VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MONITOROVÁNÍ BUDOVY POMOCÍ BEZDRÁTOVÉ SENZOROVÉ SÍTĚ S ČÁSTEČNĚ DYNAMICKOU TOPOLOGIÍ

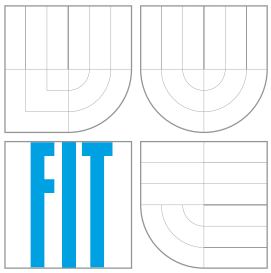
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ URBANOVSKÝ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

MONITOROVÁNÍ BUDOVY POMOCÍ BEZDRÁTOVÉ SENZOROVÉ SÍTĚ S ČÁSTEČNĚ DYNAMICKOU TOPOLOGIÍ

WIRELESS SENSOR NETWORK FOR BUILDING MONITORING WITH PARTIALLY DYNAMIC TOPOLOGY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ URBANOVSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN SAMEK, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Práce popisuje možnosti bezdrátových senzorových sítí při monitorování budovy a shrnuje nejznámější použití bezdrátových senzorových sítí v praxi. Jsou v ní popsány potřebné protokoly pro komunikaci jednotlivých uzlů a vlastnosti topologie vzniklé sítě. Je kladen důraz na implementaci plně dynamické sítě a s tím spojené automatické ustanovení topologie. Aplikace je navržena pro operační systém TinyOS a hardwarové platformy IRIS a MICAz a jejím úkolem je ustavit topologii a zasílat data získaná ze senzorů k základnové stanici. Implementovaný protokol vychází z protokolu Collection Tree Protocol a za použití nepotvrzovaného spojení je schopen ustavit dynamickou topologii. Dále je v práci popsáno uložení a způsob následné grafické i textové reprezentace nasbíraných dat.

Abstract

This thesis describes the possibilities of wireless sensor networks in monitoring buildings and summarizes the most common use of wireless sensor networks in practice. There are also described communication protocols needed for communication between each node and properties of topology of created network. Importance is lay on implementation of fully dynamic network with automatic network topology establishment. The application is designed for the TinyOS operating system and hardware platforms MICAz and IRIS, and its task is to establish the topology and send the data obtained from sensors to base station. The implemented protocol is based on the Collection Tree Protocol protocol and using best effort delivery is able to establish a dynamic topology. Further there is described the method of storage and subsequent graphical and textual representation of data collected.

Klíčová slova

Bezdrátová senzorová síť, TinyOS, TOSSIM, Collection Tree Protocol, Protokol Collection, NesC, MICAz, IRIS.

Keywords

Wireless Sensor Network, TinyOS, TOSSIM, Collection Tree Protocol, Protocol Collection, NesC, MICAz, IRIS.

Citace

Jiří Urbanovský: Monitorování budovy pomocí bezdrátové senzorové sítě s částečně dynamickou topologií, diplomová práce, Brno, FIT VUT v Brně, 2012

Monitorování budovy pomocí bezdrátové senzorové sítě s částečně dynamickou topologií

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Samka, Ph.D. Uvedl jsem všechny prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Urbanovský
21. května 2012

Poděkování

Děkuji panu Ing. Janu Samkovi, Ph.D. za vedení práce a za trpělivost, kterou se mnou měl. Dále bych chtěl poděkovat panu Ing. Janu Horáčkovi za pomoc s řešením problémů se systémem TinyOS a simulačním nástrojem TOSSIM.

Obsah

Kapitola 1

Úvod

Monitorování budov není v dnešní době doménou pouze velkých společností, proniklo již i do domácností. Abychom byli schopni budovu monitorovat, je třeba po ní rozmístit senzory snímající požadovanou veličinu a získaná data následně přenést na stanici, která je vyhodnotí, a na základě pravidel vykoná požadované akce. Čím dál tím častěji se objevují zmínky o inteligentních budovách a jejich stavbách na klíč. V takové stavbě se již s monitorovací technologií počítá a na základě toho jsou vybudovány plány, které zahrnují umístění senzorů a případných aktuátorů. Člověk žijící v "obyčejném" domě může s cílem zinteligentnění svého bydliště buď naplánovat rekonstrukci a vybudovat požadovanou datovou infrastrukturu, nebo pro přenosy dat ze senzorů použít bezdrátovou technologii.

Přenos dat vzduchem je žádaný, protože odpadá nutnost vytváření fyzické sítě. Pokud nemáme požadavky na vysokou rychlost přenosu, je bezdrátová volba řešení, ke kterému se přiklání čím dál více lidí. Práce pojednává právě o bezdrátových senzorových sítích a způsobech jejich použití v praxi.

Ve druhé kapitole se seznámíme s operačním systémem TinyOS a se základy programovacího jazyka nesC, ve kterém je implementován kód pro jednotlivé uzly.

Ve třetí kapitole jsou popsány principy bezdrátových senzorových sítí. Dále budou přiblíženy uzly IRIS a MICAz, pro které bude práce implementována. Třetí kapitola také přibližuje obecné fungování protokolů pro bezdrátové senzorové sítě.

Čtvrtá kapitola popisuje návrh topologie, do které budou uzly uspořádány a diskutuje výhody a nevýhody statického a dynamického směrování. Dále detailněji popisuje navržený komunikační protokol, jeho funkčnost a srovnává jej s již existujícími řešeními.

Pátá kapitola popisuje fungování programu pro senzorové uzly a základnovou stanici. Obsahuje také informace o práci s pakety a princip čtení dat ze senzorů. Nalezneme zde také bližší popis komponent systému TinyOS, které byly v práci použity.

Kapitola šestá popisuje zpracování dat získaných ze senzorů. Je zde popsán způsob ukládání dat a metody jejich grafické a textové reprezentace.

Výsledky testování sítě jsou popsány v kapitole sedmé. Testovala se schopnost uzlů dynamicky reagovat na výpadky v sítí, dosah jednotlivých zařízení v různých podmínkách a výdrž baterií v jednotlivých uzlech. Dále zde jsou interpretovány výsledky testovacích měření.

Kapitola 2

TinyOS a programovací jazyk nesC

Jazyk nesC je rozšíření jazyka C navržené pro strukturovaný koncept návrhu aplikací pro operační systém TinyOS. TinyOS je událostmi řízený operační systém navržený pro použití v bezdrátových senzorových sítích s velice omezenými výpočetními zdroji. První implementace TinyOS z roku 2000 byla naprogramována kombinací jazyka C a skriptů jazyka Perl. V roce 2002 by TinyOS implementován znovu, tentokrát již v jazyce nesC .

2.1 TinyOS

Operační systém TinyOS [13] je používán vývojáři po celém světě k vývoji bezdrátových senzorových sítí. Jeho zaměření na komunikaci a velká modularita je ideální pro použití v těchto sítích, kde je třeba sbírat informace ze vzdálených míst. V počátcích své existence se TinyOS používal pouze ve výzkumných centrech univerzit, později se propracoval i do komerční sféry.

Operační systém tvoří pro programátora abstrakci pro komunikaci s hardwarem, spravuje přístup k paměti, umožňuje běh více vláken, ovládá síťová rozhraní a periferie. Oddělení operačního systému od aplikací je typický přistup pro klasické počítače, ale méně obvyklé ve světě vestavěných zařízení.

Prostředí TinyOS poskytuje nástroje pro vývoj na senzorových uzlech (viz např. zmíněné uzly IRIS a MICAz v sekci 3.4). Pro vývoj se používá programovací jazyk NesC, který je popsán v sekci 2.2. Součástí operačního systému je také možnost simulovat senzorovou síť pomocí nástroje TOSSIM [8].

2.1.1 TOSSIM

TOSSIM je diskrétní simulátor pro senzorové sítě implementované pro systém TinyOS. Umožňuje tak uživateli testování vytvořené aplikace bez nutnosti nahrávat kód na jednotlivé uzly, což poskytuje značné urychlení vývoje. TOSSIM se snaží o velice věrné napodobení reality a simuluje síť na úrovni bitových operací.

Měření času je trochu odlišné od reality. TOSSIM sice přesně spustí požadované přerušení případně událost ale následné provedení proběhne okamžitě. Čas pak běží s přesností 4MHz, což je frekvence procesorů platforem `mica` a `rene`.

TOSSIM je vhodný zejména pro testování síťových řešení. Je schopen simulovat uzly bezdrátové senzorové sítě včetně síly signálu od jednotlivých uzlů.

I když je simulátor schopen poměrně věrně zachytit chování systému TinyOS, nemůžeme jej brát jako spolehlivou metriku pro fungování kódu. Musíme počítat s tím, že aplikace,

která v simulaci funguje bez problémů, může v reálném systému vykazovat známky chybovosti. Stejně tak porovnávání výsledků simulací různých řešení za účelem výběru nejlepšího, není doporučovaný postup – v reálném prostředí se výsledky mohou zásadním způsobem lišit.

Rozhraní pro Python

Jelikož je TOSSIM distribuován jako knihovna, je třeba implementovat program, který simulaci nakonfiguruje a spustí. K dispozici jsou rozhraní pro dva jazyky – Python a C++. V práci bylo použito rozhraní pro jazyk Python.

Komunikace s uzly může probíhat dvěma způsoby. První je implementace informací pro simulátor přímo v kódu uzlu. Použije se ladicí (debug) výstup systému ve formátu:

```
dbg("Channel", "content");
```

První parametr funkce `dbg(a,b)` definuje výstupní kanál, druhý parametr pak řetězec, který bude simulátorem vypsán. Výstupní kanál slouží k odlišení různého typu zpráv – můžeme vypsat zprávy, které souvisí pouze s rádiovým modulem nebo například zprávy zpracovávající data ze senzorů. Pojmenování kanálů je čistě věcí programátora, celá aplikace může mít pouze jeden kanál. Druhý parametr může být parametrizován podobně jako `sprintf` v C++. Následuje příklad parametrizovaného výpisu použitého v práci.

```
dbg("Timer","rdy, info=%hhu, thl=%hhu,etx=%hhu, origin=%hhu seqno=%hhu\n",
cPkt->info,cPkt->thl,cPkt->etx,cPkt->origin,seqCnt);
```

%hhu značí osmibitovou hodnotu bez znaménka, %s lze použít pro zastoupení řetězce. Druhý způsob komunikace s uzly je přímý přístup k obsahu proměnných. Je možné číst pouze základní datové typy; pole ani struktury číst nemůžeme. Hodnotu získanou přímým přístupem můžeme vypsat pomocí standardních prostředků daného jazyka. Nevýhodou je, že nevíme, v jakém stavu se uzel v době čtení hodnoty nachází a nedisponujeme prostředky, abychom tuto informaci byli schopni zjistit.

Příklad skriptu, který implementuje simulaci nalezneme v příloze D.1, její výstup pak v příloze D.2.

2.2 Základní koncepty nesC

Pro nesC je typické oddělení implementační části od části logiky – aplikace se skládají z komponent, které jsou navzájem pospojovány, čímž vznikne výsledný program. Chování komponenty je dáno množinou rozhraní (interfaces), ke kterým je komponenta připojena. Rozhraní mohou být danou komponentou poskytována (klíčové slovo `provides`) nebo používána (klíčové slovo `uses`). Poskytované rozhraní slouží jako výstupní port, ke kterému se v části skládání komponent může připojit port vstupní, tj. část, která naopak potřebuje data získat.

2.3 Rozhraní

Rozhraní v nesC jsou dvojsměrné a popisují komunikační kanál mezi dvěma komponentami. Jedna z komponent zpravidla poskytuje data, která druhá komponenta používá. Pro jednoduchost je nyní označme jako `poskytovatel` a `příjemce`. Rozhraní jsou rozdělena

na dvě skupiny – příkazy a události. Příkazy (commands) můžeme přirovnat k funkcím v jazyce C a v dalším textu je tak budeme i označovat. Jedná se o množinu funkcí, které jsou implementovány na straně poskytovatele. Události (events) lze připodobnit ke callback funkcím v jazyce C. Tyto naopak musí být implementovány na straně příjemce. Jednoduché rozhraní může vypadat následovně:

```
interface SendMsg {
    command result_t send(uint16_t addr, uint8_t len, message_t* msg);
    event result_t sendDone(message_t* msg, result_t err);
}
```

Poskytovatel tedy musí implementovat příkaz `send` zatímco příjemce musí implementovat reakci na událost `sendDone`.

2.4 Specifikace komponent

Komponenta v nesC je buď `modul` (module) nebo `konfigurace` (configuration). Obsahuje seznam specifikací jednotlivých prvků, ze kterých se skládá. Může se jednat o instance rozhraní, příkazy nebo události používané nebo poskytované danou komponentou. Komponenta může poskytovat a používat více rozhraní. Obě následující definice jsou syntakticky korektní.

```
module A1 {
    uses interface X;
    uses interface Y;
    provides interface Z;
} ...

module A1 {
    uses {
        interface X;
        interface Y;
    }
    provides{
        interface Z;
    }
} ...
```

Abychom byli zcela striktní, museli bychom u každé instance určit její jméno klíčovým slovem `as`:

```
module A1{
    uses interface X as naseRozhrani;
}
```

První definice `interface X` je tedy jen zkrácená verze pro `interface X as X`. Následuje kompletní příklad reálné komponenty:

```
configuration GenericComm {
    provides {
        interface StdControl as Control;
        interface SendVarLenPacket;

        interface SendMsg[uint8_t id];
        interface ReceiveMsg[uint8_t id];
    }
    uses {
        event result_t sendDone();
    }
} ...
```

V tomto příkladu `GenericComm`:

- Poskytuje jednoduché rozhraní `Control` typu `StdControl`.
- Poskytuje jednoduché rozhraní `SendVarLenPacket` typu `SendVarLenPacket`.
- Poskytuje parametrizované instance rozhraní `SendMsg` a `ReceiveMsg`, jsou pojmenovány `SendMsg` a `ReceiveMsg`.

Parametrizované rozhraní odpovídá několikanásobnému rozhraní, pro každou hodnotu parametru. Například rozhraní `interface SendMsg[uint8_t id]` reprezentuje 256 rozhraní typu `SendMsg`.

2.4.1 Moduly

Modul implementuje chování komponenty. Můžeme použít deklarace a definici známé z jazyka C, definovat nový proces a implementovat příkazy a události. Musíme implementovat všechny poskytované rozhraní a všechny události použité rozhraním. Modul může volat všechny své příkazy, signály i události. Syntax tvorby funkčních bloků je následující:

```
specifikace třídy: jedna z následujících
    command  event  async

možnost klíčového slova default

spojování identifikátorů pomocí operátoru . (tečka)
```

Implementace neparametrizovaného příkazu nebo události má stejnou syntax jako v jazyce C. Jediný rozdíl je v rozšíření o tečku v názvu funkce a nutností definovat třídou funkce. Ukázka modulu, který poskytuje rozhraní `Send`:

```
command result_t Send.send(uint16_t addr, uint8_t len, message_t* msg) {
    ... do something here ...
    return SUCCESS;
}
```

Parametrizovaný příkaz (událost) bychom vytvořili obdobně přidáním požadovaného parametru před výčet argumentů.

Události a příkazy můžeme vyvolat příkazy `call` a `signal`. `call` slouží k volání příkazů a `signal` k volání událostí. Volání příkazu Send.send bude vypadat následovně:

```
call Send.send(1, sizeof(Message_type), &msg);
```

Vykonání příkazů a událostí je okamžité, chovaní `call` a `signal` je tedy shodné s voláním funkcí. Modul může mít specifikovánu výchozí implementaci. Ta bude vykonána ve chvíli, kdy příkaz nebo událost nebudou připojeny k žádnému rozhraní. Pro komponentu `Send` vypadá výchozí rozhraní následovně:

```
default command result_t
Send.send(uint16_t addr, uint8_t len, message_t* msg) {
    return SUCCESS;
}
```

Abychom doplnili výčet možností jazyka nesC, zmíníme možnost vytvořit kus nezávislého kódu. Definuje se klíčovým slovem `task` a takto vytvořený kus kódu můžeme spustit voláním `post`. Tím spustíme nový proces.

2.4.2 Konfigurace

Konfigurace implementace se skládá z výběru běžných komponent nebo instanciací generických komponent a následného propojení komponent mezi sebou. Pro každou běžnou komponentu `X` bude vytvořena instance a reference na ni ze všech ostatních konfigurací bude stejná. Komponenta LedsC v příkladu níže bude pro všechny konfigurace daného programu odkazovat na led diody, které jsou na desce. Generickou komponentu `Y` vytvoříme instanciací rodičovské komponenty a získáme tak komponentu unikátní v rámci jedné konfigurace. Například tak umožníme komponentám disponovat větším množstvím nezávislých časovačů. Příklad vytvoření jedné běžné komponenty LedsC a dvou generických komponent s rodičovskou třídou TimerMilliC.

```
components LedsC;

components new TimerMilliC() as TimerRead;
components new TimerMilliC() as TimerSend;
```

Po vytvoření instancí potřebných komponent, je třeba komponenty spojit. K tomuto účelu slouží v nesC tyto tři příkazy:

- `A = B`: Alespoň jedna komponenta obsahuje externí specifikaci prvku. Externí specifikace jsou platformě nezávislé, například specifikace struktury paketu. Aby nedošlo k chybě, musí být splněna jedna z následujících podmínek:
  - `A` je interní, `B` je externí a `A` i `B` rozhraní buď poskytují nebo používají.
  - `A` i `B` jsou externí, jedna rozhraní poskytuje a druhá používá.
- `A -> B`: Spojení mezi dvěma interními elementy. Spojení vždy připojuje element popsaný v `A`, který komponentu používá ke komponentě `B`, který ji poskytuje.
- `A <- B`: je ekvivalentní k `B -> A`.

