VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

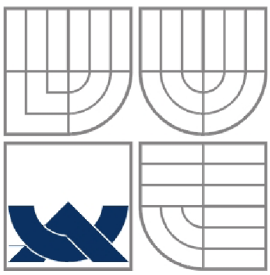
MONITOROVÁNÍ BUDOVY POMOCÍ BEZDRÁTOVÉ SENZOROVÉ SÍTĚ SE STATICKOU TOPOLOGIÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

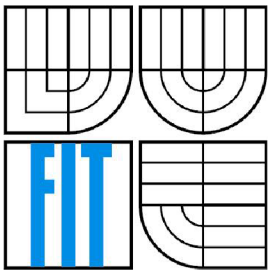
AUTOR PRÁCE
AUTHOR

JAN KOUDELKA

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 SE STATICKOU TOPOLOGIÍ

WIRELESS SENSOR NETWORK WITH STATIC TOPOLOGY FOR BUILDING MONITORING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN KOUDELKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN SAMEK, Ph.D.

BRNO 2012

Abstrakt

Tato bakalářská práce je zaměřena na realizaci bezdrátové senzorové sítě (WSN) pro monitorování budovy a získávání dat v reálném čase. Pro práci na projektu jsou zvoleny senzorové uzly micaZ. Náplní práce je vytvoření sítě se statickou topologií a tomu odpovídající návrh způsobu komunikace. Takto navržená síť se skládá z několika uzlů, z nichž jeden vykonává funkci base station (základny) připojené přes USB rozhraní k počítači, na kterém budou získaná data zpracována a vyhodnocena. Kromě návrhu samotného je úkolem implementovat navržený protokol komunikace mezi uzly a přeposílání dat, dále pak vytvořit obslužné programy pro senzorové uzly a aplikaci, která bude získaná data průběžne vyhodnocovat a textově i graficky zobrazovat. Součástí práce je navíc úvod a základní informace k simulačnímu nástroji TOSSIM, který umožnuje testování vyvíjených aplikací bez nutnosti fyzické instalace na uzlech micaZ.

Abstract

This bachelor's thesis is focused on realization of wireless sensor network (WSN) for buildings monitoring and real-time obtaining of data. Just for work on this project the micaZ sensor nodes have been chosen. The main task is to create WSN with a static topology and some communications mechanism. That network consists of several nodes, and there is a special one called base station. It also has a special function, different from the others. The function is that the base station is collecting data from all of the nodes and sending them to a computer through an USB port. In addition to designing the network topology the other tasks are to implement a comunication protocol and to create computer application and application for sensor nodes. Data will be visualised in text and graphical form. Moreover there are some information about the simulation tool called TOSSIM which allows testing applications without having a physical access to micaZ sensor nodes.

Klíčová slova

Bezdrátové senzorové sítě, měření veličin, ZigBee, nesC, TinyOS, topologie, TOSSIM, uzly micaZ, monitorování.

Keywords

Wireless sensor networks, measurements, ZigBee, nesC, TinyOS, topology, TOSSIM, micaZ nodes, monitoring.

Citace

KOUDELKA, Jan: *Monitorování budovy pomocí bezdrátové senzorové sítě se statickou topologií*, bakalářská práce, Brno, FIT VUT v Brně, 2012

Monitorování budovy pomocí bezdrátové senzorové sítě se statickou topologií

Prohlášení

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

……
Jan Koudelka
16.května 2012

Poděkování

Rád bych touto cestou poděkoval všem, kteří mě při psaní této bakalářské práce podporovali. Zvláštní poděkování pak patří vedoucímu práce Ing. Janu Samkovi, Ph.D. za jeho odborné rady a pomoc poskytnutou při řešení problémů, pravidelné konzultace a dobré vedení k úspěšnému dokončení práce.

Obsah

1 Úvod

Jak již titul publikace napovídá, zaměřuje se tato bakalářská práce na téma monitorování budovy a to konkrétně s využitím bezdrátových senzorových uzlů. Tato slova však mohou ve čtenáři vyvolat jakousi mylnou představu. Uveďme tedy hned v úvodu věci na pravou míru. Zmíněné slovo monitorování zde vystupuje ve významu soustavného sledování a vyhodnocování dat. V rámci této práce a v kontextu použitých technologií však nemá nic společného se zabezpečovacími nebo kamerovými systémy, jak si může čtenář na první pohled myslet, nýbrž jde pouze o statistické vyhodnocování sbíraných veličin v reálném čase. Když už padla zmínka o využití uzlů pro bezpečnostní systémy, nutno říci, že i tato možnost připadá v úvahu a takové systémy se skutečně vyskytují. Toto téma by však bylo na samostatnou práci a proto jej nebudeme dále rozvádět.

Projekt, který tato práce dokumentuje je zaměřen na vytvoření bezdrátové senzorové sítě se statickou topologií. Rozmístění uzlů je tedy známé již při návrhu topologie a dále neměnné. Úlohou je vytvoření topologického modelu sítě včetně způsobu komunikace mezi jednotlivými uzly a návrh jednoduchého protokolu pro směrování sesbíraných informací z různých uzlů sítě směrem k tzv. base station. Tato stanice je zvláštním typem uzlu v síti, který shromažďuje data od ostatních prvků v sítí a přes USB rozhraní komunikuje s hostitelským počítačem. Na tomto počítači jsou pak data vyhodnocována a vhodně zobrazena. Dílčími úkoly jsou tedy návrh topologie a protokolu, návrh a implementace aplikací pro senzorové uzly a base station a dále aplikace pro zobrazení naměřených hodnot.

Pro realizaci projektu, který je předmětem této práce byly zvoleny senzorové uzly MicaZ firmy MEMSIC Inc. Jedná se o relativně malá (v jednotkách cm) přenosná zařízení s bezdrátovou technologií ZigBee postavené na standardu IEEE 802.15.4. Tyto uzly jsou využitelné jak pro monitorování uvnitř budov, tak i pro venkovní použití. Typické nasazení těchto technologií je např. v tzv. intelignetních budovách jejichž řízení je zcela v režii počítačových, případně vestavěných systémů. Můžeme se však setkat s instalacemi i v těch nejextrémnějších podmínkách, jako jsou ku příkladu sopečné krátery, centra zemětřesení apod., kde v kombinaci s řadou rozšiřujících senzorových desek a dalšího vybavení mohou napomáhat při sbírání dat pro různé vědecké výpočty.

Uzly MicaZ jsou založeny na platformě operačního systému TinyOS, který je určen mimo jiné právě bezdrátovým senzorovým sítím. Stejně tak jako aplikace určené pro senzorové uzly je i TinyOS napsán v jazyce nesC, který je do značné míry odvozen z jazyka C a byl přizpůsoben požadavkům bezdrátových senzorových sítí.

Práce je určena především uživatelům se středně pokročilejšími znalostmi v oboru informatiky a programovacích jazyků. Ačkoliv se nejedná o těžkou látku, jsou k jejímu pochopení požadovány určité prerekvizitní znalosti, jako např. znalost základních programovacích principů.

Kapitola druhá, následující po tomto úvodu, se věnuje obecným základům a popisu bezdrátových senzorových sítí. Dále následuje stručný úvod do hardwarového vybavení používaného v senzorových sítích, jímž se zabývá kaptiola číslo tři. Zde jsou představeny především prvky použité při návrhu a realizaci sítě v rámci tohoto projektu. Další část práce pak přináší úvod do síťových protokolů v bezdrátových senzorových sítích a poté přechází v konkrétní návrh síťové topologie. Mimo jiné je zde znázorněn grafický návrh senzorové sítě, podle kterého bude tato sestrojena a testována. Kapitola pátá je zaměřena na samotný vývoj senzorové sítě a navíc obsahuje informace o možnostech simulace nástrojem TOSSIM. V této kapitole jsou vyzdviženy některé důležité úseky programu pro base station a pro ostatní uzly v síti. Předposlední šestá část zahrnuje informace o průběhu a způsobu testování a dále o zpracování dat a jejich následné vizualizaci v podobě grafického webového rozhraní. Tatáž kapitola poskytuje ve své poslední části kritické zhodnocení kladů a nedostatků. Na závěr následuje krátké shrnutí a zamyšlení nad dalším využitím, případně rozšířením dané aplikace a jejím použití v praxi.

2 Bezdrátové senzorové sítě (WSN)

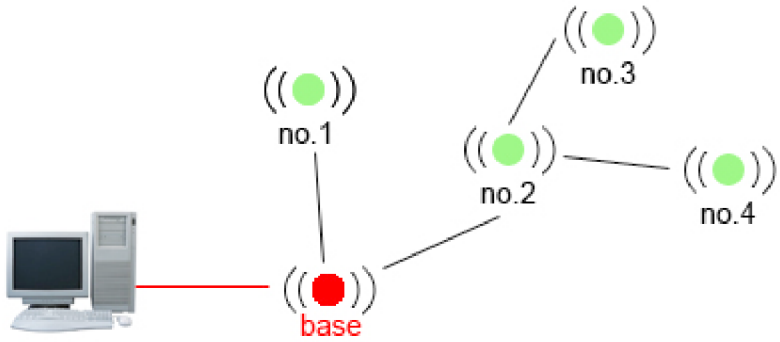
Účelem bezdrátových senzorových sítí, často označovaných zkratkou WSN (z angl. wireless sensor networks), je plošné monitorování budov, ale i venkovních oblastí na otevřených prostranstvích za účelem získávání dat z různých míst monitorovaného objektu. Na základě takto získaných dat jsou typicky odvozovány statistiky informující uživatele o stavu systému (objektu) v reálném čase. Původně byly tyto sítě vyvíjeny jako armádní projekt, kdy byly využívány např. k monitorování bitevních území. V dnešní době u nás bohužel stále nejsou tyto technologie příliš rošířené, přesto je některé moderní firmy s úspěchem využívají. Z hlediska výzkumu nových technologií a aplikací jsou však WSN stále více zmiňovaným tématem a snad se jim i u nás dostane větší pozornosti.

2.1 Obecný popis WSN

Senzorové sítě jsou systémem obecně několika uzlů vybavených senzorovými deskami pro snímání různých veličin. V základním provedení se typicky jedná o snímače teploty, vlhkosti, osvětlení, tlaku, apod. Existuje však větší škála senzorových desek obsahující např. čidla zvuku, akcelerometry nebo třeba moduly s technologií GPS, umožňující krom jiného např. snadnou lokalizaci uzlů s následným zobrazením na reálné mapě monitorované oblasti.

