohou být buď statické (kotva) nebo mobilní. U mobilních uzlů není předem známa jejich pozice v síti. Pomocí komunikace mezi jednotlivými uzly sítě dochází k cyklickému výpočtu její topologie a určení pozic jednotlivých uzlů. Pomocí senzorových uzlů dochází k snímání daných fyzikálních veličin určité oblasti. Snímaná data jsou předzpracována v daném senzorovém uzlu a následně odeslána přes komunikační jednotku k centrálnímu uzlu, který provádí shromáždění a vyhodnocení dat. Základní senzorový uzel se skládá z pěti hlavních komponent: mikrokontroler, komunikační jednotka, senzor, paměť, zdroj napájení (baterie).

2.2.2 Statický uzel (kotva)

Jedná se o senzorový uzel, který má absolutní pozici. Umístění senzorového uzlu je předem známo, nejčastěji pomocí výpočtu externího GPS. Tento typ senzorového uzlu může mít jiné parametry než většina senzorových uzlů WSN. Zdrojem napájení toho uzlu například nemusí být baterie, ale může být napájen z elektrické sítě. Dále pak tento senzorový uzel nemusí sloužit přímo pro snímání hledaných veličin v monitorovaném prostředí, ale jen jako zprostředkovatel komunikace (s výkonnějším RF vysílačem) a sběrač dat okolních senzorových uzlů pro centrální uzel WSN.

2.2.3 Centrální uzel

WSN je potřeba organizovat a řídit, shromažďovat data ze senzorových uzlů, dále je vyhodnocovat a zprostředkovat pro zobrazení v klientském zařízení. Centrální uzel musí být vysoce výkonný, aby nedocházelo ke ztrátám dat nebo aby bylo možné chybná data detekovat či alespoň částečně opravit.

V praxi centrální uzel nemusí zpracovávat data jen jedné WSN, musí tedy umět komunikovat i podle více protokolů pro práci s různorodými WSN. Schopnost komunikace by měla být jak kabelová, tak i bezdrátová. Jelikož v centrálním uzlu dochází ke shromažďování dat a vyhodnocení práce celé WSN, měl by být centrální uzel dostatečně chráněn proti vnějším vlivům jako je elektromagnetické rušení, vysoká či nízká teplota, vlhkost,...

2.3 Komunikace ve WSN

Na komunikaci ve WSN jsou kladeny následující požadavky:

- velký počet uzlů v síti oproti WLAN (wireless local area network) sítím. Tento velký počet uzlů je s výhodou použit při směřování paketů a zaručení odolnosti sítě
- velmi nízká spotřeba a dlouhá výdrž zařízení. Uzly sítě obvykle nemají přístup k napájení a jsou napájeny z baterií nebo z alternativních zdrojů energie
- krátká aktivní doba uzlů v síti
- nízká cena komponent
- malé rozměry zařízení
- odolnost vůči chybám jako je výpadek napájení nebo přerušení komunikace

- schopnost automatické konfigurace (self-organization) – síť je schopná logického uspořádání svých uzlů do definované topologie bez zásahu člověka
- schopnost automatické opravy sítě (self-healing) – síť je schopná detekovat případně i opravit své uzly bez zásahu člověka
- spolehlivost, bezpečný a šifrovaný přenos
- jednoduchá instalace, konfigurace a obsluha sítě
- komunikace v bezlicenčním frekvenčním pásmu

2.3.1 Komunikační standardy

V dnešní době máme spoustu komunikačních standardů. V oblasti WSN je nejvíce využíván standard ZigBee a jeho variace. Při použití je opět kladen důraz na spotřebu energie a potřebný výkon. Proto vznikají stále nové standardy neboť pro každou WSN může být vhodný jiný komunikační standard. Jedním z nejzajímavějších je nadcházející standard ISA SP100.11a, který by měl pokrýt většinu aplikací v oblasti bezdrátových senzorových sítí.

ZigBee

Jednoduchý bezdrátový komunikační standard, označovaný jako IEEE 802.15.4. Umožňuje vzájemnou komunikaci velkého množství zařízení ve vzdálenosti stovek metrů. Standard je citlivý na hluk a rušení, proto je nevhodný do průmyslového prostředí. Tento nedostatek odstraňuje nově vytvořený standard ZigBee PRO (2007), který je specializovaný pro průmyslový trh. Nabízí mimo jiné zlepšení bezpečnosti, možnost změny komunikačního kanálu pro omezení rušení. Jeho výhodou jsou nízké nároky na hardware a nízká spotřeba elektrické energie. Své uplatnění tak našel v oblasti spotřební elektroniky a průmyslu nebo v systémech pro řízení budov v podobě bateriově napájených bezdrátových senzorů.

WirelessHart

Otevřený standard bezdrátové komunikace schválený v roce 2007. Standard vytvořila organizace HCF (HART Communication Foundation). Zaměření standardu je pro procesní měření a monitorovací aplikace mající přísné požadavky na komunikační prodlevy, spolehlivost a bezpečnost. WirelessHART je založen na standardu IEEE 802.15.4, ale komunikuje pouze v pásmu 2,4 GHz. V systému WirelessHART lze zpracovávat pouze protokoly HART.

ISA100.11a

Multifunkční protokol pro sítě používané v průmyslu. Hlavními cílovými oblastmi je monitorování (kromě kritických dat), přenos alarmů, nadřazené řízení a automatizace procesní výroby. Úlohy využívající ISA 100.11a by měly být schopny tolerovat komunikační zpoždění v řádu stovek milisekund. Se zařízením využívajícím standard ISA 100.11a lze komunikovat pomocí jakéhokoliv dalšího zařízení používající protokol IPv6 neboť standard ISA 100.11a využívá pro adresování zařízení uvedený protokol. ISA 100.11a předpokládá průmyslovou komunikační síť splňující všechny současné i budoucí požadavky uživatelů v průmyslu a chrání jejich vynaložené prostředky – je univerzální, jednoduchá a výkonná.

Kapitola 3

Lokalizace

Při tvorbě této kapitoly jsem čerpal ze zdrojů uvedených v použité literatuře [7] a [8], stejně tak použité obrázky jsem přebral ze stejných zdrojů.

Většina senzorových sítí pro své používání potřebuje znát pozice jednotlivých uzlů. Vzhledem k minimalizaci nákladů na bezdrátové senzorové sítě, spotřeby energie, omezení plynoucích z pracovního prostředí, nebývají uzly senzorové sítě vybaveny žádným globálním polohovacím systémem (GPS). Všechny jednotlivé části senzoru jsou napájeny pouze z baterie, proto musí být každá služba energeticky úsporná a lokalizace není výjimkou.

Jedním z hlavních parametrů je přesnost určení umístění senzorového uzlu. Systémy monitorující pohyby objektů v dané oblasti můžou vyžadovat přesnost v řádu centimetrů, naopak u varovných systémů monitorující vznik lesního požáru není problém, když dojde k lokalizaci v řádu desítek metrů.

Dalším důležitým parametrem je rychlost lokalizace. Pro statické, pevné senzorové uzly by nebyl problém, pokud by lokalizace proběhla v řádu minut, neboť je potřeba ji provést pouze jednou, daný senzorový uzel se nehýbe. Na druhé straně mobilní senzorové uzly je potřeba lokalizovat co nejrychleji.

Mnoho lokalizačních metod nepotřebuje žádné další zařízení. Někdy jsou známy polohy několika uzlů, tzv. kotev, jejich souřadnice jsou absolutní.

V mnoha případech je cílem bezdrátových senzorových sítí sledovat nějaký pohybující se objekt nebo zdroj signálu.

Některé přístupy fungují lépe uvnitř, jiné zase venku. Rozdílné chování jsou například v jeskyních, lesích či dokonce pod vodou. Proto neexistuje univerzální řešení pro lokalizaci.

3.1 Fyzikální princip

Lokalizační techniky lze rozdělit do 3 tříd podle způsobu používání pomocí – rádiového signálu, akustického signálu a další méně často využívané způsoby, jako je použití optiky, snímání magnetického pole, využití obrázků z kamer, tlaku,...

3.1.1 Rádiové vlny

Nejpoužívanějším u bezdrátových senzorových sítí při lokalizaci je využití rádiových frekvencí (RF). Každý senzorový uzel má obvykle k dispozici rádiový vysílač. Výhodou je, že se rádiové vlny šíří rychlostí světla. Komunikace tak může probíhat na relativně vysokých frekvencích, často v řádu od stovek MHz do jednotek GHz. Tato nejjednodušší technika založená na přijetí RF je velmi citlivá na vícecestné slábnutí a další zdroje chyb.

3.1.2 Akustické vlny

Mnohé z problémů vyskytujících se při použití RF odstraňuje použití akustických signálů. Rychlost zvuku je o šest řádů nižší než rychlost světla, takže při měření je nutno počítat i s časem šíření signálu mezi párem uzlů. Uzly distribuované senzorové sítě často nesdílejí globální hodiny. Musí tedy docházet k synchronizaci času mezi zdrojem a cílem akustického signálu.

Nejpřesnějšími systémy bývají ultrazvukové, neboť je snadné určit čas přijetí vysokofrekvenčního signálu, díky čemuž pak lze lokalizovat v řádu centimetrů. V praxi je ovšem řada takových systémů omezena na řády metrů, neboť dochází k rychlému oslabení signálu ve vzduchu. Ultrazvukové převodníky jsou všesměrové a poskytují 360° pokrytí. Akustické metody jsou náchylné vůči vnějším přírodním vlivům, jako je hluk či vítr. Proto bývá nejčastější využití uvnitř.

3.1.3 Ostatní

Senzorové sítě kamer poskytují možnost tzv. vlastní lokalizace a orientace (kalibrace). Kamery s překrývajícími se pohledy mohou pozorovat řadu určených bodů a odvodit tak své relativní pozice. Získané souřadnice obsahují váhový koeficient, který musí být určen. Pohybující se objekt je sledován namísto množiny daných bodů.

Méně používanou metodou je tzv. maják. Optický systém používá dva směrové otáčející se světelné signály v kolmých rovinách. Uzly vybavené světelnými čidly pak jsou schopny měřit čas, po který byly světelnému zdroji vystaveny. V závislosti na rychlosti otáčení majáku jsou pak uzly schopny určit svoji vzdálenost.

Mezi dalšími možnostmi měření pak můžou být senzory využívající dvou cívek k vytvoření magnetického pole. Sledovány jsou pak změny intenzity v tomto magnetickém poli.

Pro lokalizaci pod vodou se nejčastěji používají tlakové čidla. Můžeme tak určit odhad hloubky daného čidla. Tlak vzduchu je variabilní, ale může být použit pro odhad převýšení v krátkodobém horizontu.

3.2 Měření vzdálenosti

3.2.1 Čas

Rychlost šíření signálu prostředím je dobře známa, proto je měření doby jeho šíření nejvíce přirozený způsob, jak odhadnout vzdálenost.

Technika, která vyžaduje časovou synchronizaci, ale odpadá nutnost použití inicializační rádiové zprávy, je založena na časovém rozdílu přijetí zprávy (TDoA). Více uzlů detekuje stejnou akustickou vlnu a porovnávají rozdíl mezi TDoA. Rozdíl naměřené TDoA určuje vzdálenost obou uzlů od zdroje akustického signálu, to nám definuje hyperbolu (dvou dimenzionální). Neznámé pozice jsou ohnisky hyperboly, lokalizace je v tomto případě složitá. TDoA je tedy vhodnější pro určení polohy akustických zdrojů při porovnávání TDoA mezi uzly na známých pozicích, než pro vlastní lokalizaci uzlu. Použitím více mikrofonů na více kanálech lze daný způsob využít pro vlastní lokalizaci lépe, ovšem lokalizace zdroje je opět jednoduší.

Kvůli vysoké rychlosti světla je měření ToF rádiového signálu extrémně náročné. Venkovní použití může být relativně vysoce nákladné a energeticky náročné. Přesná synchronizace času v řádu nanosekund je vyloučena. Jediný možný způsob je obousměrný ToF.

Jeden uzel vyšle signál a spustí časovač. Pokud jiný uzel signál detekuje, okamžitě odpovídá prvnímu uzlu, který po obdržení odpovědi následně svůj časovač zastaví. Zpoždění na druhém uzlu mezi příjmem signálu a vyslání odpovědi musí být velmi přesně změřeno a odečteno od výsledného času. Následně se pak pomocí změřeného času spolu se známou rychlostí světla vypočte vzdálenost mezi dvěma uzly. Nejvíce známou technologií využívající obousměrnou ToF je ultra-širokopásmová (UWB). Je založena na posílání širokopásmových impulzů, které jsou dostatečně krátké na to, aby se zabránilo vícecestnému slábnutí signálu, neboť zde není čas pro překrytí mezi ztrátou signálu (LOS) a odrazem. Díky tomuto způsobu tato metoda dobře funguje ve vnitřním prostředí, lze detekovat pozici senzoru v řádu centimetrů. Nevýhodou této technologie je vysoká cena.

Akustické systémy jsou pro měření času šíření signálu vhodné. Rychlost zvuku je dostatečně nízká pro zpracování signálu i pro senzory s omezenými zdroji. Měření doby letu (ToF) signálu je nejvíce rozšířená technika. Senzor vysílá inicializační rádiovou zprávu označující začátek měření a následně charakteristický akustický signál. Pokud sousední uzly zachytí inicializační zprávu, spustí časovač a pomocí mikrofonu začnou odebírat vzorky akustického signálu. Až dojde k zachycení onoho následně vyslaného charakteristického zvukového signálu, časovač je zastaven. Uplynulý čas společně se známou rychlostí zvuku pak určí vzdálenost mezi dvěma uzly. Výrazným zdrojem chyb je však teplotní závislost rychlosti zvuku. Tento problém lze kompenzovat měření teploty a nastavení rychlosti pro daný výpočet. Dále je zařízení velmi citlivé na hluk. Proto je měření prováděno většinou vícekrát a jednotlivá měření jsou průměrována. Výhodou ToF je, že senzorová síť nepotřebuje explicitní časovou synchronizaci.

3.2.2 Výkon

Většina rádio vysílacích čipů může poskytovat odhad přijaté RF energie v daném pásmu díky indikátoru přijaté síly signálu (RSSI). Díky tomu, že známe vysílací výkon a model šíření, můžeme RSSI použít pro odhad vzdálenosti. Tato technika je jednoduchá a levná, bohužel ale nepřesná. Průměrná chyba venkovních zařízení je 10 až 20 % a ve vnitřních prostorách ještě horší. Šíření RF signálu je vysoce závislé na prostředí a je dynamické. Skutečnost je většinou jiná než to, co předpovídá model sítě. Většina systémů používá skupinu pevných základním stanic na předem známém umístění, které vytvoří RSSI mapu hodnot pro každou takovou stanici pro celou oblast v době rozestavení systému. Lokalizace je poté prováděna měřením souboru RSSI hodnot s účelem najít největší shodu v RSSI mapě. Tento způsob poskytuje mnohem větší přesnost, ale s vysokou počáteční cenou. Pro praktické použití je požadováno velké množství základní stanic. Většina prostředí se dynamicky mění, což má vliv na dosažitelnou přesnost. Tato technika je komerčně používána. Je obtížné určit přesnost takové sytému v dynamickém prostředí. Reálná chyba tohoto systému se pohybuje pod hodnotou 10m.

3.2.3 Fáze

Měření fáze stacionárního periodického signálu mezi vysílačem a přijímačem poskytuje informace o jejich vzájemné vzdálenosti. V této metodě se ovšem vyskytuje několik problémů. V případě, že je vlnová délka kratší než měřená vzdálenost, pak je fáze poskytnuta pouze ve vzdálenost modulo vlnová délka. Z tohoto důvodu je třeba vícefrekvenčních měření a třeba vyřešit soubor diophantine rovnic (Polynomická rovnice, obvykle ve dvou nebo více neznámých tak, že pouze celočíselné řešení, jsou hledány nebo studovány. Diophantine problémy mají méně rovnic než neznámých proměnných a zahrnují zjištěná celá čísla, které

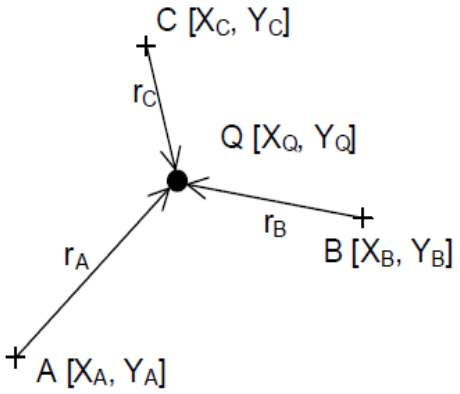
fungují správně pro všechny rovnice.). Bez přesné časové synchronizace, neznámé vysílací fáze a je třeba toto řešení kompenzovat. Variantou použití tohoto řešení je použití více antén na stejném uzlu a měření rozdílu mezi přijatými fázemi. Tímto můžeme odhadnout zdroj RF signálu. Přístupy založené na fázi jsou ovšem na vícezdrojovost citlivé. Přidáním každého dalšího zdroje vzniká fázový posun, kde můžou vznikat chyby. Toto řešení je obzvláště problematické, pokud jsou uzly umístěny na podlaze. Odraz signálu od podlahy v malém úhlu může výrazně zeslabit signál.

