VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝVOJ APLIKACÍ PRO BEZDRÁTOVÉ SENZOROVÉ SÍTĚ POMOCÍ NÁSTROJE TOSSIM

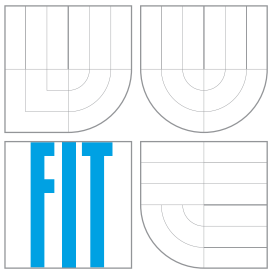
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŽÍDEK PETR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝVOJ APLIKACÍ PRO BEZDRÁTOVÉ SENZOROVÉ SÍTĚ POMOCÍ NÁSTROJE TOSSIM

APPLICATION DEVELOPMENT FOR WIRELESS SENSOR NETWORKS USING TOSSIM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŽÍDEK PETR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZBOŘIL FRANTIŠEK, Ph.D.

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Žídek Petr**
Obor: Informační technologie
Téma: **Vývoj aplikací pro bezdrátové senzorové sítě pomocí nástroje TOSSIM**
Kategorie: Počítačové sítě

Pokyny:

1. Nastudujte problematikou bezdrátových senzorových sítí a s implementačními nástroji založenými na NesC a TinyOS.
2. Seznamte se s nástrojem TOSSIM pro vývoj aplikací pro bezdrátové senzorové sítě.
3. Vytvořte pomocí tohoto nástroje jednoduchou aplikaci na směrování paketů podle metody popsané u sítí XMesh.
4. Vyzkoušejte a demonstrujte správné fungování této implementace.
5. V textu dále popište použití nástroje TOSSIM, jeho ovládání, způsob implementací a případně upozorněte na nedostatky nebo funkční chyby.

Literatura:

- Haenselmann, T.: Sensor Networks, GFDL Wireless Sensor Network textbook, 2006
- Levis, P.: TinyOS/nesC Programming Reference Manual, 2006
- Levis, P., Lee, N.: TOSSIM: A Simulator for TinyOS Networks, User Manual, 2003

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

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

Vedoucí: **Zbořil František, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

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

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Petr Žídek**
Id studenta: 78651
Bytem: Oleška 78, 281 63 Kostelec nad Černými Lesy
Narozen: 22. 08. 1985, Kolín
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Vývoj aplikací pro bezdrátové senzorové sítě pomocí nástroje TOSSIM
Vedoucí/školitel VŠKP: Zbořil František, Ing., Ph.D.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato práce se zabývá technologií bezdrátových senzorových sítí spolu s implementačními nástroji založenými na programovacím jazyku NesC a operačním systému TinyOS. Popisuje, simuluje a implementuje dva základní síťové protokoly Dissemination a Collection. V neposlední řadě také poskytuje postup pro praktickou instalaci aplikací do MicaZ ůzlů pomocí TinyOS.

Klíčová slova

bezdrátové senzorové sítě, basestation, uzel, TinyOS, nesC, TOSSIM, MicaZ, Dissemination, Collection

Abstract

This bachelors thesis is focused on wireless sensor networks implementation tools based on programming language NesC solution specific operation system TinyOS. It describes, simulates and implements two crutial net protocols Dissemination and Collection in detail. Last but not least it delinates method how to practically install applications into MicaZ nodes with support of TinyOS.

Keywords

wireless sensor networks, basestation, node, TinyOS, nesC, TOSSIM, MicaZ, Dissemination, Collection

Citace

Žídek Petr: Vývoj aplikací pro bezdrátové senzorové sítě pomocí nástroje TOSSIM, bakalářská práce, Brno, FIT VUT v Brně, 2008

Vývoj aplikací pro bezdrátové senzorové sítě pomocí nástroje TOSSIM

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbořila Františka, Ph.D.

.....
Žídek Petr
13. května 2008

Poděkování

Tímto bych chtěl poděkovat Ing. Zbořilu Františkovi, Ph.D. za umožnění praktických zkušeností s těmito nově vznikajícími technologiemi. Dále pak mým rodičům a přítelkyni za jejich trpělivost a ochotu pomoci.

Obsah

Kapitola 1

Úvod

V dnešní době jsou již relativně dlouhou dobu známa zařízení vybavené počítači. Různorodost těchto technologií umožnila posun k masově vyráběným inteligentním senzorům mající schopnost mezi sebou komunikovat a sbírat data o okolních podmínkách. Bezdrátové senzorové sítě, nebo-li *wireless sensor networks* (WSN), propojují tyto malá zařízení do bezdrátové sítě. Vše prostupující síťové technologie tak dávají bezdrátovým senzorovým sítím nové měřítko pro široké spektrum využití. Neustálá miniaturizace elektronických prvků představuje pro WSN nová pole pro své uplatnění, například v domácí automatizaci, monitorování lidského zdraví nebo monitorování životního prostředí.

Tato práce si klade za cíl nastínit čtenáři problematiku bezdrátových senzorových sítí s spolu s některými implementačními nástroji vhodnými pro jejich programování a simulaci. Stručně popsat, co to jsou bezdrátové senzorové sítě, z čeho se skládají a jaké mají uplatnění v praktickém životě. Dále pak poukázat na některé rysy operačního systému TinyOS, na kterém jsou založeny aplikace v senzorových sítích a upozornit na některé odlišnosti starší, dosud používané, verze TinyOS od verze nové. Seznámit čtenáře se základním konceptem programovacího jazyka NesC, který je používán pro programování aplikací.

V rámci tohoto tématu demonstrovat komunikaci v senzorových sítích a ukázat funkčnost dvou základních síťových protokolů Collection a Dissemination. Představit možnosti simulace na příkladu protokolu Dissemination s využitím simulační knihovny TOSSIM. V neposlední řadě také ukázat praktický postup vytvoření malé bezdrátové senzorové sítě s využitím MicaZ technologie.

Kapitola 2

Bezdrátové senzorové sítě

2.1 Stručný popis

Vznik této technologie je spojen s vývojem aplikací pro vojenské použití. Původní projekt americké armády SOSUS (*Sound Surveillance System*) pro protiponorkové zbraně, datující se k roku 1949 a následný IUSS (*Integrated Undersea Surveillance System*), lze považovat za počátek vzniku bezdrátových senzorových sítí. Typ těchto projektů definoval základní povahu technologie, tedy malé autonomní zařízení mající schopnost mezi sebou komunikovat a sbírat data z okolního prostředí. V roce 2001 následovaly projekty Smart Dust a projekt NEST, které tyto základní vize dále více obohatily.

Poté se myšlenky senzorových sítí chopila univerzita UC Berkeley, která vyrobila sérii senzorů nazvaných *Mica* a vytvořila pro ně open - source operační systém nazvaný *TinyOS*. Tím došlo k masovějšímu rozšíření této technologie mezi širší odbornou veřejnost.

Bezdrátové senzorové sítě jsou tedy sítě založené na bezdrátovém propojení autonomních zařízení. Tato zařízení jsou nazývána *uzly* a jsou vybavena radiopřijímačem nebo jiným bezdrátovým zařízením pro komunikaci. Mohou také obsahovat senzory pro snímání různorodých fyzikálních veličin, např. teploty, tlaku, pohybu nebo hluku. Tato zařízení mají jistá omezení ve formě malé rychlosti zpracování informací, úložné kapacity a komunikační šířky přenosového pásma.

Bezdrátová senzorová síť je dále tvořena zařízením nazývaným *basestation*, které slouží ke sběru dat ze sítě. Basestation je připojena pomocí některého ze standardních rozhraní (USB, RS232) k PC, na kterém může být např. spuštěn program pro analýzu dat nasbíraných ze sítě.

Senzorová síť může obsahovat tisíce uzlů a pokrývat tak velkou geografickou plochu. Uzly v ní obsažené jsou mobilní a mohou být náhodně rozmístěny v monitorované oblasti.

Ve většině případů je nutné, aby síť operovala po dlouhou dobu. Proto jsou tato bezdrátová zařízení napájena bateriemi, solárními články pro získání energie nebo kombinací obojí. Po většinu doby provozu je z důvodu minimalizace spotřeby energie většina komponent ve vypnutém stavu.

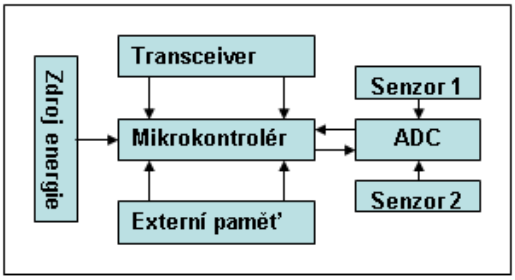
Postupem času si, i jako jiné technologie (*např. GPS*), našly uplatnění v každodenním životě. V současné době lze vidět aplikaci těchto technologií v oblastech monitorování životního prostředí, lidského zdraví nebo automatizace v domácnostech. Cena, která je odvozována od velikosti sítě a použitých senzorů, představuje jeden z důležitých faktorů, ovlivňující vývoj těchto technologií. Z toho vyplývá, že je pro masové rozšíření senzorových sítí důležitá, a proto je současným trendem cenu neustále snižovat.