Jelikož se jedná o bezdrátové sítě, jsou v nich uzly vybaveny také moduly pro bezdrátový přenos informací k centrálnímu prvku (base station), případně pro vzájemnou komunikaci mezi sebou. Ve většině případů se jedná o sítě se statickou topologií, tzn. rozmístění uzlů je předem známé a monitorovací aplikace jsou tedy přímo stavěny na požadavcích konkrétního případu užití. Ne jinak je tomu i v tomto projektu jehož topologií a routovacími protokoly se budeme zabývat v následujících kapitolách. Typicky jsou WSN navrhovány a realizovány za použití stejných uzlů. Tím rozumíme uzly se stejnými parametry bezdrátového přenosu, stejného dosahu, většinou vybaveny i stejnými senzorovými snímači.

Následující obrázek znázorňuje jednoduché schéma bezdrátové senzorové sítě se statickou topologií o čtyřech uzlech a base station připojenou k cílovému počítači, na němž jsou data vyhodnocována.



Obr.1: Základní schéma bezdrátové senzorové sítě[1].

1 Toto schéma je pouze ilustrační a vysvětluje základní principy WSN. V praxi se často jedná o rozsáhlé sítě obsahující i stovky senzorových uzlů.

2.2 Vlastnosti a využití

V této části jsou popsány obecné vlastnosti senzorových sítí z hlediska struktury, spotřeby energie a architektury jednotlivých zařízení. Poté následuje výčet různých způsobů a oblastí, ve kterých bývají, případně mohou být použity.

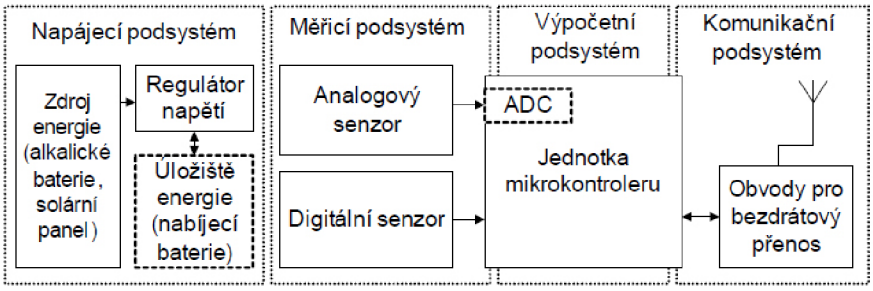
2.2.1 Vlastnosti senzorových sítí

Tato podkapitola čerpá informace z těchto zdrojů [1,2].

Mezi hlavní požadavky kladené na vývoj bezdrátových senzorových sítí se řadí především nízká pořizovací cena senzorových uzlů, nízká spotřeba energie (tzv. low-power zažízení), snadná instalace zařízení v síti, jednoduché ovládání a mobilita a možnosti vzdálené kontroly nad jednotlivými uzly.

Zdrojem energie v uzlech jsou ve většině případů dvě baterie typu AA. Cílem je pak vyvíjet aplikace tak, aby byli eliminováni největší spotřebitelé energie, případně najít jiné řešení daného problému, které je energeticky úspornější a prodlužuje tak provozní dobu zařízení. Průměrná doba provozu může být různá v závislosti na běžící aplikaci. Může se pohybovat řádově od několika dnů do měsíců, v krajních případech až roků.

Architektura uzlů, ze kterých WSN sestávají se dá rozdělit do několika podsystémů. Jedná se o část zabezpečující napájení celého uzlu a regulaci napětí, dále pak obvody určené k měření zkoumaných veličin (integrované nebo připojené na sensorové desce), výpočetní podsystém obsahující AD převodník, který se stará o zpracování a převod sesbíraných informací v uzlu a nakonec ještě bezdrátový modul, zajišťující komunikaci s dalšími zařízeními. Jednoduché schéma takové architektury znázorňuje obrázek níže (Obr.2).



Obr.2: Schéma bezdrátového senzorového uzlu [1].

2.2.2 Možnosti využití senzorových sítí

Oblastí, ve kterých je možné použít senzorové sítě existuje celá škála. Uveďme si tedy některé příklady z praxe, které mohou být člověku užitečné.

Monitorování budov

Pro případ monitorování budov je možné využití senzorových sítí za účelem zabezpečení, řízení spotřeby energií, regulace osvětlení, teploty, apod. V tomto směru jsou nejčastěji řešenou problematikou tzv. inteligentní systémy, které nachází uplatnění v továrnách, výrobních halách a firmách, ale také v některých domácnostech. Jedná se např. o regulaci teploty v místnostech na základě průběžných měření a propojení se systémem klimatizace a topení. V dnešní době je vcelku široce rozvíjející se technologií automatické řízení osvětlení na základě světelných podmínek v

místnosti. S monitorováním budov také dále souvisí různé zabezpečovací systémy, kde opět mohou hrát velkou roli senzorové uzly opatřené čidly pro detekci pohybu.

Vědecké experimenty

V oblasti vědeckého výzkumu a experimentů je hlavní výhodou a důvodem k použití senzorových uzlů jejich snadná instalace a možnost kontrolovat stav zařízení na větší vzdálenosti. Těchto výhod pak lze využít při měřeních v terénu a na špatně dostupných místech, kam jiná technika nemůže, popřípadě na místech, kde je pohyb pro člověka nebezpečným. Ve vědeckých kruzích tedy bezdrátové senzorové sítě mohou nalézat uplatnění při měření seismické aktivity a nebo například monitorování vulkanické činnosti.

Záchranné systémy

Do této kategorie řadíme WSN, které slouží k monitorování stavu určitých objektů a v případě hrozícího nebezpečí odesílají varovné signály. V této oblasti stojí za zmínku především detektory kouře (požáru). Tyto sytémy v sobě většinou mívají zabudován GSM modul, který v případě detekce požáru odesílá zprávu přímo na centrum záchranných služeb. Tím je samozřejmě šetřen čas a zásah požárních jednotek se tím urychlí, čímž se může předejít větším škodám na majetku.

Monitorování průmyslových zařízení

Jako poslední uveďme užití WSNs v průmyslovém provozu. Zde mohou zastávat roli jakýchsi dozorců, kdy dohlíží na správný chod jiných zařízení a v případě jejich výpadků, či poruchy oznamují tuto událost řídicímu systému, případně zaměstnancům technické údržby.

Možností aplikace bezdrátových senzorových sítí je samozřejmě celá řada, proto byly vybrány a zmíněny jen ty nejběžnější případy užití a aplikace v dnešní době v praxi běžně používané.

2.3 Operační systém TinyOS

Následující kapitola je věnována TinyOS, jakožto operačnímu systému, který je ve velké míře používán právě pro senzorové sítě a vestavěné systémy. Obsahuje stručnou charakteristiku obecných předpokladů pro operační systémy senzorových sítí a dále se pak zaměřuje přímo na zvolený TinyOS.

2.3.1 Charakteristika OS pro senzorové sítě

Operační systémy určené k nasazení na uzlech senzorových sítí musí splňovat určité parametry, které vychází z požadavků kladených na vývoj těchto uzlů. Protože se většinou jedná o malá přenosná zařízení, vyvíjená tak, aby bylo dosaženo co nejnižšší pořizovací ceny, jednoduchosti instalace a použití, je nutné tomu adekvátně přizpůsobit i jejich softwarové vybavení.

Výše zmíněná kriteria tedy naprosto vylučují ze hry komplexní operační systémy, jež běžně používáme například při práci na PC. Naopak do hry vstupují softwarově méně složité systémy, které staví na jednoduchosti výpočetních operací v hardware senzorových uzlů. O těchto systémech můžeme říci, že se jedná o jistou formu vestavěných systémů (tzv. embedded). Při vývoji WSN se také často zaměřujeme pouze na určité výpočetní či měřící techniky a vyvíjíme aplikace pro nasazení na jednom konkrétním místě, což může vést k dalšímu zjednodušení požadavků na použitý systém.

2.3.2 Popis systému TinyOS

TinyOS je operační systém, který vznikl na půdě kalifornské univerzity v Berkeley. Již od svého vzniku byl určen k aplikaci v tzv. low-power zažízeních, senzorových sítích a inteligentních systémech. V počátcích byl vyvíjen jako výzkumný projekt, na kterém se podíleli studenti University of California, Berkeley. Dnes obsahuje komunita vývojářů programátory z různých koutů světa a neustále se rozrůstá. Operační systém TinyOS je šířen pod BSD licencí a počet stažení se udává na cca 35 000 / rok [3].

Aplikace pro tinyOS jsou vyvíjeny v jazyce nesC, který je odnoží jazyka C, optimalizovaného hardwarovým vlastnostem zařízení, na nichž TinyOS běží. V samotném jazyku C je také napsána většina knihoven nesC.

Programy v TinyOS jsou složeny z tzv. komponent (components), které jsou vzájemně propojovány pomocí rozhraní (interfaces). K dispozici jsou rozhraní a komponenty pro bezdrátovou komunikaci, směrování, měření veličin, běžnou práci s daty, a další.

Operace vykonávané v programech se zde nazývají události (events) a jsou vykonávány asynchroně. Tyto události je možné spojovat do větších celků – tzv. tasks (úlohy). Jednotlivé úlohy pak mohou být systémem různě plánovány a prováděny dle potřeby. Při překladu je spolu se zdrojovými kódy dané aplikace přilinkován i TinyOS a výsledek je uložen do jednoho celku v binární podobě.[3]

3 Hardware v senzorových sítích

Tato část práce poskytuje základní přehled o hardwarovém vybavení, se kterým se v senzorových sítích můžeme setkat. Blíže se pak zaměřuje na zařízení použitá v této práci pro vybudování bezdrátové senzorové sítě. Jelikož není záměrem práce soustředit se detailně na hardware v těchto zařízeních, nabízí tato kapitola pouze základní technické údaje k jednotlivým výrobkům. Další informace jsou však ve většině případů k nalezení na internetových stránkách výrobců.

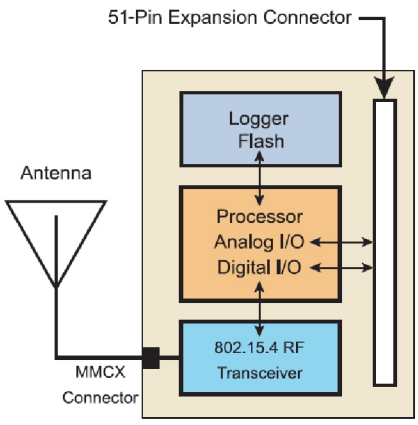
3.1 Senzorové uzly MicaZ

V rámci tohoto projektu jsou pro implementaci bezdrátové senzorové sítě využity uzly Micaz, vyvíjené původně firmou Crossbow(Xbow), jejíž výroba přešla po akvizici v lednu roku 2010 kompletně pod firmu MEMSIC. Tyto uzly poskytují rozhraní pro bezdrátovou komunikaci a rozhraní pro připojení senzorových desek k měření různých druhů veličin.