V následujícím příkladu vidíme definici a propojení některých komponent použitých v práci:

```
configuration moteCommunicationAppC {
}

implementation {
    components moteCommunicationC as App;
    components new TimerMilliC() as TimerSend;
    components new TimerMilliC() as TimerBeacon;
    components new AMSenderC(0x71);
    components new Taos2550C() as SensorLight;

    App.Packet -> AMSenderC;
    App.AMPacket -> AMSenderC;
    App.AMSend -> AMSenderC;
    App.TimerSend -> TimerSend;
    App.TimerBeacon -> TimerBeacon;
    App.ReadVisibleLight -> SensorLight.VisibleLight;
    App.ReadInfraLight -> SensorLight.InfraredLight;
}
```

Komponenta `moteCommunicationAppC` byla pro přehlednost přejmenována na `App`. Vidíme vytvoření dvou instancí časovače, ten připojíme na příslušné komponenty v aplikaci `App`. Komponenta `AMSenderC` poskytuje rozhraní `Packet`, `AMPacket` a `AMSend`. Ty jsou připojeny k adekvátním rozhraním v aplikaci. Všimněme si konvence, která šetří zdrojový kód na příkladu `App.Packet -> AMSenderC;`. Příkaz je totiž ekvivalentní k propojení `App.Packet -> AMSenderC.Packet;`. Propojení `App.ReadInfraLight -> SensorLight.InfraredLight;` zajistí, že pokud v aplikaci budeme volat funkce a události rozhraní `ReadInfraLight` dostaneme výsledek volání funkcí a událostí rozhraní `InfraredLight` komponenty `SensorLight`, která reprezentuje komponentu `Taos2550C` .

Kapitola 3

Bezdrátové senzorové sítě

Bezdrátová senzorová síť se skládá z autonomních senzorů rozmístěných v prostoru. Senzory monitorují fyzikální veličiny nebo podmínky prostředí jako například teplotu, hladinu hluku, vibrace, tlak nebo pohyb a spolupracují při přeposílání výsledků do místa, kde chceme data vyhodnocovat. Jedná se zpravidla o další uzel připojený k počítači. Dále jej budeme nazývat `základnová stanice`. Komunikace se senzory může být jednosměrná nebo dvousměrná. Při jednosměrné komunikace jsme schopni pouze přijímat data od uzlů, dvojsměrná komunikace (implementována v moderních systémech) nám kromě sběru dat umožňuje i komunikaci se senzorem a tím nám poskytuje určitou míru kontroly. Vývoj bezdrátových senzorových sítí byl zahájen pro potřeby armády v 50. letech minulého století. Jako příklad lze uvést projekt DARPY [1] Smartdust, který si klade za cíl monitorovat bojovou zónu. V dnešní době jsou tyto sítě používány v průmyslové i osobní sféře. Informace o bezdrátových sítích byly čerpány z [9] a [10].

Bezdrátová senzorová síť může být složena až z několika tisícovek uzlů a každý uzel je připojen k jednomu nebo více senzorům. Každý uzel by měl mít následující části:

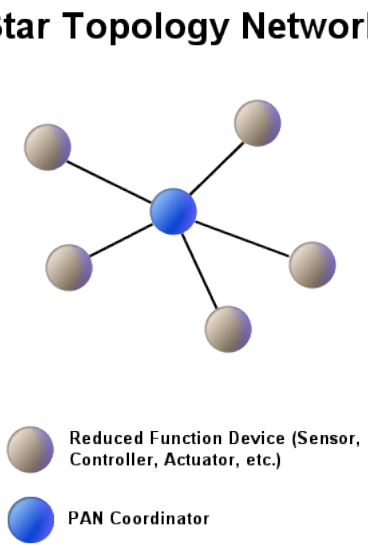
- Rádiový přijímač/vysílač s vestavěnou nebo externí anténou.
- Mikrokontrolér.
- Rozhraní pro komunikaci se senzorem.
- Zdroj energie (baterie, solární).

Cena uzlu se pohybuje v jednotkách až stovkách dolarů v závislosti na hardwarové konfiguraci.

3.1 ZigBee

S bezdrátovými technologiemi se setkáváme všude. Mezi nejznámější technologie, které jsou povědomé i neodborné veřejnosti, patří bezesporu WiFi a Bluetooth. Tyto standardy však nejsou pro použití v senzorových sítích vhodné. Proto byl navržen standard ZigBee ([1],[4]), který vyplnil díru na trhu, která nebyla pokryta ostatními standardy. Vetšina bezdrátových standardů má snahu stále zrychlovat přenos dat a přidávat do protokolu nové služby. ZigBee se zaměřuje na přenosy malého množství informace a protokol je stále stejně jednoduchý, aby

[1] DARPA – The Defense Advanced Research Projects Agency. Má za úkol udržet technologický náskok americké armády před zbytkem světa.



Obrázek 3.1: Síť typu star [11].

mohl fungovat na osmibitovém mikrokontroléru. Dalším důležitým rozdílem je schopnost ZigBee pracovat pouze z energie poskytnuté baterií po dobu řádově let, přičemž ostatní technologie jsou schopny pracovat bez připojení do sítě v lepším případě dny.

ZigBee však není jediný protokol pro rádiové moduly specifikace IEEE 802.15.4. Můžeme zmínit např. protokoly `SimpliciTI`, `Synkro` nebo `MiWi`. V práci se budeme zabývat protokoly operačního systému TinyOS [13], které umí s bezdrátovými rádiovými moduly IEEE 802.15.4 pracovat a narozdíl od ZigBee jsou open-source.

Při vytváření senzorových sítí můžeme pozorovat několik rysů typických právě pro ZigBee. Základní topologie senzorových sítí jsou `star` (viz obrázek 3.1), `mesh` (viz obrázek 3.2) a `cluster tree` (viz obrázek 3.3).

3.2 Použití bezdrátových senzorových sítí v praxi

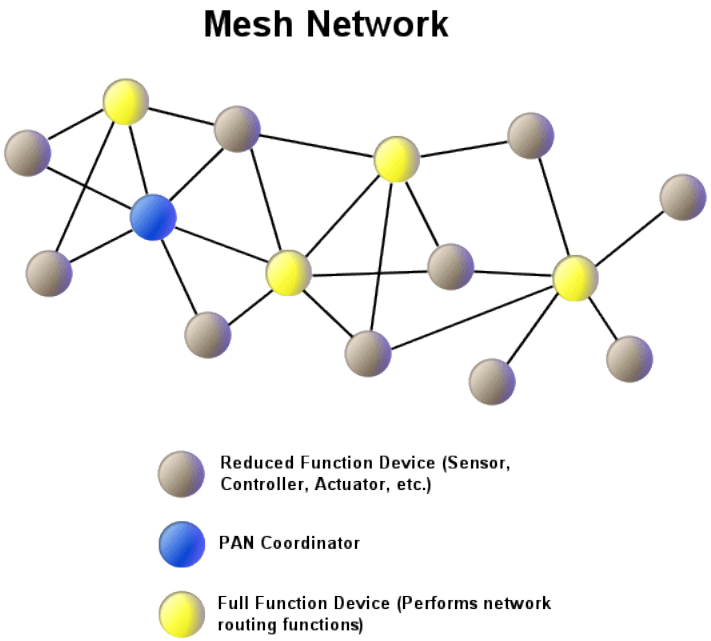
3.2.1 Monitorování oblasti

Monitorování oblasti je velice často spojeno se senzorovou sítí. Senzory jsou rozmístěny v určitém okruhu, kde chceme sledovat danou veličinu. Armáda využívá senzory například pro detekci nepřátel, průmyslová sféra k monitorování plynovodů.

Pokud senzor zachytí určitou hodnotu veličiny, která je sledována (teplota, tlak), ohlásí tuto událost základnové stanici, která vykoná adekvátní reakce.

3.2.2 Monitorování životního prostředí

Monitorování životního prostředí senzorovou sítí se používá převážně k vědeckým účelům. Patří sem například i monitorování sopečných vulkánů, oceánu, ledovců nebo lesů. Některé z hlavních oblastí jsou uvedeny níže.



Obrázek 3.2: Síť typu mesh [11].

Znečištění ovzduší

Bezdrátové senzorové sítě byly rozmístěny v několika městech (Stockholm, Londýn), aby monitorovaly koncentraci jedovatých plynů v ovzduší. Bezdrátová implementace umožňuje přemístění uzlů na různá místa popřípadě jednoduché zvětšení sítě o uzly další.

Detekce požárů

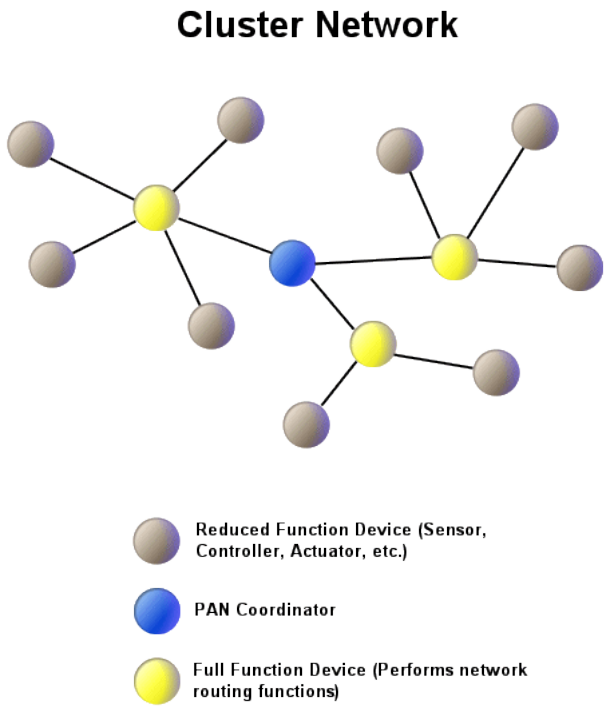
Uzly mohou být rozmístěny v lese a na základě senzorů snímajících teplotu, vlhkost a koncentraci plynů, které se uvolňují při požáru, jsou schopny rychle rozpoznat, kdy a kde vznikl požár.

3.2.3 Monitorování průmyslových strojů

Beztrátové senzorové sítě jsou instalovány do továren, aby hlídaly stav strojů. Jelikož odpadá nutnost použití kabelu, je bezdrátové řešení levnější a je možné použít hustější síť senzorů. Bezdrátový senzor je také schopen jednoduše monitorovat otáčející se součástky a špatně přístupné nebo nebezpečné oblasti výroby.

3.3 Charakteristické vlastnosti bezdrátové senzorové sítě

- Spotřeba energie uzpůsobena tomu, že je uzel napájen z baterií případně alternativně (solární energie).
- Schopnost vypořádat se s výpadkem jednoho či více uzlů.
- Mobilita uzlů.



Obrázek 3.3: Síť typu cluster tree [11].

- Dynamická topologie sítě.
- Výpadky spojení jsou běžné.
- Schopnost vypořádat se s nepříznivými vlivy okolí.
- Jednoduché použití.

Podrobnější seznámení s hardwarem použitým při implementaci práce nalezneme v sekci 3.4.

3.4 Platformy IRIS a MICAz

Společnost MEMSIC začala jako výrobce MEMS [12] komponent pro široké spektrum průmyslových výrobků. V roce 2010 rozšířila koupí společnosti Crossbow své pole působnosti do odvětví bezdrátových senzorových sítí. V práci byly použity uzly IRIS [19] a MICAz [20] (zakoupeny před rokem 2010 ještě pod hlavičkou Crossbow), které popíšeme podrobněji v následujících odstavcích. Dále jsou pro trh k dispozici např. uzly MICA2, Imote2, TelosB.

Senzorové uzly IRIS a MICAz jsou technologicky velice podobné. Oba disponují 2.4 GHz IEEE 802.15.4 bezdrátovým rádiovým modulem a jsou navrženy pro použití ve vestavěných senzorových sítích. Jsou schopny přenášet data rychlostí až 250kbps a každý uzel je schopen data směrovat. Uzly můžeme pomocí standardního 51 pinového konektoru rozšířit o senzorové desky snímající např. světlo, teplotu, tlak nebo zrychlení. Uplatnění naleznou hlavně při monitorování budov a vytváření bezdrátových senzorových sítí s malou spotřebou energie. Senzorové sítě mohou být i velmi rozsáhlé; lze zapojit řádově tisícovky uzlů.

	IRIS	MICAz
Procesor	ATmega1281, 8MHz	ATmega128L, 8MHz
RAM	8KB	4KB
Sériové rozhraní	UART	UART
AD převodník	10 bit	10 bit
Spotřeba procesoru	8 mA (aktivní režim) 8μA (režim spánku)	8 mA (aktivní režim) <15μA (režim spánku)
Frekvence rádiového modulu	2405 – 2480 MHz	2400 – 2483,5 MHz
Maximální přenosová rychlost	250kbps	250kbps
Dosah venku (m)	>300	75-100
Dosah uvnitř (m)	>50	20-30
Rozměry (mm)	58x32x7	58x32x7
Hmotnost (g)	18	18
Rozšiřující rozhraní	51 pinové	51 pinové

Tabulka 3.1: Srovnání platforem MICAz a IRIS dle informací obsažených v [19] a [20].

Uzly IRIS jsou nejnovějším produktem a poskytují navíc některé vylepšení, kterými platforma MICAz, ani její předchůdci, nedisponují. V porovnání s MICAz nabídne IRIS až trojnásobný dosah rádiového modulu (venkovní dosah udávaný výrobcem až 500 metrů bez použití zesilovače) a dvojnásobnou paměťovou kapacitu (8KB RAM). Uzly IRIS jsou osazeny procesorem Atmel ATmega1281 a deskou XM2110CZ a uzly MICAz procesorem Atmel ATmega128L a deskou MPR2400. Oba procesory pracují na frekvenci 8MHz a programují se přes flash paměť. Obě desky jsou schopny paralelního zpracování informací ze senzorů a síťové komunikace. Detailnější porovnání platforem MICAz a IRIS je uvedeno v tabulce 3.1.

3.5 Komunikační protokoly

V následující sekci budou popsány obecné principy, které by měly splňovat komunikační protokoly pro senzorové sítě dle informací v [9] a [10]. Senzorové sítě se skládají z velkého množství uzlů různých typů. Uzly mohou být použity na nepřetržité sledování, registrování a rozpoznání události, zjišťování polohy nebo na ovládání aktuátorů. Dříve než blíže přiblížíme principy komunikace v senzorové sítí, ukážeme si rozdíl mezi senzorovou sítí a ad-hoc[2] sítěmi. Bylo by totiž nasnadě použít již vytvořené řešení. Zjistilo se však, že velké množství ad-hoc protokolů není vhodné pro nasazení v senzorové síti. Pro ilustraci následuje seznam rozdílů mezi senzorovými a ad-hoc sítěmi.

- Počet uzlů senzorové sítě může být řádově vetší než u ad-hoc sítě.
- Uzly senzorové sítě jsou rozmístěny s větší hustotou.
- Senzorové uzly jsou náchylnější k selhání.
- Topologie senzorové sítě se velmi často mění.
- Hlavní komunikační nástroj senzorových uzlů je broadcast, u ad-hoc sítí se používá směrovaná komunikace.

[2]Dočasné spojení mezi dvěma prvky.

- Uzly jsou limitovány životností baterie, výpočetním výkonem a kapacitou paměti.
- Senzorové uzly nemusí mít, z důvodů velkého počtů uzlů a následné velké režie, globální identifikátor.
- Senzorové sítě jsou budovány za účelem měření určité veličiny, ad-hoc sítě slouží většinou pouze k datové komunikaci.

Výzkumy dokazují, že bezdrátové senzorové sítě mají veliký potenciál a přitahují stále větší pozornost odborné veřejnosti. V následující části textu se zaměříme na základy tvorby protokolů pro tyto sítě. Budou popsány myšlenky a principy, které je třeba při tvorbě protokolů aplikační, transportní, síťové, linkové a fyzické vrstvy brát na zřetel.

3.5.1 Aplikační vrstva

Představíme si dva protokoly aplikační vrstvy – `Sensor Management Protocol` a `Sensor Query and Data Dissemination Protocol`. Tyto protokoly mohou vyžadovat spolupráci protokolů nižších vrstev. Některé budou zmíněny dále.