3.3 Metody lokalizace

3.3.1 Triangulační algoritmus

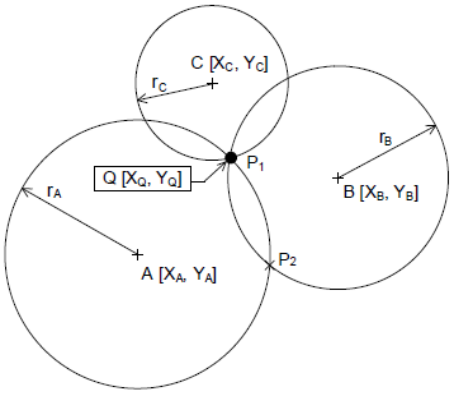
Jedním ze způsobů k určení polohy nového bodu v prostoru je využití tří okolních referenčních bodů, jejichž poloha je v daný moment známá. Algoritmus vychází ze znalostí aplikované geometrie, kdy díky znalosti polohy referenčních bodů a vzdálenosti k nim můžeme určit pozici nově přidaného bodu. Při výpočtu je opakovaně využita Pythagorova věta.

Princip algoritmu je následující: nový senzorový uzel, který zatím nezná svoji polohu (Q), získá ze svého okolí vzdálenost a polohy třech referenčních uzlů (A, B, C), které svoji polohu znají. Následně po získání těchto hodnot začne samotná lokalizace.



Obrázek 3.1: Možné rozmístění uzlů

Nejdříve vybereme dva ze tří referenčních uzlů. Kolem každého z uzlů vytvoříme kružnici, jejíž poloměr je dán hodnotou vzdálenosti mezi daným referenčním uzlem a naším novým uzlem. Pomocí matematického výpočtu získáme polohy dvou průsečíků kružnic (P_1, P_2). Jeden z průsečíků určuje pravděpodobnou polohu našeho nového uzlu. K výběru správného průsečíku využijeme třetí referenční uzel, kdy dojde k porovnání vzdálenosti mezi třetím referenčním uzlem a naším novým uzlem s porovnáním vzdálenosti obou průsečíků od třetího referenčního bodu.



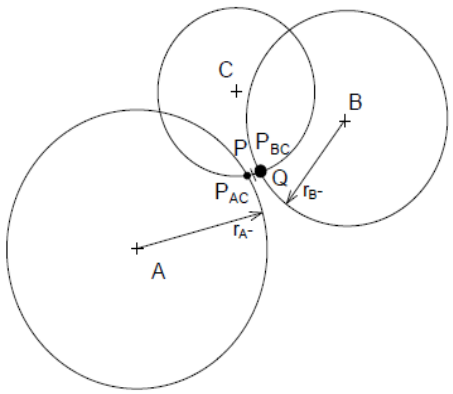
Obrázek 3.2: Lokalizace nového uzlu

Kritické části

Funkčnost triangulačního algoritmu je závislá na vzájemném uspořádání všech tří referenčních uzlů a nové uzlu, který provádí svou lokalizaci. Mohou nastat tři situace, které vedou od nepřesností až k nefunkčnosti algoritmu:

- nový uzel se nachází na přímce mezi dvěma referenčními uzly
- dva referenční uzly mají stejnou y souřadnici
- všechny referenční uzly se nachází na přímce

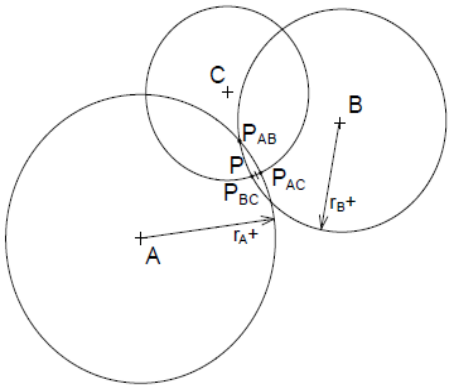
Nový uzel se nachází na přímce mezi dvěma referenčními uzly
Pokud dojde k chybnému naměření vzdálenosti, můžou nastat dvě chybné situace. Je-li naměřená vzdálenost kratší než skutečná, i při malé chybě dojde k tomu, že se kružnice neprotnou. Nevznikne tedy žádný průsečík a nový uzel tak nemůže být správně lokalizován. Této situaci se dá předejít díky tzv. mechanismu rotace referenčních uzlů. Na obrázku níže je možné vidět, že se kružnice vzájemně neprotnuly. Je nutné provádět detekci této situace.



Obrázek 3.3: Naměření krátké vzdálenosti

Obdobná situace nastane, pokud je chybně naměřená vzdálenost větší než skutečná. Vzniknou tak dva průsečíky, které jsou stejně vzdálené od pozice nového uzlu. Opět vzniká

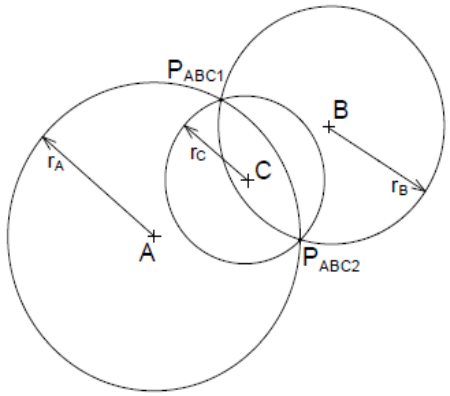
chyba lokalizace nového uzlu. V poslední části algoritmu totiž není jasné, které z chybně vzniklých průsečíků zvolit. Při správně naměřené vzdálenosti by měl vzniknout jeden tečný bod.



Obrázek 3.4: Naměření větší vzdálenosti

Dva referenční uzly mají stejnou y souřadnici
Pokud nastane situace, kdy budou mít dva referenční uzly stejné y souřadnice, může dojít k nalezení pouze jednoho ze dvou průsečíků. Druhý průsečík ovšem mohl být tím správným. Dochází tak k chybné lokalizaci nového uzlu. Uvedenou situaci lze detekovat kontrolou shodnosti souřadnic. Pokud máme k dispozici více referenčních uzlů, řešením je vybrat takové uzly, u kterých k tomuto problému nedochází. Pokud nejsou k dispozici další referenční uzly, můžeme použít mechanismus rotace referenčních uzlů. Danou dvojici ovšem musíme z možných kombinací vypustit. Matematicky je tento problém možné obejít záměnou x a y osy.

Všechny referenční uzly se nachází na přímce
Jako je patrné z uvedeného obrázku, oba dva průsečíky jsou stejně vzdálené od třetího rozhodujícího referenčního uzlu.



Obrázek 3.5: Referenční uzly na přímce

Tento jev se nazývá zrcadlení. Řešením této situace nemůže být mechanismus rotace referenčních uzlů, neboť se chyba může objevit ve dvou ze tří možných kombinací. V případě chybně naměřené vzdálenosti, může dojít k označení nesprávného průsečíku.

Zvyšování přesnosti a stability mechanizmem rotace referenčních uzlů

Dva referenční uzly jsou použity k nalezení dvou průsečíků. Pro výběr pravděpodobnějšího z těchto dvou průsečíků je využit třetí referenční uzel. Existují tedy právě tři možnosti výběru dvojic referenčních uzlů, ze kterých budou počítány průsečíky. Pokud se tedy nový uzel nachází na přímce mezi dvěma referenčníma uzly, máme v jedné ze tří možností větší odchylku. Tohoto je možné využít ke zvýšení přesnosti. Vypočítáme průsečíky pomocí všech tří možností a vzájemně porovnáme. Výsledný průsečík se bude nacházet mezi nejbližšími průsečíky. Tímto způsobem můžeme eliminovat chybu, která by se objevila při výběru nejméně vhodné možnosti referenčních uzlů.

3.3.2 Iterativní algoritmus

Senzorový uzel je schopný určit svoji polohu, pokud se nachází v dosahu signálu tří kotevních uzlů. Každý kotevní uzel totiž zná svoji polohu, kterou předává ostatním uzlům. Pokud se senzorový uzel nachází v dosahu kotevního uzlu je schopen získat, vysláním signálu, svoji vzdálenost od kotevního uzlu. Vytvořením tří kružnic z pozic kotevních uzlů o poloměru daném vzdáleností daného senzorového uzlu dostaneme tedy průsečík a polohu daného senzorového uzlu. Tímto způsobem jsou senzorového uzly v dosahu tří kotevních uzlů schopny určit svoji polohu. Ovšem kotevních uzlů může být pouze nepatrná část oproti počtu senzorových uzlů. Proto senzorové uzly, kterým se v dané iteraci podaří získat svoji polohu, se v další iteraci stávají referenčními uzly. Tímto iteračním způsobem, pokud je senzorová síť dostatečně hustá, jsme schopni rozšířit informace o lokalizaci senzorů po celé ploše sítě.

3.3.3 Spojový algoritmus

Nejméně tři kotevní uzly mohou být umístěny kdekoliv v senzorové síti. Na jejich rozmístění nezávisí funkčnost, ale chybovost. Každý kotevní uzel vysílá svoji polohu. Senzorový uzel informaci zpracuje a uloží si ji do paměti. Do svého okolí pak posílá pozici kotevního uzlu a k této informaci přidá ještě svoji vzdálenost od kotevního uzlu, kterou získá z výkonové úrovně signálu. Další senzorové uzly si pak přičítají své vzdálenosti. Pokud se k danému senzorovému uzlu dostane informace o vzdálenosti stejného kotevního uzlu vícekrát, uloží si vždy hodnotu s nejmenší vzdáleností. Informace o polohách kotevních uzlů se tak rozšíří po celé síti. Pokud senzorový uzel získá informaci o poloze a vzdálenosti tří kotevních uzlů, je schopen pomocí triangulačního algoritmu vypočítat svoji polohu.

Kapitola 4

Operační systém Contiki

Při tvorbě této kapitoly jsem čerpal ze zdrojů uvedených v použité literatuře [1] a [2].

Contiki je open source operační systém. Je vysoce přenosný a specializovaný pro efektivní práci s pamětí síťových vestavěných systémů a bezdrátových senzorových sítí. Contiki je navržen pro mikroprocesory s malou pamětí.

Contiki je vyvíjeno skupinou vývojářů jak z průmyslu, tak i z akademických sfér. Hlavou vývojářů je Adam Dunkels ze Švédského institutu počítačových věd (Swedish Institute of Computer Science). V současné době se tým skládá z šestnácti vývojářů z SICS, SAP AG, Cisco, Atmel, NewAE a TU Munich.

Contiki obsahuje dva komunikační zásobníky μIP a Rime. Komunikace mezi jednotlivými uzly senzorové sítě zajišťuje protokol IP. Je podporována verze IPv4 i IPv6.

Contiki je multi-platformní systém navržený pro mikrokontrolery jako jsou MSP430 a AVR pro starší domácí počítače. Velikost kódu se pak pohybuje v rozmezí kilobajtů a paměť je pak vytvořena tak, aby se pohybovala v rozmezí desítek bajtů.

Contiki je napsáno pomocí programovacího jazyka C. Je volně k dispozici jako open source pod licencí BSD.

Mnoho klíčových mechanizmů i myšlenek pocházejících z Contiki je využito v průmyslu. μIP vestavěné v IP zásobníku původně navrženo v roce 2001 dodnes využívají stovky společností například pro nákladní lodě, satelity nebo zařízení k těžbě ropy.

Contiki a μIP používá pro skenování sítě populární nástroj nmap (hledání hostitelských počítačů a služeb k vytvoření „mapy“ sítě).

4.1 μIP

Ke komunikaci pomocí protokolů TCP/IP na 8bitových mikrokontrolerech slouží protokol μIP TCP/IP. Pomocí μIP pak podobná zařízení dokážou spolu komunikovat i přes to, že je zásobním malý a jednoduchý. Velikost kódu je v řádu několika kilobajtů paměti RAM a může být nakonfigurován tak, aby byl nižší než stovky bytů.

Díky úspěchu a rozšíření internetu se protokoly TCP/IP staly celosvětovým standardem pro komunikaci. TCP/IP je základní protokol používaný pro převod webových stránek, e-mailové přenosy, přenosy souborů a peer-to-peer spojení přes internet. U vestavěných systémů, aby byly schopny provozovat nativní TCP/IP, umožňuje propojení systému přímo k intranetu nebo dokonce globálnímu internetu.

Běžná TCP/IP implementace vyžaduje příliš mnoho zdrojů z hlediska velikosti kódu a paměti pro použití v malých 8 nebo 16bitových systémech.

Implementace μIP je navržena tak, aby byla použita pouze minimální sada funkcí, potřebná k úplné TCP/IP komunikaci. Proto se využívá jen jednoduchého síťového rozhraní s protokoly IP, ICMP, UDP a TCP. μIP protokol je napsán v programovacím jazyce C.

Jiné TCP/IP implementace pro malé systémy pak předpokládají, že vložený snímač vždy komunikuje v plném rozsahu TCP/IP běžící na pracovní stanici. Je možné odstranit určité mechanismy TCP/IP, které jsou v daných situacích velmi málo používané. Mnohé z těchto mechanismů jsou důležité, ovšem probíhá-li komunikace vestavěného zařízení s jiným podobně omezeným zařízením, nemusí být potřebné, například systém distribuované služby peer-to-peer a protokoly. μIP je navržena v souladu s RFC, aby se zařízení jevilo jako běžný uzel sítě. Implementace μIP TCP/IP není přizpůsobena pro obecné použití u každé aplikace.

4.1.1 TCP/IP komunikace

TCP/IP se skládá z mnoha protokolů, od protokolů nízké úrovně jako je ARP k překladu IP adresy na MAC adresu, po protokoly na aplikační úrovni jako je SMTP, který se používá k přenosu e-mailů. μIP se nejvíce zabývá protokoly TCP a IP vyšších vrstev označovaných jako „aplikační“. Protokoly nižších vrstev jsou často implementovány v hardwaru nebo firmwaru a jsou označovány jako „síťové zařízení“, které jsou řízeny ovladačem síťového zařízení.

TCP/IP poskytuje spolehlivý přenos dat pomocí protokolů vrchních vrstev. Data jsou rozdělena do paketů. Pokud dochází k přenosu mezi více sítěmi, kde je rozdílná velikost přenášených paketů, jsou pakety fragmentovány směrovačem, konečné zařízení pak musí fragmenty opět složit.

Formální požadavky na protokoly TCP/IP jsou specifikovány v několika RFC dokumentech zveřejněných Internet Engineering Task Force, IETF. Každý z protokol je definován v jednom z několika RFC dokumentů, RFC1122 shromažďuje všechny požadavky a aktualizuje ostatní dokumenty RFC.

V μIP jsou implementovány všechny požadavky RFC, které ovlivňují host-to-host komunikaci. Nicméně s cílem snížit velikost kódu, jsou odstraněny určité mechanismy v rozhraní mezi aplikací a zásobníkem, jako je měkký mechanismus podávání zpráv o chybách a dynamicky konfigurovatelné typ-of-service bity pro TCP spojení. Existuje však velmi málo aplikací, které tyto funkce využívají a mohou tak být odstraněny bez ztráty na obecnosti.

4.1.2 Hlavní řídící smyčka

μIP zásobník lze spustit jako jednotnou operaci či jako proces v rámci multitaskingu. V obou případech hlavní řídící smyčka kontroluje dvě věci:

- zda paket dorazil,
- zda nedošlo k vypršení pravidelného časového limitu.

Pokud paket dorazil, funkce vstupu *uip_input*(), je volána hlavní řídící smyčkou. Funkce vstupu nesmí blokovat, vrátí výstup. Pokud dojde k výstupu funkce, zásobník nebo aplikace, kterým je zpracovaný paket určen, můžou odpovědět pomocí více paketů. Pokud se tak stane, síťový ovladač zařízení odešle tyto pakety.

U TCP mechanizmů používajících časovače se používají pravidelní časové limity, jako je zpožděné potvrzení, znovu odeslání paketu nebo odhad round-trip času. Pokud hlavní

smyčka zjistí, že došlo k vypršení časovače, je volána funkce časovače *uip_periodic*(). TCP/IP zásobník tedy může provést znovu odeslání, pokud dojde k této události.