Pro bezdrátovou komunikaci je v tomto případě použito technologe ZigBee na standardu IEEE 802.15.4 s pracovní frekvencí 2,4 GHz. Tento komunikační modul disponuje rychlostí přenosu dat až 250 kbps a hardwarovým zabezpečením v podobě šifrování AES-128.

Uzly obsahují mikrokontroler Atmel ATmega128L a ve srovnání s jinými senzorovými uzly mají téměř nejmenší kapacitu konfigurační EEPROM paměti (4KB). Naopak programová flash paměť má kapacitu 128KB, čímž se řadí mezi uzly s větší kapacitou paměti. Detailní srovnání těchto pamětí a použitých mikrokontrolerů lze vyčíst z níže uvedeného obrázku (Obr.5).

Na obrázku (Obr.4), který znázorňuje senzorvý uzel MicaZ je dobře vidět rozhraní pro připojení různých periferií. Je to 51-pinová patice podporující připojení přídavných modulů.
Pomocí tohoto rozhraní je možné připojovat různé analogové výstupy, digitální vstupně/výstupní zařízení, I2C, SPI nebo UART. Jednoduché schéma takového uzlu zobrazuje obrázek (Obr.3), na kterém jsou ve zjednodšené formě popsány hardwarové části a vysvětleny propojení mezi jednotlivými komponentami uzlu.



Obr.3: Jednoduché schéma uzlu Micaz [5].

Obr.4: Hardwarová realizace uzlu MicaZ [5].

Uzly Micaz jsou určeny především k použití pro monitorování budov a zabezpečovací systémy. Jsou vhodné i pro větší senzorové sítě obsahující tisíc a více uzlů. Na cestě shromažďovaných dat z jednotlivých uzlů a počítačem stojí prostředník, který poskytuje rozhraní pro přenos dat případně pro řízení uzlů z počítače. Jedná se o tzv. base station, což je zařízení, díky kterému mohou být data vyhodnocována, ale je to zároveň prostředek, pomocí kterého jsou jednotlivé

uzly programovány. V případě uzlů MicaZ to mohou být například desky MIB510 nebo MIB520[2] opatřené rozhraním USB pro připojení k počítači. Firma MEMSIC však poskytuje řadu dalších produktů, které lze zapojovat třeba i přímo do LAN sítě a komunikovat přes TCP/IP protokol. Následující obrázek (Obr.5) uvádí srovnání nejpoužívanějších senzorových uzlů. Zaměřuje se na porovnání pamětí, typů použitých mikrokontrolerů a technologií pro bezdrátový přenos.

Název senzoru	Mikrořadič	Vysílač/přijímač	Paměť	Externí paměť (Flash)
BTnode	Atmel ATmega128L	TI[7] Chipcon CC1000 a Bluetooth	64+180 KB RAM	128 KB Flash, 4 KB EEPROM
EyesIFX v1	TI MSP430 F149	IF[8] TDA5250	*	8 MB
EyesIFX v2	TI MSP430 F1611	IF TDA5250	*	8 MB
IMote	ARM[9] 7TDMI, 12 MHz	Bluetooth	64 KB SRAM	512 KB
IMote 1.0	ARM 7TDMI, 12-48 MHz	Bluetooth	64 KB SRAM	512 KB
IMote 2.0	Marvell PXA271, 11-400 MHz	TI Chipcon CC2420	32 MB SRAM	32 MB
Intel SHIMMER	TI MSP430 F1611	TI Chipcon CC2420	10 KB RAM	48 KB
Iris	Atmel ATmega1281	Atmel AT86RF230	8 KB RAM	128 KB
KMote	TI MSP430	**	10 KB RAM	48 KB
Mica	Atmel ATmega103	RFM TR1000	128+4 KB RAM	512 KB
Mica2	Atmel ATmega128L	TI Chipcon CC1000	4 KB RAM	128 KB
Mica2Dot	Atmel ATmega128	TI Chipcon CC1000	4 KB RAM	128 KB
MicaZ	Atmel ATmega128	TI Chipcon CC2420	4 KB RAM	128 KB
Rene	Atmel ATmega8535	RFM TR1000	512 B RAM	8 KB
SenseNode	TI MSP430 F1611	TI Chipcon CC2420	10 KB RAM	48 KB
TelosB	TI MSP430	TI Chipcon CC2420	10 KB RAM	48 KB
T-Mote Sky	TI MSP430 F1611	TI Chipcon CC2420	10 KB RAM	48 KB

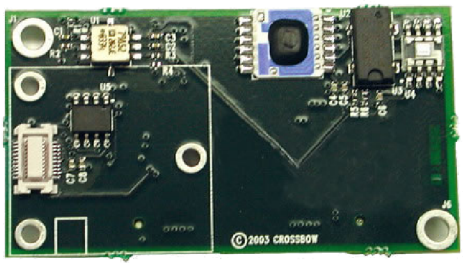
Obr.5: Porovnání hardwarových vlastností senzorových uzlů [4].

Podkapitola 3.1 čerpá informace především z těchto zdrojů [4,5].

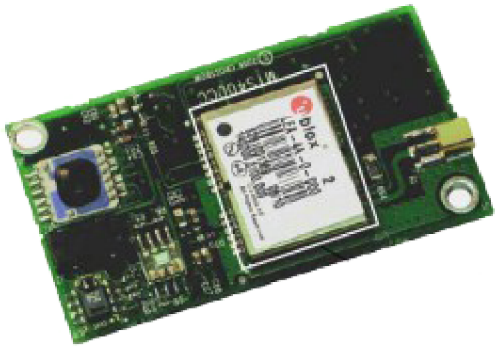
2 MIB510 a MIB520 jsou programovací rozhraní pro senzorové uzly. Obrázek tohoto zařízení je možné najít v kapitole 5.3.2 (Obr.16), která se zaměřuje na překlad aplikací a jejich instalaci na uzly.

3.2 Senzorové desky MTS400CA / MTS420CA

V předchozích kapitolách bylo zmíněno, že k senzorovým uzlům můžeme pomocí 51-pinového rozhraní připojovat různé typy senzorových desek, které mohou obsahovat široké spektrum rozličných senzorů a čidel. Příkladem mohou být například desky MTS400 nebo MTS420, jež se řadí mezi nejčastěji používané právě v kombinaci s uzly MICAz. Obě tyto desky jsou uvedeny na následujících obrázcích (Obr.6) a (Obr.7). Rozdíl v těchto dvou produktech je pouze v přítomnosti GPS modulu u druhého zmíněného. Tento je také na obrázku (Obr.7) vidět.



Obr.6: Senzorová deska MTS400[3].



Obr.7: Senzorová deska MTS420[4].

Pomineme-li GPS modul, nacházející se pouze u modelu MTS420, pak obě tyto desky obsahují stejné senzory. Mezi těmi můžeme nalézt např. senzory snímající teplotu, tlak, vlhkost vzduchu, akcelerometr nebo čidlo intenzity světla. Konkrétní typy a značky použitých senzorů následují dále.

Produkty MTS400CA a MTS420CA poskytují pět senzorů, použitelných pro aplikování v oblastech zemědělství, průmyslu a monitorování životního prostředí.

3.2.1 Senzory použité v aplikaci

Výčet konkrétních typů a značek použitých senzorů na výše zmíněných deskách ukazuje následující tabulka (Tab.1). Z těchto senzorů byly pro vyvíjenou aplikaci vybrány dva z nich. Jedná se o senzor Sensirion SHT11 a TAOS TSL2550.

Senzor	**Popis**
Sensirion SHT11	Senzor pro snímání teploty a vlhkosti vzduchu; obsahuje AD převodník; poskytuje kalibrovaný digitální výstup
TAOS TSL2550	Senzor snímající intenzitu osvětlení. Obsahuje dva kanály. Měří intenzitu viditelného světla a infračerveného záření.
Intersema MS55ER	Senzor pro měření barometrického tlaku a teploty. Narozdíl od Sensirion je potřeba zařízení kalibrovat
ADXL202JE	Akcelerometr – měří pohyb uzlu ve dvou osách X a Y (2 kanály), analogové rozhraní [13]
Leadtek GPS-9546	GPS modul připojený na sériové rozhraní USART1 [13]

Tab.1: Tabulka senzorů použitých na deskách MTS400CA / MTS420CA.

3 Zdroj: http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/MTS420.html.
4 Zdroj: http://www.tinyosmall.co.kr/shop/shopdetail.html?branduid=12.

Senzor Sensirion SHT11

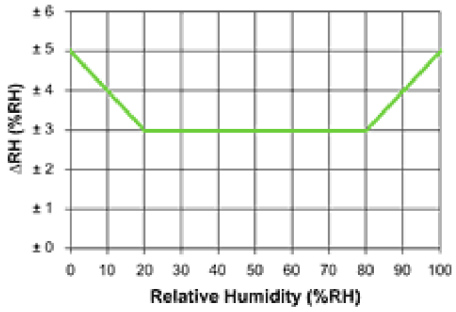
Ve vyvíjeném projektu se využívá obou čidel tohoto senzoru (teplota i relativní vlhkost). Následuje výčet některých důležitých technických parametrů senzoru. Tyto parametry jsou čerpány z těchto zdrojů [13,14]. Obrázky (Obr.8) a (Obr.9) znázorňují přesnost výsledků měření vzhledem k rozsahu, ve kterém tato čidla pracují.

Relativní vlhkost:

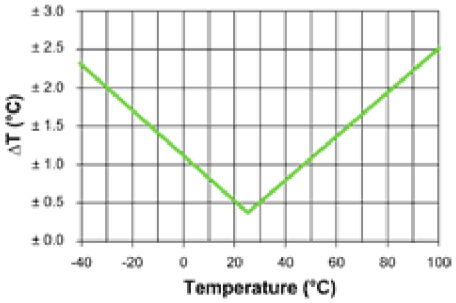
- rozsah: 0 – 100%
- přesnost: +- 3.5 %
- potřebné napětí: 3.6 – 2.4 V

Teplota:

- rozsah: -40°C – 80°C
- přesnost: +- 2°C
- potřebné napětí: 3.6 – 2.4 V



Obr.8: Přesnost měření relativní vlhkosti [14].



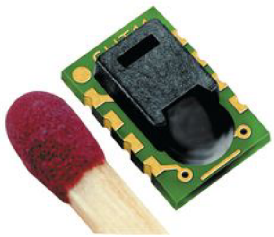
Obr.9: Přesnost měření teploty [14].