Sensor Management Protocol (SMP)

Jelikož jsou senzorové sítě užívány v širokém spektru aplikací a je žádoucí aby byly přístupné skrz sítě jako je internet, je třeba mít protokol, který bude vytvářet abstrakci nad nižšími vrstvami hardwaru a softwaru a dovolí aplikacím jednoduchou komunikaci. Tu poskytne správci sítě protokol SMP.

Narozdíl od jiných sítí, senzorová síť je složena z uzlů, které nemusí mít vždy globální identifikátor a síť zpravidla postrádá pevně definovanou strukturu. Proto SMP při přistupování k uzlům používá pojmenování založené na atributech a adresování založené na pozici. SMP nabízí následující nástroje pro správu:

- Pojmenování uzlů dle atributů, shlukování uzlů.
- Výměna dat v rámci algoritmu hledajícího pozici uzlu.
- Časová synchronizace uzlů.
- Přesun senzorového uzlu.
- Vypnutí a zapnutí uzlu.
- Autentizace a zabezpečení komunikace.

Sensor Query and Data Dissemination Protocol (SQDDP)

SQDDP poskytuje uživateli rozhraní pro vytvoření dotazu, pro odpověď na dotaz a sběr příchozích odpovědí. Jak bylo avizováno výše, dotazy nejsou směřovány na konkrétní uzly, většinou je použito adresování na základě atributu (“kde je uzel, který snímá teplotu větší než 70° C”) nebo lokace (“vypiš teploty naměřené uzly v sektoru A”).

3.5.2 Transportní vrstva

Bezdrátová senzorová síť je událostmi řízený model, který závisí na schopnosti uzlů sbírat a posílat data. Uzly spolu spolupracují což přináší vyšší přesnost a schopnost pokrýt velkou plochu. Realizace zmíněných akcí ovšem závisí na spolehlivé komunikaci mezi jednotlivými prvky, tj. mezi uzly a základnovou stanicí. Abychom toho docílili, musíme implementovat spolehlivý přenosový mechanismus.

Hlavními úkoly transportní vrstvy jsou:

1. Spojení mezi aplikační a síťovou vrstvou multipexováním a demultiplexováním.
2. Umožnit doručení zpráv mezi uzlem a základnovou stanicí s použitím kontroly chyb aplikované dle požadavků protokolu aplikační vrstvy.
3. Regulovat množství dat zasílaných do sítě kontrolou datového toku a přetížení linky.

Naším cílem je zajistit plnění těchto úkolu, ale musíme vzít v potaz charakter zařízení, pro která budeme vyvíjet. Uzly musí počítat například s malými energetickými výdaji a limitovaným výpočetním výkonem. Například mechanismus `Sliding window` užívaný v protokolu TCP nemusí být ideální řešení a vyústí v plýtvání pásmem. Následuje výčet vlastností, které by měl každý protokol nabízet.

Spolehlivý přenos dat

Na základě požadavků aplikace se snažíme aby byl přenos dat ze senzorů do základnové stanice spolehlivý. Stejně tak přenos například konfiguračních zpráv směrek k senzorům musí být bez chyb, abychom zajistili správné fungování sítě.

Kontrola přetížení linky

Paket ztracený kvůli přetíženému pásmu může ovlivnit správný výsledek na základnové stanici, i když senzor odeslal dostatečné množství dat. Proto je kontrola přetížení nutná pro spolehlivou funkčnost. Zajišťuje také větší efektivitu sítě a pomáhá šetřit omezenými zdroji uzlu.

Dynamičnost sítě

Musíme počítat s tím, že topologie sítě se bude měnit. To nastane při přesunutí uzlu na jinou pozici, při poruše zařízení, případně dočasným vypnutím uzlu.

Energetická úspora

Protokol transportní vrstvy by si měl být vědom skutečnosti, že veškeré akce musí vykonat s co nejmenšími energetickými výdeji. Například pokud základnová stanice zjistí, že od uzlu dostává více dat než potřebuje, redukuje počet zasílaných informací nebo zařízení dočasně vypne. Důvodem může být volba vyšší zasílací frekvence, jelikož se počítalo se špatnou kvalitou přenosového pásma.

Nevyrovnaná implementace

Algoritmus musí být navržen tak, aby běžel převážně na základnové stanici, která je napájená ze sítě a ušetřil tak omezené zdroje senzoru.

3.5.3 Síťová vrstva

Síťová vrstva se stará o směrování v senzorové síti. Vzhledem k tomu, že uzly mohou být fyzicky velice blízko u sebe, je třeba této skutečnosti využít. Můžeme upravit (snížit) vysílací výkon jednotlivých uzlů a tím zajistit menší spotřebu energie a s tím spojenou vyšší výdrž na baterii.

Nyní stručné popíšeme některé z možných metrik při hledání ideální cesty.

Maximum dostupné energie

Preferujeme cestu, jejíž uzly mají k dispozici nejvíce energie. Celkovou energii spočítáme jako součet energií všech uzlů na cestě. Musíme si dát pozor, aby vyšší dostupná energie nebyla zapříčiněna velkým počtem nepotřebných uzlů v cestě.

Minimální energie

Zvolíme cestu, která při přenosu dat ze senzoru k základnové stanici spotřebuje nejméně energie.

Minimální počet hopů

Je zvolena cesta, která projde nejmenším počtem uzlů.

Maximum z minim dostupných energií

Je preferována cesta, jejíž minimální dostupná energie je vyšší než minimální dostupné energie ostatních cest. Můžeme tím prodloužit životnost frekventovaně využívaných uzlů, kterým již nezbývá velké množství energie.

3.5.4 Linková vrstva

Linková vrstva se stará o multiplexování dat, detekci datových rámců, kontrolu přístupu k médiu a správu chyb. V následující části bude přiblížena kontrola přístupu k médiu a správa chyb.

Správá přístupu k médiu

V bezdrátových multi-hop[3] sítích, které jsou schopny samy ustavit topologii, se snažíme o dvě věci. Prvně chceme ustanovit komunikační spojení, které nám umožní vytvořit infrastrukturu, kde bude fungovat multi-hop komunikace. Díky té je pak síť schopna samouspořádání. Druhou věcí, kterou je třeba kontrolovat, je rovnoměrné rozdělení komunikačních prostředků mezi jednotlivé uzly.

Kontrola chyb

Vedle správy přístupu k médiu je kontrola chyb u přenášených dat další důležitou funkcí linkové vrstvy. Kontrolní mechanismy mohou být popsány dvěma přístupy – Forward Error Correction (FEC) a Automatic Repeat reQuest (ARQ). Základní myšlenka FEC je zavedení

[3] Dva uzly nekomunikují přímo, ale skrz jiné uzly v síti.

redundace do dat kódováním zprávy některým z lineárních kódů[4]. ARQ pak spoléhá na znovuzaslání ztracených dat.

3.5.5 Fyzická vrstva

Fyzická vrstva převádí data na signály, které jsou pro komunikaci v daném kanálu vhodné. Probíhá zde volba správné frekvence, volba frekvence nosné, detekce signálu, modulace a šifrování dat.

3.6 Protokol Collection

Sběr dat a jejich následná distribuce k základnové stanici je běžným požadavkem kladeným na senzorové sítě. Základní přístup je sestavení stromu, kde kořen je reprezentován základnovou stanicí. Operační systém TinyOS poskytuje řadu síťových protokolů. Pro sběr informací ze senzorů a jejich přenos do základnové stanice je nejvhodnější použít protokol `Collection` [2], který řeší problém přenosu dat z více uzlů do jednoho. Protokol `Collection` využívá pro fungování protokol `Collection Tree Protocol (CTP)` a tvoří nad ním rozhraní lehce použitelné pro uživatele. Protokol CTP je popsán v sekci 3.7.

Protokol `Collection` se snaží o spolehlivé doručení paketů k základnové stanici. Nicméně není schopen garantovat spolehlivé doručení, může dojít k doručení duplicitních paketů a není kontrolováno pořadí příchozích paketů.

Vzhledem k omezeným paměťovým možnostem uzlů a k nutnosti implementovat algoritmus, který bude schopen dynamicky ustavit spojení mezi uzly ve stromové struktuře, vyvstávají problémy, které je třeba řešit. Tyto problémy však nejsou doménou senzorových sítí, setkáváme se s nimi u všech druhů sítí. Následuje seznam těchto problémů:

- Detekce smyček – rozpoznat, kdy si má uzel zvolit svého potomka za svého rodiče.
- Potlačení duplikace paketů – vyrovnat se se ztrátou potvrzovacího paketu, je třeba šetři pásmo.
- Odhad kvality spojení k sousedním uzlům (jeden hop).
- Ujistit se, že neruším okolní komunikaci.

3.6.1 Rozhraní protokolu

Senzorový uzel může zastávat čtyři funkce (nebudeme překládat, použijeme anglické termíny): `producer`, `snooper`, `in-network processor` a `consumer`. Tyto funkce jsou do jisté míry dynamické a mění se v průběhu času dle potřeby. V závislosti na roli pak uzel použije náležité rozhraní komponenty Collection. Senzorová síť může být sdílena více aplikacemi. Pro identifikaci aplikace, která používá dané uzly, se použije `collection identifier`. Pokud je síť sdílena více aplikacemi, jsou data duplikována, ale řídicí signály nikoli.

Uzel, který produkuje data pro základnovou stanici označujeme jako `producer`. Používá rozhraní Send k odeslání dat ke kořenu stromu.

Uzly, které odposlouchávají zprávy během přenosu nazýváme `Snoopers`. Pro přečtení odposlechnuté zprávy používají rozhraní Receive.

Uzly typu `in-network processors` slouží k přeposílání paketů. Je použito rozhraní Intercept, které má následující strukturu:

[4] Error-correcting code, např. Hammingův.

```
interface Intercept {
    event bool forward(message_t* msg, void* payload, uint8_t len);
}
```

Intercept reaguje na jedinou událost a to Intercept.forward(). Tato událost by měla být vyvolána pokaždé, když je nutné přijatý paket přeposlat. Pokud je návratová hodnota této funkce FALSE, pak se paket přeposílat nesmí. Toto rozhraní umožní vyšší vrstvě prozkoumat obsah paketu a v případě potřeby paket zahodit nebo jeho obsah vložit do paketu existujícího.

Uzel, který zastává pozici základnové stanice, je označován jako `Consumer`. Používá rozhraní Receive pro přijetí zprávy. Abychom uzel učinili kořenem stromu, použijeme rozhraní RootControl:

```
interface RootControl {
    command error_t setRoot();
    command error_t unsetRoot();
    command bool isRoot();
}
```

3.6.2 Funkce protokolu

Protokol Collection je popsán komponentou `CollectionC`, jejíž funkčnost je následující:

```
configuration CollectionC {
    provides {
        interface StdControl;
        interface Send[uint8_t client];
        interface Receive[collection_id_t id];
        interface Receive as Snoop[collection_id_t];
        interface Intercept[collection_id_t id];
        interface RootControl;
        interface Packet;
        interface CollectionPacket;
    }
    uses {
        interface CollectionId[uint8_t client];
    }
}
```

Vlastnosti většiny rozhraní byly popsány v předchozí sekci. Rozhraní Receive, Intercept a Send dostanou při inicializaci jako parametr `collection identifier`, který určí, se kterou aplikací budou komunikovat.

3.7 Collection Tree Protocol

Collection Tree Protocol (CTP) [3] je protokol pracující nad stromovou strukturou. Některé uzlu jsou prohlášeny za kořeny stromu a ostatní uzly se snaží přenést data právě k nim. Jedná se o bezadresový (address-free) protokol. To znamená, že uzel neadresuje paket konkrétnímu uzlu, ale snaží se dopravit data do základnové stanice (kořen stromu). Pokud je

přítomno více základnových stanic, směruje uzel data ke stanici, ke které vede nejlevnější cesta. Uzly generují cesty ke kořenu na základě směrového gradientu.

CTP má několik mechanismů pro zlepšení spolehlivosti doručení, ale nezaručuje stoprocentní spolehlivost. Je navržen pro malé přenosy dat, neumí zabalit více malých rámců do jednoho paketu.

Mechanismy CTP

CTP využívá hodhotu ETX (expected transmissions, očekávaný počet přechodů) jako směrový gradient. Kořenový uzel má hodnotu ETX rovnu nule. ETX uzlu je rovna součtu ETX rodiče a ETX cesty k rodiči (cesta k rodiči má zpravidla hodnotu jedna). Ideální cesta by se měla vybírat na základě nejmenší hodnoty ETX. Ta je v CTP protokolu reprezentována jako 16ti bitová hodnota.

Jedním z problémů, které mohou při směrování nastat je vznik smyček. Většinou to nastane, pokud uzel začne směrovat data k uzlu s výrazně vyšším ETX než doposud, například na základě ztráty spojení s původním uzlem. Pokud nová cesta obsahuje uzel, jež je potomkem odesílajícího uzlu, vznikne smyčka. CTP implementuje dva mechanismy, které problém smyček řeší.

1. Každý CTP paket obsahuje aktuální hodnotu gradientu uzlu. Pokud uzel obdrží rámec s hodnotou gradientu menší, než je jeho vlastní, indikuje nekonzistenci stromu. Následně pak broadcastuje signalizační rámec a doufá, že uzel, který mu data poslal, tuto zprávu zachytí a upraví si náležitě směrovací tabulku. Pokud je skupina uzlů odtržena od zbytku sítě, vytvoří smyčku, ve které se ETX bude do nekonečna zvyšovat.

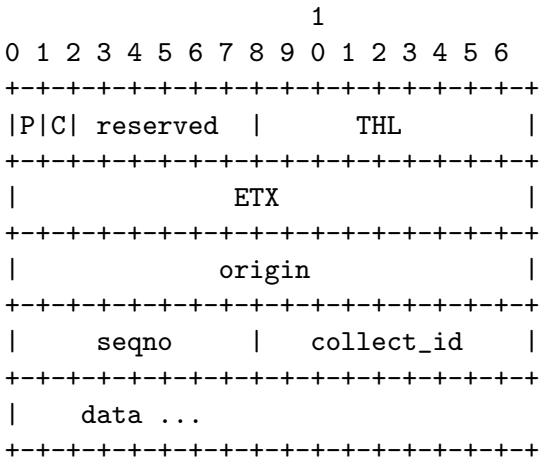
2. Uzly ignorují cesty s ETX vyšším, než je hodnota prahu. Tato hodnota je závislá na dané implementaci.

Duplikace paketů nastane, pokud příjemce příjme paket, odešle potvrzení přijetí (ACK), ale to není doručeno. Odesílatel pak znova odešle paket čímž může dojít k zahlcení celé sítě, jelikož duplikace paketů může mít v nejhorším případě exponenciální průběh.

Eliminaci duplicitních paketů komplikují smyčky. Kvůli nim může být paket korektně doručen více než jednou. Kdyby uzly rozpoznávaly duplikáty pouze podle adresy odesilatele a sekvenčního čísla, nebyly by schopny se se smyčkami vypořádat a mohly by přijímat duplicitní pakety. Proto datový rámec CTP obsahuje hodnotu THL (time has lived), která je inkrementována při každém hopu. Opakované přenosy ze stejné vrstvy mají stejné THL, pakety, které byly ve smyčce, mají THL vyšší.

Datový rámec CTP

Struktura datového rámce CTP:



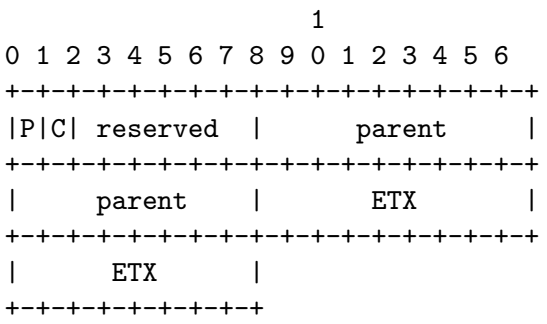
Následuje popis jednotlivých položek:

- P (Routing pull): Tento bit umožňuje uzlům vyžádat si od ostatních uzlů směrovací informace. Pokud uzel se správnou cestou odposlechne paket s nastaveným P bitem, měl by co nejdříve odeslat signalizační rámec (viz sekce 3.7).
- C (Congestion notification): Pokud uzel ztratí datový rámec, musí nastavit hodnotu C při příštím přenosu.
- THL (Time Has Lived): Pokud uzel vytvoří datový rámec, nastaví hodnotu THL na 0. Pokud uzel přijme rámec inkrementuje hodnotu THL. Pokud uzel příjme rámec s hodnotou THL rovnou 255, nastaví THL na 0.
- ETX: Pokud uzel odesílá datový rámec, nastaví hodnotu ETX na svoji vlastní. Pokud přijme paket s ETX menší než jeho vlastní, musí co nejdříve odeslat signalizační rámec.
- origin: Původní adresa paketu. Přeposílající uzel ji nesmí měnit.
- seqno: Původní sekvenční číslo. Původce jej nastaví, přeposilatel tuto hodnotu nesmí měnit.
- collect_id: Identifikátor protokolu pro vyšší vrstvu. Původce jej nastaví, přeposilatel tuto hodnotu nesmí měnit.
- data: 0-n bytů dat. Přeposílající uzel tuto hodnotu nesmí měnit.