4.1.3 Architektura a specifické funkce

U μIP je vyžadováno zvláště specifikovat několik funkcí v závislosti na použité architektuře, na které je μIP spuštěno. Tyto funkce by měly být ručně upraveny pro konkrétní architekturu, ale v obecné implementaci v C jsou uvedeny jako součást distribuce μIP.

Výpočet kontrolní sumy
Protokoly TCP/IP provádí kontrolní součet, který zahrnuje hlavičku a datovou část paketu. Jelikož se kontrolní součet provádí pro všechny bajty v každém odeslaném i přijatém paketu, je důležité, aby funkce pro výpočet byla efektivní. To znamená, že se často funkce opět vylaďuje pro konkrétní architekturu, kde je μIP spuštěno.

μIP obecně zahrnuje funkce pro kontrolní součet, ale také je často ponechává otevřeny. Pro konkrétní implementaci závislé na zvolené architektuře slouží dvě funkce $uip_ipchksum()$ a *uip_tcpchksum*(). Výpočty kontrolního součtu těchto funkcí bývají spíše napsány ve vysoce optimalizovaném assembleru než v generickém C kódu.

32 bitová architektura
U TCP protokolu je použita 32bitová architektura. Oproti k tomu 32bitová aritmetika není nativně k dispozici na mnoha platformách, pro které je μIP je určeno. Pokud je to nutné, μIP podporuje obecné 32bitové dodatky, které jsou pak ručně specifikovány ve funkci *uip_add*32().

4.1.4 Správa paměti

Většinou je v zařízení, ve kterých je použito μIP, dostupných pouze několik kilobajtů RAM. Paměť je tedy velmi vzácným zdrojem. Mechanizmy tradičně používané v TCP/IP tedy nelze přímo použít.

μIP nevyužívá explicitní dynamické přidělování paměti. Používá jednoduchou globální vyrovnávací paměť pro práci s pakety a tabulku s indikátorem stavu spojení. Globální vyrovnávací paměť je dostatečně velká pro jeden paket maximální velikosti. Když ze sítě přijde paket, ovladač zařízení jej umístí do globálním vyrovnávací paměti a volá TCP/IP. Pokud paket obsahuje data, TCP/IP zásobník oznámí tuto skutečnost odpovídající aplikaci. Jelikož můžou být data v globální vyrovnávací paměti přepsána dalším příchozím paketem, musí daná aplikace okamžitě jednat nebo data přemístit do sekundární paměti k pozdějšímu zpracování. Data paketu ve vyrovnávací paměti ovšem nebudou přepsány daty nového paketu, pokud právě dochází ke zpracování danou aplikací. Pakety, které dorazí, pokud daná aplikace právě zpracovává data, jsou zařazeny do fronty a to buď pomocí síťového zařízení, nebo pomocí ovladače daného zařízení. Většina jednočipových Ethernet zařízení má vyrovnávací paměti o velikosti maximálně čtyř Ethernetových rámců. Zařízení, která jsou zpracovány procesorem, jako jsou například RS-232 porty, můžou kopírovat příchozí bajty do samostatné vyrovnávací paměti i při zpracovávání aplikací. Pokud jsou vyrovnávací paměti plné, dochází k zahození příchozího paketu. V případě, že jde o paralelní připojení více zařízení, dochází tak ke snížení výkonu.

V μIP je použita stejná globální vyrovnávací paměť jak pro příchozí pakety, tak i pro TCP/IP hlavičky odchozích dat. Pokud aplikace odesílá data dynamicky, může vy-

užít části globální vyrovnávací paměti, které nejsou využity pro hlavičky, jako dočasného úložiště paměti. K odeslání dat předává aplikace ukazatel na zásobník, na data i délku dat. Jakmile jsou TCP/IP hlavičky zapsány do globální vyrovnávací paměti, ovladač zařízení je posílá spolu s aplikačními daty do sítě. Data nejsou ve frontě pro opakované přenosy. Pokud je nutné data reprodukovat, musí se o to postarat aplikace.

Využití celkové paměti v μIP opět závisí na konkrétní aplikaci a na zařízení, na kterém je spuštěna. Konfigurace paměti pak určuje, jaké maximální množství současných připojení a intenzitu je dané zařízení schopno zvládnout. Zařízení, které odesílá velké e-maily a ve stejnou dobu spouští webový server s vysoce dynamickými webovými stránkami pro více uživatelů, bude vyžadovat více RAM než jednoduchý Telnet server. Je možné spustit μIP s méně než 200 bajty paměti RAM, ale takové konfigurace budou poskytovat velmi nízkou propustnost a umožní pouze malý počet současných připojení.

4.1.5 Aplikační rozhraní programu (API)

API definuje způsob, jakým program aplikace spolupracuje s TCP/IP. Nejčastěji používaným API pro TCP/IP je BSD rozhraní API, který je použito ve většině unixových systémech a silně ovlivnilo Microsoft Windows WinSock API. Protože rozhraní API používá stop-and-wait sémantiku, vyžaduje podporu od podkladového multitasking operačního systému. BSD rozhraní tedy pro naše účely není vhodné, neboť režie pro správu úloh, přepnutí kontextu, přidělení místa na zásobníku by byla v zamýšlených zařízeních používajících μIP příliš vysoká.

μIP poskytuje dvě programové API rozhraní:

- protosoket – BSD rozhraní pro API bez režie, plný multi-threading
- „raw“ API založené na událostech, které je nízko úrovňové oproti protosoket, a používá méně paměti.

Prvotní μIP API (RAW)
μIP API používá událostmi řízené rozhraní, kdy je aplikace volána k reakci na určitou událost. Aplikace běžící v horní vrstvě μIP je implementována jako funkce v C, která je volána μIP na reakci na událost. μIP volá aplikaci při příjmu dat, když jsou data úspěšně doručena na druhý konec spojení, když bylo navázáno nové spojení nebo když musí být data znovu odeslána. Aplikace je také v pravidelných intervalech žádá o nová data. Aplikační program má pouze jednu funkci zpětného volání. Je na aplikaci, aby se vypořádala s mapováním různých síťových služeb na různé porty a připojení. Až je přijat paket na TCP/IP zásobníku, může aplikace provádět svoji činnost, je důležitá nízká odezva.

μIP se liší od TCP/IP tím, že potřebuje pomoc od aplikace při znovu odeslání dat. TCP/IP drží data ve vyrovnávací paměti, dokud nedojde k úspěšnému doručení dat druhé straně. Pokud je tedy potřeba data znovu odeslat, zásobník je odešle bez uvědomění aplikace. Data tedy musí být ve vyrovnávací paměti, dokud nepřijde potvrzení úspěšného doručení, aby v opačném případě bylo možné data opakovaně odeslat.

Aby bylo využití paměti u μIP sníženo, využívá se toho, že je aplikace schopná odeslaná data znovu vytvořit a účastní se tak znovu odeslání dat. Ovladač zařízení u μIP neukládá stopu obsahu paketu poté, co byl odeslán. μIP vyžaduje, aby se aplikace podílela na znovu zaslání dat. Pokud dojde k tomu, že by data měla být znovu odeslána, je aplikace volána s příznakem indikujícím, že je zapotřebí opakovaného přenosu dat. Aplikace zkontroluje příznak indikující opakování přenosu a vystaví stejná data, která byla dříve

odeslána. Z pohledu aplikace se pak provádění opětovného odeslání dat nijak neliší od původního odeslání. V aplikaci je pak možné využití stejného kódu jak pro odeslání nových, tak i pro opětovné odeslání dat. Zodpovědnost, jestli mají být data znovu odeslána, má tedy zásobník a nezvyšuje se tím tedy složitost aplikace.

Aplikační události
Aplikace musí být realizována jako funkce v C, *uip_app_call*(), pokud dojde k události, je volána pomocí μIP. Každá událost má odpovídající testovací funkci, která se používá pro rozlišení mezi různými událostmi. Funkce jsou realizovány jako C makra, která jsou hodnocena buď nulovou, nebo nenulovou hodnotou. Některé události se mohou vyskytnout souběžně.

Ukazatel na připojení
Je-li aplikace volána, globální proměnná *uip_conn* protokolu μIP je nastavena, aby ukazovala na *uip_conn* strukturu pro připojení, které je v současné době zpracováváno a nazýváno jako „aktuální připojení“. Položky *uip_conn* struktury pro aktuální spojení mohou být použity jako, například pro rozlišení mezi různými službami nebo jako kontrola, která IP adresa připojení je připojena. Typické využití by bylo podívat se na $uip\_conn > lport$ (lokální TCP port), aby podle něj bylo možné rozhodnout jakou službu připojení poskytnout. Například aplikace se může chovat jako HTTP server, pokud je hodnota $uip\_conn > lport$ rovna 80, a jako TELNET server, pokud je hodnota 23.

Přijímání dat
Pokud μIP test funkce *uip_newdata*() je nenulový, vzdálený hostitel poslal nová data. *uip_appdata* ukazatel je ukazatelem na tyto aktuální data. Velikost dat lze zjisti pomocí funkce *uip_datalen*(). Data nejsou uložena ve vyrovnávací paměti μIP, ale budou přepsány po návartu z aplikační funkce. Aplikace musí příchozí data buď přímo zpracovat, nebo je zkopírovat do vyrovnávací paměti pro pozdější zpracování.

Odesílání dat
Při odesílání dat dochází u μIP k úpravě délky dat vystavených aplikací podle dostupné vyrovnávací paměti a podle velikosti aktuálního TCP okénka dané přijímačem. Velikost vyrovnávací paměti je dána její konfigurací. Je tedy možné, že všechna data odeslaná z aplikace nedorazí k přijímači, aplikace může pro kontrolu použít funkci *uip_mss*(), aby zjistila, kolik dat bylo skutečně odesláno ze zásobníku.

Aplikace odešle data pomocí μIP funkce *uip_send*(). Funkce *uip_send*() má dva argumenty: ukazatel na data k odeslání a jejich velikost.

V jednom spojení není možné volat funkci *uip_send*() vícekrát než jednou. V případě vícenásobného volání budou odeslána data pouze z posledního volání.

Opakované přenosy
Při odeslání dat je využit časovač. Pokud dosáhne nuly, odeslání dat by mělo být opakováno. Protože μIP nemá uložen obsah odeslaného paketu ovladačem zařízení, μIP vyžaduje, aby si aplikace převzala režii nad znovu odesláním dat. Pokud tedy μIP rozhodne, že má dojít k znovu zaslání, je volána funkce *uip_rexmit*() s nastaveným příznakem. Aplikace kontroluje příznak, z hlediska aplikace se provádění opakovaného přenosu neliší od toho, jak byla data původně odeslána. Pro znovu odeslání dat může pak být použito stejného

aplikačního kódu jako u prvotního odeslání. Zodpovědnost za odeslání dat má zásobník a nezvyšuje se tak složitost aplikace.

Uzavírání spojení
Uzavření aktuálního spojení je pomocí funkce *uip_close*(). Spojení musí být uzavřeno čistě. Pokud došlo k závažné chybě, aplikace přeruší spojení voláním funkce *uip_abort*().

V případě, že bylo spojení ukončeno z druhé strany, test funkce *uip_closed*() je roven hodnotě *true*. Musí aplikace provést nezbytný úklid paměti.

Chybová hlášení
Existují dvě fatální chyby, které mohou nastat: spojení bylo přerušeno vzdáleným hostitelem nebo příliš mnohokrát dochází k znovu odesílání dat či přerušení. Pro test chyb je možné využít dvě funkce *uip_aborted*() nebo *uip_timedout*().

Polling
μIP se periodicky dotazuje aplikace, není-li připojení nečinné. Aplikace používá testovací funkci *uip_poll*().

Událost dotazování má dva účely. Prvním z nich je, aby aplikace věděla, je-li spojení nečinné a umožnit tak aplikaci uzavřít spojení, pokud trvá nečinnost dlouho. Druhým účelem je, aby aplikace odesílat nová data. Aplikace může odesílat data pouze po volání z μIP, a proto je toto dotazování jediný způsob, jak posílat data na jinak nečinné spojení.

Naslouchání na portech
μIP udržuje seznam aktivních portů TCP. Nový port je vždy pro poslech otevřen funkcí *uip_listen*(). Pokud na naslouchaný port přijde žádost o vytvoření spojení, μIP vytvoří nové spojení a volá funkci aplikace. Test funkce *uip_connect*() je rovna *true*, pokud byla volána aplikace a bylo vytvořeno nové spojení.

Aplikace může kontrolovat lport položku v *uip_conn* struktuře ke kontrole, na který port bylo nové spojení vytvořeno.

Nové spojení
Nová spojení lze vytvořit μIP pomocí funkce *uip_connect*(). Tato funkce přidělí nové spojení a nastaví příznak připojovacího stavu, který se otevře TCP spojení na zadanou IP adresu a port. Funkce *uip_connect*() vrací ukazatel na strukturu *uip_conn* pro nové připojení. Pokud nejsou k dispozici žádné volné sloty spojení, funkce vrací hodnotu *null*.

4.2 Rime

Rime komunikační zásobník poskytuje sadu lehkých komunikačních primitiv v rozsahu nejusilovnějšího anonymního lokálního broadcast vysílání na spolehlivé zaplavení sítě.

Protokoly v Rime zásobníku jsou uspořádány do vrstev způsobem, kdy více komplexní protokoly jsou realizovány použitím méně komplexních protokolů.

Byly vybrány komunikační primitiva v Rime zásobníku založené na tom, co typické protokoly senzorových sítí používají. Aplikace nebo protokoly běžící na vrcholu Rime zásobníku jsou připojeny na některou vrstvu zásobníku a používají některá komunikační primitiva.

Rime zásobník podporuje jednostupňové a vícestupňové komunikační primitiva. U vícestupňových primitiv není určeno, jaký způsobem jsou pakety směrovány v síti. Naproti

tomu je volána, po odeslání paketu skrz síť, aplikace nebo horní vrstva protokolu, a v každém uzlu je zvolen další stupeň. Takto je možné realizovat libovolné množství směrovacích protokolů v horní části vícestupňových primitiv.

Protokoly nebo aplikace běžící na vrcholu Rime zásobníku mohou implementovat přídavné protokoly, které nejsou v Rime zásobníku. Pokud protokol nebo aplikace běží na vrcholu Rime zásobníku může potřebovat komunikační primitiva, která nejsou v současné době v Rime zásobníku. Aplikace nebo protokol tak může vytvořit jiné komunikační primitiva a jejich realizaci přímo na vrcholu zásobníku.

Kapitola 5

Implementace

5.1 Požadavky

Podle zadání mé diplomové práce jsem měl implementovat potřebné algoritmy pro senzorové uzly i mobilní zařízení, které určí pozici daného mobilního zařízení. Nabízí se tak dvojí řešení. V prvním řešení hlavní práci obstarává mobilní uzel, který řídí komunikaci s uzly senzorové sítě, sbírá potřebné údaje a provádí výpočet vlastní pozice. V druhém řešení se mobilní uzel pouze pohybuje v prostředí senzorové sítě a periodicky vysílá signál. Uzly senzorové sítě si pak distribuují informace a daný uzel senzorové sítě provádí výpočet pozice mobilního zlu.

Po dohodě a doporučení vedoucím mé diplomové práce jsem se rozhodl implementovat zadání v prostředí Contiki-3.0 OS. Testování a zpracování výsledků pak ověřit v simulátoru Cooja.

Pro zpracován je tedy nutné navrhnout způsob komunikace mezi senzorovými uzly, určení vzdálenosti mezi nimi a vybrat algoritmus pro určení pozice mobilního senzorového uzlu.

Pro řešení problému, který zpracovávám ve své diplomové práci, jsem měl primárně využít senzorového uzlu MicaZ. Jedná se o senzorový uzel, který je dostupný na fakultě informačních technologií VUT v Brně. Bylo by tedy případně možné mé řešení vyzkoušet i v reálném prostředí.

5.2 Návrh řešení

Senzorové uzly jsem rozdělil do dvou typů. Každý typ má v senzorové síti své specifické chování.

Kotevní uzel - senzorový uzel, který zná svoji pozici. Přijímá žádosti jiných senzorových uzlů a zprostředkovává své údaje. Jedná se o statický, nepohybující se uzel.

Mobilní uzel – uzel, který se může v dané senzorové síti libovolně pohybovat. Jeho pozice se tedy může měnit. Žádá tedy kotevní uzly ze senzorové sítě, aby mu v daný moment poskytly údaje o svých pozicích.