Senzor TAOS TSL2550

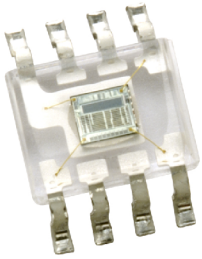
Jedná se o digitální senzor intenzity osvětlení s SMBus[5] sériovým rozhraním. Obsahuje dvě fotodiody a vestavěný AD převodník. Fotodiody jsou citlivé na viditelné a infračervené záření [13].

Parametry:

- rozsah: 400 – 1000nm
- potřebné napětí: 3.6 – 2.7 V
- pracovní teplota: -40°C - 80°C



Obr.10: Porovnání velikosti Sensirion SHT11[6].



Obr.11: Senzor osvětlení TSL2550[7].

5 SMBus = System Management Bus – rozhraní vyvinuté firmou Intel pro nízkofrekvenční komunikace (zdroj: http://smbus.org).
6 Zdroj: http://cz.farnell.com/sensirion/sht11/sensor-humidity-temp-v4/dp/1590511.
7 Zdroj: http://www.taosinc.com/ProductDetails.aspx?id=133.

4 Návrh topologie a síťové komunikace

V předchozích kapitolách bylo zmíněno, že bezdrátové senzorové sítě se mohou skládat řádově i z tisíců uzlů. V těchto případech je pak kladen důraz na hledání nejlepších cest, kudy směřovat data z koncových uzlů do base station a další optimalizace síťového provozu. Při takto vysokém počtu přenášených paketů je častým problémem, který musí být řešen zahlcení paměti uzlu a neschopností přijímat/odesílat další data. Nejčastějším řešením pro tyto sítě jsou dynamické routovací protokoly, díky kterým lze v síti průběžne hledat nejoptimálnější řešení pro posílání paketů. Tyto protokoly většinou řeší i výše zmíněné problémy a to tím způsobem, že uzly mezi sebou neustále udržují aktuální informace o svém stavu. Tato informace je poté zohledněna při rozhodování o směrování datových paketů.

Náplní této práce však není návrh takto rozsáhlé sítě, nýbrž konstrukce senzorové sítě o velmi malém počtu (v řádu jednotek) uzlů, na které budou vyzkoušeny techniky programování na platformě TinyOS. V rámci takto malé sítě je tedy použití dynamických routovacích protokolů ve své podstatě zbytečné a možná i z tohoto důvodu byla pro zadání práce preferována statická topologie sítě.

4.1 Typy protokolů používaných v TinyOS

Jak již bylo řečeno, v tomto projektu se budeme zabývat pouze statickou topologií sítě, proto se o dynamických protokolech zmíním pouze několika větami. Existuje řada implementací dynamických protokolů, zde bych však zmínil jeden, jehož výskyt při studii síťových protokolů byl relativně častý v různých dokumentech.

Jedná se o TYMO protokol nebo některé jeho modifikace. Tato zkratka značí použití DYMO protokolu v TinyOS. Jedná se o tzv. multi-hop protokol, pomocí kterého jednotlivé uzly v síti získávají routovací informace dynamicky prostřednictvím zasílání požadavků ke zjištění nejlepší vhodné cesty do okolních uzlů sítě. Tento způsob se nazývá Dissemination, případně Flooding [6]. Technika Dissemination je ve své podstatě samostatným protokolem, často kombinovaným s technikou zvanou Collection.

Již bylo naznačeno, že Dissemination je protokolem používaným pro zasílání paketů na více uzlů současně (většinou na všechny uzly v síti). Toho může být využito k určitým kontrolním nebo řídícím operacím. Tímto způsobem je také možno do jisté míry přeprogramovávat uzly za běhu [7]. Druhý zmíněný protokol je dá se říct opakem k tomu předchozímu. Slouží ke shromažďování dat ze všech uzlů sítě směrem do base station. V tomto případě jsou vytvářeny stromy, po kterých jednotlivé uzly směřují své pakety, případně přeposílají obdržené pakety. Kořen každého takového stromu pak představuje base station, kde jsou data shromážděna a dále vyhodnocována [7]. Tento projekt se do jisté míry tímto protokolem inspiruje a bude popsán v následujících kapitolách.

4.2 Návrh sítě se statickou topologií

Přejděme nyní ke konkrétnímu návrhu bezdrátové senzorové sítě. Naše síť bude mít statickou topologii. Rozmístění jednotlivých uzlů tedy bude neměnné a každý prvek bude mít informaci o tom kudy směřovat data do base station pevně danou. Tato konkrétní síť je navržena a bude testována v prostorách budovy Fakulty informačních technologií Vysokého učení technického v Brně. Dále uvedený model vychází z vlastnostní senzorových uzlů MicaZ a ze situačních parametrů budovy fakulty. Při návrhu jsou brány v úvahu technické parametry a to především dosah radiového signálu použitých uzlů.

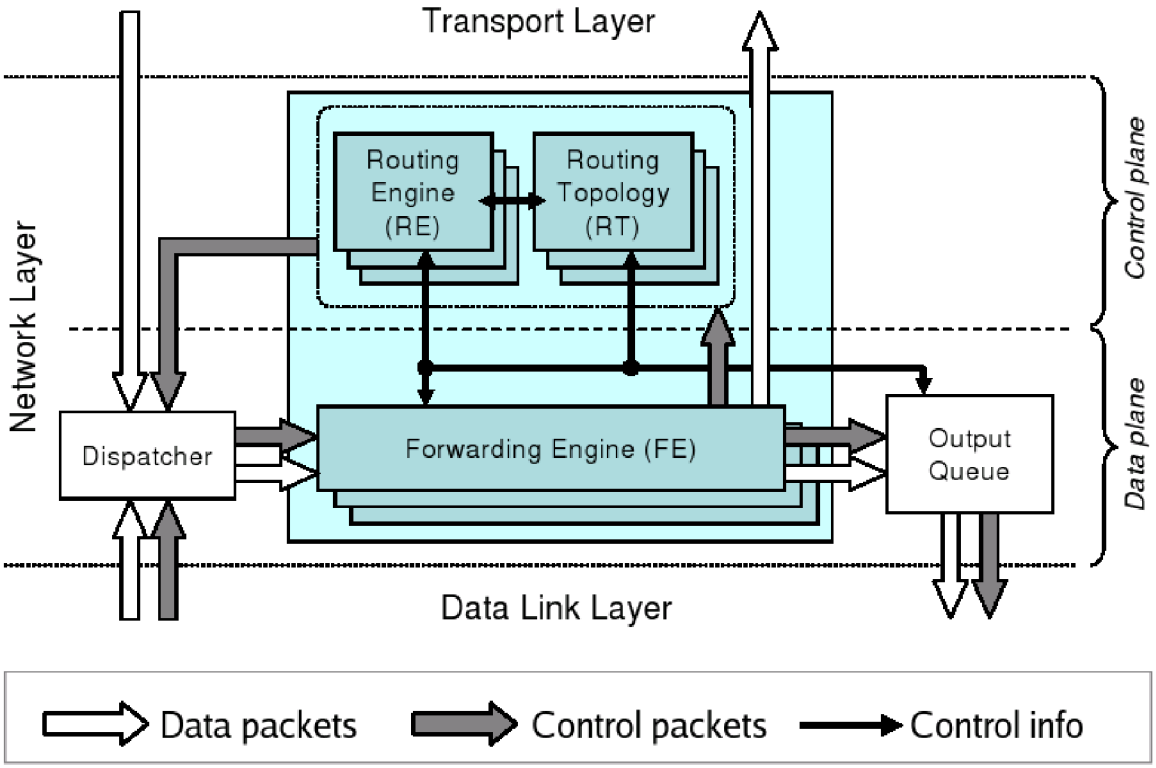
Samotný návrh však nepočítá s žádným rušením či jinými vlivy, které by mohly narušit správný chod celého systému. Situace uvedená v návrhovém modelu tedy nemusí v konečné fázi zcela odpovídat realitě. Proto bude v rámci projektu provedeno několik měření, jejichž účelem bude získat bližší informace o optimálním rozmístění jednotlivých uzlů sítě. Tato měření lze však zařadit

až do fáze testování. Ve fázi návrhu se budeme zabývat pouze způsobem komunikace mezi jednotlivými prvky a přibližnou strukturou (topologií) bezdrátové senzorové sítě.

4.2.1 Popis navrhovaného protokolu

Jelikož se bude jednat o protokol ve statické síti o několika málo prvcích (konkrétně 5 uzlech a 1 base station), bylo by zbytečné vymýšlet nové nebo používat již zaběhlé protokoly určené k nasazení v sítích mnohem větších rozměrů. Na místo toho bude použit jednoduchý algoritmus. Jádrem tohoto algoritmu jsou funkce tzv. *collection tree* protokolu jak bude vysvětleno dále. Protože se jedná o velmi jednoduchou topologii, bylo by možné konfigurovat každý uzel zvlášť a určitým způsobem tak částečně zdrojový kód dále optimalizovat. Na druhou stranu by v takovém případě bylo nutné při změnách přeprogramovávat každý jednotlivý uzel samostatně.

Následující obrázkek (Obr.12) je přejat z literatury týkající se dynamického protokolu TYMO zmiňovaného v kapitole 3.1. Ačkoliv se zabýváme statickou topologií (ne dynamickou), může nám toto schéma posloužit k vysvětlení základních principů našeho protokolu.



Obr.12: Síťový model představující propojení jednotlivých vrstev [6].

Ten je postaven na několika vrstvách z nichž pro síťovou komunikaci a práci s přenášenými daty jsou nejdůležitější tyto tři:

- Aplikační vrstva
- Transportní vrstva
- Síťová vrstva

Aplikační vrstva

Hlavní podstatou a úkolem aplikační vrstvy je poskytovat aplikacím přístup ke komunikaci na síťovém rozhraní. Na této úrovni jsou data ve formě, kdy mohou být zpracována a vyhodnocována různými aplikacemi.

Transportní vrstva

Tato vrstva je dá se říct přechodem mezi aplikačními daty a daty posílanými přes síťové rozhraní. Data jsou zde reprezentována již ve formě paketů. Transportní vrstva zaručuje spolehlivost přenosu dat, nestará se však o přenos samotný. Součástí této vrstvy tedy budou např. algoritmy pro potvrzování korektního přijetí paketů od příjemce paketu jeho odesílateli.