Uzel musí poslat datový rámec CTP jako unicastovou zprávu s povoleným potvrzováním na linkové vrstvě.

Signalizační rámec CTP

Struktura signalizačního rámce CTP:



Následuje popis jednotlivých položek:

- P: stejné jako u datového rámce
- C: Pokud je zahozen směrový rámec, je nutné při dalším přenosu nastavit tento bit.
- parent: Aktuální rodič uzlu.
- ETX: Aktuální metrika uzlu.

Pokud uzel přijme signalizační rámec, musí aktualizovat svoji směrovací tabulku na základě nových hodnot ETX. Při značné změně vlastního ETX by se měl broadcastovat signalizační rámec, aby byly informovány i ostatní uzly.

Kapitola 4

Návrh topologie a protokolu

V následující kapitole bude popsán návrh topologie vhodné pro vytvoření senzorové sítě a s tím spojený návrh komunikačního protokolu.

4.1 Návrh topologie sítě

K návrhu správné topologie je třeba se zamyslet nad tím, v jaké situaci bude daná implementace používána. K monitorování místnosti nebo menší haly, bude stačit statická topologie, kdy každý uzel bude posílat data základnové stanici (topologie hvězda, viz 3.1). To je však maximum, jaké nám statická topologie sítě může nabídnout.

4.1.1 Sítě s prvky statické topologie

Jak již bylo uvedeno v sekci 3.3, velkou výhodou senzorových sítí je jejich velká flexibilita a odolnost vůči výpadkům jednotlivých uzlů. Pokud bychom vytvořili rozsáhlou senzorovou síť se statickou topologií, odstranili bychom jednu z velkých předností bezdrátové senzorové sítě. Vylepšení statické topologie je transformace sítě na částečně dynamickou. Zde bychom ponechali jakési páteřní uzly (pseudo základnové stanice) a kolem každého z nich bychom vytvořili topologii hvězdy. Data za základnových stanic by se pak mohla centralizovat pomocí internetu nebo v rámci sítě přeposláním dalším základnovým stanicím. Zde vyvstává omezení na horní limit vzdálenosti jednotlivých základnových stanic a jsme také závislí na spolehlivosti páteřních uzlů – pokud dojde k výpadku, může být odříznuta velká část sítě.

Pokud bychom chtěli, aby kolem každé páteřní základnové stanice byly uzly rozmístěny ve složitější struktuře než je hvězda, je vhodné, aby topologie "za" páteřním uzlem byla dynamická.

Výhody statické topologie

- Páteřní základnové stanice mohou mít zdroj napájení a přístup k datovému úložišti.
- Je možné monitorovat více vzdálených objektů a data sbírat na jednom místě.

Nevýhody statické topologie

- Náchylné vůči poruchám páteřních uzlů.
- Nízká mobilita – přemístění na jiné místo je relativně obtížné.

4.1.2 Plně dynamická topologie

U bezdrátových senzorových sítí je zpravidla žádoucí, aby měly plně dynamickou topologii. To zajistí jednoduchost přidání nového uzlu, zvýší odolnost vůči výpadku jiného uzlu a umožní síti velkou mobilitu – odolné vůči změně podmínek.

Idea fungování senzorové sítě s plně dynamickou topologií je taková, že existuje jedna základnová stanice, která sbírá data a velké množství uzlů, které data do základnové stanice odesílají. Uzel má po spuštění informaci pouze o tom, jaké ID má základnová stanice a na základně toho k ní doručí data.

Výhody dynamické topologie

- Velká robustnost.
- Jednoduché přidávání/odebírání uzlů.
- Vysoká odolnost vůči selhání jednotlivých uzlů.

Nevýhody dynamické topologie

- Složitější směrovací protokol.

4.1.3 Volba topologie

Pro potřeby monitorování budovy přichází v úvahu užití plně dynamické sítě, nebo použití sítě se statickou páteří a dynamickou částí kolem každé základnové stanice páteře. Druhá zmíněná tvoří abstrakci nad větším množstvím plně dynamických sítí, zbývá už jen zvolit, zda budou základnové stanice páteřní sítě komunikovat pomocí rádiového modulu na desce uzlu nebo zda budou komunikovat prostřednictvím počítačové sítě, ke které jsou základnové stanice připojeny.

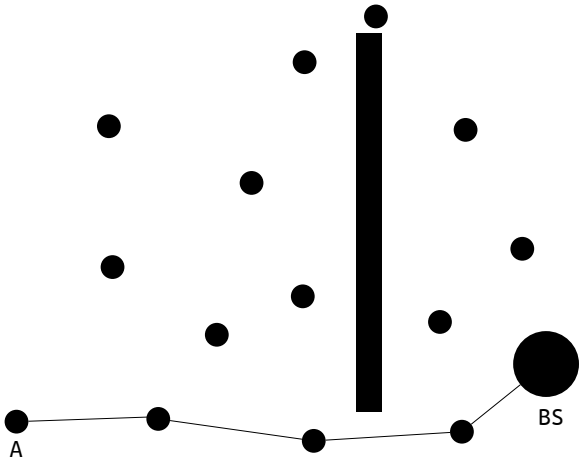
Plně dynamická síť je vhodná pro použití na relativně malém prostoru s vyšší hustotou uzlů. Ta zajistí odolnost vůči výpadkům. Částečně statická síť je pak dobře použitelná k monitorování například dvou odlehlých hal. Bylo by zbytečné instalovat nové uzly mezi těmito halami pouze z důvodu přenosu informací mezi základnovými stanicemi, je lepší využít stávající datovou infrastrukturu.

Na obrázcích 4.1 a 4.2 je ukázáno ustavení nové cesty pro přenos dat z uzlu `A` do základnové stanice (BS) po výpadku jednoho z klíčových uzlů. Oblast je předělena stěnou, která ruší signál.

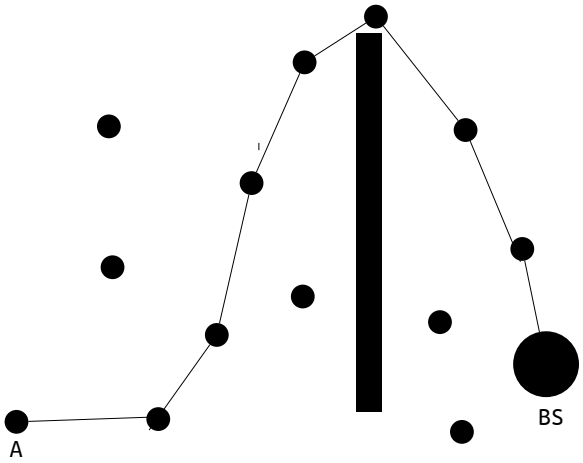
4.2 Návrh komunikačního protokolu

V následující sekci se zaměříme na návrh komunikačního protokolu pro plně dynamickou síť. Jak bylo předesláno, TinyOS nabízí několik připravených komunikačních protokolů. Pro sběr dat a doručování k základnové stanici vyhovuje protokol Collection popsaný v sekci 3.6. Ten využívá mechanismů protokolu CTP (Collection Tree Protocol), jehož struktura je popsána v sekci 3.7. V práci byly testovány vlastnosti protokolu Collection z operačního systém TinyOS a dále byl navržen nový protokol, který vychází z vlastností protokolu CTP. V následujících sekcích budou popsány a porovnány oba přístupy.

Fungování sítě je složeno ze dvou základní činností – ustavení topologie a posílání dat v sítí. Pro každý přístup budou diskutovány obě tyto problematiky.



Obrázek 4.1: Cesta nalezena dynamickým ustanovením sítě.



Obrázek 4.2: Došlo k selhání uzlu. Topologie se automaticky rekonfigurovala. Pokud by došlo k výpadku na páteřním uzlu, byla by nyní odříznuta více než polovina uzlů v síti.

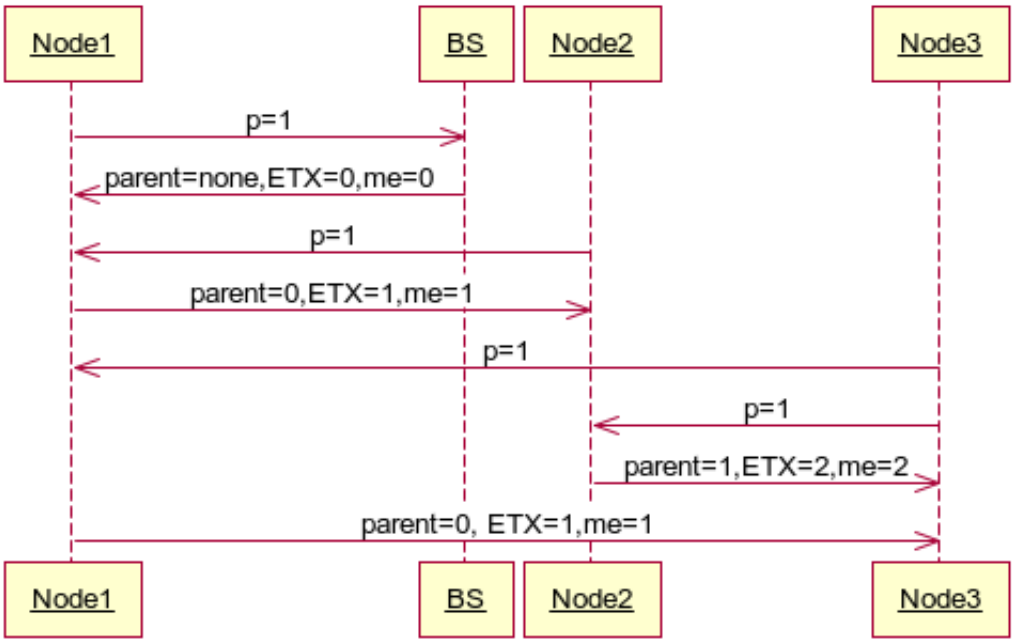
4.2.1 Collection v TinyOS

Ustavení topologie

Pro komunikaci je nutné, aby si každý uzel vytvořil směrovací tabulku. V té je uveden uzel, přes který se data směrují k základnové stanici. Topologie se vytvoří při aktivování uzlů. K jejím změnám pak dochází pokud uzly odebereme případně přidáme nebo pokud se změní okolní podmínky (z nějakého důvodu se začnou pakety ztrácet).

Protokol CTP k ustavení topologie používá signalizačních rámců (viz 3.7). Ty jsou vysílány v proměnlivém intervalu s intenzitou 64ms – 1 hodina [5]. Interval se zdvojnásobí vždy, když nebyl vyslán signalizační rámec. Je naopak nastaven na minimum, pokud je přijat rámec s P-bitem nastaveným na hodnotu 1. Dále k vynulování dojde, pokud uzel detekuje velkou ztrátu dat nebo případnou nekonzistenci v síti.

Obrázek 4.3 ukazuje komunikaci při ustavení sítě. Nově přidaný uzel broadcastuje zprávu s nastaveným příznakem P. Tím signalizuje dostupným uzlům, že nezná topologii a po-



Obrázek 4.3: Ukázka ustanovení sítě. Nejprve je zapnuta základnová stanice BS spolu s uzlem Node1. Další v pořadí je zapnut uzel Node2 a nakonec uzel Node3.

třebuje data pro sestavení směrovací tabulky. Ostatní uzly na tuto žádost musí reagovat zasláním signalizačního paketu, který obsahuje informaci o tom, kdo je jejich rodič a jak to k rodiči mají daleko (hodnota ETX je při spuštění uzlu rovna jedné). Uzel také ví, od koho zprávu přijal (není obsahem signalizačního rámce). Na základě obdržených informací inkrementuje vlastní ETX o nejmenší z přijatých ETX a uzel s nejmenším ETX prohlásí za svého rodiče.

Posílání dat v sítí

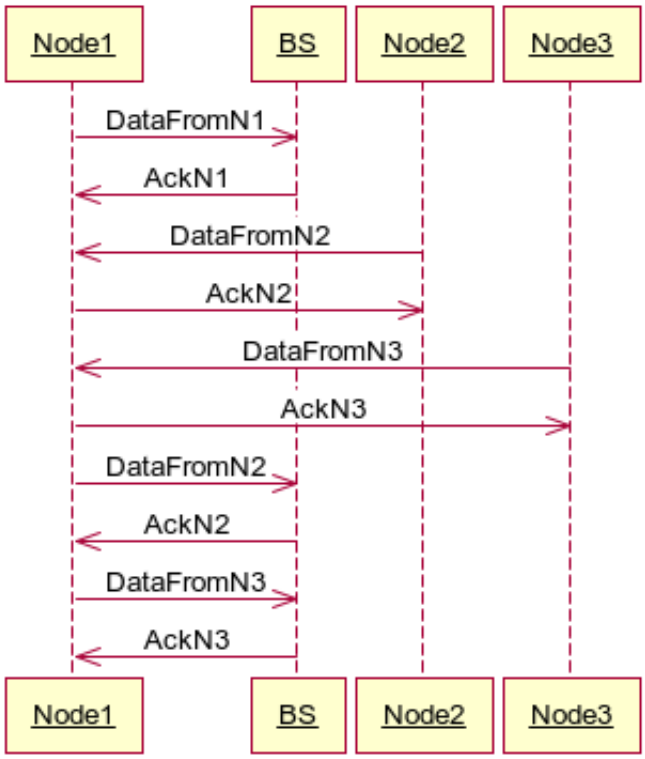
Data se šíří výhradně zasíláním unicastových zpráv dle směrovací tabulky vytvořené dle postupu v sekci 4.2.1. U protokolu CTP slouží datové rámce mimo přenosu dat i ke kontrolování chyb v sítí. Není tedy třeba periodické zasílání signalizačních rámců a šetří se tím využití pásma. Každý datový rámec totiž může nastavit P-bit na jedničku a tím vyvolat žádost o ustavení nové topologie. Komunikace uzlů se základnovou stanicí je demonstrována na obrázku 4.4.

Pří posílání dat nedochází k zahazování paketů. Pokud dojde ke ztrátě je nejdříve poslán signalizační paket a následně data. Doufáme přitom, že bude nalezena správná cesta.

4.2.2 Upravená verze protokolu CTP

Použité prostředky

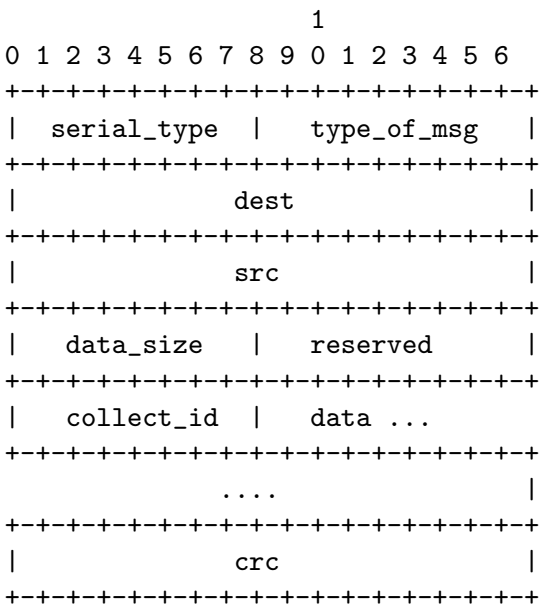
Ke komunikaci mezi uzly bylo použito komponent `AM_Sender` a `AM_Receiver`, které poskytují rozhraní `AMPacket`, `AMSend` a `Receive`. Komponenty typu AM (Active Message) jsou



Obrázek 4.4: Distribuce dat k základnové stanici. Uzel Node1 vidí přímo na základnovou stanici BS, uzly Node2 a Node3 vidí na sebe navzájem a na uzel Node1.

schopny sdílet rádiový modul ke komunikaci, jsou tedy schopny současného příjmu i vysílání dat. Podrobnější popis komponent nalezneme v sekci 5.1.

Každý AMPacket, dále jen paket, má strukturu, která vychází z definice datového typu `message_t`, který má následující strukturu.