Vizí senzorových sítí budoucnosti [6] je vytvoření permanentně fungujících uzlů, které dokáží využívat okolní energie pro jejich průběžné dobíjení.

2.2 Senzorový uzel

Počátečním bodem pro návrh této komponenty bylo kvalitní pozorování požadavků na bezdrátové senzorové sítě. Velký důraz byl kladen zejména na malé rozměry uzlů, malou spotřebu energie a dobré přizpůsobení okolním podmínkám. Senzorový uzel, někdy též nazýván *mote* je mikro-elektronické zařízení, které je schopno vykonávat běh programu, provádět sběr informací o okolních podmínkách a komunikovat s okolními uzly v síti. Obrázek 2.1 znázorňuje strukturu základních komponent a obrázek 2.2 praktickou ukázku senzorového uzlu.

Mezi základní komponenty patří [17] mikrokontrolér, transceiver, externí paměť, zdroj energie a senzory.



Obrázek 2.1: struktura senzoru

Důležitou součástí senzorového uzlu jsou **senzory**, které určují typ snímaných dat z okolního prostředí, jimiž mohou být např. teplota nebo tlak. Data jsou nejprve snímaná analogově a poté převedena analogově digitálním převodníkem (*ADC*) a uložena pro pozdější zpracování. Senzory lze rozdělit do několika skupin v závislosti na veličině kterou snímají:

- tepelné,
- elektromagnetické,
- mechanické,
- chemické,
- světelné,
- zvukové,
- ostatní.

Cílem **mikrokontroléru** je vykonávat procesy a řídit činnost ostatních komponent. Jejich velkou výhodou je malá spotřeba v řádech mW. Protože lze část mikrokontroléru vypnout, lze spotřebu snížit až řádově na nW, což je ideální pro dlouhou dobu provozu. Dalšími vlastnostmi jsou dobrá programovatelnost, flexibilita, malé rozměry a v neposlední

řadě také nízká cena. Mezi nevýhody lze řadit malou rychlost v řádu jednotek MHz, která ovšem pro tyto účely dostačuje. Namísto mikrokotrolérů lze využít digitální signální procesory (*DSP*), programovatelná pole nebo obecné desktopové procesory, které mají také své pro a proti, ale použití mikrokotroléru se zdá být nejefektivnější.

Transceiver je zařízení, které obsahuje vysílač a přijímač radiových vln. Tento druh záření je velice vhodný pro broadcastové vysílání, na rozdíl od infračervených nebo laserových vln, které lze také použít. Komunikace je možná mezi uzly navzájem nebo uzly a basestation. Ke své činnosti používá přenosového pásma v rozmezí 433 MHz až 2.4 GHz a umožňuje několik módů, jimiž jsou: vysílání, příjem, idle a režim spánku.

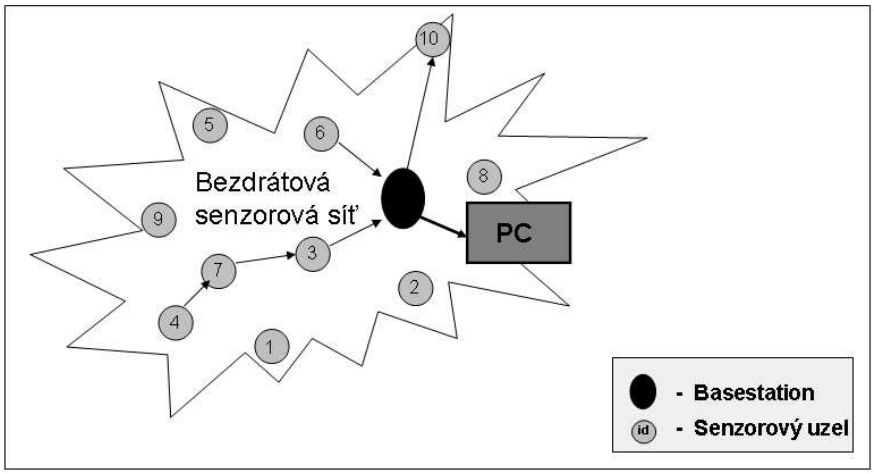
Nejčastěji používanou **externí pamětí** je paměť typu flash. Je to dáno její malou energetickou náročností, velkou kapacitou a samozřejmě také nízkou cenou.

Další důležitou součástí každého senzorového uzlu je **zdroj energie**, který je využíván všemi komponentami uzlu. Největší nároky na spotřebu energie jsou vynaloženy při komunikaci zařízení. Pro napájení se nejčastěji používají tužkové baterie.

Obrázek 2.2: Senzorový uzel (MicaZ platforma)

2.3 Senzorová síť

Typická senzorová síť tvoří bezdrátovou *ad-hoc* síť, kde každý senzor podporuje *multihop* směrovací algoritmus. Ad-hoc je síť, kde si jsou všechna zařízení rovna (*peer-to-peer*). Příkladem tohoto typu sítě může být například propojení několika notebooků přes bezdrátovou síťovou kartu(*wifi*). Ukázka multihop komunikace je ilustrována na obr. 2.3, který znázorňuje bezdrátovou senzorovou síť obsahující několik uzlů a jednu *basestation* připojenou k počítači. Pokud uzel číslo 4 potřebuje komunikovat s basestation, je to realizováno přes uzly číslo 7 a 3. Multihop tedy umožňuje nepřímou komunikaci prvků v sítí za pomoci mezilehlých prvků. Z praktického hlediska to znamená, že všechny uzly v síti nemusí být v přímém dosahu např. basestation a přesto s ní mohou komunikovat.



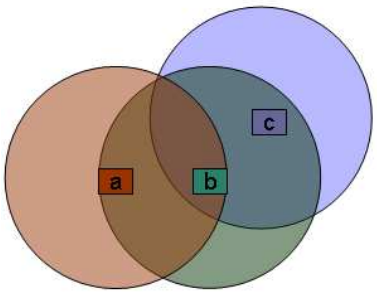
Obrázek 2.3: Multihop architektura

2.3.1 Problém skrytého terminálu

Tento problém souvisí s detekcí kolizí [18] v síti a způsoby, jak se s nimi vypořádat. Kolize vznikají při situacích, kdy ve stejnou dobu chce více uzlů vysílat data. Více vysílání ve stejnou dobu interferuje a dochází tím ke zkreslení dat, přijímače nejsou schopny tyto přijaté signály rozlišit. Důležité je upozornit na skutečnost, že v sítích nelze zabránit kolizím, ale existují možnosti, jak se s nimi vypořádat.

CSMA/CA (*Carrier Sense Multiple Access With Collision Avoidance*) je mechanismus předcházení kolizí, který funguje tak, že pokud chce nějaký uzel komunikovat, musí to oznámit ostatním uzlům. Jakmile tuto informaci uzly obdrží, může komunikace začít. Protože v bezdrátových senzorových sítích nemohou zařízení zároveň vysílat i přijímat, používá se tento mechanismus i zde.

Mějme uzly a,b,c mající mezi sebou vazby *(a,b)*, *(b,c)* viz obr. 2.4, neexistuje zde ovšem vazba *(c,a)*. Pokud chce tedy uzel *c* komunikovat s *a*, musí tak učinit přes uzel *b*.



Obrázek 2.4: Problém skrytého terminálu

Příkladem tohoto problému je hvězdicová topologie. Uvnitř umístěný access point může komunikovat se všemi uzly v síti, ale některé uzly se navzájem nevidí, což může mít za následek vznik kolizí při komunikaci.

2.4 Užití v praxi

Použití těchto technologií je velice různorodé v závislosti na použitých senzorech pro monitorování okolních veličin. Lze je rozdělit do několika skupin:

- monitorování prostoru,
- monitorování objektů,
- monitorování a vzájemné působení objektů v prostoru.

První kategorie zahrnuje sledování životního prostředí, obyvatelstva nebo klimatických změn na naší planetě. Do další kategorie lze zahrnout monitorování stavebních objektů, přizpůsobení fyziologie organizmů životním podmínkám nebo lékařským diagnostikám. Poslední kategorie zahrnuje sledování života ve volné přírodě, monitorování přírodních katastrof nebo havarijních situací.