Síťová vrstva

Úkolem síťové vrstvy je směrování datových paketů v síti. Tato vrstva je tedy v podstatě hlavní částí navrhovaného protokolu. V našem případě bude o směrování paketů rozhodovat mechanismus protokolu CTP (collection tree protocol). Pro ilustraci paketu v TinyOS je uveden obrázek (Obr.13).

Z obrázku (Obr.12) je možné vyčíst následující postup při zpracování paketů v konkrétním uzlu. Jako první zde vystupuje přijímací rozhraní (DISPATCHER), na které jsou přivedeny pakety. Jak je vidět DISPATCHER je tady řekněme jakousi vstupní bránou. Na tuto bránu jsou tedy přiváděny i pakety vlastního uzlu, které jsou určeny pro směrování na další uzly v síti. Jako další prvek na cestě dat figuruje modul, který má za úkol rozpoznat, zda se jedná o pakety určené tomuto uzlu nebo je zapotřebí dalšího směrování. V našem případě jsou data směrována do jednoho centrálního prvku, kterým je base station. Všechny ostatní uzly by tím pádem neměli přijímat žádné pakety ke zpracování, ale pouze plnit funkci tzv. FORWARDER. Ten ve spolupráci s routovacím algoritmem rozhoduje, kudy data směřovat.

Abychom docílili správného zobrazování měřených hodnot v koncové aplikaci, která bude běžet na stanici připojené k base station, je nutné zajistit, aby všechny uzly v síti posílali aktuální data v reálném čase. Ve stanovené topologii k tomu bude použito některých rozhraní výše zmíněného protokolu. Při výpadku jednoho z uzlů, přes nějž jsou směřována data z ostatních zařízení, bude vyhledána jiná možná cesta (pokud je v dosahu signálu). Tento mechanismus a způsob stanovení tzv. root prvku sítě vypadá následovně:

```
call RoutingControl.start();
if (TOS_NODE_ID == 1)
  call RootControl.setRoot();
```

Zde použitá rozhraní `RoutingControl` a `RootControl` využívají funkcí komponenty `CollectionC`.

Obr.13: Formát paketu v TinyOS.

4.2.2 Použitá rozhraní a komponenty

Pro realizaci síťové komunikace a zasílání dat ve formě paketů je využito kromě vlastních algoritmů již implementovaných komponent a rozhraní TinyOS. Následující řádky se do jisté míry inspirují volně dostupnými dokumentacemi ke zmiňovaným rozhraním.

Jako první jistě stojí za zmínku datová struktura `message_t`, se kterou se budeme v následujících rozhraních setkávat a která se velmi často vyskytuje. Je to základní datová struktura, používaná pro radiovou komunikaci v TinyOS ve verzích 2.x. Nahrazuje svého předchůdce ve verzích TinyOS 1.x, nazývaného TOS_Msg.

Datová struktura message_t

Struktura `message_t` je v TinyOS definována pod tímto umístěním `tos/types/message.h` a vypadá následovně [8]:

```
typedef nx_struct message_t {
    nx_uint8_t header[sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
```

Tato struktura na rozdíl od svých předchůdců uchovává data ve fixní podobě, ať se jedná o použití na jakékoliv platformě. Tím se předchází problémům, které mohou vznikat při přenosu zpráv mezi dvěma odlišnými síťovými rozhraními. Pokud by data měla pro každý typ síťového rozhraní jinou pozici vzhledem k celkovému paketu, jejich zpracování by vyžadovalo navíc operace pro korektní posunutí v paměti na místo výskytu dat [9].

Nutno zmínit, že obsah této struktury kromě datové oblasti, není transparentní. To znamená, že k hlavičce (header), zápatí (footer) a metadatům nelze přistupovat přímo. Existují však rozhraní v jazyce nesC, která přístup k těmto polím umožňují [9].

Základní komunikační rozhraní

Nyní se podívejme na základní komunikační rozhraní a komponenty, které jsou dostupné v TinyOS ve složce `tos/interfaces`.

Packet - Toto základní rozhraní poskytuje přístup k paketům v datové struktuře `message_t`. Přes rozhraní Packet lze operovat s datovou částí, získávat informace o velikosti dat a získávat ukazatele na část nesoucí data.

Send - Funkce rozhraní `Send` je také snadno domyslitelná již z názvu. Jde o odesílací rozhraní, které poskytuje základní operace pro zasílání paketů. Umožňuje kontrolovat, zda byl daný paket korektně odeslán a navíc stejně jako rozhraní `Packet` poskytuje funkce pro přístup k metadatům týkající se datové oblasti paketu.

Receive - A do třetice nám zbývá poslední rozhraní určené k příjmu paketů. Stejně jako předchozí dvě má i toto rozhraní možnosti zjišťování velikosti přijímaných dat a získání ukazatele pro přístup k datové části paketu.

Rozhraní ActiveMessage (AM)

`ActiveMessage` je rozhraním, které bylo vyvinuto kvůli potřebě používat bezdrátový přenos více službami zároveň. Komunikace přes toto rozhraní je časovým multiplexem. Na rozdíl od předchozího základního rozhraní `Send` umožňuje `ActiveMessage` konkrétní adresaci uzlů zadáním konkrétní AM adresy. Všechna rozšiřující rozhraní, mezi která patří např. AMSend a AMPacket se opět nachází na tomto místě – `tos/interfaces`. Blíže se podíváme na tyto dvě:

AMPacket - Ve své podstatě je podobné jako rozhraní `Send`. Posyktuje opět základní operace nad datovým typem `message_t`. Obsahuje příkazy pro získání adresy uzlu, pro zjištění cílové adresy paketu a zjištění typu paketu. Rovněž umožňuje měnit cílovou adresu a typ paketu a zjišťovat zda se cílová adresa neshoduje s adresou aktuálního uzlu.

AMSend - Opět se jedná o podobné rozhraní jako předchozí `Send`. Hlavní rozdíl je v přítomnosti cílové AM adresy.

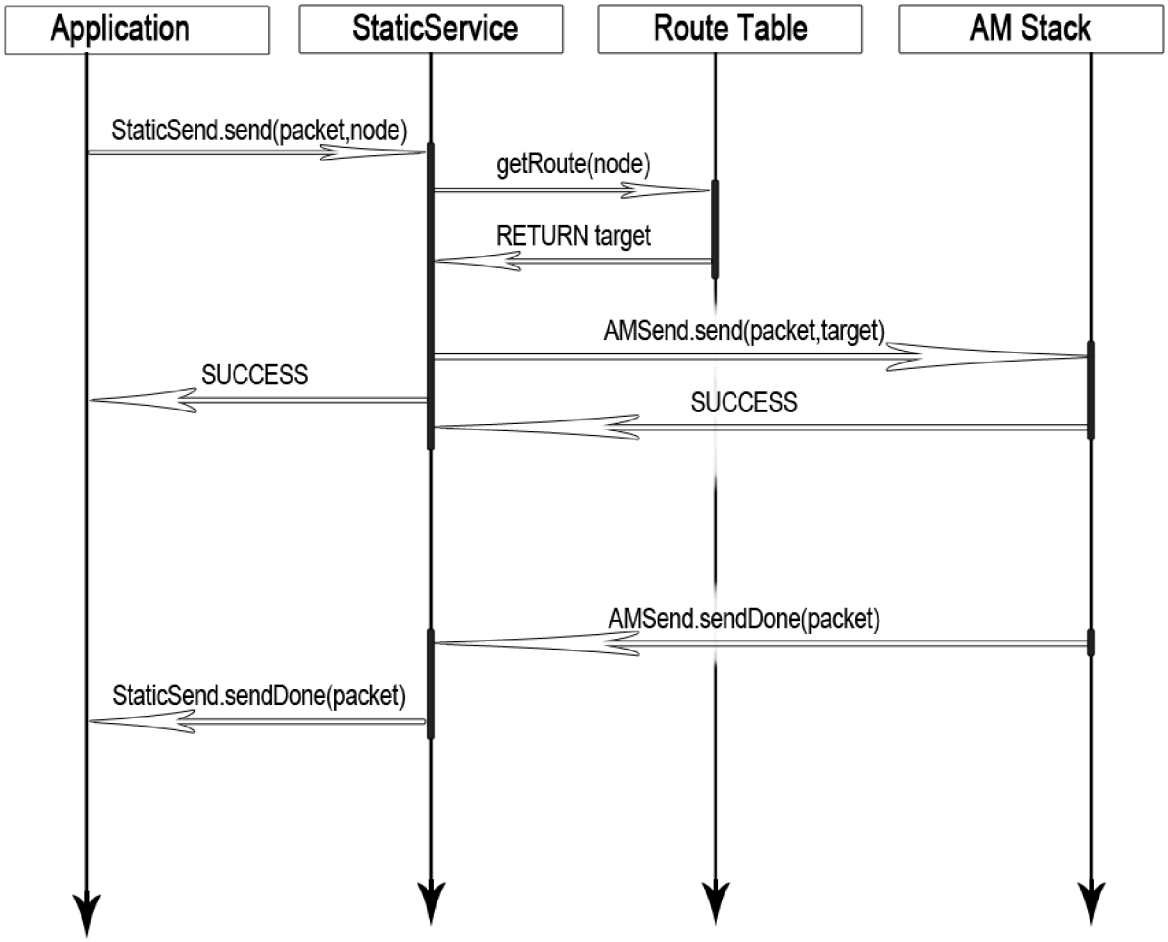
Následují úryvky zdrojového kódu, na kterých je možné porovnat rozdíly v právě zmiňovaných dvou rozhraních `Send` a `AMSend`. Jak lze vidět jediná změna spočívá v uvedení cílové adresy u příkazu `command error_t send.`

```
command error_t send(am_addr_t addr, message_t* msg, uint8_t len);
```

Podkapitola 4.2.2 čerpala informace z těchto zdrojů [8,9].

4.2.3 Komunikace jednotlivých modulů

Následující schéma (Obr.14) znázorňuje návrh základních principů při uskutečnění síťové komunikace. Vidíme že pro odesílání dat je ve všech případech použito události `send`, která navrací konstantu SUCCESS v případě korektního odeslání dat. Po dokončení odesílací fáze je vyvolána událost `sendDone`, ve které můžeme implementovat chování uzlu po dokončení přenosu. Minimálně je vhodné alespoň nějakým způsobem signalizovat tuto činnost, což můžeme provést např. rozsvícením jedné z dostupných LED diod.



Obr.14: Diagram návrhu síťové komunikace.