Význam jednotlivých položek:

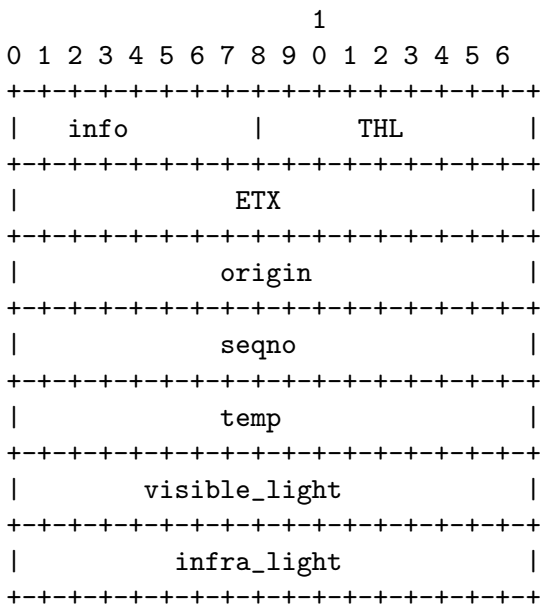
- serial_type: Označuje, zda se jedná o potvrzované nebo nepotvrzované spojení, hodnota nemusí být známa. Může obsahovat hodnoty SERIAL_PROTO_PACKET_ACK pro potvrzované spojení, SERIAL_PROTO_PACKET_NOACK pro nepotvrzované spojení, SERIAL_PROTO_ACK pro označení potvrzovací zprávy při potvrzovaném spojení a SERIAL_PROTO_PACKET_UNKNOWN pro neznámý paket [21]. V práci jsou všechny pakety typu SERIAL_PROTO_PACKET_NOACK.
- type_of_msg: Označuje, o jaký typ zprávy se jedná. Tuto informaci využívá základnová stanice při přijetí paketu a následné komunikaci se sériovým portem. Může obsahovat hodnoty TOS_SERIAL_802_15_4_ID pro komunikaci pomocí 802.15.4 zpráv a TOS_SERIAL_ACTIVE_MESSAGE_ID pro platformě nezávislou komunikaci (použito v práci). Dále dvě vyhrazené hodnoty TOS_SERIAL_CC1000_ID pro rádiové moduly cc1000 uzlů mica2 a TOS_SERIAL_UNKNOWN_ID pro chybový kód [6].
- dest: Adresa příjemce, pro broadcast je vyhrazena adresa 0xFFFF.
- src: Adresa odesilatele paketu.
- data_size: Velikost dat, které paket obsahuje. Výchozí velikost dat je nastavena na 28 bytů [7]. Tuto hodnotu můžeme změnit při překladu nastavením parametru –DTOSH_DATA_LENGTH=nova_velikost.
- reserved: Vyhrazeno pro další použití.
- collect_id: Každá skupina paketů má tohle číslo stejné. Nastavuje se při vytváření komponent `AM_SenderC` a `AM_ReceiverC`, které zajišťují komunikaci mezi uzly. Vzhledem k tomu, že aplikace pracuje pouze s jednou skupinou uzlů, je tato hodnota irelevantní a byla náhodně nastavena na hodnotu 0x71.
- data: Obsah paketu. Podrobněji popsáno v sekci 4.2.2.
- crc: Kontrolní součet.

Paket odesílaný přes sériovou linku od základnové stanice do aplikace, která sbírá data, je navíc obalen hodnotami 0x7e – ty značí začátek a konec každého paketu. Jelikož hodnota 0x7e může být obecně obsažena v těle paketu, musí se takové hodnoty vhodně zakódovat, aby byla hodnota 0x7e unikátní. Hodnota 0x7e je proto zakódována jako dvojice 0x7d 0x5e, hodnotu 0x7d pak reprezentuje dvojice 0x7d 0x5d.

S touto skutečností musíme počítat při dekódování dat, které přijímáme ze základnové stanice.

Obsah datové části paketu

Každý paket nese ve své datové části informace potřebné pro ustavení topologie a sběr dat. Na základě protokolu CTP (3.7) byla navržena struktura datového a směrovacího paketu. Datový paket:



Význam jednotlivých položek:

- info: Pro datový paket může nabývat hodnot 0 a 1. 0 značí paket, který je unicastovaný příjemci, který je uložen ve směrovací tabulce jako další hop. 1 značí paket, který obsahuje data, ale je odeslán broadcastem všem dostupným stanicím. Uzel tím signalizuje, že neví, kam má data směrovat. Pokud uzel obdrží paket s hodnotou info=1, musí odeslat směrovací paket.
- THL: Doba, kterou paket strávil v sítí. Uzel, který paket vytvoří, inicializuje tuto hodnotu na nulu, přeposílající uzel ji inkrementuje o jedna.
- ETX: Počet hopů od základnové stanice. Základnová stanice má hodnotu ETX=0, nově zapojený uzel má hodnotu ETX nastavenu na 0xFFFF, která značí, že uzel neví, kam směrovat.
- seqno: Sekvenční číslo. S každým odeslaným paketem se zvyšuje o jedničku, přeposílající uzel tuto hodnotu nesmí měnit.
- temp: Teplota změřená senzorem.

- visible_light: Intenzita viditelneho světla změřená senzorem.
- infra_light: Intenzita infra světla změřená senzorem.

Směrový paket:

```
                    1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   info        |      ETX
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    ETX         |      seqno
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    seqno       |
+-+-+-+-+-+-+-+-+
```

Význam jednotlivých položek:

- info: U směrového paketu má hodnotu info=3. Je to signál pro aplikaci, která data zpracovává, že tento paket se bude ignorovat.
- ETX: Hodnota ETX odesilatele paketu.
- seqno: Sekvencí číslo, stejný význam jako u datového paketu.

Ustavení topologie

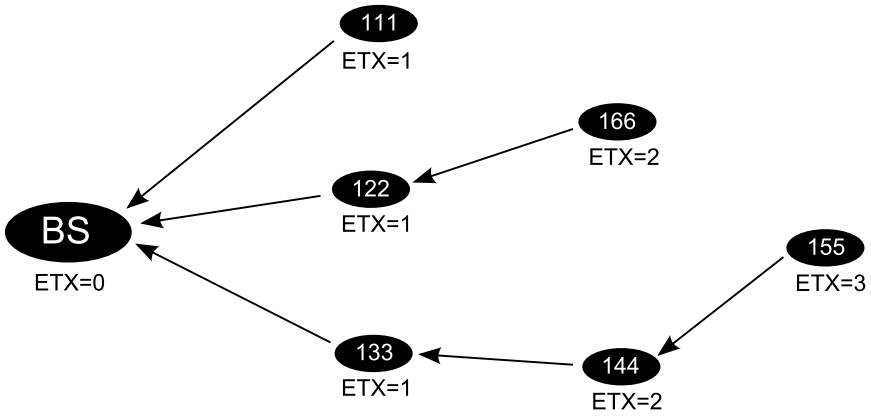
Oproti implementaci v TinyOS bylo zvoleno nepotvrzované spojení. Ušetřila se tak polovina přenosů. Vycházelo se z předpokladu, že pokud uzel ztratí signál, potvrzení přijetí paketu stejně nepřijde. Absence potvrzení přijetí paketu je kompenzována zavedením časovače, který každých 30 sekund odešle směrový paket.

Topologie se ustavuje vytvářením stromu, ve kterém kořen tvoří základnová stanice a listy jsou jednotlivé uzly. Ty data jak generují, tak případně přeposílají. Směrovací metrikou je hodnota ETX každého uzlu. Hodnota ETX u základnové stanice je neměnná a je nastavena na hodnotu 0. Všechny ostatní uzly mají hodnotu ETX vyšší. Ukázku ETX v ustavené síti můžeme vidět na obrázku 4.5.

Nyní předpokládáme, že základnová stanice je připojena ke zdroji napětí a je zapnuta. Každý uzel má po startu hodnotu ETX=0xFFFF a má nastaven příznak `readyToSend` na hodnotu `false`. Tento příznak indikuje, že nezná další hop, tj. směrovací tabulka je prázdná. Směrovací tabulka u každého uzlu obsahuje vždy jeden záznam – nejedná se tedy o tabulku v pravém slova smyslu, ale pouze o adresu uzlu, na který je nejvýhodnější data odeslat. Uzel tedy broadcastuje datový rámec s hodnotou info=1 do sítě a čeká na směrový rámec.

Při přijetí směrového rámce porovná hodnotu ETX odesilatele se svojí vlastní. Pokud je hodnota nižší, nastaví odesilatele paketu jako svůj další hop (aktualizace směrovací tabulky) a svoje ETX nastaví na hodnotu jedna vyšší, než je hodnota ETX odesilatele.

Aby byla síť schopna reagovat na výpadky jednotlivých uzlů, je směrovací tabulka po uplynutí určitého časového intervalu smazána a je tak vynucena její aktualizace. Délka intervalu je v rozmezí 4-5 minut, hodnota je určena generátorem náhodných čísel. Náhodné číslo z intervalu je zde místo fixní hodnoty kvůli případu, kdy jsou všechny uzly napájeny



Obrázek 4.5: Hodnoty ETX pro síť použitou při testování.

centrálně ze sítě a jsou připojeny k jednomu vypínači. Pokud bychom použili fixní délku intervalu a uzly bychom spustili současně, došlo by v daném intervalu k výpadku celé sítě a takové chovaní je nežádoucí. Kdyby tento mechanismus chyběl, nebyla by síť schopna reagovat na výpadky uzlů, protože by po počátečním ustavení topologie každý uzel směroval data dle směrovací tabulky bez ohledu na její aktuálnost.

Posílání dat v síti

Každý uzel generuje data pro základnovou stanici, některé uzly pak dle potřeby přeposílají data od vzdálenějších uzlů. Základnová stanice odesílá veškerou příchozí komunikaci přes sériovou linku do připojené stanice (podrobnější informace jsou uvedeny v sekci 5.5), odpovídá pouze na zprávy s příznakem info=1 a to zasláním směrového rámce. Ten vysílá také nezávisle na externím požadavku každých 10 sekund.

Základní funkcí uzlu je generovat data pro základnovou stanici. Tato jsou čtena ze senzorů každé 3 sekundy a ihned po dokončení čtení jsou odeslána.

Po přijetí paketu uzel zkontroluje, zda se jedná o datový nebo směrový paket. Při přijetí směrového paketu zkontroluje, zda nebude měnit své směrovací údaje. Pokud přijme datový paket, zkontroluje hodnotu `info`. Pokud je nastavena na hodnotu 1, posílá směrový paket, pokud na hodnotu 0, přeposílá paket dále. Pokud příchozí paket obsahuje menší ETX než je ETX uzlu, který jej přijal, je vyslán směrový paket, protože zřejmě došlo k nekonzistenci v síti.

Kapitola 5

Implementace aplikace pro jednotlivé uzly

V následující kapitole jsou popsány komponenty, ze kterých je aplikace sestavena.

5.1 Práce s pakety

Pro práci s pakety je použita komponenta `AM_Sender`, která slouží k odesílání paketů a `AM_Receiver` k příjmu paketů. Obě jsou standardní součástí systému TinyOS. Komponenta `ActiveMesssageC` slouží k obsluze rádiového modulu. Po startu aplikace je voláno rozhraní komponenty `ActiveMessageC start`, které rádiový modul spustí. Úspěšné spuštění rádiového modulu je doprovázeno událostí `startDone`. V této chvíli se inicializují časovače a uzel je připraven ke komunikaci.

5.1.1 Odesílání paketu

Komponenta `AM_Sender` poskytuje rozhraní `Packet`, `AMPacket` a `AMSend`. Pro sestavení paketu je nutné získat ukazatel na jeho datovou část. Funkce `Packet.getPayload(message_t * msg, int len)` vrací ukazatel na data délky `len`, která budou připojena k paketu, který je reprezentován zprávou `msg`. Takto vytvořený paket odešleme voláním `AMSend.send(int addr, message_t * msg, uint8_t len)` kde `addr` je adresa příjemce, `msg` reprezentuje paket a `len` je délka datové části paketu.

5.1.2 Příjem paketu

Komponenta `AM_Receiver` poskytuje rozhraní `Receive`, které implementuje událost `event message_t * Receive.receive(message_t * msg, void * payload, uint8_t len)`. Ta je volána pokaždé, když je detekován paket určený danému uzlu. Hodnota `msg` je ukazatel na příchozí paket. `payload` obsahuje ukazatel na datovou část paketu. Tuto hodnotu je vhodné přetypovat na očekávaný datový typ – v aplikaci jsou pro datovou část implementovány struktury `DataPacket` a `RoutingPacket`, které obsahují hodnoty pro datový a směrovací paket dle sekce 4.2.2.

Ke zjištění odesilatele paketu se použije funkce `am_addr_t AMPacket.source(message_t * msg)`.

5.2 Časovače

Časovače se využívají ke generování periodických událostí. V aplikaci jsou použity tři instance časovače `TimerMilliC`. Časovač `TimerRead` volá čtení nových dat ze senzoru, `TimerReset` periodicky maže směrovací tabulku a `TimerBeacon` obsluhuje zasílání směrovacích paketů. Časovač se inicializuje voláním funkce `startPeriodic(int interval)`, která jako parametr přijímá délku intervalu v milisekundách. Po uplynutí intervalu je generována událost `event void Timer.fired()`.

5.3 LED diody

Uzly IRIS a MICAz jsou osazeny třemi LED diodami, které jsou vhodné pro vizuální interakci s uživatelem. Obě platformy disponují červenou, zelenou a oranžovou diodou. Diody jsou ovládány prostřednictvím komponenty LedsC, která poskytuje rozhraní Leds, které implementuje funkce pro ovládání jednotlivých diod. Obecný tvar funkcí je `Leds.ledXOn()` pro zapnutí, `Leds.ledXOff()` pro vypnutí a `Leds.ledXToggle` pro přepnutí stavu. `X` nabývá hodnot 0 pro červenou diodu, 1 pro zelenou diodu a 2 pro oranžovou diodu.

5.4 Čtení dat ze senzorů

Abychom mohli získávat požadovaná data, je třeba k uzlům připojit pomocí 51-pinového rozšiřujícího konektoru senzorovou desku. Na trhu je k dispozici široká škála senzorových desek. V práci byly použity desky MTS420/MTS400CC, které disponují dvouosým akcelerometrem, senzory tlaku, intenzity osvětlení, vlhkosti vzduchu a teploty. Za účelem monitorování budovy byly jako nejvhodnější veličiny zvoleny teplota a intenzita osvětlení.

5.4.1 Popis senzorů

Deska MTS420/MTS400cc je vybavena senzorem Sensirion SHT11 [14] pro snímání teploty a senzorem TAOS TSL2550D[16] pro měření intenzity osvětlení.

Teplotní senzor je schopen snímat teplotu v rozmezí $-40°C - 123.8°C$ s přesností $0.01°C$ a je vybaven 14-bitovým ADC převodníkem. Výsledná teplota `T` musí být upravena na základě napájecího napětí a požadovaných výstupních jednotek dle vztahu 5.1. Hodnoty parametrů d_1 a d_2 jsou uvedeny v tabulce 5.1, hodnota SO_T je výstup senzoru.

$$T = d_1 + d_2 \cdot SO_T \tag{5.1}$$

Senzory jsou napájeny napětím 3V a jak již bylo řečeno výše, máme k dispozici 14 bitů. Budeme požadovat teplotu ve stupních Celsia. Výsledný vztah pro výpočet teploty je uveden v rovnici 5.2.

$$T = -39.7 + 0.01 \cdot SO_T \tag{5.2}$$

Senzor intenzity osvětlení je schopen snímat světlo vlnových délek 400-1000 nm [22], což je ekvivalent citlivosti lidského oka. Senzor se skládá ze dvou fotodiod. První je citlivá jak na viditelné světlo, tak na infračervenou složku (kanál 0), druhá pak hlavně na infračervené světlo (kanál 1). Programově se jedná o komponenty `VisibleLight` a `InfraredLight`. Přepočet na jednotku světelné intenzity Lux pak provedeme aplikací vztahu 5.3, kde `Ch0` odpovídá kanálu 0, `Ch1` kanálu 1 a $R = Ch1/(Ch0 - Ch1)$.

$$intenzity = (Ch0 - Ch1) \cdot 0.390 \cdot e^{-0.181 \cdot R^2} \tag{5.3}$$

VDD	d_1 (°C)	d_1(°F)
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

SO_T	d_2 (°C)	d_2(°F)
14bit	0.01	0.018
12bit	0.04	0.072

Tabulka 5.1: Převodní tabulky pro výpočet teploty.

V sekci 5.4.2 jsou popsány komplikace provázející měření intenzity osvětlení.

5.4.2 Problémy při implementaci

Jelikož je TinyOS neustále vyvíjen, je třeba brát ohled na aktuálnost ovladačů. Komponenty nainstalované z balíčků obsažených v repozitářích linuxových systémů budou zřejmě zastaralé. Pro korektní fungování je nutné stáhnout aktuální verze z repozitáře[1].

Jedním z největších problémů je zřejmě chyba v TinyOS, která neumožňuje použít vztah 5.3 k výpočtu intenzity osvětlení. Vztah očekává dvě 16 bitová nezáporná čísla. Komponenty pro měření hodnot však vrací pouze složku `Ch0` (viditelné i infra světlo), a to pouze jako 8 bitovou hodnotu. Převod na dat na jednotku Lux tak není možné reálně uskutečnit, protože výsledné hodnoty jsou řádově menší, než bychom očekávali.