5.2.1 Mobilní uzel vyžaduje komunikaci a provádí výpočet

Komunikace mezi uzly probíhá na základě protokolu μIP. Senzorové uzly po inicializaci očekávají žádosti o své údaje na daném portu. Mobilní uzel po své inicializaci vyšle broadcast zprávu a zpracovává získané informace. Po nashromáždění dostatečného počtu dat pokračuje k výpočtu své pozice. Jelikož se pozice mobilního senzorového uzlu může měnit, dotazuje se na pozice senzorových uzlů periodicky.

Vzdálenost mezi jednotlivými senzory pak může být určena buď podle měření délky doby letu radiového signálu nebo pomocí hodnoty RSSI získané z paketu odpovědi od senzorových uzlů. Pomocí zvolené varianty je pak přepočítána vzdálenost mezi uzlem mobilním a daným senzorovým.

Pokud se tedy mobilnímu senzorovému uzlu podaří v daném cyklu získat potřebné údaje – nejméně tři pozice kotevních senzorů a jejich vzdálenost od mobilního uzlu, může se pokusit vypočítat svoji pozici. Metodou triangulace tedy provádí nejprve řešení soustavy dvou kvadratických rovnic o dvou neznámých – nalezení průsečíků dvou kružnic. Pomocí třetího údaje pak určí svou pozici ze dvou možných.

5.2.2 Uzly senzorové sítě lokalizují mobilní uzel

Komunikace zajišťuje rovněž, jako v prvním případě, protokol μIP. Uzly senzorové sítě očekávají signál od mobilního uzlu. Mobilní uzel vysílá v daném intervalu broadcast zprávu, ze které poté senzorové uzly zjišťují jeho pozici.

Jednotlivé uzly senzorové sítě tedy po zachycení signálu mobilního uzlu vypočítají zvolenou metodou svoji vzdálenost od mobilního uzlu a tento údaj, spolu s údaji o své pozici, předají centrálnímu nebo vybranému senzorovému uzlu, který provede lokalizaci mobilního uzlu.

Lokalizace je provedena opět metodou triangulace a vypočítána v případě, že v dané periodě obdrží centrální nebo vybraný senzorový uzel nejméně dvě zprávy ze senzorové sítě. Tedy spolu se svými údaji má nejméně tři potřebné údaje k provedení metody triangulace.

5.3 Implementace řešení – vývojové prostředí

Implementaci chování senzorových uzlů jsem provedl v prostředí Contiki-3.0 OS, který je specializovaný pro efektivní práci s pamětí síťových vestavěných systémů a bezdrátových senzorových sítí. Contiki je napsáno v jazyce C. Nejedná se o standardní verzi C, ale o jeho zjednodušenou a chudší verzi. Bylo tedy nutné si vytvořit i některé funkce, které jsou při práci v jazyce C naprosto běžné, jako je například vypsání hodnoty proměnné datového typu *float* na standardní (simulační) výstup, převedení její hodnoty do řetězce datového typu *char* nebo naopak. Contiki jsem blíže popsal v kapitole 4. K simulaci chování uzlů jsem využil simulátoru Cooja.

Ve svém řešení jsem použil zmíněné dva typy senzorových uzlů, kdy je jeden typ statický nebo-li kotevní, který se nepohybuje a předává své údaje. Druhým typem je mobilní, může se pohybovat, potřebuje se tedy dotazovat statických uzlů na jejich pozice nebo si jeho pozici zjišťují uzly dané senzorové sítě, což je druhá varianta návrhu, uvedená v podkapitole 5.2.2.

5.4 Implementace na zařízení MicaZ

Informace ohledně zařízení MicaZ a jeho parametry jsem čerpal ze zdroje uvedeného v seznamu použité literatury [4].

MicaZ zařízení umožňuje vývoj vlastních aplikací čidel a je speciálně optimalizováno pro sítě s nízkou spotřebou energie s napájením pomocí baterií. Je založeno na open-source TinyOS operačním systému a poskytuje spolehlivé připojování ad-hoc sítě. Používá se jako middleware serverů pro integraci podnikových sítí nebo klientské uživatelské zařízení pro analýzu a konfiguraci.

Obrázek 5.1: MicaZ zařízení

5.4.1 Problém s překladem zdrojových kódů v Cooja

Při implementaci na zařízení MicaZ se mi objevil problém, kdy zdrojové soubory nešly na daném zařízení přeložit. Jednalo se o chybu v prostředí Contiki. Řešení problému jsem nalezl[1] a překlad tak nakonec úspěšně proběhl.

Problém spočíval v hlavičkovém souboru daného mikrokontroleru. Soubor v umístění *home/contiki/cpu*99/*include/avr/boot.h*[2] bylo nutné nahradit souborem, který jsem umístil do zdrojových souborů *pom/boot.h*, nebo provést aktualizaci knihovny avr-libc alespoň na verzi 1.8.1.

Další problém vznikl při vytváření síťové komunikace mezi senzorovými uzly platformy MicaZ. Využití protokolu μIP nebylo možné použít, viz níže podkapitola 5.4.3. Zprovoznění síťové komunikace pomocí protokolu Rime pak vyžadovalo nahrazení souboru v umístění *home/contiki/platform/micaz/contiki* − *conf.h* souborem, který jsem opět umístil do svých zdrojových souborů *pom/contiki* − *conf.h*, anebo nastavit konstantu CC2420_ CONF_ AUTOACK na hodnotu nula.

[1] Řešení daného problému jsem nalezl na uvedeném zdroji [5].

[2] Zdroj pro stažení knihovny avr jsem uvedl v seznamu použité literatury [6], taktéž se zde uvedl zdroj pro manuál k instalaci [12].

Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128KB	
Measurement (Serial) Flash	512KB	>100,000 Measurements
Configuration EEPROM	4KB	
Serial Communications	UART	0-3 V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	$< 15\ \mu$A	Sleep mode
RF Transceiver		
Frequency band	2400 MHz to 2483.5 MHz	ISM band
Transmit (TX)	data rate 250kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+5 MHz channel spacing
	38 dB	-5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 μA	Idle mode
	1 μA	Sleep mode
Electromechanical		
Battery	2x AA	Attached pack
External Power	2.7 V – 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (mm)	58x32x7	Excluding batteries
Weight (grams)	18	Excluding battery pack
Expansion Connector	51-pin	All major I/O signals

Tabulka 5.1: Parametry zařízení MicaZ

5.4.2 Komunikace mezi senzorovými uzly

Při implementaci komunikace mezi senzorovými uzly jsem narazil na problém. Za použití protokolu μIP, konkrétněji s využitím protokolu UDP, došlo sice k úspěšnému překladu zdrojového kódu na platformě MicaZ, ale v simulačním prostředí byly vytvořeny uzly, které měly všechny přiděleny IP adresu 0. Mezi vytvořenými uzly tedy neprobíhala žádná komunikace. Vyřešení toho problému přineslo využití protokolu Rime, popsaného v podkapitole 4.2. Pomocí protokolu Rime byl každý uzel vytvořen se specifickou IP a MAC adresou na základě svého ID.

Na platformě MicaZ jsem tedy nejprve vytvořil jednoduchou komunikaci. Na každém senzorovém uzlu běžely dvě vlákna.

Za pomocí hlavičkového souboru *net/rime.h* první vlákno vytvoří broadcast komuni-

kaci *broadcast_open*(). První vlákno v periodických intervalech odesílá zprávu na broadcast pomocí funkce *broadcast_send*(). Pokud některý senzorový uzel obdrží broadcast zprávu, do globální proměnné typu *rimeaddr_t* uloží IP adresu odesílatele a aktivuje příznak zprávy. O zachycení broadcast zprávy se starají funkce definované ve struktuře *broadcast_callbacks*.

Druhé vlákno ve svém životním cyklu čeká na aktivaci příznaku zprávy. Pokud je příznak aktivní, odešle odpověď. Je vytvořeno spojení na konkrétní uzel za pomocí funkce *unicast_open*(). Na IP adresu z globální proměnné *rimeaddr_t* je odeslána pomocí funkce *unicast_send*() odpověď a je deaktivován příznak zprávy. Toto vlákno taky obstarává zachycení unicast zprávy pomocí funkce definované ve struktuře *unicast_callbacks*.

5.4.3 Souhrn implementace na platformě MicaZ

Komunikaci na platformě MicaZ se mi nepodařilo správně zprovoznit. Senzorové uzly si posílají zprávy přes broadcast, ale žádným způsobem na ně nereagují.

Zdrojový kód jsem umístil do souboru *broMicaZ/my_broadcast.c*.

5.5 Implementace na zařízení Z1

Informace ohledně zařízení Z1 a jeho parametry jsem čerpal ze zdroje uvedeného v seznamu použité literatury [17] a [16].

Zařízení Z1 od Zolertie je nízkonapěťový bezdrátový modul, který slouží jako platforma pro všeobecné účely pro vývojáře bezdrátových senzorových sítí. Dodává se s podporou některých v současné době nejvíce používaných open source operačních systémů ze strany komunity bezdrátových senzorových sítí, jako jsou Tiny OS a Contiki OS. Modul je kompatibilní s IEEE 802.15.4 a Zigbee protokoly. Architektura je založena na MSP430+CC2420 mikrokontroleru a radio vysílači od Texas Instruments.

Obrázek 5.2: Zolertia Z1 zařízení

5.5.1 Parametry Z1

- ideální vývojová platforma pro vytváření prototypů a vývoj bezdrátových senzorových sítí
- teplotní rozsah průmyslové třídy (-40 °C až 85 °C)
- 52-pinový konektor expanze
- 2. generace MSP430 ultra-nízká spotřeba 16-bit MCU 16MHz
- 2,4 GHz IEEE 802.15.4, 6LoWPAN vyhovující, ZigBee ready
- 3 osí, ±2/4/8/16 g digitální akcelerometr
- nízkospotřebový digitální snímač teploty s přesností ±0.5°C (-25 °C až 85 °C)
- volitelná externí anténa přes U.FL konektor
- konektor microUSB pro napájení a ladění
- 2x AAA nebo 2x AA baterie
- napájecí napětí Vcc -0.3 V na +3,6 V
- napětí na jakémkoliv digitálním pinu -0,3 až Vcc + 0,3 V
- maximální RF vstupní výkon 10 dBm
- MSP430f2617 od 1,8 V do 3,6 V, spotřeba:
  - 0,1 μA mód vypnutí
  - 0,5 μA pohotovostní režim
  - 0,5 mA aktivní režim @ 1 MHz
  - <10 mA aktivní režim @ 16 MHz
- CC2420 od 2,1 V do 3,6 V, spotřeba:
  - < 1 μA mód vypnutí
  - 20 μA Power Down
  - 426 μA klidový režim
  - 18,8 mA režim RX
  - 17,4 mA režim TX @ 0 dBm

5.5.2 Kotevní senzorový uzel

Statický senzorový uzel, který se tedy nepohybuje. Senzor tedy musí znát svoji pozici, která je mu přidělena manuálně.

Ve své práci jsem v každé simulaci vytvořil údaje o rozmístění daných kotevních uzlů. Tyto údaje jsem vložil do pole datového typu $float$. Každý kotevní uzel pak dostane údaje přiděleny na základě svého ID (v simulaci). Toto přidělení údajů provádí funkce $init()$.

Kotevní uzel své údaje propíše do připravené struktury *sensor_position*. Jedná se o vytvořenou strukturu, která obsahuje základní vlastnosti senzorového uzlu pro řešení problému určení pozice mobilního senzorového uzlu. Struktura obsahuje vlastnosti:

X - údaj o pozici x senzoru v dané senzorové síti,
Y - údaj o pozici y senzoru v dané senzorové síti,
D - používá se pro uložení vypočtené vzdálenosti mezi kotevním a mobilním uzlem.

Své ID v rámci simulace, v prostředí Contiki simulátoru Cooja, získá senzorový uzel díky vlastnosti hlavičkového souboru *node-id.h*.

Ve svém životním cyklu pak kotevní uzel přijímá a poté odpovídá na žádosti o údaj o své pozici. Což je implementace varianty jedna uvedené v podkapitole 5.2.1 a zdrojový kód kotevního uzlu je uložen v souboru *vyp/sen.c*.

V druhé variantě ve svém životním cyklu kotevní uzly přijmou pouze signál mobilního uzlu. Ten nevyžaduje žádnou odpověď. Ovšem kotevní uzly vypočítají vzdálenost od mobilního uzlu a údaj předají senzorovému uzlu s ID 1. Senzorový uzel s ID 1 provede shromáždění informací, výpočet pozice mobilního uzlu a provede výpis směru pohybu mobilního uzlu. V tomto případě provádí uzel s ID 1 shromažďování údajů pro konkrétní výpočet po dobu určenou časovým intervalem. Po vypršení daného intervalu a nashromáždění potřebných údajů dojde k výpočtu pozice mobilního uzlu. V případě, že nedojde k nashromáždění potřebných údajů (nelze provést výpočet lokalizace), dochází k resetování časovače a jsou sbírány nové hodnoty pro následující výpočet. Zdrojový kód pro tuto variantu chování senzorových uzlů je uložen v souboru *sen.c*.

5.5.3 Mobilní senzorový uzel

Jedná se o senzorový uzel, který se v dané senzorové sítí může libovolně pohybovat. Jeho pozice se tedy může měnit.

Po spuštění je stejně jako u kotevního uzlu provedena inicializace pomocí struktury *sensor_position*.

Dále pak dochází k nastavení časovače, tuto funkčnost zajišťuje hlavičkový soubor *sys/ctimer.h*, časovač jsem nastavil na dobu 5 s.

V první variantě uvedené v podkapitole 5.2.1 pak dochází v dané periodě vždy k odeslání žádosti o pozice kotevních uzlů. Ve svém životním cyklu pak mobilní uzel zpracovává odpovědi kotevních uzlů. Pokud dojde k vypršení daného času, nastaveného v časovači, dochází ke kontrole, jestli se mobilnímu uzlu podařilo nashromáždit potřebný počet údajů – nejméně od tří kotevních uzlů. Pokud ano, senzor se pomocí funkce *posComputation*() pokusím vypočítat svoji pozici. V případě úspěchu i neúspěchu pak vždy dochází k resetování časovače. A znovu odeslání žádostí o pozice kotevních uzlů. Zdrojový kód mobilního uzlu je uložen v souboru *vyp/mob.c*.

V druhé variantě uvedené v podkapitole 5.2.2 pak dochází pouze k periodickému vysílání signálu do senzorové sítě. Mobilní uzel v této variantě výpočet neprovádí a tudíž ani nemá zapotřebí přijímat odpovědi od kotevních uzlů. Zdrojový kód tohoto typu chování mobilního uzlu je uložen v souboru *mob.c*.

5.5.4 Komunikace mezi senzorovými uzly

Komunikaci mezi senzorovými uzly jsem implementoval v prostředí Contiki pomocí protokolu μIP. Konkrétněji s využitím protokolu UDP. Myslím, že není potřeba zajišťovat spolehlivé doručení paketu a prodlužovat tak dobu určení pozice mobilního senzorového uzlu.

Po počáteční inicializaci dochází u každého typu senzorového uzlu k inicializaci jeho radiového spojení. O tuto skutečnost se stará funkce *udp_new*() implementovaná v *net/ip/uip.h*. Funkce vyžaduje jako vstupní parametr pouze port druhého zařízení, se kterým bude probíhat komunikace.

Pokud je spojení správně vytvořeno, pomocí funkce *udp_bind*() dojde k provázání spojení se vstupním portem senzoru.

Mobilní uzel vyžaduje komunikaci a provádí výpočet
V případě kotevního uzlu dochází k nekončícímu zpracovávání žádostí o přeposílání pozice. Ve svém cyklu proces čeká na událost. Pokud nastane událost přijetí paketu, obsluha je předána funkci *sen_revc*(), která se zajišťuje zpracování žádosti. V uvedené funkci dochází k identifikaci žádosti. Pokud byla rozpoznána žádost typu $rfPOS$, dojde k vložení pozice kotevního senzoru do zprávy, pomocí funkce $floatToString()$, a k odeslání odpovědi na konkrétní adresu žadatele pomocí funkce *uip_udp_packet_send*(). Po odeslání dochází opět k čekání na událost přijetí další žádosti/paketu.