Tyto nové technologie si postupně nacházejí cestu do různých oblastí lidského života a s postupem času se s nimi budeme setkávat čím dál tím více.

Kapitola 3

TinyOS, NesC

3.1 Úvod

Na počátku vývoje tohoto systému stála spolupráce mezi univerzitami University of California a Berkeley spolupracující s firmou Intel. TinyOS je volně dostupný open source operační systém, napsaný v programovacím jazyku NesC a vytvořený pro platformu bezdrátových senzorových sítí.

Současná verze TinyOs 2.0 [11] vznikla přeimplementaci některých částí verze původní a přidáním nových konceptů. Nový návrh byl motivován nedostatky, které vyplynuly z původní verze, kdy při jejím návrhu a implementaci nebyly zřejmé. Obsahovala několik podstatných omezení, která spočívala převážně v nesnadném propojování komponent, těžce hledatelných interakcích a velice obtížném programování pro nové programátory. Mnoho základních služeb je nyní virtualizováno, což přispělo k snadnějšímu používání komponent a omezilo chyby, které vznikly jejich nesprávným propojením. Současná verze tedy obsahuje několik vylepšení a změn vedoucích k lepšímu porozumění tohoto systému a snadnějšímu programování.

Je nutné nicméně podotknout, že změnou důležitých částí systému došlo k nekompatibilitě mezi verzemi. Existuje ovšem možnost upgrade [2] zdrojového kódu aplikace pro verzi novou. V následujícím textu bude věnována pozornost několika základním konceptům systému.

3.2 Program a struktura

NesC (*network embedded systems C*) je programovací jazyk vycházející z programovacího jazyka C. Obsahuje některá rozšíření (např. komponenty a jejich propojení), navržená pro potřeby operačního systému TinyOS a bezdrátových senzorových sítí.

Program se skládá z několika propojených *komponent* [1], které navzájem poskytují a využívají funkce ostatních komponent obsažených v TinyOS. Komponenty si lze představit jako objekty zapouzdřující svoji vnitřní strukturu a nabízející funkce skrz svá rozhraní. Největší rozdíl od C++ nebo Java objektů je v tom, že nesC komponenty používají pro názvy identifikátorů lokální jmenný prostor (*local namespace*). Pokud tedy implementace komponenty A obsahuje funkci B, je to reprezentováno jménem A.B v globálním jmenném prostoru. Komponenta C může také obsahovat funkci B, ovšem může jít o naprosto odlišnou implementaci funkce B.

Každá komponenta je specifikována funkcemi, které poskytuje (*provide*) ostatním kom-

ponentám a funkcemi, které využívá(*uses*) od jiných komponent. Dále je komponenta specifikována blokem obsahující její vlastní implementaci. Například specifikace komponenty SmoothingFilterC [12], sloužící pro práci se sbíranými daty vypadá následovně :

```
module SmoothingFilterC {
  provides command uint8_t topRead(uint8_t* array, uint8_t len);
  uses command uint8_t bottomRead(uint8_t* array, uint8_t len);
}

implementation {
  ...
}
```

SmoothingFilterC tedy poskytuje funkci topRead a proto ji musí také definovat. Je to proto, aby mohla být v případě potřeby zavolána jinou komponentou. Modul využívá ke své činnosti funkce bottomRead, kterou poskytuje jiná komponenta. Musí proto vytvořit odkaz na tuto funkci a tím vytvoří závislost na jiné komponentě definující ji.

Tento způsob, kdy jsou deklarovány individuální funkce v komponentě, není v praxi moc používán. Místo toho se používají nesC rozhraní, která obsahují souhrn souvisejících funkcí. Například řízení spotřeby obsahuje funkce, které aplikace potřebuje pro zapínání a vypínání nejrůznějších komponent jako např. senzorů nebo radiokomunikace. Rozhraní vyjadřující tuto funkcionalitu vypadá následovně:

```
interface StdControl {
  command error_t start();
  command error_t stop();
}
```

Rozhraní tedy vyjadřuje abstrakci reprezentující funkce, které mohou být využívána pro zapínání a vypínání ostatních komponent v aplikaci.

V TinyOS je provádění mnoha běžných operací, jako např. sběr dat senzorů či odesílání dat, rozděleno na oddělené části (*split phase*). Charakteristickou vlastností těchto operací je jejich obousměrnost (*bidirectional*) – existuje zde vstupní bod (*downcall*), který začíná operaci a výstupní bod (*upcall*), který označuje dokončení operace. V programovacím jazyku nesC jsou typicky vstupními body funkce (*command*) a výstupními body události (*events*). Tuto obousměrnost obsahuje rozhraní Send sloužící pro odeslání paketů v TinyOS:

```
interface Send {
  command error_t send(message_t* msg, uint8_t len);
  event void sendDone(message_t* msg, error_t error);
  command error_t cancel(message_t* msg);
...
}
```

Zda komponenta poskytuje nebo využívá Send rozhraní je určeno, kterou část rozdělené operace reprezentuje. Poskytovatel Send definuje funkci send, cancel a událost sendDone. Uživatel Send rozhraní potřebuje definovat sendDone událost a může volat send a cancel funkce. Když zavolání funkce send vrátí SUCCESS, msg parametr funkce povolí poskytovateli, aby se pokusil odeslat paket. Pokud vše proběhne v pořádku, poskytovatel signalizuje sendDone, posláním ukazatele nazpět uživateli.

3.2.1 Konfigurace a propojení

Předcházející část se zabývala moduly, které jsou základními stavebními kameny TinyOS aplikace. Obsahují stavy a implementaci programové logiky. Moduly pojmenovávají funkce a proměnné bez jejich lokálních jmenných prostorů, proto musí existovat mechanismus mapující množinu jmen jedné komponenty do množiny jmen v jiných komponentách. Tento mechanismus, kdy propojujeme nejméně dvě komponenty, nazýváme propojování neboli *wiring*. Aplikace se tedy kromě modulů, které využívají a poskytují rozhraní, skládá z konfiguračních souborů, které implementují vzájemné propojení komponent do větších aplikací. Pro zajímavost lze uvést, že TinyOS obsahuje více těchto konfiguračních souborů než modulů. Konfigurační soubory mají velmi podobnou strukturu jako moduly. Obsahují svoji specifikaci a implementaci.

Propojení komponent ukážeme na příkladu z naší implementace. Aplikace demonstruje možnosti Dissemination protokolu, který slouží pro aktualizaci dat v senzorové bezdrátové síti. Každých šest vteřin se aktualizuje hodnota čítače, která je poté rozšířena do sítě, a na základě této hodnoty se rozsvítí příslušné diody. První část aplikace tvoří modul SimDisseminationC.nc. Obsahuje rozhraní komponent, která využívá ke své činnosti a dále pak vlastní implementaci aplikace. Druhou částí aplikace je konfigurační soubor SimDisseminationAppC.nc obsahující propojení komponent.

```
module SimDisseminationC {
  uses interface Boot;
  uses interface SplitControl as RadioControl;
  uses interface StdControl as DisseminationControl;
  uses interface DisseminationValue<uint16_t> as Value;
  uses interface DisseminationUpdate<uint16_t> as Update;
  uses interface Leds;
  uses interface Timer<TMilli>;
}

implementation {
  ...
  call Leds.led0On();
  ...
}
```

Komponenta LedsC poskytuje Leds rozhraní:

```
configuration LedsC {
  provides interface Init @atleastonce();
  provides interface Leds;
}
```

Když dojde v programu k zavolání Leds.led0On(), tedy rozsvícení první diody, pojmenuje si funkci v jejím vlastním lokálním jmenném prostoru (*SimDisseminationC*). SimDisseminationC zavolá funkci SimDisseminationC.Leds.Led0on(), LedC tedy poskytuje funkci SimDisseminationC.Leds.Led0on(). Část konfiguračního soubor SimDisseminationAppC.nc z naší aplikace obsahuje toto propojení:

```
configuration SimDisseminationAppC {}
```

```
implementation {
  components SimDisseminationC;
  ...
  components LedsC;
  SimDisseminationC.Leds -> LedsC;
  ...
}
```

Syntakticky jsou konfigurace velmi jednoduché. Mohou obsahovat operátory →, ← a =. Šipka směřující vpravo určuje vztah uživatel → poskytovatel a šipka směřující vlevo vztah inverzní.

TinyOS tedy rozlišuje dva typy komponent, jimiž jsou moduly poskytující implementaci rozhraní a konfigurace poskytující rozhraní pro vzájemné propojení komponent. Programování aplikací pro TinyOS nespočívá ve vytváření nových softwarových komponent, nýbrž v kombinaci již existujících a jejich vzájemným propojením do fungující aplikace.