Pokud uzel vystavíme přímému slunečnímu svitu, světelná intenzita by měla být alespoň 32000Lux [18]. Uzel však vrátí hodnoty $Ch0 = 255$ a $Ch1 = 0$. Po dosazení do rovnice 5.3 dostaneme vztah 5.4.

$$(255 - 0) \cdot 0.390 \cdot e{-}0.181 \cdot 0 = 99.45 \tag{5.4}$$

Tato hodnota se očekávanému řešení ani zdaleka neblíží. Za správný výsledek jsou tedy brány přímo hodnoty `Ch0`, které alespoň reflektují změnu světelných podmínek, i když z naměřených hodnot nemůžeme odhadnout přesnou intenzitu osvětlení. Experimenty bylo zjištěno, že osvětlení v běžné kanceláři odpovídá hodnotám `Ch0` v intervalu <210-240>, při kontaktu se slunečním svitem pak dostaneme hraniční hodnotu 255, která signalizuje maximální hodnotu, kterou je uzel schopen naměřit. Noční hodnoty světelné intenzity v budově školy byly z intervalu <128-130>. Hodnota 128 se ukázala jako spodní hranice citlivosti uzlu – taková hodnota byla naměřena v noci v místnosti bez oken a světel. Na základě empiricky získaných hodnot by tedy bylo možné naprogramovat například systém automatického rozsvěcení světel i bez plně funkčního převodníku.

5.5 Základnová stanice

Uzel, ke kterému je veškerá komunikace směrována, se nazývá základnová stanice a je připojen k počítači USB kabelem, přes který pomocí rozhraní UART sériově komunikuje a je z něj také nabíjen. U základnové stanice se tudíž nemusíme starat o energetickou úsporu.

Komunikační část je implementována stejně jako u ostatních uzlů. Navíc přibývá nutnost implementovat sériovou komunikaci s připojenou stanicí. Tu poskytuje standardní komponenta `SerialActiveMessageC`, která poskytuje komponenty `UartSend`, `UartReceive`, `UartPacket` a `UartAMPacket`. Funkce zmíněných komponent jsou stejné jako u ekvivalentů pro rádiovou komunikaci s tím rozdílem, že komunikace probíhá po sériové lince.

[1] http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/

Implementace sériové komunikace byla převzata z ukázkových příkladů systému TinyOS. Základnová stanice tak každý příchozí paket posílá skrz sériové rozhraní do připojené stanice.

Stejně jako ostatní uzly je i základnová stanice schopna reagovat na pakety, které obsahují žádost o zaslání směrového paketu. Dále nezávisle na příchozích datech vysílá každé 4 vteřiny směrový paket, aby informovala své okolí.

Kapitola 6

Reprezentace nasbíraných hodnot

Data, která základnová stanice odešle do počítače, musí být zpracována a srozumitelně reprezentována. Za tímto účelem byla implementována aplikace, která data čte a ukládá do databáze a dále bylo navrženo webové rozhraní, které zobrazuje grafický průběh naměřených hodnot v závislosti na čase. K dispozici je i možnost exportu naměřených dat v textové podobě.

6.1 Čtení a uchování dat

Pro implementaci aplikace, která bude číst data ze sériového portu, byl zvolen jazyk Python ve verzi 2.7. Aplikace se skládá ze dvou částí – klienta a serveru. Klient za pomoci vestavěné knihovny `serial` čte data po jednotlivých bytech ze sériového portu `\dev\ttyUSB1`, který základnová stanice používá pro odchozí komunikaci. Na základě struktury příchozího paketu rozpozná, které pakety je třeba uložit, tj. ignoruje pakety, které se starají o ustavení síťové topologie. Funguje jako stavový stroj, který po přijetí paketu s hodnotou 0x7e (ohraničení paketu) začne načítat relevantní data. Těmi jsou adresa původce paketu, hodnota teploty a hodnota intenzity osvětlení. Po přečtení těchto hodnot vytvoří socket, ve kterém získané data odešle serverové části aplikace.

Serverová část aplikace poslouchá na portu 31337 a veškerou příchozí komunikaci ukládá do SQLite databáze. Ta obsahuje jedinou tabulku, která udržuje nasbíraná data. Tabulka obsahuje čtyři sloupce, které označují postupně ID uzlu, naměřenou teplotu, intenzitu osvětlení a čas uložení předchozí trojice hodnot. Čas je měřen s přesností na sekundy a je uchováván jako počet milisekund od 1. 1. 1970. Tato konvence je zde kvůli knihovně vykreslující grafy – ta požaduje takto formátovaný čas. Část databáze je prezentována v tabulce 6.1.

Díky implementaci typu klient-server je možné posílat data z místa, kde je základnová stanice do místa, kde se budou data ukládat a kde k nim bude umožněn přístup z venčí. To přináší výhodu ve chvíli, kdy monitorujeme oblast se špatnou konektivitou.

Aktuálně je aplikace nastavena tak, že jak klient, tak server běží na jednom počítači, který data přijímá a zároveň ukládá.

6.2 Grafická reprezentace

Abychom byli schopni nasbíraná data přehledně zobrazovat, bylo implementováno webové rozhraní, které vykresluje grafy s průběhem nasbíraných hodnot – teploty a intenzity osvětlení. Vzhled rozhraní můžeme vidět v příloze C.1. Webová aplikace se skládá z PHP skriptu,

nodeID	temp	light	time
122	23.87	219.0	1335357487000.0
122	23.87	219.0	1335357487000.0
122	23.88	220.0	1335357488000.0
122	23.89	220.0	1335357490000.0
122	23.9	211.0	1335357492000.0
166	24.54	217.0	1335357501000.0
155	23.98	218.0	1335357502000.0
166	24.56	219.0	1335357503000.0
144	24.35	219.0	1335357503000.0
144	24.35	219.0	1335357503000.0
155	23.99	222.0	1335357504000.0

Tabulka 6.1: Ukázka části tabulky, která tvoří databázi.

který přistupuje k databázi vytvářené serverovou částí alikace popsané v sekci 6.1 a z JavaScriptu, který vykresluje grafy.

6.2.1 PHP

Jelikož klientský JavaScript z bezpečnostních důvodů nemůže přistupovat k datům uloženým na pevném disku, bylo třeba implementovat rozhraní v PHP, které bude data zprostředkovávat. Webová aplikace tedy s využitím jQuery posílá požadavek pro PHP skript, který vrátí požadovaná data. Pro přístup k databázi bylo použito rozšíření `PDO`[1], které poskytuje lehké a univerzální rozhraní pro přístup k jakémukoli typu databáze. Toto rozšíření bylo použito hlavně z toho důvodu, že PHP verze 5 nemá nativní podporu databázového systému SQLite3, ve kterém jsou uložena data. Pokud by v budoucnu byl změněn způsob ukládání dat, bude stačit změnit typ databáze na aktuální a díky PDO bude vše fungovat stejně jako v původní verzi.

Skript `getNewData.php` volaný JavaScriptovou aplikací vrací pole hodnot potřebná pro sestavení grafů (pro podrobnosti o vykreslování grafů viz sekci 6.2.2). Při zavolání nejprve přečte nastavení, která jsou poslána metodou POST. Jsou očekávány následující parametry:

- Hodnota určující dle jakých kritérií budeme vybírat. Může nabývat hodnot -1 pro zobrazení poslední minuty (testovací účely), -2 pro zobrazení poslední hodiny a -3 pro zobrazení hodnot zadaných datem.
- Počáteční datum intervalu.
- Koncové datum intervalu.
- Čas počátečního data.
- Čas koncového data.
- Název použité databáze.

Dle délky zvoleného intervalu je upraven počet zobrazených hodnot pro každý uzel, aby nedocházelo k zahlcení vykreslovacího mechanismu. Základní naprogramování uzlů zajišťuje

[1] PHP Data Objects [17]

sběr hodnot každé tři sekundy, tj. 20 hodnot do minuty. Tato hodnota je dostatečně malá, pro zobrazení poslední minuty tedy nemusíme provádět další úpravy.

Za hodinu každý uzel vygeneruje již 1200 hodnot. Vzhledem k omezené ploše, kterou graf může zabírat, je zbytečné zobrazovat hodnoty všechny. Pro každý uzel tedy vybereme 20 hodnot, které budou reprezentovat průběh za poslední hodinu. Abychom zachovali kontext času, musíme vybírat hodnoty rovnoměrně skrz celý hodinový interval. Toho docílíme výběrem každé šedesáté hodnoty uložené v databázi.

Selekce reprezentujících hodnotu u intervalového zobrazení musí probíhat dynamicky. Zjistíme rozdíl počátečního a koncového času, tuto hodnotu převedeme na minuty a označíme ji `n`. Nyní budeme brát v potaz každou `n`-tou hodnotu, tak získáme výsledných 20 hodnot pro každý uzel. Můžeme si všimnout, že i hodnota 60 z hodinového zobrazení byla zvolena dle stejné úvahy.

6.2.2 Vykreslování grafů

Vykreslování grafů je realizováno pomocí knihovny Flot [15], která je implementovaná v JavaScriptu a je postavena na jQuery. Webové rozhraní poskytuje uživateli možnost volby zobrazení. Po spuštění bude aplikace sledovat všechny uzly, které jsou k dispozici v databázi a bude zobrazovat jejich výstup za poslední minutu. Uživatel může selektivně vybrat pouze některé uzly a může si vybrat, zda bude sledovat aktuální stav senzorů v poslední minutě, hodině nebo ze zadaného intervalu.

Nyní přiblížíme implementaci vykreslování grafů. Následující kus kódu ukazuje volání funkce, která grafy kreslí.

```
$.plot(
    $("#tempDiv"),
    dataT,
    {series:
        {points:{show:true},
        lines:{show:true}},
        xaxis:{mode: "time",tickLength:5 }
    }
);
```

První parametr `#tempDiv` je jméno HTML DIVu, který byl prázdný vytvořen a bude naplněn grafem.

Zdrojová data

Druhý parametr obsahuje data pro vykreslení grafu. Flot nabízí poměrně širokou škálu možností, jakými graf vykreslit. Dle implementace jsou vstupní data reprezentována vždy polem. Pokud bychom chtěli zobrazit pouze X-Y spojnicový graf, vypadala by položka `dataT` následovně:

```
dataT = [ [[1,2],[3,4],[6,7]] ]
```

Pro přidání druhé funkce do stejného grafu, stačí vložit další dvojrozměrně pole s novými hodnotami:

```
dataT = [
    [[1,2],[3,4],[6,7]],
    [[5,5],[6,6],[9,9]]
];
```

Pokud vyžadujeme u každé položky popisek s legendou, je třeba opět upravit vstupní data. Každá funkce je reprezentována asociativním polem s klíči `label` pro popis v legendě a `data` pro hodnoty, které budou vykresleny.

```
dataT = [
    { label: "legenda 1",  data: [[1,2],[3,4],[6,7]] },
    { label: "legenda 2",  data: [[5,5],[6,6],[9,9]] }
];
```

Skrývání grafů

Na základě hodnot v databázi jsou při načtení stránky vygenerovány checkboxy se jmény jednotlivých uzlů. Při změně hodnoty v některém checkboxu dojde k překreslení grafu tak, aby byly zobrazeny pouze vybrané uzly. Každému checkboxu bude přiřazeno jméno a id odpovídající hodnotě klíče do pole pro každý uzel.

Kvůli uživatelské přívětivosti je třeba upravit vstupní data tak, abychom byli schopni selektivně vybrat pouze vybrané datové řady. Dále je vhodné každému grafu předdefinovat barvu, kterou bude vykreslován. Pokud bychom tak neučinili, došlo by při překreslení grafů ke změně barev, což z uživatelského hlediska není akceptovatelné.

Výsledná datová struktura, kterou potřebujeme získat má tedy následující tvar.

```
dataT = {
    "id1": {
        label: "legenda 1",
        data: [[1,2],[3,4],[6,7]]
    },
    "id2": {
        label: "legenda 2",
        data: [[5,5],[6,6],[9,9]]
    }
};
```

Formátování grafu

Poslední parametr slouží k formátování os `x` a `y`. Můžeme vidět, že v aplikaci bude na osu `x` vynášen čas, je nutné nastavit formát osy x jako `xaxis:{mode: "time",tickLength:5}`. `Flot` očekává čas jako ve formátu JavaScriptu, tedy jako počet milisekund od 1. 1. 1970 00:00:00 UTC. Vzhledem k tomu, že v databázi jsou data uloženy ve formě sekund od téhož počátečního data, je třeba brát zřetel na následnou konverzi. Hodnota `tickLength` nastavuje vzdálenost mezi jednotlivými kótami v grafu. Knihovna `Flot` neumí definovat popisky pro jednotlivé osy. V práci byly popisky os implementovány pomocí css.

Pro lepší představu o tom, kdy byl daný uzel v dosahu, není výsledná křivka vyhlazena a každá hodnota je označena kolečkem. Můžeme tedy rozeznat intervaly, kdy uzel zasílal data pravidelně a kdy docházelo k výpadkům.

6.2.3 Textová reprezentace

Mimo vykreslování grafů je možné zdrojová data uložit do `csv` souboru. K tomu slouží tlačítko "Export source to dbOut.csv file", které se nachází ve spodní části levé nabídky (viz přílohu C.1). Je přítomna také volba počtu exportovaných řádků. Při zadání nekladné hodnoty signalizujeme, že chceme exportovat celou databázi. Musíme brát na zřetel, že export velkého množství řádků, zabere větší množství času. Výsledný soubor `dbOut.csv` bude po vygenerování nabídnut ke stažení.

Kapitola 7

Testování bezdrátové senzorové sítě

V následující kapitole budou diskutovány praktické zkušenosti s používáním senzorových uzlů a s chováním výsledné sítě. Důraz byl kladen na ověření funkčnosti dynamičnosti topologie, zjištění reálného dosahu uzlů a potenciální výdrže na baterii.

Dle zadání byla bezdrátová senzorová síť testována především v prostorách VUT FIT Brno. Dále testování probíhalo ve vnitřních i venkovních prostorách kolejí Pod Palackého vrchem v Brně.

7.1 Dynamičnost topologie

Volba plně dynamické topologie sítě byla diskutována v kapitole 4. Spoléháme tedy na to, že uzly pouze umístíme na požadované místo a spustíme. Sběr dat pak bude probíhat automaticky.

7.1.1 Rozmístění uzlů

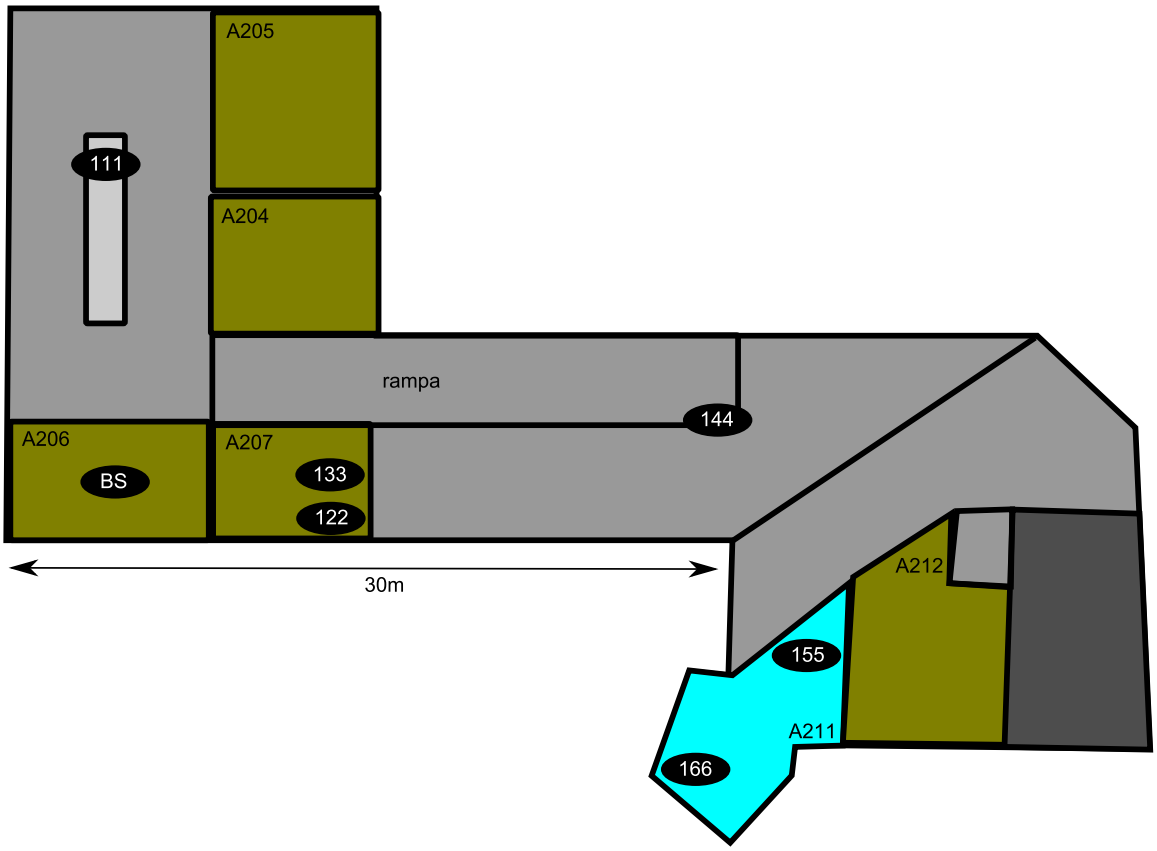
Za účelem testování sítě byla vybrána místa, která umožní testovat dynamické vlastnosti. Uzly tedy mohou data do základnové stanice dopravit více než jednou cestou a my budeme moci simulovat výpadky důležitých uzlů a sledovat, zda dojde k znovuustavení topologie.