Mobilní uzel odešle broadcast žádost o pozice kotevních uzlů a pak po dobu určenou časovačem, čímž je určena jedna perioda, provádí zpracování odpovědí. Proces čeká na událost přijetí paketu. Pokud nastane daná událost, dochází k zpracování odpovědi, které zajišťuje funkce *mob_recv*(). Pokud byla rozpoznána odpověď typu *ans*, dojde k vytažení údajů pozice kotevního uzlu. V každé periodě mobilní uzel ví o počtu aktuálně získaných údajů pozic a vzdáleností kotevních senzorů. Údaj o daném počtu je zaznamenán v globální proměnné. Proto uložení údajů o pozicích a vzdálenostech kotevních senzorů slouží struktura pole typu *sensor_position*. O vytažení údajů z paketu se stará funkce $stringToFloat()$. Formát zprávy:

$$ans\, x\_pozice\, y\_pozice.$$

Podle zvolené metody pak dochází k výpočtu vzdálenosti mobilního uzlu od daného senzorového uzlu a přidělení této vypočtené hodnoty k údajům pozice daného senzorového uzlu. Po vypršení časovače, ať už dojde k určení pozice s využitím funkce $posComputation()$ mobilního uzlu nebo ne – vlivem nedostatku přijatých odpovědí, dochází k nulování čítače přijatých odpovědí. V následující periodě jsou tedy staré hodnoty přepsány. Hodnotu maximálního počtu přijatých odpovědí jsem si stanovil a lze ji změnit podle velikosti simulované senzorové sítě. Ovšem nesmí být nastavená na nižší hodnotu než 3 – nikdy by nedošlo k určení pozice mobilního uzlu.

Uzly senzorové sítě lokalizují mobilní uzel
V tomto případě je chování mobilního uzlu velmi jednoduché, pouze periodicky vysílá broadcast zprávu typu $mobSIG$. Mobilní uzel nežádá od senzorových uzlů žádné údaje a nevyžaduje tak ani žádné odpovědi.

Oproti variantě, kdy si výpočet lokalizace provádí sám mobilní uzel, je potřeba, aby si uzly senzorové sítě ještě vytvořily spojení pro komunikaci mezi sebou. Každý uzel pak čeká na signál $mobSIG$ od mobilního uzlu.

Pokud uzel senzorové sítě daný signál obdrží, pomocí zvolé metody vypočte svoji vzdálenost od mobilního uzlu, pomocí funkce *getDist*(). Poté vloží svoje údaje, s využitím funkce *floatToString*(), o své pozici spolu s vypočtenou vzdáleností do zprávy typu *myD* a údaje odešle senzorovému uzlu s ID 1. U této varianty lze vypočíst vzdálenost od mobilního uzlu pouze pomocí hodnoty RSSI. Pokud bychom chtěli vypočítat vzdálenost pomocí délky letu rádiového signálu, jednou z variant by bylo přidat časový údaj vyslání signálu *mobSIG* na broadcast, uzly senzorové sítě by pak musely být časově synchronní s mobilním uzlem.

Senzorový uzel s ID 1 pak navíc oproti jiným uzlům senzorové sítě nastaví časový interval, ve kterém přijímá jak signál od mobilního uzlu, tak údaje od ostatních uzlů senzorové sítě. Každý údaj od senzorového uzlu zpracuje za pomoci funkce *stringToFloat*(), vloží do struktury pole typu *sensor_position* a inkrementuje čítač. Formát zprávy:

$$myD\, x\_pozice\, y\_pozice\, vzdlenost\_od\_mobilnho\_uzlu.$$

Pokud v daném časovém intervalu nashromáždí potřebný počet údajů, provede výpočet lokalizace mobilního uzlu pomocí funkce *posComputation*(). Potřebný počet údajů je zaznamenáván pomocí čítače. Čítač je po vypršení časového intervalu nulován a staré údaje senzorových uzlů jsou v následujícím časovém intervalu přepsány novými hodnotami.

5.5.5 Výpočet vzdálenosti mezi dvěma senzory

Údaj vzdálenosti mezi kotevním a mobilním uzlem je nutné znát pro hlavní výpočet pozice mobilního senzoru.

V první variantě uvedené v podkapitole 5.2.1 tento výpočet provádí mobilní uzel pro každou odpověď od senzorových uzlů. V druhé variantě tento výpočet provádí každý senzorový uzel, který zachytí signál od mobilního uzlu.

Určení vzdálenosti pomocí délky doby letu rádiového signálu

Jako jedna možností se nabízí měřit délku doby letu rádiového signálu. Mobilní uzel vyšle žádost o pozice kotevních uzlů a zapne časovač. Pokud přijme odpověď, uloží si daný čas vydělený hodnotou dvě (cesta tam i zpět).

Podle rovnice, která vychází ze základní rovnice pro výpočet rychlosti:

$$d = c \times t \tag{5.1}$$

kde c - je rychlost světla ve vzduchu 299 714 532 m/s, což je rychlost šíření rádiových vln,

t - je námi zaznamenaný čas, který uběhl při přenosu paketu mezi senzorovými uzly.

Jelikož je rychlost světla velmi velká, je nutné měřit čas v jednotkách nanosekund. V Contiki lze provádět měření času pomocí tiků MCU hodin. Po odeslání žádosti je tedy nastaven časovač. Po přijetí odpovědního paketu je pak vždy zaznamenán čas. Zaznamenaný čas je vydělen dvojkou a odečtena hodnota délky zpracovávání odpovědi na kotevním uzlu.

Výše uvedený způsob platí pro obousměrnou komunikaci. Pokud by byla vyžadována pouze jednosměrná komunikace, bylo by nutné k zprávě přiložit časový údaj označující dobu, kdy byl rádiový signál vyslán. Byla by taky nutnost časové synchronizace mezi senzorovými uzly. Poté by podle uvedeného vzorce mohl proběhnout výpočet vzdálenosti mezi senzorovými uzly.

Problémy

V datasheet daných platforem je uváděno, že jedna sekunda je rovna 128 tikům hodin. Při měření a implementaci v Cooja simulátoru jsem ovšem podle výpisu standardního výstupu přišel k zjištění, že jedné sekundě zde odpovídá 64 tiků hodin.

Při implementaci jsem využil knihovny *sys/clock.h*. Nastal problém, kdy odpovědi přicházely až po dlouhém čase. Na délku letu rádiového signálu neměla vliv vzdálenost mezi senzorovými uzly, nejčastěji letěl signál přes broadcast 0,1 s, odpověď senzorovému uzlu přišla nejčastěji až po 0,734 s, což by odpovídalo asi 87 573 km. Proč docházelo k tomuto problému, se mi nepovedlo odhalit.

Při použití funkce *clock_init*(), která je využita k inicializaci hodin, se mi stávalo, že mobilní uzel nepřijímal žádné odpovědi.

Proto jsem se rozhodl, že tuhle variantu výpočtu vzdálenosti mezi senzorovými uzly nepoužiji.

Určení vzdálenosti pomocí síly rádiového signálu RSSI

RSSI (received signal strenght indicator) je hodnota síly signálu naměřená na přijímacím zařízení. Pomocí této hodnoty lze vypočítat vzdálenost mezi vysílacím a přijímacím zařízením.

Nejčastěji uváděný vztah[3] mezi vzdáleností a hodnotou RSSI je:

$$RSSI_i = -10 n_i \, log_{10}(d_i) \, + \, A \tag{5.2}$$

po úpravě tedy:

$$d_i = 10^{-\frac{RSSI_i - A}{10 n_i}} \tag{5.3}$$

kde d_i - je vzdálenost mezi vysílačem a přijímačem v metrech,
$RSSI_i$ - je síla přijímaného signálu v dBm,
n_i - je konstanta šíření signálu,
A - síla přijímaného signálu ve vzdálenosti 1 m.

Uváděná hodnota konstanty n pro volný prostor je rovna hodnotě 2. Pokud jsem se ovšem pokusil implementovat tento vztah v prostředí contiki, nedosahoval jsem správných výsledků.

Proto jsem se rozhodl pro určení vzdálenosti mezi vysílačem a přijímačem použít vlastní vztah. Provedl jsem sérii měření, viz kapitola 7.1. Z naměřených hodnot jsem pak vytvořil polynom 3. stupně, který popisuje vztah mezi hodnotou $RSSI$ a vzdáleností d. Měření a určení polynomu blíže popisuji v kapitole 7.1. Můj vztah závislosti d na hodnotě $RSSI$:

$$d_i = 3 \times 10^{-6} \times |RSSI_i|^3 - 3 \times 10^{-4} \times |RSSI_i|^2 + 0,5911 \times |RSSI_i| - 5,519 \tag{5.4}$$

Jelikož MCU senzorových zařízení pracují většinou v 16bitové soustavě, musel jsem v kódu použít postupné násobení mocnin a vhodné vynásobení hodnotou 10^{-6}, abych nepřišel o desetinou část hodnoty. Vztah je implementovaný ve funkci *getDist*(), která jako

[3] Tento vztah jsem nalezl v dokumentu, jehož zdroj jsem uvedl do seznamu použité literatury [3].

vstupní parametr vyžaduje naměřenou hodnotu $RSSI$, výstupem funkce je vzdálenost v metrech.

Pro změření hodnoty $RSSI$ jsem využil hlavičkového souboru $cc2420.h$, zde proměnou $cc2420\_last\_rssi$, která zastupuje hodnotu $RSSI$ posledního přijatého paketu.

5.5.6 Výpočet pozice mobilního uzlu

Tuto vlastnost využívá mobilní senzorový uzel v případě první varianty uvedené v podkapitole 5.2.1 anebo pouze uzel senzorové sítě s ID 1, varianta popsaná v podkapitole 5.2.2.

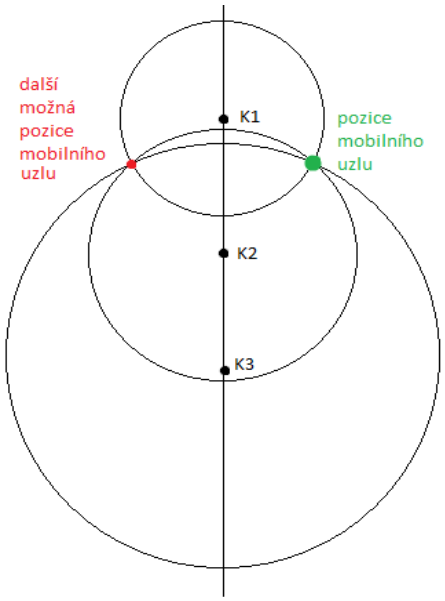
Výpočet pozice mobilního uzlu provádí funkce $posComputation()$ umístěná ve zdrojovém souboru $myH/posComp.h$. Do proměnných x a y, datového typu $float$, zapisuje funkce vypočtené souřadnice. Dalšími požadovanými parametry jsou pole senzorů struktury typu $sensor\_position$ a počet kotevních uzlů, od kterých byly přijaty odpovědi.

Pole $sensor\_position$ obsahuje údaje spojené s kotevními uzly. Obsahuje hodnoty souřadnic x a y daného kotevního uzlu a hodnotu d, která udává vzdálenost od mobilního uzlu.

K výpočtu pozice mobilního uzlu se primárně používají první tři přijaté odpovědi kotevních uzlů, pořadí je dáno časem přijetí odpovědi na žádost o pozice kotevních uzlů.

Kontrola výpočetní podmínky

Před zahájením samotného výpočtu je nejprve zkontrolována podmínka, jestli nejsou dané kotevní uzly umístěny v jedné přímce. O tuto kontrolu se stará funkce $isLine()$. Pokud by k této situaci došlo, nebylo by možné pozici mobilního uzlu vypočíst. Při výpočtu bychom dostali dvě pozice a nemohli bychom určit, která je ta správná, viz ilustrace obrázek 5.3.



Obrázek 5.3: Pozice kotevních uzlů v přímce

Funkce $isLine()$ tedy nejprve vytvoří rovnici přímky. K této činnosti jsou požity údaje ze dvou prvních kotevních uzlů.

Nejdříve vytvořím směrový vektor a z něj poté normálový $n = (nX, nY)$. Pomocí údajů prvního kotevního uzlu pak dopočítám koeficient c a vytvořím tak obecnou rovnici přímky[4], která prochází dvěma prvními kotevními uzly.

$$nX \times x + nY \times y + c = 0 \tag{5.5}$$

Po vytvoření rovnice je vzat údaj třetího kotevního uzlu a je určena jeho náležitost daném přímce. Souřadnice kotevního uzlu jsou dosazeny do uvedené rovnice. Pokud je výpočet uvedené rovnice roven nule, kotevní uzel leží na stejné přímce. Mezi dalšími kotevními uzly se tedy pokusíme najít takový, který na stejné přímce neleží.

K výslednému výpočtu pozice mobilního uzlu se vždy využívají první tři kotevní uzly (pořadí určeno dobou přijetí odpovědi) a pokud dochází k záměně, je vždy vyměněn třetí kotevní uzel s nalezeným jiným vhodným kotevním uzlem, tedy s kotevním uzlem s indexem vyšším než 2.

V simulačním prostředí můžu kotevní uzly umístit na přesné pozice a dosáhnout tak toho, že budou opravdu ležet na přímce. V reálném prostředí toho, aby kotevní uzly ležely přesně na přímce, nedocílíme tak snadno, tento problém nemusí tedy v reálném prostředí ani nastat a pozice mobilního uzlu může být vypočtena vždy.

Výpočetní algoritmus metodou triangulace

Pokud funkce $isLine()$ skončí s návratovou hodnotou $false$, tedy byl nalezen dostatečný počet kotevních uzlů, které neleží na stejné přímce. Můžeme přistoupit k samotnému výpočtu pozice mobilního uzlu.

Nejprve jsou z prvních dvou kotevních uzlů vytvořeny obecné rovnice kružnic se středem určeným pozicí daného kotevního uzlu a o poloměru, který je dán vzdáleností od mobilního uzlu.

Obecná rovnice kružnice[5]:

$$(x - m)^2 + (y - n)^2 = r^2 \tag{5.6}$$

kde m - je x souřadnice kotevního uzlu,
n - je y souřadnice kotevního uzlu,
r - je vzdálenost d od mobilního uzlu.

Z obecné rovnice kružnice pak vytvořím rozvinutou rovnici kružnice s konstantou p:

$$x^2 + y^2 - 2mx - 2ny + p = 0 \tag{5.7}$$

$$p = m^2 + n^2 - r^2 \tag{5.8}$$

[4] Uvedenou obecnou rovnici přímky jsem nalezl v knize, jejíž zdroj jsem zapsal do seznamu použité literatury [11].

[5] Rovnici kružnice jsem přebral z knihy, jejíž zdroj jsem zapsal do seznamu použité literatury [11].

Dále tedy řeším soustavu dvou kvadratických rovnic o dvou neznámých. Matematickými úpravami vyjádřím x, dosadím do rovnice kružnice a vypočtením diskriminantu získám hodnoty $y_{1,2}$. Dosazením $y_{1,2}$ k dříve vyjádřenému x vypočtu hodnoty $x_{1,2}$. Získal jsem tedy souřadnice obou průsečíků.

Pomocí získaných souřadnic průsečíků vypočtu vzdálenosti od třetího kotevního uzlu. Pomocí vztahu pro výpočet vzdálenosti mezi dvěma body:

$$|D_{1,2}| = \sqrt{(P_{1,2x} - K_{3x})^2 + (P_{1,2y} - K_{3y})^2} \tag{5.9}$$

Vypočtené vzdálenosti pak porovnám s naměřenou vzdáleností třetího kotevního uzlu od uzlu mobilního. Kratší vzdálenost, v ideálním případě nulová, určuje pozici mobilního uzlu.

5.5.7 Pomocné funkce

Jelikož je Contiki založeno na zjednodušené verzi jazyka C. Bylo nutno implementovat některé z funkcí, které by na průměrném procesoru mohly být použity v rámci knihovny. Mnou implementované funkce jsem umístil do adresáře *myH*.

Funkce pro vypsání hodnoty čísla s pohyblivou desetinou čárkou

V prostředí Contiki nelze vypsat hodnotu proměnné typu *float* na standardní výstup. Pro tento účel jsem si tedy vytvořil vlastní funkci *printFloat*().

Jako vstupní parametry vyžaduje funkce hodnotu typu *float* určenou k výpisu na standardní výstup a parametr udávající počet desetinných míst.

Pokud je hodnota proměnné záporná, je nejprve vytištěno znaménko a následně je celá část čísla přetypována na celočíselný typ *int*, který Contiki dokáže pomocí funkce *printf*() vypsat na standardní výstup. Celá část čísla tedy může být vypsána a odečtena od vstupní hodnoty. Po vypsání znaku desetinné tečky je pak desetinná část čísla postupně násobena konstantou 10, simuluji tak posun desetinné čárky doleva. Vzniklé číslo pak opět přetypuji na celočíselný typ *int*. Toto celé číslo je pak odečteno od zvětšeného reálného čísla a proveden jejich rozdíl. Pokud je rozdíl roven nebo větší než hodnota 0,5, provedl jsem zaokrouhlení desetinného čísla, celé číslo je inkrementováno a pak vypsáno na standardní výstup za desetinou tečkou pomocí funkce *printf*().

Během posunu desetinné čárky ještě dochází ke kontrole, je-li celá část nulová. Pokud ano, dochází jednotlivě k výpisu hodnoty 0 na standardní výstup.