3.3 Platformy

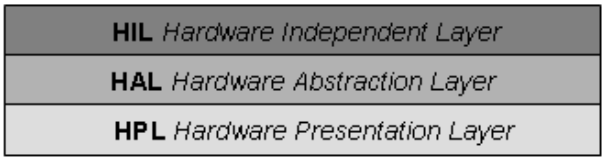
Pojem *platforma* zde představuje sbírku *čipů* a programový kód je propojující. Seznam je umístěn v souboru /tos/platform a při kompilaci programu nesC překladačem prohledáván. Například seznam pro micaZ platformu sděluje kompilátoru (*ncc*), aby použil radiočip CC2420 nacházející se v /tos/chips/CC2420 a mikrokontrolér ATmega128 z adresáře tos/chips/atm128.

Hardwarová abstrakce je v TinyOS 2.0 tvořena tří-úrovňovou hierarchií, nazvanou Hardware Abstraction Architecture (*HAA*), která je znázorněna na obr. 3.1.

Na vrcholu této architektury je hardwarově nezávislá vrstva (*HIL*), která poskytuje hardwarově nezávislou abstrakci. Komponenty na této vrstvě nemají speciální prefix v názvu na rozdíl od nižších vrstev a představují abstrakci aplikací, které mohou být bezpečně kompilovány na různých platformách. Pro ilustraci, HIL komponenta mica2 platformy CC1000 je označena ActiveMessageC, která reprezentuje komunikační vrstvu pro posílání zpráv.

Vrstva hardwarové abstrakce (*HAL*) poskytuje úplné použití hardware, avšak použití je komplikovanější než v případě HIL. Název komponent obvykle obsahuje prefix začínající CC1000.

Hardwarově prezenční vrstva (*HPL*) znázorňuje tenkou softwarovou vrstvu na vrcholu hardware reprezentující vstupy a výstupy nebo registry jako nesC rozhraní. Název je tvořen prefixem Hpl následovaným názvem čipu, tedy např. HplCC1000.



Obrázek 3.1: HAA architektrura

Mezi současné podporované platformy patří mica2, micaZ, telosb, tinynode. Seznam všech podporovaných lze nalézt v adresáři /tos/platforms.

3.4 Plánovač

Operační systém TinyOS neobsahuje žádné jádro ani procesy. Vše se děje na základě událostí (*event*), které jsou generovány v případě nějaké činnosti, jako např. tiknutí časovače nebo poslání zprávy okolním uzlům. Plánovač je nepreemptivní FIFO (*first in first out*) fronta pro ukládání těchto událostí. Původní přístup byl založen na myšlence sdílené fronty pro všechny úlohy a možností posílat úlohy vícekrát. To se ukázalo být problematické v případě úloh rozdělených na více částí *split phases*, protože pokud byla fronta plná, operace odeslání selhala a došlo tak k zablokování následující komponenty, která čekala na zprávu o dokončení. Nový přístup navržený v TinyOS 2.x přiděluje každé úloze vlastní slot obsažený ve frontě a úloha může být poslána pouze jednou. Pokud potřebuje komponenta poslat úlohu několikrát, provede se to nastavením vnitřního stavu proměnné tak, že při vykonání je přeposlána sama sobě. Za nevýhodu tohoto přístupu lze považovat nutnost rezervovat pro každou úlohu 1 byte ve frontě. V případě velkých aplikací, jako např. TinyDB, se paměťové nároky pohybují okolo 1 KB.

Systém umožňuje uplatňovat nové plánovací politiky, např. prioritní, založené na časových razítkách aj..

3.4.1 Časovač

Mikrokontroléry nabízejí bohatý systém časovačů, zahrnující různorodý rozsah časování, přesnost a rozmanité možnosti. Dále pak jeden nebo více porovnávacích registrů určených pro spuštění, přerušení nebo změny hodnoty časovače. TinyOS se nesnaží postihnout tuto různorodost ve své platformě nezávislé architektuře, naopak využívá principů HAA architektury pro vytvoření funkcionality skrz komponenty a rozhraní na HPL vrstvě. Pro všechny časovače platí dva společné rysy, jimiž jsou měření času a periodické generování událostí v čase. Časovač tedy představuje komponentu, která měří čas a v určitém čase vygeneruje událost, nebo periodicky generuje události se zvolenou periodou. Podporovanými časovači z hlediska přesnosti jsou millisekundové, mikrosekundové a 32 KHz cyklus hodin. Z hlediska rozsahu čítaných hodnot lze použít 8, 16, 32 nebo 64-bitové čítače. Dále systém poskytuje velice přesné asynchronní časovače (jsou označeny slovem async).

3.5 Komunikace a síťový protokol

TinyOs poskytuje abstrakci na úrovni paketů ve formě Active Message *(AM)* [4], která představuje asynchronní komunikační mechanismus, sloužící k plné hardwarové flexibilitě a výkonu uvnitř sítě. Pro účel komunikace obsahuje TinyOS strukturu nazvanou *message_t* [9] obsahující hlavičku, data, metadata a patičku (viz obr. 3.2). Definici lze nalézt v tinyos-2.x/tos/types/message.h. Každá platforma definuje vlastní specifikaci hlavičky, patičky a metadat v message_t.

Obrázek 3.2: Struktura message_t

Hlavička je tvořena polem bajtů, jejichž velikost je dána spojením délek hlaviček platformy. Protože zásobník často upřednostňuje pakety uložené souvisle, rozložení paketů

v paměti nemusí odrážet rozložení nesC struktury. Platforma MicaZ (CC2420) obsahuje 11 bytovou hlavičku.

Datové pole `message_t` dokáže pojmout data o velikosti dané konstantou TOSH_DATA_LENGTH, jejíž implicitní hodnota je 28 bytů. Tato hodnota se dá změnit, avšak může tím dojít k situaci, kdy budou mít dvě odlišné verze aplikace rozdílnou velikost MTU (*maximum transmission unit*). Obdrží-li datová vrstva paket, jehož datová část je delší než TOSH_DATA_LENGTH tak je zahozen.

Patička zajišťuje, že `message_t` má dostatečně velký paměťový prostor pro uchování zápatí všech náležících linkových vrstev, pokud jsou dány velikosti MTU. Uložení zápatí je závislé na jednotlivých platformách a nemusí korespondovat s nesC strukturou.

Metadata uchovávají informace, které nejsou posílány nebo je používá zásobník. Tento mechanismus umožňuje datové vrstvě uložit např. informace o RSSI (*Received Signal Strength Indication*) či časových razítkách.

Struktura je dostatečně velká pro uchování paketů z každého komunikačního rozhraní uzlu. Z důvodu zapouzdření je přístup k ní realizován pomocí rozhraní. Například získání cílového AM paketu nazvaného msg je provedeno zavoláním AMPacket.destination(msg) komponentou. Použitím rozhraní pro odeslání zároveň určujeme adresní mód pro komunikaci, tedy např. AM komunikace obsahuje AMSend rozhraní požadující AM cílovou adresu.

Active messages komunikace je virtualizována skrz čtyři základní komponenty, kterými jsou AMSenderC, AMReceiverC, AMSnooperC a AMSnoopingReceiverC.

V operačním systému jsou implementovány dva základní síťové protokoly [16], respektive komponenty, které je poskytují. První (*Dissemination*) z nich spolehlivě doručuje malé (*20 byte*) objemy dat ke každému uzlu v síti, druhý (*Collection*) slouží pro sběr dat od uzlů v síti. Společně tyto protokoly dokáží poskytnout široký rozsah služeb funkcí aplikacím sbírající data. [5]

3.6 Limity TinyOS

Zkušenosti poukázaly na některá omezení [3] tohoto operačního systému. Přidání podpory nové platformy je realizováno zkopírováním velkého množství kódu z již existující platformy a následnou modifikací vznikne platforma nová. Protože abstrakce TinyOS nedefinuje přesné hranice, přistupují komponenty často k hardwarovým zdrojům přímo. Pro ilustraci micaZ implementace CC2420 používá hardwarový časovač sloužící při výskytu kolizí pro generování náhodně zvolené doby pro další pokus. Nová platforma, která dědí z micaZ se musí ujistit, zda-li neužívá tohoto časovače jinde, ovšem žádná struktura nedefinuje je-li časovač používán.

TinyOs obsahuje velké množství komponent a jejich propojováním může dojít k nepředvídatelným vzájemným interakcím a závislostem. Příkladem tohoto problému je sdílení sériově periferního rozhraní (*SPI*) sběrnice. Může být využívána radiovysílačem či flash pamětí. Sběrnice je také používána pro připojení externích senzorů. V aplikaci využívající SPI rozhraní je proto nutné velice opatrně s těmito komponentami pracovat, aby nedošlo např. k současné inicializaci radiokomunikace a flash systému, což by mělo za následek neúspěch některého z procesů.