4.2.4 Situační návrh síťové topologie

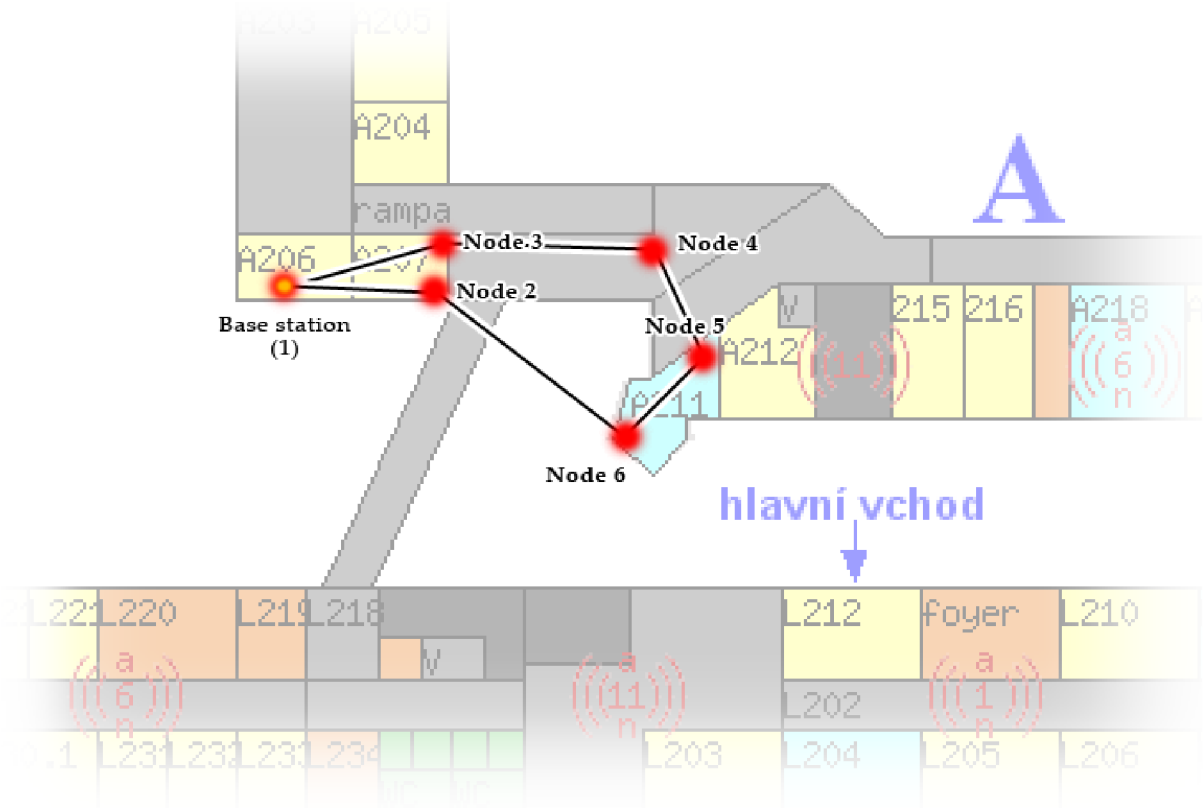
Nyní se podívejme na konkrétní návrh síťové topologie v reálném prostředí, kde bude tato síť testována, tedy v areálu budovy FIT VUT v Brně. Na základě počtu dostpných zařízení byla vytvořena topologie o pěti uzlech a base station. Následující tabulka (Tab.2) uvádí předpokládané směrování paketů. Podíváme-li se např. na uzel 6 (Node 6), pak vidíme, jak jsou postupně data směrována přes uzel Node 2, případně přes Node 5 a z něj pak dále směrem k Base station.

Zdroj	Node 2	Node 3	Node 4	Node 5	Node 6
Cíl	Base station	Base station	Node 3	Node 4	Node2 (Node5)

Tab.2: Jednoduchá routovací tabulka pro uzly v navrhované síti.

Níže uvedený obrázek (Obr.15) pak znázorňuje návrh rozmístění jednotlivých uzlů v testovacím prostředí fakulty, přičemž toto rozvržení by mělo odpovídat výše zmíněným pravidlům. Při návrhu byly uváženy informace z dokumentace k uzlům MicaZ týkající se maximálního dosahu radiového signálu. Tyto údaje uvádějí dosah 20m až 30m v uzavřených vnitřních prostorech a 75m až 100m na volném venkovním prostranství [5].

Síť je navržena tak, aby otestovala různé situace, jež mohou během měření nastat. Tím je myšlena především různá úroveň zatížení prvků, což je z předchozí tabulky a následujícího obrázku zřejmé. Vidíme, že přes uzel Node 3 mohou být směrovány pakety z Node 4, Node 5 a Node 6. Naproti tomu přes Node 6 by měli putovat pakety pouze z tohoto uzlu, případně z uzlu Node 5 (při výpadku větve uzlu Node 4).



Obr.15: Situační návrh topologie sítě na půdorysu budovy FIT VUT v Brně.

5 Vývoj aplikací a simulace

Následující kapitola se bude blíže zaměřovat na praktickou (implementační) stránku vyvíjeného projektu. Budou probrány důležité části jednotlivých aplikací s odkazy na úryvky zdrojových kódů uvedené v přílohách této práce. Další podkapitoly jsou logicky členěny do samostatných celků, kde každá tato oblast má dá se říct odlišný způsob implementace. V projektu je použito několika různých programovacích jazyků, mezi které patří především nesC (vývoj aplikací pro senzorové uzly a base station), Java (pro implementaci programu na čtení paketů z USB) a PHP (pro vytvoření webového rozhraní potřebného k vizualizaci naměřených hodnot).

5.1 Instalace a příprava prostředí

Ještě předtím než se pustíme do samotného programování, je zapotřebí mít připraveno prostředí, ve kterém budeme naše aplikace vyvíjet. Jelikož projekt TinyOS ještě není tolik rozšířen a svoji pevnou pozici a uznání si prozatím stále buduje, jsou k dispozici různé manuály a návody, avšak neexistuje žádné unifikované řešení, pomocí kterého nainstalujete vše potřebné vybavení. Instalace vývojového prostředí tedy nebyla zcela bezproblémová a bylo nutné vyzkoušet několik různých způsobů. Jako správné řešení se ukázalo použití návodu na instalaci systému Xubuntu a manuální doinstalaci všech potřebných knihoven a programů. Tento návod se nachází na wiki stránkách dokumentace k TinyOS[8]. V době psaní této práce se jednalo o Xubuntu verze 10.04 a SVN repozitář se zdrojovými soubory TinyOS ve verzi 2.1.1.

5.1.1 Požadavky na vybavení

Co se týče konkrétních požadavků na to, aby bylo možné nainstalovat a používat prostředí pro vývoj aplikací pro TinyOS, neexistuje přesná specifikace minimálních požadavků na systém a hardwarové vybavení počítače. Jednoduše řečeno, je zapotřebí takového počítače, na kterém bude možné zprovoznit uvedený systém Xubuntu, případně jinou Linuxovou distribuci. Aplikace však lze vyvíjet i pod operačním systémem Microsoft Windows.

Pro představu uvádím konfiguraci (Tab.3) notebooku, na kterém byla tato práce a systém monitorování budovy pomocí bezdrátové senzorové sítě vyvíjen.

Procesor:	Intel Core 2 Duo, T6500 2.10GHz
Operační paměť(RAM):	4.00 GB
Operační systém:	Windows 7 Professional (Service pack 1)
Typ systému:	64-bit
Pevný disk(HDD):	300 GB
Grafická karta:	ATI Mobility Radeon HD 4500 Series

Tab.3: Konfigurace a vybavení počítače s instalovaným vývojovým prostředím.

8 Webové stránky s návodem k instalaci vývojového prostředí TinyOS http://docs.tinyos.net/tinywiki/index.php/Xubuntu_8.04,_CVS_T2.1,_Step_by_step.

5.1.2 Postup instalace

Uvedený postup byl aplikován při výše zmíněné instalaci systému Xubuntu 10.04 na notebooku se systémem Windows 7 Professional. Instalace byla provedena v prostředí virtualizačního nástroje Oracle VM VirtualBox 4.1.12.

Krok č.1 – Instalace systému:

Jako první je nutné nainstalovat potřebnou distribuci a verzi systému Linux. V našem případě volně dostupná instalace Xubuntu. Tento systém na rozdíl od ostatních typů Ubuntu používá méně zdrojů pro instalaci. Doporučuje se právě systém Xubuntu10.04(LTS[9]) [10]. Po instalaci systému následuje přihlášení – nejlépe pod účtem správce. Nyní je zapotřebí doinstalovat následující balíky:

```
autobook  autoconf  automake  build-essentials
libc6-dev  libtool  openssh-client  openssh-server
subversion  stow

git-core git-doc gitk gitmagic
```

K tomu použijeme terminál a příkazy `sudo -s` a následně `apt-get install [nazev baliku]`. [10] Případně je možné použít *Správce balíků synaptic.*

Krok č.2 – Instalace nástrojů TinyOS:

Budeme instalovat nástroje *msp430* a *avr* pro uzly telosb resp. micaz, mica2[10]. Před samotnou instalací je zapotřebí upravit soubor */etc/sources.list*, obsahující zdroje, ze kterých se mají stahovat balíky. Tento soubor upravíme přidáním nového řádku s následujícím obsahem :

```
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main
```

Poté provedeme instalaci následujících balíků pomocí příkazů `sudo -s, apt-get update, apt-get install [nazev baliku].`

```
avr-binutils-tinyos  avrdude-tinyos  avr-gcc-tinyos  avr-libc-tinyos
avr-tinyos  avr-tinyos-base  deputy-tinyos  msp430-binutils-tinyos
msp430-gcc-tinyos  msp430-libc-tinyos  msp430-tinyos  msp430-tinyos-base
nesc  tinyos  tinyos-2.1  tinyos-base  tinyos-required  tinyos-tools
```

Krok č.3 – Stažení zdrojových souborů z SVN repozitáře:

Aktuální zdrojové kódy projektu TinyOS jsou dostupné na internetu přes službu Google Code [10]. V době psaní této technické zprávy je aktuální verze TinyOS 2.1.1 a s touto verzí budeme také pracovat. Vytvoříme si složku, do které budeme stahovat zdrojové soubory nebo můžeme použít například složku `/opt` v souborovém systému. Přejdeme do této složky a provedeme tzv. checkout[10] pomocí následujícího příkazu:

```
svn checkout http://tinyos-main.googlecode.com/svn/trunk/ tinyos-2.x
```

9 LTS = Long time support – přeloženo do češtiny jako dlouhotrvající podpora. V praxi to znamené, že jsou pro tento systém dlouhodobě poskytovány pravidelné aktualizace a technická podpora.