Funkce pro převedení hodnoty řetězce na číslo s pohyblivou desetinou čárkou

Funkce, která provádí převod proměnné datového typu *char** na reálné číslo datového typu *float*.

Jako vstupní parametr funkce vyžaduje ukazatel na proměnou datového typu *char**.

Nejprve dochází k identifikace, jedná-li se o záporné číslo. Dále je detekováno, na které pozici řetězce se vyskytuje desetinná tečka. Po nalezení desetinné tečky pak můžou být znaky vyskytující se před ní převedeny na celou část čísla. Znak čísla na určité pozici je patřičně vynásoben konstantou 10 a přičten k průběžné hodnotě výsledného reálného čísla. Naopak číselné znaky nacházejících se v řetězci za desetinou tečkou jsou podle své pozice poděleny konstantou 10 a rovněž přičteny k průběžné hodnotě výsledného reálného čísla.

Návratovou hodnotou funkce *strToFloat*() je tedy pak reálné číslo datového typu *float*.

Funkce pro převod čísla s pohyblivou desetinou čárkou na řetězec

Opačný proces než převod proměnné datového typu *char** na reálné číslo pak provádí funkce $floatToStr()$. Uvedená funkce vyžaduje jako parametry ukazatel na proměnou typu *char**, index určující, odkud do řetězce se má reálné číslo postupně zapisovat, hodnotu reálného čísla datového typu $float$ a počet desetinných míst, jež se mají do řetězce zapsat.

Reálné číslo je dle svého řádu poděleno konstantou 10. Když tedy dostaneme číslo v řádu jednotek, je přetypováno na datový typ *int* a tako hodnota je zapsána jako znak do pole datového typu *char*. Zapsaná hodnota je vždy odečtena a pak se provádí opačný proces, tedy násobení čísla konstantou 10. Číslo v řádu jednotek je opět vždy přetypováno na celočíselný datový typ *int*, odečteno od vstupní hodnoty a celočíselná pomocná proměnná vložena do řetězce. Podle počátečního řádu reálného čísla je vložena desetinná tečka a stejný postupem jako celá část je vložena do řetězce i desetinná část čísla. Z desetinné části je ovšem do řetězce vložen jen počet cifer zadaný jako vstupní parametr.

5.5.8 Souhrn implementace na platformě Z1

Zadání mé diplomové práce se mi na senzorovém uzlu Z1 podařilo naimplementovat. Bohužel stále dochází k jistým nesrovnalostem:

1. Senzorový uzel ať mobilní či kotevní s ID 1 začíná přijímat odpovědi od ostatních uzlů až zhruba po 40 s spuštění simulace. Důvod, proč k tomu dochází, se mi nepodařilo odhalit.

2. Občas nastane situace, kdy v daném časovém intervalu nepřijme mobilní či kotevní uzel s ID 1 žádnou zprávu od ostatních senzorových uzlů.

3. Dochází k výpočetním nepřesnostem, velikost chyby je blíže popsána v kapitole 7.

Kapitola 6

Simulator Cooja

Cooja je síťový simulátor speciálně navržený pro bezdrátové senzorové sítě. Je přidružený ke Contiki OS. Provádí tedy překlad zdrojových souborů a simulaci navržených algoritmů v prostředí Contiki.

Lze simulovat různé typy senzorových uzlů a vytvářet tedy senzorové sítě jak homogenního tak heterogenního typu.

6.1 Spuštění simulátoru

Pro spuštění simulátoru lze využít souboru *cooja.desktop*, který je umístěn v adresáři *pom* se zdrojovými soubory na přiloženém cd. Soubor zkopírovat na plochu a přiřadit právo pro spuštění pomocí příkazu *chmod* 755 *cooja.desktop*. Poté stačí jen soubor spustit.

Další možností je v terminálu zadat následující příkazy:

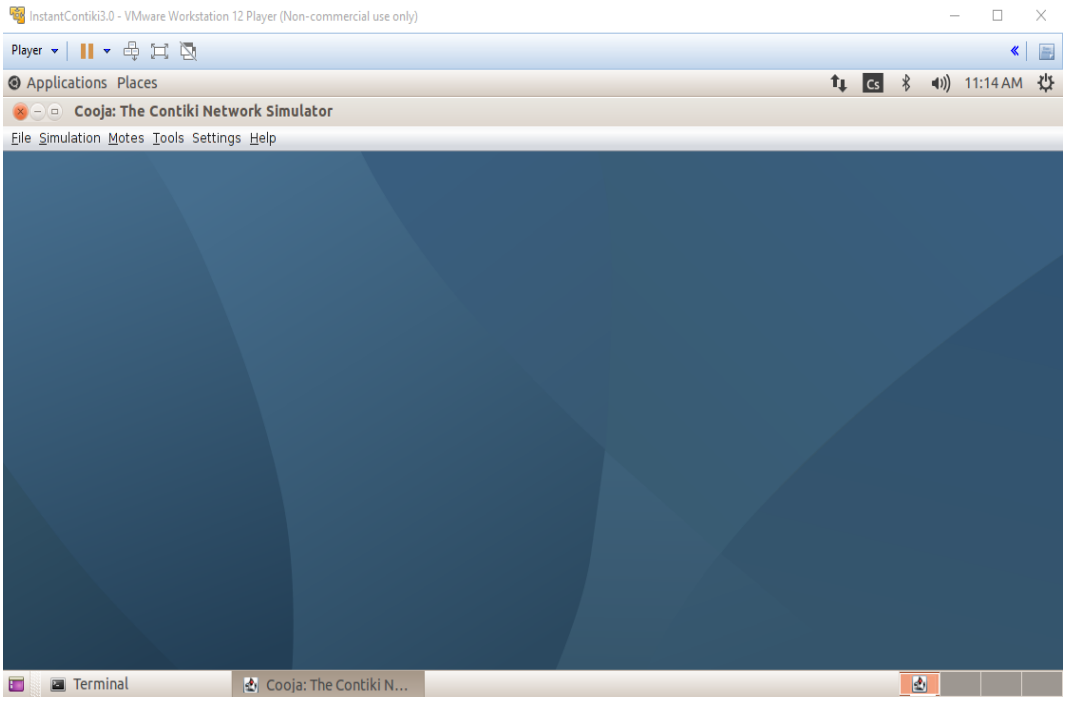
cd contiki/tools/cooja
ant run

6.2 Simulace

Vytvoření simulace je možné provést kliknutím na *File* > *New simulation*, kde je pak možné zvolit jméno simulace a prostředí rádiového signálu.

Pomocí tlačítka *View* v okně *Network* je možné si v simulaci zapnout vizualizační pomůcky, jako jsou:

- ID uzlu
- Adresa IP nebo Rime
- Pozice
- Rádiová komunikace
- 10 metrová mřížka
- ...



Obrázek 6.1: Spuštění Cooja simulátoru

6.3 Přidání senzorových uzlů

Simulační senzorový uzel je pak přidán pomocí tlačítka:

Motes > *Add motes* > *Create new mote type*.

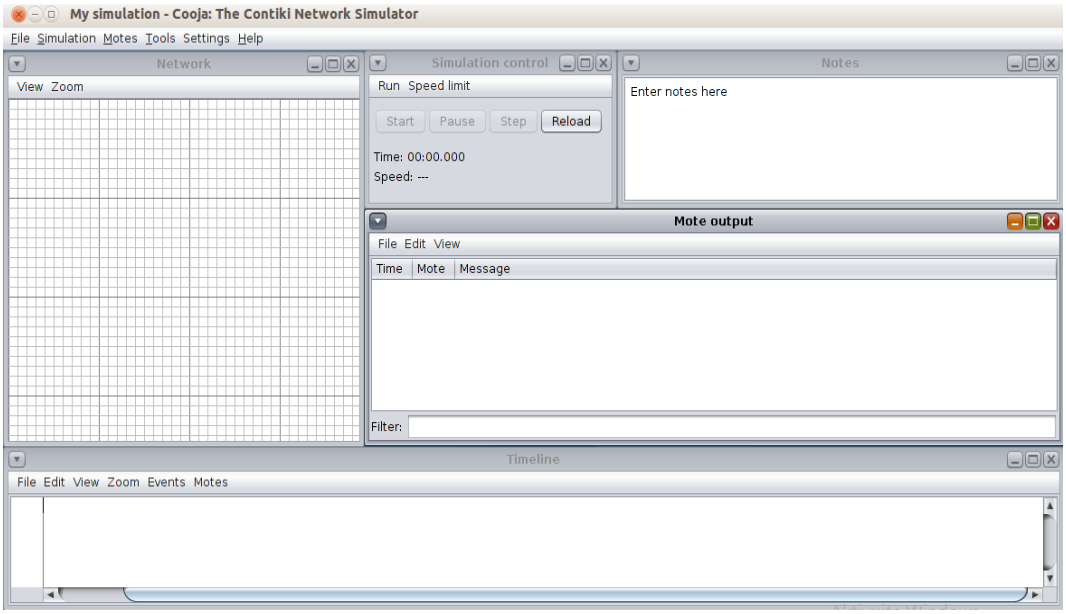
Zde je několik možností výběru typu senzorových uzlů, která simulátor Cooja podporuje: MicaZ, CC430, Wismote, Z1, Sky, ...

Po výběru daného senzorového uzlu jsme pak vyzváni k zadání cesty ke zdrojovému souboru pro vybraný senzorový uzel. Před vložením senzorového uzlu do simulace je nejprve nutné zdrojový kód zkompilovat. Poté, co je senzorový uzel vložen do simulace, pak máme možnost zadat počet vložených senzorových uzlů do simulačního prostředí a jejich pozice.

V okně *Network* lze vidět rozmístění jednotlivých senzorových uzlů. Jednotlivé typy senzorových uzlů lze rozlišit díky barevnému zvýraznění. Zobrazené vlastnosti lze dynamicky měnit i v průběhu simulace.

Spuštění simulace je provedeno pomocí tlačítka Start v okně *Simulation control*, zde je možné ještě zvolit rychlost simulace uvedenou v procentech.

Okno *Mote output*, po spuštění simulace, zobrazuje standardní výstupy jednotlivých senzorů.



Obrázek 6.2: Nová simulace v Cooja

6.4 Vlastní simulace

Pro simulaci mé diplomové práce jsem vytvořil šablony simulací ve formátu *csc*. Tyto šablony jsou umístěny vždy v adresáři se zdrojovými soubory pro senzorové uzly na přiloženém cd. Jedná se o různá rozmístění jak senzorových uzlů kotevních tak mobilních. Například šablona 9+1.*csc* obsahuje pravidelnou senzorovou síť s jedním mobilním senzorovým uzlem.

6.4.1 Struktura simulačních šablon

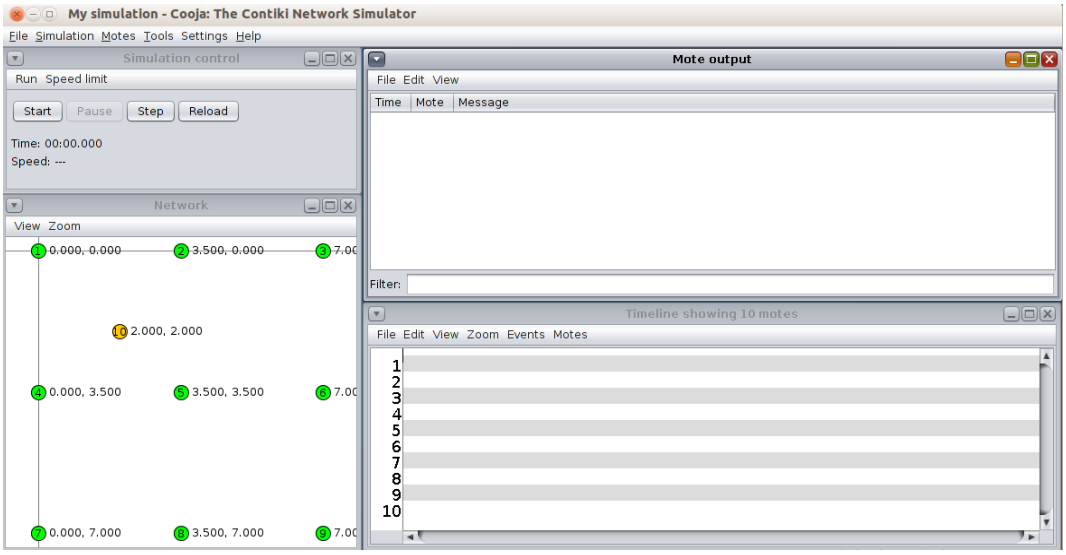
Soubory formátu *csc* jsou XML struktury. V každém souboru lze dodatečně měnit vlastnosti dané simulace: název, rychlost simulace, vlastnosti radiového vysílače,... Jednotlivé identifikátory XML struktury určují dané vlastnosti simulace.

Mezi nejdůležitější identifikátory patří:

motetype
Zde jsou zadány vlastnosti použitého typu (MicaZ, Z1, Sky,...) senzorového uzlu, jako je jeho označení, cesta ke zdrojovému kódu, příkaz ke kompilaci a rozhraní senzoru.

mote
Zde je popsáno umístnění senzorového uzlu a jeho označení v rámci simulace.

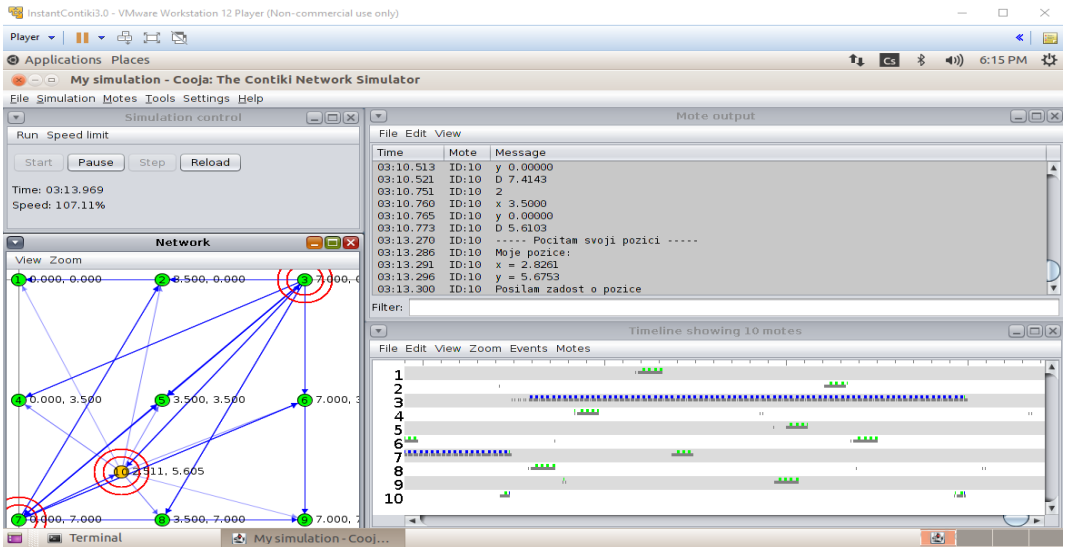
plugin
Je identifikátor, ve kterém jsou zadány parametry pro vizuální informace dané simulace, jako je například: zobrazení 10 metrové mřížky, zobrazení ID a pozic senzorových uzlů nebo radiové komunikace mezi senzorovými uzly.



Obrázek 6.3: Cooja simulace s vytvořenými senzorovými uzly

6.4.2 Spuštění simulace

Spuštění simulace je v Cooja simulátoru provedeno pomocí *File* > *Open simulation* > *Browse* . . ., kde pak v adresářové struktuře vybereme příslušný soubor. Pokud nebyly zdrojové soubory senzorových uzlů zkompilovány, jsou zkompilovány během nahrávání simulace. Zahájení simulace pak stačí spustit kliknutím na tlačítko *Start* v okně *Simulation control*.



Obrázek 6.4: Vlastní simulace

Kapitola 7

Měření a simulace na Z1

Při řešení mé diplomové práce jsem navrhl několik testovacích scénářů. Jedná se o testy validace implementovaných algoritmů. Zdrojové kódy pro měření a testování jsem umístil do adresáře *test*. Testy jsem implementoval v jazyce C++. V podadresáři *in* se nacházejí soubory se vstupními hodnotami pro jednotlivé testy. Naopak v podadresáři *out* se nacházejí soubory s očekávanými výstupními hodnotami navržených testů a měření. Skript *test.sh* provádí překlad zdrojových souborů, spuštění jednotlivých testů a provede porovnání výstupů z testů s očekávanými výsledky.

Měření jsem prováděl pomocí simulátoru Cooja na platformě Zolertia Z1 a tak si stanovil očekávané výstupy.