Radiokomunikace obsahuje několik problémů, pro něž v současné době nejsou vhodná řešení. Práce s pakety v TinyOs funguje tak, že aplikace obdrží přerušení a následně skrz SPI přečte přijatý paket. Dále je poslána úloha do fronty, čímž je signalizován nadřazeným komponentám příjem dat. Jelikož je ale fronta sdílena, je zde možnost, že operace odeslání úlohy

selže a tím vznikne problém. Na hardwarové vrstvě CC2420 došlo k úspěšnému obdržení paketu a bylo tím odesláno potvrzení o doručení, ovšem radiosoftware nedokáže doručit paket aplikaci. Opakování odeslání vyžaduje, aby někdo zavolal komponentu znovu v budoucnu, neexistuje zde ovšem správný způsob, jak to zajistit. Daleko větší problém ovšem nastává při odesílání paketu, kdy je přenos realizován v několika fázích. Komponenty na vyšší vrstvě očekávají poslání události sendDone() před znovupoužitím bufferu k poslání dalšího paketu. Radiokomponenta nemůže poslat úlohu do fronty signalizující událost odeslání, a tím dochází k zablokování odesílání dat na dobu neurčitou, což má za následek asynchronní vykonávání, které by mohlo vést k chybám v paměti. Tento problém se netýká jen komunikace, nýbrž všech komponent oznamujících dokončení některé části operace přes systémovou frontu.

Kapitola 4

Síťové protokoly

4.1 Dissemination protokol

Dissemination [13] je jeden ze dvou základních síťových protokolů. Slouží pro zajištění konzistence dat napříč celou bezdrátovou sítí. Je zde využito sdílené proměnné, jejíž kopii si uchovává každý uzel ve své RAM paměti. Konzistence v tomto kontextu neznamená, že každý uzel uvidí všechny hodnoty, které proměnná nabývá, nýbrž že se síť dokáže shodnout na nejnovější verzi proměnné. Datová konzistence je robustní k dočasnému odpojení uzlů a vysoké datové ztrátě. Dissemination je tedy protokol informující uzly, změní-li se hodnoty sdílené proměnné. Tohoto mechanismu je běžně využíváno pro rekonfiguraci sítě a přeprogramování uzlů. Protokol se skládá ze dvou částí: řízení dopravy a dopravy dat. Doprava dat značně závisí na velikosti položek dat, což platí i pro řízení dopravy.

Dissemination model: protokol usiluje o to, aby se každý uzel shodoval s poslední verzí proměnné. Tímto způsobem může uzel vyžívat síť k tomu, aby dosáhl konzistence hodnoty proměnné informováním sítě.

TinyOs 2.0 obsahuje komponentu DisseminatorC, která poskytuje dvě rozhraní pro tento protokol. Rozhraní DisseminationUpdate slouží pro odeslání nové aktualizované hodnoty do sítě.

```
interface DisseminationUpdate<t> {
  command void change(t* newVal);
}
```

Pokaždé, když chce odesílatel *(producer)* hodnotu v síti aktualizovat, musí zavolat funkci DisseminationUpdate.change(), jejímž jediným parametrem je nová hodnota proměnné.

Komponenta dále nabízí rozhraní DisseminationValue, které je využíváno příjemcem *(user)* pro aktualizaci hodnoty.

```
interface DisseminationValue<t> {
  command const t* get();
  command void set(const t*);
  event void changed();
}
```

Událost DisseminationValue.changed() signalizuje, že došlo ke změně sdílené proměnné, tedy odesílatel použil funkci `changed()`. Funkce get() slouží pro získání nové aktualizované proměnné. Ukázka funkce Dissemination protokolu včetně podrobnějšího popisu aplikace je

předvedena v aplikaci SimDissemination na přiloženém CD. Obrázek A.1 zobrazuje ukázku malé senzorové síťě, kterou jsme vytvořili pro simulaci tohoto protokolu.

4.2 Collection protokol

Sběr dat je jedním ze základních požadavků aplikací pro senzorové sítě. Data jsou sbírána uzly v síti, které tvoří jeden a nebo více *sběrných stromů (collection trees)*. Sběrný strom může obsahovat až stovky uzlů a je propojen s jednou basestation, která sbírá data od uzlů v daném stromu. Nasbíraná data mohou být poté basestation poslána do počítače pro další zpracování nebo vyhodnocení. V situaci, kdy uzel změří pomocí senzorů potřebná data, pošle je skrz ostatní uzly ve stromu do basestation – collection [14] tedy poskytuje multihop doručování paketů.

Uzel může vykonávat několik odlišných rolí: producer, consumer a snooper. V závislosti na této roli musíme použít příslušné rozhraní komponenty Collection protokolu. **Consumer** je basestation, která tvoří kořen (*root*) stromu a obsahuje množina všech kořenů a cest vedoucím k nim v infrastruktuře sítě. Infrastruktura může být sdílena mezi několik nezávislých aplikací, které jsou rozlišeny pomocí *collection identifikátorů* (collection_id_t). Je důležité podotknout, že datový provoz může být multiplexován, zatímco řízení provozu nikoliv. Consumer pro příjem dat z collection stromu používá standardní rozhraní Receive [10], které má jako parametr collection identifikátor. Consumer dále využívá rozhraní RootControl, které nabízí funkci RootControl.setRoot() pro nastavení kořenu v infrastruktuře senzorové sítě. Protokol automaticky nenastavuje kořen stromu, proto je pro správné fungování protokolu nutné ho nastavit (*např. TOS_NODE_ID == 0*). Rozhraní obsahuje dále další dvě funkce, jejichž význam je zřejmý z jejich názvů.

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

Producer je uzel, který sbíraná data z senzorů posílá skrz sběrný strom do basestation využitím rozhraní Send [10], které má jako parametr collection identifikátor.

Snoopers jsou uzly, které odposlechnou zprávu během přenosu skrz síť. Využívají také Receive rozhraní s parametrem specifikovaným collection identifikátorem.

Collection protokol poskytuje TinyOS komponentu CollectionC, která obsahuje rozhraní pro jeho funkčnost.

```
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];
  }
}
```

Rozhraní CollectionC dále obsahují další funkce pro uživatele a události pro poskytovatele. V rozhraní si lze povšimnout, že Receive, Snoop a Intercept rozhraní jsou parametrizována identifikátorem *collection_id_t*. Každý identifikátor představuje rozdílnou protokolovou operaci na vrcholu protokolu – odlišné hodnoty reprezentují odlišné protokolové operace na vrcholu active message. Znamená to tedy, že všechny pakety obsahující stejný identifikátor mají stejný datový formát paketu, a proto mohou být data těmito rozhraními (Receive, Snoop a Intercept) správně zpracovávána. Událost rozhraní Receive.receive(), která signalizuje, že byla data v pořádku přijata, může být zavolána pouze na kořenových uzlech, tedy basestation. Tuto událost lze kombinovat s funkcemi rozhraní Send, avšak CollectionC musí vytvořit kopii zásobníku, protože se při odesílání pracuje se stejným zásobníkem.

Událost Intercept.forward() je použita, pokud nekořenový uzel obdrží přeposílaná data od jiného uzlu.

Aplikace SimCollection, obsažena v elektronické příloze, demonstruje praktickou ukázku tohoto protokolu.

Kapitola 5

TOSSIM

5.1 Úvod

Z historického hlediska bylo vytvoření přesné a měřitelné simulace klíčem k systémovému vývoji. Zkoumáním návrhu TinyOS a bezdrátových senzorových sítí byl vytvořen simulační nástroj TOSSIM [8]. Jedná se o jednoduchou, avšak velice mocnou simulační knihovnu, která dokáže s využitím pravděpodobnostního modelu zachytit s vysokou přesností síťové chování až tisíců uzlů. Architektura TOSSIMu obsahuje:

- podporu pro kompilování komponent grafů TinyOS do simulační infrastruktury,
- diskrétně událostní frontu,
- přeimplementovanou abstrakci některých komponent TinyOS,
- mechanismus pro radio a ADC modely,
- komunikační služby pro externí programy.

Diskrétně událostní simulace je generována přímo z TinyOs grafů komponent. Znamená to tedy, že je simulován kód, který je spuštěn v uzlech senzorové sítě. Nahrazením několika low-level komponent, TOSSIM transformuje hardwarová přerušení v diskrétně simulační události, což má za následek doručování přerušení vykonávaných TinyOS do událostní fronty simulátoru.