Na obrázku 7.1 vidíme rozmístění uzlů v areálu FIT VUT. Uzly jsou reprezentovány bílým číslem na černém poli. Na obrázku 7.2 je znázorněn dosah jednotlivých uzlů. Spojnice značí, že spolu uzly mohou komunikovat. Vidíme, že uzly 144, 155 a 166 jsou závislé na funkčnosti alespoň jednoho uzlu z dvojice 122, 133.

7.1.2 Simulace výpadků v síti

Pokud má uzel při spuštění možnost výběru z několika stejně kvalitních rodičovských uzlů, je věcí náhody, na který začne směrovat data. Vzhledem k možnostem, které se při vymýšlení topologie naskytly, má při funkčnosti všech uzlů možnost volby pouze uzel 155. Ten jako jediný bude mít k dispozici dvě stejně kvalitní cesty k základnové stanici, všem ostatním bude nabídnuta pouze jedna minimální varianta.

V testovací síti jsou nejvytíženější uzly 122 a 133 a simulace výpadku byla prováděna právě na nich. Za ideálních podmínek jsou totiž po ustavení nové topologie vzájemně nahraditelné. Pokud jsou všechny uzly zapnuté, jsou vždy ustaveny cesty `144 -> 133 -> BS`



Obrázek 7.1: Mapa rozmístění uzlů v areálu FIT VUT.

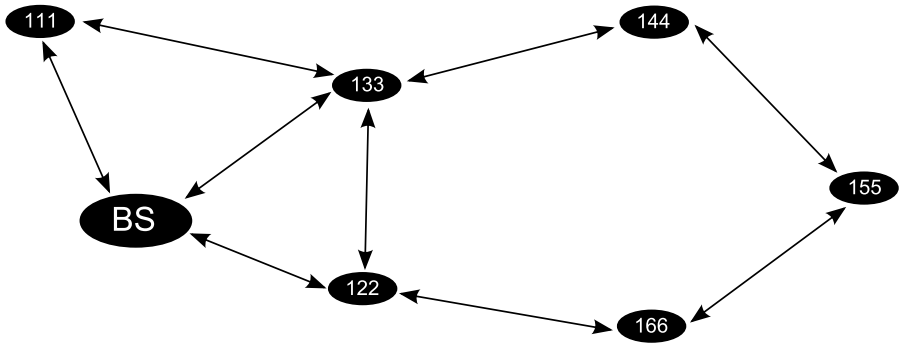
a `166 -> 122 -> BS` přičemž uzel 155 je na začátku jedné nebo druhé z cest. Po výpadku jednoho z uzlů dojde k ustavení nové topologie, která použije pouze jeden přístupový bod. Topologie po výpadku uzlu 122 je ukázána na obrázku 7.3.

Vzhledem k poměrně krátce nastavenému intervalu, po kterém se vynuluje směrovací tabulka, byla síť schopna znovu ustavit topologii po přibližně 10 sekundách. Čím delší bychom zvolili interval, tím déle by ustavení nové topologie trvalo. Jelikož se vždy aktualizovaly pouze dva nejvzdálenější uzly, bylo nutné provést maximálně dva dotazy – jeden požadavek na okolí pro první uzel a dva požadavky pro druhý.

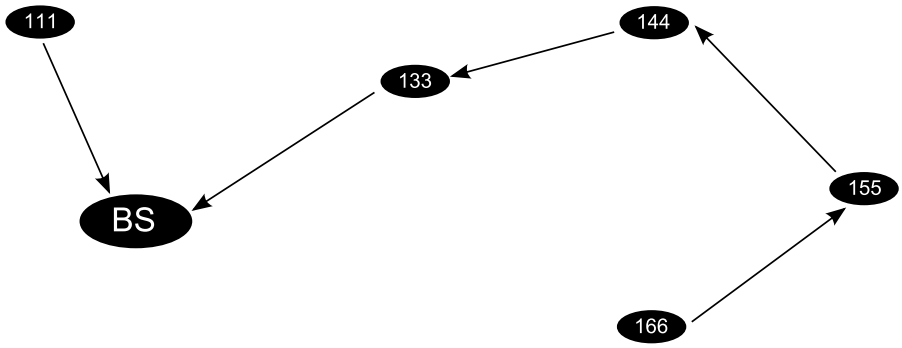
V nejhorším případě musí nejvzdálenější uzel provést `m` vysílání směrového paketu, kde `m` je počet uzlů mezi ním a základnovou stanicí. Obecně však dojde k ustavení sítě daleko dříve, jelikož zmíněný extrémní stav by nastal pouze ve chvíli, kdy by byly všechny uzly spuštěny zároveň a tudíž by v jeden okamžik mazaly své směrovací tabulky. Neexistuje však žádný rozumný způsob ani důvod, proč bychom síť tímto způsobem konstruovali. Při běžném zapnutí uzlů bude docházet vždy pouze k dílčím úpravám topologie.

7.2 Reálný dosah uzlů

Výrobcem udávaný dosah rádiových modulů jednotlivých uzlů můžeme vidět v tabulce 7.1.



Obrázek 7.2: Dosah jednotlivých uzlů. Uzly, které jsou spojeny šipkou, spolu mohou přímo komunikovat.



Obrázek 7.3: Topologie po výpadku uzlu 122.

7.2.1 Vnitřní dosah

Reálný dosah obou typů uzlů byl testován ve vnitřních prostorách, uzly IRIS pak byly testovány i mimo budovu. Jak můžeme vyčíst z obrázků 7.1 a 7.2 maximální dosah mezi uzly, které na sebe vidí, je kolem 15 metrů. Vzhledem k tomu, že nebylo k dispozici žádné přesné měřicí zařízení, je údaj o vzdálenosti pouze orientační; jelikož nepotřebujeme znát přesnou vzdálenost, nám tato přibližná hodnota dostačuje.

Uzly IRIS i MICAz měly při avizovaném rozložení uzlů víceméně totožný dosah. Papírová výhoda uzlů IRIS se tedy v prostorách FIT VUT neprojevila. Je to zřejmě dáno tím, že tyto prostory jsou pro provozování bezdrátové senzorové sítě velice nevhodné. Tloušťka stěn v areálu se blíží k jednomu metru a Wi-Fi síť, která pokrývá celou budovu, signál uzlů rušila. Nutno podotknout, že v budově je i problém s Wi-Fi signálem, i když jsou přístupové body vybaveny o poznání většími anténami než uzly senzorové sítě.

Uzly IRIS byly testovány i v budovách kolejí Pod Palackého vrchem. Ve zdejších prostorách byl dosah přibližně 45 metrů. Překážku mezi dvěma uzly tvořily však pouze dvoje prosklené dveře. Před samotnou aplikací uzlů ve vytipované lokalitě bude tedy nutná prvotní analýza okolních vlivů, která nám lépe ukáže, kolik uzlů na pokrytí daného prostoru budeme potřebovat.

	IRIS	MICAz
Dosah venku (m)	>300	75-100
Dosah uvnitř (m)	>50	20-30

Tabulka 7.1: Dosah uzlů IRIS a MICAz dle [19] a [20].

7.2.2 Venkovní dosah

Oproti hodnotám udávaným výrobcem byl venkovní dosah uzlů IRIS (uzly MICAz nebyly pro venkovní testování k dispozici) třetinový. Základnová stanice byla schopna přijímat data od uzlů vzdálených 100 metrů. Dosah 300 metrů udávaný výrobcem by teoreticky bylo možné dosáhnout v ideálním prostředí – tedy bez překážek a bez okolního rušení.

7.3 Výdrž baterií

Jedním z hlavních požadavků na senzorovou síť je schopnost uzlů fungovat dlouhou dobu bez potřeby výměny baterií. Uzly IRIS i MICAz jsou uzpůsobeny úspornému režimu. Pokud nejsou vykonávány úkony spojené s vysíláním zpráv případně čtením dat ze senzorů, je uzel uveden do úsporného režimu, ve kterém vykazuje minimální spotřebu.

Vzhledem k očekávané vysoké výdrži uzlů, byl zvolen zátěžový test, na základě kterého, byla potenciální výdrž odhadnuta. Testovací uzel byl naprogramován, aby každou sekundu zasílal informace o teplotě a intenzitě osvětlení. K vybití baterií došlo v horizontu osmi dní. Pokud bychom tedy interval prodloužili na interval 30 sekund, zvýšili bychom výdrž teoreticky 30 krát. Tím se dostaneme na 8 měsíců teoretické výdrže. Musíme však počítat s tím, že klimatické podmínky mají neblahý vliv na životnost akumulátorů. Při monitorování venkovních prostor v zimním období, bude výdrž baterií nižší.

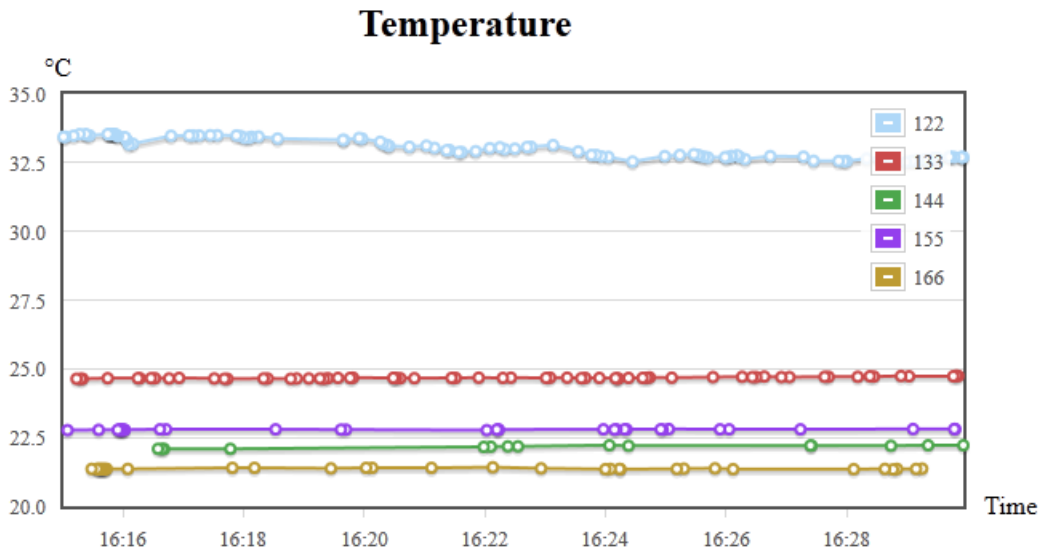
7.4 Naměřené výsledky

Testování v budově FIT VUT trvalo 24 hodin a pomohlo blíže objasnit chování senzorové sítě v reálném prostředí. Při vyhodnocování naměřených dat byl kladen důraz na rozpoznání faktorů, které ovlivňují kvalitu přenosu dat.

Na obrázku 7.4 můžeme vidět graf průběhu teploty v době 16:15 - 16:30. Tou dobou se v místech testování sítě pohybovalo větší množství lidí. Zaměříme se na uzel číslo `144` (v grafu vyznačen zelenou barvou), který byl okolím ovlivněn nejvíce. Mezi uzlem `144` a uzlem `133`, na který se směrovala data, se nacházelo několik lidí, kteří používali notebooky s wi-fi připojením k internetu. V grafu si můžeme všimnout relativně velkých výpadků.

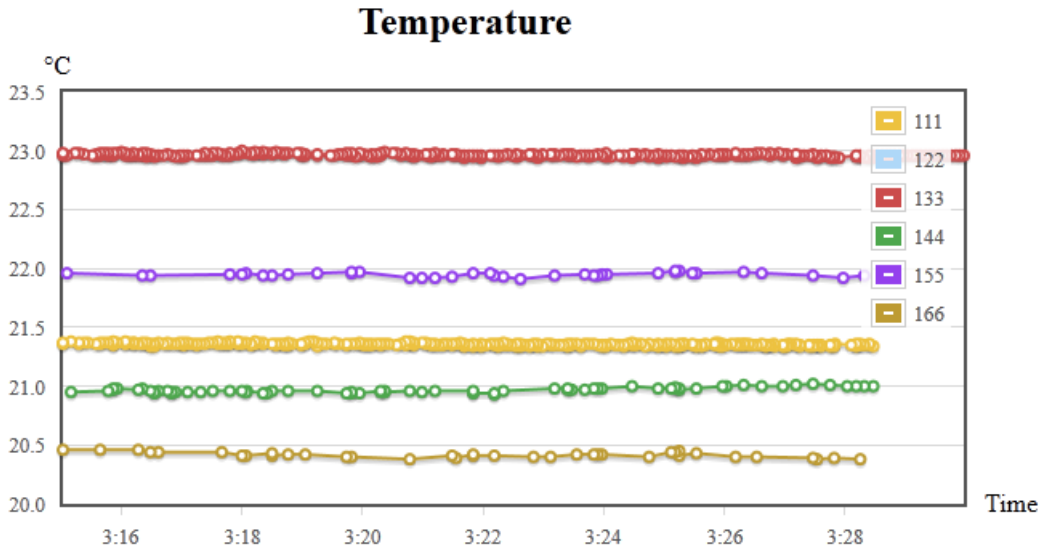
Obrázek 7.5 pak reprezentuje hodnoty nasbírané za stejný časový úsek. Rozdíl je ten, že data pro obrázek 7.5 byla nasbírána v intervalu 3:15 - 3:30, tedy v době, kdy se v prostorách testování nikdo nepohyboval. Můžeme si všimnout rapidního rozdílu v počtu doručených hodnot.

Dále byl testován vliv volby umístění uzlů na počet přijatých dat. Pokud bychom měli síť navrženou tak, že by určité uzly byly přetěžovány, bude docházet ke ztrátám paketů, protože rádiové moduly nejsou schopny zpracovat neomezené množství požadavků. Přetížení jednoho uzlu bylo simulováno vypnutím uzlu `122`. Uzly `144`, `155` a `166` tedy musely zasílat data prostřednictvím uzlu `133`. Vzhledem k vysoké frekvenci zasálání hodnot a obnovy směrové tabulky, byl uzel 133 přetížen a docházelo k relativně velkým datovým ztrátám.

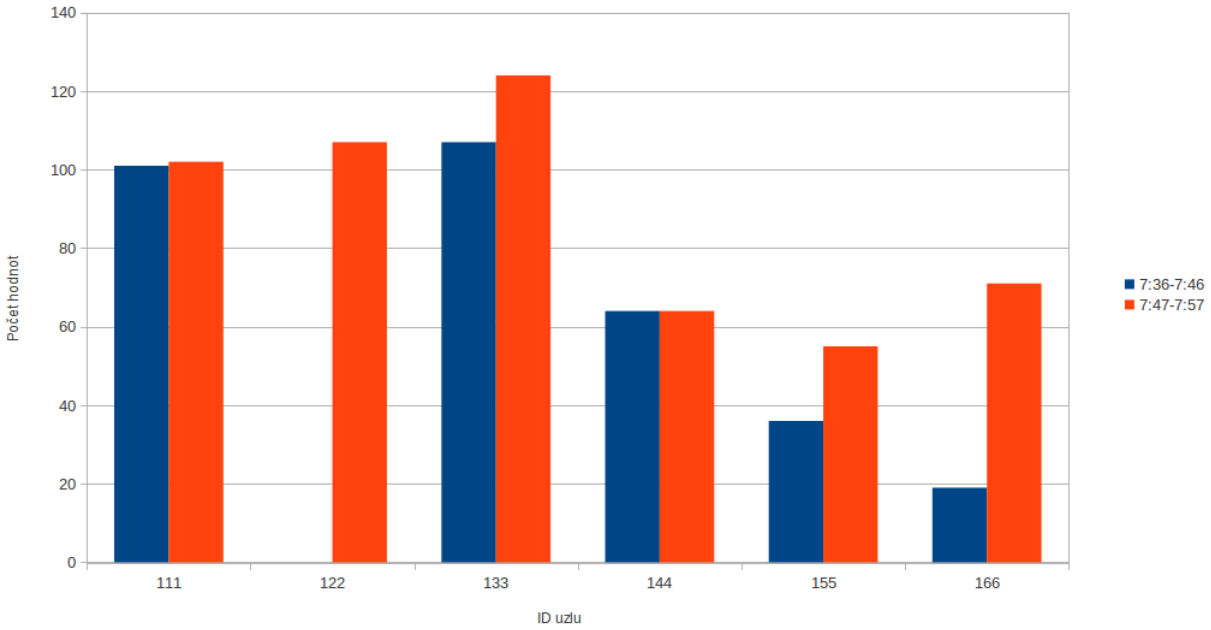


Obrázek 7.4: Naměřená teplota v době 16:15 - 16:30.