7.1 Měření vzdálenosti mezi senzorovými uzly

Znát vzdálenost mezi senzorovým a mobilním uzlem je důležité pro samotný výpočet pomocí metody triangulace. Při měření jsem použil vždy dva senzorové uzly, které jsem umístil do předem známé vzdálenosti. Podle získaných výsledků jsem pak očekával pomocí výpočtu mnou navrženého algoritmu výpočet stejné vzdálenosti, jako byla ta v simulačním prostředí.

7.1.1 Testování podle délky letu rádiového signálu

Po odeslání žádosti byl zapnut časovač pro měření délky letu signálu. Po obdržení odpovědi byla výsledná doba vydělena dvěma a odečtena doba zpracování. Naměřené hodnoty jsem porovnával s předpokládanými.

Z uvedených výsledků je patrné, že tato varianta nemohla být pro řešení mého diplomového projektu použita. Průměrně mi vycházel, že druhý uzel přijal žádost asi za 0,1 s a odpověď se vrátila po 0,3672 s.

Na dobu přijetí zpráv neměla vliv vzdálenost mezi senzorovými uzly. Na důvod, proč v simulátoru docházelo k tak dlouhému čekání, než došlo k přijetí odpovědi, a proč neměla vzdálenost vliv na délku letu signálu, se mi nepodařilo přijít.

vzdálenost (skutečná) [m]	**naměřený průměrný čas [s]**	**vzdálenost (vypočtená) [m]**
0,5 **1** **5** **10** **15** **20** **25** **30** **35** **40**	0,3672	87 573 127

Tabulka 7.1: Měření délky letu rádiového signálu

7.1.2 Testování podle naměřené hodnoty RSSI

Jako první implementace se nabízelo řešení podle vztahu 5.3.

V literatuře je uvedeno, že koeficient n by ve volném prostoru měl být roven hodnotě 2. Podle výsledků měření v tabulce 7.2, jsem zjistil, že hodnota n je závislá spolu s $RSSI$ na vzdálenosti. Pokud bych použil například průměrnou hodnotu n, docházelo by k velkým nepřesnostem. Proto jsem se rozhodl uvedený vzorec nepoužít.

Pro řešení tohoto problému jsem se rozhodl vytvořit vztahy mezi naměřenými hodnotami. Pomocí naměřených hodnot jsem vytvořil graf 7.1. Výsledný graf jsem pak proložil lineární přímkou a následně i křivkou pomocí polynomu 2., 3. a 4. stupně. Pomocí vzniklých vztahů provedl výpočty, jejichž výstupy jsem uvedl v tabulce 7.3 a výsledky vzájemně porovnal.

Vztahy mezi hodnotou RSSI a vzdáleností:
Polynom 2. stupně

$$d_i = 3 \times 10^{-4} \times |RSSI_i|^2 + 0,5736 \times |RSSI_i| - 5,3546 \tag{7.1}$$

Polynom 3. stupně

$$d_i = 3 \times 10^{-6} \times |RSSI_i|^3 - 3 \times 10^{-4} \times |RSSI_i|^2 + 0,5911 \times |RSSI_i| - 5,519 \tag{7.2}$$

Polynom 4. stupně

$$d_i = 2 \times 10^{-7} \times |RSSI_i|^4 - 3 \times 10^{-5} \times |RSSI_i|^3 + 0,0018 \times |RSSI_i|^2 + 0,5393 \times |RSSI_i| - 5,146 \tag{7.3}$$

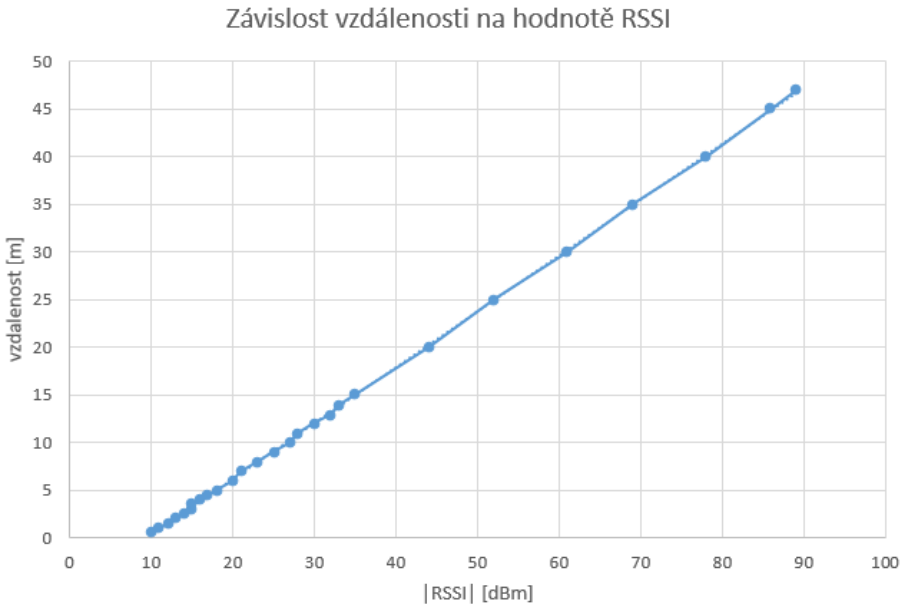
Lineární rovnice

$$d_i = 0,5869 \times |RSSI_i| - 5,5641 \tag{7.4}$$

vzdálenost [m]	**|RSSI| [dBm]**	n_i
0,5	10	0,332193
1,0	11	-
1,5	12	0,567887
2,0	13	0,664386
2,5	14	0,753882
3,0	15	0,838361
3,5	15	0,735202
4,0	16	0,830482
4,5	17	0,918537
5,0	18	1,001474
6,0	20	1,156587
7,0	21	1,183295
8,0	23	1,328771
9,0	25	1,467132
10,0	27	1,600000
11,0	28	1,632429
12,0	30	1,760594
13,0	32	1,885195
14,0	33	1,919506
15,0	35	2,040658
20,0	44	2,536452
25,0	52	2,932887
30,0	61	3,384962
35,0	69	3,756311
40,0	78	4,182116
45,0	86	4,536622
47,0	89	4,664799

Tabulka 7.2: Testovaní pomocí vztahu 5.3

V následující tabulce 7.4 jsem provedl výpočet odchylek od předpokládané hodnoty vzdálenosti.



Obrázek 7.1: Závislost vzdálenosti na hodnotě RSSI

Tabulka 7.5 pak obsahuje medián, průměrnou a maximální odchylku pro mnou navržené vztahy mezi hodnotou RSSI a vzdáleností.

|RSSI| [dBm]	**vzdálenost** [m]	**polynom 2. stupně** [m]	**polynom 3. stupně** [m]	**polynom 4. stupně** [m]	**lineární** [m]
10	0,5	0,3914	0,36500	0,39950	0,3049
11	1,0	0,9671	0,95079	0,96760	0,8918
12	1,5	1,5430	1,53618	1,53761	1,4787
13	2,0	2,1191	2,12119	2,10940	2,0656
14	2,5	2,6954	2,70583	2,68286	2,6525
15	3,0	3,2719	3,29013	3,25788	3,2394
15	3,5	3,2719	3,29013	3,25788	3,2394
16	4,0	3,8486	3,87409	3,83433	3,8263
17	4,5	4,4255	4,45774	4,41211	4,4132
18	5,0	5,0026	5,04110	4,99113	5,0001
20	6,0	6,1574	6,20700	6,15250	6,1739
21	7,0	6,7351	6,78958	6,73467	6,7608
23	8,0	7,8911	7,95410	7,90156	7,9346
25	9,0	9,0479	9,11785	9,07138	9,1084
27	10,0	10,2055	10,28105	10,24360	10,2822
28	11,0	10,7846	10,86246	10,83047	10,8691
30	12,0	11,9434	12,02500	12,00550	12,0429
32	13,0	13,1030	13,18730	13,18198	13,2167
33	14,0	13,6831	13,76841	13,77067	13,8036
35	15,0	14,8439	14,93063	14,94888	14,9774
44	20,0	20,0774	20,16415	20,26260	20,2595
52	25,0	24,743	24,82882	25,00938	24,9547
61	30,0	30,0071	30,10274	30,40934	30,2368
69	35,0	34,6999	34,82413	35,31415	34,9320
78	40,0	39,9946	40,18526	41,03755	40,2141
86	45,0	44,7146	45,00497	46,40558	44,9093
89	47,0	46,4879	46,82751	48,50938	46,6700

Tabulka 7.3: Vytvoření vztahů mezi naměřenými hodnotami

Podle uvedených výstupů v tabulce 7.5 je patrné, že nejpřesnějších výsledků bude dosahovat implementace podle vztahu určeným pomocí polynomu 3. stupně. V řešení mé diplomové práce jsem tedy využil tuto možnost. Vzorec polynomu 3. stupně je uveden v rovnici 5.4. Implementace toho testu se nachází ve zdrojovém souboru *test/test_distance.cpp*.

7.2 Testování vzájemné polohy senzorových uzlů

Další série testů byla navržena tak, aby bylo vyloučeno, že nastane problém, kdy se senzorové uzly vybrané k finálnímu výpočtu pozice mobilního uzlu pomocí metody triangulace, nacházejí v přímce. Daný problém byl detailněji popsán v podkapitole 5.5.6.

Test tohoto problému byl 100 % úspěšný. Pro všechny mé vstupy byly výstupy mého řešení vždy validní.

Implementace toho testu se nachází ve zdrojovém souboru *test/test_line.cpp*.

polynom 2. stupně	polynom 3. stupně	polynom 4. stupně	Lineární
0,1086	0,1350	0,1005	0,1951
0,0329	0,0492	0,0324	0,1082
0,0430	0,0362	0,0376	0,0213
0,1191	0,1212	0,1094	0,0656
0,1954	0,2058	0,1829	0,1525
0,2719	0,2901	0,2579	0,2394
0,2281	0,2099	0,2421	0,2606
0,1514	0,1259	0,1657	0,1737
0,0745	0,0422	0,0879	0,0868
0,0026	0,0411	0,0089	0,0001
0,1574	0,2070	0,1525	0,1739
0,2649	0,2104	0,2653	0,2392
0,1089	0,0459	0,0984	0,0654
0,0479	0,1179	0,0714	0,1084
0,2055	0,2811	0,2436	0,2822
0,2154	0,1375	0,1695	0,1309
0,0566	0,0250	0,0055	0,0429
0,1030	0,1873	0,1820	0,2167
0,3169	0,2316	0,2293	0,1964
0,1561	0,0694	0,0511	0,0226
0,0774	0,1642	0,2626	0,2595
0,2570	0,1712	0,0094	0,0453
0,0071	0,1027	0,4093	0,2368
0,3001	0,1759	0,3142	0,0680
0,0054	0,1853	1,0376	0,2141
0,2854	0,0050	1,4056	0,0907
0,5121	0,1725	1,5094	0,3300

Tabulka 7.4: Odchylky od referenční hodnoty vzdálenosti

	polynom 2. stupně	polynom 3. stupně	polynom 4. stupně	Lineární
průměrná odchylka	0,1594	0,1388	0,2830	0,1491
maximální odchylka	0,5121	0,2901	1,5094	0,3300
medián odchylky	0,1514	0,1375	0,1695	0,1525

Tabulka 7.5: Zhodnocení navržených vztahů

7.3 Testování implementace metody triangulace

Pro otestování samotné implementace algoritmu triangulace jsem vytvořil test ve zdrojovém souboru *test/test_triangulation.cpp*. Test pro mnou zadané vstupní hodnoty dopadl úspěšně.

Důležitější ale je otestovat navržený algoritmus v simulačním či reálném prostředí. Proto jsem prováděl testování v simulátoru Cooja, kdy jsem se náhodně pohyboval s mobilním uzlem v rámci senzorové sítě. V tomto měření prováděl mobilní uzel výpočet své pozice.

referenční senzory pozice [x; y]	**vzdálenost [m]**	**mobilní uzel (skutečná pozice) [x; y]**	**mobilní uzel (vypočtená pozice) [x; y]**
[0,0; 7,0] [0,0; 0,0] [7,0; 3,5]	5,6242 2,7058 5,0411	[2,000; 2,000]	[2,0521; 1,7636]
[7,0; 3,5] [7,0; 0,0] [0,0; 3,5]	3,2901 3,8741 4,4577	[4,047; 2,244]	[3,9183; 2,3477]
[3,5; 7,0] [0,0; 3,5] [7,0; 0,0]	3,2901 5,6242 5,0410	[5,404; 4,470]	[5,5476; 4,4247]
[0,0; 0,0] [3,5; 0,0] [7,0; 7,0]	6,7896 6,7896 4,4577	[2,534; 6,539]	[1,7500; 6,5602]
[7,0; 3,5] [7,0; 0,0] [7,0; 7,0]	5,6242 7,9541 4,4577	[2,534; 6,539]	nevypočteno
[7,0; 3,5] [3,5; 7,0] [7,0; 7,0]	6,7896 3,8741 7,3719	[0,241; 4,826]	[0,3306; 4,7721]
[0,0; 0,0] [3,5; 3,5] [-; -]	3,8741 5,0417 -	[1,977; 0,531]	nevypočteno
[7,0; 3,5] [7,0; 7,0] [3,5; 0,0]	7,3719 9,1179 3,2901	[0,241; 0,998]	[0,0170; 1,1371]
[7,0; 3,5] [0,0; 3,5] [7,0; 0,0]	3,2901 3,8741 5,0411	[3,580; 3,290]	[3,7988; 4,2599]
[3,5; 7,0] [0,0; 3,5] [7,0; 0,0]	2,7058 6,7896 6,7896	[5,827; 6,628]	[6,1387; 6,4009]
[7,0; 3,5] [7,0; 7,0] [3,5; 0,0]	3,2901 1,5362 6,7896	[5,827; 6,628]	[5,5621; 6,4593]

Tabulka 7.6: Výpočet triangulace mobilním uzlem

V tabulce 7.6 si můžeme všimnout, že u dvou případů neproběhl výpočet, což ovšem v uvedených případech není chybou. V prvním případě se referenční senzorové uzly nacházely na přímce, došlo by tedy k výpočtu dvou potencionálních pozic mobilního uzlu. V druhém případě pak mobilnímu uzlu chyběly data pro výpočet triangulace. Po naměření hodnot jsem pak provedl výpočty mediánu, průměrné a maximální odchylky. Výpočty jsou zaznamenány v tabulce 7.7.

	x souřadnice	y souřadnice
průměrná odchylka	0,1433	0,1944
maximální odchylka	0,3117	0,9699
medián	0,1362	0,1214

Tabulka 7.7: Odchylky výpočtu při výpočtu triangulace mobilním uzlem

V dalším testování jsem opět náhodně pohyboval mobilním uzlem, ovšem výpočet pozice mobilního uzlu prováděla senzorová síť.

Vypočet prováděl vždy senzorový uzel s ID 1. Podle výstupů z tabulky 7.8 lze odhadnout, že senzorový uzel s ID 1 se nacházel na pozici [0,0; 0,0], jeho údaje jsou při výpočtu přítomny vždy. Opět můžeme vidět, že u dvou případů nedošlo k výpočtu, což ovšem zase není chybou, neboť se referenční senzorové uzly nacházely na přímce.

referenční senzory pozice [x; y]	**vzdálenost [m]**	**mobilní uzel (skutečná pozice) [x; y]**	**mobilní uzel (vypočtená pozice) [x; y]**
[0,0; 0,0] [7,0; 0,0] [3,5; 3,5]	2,7058 5,6241 5,0411	[2,000; 2,000]	[1,7636; 2,0521]
[0,0; 0,0] [3,5; 7,0] [0,0; 7,0]	5,0411 6,7895 7,9541	[4,715; 0,553]	[5,2064; 0,3843]
[0,0; 0,0] [0,0; 7,0] [7,0; 0,0]	5,0411 6,2069 7,8740	[4,317; 2,645]	[4,3407; 2,5634]
[0,0; 0,0] [0,0; 3,5] [0,0; 7,0]	5,0411 4,4577 6,2069	[4,317; 2,645]	nevypočteno
[0,0; 0,0] [0,0; 3,5] [7,0; 7,0]	9,1179 6,7895 0,9507	[6,339; 6,391]	[5,7929; 7,0412]
[0,0; 0,0] [3,5; 3,5] [7,0; 7,0]	7,9541 3,2901 3,2901	[4,002; 6,517]	nevypočteno
[0,0; 0,0] [3,5; 7,0] [7,0; 7,0]	6,2070 2,7058 3,8740	[4,269; 4,336]	[4,3473; 4,4303]
[0,0; 0,0] [3,5; 0,0] [3,5; 3,5]	6,7895 6,7895 3,2901	[1,577; 6,406]	[1,7502; 6,5601]
[0,0; 0,0] [7,0; 7,0] [3,5; 7,0]	3,8741 6,2069 3,2901	[2,623; 2,687]	[2,0058; 3,1331]
[0,0; 0,0] [3,5; 0,0] [0,0; 7,0]	3,8741 3,2901 5,0410	[2,623; 2,687]	[2,3477; 3,0817]
[0,0; 0,0] [7,0; 0,0] [0,0; 3,5]	0,9508 6,7895 2,7058	[0,553; 0,623]	[0,2719; 0,9111]

Tabulka 7.8: Výpočet triangulace senzorovou sítí

Po naměření hodnot jsem poté opět provedl výpočty mediánu, průměrné a maximální odchylky. Výpočty jsou zaznamenány v tabulce 7.9.

	x souřadnice	y souřadnice
průměrná odchylka	0,3021	0,2504
maximální odchylka	0,6172	0,6502
medián	0,2782	0,1715

Tabulka 7.9: Odchylky výpočtu při výpočtu triangulace senzorovou sítí

7.3.1 Porovnání obou způsobů výpočtu triangulace

Jelikož se mobilní uzel může libovolně pohybovat, předpokládám, že přesnější by měla být varianta, kdy výpočet pozice mobilního uzlu provádí sám mobilní uzel. K tomu předpokladu jsem dospěl tak, že v navrženém řešení má mobilní uzel těsně před zahájením výpočtu pravděpodobně aktuálnější informace. Údaje o pozicích uzlů senzorové sítě se nemění, ovšem údaj o jednotlivých vzdáleností mobilního uzlu od senzorových ano.