5.2 Vytvoření simulačního modelu

TOSSIM podporuje C++ a Python rozhraní pro vytvoření a práci s modelem. Námi vytvořená simulace bude využívat rozhraní Pythonu. Ve skutečnosti to funguje tak, že zavoláním Python objektu se zavolá C++ objekt, který skrz C rozhraní komunikuje s TOSSIMem. Simulaci je možné realizovat dvěma způsoby. Lze spustit Python a interaktivně zadávat příkazy nebo vytvořit skript, který pak dávkově spustit.

Jako ukázkovou aplikaci pro demonstraci možností TOSSIMu zvolíme námi vytvořenou SimCollection. Uzel s id = 0 je zvolen jako collection root a ostatní uzly pravidelně ve čtyřech sekundových intervalech zvyšují hodnotu sdílené proměnné, na jejímž základě se rozsvítí příslušné diody na uzlu s id = 0, který tyto data sbírá.

V každé simulaci je nejprve nutné importovat TOSSIM:

```
from TOSSIM import *
```

Dále pak vytvořit TOSSIM objekt:

```
t = Tossim([])
```

Volitelný argument v hranatých závorkách dovoluje přistupovat k proměnným v simulaci. V tomto případě říkáme TOSSIMu, že nechceme sledovat žádné proměnné.

Velice užitečnou funkcí je *dir(t)*, která slouží k zobrazení funkcí objektu a jediným parametrem je sám objekt. Mezi zajímavé funkce objektu t stojí zmínit:

```
currentNode(): vrací ID aktuálního uzlu,
getNode(id): vrací objekt reprezentovaný id uzlu,
runNextEvent(): vykoná jeden krok simulace,
timeStr(): vrací textovou reprezentaci aktuálního času,
mac(): vrací objekt reprezentující media access layer,
radio(): vrací objekt reprezentující radio komunikační model,
addChannel(ch, output): přidá vystup kanálu do simulace,
ticksPerSecond(): vrací kolik proběhlo simulačních tiků.
```

TOSSIM umožňuje použít pro ladění programů *gdb* systém. Gdb vytvoří kanál, skrz který aplikace posílá simulátoru informace. Tyto kanály se musí do simulace přidat, aby TOSSIM věděl o jejich existenci (např. t.addChannel("Boot", sys.stdout);). Pro větší ilustraci zde uvedeme úsek NesC kódu z námi vytvořeného modelu.

```
event void Boot.booted() {
  local.id = TOS_NODE_ID;
  if (local.id == 0)
    dbg("Boot","Basestation: booted at %s \n", sim_time_string());
  else
    dbg("Boot","Node: booted at %s \n",sim_time_string());
```

Programový kód obsahuje událost, která nastane po nabootování uzlu. Pokud je uzel basestation, tedy id = 0, je vypsána do terminálu přes Boot kanál zpráva, že došlo k nabootování basestation a simulační čas v němž k událostí došlo. V námi vytvořených aplikacích používáme dva typy gdb volání, jimiž jsou dbg a dbgerror(pro chybové stavy).

Dalším důležitým krokem vytvoření modelu bylo nastavení síťové komunikace mezi uzly v síti. Pro tuto potřebu byl vytvořen radioobjekt r, který reprezentuje fyzickou vrstvu bezdrátové sítě.

```
r = t.radio();
```

obsahující funkce pro radiokomunikaci. Nejdůležitější používané funkce jsou:

```
add(src, dest, gain): přidá spojení od odesílatele k příjemci se ziskem,
connected(src, dest): ověřeje existenci spojení od odesílatele k příjemci.
```

Topologii sítě je možné specifikovat přímo v simulačním skriptu např.

```
r.add(0, 1, -70.0);
r.add(1, 0, -70.0);
r.add(2, 1, -49.0);
r.add(1, 2, -49.0);
```

Pokud tedy zařízení s id = 0 komunikuje s zařízením s id = 1 je zisk -70dBm. Pro větší sítě je výhodnější načíst do TOSSIMu topologii z externího souboru, jehož struktura může vypadat následovně:

```
0   1   -64.71
1   0   -66.06
....
```

A vytvoření topologie sítě lze provést takto:

```
f = open("topology.txt", "r")
lines = f.readlines()
for line in lines:
  s = line.split()
  if (len(s) > 0):
    r.add(int(s[0]), int(s[1]), float(s[2]))
```

Implicitní hodnoty TOSSIM radiomodelu jsou založeny na , transceiveru CC2420. Ty byly získány na základě křivky signál-šum experimentálním sběrem dat dvou micaZ uzlů. TOSSIM dokáže simulovat radiofrekvenční šum a interference, které uzel slyší od ostatních uzlů a okolního prostředí. Tohoto je docíleno Closest Pattern Matching *(CPM)* algoritmem [7], který ze stopy šumu generuje statistický model a dokáže tak poskytnout kvalitní RF simulaci. Je nutné podotknout, že model není úplně ideální, protože nepostihuje některé situace interferencí mezi uzly, je ovšem lepší než tradiční Packet loss modely.

K nakonfigurování CPM je třeba přidat šum vytvořeným objektům uzlů zavoláním `addNoiseTraceReading(noise)`, kde noise je hodnota šumu v dBm. Demonstrační soubory s vzorkovacími hodnotami lze v TinyOS nalézt v adresáři tos/lib/tossim/noise. Sobory mají jednoduchou strukturu, kde každý řádek obsahuje jednu hodnotu šumu a musí obsahovat nejméně 100 řádků, jinak CPM nemá dostatek dat pro generování statistického modelu. Příkladem takových hodnot je meyer-heavy.txt vytvořený na Standfordské univerzitě.

```
noise = open("meyer-heavy.txt", "r")
lines = noise.readlines()
for line in lines:
  str = line.strip()
  if (str != ""):
    val = int(str)
    node0.addNoiseTraceReading(val)
    node1.addNoiseTraceReading(val)
    node2.addNoiseTraceReading(val)
```

Dále je nutné u každého uzlu vytvořit *noise model* funkcí `createNoiseModel()`. Zatímco radioobjekt reprezentuje propojení uzlů na fyzické úrovni, linková vrstva je reprezentována MAC objektem. Ten v sobě obsahuje velké množství funkcí, které nastavují parametry komunikace mezi uzly, např. ovládání backoff [1] chování, šířku přenosového pásma či velikost preambule paketu. V default nastavení je MAC objekt nakonfigurován jako standard radio zásobníku CC2420 definovaný ve standardu TinyOS 2.0.

Objekt lze jednoduše vytvořit zavoláním funkce *mac()* TOSSIM objektem a vytvořit tak objekt nový, reprezentující tuto vrstvu. Protože jsme použili platformu Micaz, která obsahuje výše zmíněný zásobník, nebylo při simulaci nutné tyto parametry speciálně nastavovat.

[1] Časová perioda po kterou zařízení čeká než začne další pokus o vysílání.

5.3 Kompilace TOSSIMu

Zdrojové kódy jsou umístěny v /tos/lib/tossim a obsahují i užitečné soubory pro vytvoření simulace jako například data pro vytvoření modelu či topologie. V současné době je jedinou podporovanou platformou MicaZ platforma. Kompilace nesC aplikace pro simulaci se provede příkazem `make micaz sim`.

Prvním krokem, který je vykonán je za použití nástroje nesc-dump vytvoření XML dokumentu, popisující mimo jiné jména a typy všech proměnných v aplikaci. Dále je provedena změna cest include pro simulaci a kompilace TinyOS aplikace. Dalším krokem se zkompiluje podpora pro C++ a Python programové rozhranní skrz které je realizováno vytvoření vlastního simulačního modelu. Poté je vytvořen sdílený objekt obsahující TOSSIM kód, C++ a Python podporu.