Ztráty byly také způsobeny špatným signálem mezi danými uzly – uzel `122` poskytoval kvalitnější cestu. Na obrázku 7.6 je znázorněno množství paketů přijatých základnovou stanicí v desetiminutových intervalech. V prvním intervalu byl uzel `122` vypnutý, ve druhém byl součástí sítě. Je jasně vidět značné zlepšení v doručování paketů od odlehlých uzlů.



Obrázek 7.5: Naměřená teplota v době 03:15 - 03:30.



Obrázek 7.6: Vliv funkčnosti důležitého uzlu na množství doručených paketů. Modré sloupce reprezentují počet paketů přijatých základnovou stanicí od jednotlivých uzlů v době 7:36-7:46, červené sloupce pak interval 7:47-7:57. Uzel 122 byl zapnut v čase 7:47. Můžeme si všimnout výrazného zlepšení v doručení paketů od odlehlých uzlů.

Kapitola 8

Závěr

Cílem práce bylo implementovat řešení pro monitorování budovy pomocí bezdrátové senzorové sítě s částečně dynamickou topologií pro operační systém TinyOS za použití uzlů IRIS a MICAz. Důraz byl kladen na schopnost uzlů ustavit plně dynamickou topologii sítě a následnou odolnost vůči výpadkům jednotlivých uzlů.

V práci byly popsány různé možnosti použití bezdrátových senzorových sítí a také obecné vlastnosti komunikačních protokolů pro tyto sítě. Podrobněji byl popsán protokol systému TinyOS Collection a obecné fungování protokolu Collection Tree Protocol, ze kterých vychází návrh výsledného protokolu.

Navržený komunikační protokol vychází z principů protokolu Collection Tree Protocol, který byl pro směrování dat v senzorových sítích navržen. Základní myšlenkou je vytvoření pomyslného stromu, kde kořenem je základnová stanice a listy jsou jednotlivé uzly. Směrování pak probíhá vždy na rodiče, který je v hierarchii nad daným uzlem. Oproti řešení v TinyOS bylo zvoleno nepotvrzované spojení. Ušetřila se tak polovina komunikačních zpráv. Navíc přibyly zprávy synchronizační, které jsou vysílány v nastavitelných intervalech.

Jak již bylo zmíněno, topologie výsledné sítě je plně dynamická, uzly tedy stačí zapnout a umístit na požadované místo. Po ustavení topologie začnou uzly posílat data do základnové stanice. Požadované veličiny jsou teplota a intenzita osvětlení.

Zřejmě kvůli chybě v systému TinyOS se nepodařilo implementovat převod naměřených hodnot intenzity osvětlení na jednotku Lux. Bylo to způsobeno neschopností daného rozhraní přečíst data ze senzoru. Po vydání záplaty, která tuto chybu opraví, by mělo fungovat vše správně.

Nasbíraná data jsou ukládána do databáze typu SQLite. Pro reprezentaci nasbíraných dat byla implementována webová aplikace, která vykresluje grafické průběhy naměřených hodnot. Dále je možné data exportovat do textového souboru csv.

Testování řešení probíhalo převážně v budově VUT FIT. Byla ověřena schopnost sítě dynamicky ustavit topologii po spuštění uzlů a také po výpadku jednoho z klíčových uzlů.

Reálný dosah uzlů v testovacích prostorách dosahoval 15 metrů. Tato hodnota je třetinová vzhledem k dosahu udávanému výrobcem. Bylo to dáno vlivy okolního prostředí – metr tlusté zdi a velké množství okolních bezdrátových sítí, měly na dosah uzlů špatný vliv. Uzly byly testovány i v budovách kolejí Pod Palackého vrchem a zde byl dosah přibližně 45 metrů, což odpovídá údajům výrobce. Venkovní dosah uzlů byl 100 metrů, což bylo oproti hodnotám udávaným výrobcem třikrát méně. Musíme brát zřetel na to, že podmínky pro venkovní měření nebyly ideální.

Implementovaná aplikace je schopna ustavit dynamickou topologii v bezdrátové senzorové sítí a je schopna reagovat na výpadky jednotlivých uzlů. Bylo implementováno webové

rozhraní, které graficky zobrazuje průběh naměřených hodnot. Toto je také schopno data exportovat do textového formátu csv.

Komunikační protokol ve své první verzi poskytuje základní řešení pro bezdrátovou senzorovou síť. Další verze protokolu by měla obsahovat schopnost uzlů dynamicky měnit interval zasílání směrových rámců na základě kvality signálu.

Webové rozhraní poskytuje základní uživatelské funkce. Bylo by vhodné implementovat vhodnou grafickou reprezentaci stavu databáze – tj. jaké období daná databáze pokrývá. Pokud by byla aplikace nasazena, bylo by vhodné, vzhledem k rychlosti i archivaci dat, použít pro každý týden odlišný databázový soubor. Dále by bylo vhodné poskytnout ve webovém rozhraní možnost volby délky intervalu zasílání naměřených hodnot od uzlů.

Literatura

[1] Eady, F.: *Hands-On ZigBee: Implementing 802.15.4 with Microcontrollers*. Elsevier, 2007, iSBN 978-0-12-370887-8.

[2] Fonseca, R.; Gnawali, O.; Jamieson, K.; aj.: Collection protocol. [online], 2006 [cit. 2012-01-07].
URL www.tinyos.net/tinyos-2.x/doc/html/tep119.html

[3] Fonseca, R.; Gnawali, O.; Jamieson, K.; aj.: The Collection Tree Protocol. [online], 2006 [cit. 2012-01-07].
URL www.tinyos.net/tinyos-2.x/doc/html/tep123.html

[4] Gislason, D.: *ZigBee Wireless Networking*. Elsevier, 2008, iSBN 978-0-7506-8597-9.

[5] Gnawali, O.; Fonseca, R.; Jamieson, K.; aj.: Collection Tree Protocol. [online], [cit. 2012-01-07].
URL sing.stanford.edu/pubs/sing-09-01.pdf

[6] Greenstein, B.; Levis, P.: Serial Communication. [online], 2006 [cit. 2012-04-25].
URL www.tinyos.net/tinyos-2.x/doc/html/tep113.html

[7] Levis, P.: message_t. [online], 2006 [cit. 2012-04-25].
URL www.tinyos.net/tinyos-2.x/doc/html/tep111.html

[8] Levis, P.; Lee, N.: TOSSIM: A Simulator for TinyOS Networks. [online], 2003.
URL www.cs.berkeley.edu/~pal/pubs/nido.pdf

[9] Li, Y.; Thai, M. T.; Wu, W.: *Wireless Sensor Networks and Applications*. Springer Science+Business Media, 2008, iSBN 978-0-387-49591-0.

[10] Raghavendra, C. S.; Sivalingam, K. M.; Znati, T.: *Wireless sensor networks*. Springer Science+Business Media, 2004, iSBN 1-4020-7883-8.

[11] Trchalík, R.: Senzorové sítě a směrovací protokoly v bezdrátových senzorových sítích. [online], 2010 [cit. 2012-01-07].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/PDS-IT/lectures/pds-06-senzor.pdf?cid=7539

[12] WWW stránky: MEMSnet. [online], 2000 [cit. 2012-05-11].
URL www.memsnet.org/mems/what_is.html

[13] WWW stránky: Getting Started With TinyOS. [online], 2010 [cit. 2012-01-07].
URL docs.tinyos.net/tinywiki/index.php/Getting_Started_with_TinyOS

[14] WWW stránky: Datasheet SHT1x Humidity and Temperature Sensor IC. [online], 2011 [cit. 2012-04-30].
URL www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf

[15] WWW stránky: Flot - Attractive JavaScript plotting for jQuery. [online], 2011 [cit. 2012-04-30].
URL code.google.com/p/flot/

[16] WWW stránky: TAOS an austriamicrosystem subsidiary. [online], 2011 [cit. 2012-04-30].
URL www.taosinc.com/ProductDetails.aspx?id=133

[17] WWW stránky: PHP Data Objects. [online], 2012 [cit. 2012-04-30].
URL php.net/manual/en/book.pdo.php

[18] WWW stránky: Lux. [online], 2012 [cit. 2012-05-11].
URL en.wikipedia.org/wiki/Lux

[19] WWW stránky: IRIS Wireless Measurement System. [online], [cit. 2012-01-03].
URL www.dinesgroup.org/projects/images/pdf_files/iris_datasheet.pdf

[20] WWW stránky: MICAz Wireless Measurement System. [online], [cit. 2012-01-03].
URL www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[21] WWW stránky: Constant Field Values. [online], [cit. 2012-04-20].
URL www.tinyos.net/dist-2.0.0/tinyos-2.x/doc/javadoc/constant-values.html

[22] WWW stránky: MTS420/400. [online], [cit. 2012-05-11].
URL www.instrumentation.it/main/pdf/crossbow/MTS400-420_Datasheet.pdf

Příloha A

Obsah CD

- `/`
  - `src`
    - `BaseStation` ..... Zdrojové kódy pro základnovou stanici
    - `moteCommunication` ..... Zdrojové kódy pro uzly
    - `python` ..... Zdrojové kódy pro získávání dat z uzlů
    - `web` ..... Zdrojové kódy webového rozhraní
  - `technicka zprava` ..... Pdf soubor s technickou zprávou
    - `tex` ..... Zdrojové kódy technické zprávy
  - `virtual kubuntos` ..... Obraz virtuálního počítače s TinyOS

Příloha B

Potřebný software, instalace, spuštění

Následující část přílohy předpokládá základní znalost operačního systému TinyOS a práci s uzly jako jsou IRIS nebo MICAz.

B.1 Nutná softwarová výbava

V této sekci budou popsány knihovny a služby, které je třeba nainstalovat pro správné fungování aplikace. Na přiloženém DVD je ve složce `virtual kubuntos` uložen obraz virtuálního počítače, ve kterém jsou všechny potřebné náležitosti nainstalovány a zdrojové soubory jsou umístěny ve složce `/home/fit/src`. Kořenový adresář pro apache je pak pro jednoduchost nastaven do složky `/home/fit/src/web`. Uživatelské jméno je `fit`, heslo pak `kubuntos`.
Následuje seznam balíčků, které jsou potřeba pro fungování práce:

- TinyOS verze 2.1.1,
- Python verze 2.7,
- PHP verze 5.3.16-13,
- Apache verze 2.2.20,
- podpora databázového systému SQLite pro PHP.

B.2 Spuštění a překlad

B.2.1 Instalace programu do uzlů

Kód pro uzel IRIS nahrajeme do uzlu příkazem

```
make iris install,<node_id> mib520,/dev/ttyUSB0
```

kde `node_id` je číslo daného uzlu. Při instalaci kódu do základnové stanice musí být `node_id=0`.

B.2.2 Spuštění celého systému

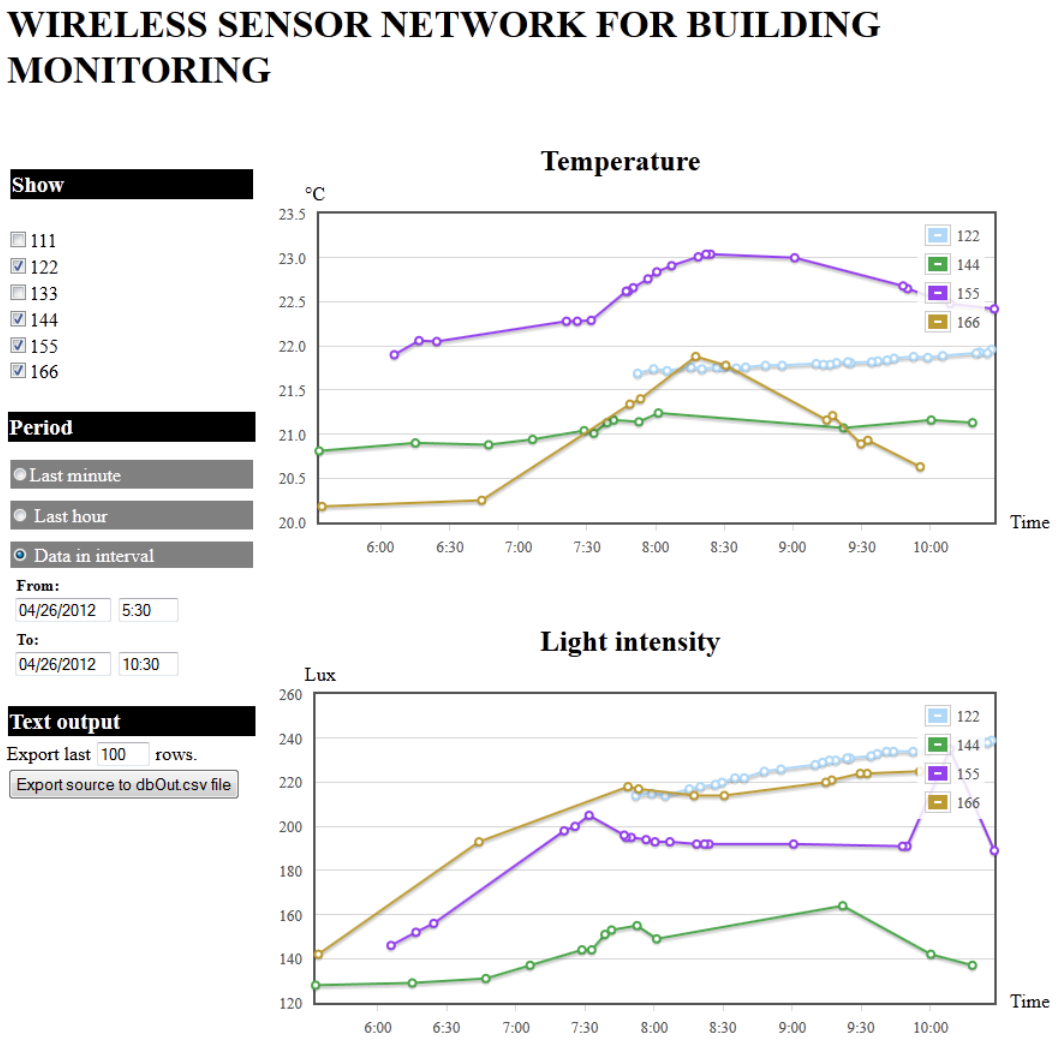
Pro spuštění celého systému je třeba provést několik postupných kroků. Postup uvádíme pro řešení v přiloženém virtuálním počítači. Řešení na jiném stroji se může lišit - bude záležet na nastavení prostředí. Je třeba vykonat následující kroky:

1. Nahrát program do uzlů a základnové stanice. Spustit uzly a základnovou stanici. Základnová stanice musí být připojena k počítači.
2. Povolit USB zařízení reprezentující základnovou stanici (Crossbow MIB520CA) ve virtuálním prostředí.
3. Spustit skript /home/fit/python/client.py. Bude číst data ze sériového portu. Pokud spuštění skončí chybou stačí počkat, než se zařízení inicializuje a poté spustit skript znova.
4. Spustit skript /home/fit/python/server.py. Bude obnovovat data v databázi.

Nyní můžeme v prohlížeči otevřít stránku `localhost/index.php`, kde bude zobrazeno webové rozhraní.

Příloha C

Vzhled webového rozhraní



Obrázek C.1: Webové rozhraní aplikace.

Příloha D

Simulace pomocí nástroje TOSSIM a její výstup

D.1 Ukázka zdrojového kódu

Následující kód vytvoří dva uzly, které se vidí s útlumem -34db. Bude přidán šum okolí ze souboru `meyer-heavy.txt`, který je k dispozici přímo na stránkách TOSSIMu.

test.py

```
#!/usr/bin/python2.7

from TOSSIM import *
import sys
from tinyos.tossim.TossimApp import *

t = Tossim([])
r = t.radio()
t.addChannel("Boot", sys.stdout)

r.add(1,2,-34.0)
r.add(2,1,-34.0)

noise = open("meyer-heavy.txt","r")
lines = noise.readlines()
for line in lines:
    str = line.strip()
    if (str != ""):
        val = int(str)
        for i in range (0,2):
            t.getNode(i).addNoiseTraceReading(val)

for i in range (0,2):
    t.getNode(i).createNoiseModel();

for i in range (0,2):
    t.getNode(i).bootAtTime(i*2351217 + 2354399)
```

```
28
29  while t.runNextEvent():
30     # print v.getData()
31      pass
```

D.2 Výstup simulace

Ukázka části výstupu simulace.

simulation output

```
1  noise for  0
2  noise for  1
3
4  DEBUG (0): nabootovano
5  DEBUG (1): nabootovano
6  DEBUG (1): nove routing info: budu posilat uzlu 0 a mam etx 1
7  DEBUG (1): rdy, info=0, thl=0,etx=1, origin=1 seqno=5
8  DEBUG (0): prijato 1 od 1 puvodne od 1 v siti byl 0
```