10 Checkout – jedná se o prvotní stažení kopie SVN repozitáře z jeho úložiště.

Krok č.4 – Nastavení systémových proměnných:

V tuto chvíli máme k dispozici všechny potřebné základní nástroje pro vývoj aplikací na platformě TinyOS. Nyní je potřeba provést konfiguraci některých systémových proměnných a nastavení cest k adresářům obsahujícím potřebné nástroje.

Jedná se především o nastavení těchto proměnných [10]:

```
TOSROOT, TOSDIR, MAKERULES, CLASSPATH, TOSMAKE_PATH
```

Pro nastavení těchto proměnných čtěte uvedený návod – krok 4 až 5. Nastavení může být různé podle umístění adresářů a podle verzí instalovaných nástrojů. Další kroky instalace popsané ve zmíněném návodu se týkají různých dalších nastavení. Podle potřeby je možné pokračovat v těchto krocích, nám však postačí instalace a konfigurace provedená do tohoto bodu.

5.2 Aplikace pro senzorové uzly

Tato část implementace je bezesporu stěžejním prvkem celého systému pro monitorování budovy pomocí bezdrátové senzorové sítě, protože bez uzlů by samozřejmě nebylo možné žádné měření provádět. V úvodu již bylo řečeno, že bude použita platforma uzlů micaZ. Vývoj aplikací pro jiné uzly (mica2, telosb, iris) se příliš neliší, dokonce uzly micaZ a Iris jsou si velmi podobné, až skoro totožné co se týče hardwarového vybavení i způsobu instalace. Nicméně při testování, kdy byly dostupné obě dvě tyto platformy se ukázalo, že instalací stejného programu na uzly Iris jako na uzly MicaZ nebylo dosaženo požadovaných výsledků. Instalace programu do paměti uzlu Iris proběhla v pořádku a uzel se navenek projevoval správně, nicméně tato dvě zařízení různého typu spolu nedokázala komunikovat. Oba dva typy uzlů jsou přitom hardwarově natolik podobné a co se týče bezdrátových komunikačních modulů zcela totožné.

5.2.1 Vývoj zdrojových kódů, prostředí

O jazyku nesC již bylo řečeno, že je odnoží programovacího jazyka C. Pro vývoj a překlad však nepotřebujeme žádné specializované prostředí. Samotný překlad většinou přímo předchází nahrávání programu do flash paměti senzorového uzlu. Pro psaní zdrojových kódů si tedy naprosto vystačíme i s obyčejným textovým editorem, kterým může být např. program *gedit* (na systému Linux) případně *Notepad* (na systémech Windows).

5.2.2 Obecné vlastnosti zdrojových souborů

Programovací model v TinyOS je takový, že existují dvě separátní části. Je to část implementační obsahující veškerý „výkonný“ kód neboli funkčnost a na druhé straně část definující provázání jednotlivých komponent. Každá komponenta má své rozhraní, pomocí kterého může být zpřístupněna její funkčnost dalším komponentám a programům. Do jisté míry je zde podobnost s objekty v jiných programovacích jazycích jako např. C++ nebo Java. Zásadní rozdíl je v tom, že kromě deklarace funkcí, které implementují, musí komponenty deklarovat také funkce, kterými jsou volány [11].

Každá ze zmíněných částí je realizováná samostatným souborem s příponou `.nc`.

Pro většinu aplikací, které jsou k nalezení v repozitáři zdrojových souborů a ukázkových aplikací TinyOS platí nepsaná konvence, že soubory obsahující popis propojení komponent se označují přidáním `AppC` za název aplikace a soubory zahrnující implementaci písmenem `C` uváděným za názvem aplikace.

Jako příklad může posloužit třeba aplikace Blink v adresáři `tinyos-2.1.1/apps/Blink/`.

Názvy souborů pak tedy vypadají následovně: `BlinkAppC.nc, BlinkC.nc`

Existuje jiná konvence, která používá označení `NazevAplikaceP.nc` pro označení souborů s implementací a `NazevAplikaceC.nc` obsahující popis propojení komponent. V tomto projektu bylo použito druhého způsobu zápisu pro větší přehlednost. Pro lepší zapamatování může sloužit pomůcka P (=program), C (=components).

5.2.3 Struktura aplikace a důležité části

Obsahem této podkapitoly týkající se vývoje aplikací bude pohled na některé důležité úseky programu, jejichž funkce bude popsána a vysvětlena.

V první řadě je potřeba zmínit se o způsobu aplikace protokolu pro směřování paketů v síti. Zde je použito tzv. collection tree protokolu, který cíl odesílaného paketu určuje sám podle dostupnosti ostatních uzlů v dosahu zdrojového zařízení. Je vytvořena stromová struktura cest od koncových uzlů směrem k base station, která je stanovena jako tzv. root (kořen) [18]. V naší síti vystupují senzorové uzly ve třech různých rolích. Tyto role můžeme nazvat jako *producent*, *prostředník* a *konzument*.

Konzumentem v našem případě bude base station. Tím se v tuto chvíli nebudeme zabývat. Zajímat nás ovšem budou zbývající dvě role. Producentem se rozumí každý uzel generující vlastní data, prostředníkem pak uzel, generující vlastní data a přeposílající přijatá data z dalších uzlů. Pro implementaci těchto funkcí je použito dvou rozhraní komponenty `CollectionC`. Jsou to `Intercept` a `Send`, přičemž parametry `Intercept` jsou totožné jako u rozhraní `Receive`. Obsahuje pouze jedinou událost `forward`, která obsahuje implementaci přeposílání paketu [18].

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

Nyní se podívejme na fázi měření dat. Ta se odehrává v pravidelných intervalech a uskuteční se vždy po vypršení jednoho cyklu časovače, nastaveného na daný časový úsek. Čtení jednotlivých veličin ze senzorů se děje postupně a při dokončení jednoho se okamžitě vyvolá další ze tří stanovených měření. Zjednodušený příklad toho jak tento proces probíhá je vidět na následujícím příkladu:

```
//cteni dat o teplote
call Temperature.read();

event void Temperature.readDone(error_t err, uint16_t data){
    temp = data;
    //volame dalsi mereni
    call Humidity.read();
}
```

5.2.4 Překlad a instalace aplikací na uzly

Překlad probíhá standardním způsobem jak je na unixových systémech běžné, a to pomocí programu *make*. Tento program je nástrojem pro automatizaci kompilace zdrojových souborů do výsledného proveditelného programu. Je využíván pro překlad zdrojových souborů mnoha rozličných programovacích jazyků.

Pro použití příkazu *make* musíme mít k dispozici soubor Makefile, v němž jsou obsažena pravidla pro překlad a jsou zde uvedeny potřebné zdrojové soubory. Implicitně se po spuštění příkazu *make* bez jakýchkoliv parametrů vyhledává soubor Makefile v aktuálním adresáři, ze kterého je make spuštěn.

Překlad aplikací pro senzorové uzly tedy probíhá většinou z adresáře obsahujícího zdrojové kódy a právě zmíněný soubor Makefile. Obsah tohoto souboru pak může vypadat následovně:

```
COMPONENT=NodeApplicationC
include $(MAKERULES)
```

Na prvním řádku vidíme specifikaci názvu zdrojového souboru s popisem propojení komponent. Druhý řádek pak obsahuje volání funkce include s parametrem proměnné `MAKERULES`, o které byla zmínka v kapitole 5.1.2. Tato proměnná obsahuje cestu k adresáři, ve kterém se nacházejí další přídavné soubory, v nichž jsou specifikovány další pravidla pro překlad. Konkrétní použití příkazu make pro překlad zdrojových souborů na platformu micaz, by vypadal následovně:

```
make micaz
```

Můžeme také připojovat i další přídavné knihovny. Např. pro práci s vlákny je to knihovna *threads*. Příkazem `make micaz threads` jsou pak přiloženy k lokálnímu Makefile přídavné `micaz.target` a `threads.extra` [12].

Často je s překladem přímo spojena i instalace na danou platformu. Příkaz pak modifikujeme takto:

```
make micaz install, 3 mib520,/dev/ttyUSB0
```

První dvě slova jsou stejná jako v předchozím případě a určují pro jakou platformu se mají zdrojové kódy překládat. Poté může (ale nemusí) následovat přidělení adresy uzlu[11], která pak dále může být použita v aplikaci např. pro adresaci zasílaných paketů. V tomto konkrétním případě je uzlu přidělena adresa 3. Další část definuje programovací rozhraní – v našem případě USB rozhraní MIB520 (Obr.16), přes které jsou uzly programovány. Poslední část tohoto příkazu značí port, na němž je dané programovací rozhraní připojeno.

Při vývoji aplikací používajících podporované senzorové desky je nutné tuto skutečnost zohlednit i v příslušném Makefile k dané aplikaci. Pro použití senzorů na těchto deskách pak stačí pouze v souboru Makefile přidat tento řádek:

```
SENSORBOARD=mts400
```

V našem případě jsme použili senzorovou desku typu MTS400. Obdobně je možné použít i další kompatibilní desky.

11 Adresa uzlu je v tinyOS 16-bitová celočíselná hodnota datového typu `am_addr_t`.

5.3 Aplikace pro base station

Hlavním a v našem případě jediným úkolem base station je shromažďování dat ze všech uzlů sítě a jejich přeposílání na sériové rozhraní USB odkud jsou data dále zpracována v hostitelském počítači. Pro vytvoření base station budeme potřebovat jeden uzel s bezdrátovým komunikačním rozhraním. Není zapotřebí, aby byl tento uzel vybaven senzorovou deskou (pokud si nepřejeme měřit data i na base station). Počítejme s tím, že tento uzel bude mít přidělenou adresu 1.

5.3.1 Struktura aplikace a důležité části

V porovnání s ostatními uzly sítě je aplikace pro base station ve své struktuře o dost jednodušší. Jelikož zde nepotřebujeme měřit veličiny jako u ostatních zařízení, odpadá nám tak nutnost implementace zasílacích a přeposílacích mechanismů, časovače pro periodické opakování měření a dalších součástí, které byly u předchozí aplikace nezbytné.

Jedna věc je však naprosto totožná a společná pro obě tyto aplikace. Je jí datová struktura paketu pro sbírání dat ze sítě. Tato struktura se nachází v hlavičkových souborech obou aplikací a vypadá následovně:

```
typedef nx_struct RadioMsg {
  nx_uint16_t source_id;   //adresa zdrojoveho uzlu
  nx_uint16_t Temp_data;   //data teploty
  nx_uint16_t Hum_data;    //data vlhkosti
  nx_uint8_t Light_data;   //data osvetleni
} RadioMsg;
```

Syntaxe je podobná jako u jazyka C, pouze s tím rozdílem, že místo klíčového slova `struct` uvádíme `nx_struct`. Předpona `nx_` se pak většinou vyskytuje i u datových typů (zde `uint16_t` a `uint8_t`).