V době, kdy mobilní uzel přijme údaj pozice od senzorového uzlu, provede výpočet vzdálenosti. Naopak, pokud provádí výpočet senzorová síť, dochází k výpočtu vzdálenosti v každém uzlu senzorové sítě a údaje jsou pak ještě distribuovány jednomu uzlu, který provádí daný výpočet. Dochází tedy k větší prodlevě, kdy se mobilní uzel může pohnout. Vypočtené vzdálenosti už se tedy můžou lišit od skutečných.

	menší hodnoty dosahuje varianta
průměrná odchylka x	1
průměrná odchylka y	1
max x	1
max y	2
medián x	1
medián y	1

Tabulka 7.10: Porovnání variant výpočtu triangulace

Po porovnání výsledků tabulek 7.7 a 7.9 jsem přišel k závěru, že varianta 1, což je případ, kdy výpočet triangulace provádí sám mobilní uzel, dosahuje přesnějších výsledků. Z tabulky 7.10 lze vyčíst, že varianta 2 dosáhla menší odchylky pouze u maximální hodnoty při výpočtu y souřadnice. Hodnoty maximálních odchylek jsou spíše jen orientační. Důležitější pro srovnání jsou průměrné odchylky nebo medián odchylky, kde varianta 1 dosáhla vždy lepších výsledků.

Kapitola 8

Závěr

Cílem mého semestrálního projektu a diplomové práce bylo nastudovat a popsat problematiku bezdrátových senzorových sítí. Tuto skutečnost jsem popsal v kapitole 2.

Hlavním cílem mé diplomové práce je implementovat metodu pro určení pozice mobilního senzorového uzlu. Proto jsem nastudoval a popsal nejčastější algoritmy, které jsou v této problematice využívány.

Pro řešení daného problému je důležité postup simulovat. K tomu mi posloužilo simulační prostředí Cooja prostředí Contiki. Jedná se o operační systém vhodně navržený k simulaci bezdrátových senzorových sítí. Ve 4. kapitole jsem popsal jeho používání a způsob komunikace mezi jednotlivými zařízeními pomocí protokolu μIP, který jsem při zpracování projektu využíval. Dále jsem navrhl a implementoval způsob komunikace mezi senzorovými uzly pro řešení daného problému, kterým se má diplomová práce zabývá.

V řešení diplomové práce jsem implementovat metodu triangulace za pomocí rádiového spojení a další pomocné výpočetní funkce pro zajištění co nejpřesnějších výsledků. V simulátoru Cooja se ukázala metoda určení vzdálenosti mezi jednotlivými senzory pomocí měření délky doby letu rádiového signálu jako nevhodná. Proto jsem pro určení vzdálenosti využil naměřenou sílu signálu RSSI. Tato varianta je teoreticky méně přesná než měření délky doby letu rádiového signálu, ale použitelná v simulačním prostředí Contiki. Metodou triangulace jsem tedy dokázal určit polohu mobilního senzorového uzlu v dané senzorové síti.

Primárně jsem měl své řešení implementovat pro senzorový uzel MicaZ. V průběhu řešení jsem narazil na problém s komunikací mezi senzorovými uzly a řešení se mi pro MicaZ nepovedlo implementovat. Proto jsem vybral jiný senzorový uzel, který se svými vlastnostmi uzlu MicaZ podobal. Jedná se o senzorový uzel Zolertia Z1.

Na senzorovém uzlu Z1 jsem pak provedl několik simulací a měření, které jsem popsal v kapitole 7. Celkově docházelo k průměrné chybě 0,1388 m při učení vzdálenosti mezi senzorovými uzly a průměrně k odchýlení 0,1433 m v ose x a 0,1944 m v ose y při určení pozice mobilního senzorového uzlu, kdy výpočet pozice prováděl sám mobilní uzel.

Jako rozšíření by bylo možné doplnit novou funkčnost jako je propagace informací od vzdálených senzorových uzlů (mimo rádiový dosah) k senzorům bližším k mobilnímu uzlu. S narůstající vzdáleností by se sice zvětšoval koeficient chyby určení přesné pozice mobilního senzorového uzlu, ovšem bylo by možné určit jeho pozici i bez toho, že se v rádiovém dosahu mobilního senzorového uzlu nacházejí nejméně tři senzorové uzly, které znají svou pozici.

Další možností rozšíření by mohlo být postupná navigace mobilního uzlu. Pokud si představíme senzorovou síť jako mapu, mohl by být mobilní senzorový uzel postupně naváděn

k danému senzorovému uzlu. Orientace v prostoru by mohla být signalizována například pomocí LED diod umístěných na zařízení. Za předpokladu, že by senzorové uzly byly napájeny z elektrické sítě, mohly by svoji pozici inicializovat samy pomocí GPS modulu.

Využití mého projektu by při dalších rozšiřujících implementacích mohlo být i například armádní. V senzorové síti vytvořené pomocí miniaturních senzorových zařízení jako je SmartDust by mohl být navigován robot například do míst, které by mohly být pro člověka životu nebezpečné. Stejně tak v budoucnosti například robot pro hašení lesních požárů může být navigován k senzorovému uzlu, který detekoval problém.

Pro řešení mé diplomové práce jsem využíval prostředí Contiki. Po počátečním nadšení se jsem musel řešit několik problémů jak s překladem zdrojových souborů, kdy problém vznikl kvůli chybě v prostředí Contiki, tak s komunikací mezi uzly. Zpětná kompabilita mezi jednotlivými verzemi Contiki OS taky není úplně dokonalá. Zdrojové soubory vytvořené a přeložitelné na verzi 3.0 (25. 8. 2015) nejsou jednoduše přeložitelné na starší verzi 2.7 (15. 11. 2013). Pro simulátor Cooja pak mezi zmíněnými verzemi dochází k rozdílu, že na starší verzi 2.7 dochází k inicializaci senzorového uzlu okamžitě, kdežto ve verzi 3.0 je první senzorový uzel inicializován až po 2,9 simulačních sekundách. Proč k tomuto dochází, se mi nepovedlo odhalit. Celkově bych tedy prostředí Contiki pro vývoj a simulaci bezdrátových senzorových sítí nedoporučoval.

Literatura

[1] A., D.: *The Contiki Operating System*. 2012, [Online; navštíveno 28.12.2015].
URL http://contiki.sourceforge.net/docs/2.6/

[2] A., D.: *The Open Source OS for the Internet Things*. 2012, [Online; navštíveno 20.12.2015].
URL http://www.contiki-os.org/

[3] Bekcibasi U., T. M.: *Increasing RSSI Localization Accuracy with Distance Reference Anchor in Wireless Sensor Networks*. 2014, [Online; navštíveno 8.4.2016].
URL https://uni-obuda.hu/journal/Bekcibasi_Tenruh_54.pdf

[4] Crossbow: *MicaZ*. San Jose, California, 2011, [Online; navštíveno 3.3.2016].
URL http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[5] E., J.: *The Contiki Operating System*. 2016, [Online; navštíveno 19.3.2016].
URL https://sourceforge.net/p/contiki/mailman/message/32776901/

[6] Foundation, F. S.: *Index of /releases/avr-libc*. 2016, [Online; navštíveno 3.3.2016].
URL http://download.savannah.gnu.org/releases/avr-libc/

[7] Holešinský P., S. M., Komosný D.: *Lokalizační techniky bezdrátových senzorových sítí založené na triangulačním mechanismu*. Brno, 2009, [Online; navštíveno 27.11.2015].
URL http://www.elektrorevue.cz/cz/clanky/komunikacni-technologie/40/lokalizacni-techniky-bezdratovych-senzorovych-siti/

[8] J., P.: *Lokalizace komunikačních uzlů v bezdrátových senzorových sítích*. Brno, 2009, [Online; navštíveno 5.12.2015].
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=15907

[9] Ákos Lédeczi, M. M.: *Wireless sensor node localization*. 2011, [Online; navštíveno 23.11.2015].
URL http://rsta.royalsocietypublishing.org/content/370/1958/85

[10] M., P.: *Bezdrátové senzorové sítě v průmyslové praxi*. Brno, 2012, [Online; navštíveno 27.12.2015].
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=55230

[11] Mikulčák J., C. J. a. d.: *Matematické, fyzikální a chemické tabulky a vzorce*. Prometheus, 2006, ISBN 80-7196-264-3.

[12] P., K.: *Installation*. 2016, [Online; navštíveno 3.3.2016].
URL http://anrg.usc.edu/contiki/index.php/Installation

[13] System, T. C. O.: *Instant Contiki 3.0*. 2015, [Online; navštíveno 18.11.2015].
URL https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/

[14] team, C.: *Get Started with Contiki*. 2011, [Online; navštíveno 20.11.2015].
URL http://www.contiki-os.org/start.html

[15] VMWare: *VMware Workstation 12.1.1 Player for Windows 64-bit operating systems*. 2015, [Online; navštíveno 18.11.2015].
URL https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12_0

[16] Zolertia: *Z1 Features: Quick Hardware Tour*. 2013, [Online; navštíveno 18.4.2015].
URL http://zolertia.sourceforge.net/wiki/index.php/Z1#Ultra-Low_Power_MCU_and_2.4GHz_Transceiver

[17] Zolertia: *Z1 Datasheet*. 2016, [Online; navštíveno 18.4.2015].
URL
http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf

Přílohy

Seznam příloh

Příloha A

Obsah CD

Adresářová struktura přiloženého DVD:

xhyrak00	
název souboru	**popis**
broMicaZ	adresář obsahuje zdrojové soubory implementace broadcast komunikace pro zařízení MicaZ
broMicaZ.csc	šablona rozmístění senzorových uzlů pro simulátor Cooja
Makefile	soubor pro překlad zdrojových souborů
my_broadcast.c	zdrojový soubor implementace broadcast komunikace na zařízení MicaZ v jazyce C
latex	adresář obsahuje zdrojové soubory této diplomové práce v systému LaTeX
mob	adresář obsahuje zdrojové soubory implementace lokalizace mobilního uzlu, kdy výpočet provádí sám mobilní uzel
Makefile	
mob.c	zdrojový soubor implementace životního cyklu mobilního uzlu na zařízení Z1 v jazyce C
$mob9 + 1.csc$	šablona rozmístění senzorových uzlů pro simulátor Cooja
sen.c	zdrojový soubor implementace životního cyklu senzorového uzlu na zařízení Z1 v jazyce C
myH	adresář hlavičkových souborů obsahujících implementaci výpočetních algoritmů
floatToStr.h	zdrojový soubor funkce pro převod reálného čísla na řetězec
myH.h	hlavičkový soubor inkludující ostatní hlavičkové soubory
posCopm.h	zdrojový soubor výpočetních algoritmů triangulace a pomocných funkcí
printFloat.h	zdrojový soubor funkce, která v prostředí Contiki vypisuje reálné číslo
strToFloat.h	zdrojový soubor funkce pro převod řetězce na reálné číslo
pom	adresář obsahující doplňující soubory ke správnému překladu zdrojových souborů v prostředí Contiki-3.0
boot.h	pro překlad na MicaZ zařízení
contiki-conf.h	konfigurační soubor pro zprovoznění komunikace pomocí Rime protokolu na MicaZ zařízení

cooja.desktop	skript pro jednoduché spuštění simulátoru Cooja
sen	adresář obsahuje zdrojové soubory implementace lokalizace mobilního uzlu, kdy výpočet provádí uzel senzorové sítě
Makefile	
mob.c	zdrojový soubor implementace životního cyklu mobilního uzlu na zařízení Z1 v jazyce C
sen.c	zdrojový soubor implementace životního cyklu senzorového uzlu na zařízení Z1 v jazyce C
$sen9+1.csc$	šablona rozmístění senzorových uzlů pro simulátor Cooja
test	adresář obsahující zdrojové soubory pro testování výpočetních algoritmů trialngulace
in	adresář se vstupními soubory pro jednotlivé testy
Makefile	
myH.h	
out	adresář s referenčními výstupními soubory
outMy	adresář obsahující zpočtené výstupy jdenotlivých testů
test.sh	skript pro překlad zdrojových souborů pro testy, spuštění testů a vyhodnocení
test_distance.cpp	zdrojový soubor pro test výpočtu vzdálenosti mezi dvěma uzly v jazyce C++
test_line.cpp	zdrojový soubor pro test vzájemné pozice referenčních uzlů pro výpočet triangulace v jazyce C++
test_triangulation.cpp	zdrojový soubor pro test výpočtu pozice mobilního uzlu pomocí triangulace v jazyce C++
ToF	adresář obsahuje zdrojové soubory implementace měření délky letu rádiového signálu pro zařízení Z1
Makefile	
ToF.c	zdrojový soubor implementace měření délky letu rádiového signálu
$ToF2.csc$	šablona rozmístění senzorových uzlů pro simulátor Cooja
xhyrak00-dp.pdf	text diplomové práce
výpočty.xlsx	zaznamenané hodnoty testů a prováděné výpočty
readme.txt	popis DVD

Příloha B

Manual

Popis spuštění projektu této diplomové práce v prostředí Contiki-3.0[1]:

1. Stáhnout Contiki[2] (1 GB) a rozbalit soubor.
2. Nainstalovat VMWare Player[3]
3. Spustění Contiki OS pomocí VMWare Playeru:
   (a) vmwareplayer.exe
   (b) Open a Virtual Machine > z umístění staženého souboru s Contiki načíst virtuální stroj (soubor s příponou *vmx*).
   (c) Spuštění InstantContiki3.0.
4. Heslo pro přihlášení: „user“.
5. Zkopírovat adresář *xhyrak*00 z přiloženého DVD do adresáře *home*.
6. Spuštění simulátoru Cooja:
   (a) Pomocí skriptu *cooja.desktop*:
      i. Skript *cooja.desktop* můžete umístit například na plochu.
      ii. Je nutné mu přiřadit práva, například pomocí příkazu: *chmod* 755 *cooja.desktop*.
      iii. Spuštěním skriptu *cooja.desktop* se spustí simulátor Cooja.
   (b) Otevření terminálu a zadání posloupnosti příkazů:
      i. *cd contiki/tools/cooja*,
      ii. *ant run*.
7. Nastavení simulace:
   (a) Nová simulace:
      i. *File > New simulation*,
      ii. Nastavaní základních parametrů > *Create*.

[1] Postup jsem nalezl na zdroji uvedeném v seznamu použité literatury [14].
[2] Odkaz na stažení Contiki [13]
[3] Odkaz na stažení VMWare Playeru [15]

iii. V okně *Network* v záložce *View* pak lze zapnout různé vizualizační pomůcky.

iv. Vložení mote:

A. Pomocí záložek *Motes* > *Add motes* > *Create new mote type*"vyberte zvolený typ senzorového uzlu.

B. Zadejte cestu ke zdrojovým souborům, zkompilujte kód a vytvořte uzel pomocí tlačítka *Create*.

C. Zvolte umístnění a počet uzlů.

(b) Otevření vytvořené seimulace:

i. Pomocí záložek *File* > *Open simulation* > *Browse*... vyberte cestu k souboru simulace *csc*.

8. Spuštění samotné simulace pak pomocí tlačítka *Start* v okně *Simulation control*.