5.4 Výstup simulace Dissemination protokolu

Po naprogramování aplikace využívající ke své funkčnosti možnosti Dissemination rozhraní byl vytvořen simulační model v TOSSIMU. Následující text je výstupem naší simulace:

```
DEBUG (0): BaseStation: aplication booted at 0:0:0.000034532
DEBUG (2): Node: aplication booted at 0:0:0.000034532
DEBUG (1): Node: aplication booted at 0:0:0.008212341
DEBUG (3): Node: aplication booted at 0:0:0.008541532
DEBUG (0): Basestation: dissemenate message at 0:0:5.859409542
DEBUG (0): Led 1 On
DEBUG (3): Node: received message at 0:0:5.861164292
DEBUG (1): Node: received message at 0:0:5.861164292
DEBUG (3): Led 1 On
DEBUG (1): Led 1 On
DEBUG (0): Basestation: dissemenate message at 0:0:11.718784542
DEBUG (0): Led 2 On
DEBUG (3): Node: received message at 0:0:11.7287835724
DEBUG (3): Led 2 On
DEBUG (1): Node: received message at 0:0:11.728778990
DEBUG (1): Led 2 On
DEBUG (2): Node: received message at 0:0:12.387835724
DEBUG (2): Led 2 On
DEBUG (0): Basestation: dissemenate message at 0:0:17.578159542
...
```

Čísla v kulatých závorkách odpovídají id uzlů, 0 je basestation. Na začátku simulace byli všechny prvky sítě vypnuté. V čase 0.0345 ms došlo k spuštění aplikace v basestation a poté se nabootovali tři uzly tvořící síť v pořadí 2, 1 a 3. V čase 5.859 s došlo k zvětšení hodnoty čítače o jedničku a tím rozsvícení první diody na basestation. Tato změna hodnoty byla vyslána do sítě a jako první ji obdržel uzel 3 v čase 5.861 sekund a uložil si ji jako aktuální hodnotu do paměti. Poté uzel rozsvítil stejnou diodu jako je rozsvícena na basestation, protože je hodnota čítače stejná. To samé nastalo pro uzel č. 1. Druhému uzlu se nepodařilo novou hodnotu přijmout. V čase 11.718 sekund došlo k další aktualizaci hodnoty čítače do sítě, což bylo reflektováno rozsvícením příslušných diod na uzlech.

Obdobným způsobem byl vytvořen simulační model aplikace SimCollection, která je součástí elektronické přílohy. Dodatek B obsahuje výstup z této simulace.

Kapitola 6

Použitý hardware, praktická realizace

6.1 MicaZ

V této podkapitole bude stručně popsán hardware, který byl použit pro praktickou realizaci.

MicaZ je platforma pro bezdrátové senzorové sítě, která byla vyvinuta a vyrobena firmou Crossbow Technology [15]. Tato firma stojí za vznikem dalších komponent pro senzorové sítě, např. Imote2, Mica2. MicaZ je bezdrátový *modul*, který pracuje v pásmu 2.4 - 2.48 GHz a je realizován v souladu s normou IEEE 802.15.4 a standardem ZigBee. Modul kromě napájení ve formě dvou tužkových baterii obsahuje *radio platformu*, která zajišťuje radio komunikaci a řízení sběru sbíraných dat. Bezdrátový senzorový uzel, který lze vidět na obrázku(2.2), je složen z senzorové desky, radio platformy a napájení.

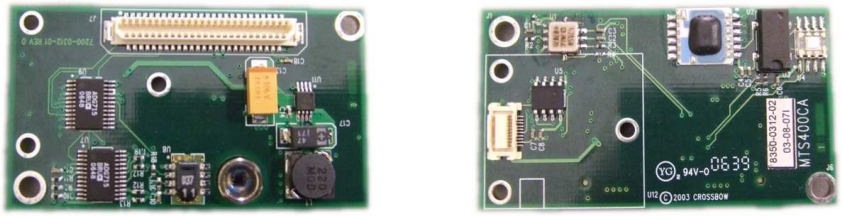
6.1.1 Radio platforma

Jedno procesorová deska MPR2600 je založena na mikrokontroléru ATmega128. Platforma obsahuje rozšiřují 51 pinový slot, který slouží pro připojení *senzorové desky*. Pracuje v radiovém pásmu 2,4 Ghz a přenosová rychlost komunikace může dosahovat až 250 Kbps v závislosti na okolních podmínkách.

Obrázek 6.1: MRP2600

6.1.2 Senzorová deska

MTS400 je senzorová deska, která je vybava různými typy senzorů, sloužící pro měření teploty, barometrického tlaku, okolního osvětlení nebo vlhkosti. Tepelný senzor dokáže snímat teploty v rozmezí -10° C až +60° C. Senzor pro měření vlhosti je schopen snímat vlhkost až do velikosti 60% RH nekondenzující. Barometrický senzor pracuje v rozmezí 300-1100 mbar. Světělný senzor dokáže zachytit spektrální citlivost v rozmezí 400-1000 nm, která je podobná lidskému oku.



Obrázek 6.2: MTS400CA

6.1.3 Basestation

MIB520 je programovací deska a zároveň basestation pro bezdrátové senzorové sítě. Programovaní a komunikace skrz PC je realizována, narozdíl od starší verze (MIB510), USB portem, který také slouží pro napájení. Obsahuje 51-pinový konenktor pro připojení hardwarových platforem IRIS, MICA2 a MICAZ.

Prakticky bylo vyzkoušeno, že maximální vzdálenost, na kterou je s uzly basestation schopna komunikovat, je ve vnitřních prostorách okolo 30 m.

Obrázek 6.3: Basestation MIB520

6.2 Praktická realizace

Po úspěšné simulaci aplikací v TOSSIMu došlo na praktickou část, ve které jsme se snažili dosáhnout nainstalování aplikací do senzorových uzlů a basestation. Vytvořit tak malou bezdrátovou senzorovou síť autonomních zařízení, která budou mezi sebou komunikovat. V následujícím textu bude vysvětlen postup instalace aplikace SimDissemination do uzlů a vytvoření bezdrátové sítě. Jako hardware jsme použili programovací desku MIB520, dále pak desku MTS400CA vybavenou senzory a radiovysílačku MPR2600.

Pro realizaci jsme použili nejnovější verzi TinyOs 2.0.2, kterou jsme spustili na notebooku z live CD linuxové distribuce *XubunTOS*. Tato distribuce v sobě zahrnuje operační systém TinyOs a dovoluje jej relativně jednoduše používat pouhým nabootováním z CD. TinyOs je umístěn v adresáři /opt/tinyos-2.x.

Jak již bylo zmíněno MIB520 obsahuje USB port, který je využit jak pro programování tak i pro komunikaci basestation s počítačem. V současné době nepodporuje TinyOs tuto platformu. Při pokusu nahrání aplikace do uzlu (make micaz install) došlo k chybě s flash pamětí:

```
Verifying: flash
flash error at address 0x0: file=0x0c, mem=0xff
flash error at address 0x1: file=0x94, mem=0xff
flash error at address 0x2: file=0x46, mem=0xff
...
```

Bylo proto nutné instalovat aplikaci přes softwarové rozhraní platformy MIB510, která je TinyOs podporována.

XubunTOS po připojení programovací desky MIB520 dokázal tuto desku hotplugem detekovat [1] a vytvořil virtuální porty pro programování a komunikaci. ttyUSB0 je určen k programování uzlů nebo basestation, zatímco ttyUSB1 pro komunikaci basestation s PC.

Další důležitým krokem bylo nastavení kanálu na kterém bude probíhat radiokomunikace a nastavení virtuálního portu pro platformu MIB510 na USB0. Číslo kanálu specifikuje na jakém frekvenčním rozsahu bude komunikace probíhat. Jako vhodný komunikační kanál se ukázal vzhledem k okolním podmínkám kanál 25, kdy komunikace probíhala v pořádku bez datových ztrát. U ostatních zkoušených kanálu komunikace neprobíhala vůbec. Bylo to pravděpodobně zapříčiněno okolní bezdrátovou wifi sítí, která rušila radiokomunikaci.

Nastavení těchto dvou důležitých parametrů se provede přidáním:

```
MIB510 ?= /dev/ttyUSB0
PFLAGS = -DCC2420_DEF_CHANNEL=25
```

do *makedefaults*, který je uložen v adresáři /opt/tinyos-2.x/support/make. Pravidla uplatňovaná při překladu aplikace lze vypsat zadáním `printenv MAKERULES` do konzole.

Tímto krokem došlo k nastavení parametrů makefilu a bylo tak možné aplikaci zkompilovat pro námi použitou programovací desku MIB520. Kompilace se provedla příkazem *make micaz* a tím došlo v adresáři build k vytvoření souboru main.exe obsahující binární podobu aplikace. Následná instalace do uzlu se provedla příkazem make micaz reinstall.id mib510,/dev/ttyUSB0, kde id představuje číslo uzlu do kterého se program nainstaluje. Tyto id mohou být libovolně zvolena, ovšem typicky je 0 vyhrazena pro basestation. Nahráli

[1] V případě, že tinyOs nedokáže nalézt hardware, je potřeba nainstalovat ovladače, které lze nalézt na http://www.ftdichip.com/Drivers/VCP.htm.

jsme aplikaci do několika uzlů a basestation a vytvořili tak malou bezdrátovou síť se třemi uzly s id = 1,2,3 a basestation s id = 0. V průběhu nahrávání programu do uzlu bylo na programovací desce vidět blikání červené diody, což signalizovalo úspěšné nahrávání do paměti uzlu. Po naprogramování všech uzlů a basestation bylo zapnuto napájení na jednotlivých prvcích bezdrátové sítě, čímž došlo k spuštění boot sequence programů v uzlech. Po zapnutí basestation v ní docházelo v 4 sekundových intervalech k periodickému zvyšování čítače a rozsvícení příslušných diod. Tato hodnota byla basestation šířena do sítě a uzly tuto novou hodnotu reflektovaly stejně rozsvícenými diodami jako na basestation. Byly provedeny pokusy s různým rozmístěním uzlů a test multihop komunikace.

Tímto obdobným způsobem jsme vyzkoušeli funkčnost i druhé aplikace SimCollection, demonstrující Collection protokol.

Praktická realizace ukázala, že se aplikace chovaly stejně jako v simulačním modelu vytvořeným v TOSSIMu. Prakticky jsme tedy vyzkoušeli funkčnost aplikace SimDissemination a SimCollection a demonstrovali tím fungování dvou základních protokolů pro sběr a distribuci dat v bezdrátových senzorových sítích.

Kapitola 7

Závěr

Tato práce se zaměřuje na problematiku bezdrátových senzorových sítí a bylo v ní kladeno za cíl předložit čtenáři podrobnější úvod do těchto technologií – seznámit ho s operačním systémem TinyOs 2.0 a poukázat na některé odlišnosti od ještě stále používané starší verze. Dále pak poukázat jakým způsobem se v TinyOS pomocí programovacího jazyka nesC programují aplikace pro senzorové sítě za použití komponent. V neposlední řadě také předvést možnosti simulační knihovny TOSSIM a ukázat praktickou realizaci aplikací v bezdrátové senzorové síti za použití platformy MicaZ.

Byly zde vysvětleny dva základní protokoly pro síťovou komunikaci, které se využívají u složitějších aplikací. V programovacím jazyku NesC byly vytvořeny aplikace demonstrující jejich možnosti.

Dále byly tyto aplikace za použití simulátoru TOSSIM odsimulovány. Poté došlo k praktické realizaci teoreticky odsimulovaných aplikací. Byla vytvořena malá senzorová síť o několika uzlech a jedné basestation. Do těchto zařízení byly přes operační systém TinyOs nainstalovány aplikace. Tímto krokem došlo ke konfrontaci teoretického modelu s podmínkami reálného světa. Výsledkem tedy byla demonstrace reálné funkčnosti dvou základní síťových protokolů poskytujících sběr a šíření dat v bezdrátových sítích. Vlastním přínosem této práce bylo seznámení se s novou, rychle se rozvíjející technologií, která si stále hledá uplatnění v různorodých oblastech lidského života.

Jako další pokračování práce bych chtěl přispět při tvorbě implementace a simulace interpretu agentního systému pro bezdrátovou senzorovou síť.

Literatura

[1] Getting Started with TinyOS. [online], [cit. 2008-05-3].
URL <http://docs.tinyos.net/index.php/Getting_Started_with_TinyOS>

[2] Azim, T.; Levis, P.: Porting TinyOS 1.x Code to TinyOS 2.0. [online], [cit. 2008-05-3].
URL <http://www.tinyos.net/tinyos-2.x/doc/html/porting.html>

[3] Buonadonna, P.; Culler, D.; Gay, D.; aj.: T2: A Second Generation OS For Embedded Sensor Networks. Technická Zpráva TKN-05-007, Telecommunication Networks Group, Technische Universität Berlin, Listopad 2005.
URL <http://www.tkn.tu-berlin.de/publications/papers/T2_TR.pdf>

[4] Culler, D. E.; von Eicken, T.; Goldstein, S. C.; aj.: Active Messages: a Mechanism for IntegratingCommunication and Computation. In *In Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992, s. 256–266.

[5] Haenselmann, T.: Sensor networks, GFDL Wireless Sensor Network textbook. *f*, April 2006.

[6] Jiang, D. C. J. P. X.: Perpetual Environmentally Powered Sensor Networks. *Computer Science Department*.

[7] Lee, H.; Cerpa, A.; Levis, P.: Improving wireless simulation through noise modeling. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-638-X, s. 21–30.
URL <http://doi.acm.org/10.1145/1236360.1236364>

[8] Levis, P.: Simulating TinyOS Networks. [online], [cit. 2008-04-14].
URL <http://www.cs.berkeley.edu/~pal/research/tossim.html>

[9] Levis, P.: TEP:111 - message t. [online], [cit. 2008-04-10].
URL <http://www.tinyos.net/tinyos-2.x/doc/pdf/tep111.pdf>

[10] Levis, P.: TEP:116 - Packet Protocols. [online], [cit. 2008-04-19].
URL <http://www.tinyos.net/tinyos-2.x/doc/pdf/tep116.pdf>

[11] Levis, P.: TinyOS 2.0 Overview. [online], [cit. 2008-04-14].
URL <http://www.tinyos.net/tinyos-2.x/doc/html/overview.html>

[12] Levis, P.: Tinyos programming. [online], [cit. 2008-04-14].
URL <http://csl.stanford.edu/~pal/pubs/tinyos-programming-1-0.pdf>

[13] P.Levis; Tolle, G.: TEP:118 - Dissemination of Small Values. [online], [cit. 2008-04-10].
URL <http://www.tinyos.net/tinyos-2.x/doc/pdf/tep118.pdf>

[14] R. Fonseca, K. J., O. Gnawali; Levis, P.: TEP:119 - Collection. [online], [cit. 2008-04-20].
URL <http://www.tinyos.net/tinyos-2.x/doc/pdf/tep119.pdf>

[15] www stránky: Crossbow technology.
URL <http://www.xbow.com>

[16] Wikipedia: Network Protocols. [online], [cit. 2008-05-10].
URL <http://docs.tinyos.net/index.php/Network_Protocols>

[17] Wikipedia: Sensor node. [online], [cit. 2008-04-14].
URL <http://en.wikipedia.org/wiki/Sensor_node>

[18] Wikipedia: Sensor node. [online], [cit. 2008-05-10].
URL <http://cs.wikipedia.org/wiki/CSMA>

Seznam příloh

Dodatek A

Obrázky

Obrázek A.1: Ukázka vytvořené bezdrátové sítě pro praktické ověření funkčnosti Dissemination protokolu

Dodatek B

Výstup simulace Collection protokolu

```
DEBUG (0): Basestation: booted at 0:0:0.000034532
DEBUG (2): Node: booted at 0:0:0.000316544
DEBUG (4): Node: booted at 0:0:0.000457657
DEBUG (1): Node: booted at 0:0:0.006512341
DEBUG (3): Node: booted at 0:0:0.008541532
DEBUG (2): Node: sending 0x1 to basestation 0:0:3.906566554
DEBUG (4): Node: sending 0x1 to basestation 0:0:3.906707667
DEBUG (0): Basestation: received packet from 2 with 0x1 at 0:0:3.909450458
DEBUG (0): Led 1 toggle
DEBUG (1): Node: sending 0x1 to basestation 0:0:3.912762351
DEBUG (0): Basestation: received packet from 4 with 0x1 at 0:0:3.915542464
DEBUG (0): Led 1 toggle
DEBUG (0): Basestation: received packet from 1 with 0x1 at 0:0:3.923580779
DEBUG (0): Led 1 toggle
DEBUG (2): Node: sending 0x2 to basestation 0:0:7.812816554
DEBUG (4): Node: sending 0x2 to basestation 0:0:7.812957667
DEBUG (0): Basestation: received packet from 4 with 0x2 at 0:0:7.817489511
DEBUG (0): Led 2 toggle
DEBUG (1): Node: sending 0x2 to basestation 0:0:7.819012351
DEBUG (0): Basestation: received packet from 2 with 0x2 at 0:0:7.820583242
DEBUG (0): Led 2 toggle
DEBUG (0): Basestation: received packet from 1 with 0x2 at 0:0:7.829174655
DEBUG (0): Led 2 toggle
DEBUG (2): Node: sending 0x3 to basestation 0:0:11.719066554
DEBUG (4): Node: sending 0x3 to basestation 0:0:11.719207667
DEBUG (0): Basestation: received packet from 2 with 0x3 at 0:0:11.723079602
DEBUG (0): Led 1 toggle
DEBUG (0): Led 2 toggle
DEBUG (1): Node: sending 0x3 to basestation 0:0:11.725262351
DEBUG (0): Basestation: received packet from 4 with 0x3 at 0:0:11.729919284
DEBUG (0): Led 1 toggle
DEBUG (0): Led 2 toggle
```