Nejdůležitější a nejpodstatnější částí této aplikace je definice události přijímající pakety z okolních uzlů sítě, jejich zpracování a přeposílání na sériové rozhraní USB, ze kterého jsou data dále čtena a zpracována v počítači. Ve zdrojovém souboru nalezneme tuto událost v tomto tvaru:

```
event message_t* Receive.receive(message_t* msg, void* payload,
uint8_t len) { . . . }
```

Co se týká zasílání dat na sériové rozhraní, neliší se jeho implementace příliš od odesílání prostřednictvím bezdrátové komunikace. V implementační části se vlastně neliší vůbec. Rozhraní `SerialForward`, které je zde uvedeno jako příklad pouze využívá komponenty `SerialAMSenderC`, na rozdíl od rozhraní pro zasílání po síti, které využívá komponenty `AMSenderC`.

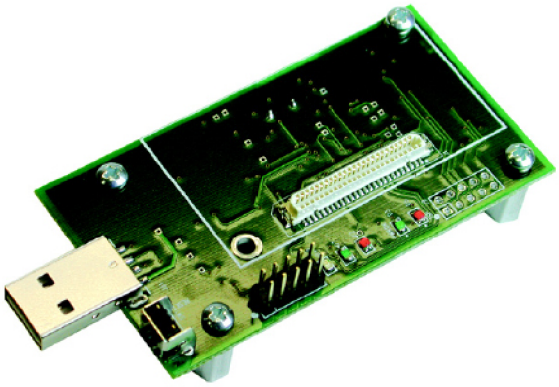
```
call SerialForward.send(AM_BROADCAST_ADDR, &pkt, sizeof(RadioMsg));
```

Předchozí příklad znázorňuje volání události pro zaslání paketu na USB sériové rozhraní. Paket je vysílán všesměrově pomocí definované konstanty `AM_BROADCAST_ADDR`.

5.3.2 Překlad a instalace aplikace na base station

Překlad aplikace určené pro base station a její následná instalace probíhá naprosto stejným způsobem, jako u předchozích aplikací instalovaných na koncové uzly sítě. Je zde rozdíl pouze v adresaci, a to ten, že uzly vystupující jako base station mají ve většině případů přidělenou adresu 1. Jedná se opět o jakousi nepsanou konvenci, která se často používá.

V momentě, kdy máme base station naprogramovanou, ji potřebujeme připojit k počítači, na němž budou data vyhodnocována. Není zapotřebí žádného dalšího vybavení než doposud. Pro čtení dat a komunikaci mezi počítačem a base station je využito stejného rozhraní, jako pro programování. V našem případě tedy použijeme programovací rozhraní MIB520. To zároveň zajišťuje napájení připojeného uzlu (base station) energií z USB rozhraní.



Obr.16: Programovací rozhraní pro uzly MICAz – MIB520.

5.4 Čtení dat ze sériového rozhraní USB

Ve zdrojových souborech TinyOS jsou k dispozici nástroje pro čtení (resp. zápis) z (resp. na) sériové rozhraní. Tyto programy nám většinou mohou posloužit pro testování komunikace mezi base station a hostitelským PC. Pro složitější aplikace je však vhodnější implementace vlastního programu pro zachytávání paketů a jejich další zpracování.

Jedním ze zmíněných programů je SerialForwarder. Tento nástroj psaný v Javě mimo jiné umožňuje uživateli zpřístupnění sériového rozhraní přes TCP/IP. To znamená, že ostatní uživatelé mohou mít k tomuto rozhraní přístup např. prostřednictvím internetu nebo lokální sítě, bez nutnosti přímého fyzického propojení s koncovým zařízením. Navíc také poskytuje připojení více aplikacím současně [18].

Druhou aplikací dostupnou z repozitáře zdrojových kódů TinyOS je program `Listen`, nacházející se v adresáři `support/sdk/java/net/tinyos/tools/`. Pomocí tohoto programu je možné zachytávat pakety na daném sériovém rozhraní a zobrazovat hexadecimálně výpis jejich obsahu. Pro spuštění tohoto programu stačí zadat příkaz:

```
java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB1:micaz
```

První část asi není třeba vysvětlovat, pouze spouští daný Java program. Aplikace je spuštěna s jedním parametrem (`-comm`) za kterým následuje slovo serial (sériové rozhraní), poté oddělovací znak `@` a za ním je uveden konkrétní port, na němž má tato aplikace naslouchat. Posledním údajem je tzv.

baudová rychlost[12]. Ta může být uvedena číselně a nebo pomocí zápisu konkrétního názvu platformy (zde micaz) jako je tomu u tohoto příkladu.

Výstup aplikace vypadá následovně (bližší informace o struktuře paketů na sériovém rozhraní najdete v dokumentaci TinyOS[13]) :

```
00 FF FF 00 00 04 22 06 00 03 00 01
00 FF FF 00 00 04 22 06 00 03 00 02
00 FF FF 00 00 04 22 06 00 03 00 03
```

Podobný výstup můžete získat při spuštění aplikace `BlinkToRadio` na minimálně dvou uzlech a aplikace `BaseStation` na base station připojené k sériovému rozhraní definovaném v parametru programu `Listen`. Obě aplikace naleznete v adresáři `apps` repozitáře TinyOS. Detailní popis struktury tohoto výpisu je k dispozici v dokumentaci [18].

Pro aplikaci vyvíjenou v rámci tohoto projektu bude vzhledem k odlišné struktuře datových paketů vypadat výstup tohoto programu jinak. Především bude obsahovat větší množství informací – měření tří různých veličin ze senzorů – oproti aplikaci `BlinkToRadio`, kde pakety obsahují pouze adresu zdrojového uzlu a čítač, který je inkrementován s každým nově generovaným paketem.

Naše aplikace se z části inspiruje programem `Listen`, ale přidává navíc převod jednotlivých částí paketu na srozumitelné hodnoty a poskytuje konektivitu s MySQL databází, do které jsou přijatá data zaznamenávána. Jednotlivé byty (bajty) jsou převedeny na decimální reprezentaci a poté pomocí vzorců převedeny na výsledné hodnoty daných veličin.

Tyto vzorce byly převzaty z testovacího souboru `Mts400Tester.java` pro senzorovou desku MTS400, který se nachází v adresáři `apps/tests/mts400/`.

Pro připojení a práci s databází MySQL je využito knihovny JDBC. Toto řešení se ukázalo jako velmi jednoduché a použití MySQL zvítězilo nad SQLite, pro širší škálu funkcí a lepší správu dat. Navíc máme k dispozici hostitelský počítač s neomezeným přístupem a nejsme tedy tolik omezeni pamětí a dostupným místem na disku jako u mobilních zařízení. U těch by se naopak použití SQLite jevilo jako přijatelnější.

Abychom mohli použít JDBC potřebujeme přidat do programu externí knihovnu, kterou můžeme volně stáhnout na internetu[14]. Připojení k databázi pak vypadá opravdu jednoduše:

```
//registrace ovladace JDBC
Class.forName("com.mysql.jdbc.Driver");

//parametry pripojeni
String url = "jdbc:mysql://localhost/sensor_data?
user=uzivatel&password=mojeheslo";

this.con = DriverManager.getConnection(url);
```

Další části této aplikace včetně čtení dat ze sériového rozhraní, jejich převodu a zápisu do databáze naleznete v příloze (Příloha A).

12 Baudová rychlost – udává počet změn signálu za sekundu; stanovuje dobu trvání vysílání jednoho bitu.
13 http://docs.tinyos.net/tinywiki/index.php/Mote-PC_serial_communication_and_SerialForwarder
14 http://dev.mysql.com/downloads/connector/j/ - webové stránky pro stažení JDBC ovladače.

5.5 TOSSIM – nástroj pro simulaci WSNs

Nástroj TOSSIM je simulační framework pro testování, vývoj a simulaci aplikací v TinyOS. Hlavní výhodou použití simulátoru (stejně jako u většiny ostatních) je možnost vyzkoušení aplikací ještě před samotným nasazením na skutečnou platformu – v našem případě na uzly MICAz.

Hlavním cílem TOSSIM je poskytnout uživateli věrnou simulaci skutečného chování při nasazení aplikací v reálném prostředí. Přesnosti v simulacích tímto nástrojem je dosaženo především díky detailnímu napodobení chování reálných systémů již od velmi nízké hardwarové úrovně. Mimo jiné je možné simulovat jakákoliv systémová přerušení nebo např. funkčnost AD převodníků [15]. Co však TOSSIM v současné době postrádá je simulované měření stavu baterií (obecně zdoje energie)

Simulace funguje na principu nahrazení chování hardwarových komponent jim příslušejícími simulačními ekvivalenty. Tato abstrakce je velmi flexibilní a poskytuje širokou škálu možností pro různé typy zařízení.

Důležité je také zmínit, že simulace funguje na principu diskrétních událostí. Tyto události jsou řazeny ve frontě podle času a postupně vykonávány dle plánovače (kalendáře) událostí. Pro spouštění a řízení simulací jsou k dispozici dvě možnosti. První z nich je programovací jazyk C++. Druhým a pravěpodobně užívanějším řešením je Python, poskytující možnost flexibilně a dynamicky řídit simulaci. Výhoda Pythonu je v tom, že oproti C++ můžeme přímo zadávat příkazy na příkazovou řádku jeho interpretru, zatímco v C++ je nutné vždy přeložit celý program znovu.

Předchozí část kapitoly 5.5 čerpala informace z [16].

5.5.1 Překlad zdrojových kódů pro TOSSIM

Pro překlad použijeme z předchozích kapitol známý program *make*. Ten opět spouštíme z adresáře, ve kterém se nacházejí zdrojové soubory naší aplikace, jež chceme simulovat. Jak bylo řečeno, TOSSIM nahrazuje veškeré chování hardwarových komponent svými softwarovými ekvivalenty. Pro každou takovou komponentu pak existuje v její složce zdrojových souborů TinyOS podsložka `sim`, v níž se nachází zmíněné kódy určené k simulaci těchto komponent [16].

Simulátor TOSSIM je v TinyOS chápán jako knihovna. Pokud byste tedy hledali jeho zdrojové soubory a různé příklady, naleznete je ve složce `tos/lib/tossim`.

Nyní se podívejme na samotný příkaz, po jehož provedení budou přeloženy zdrojové soubory TOSSIM spolu se všemi potřebnými kódy pro simulaci použitých komponent.

```
make micaz sim
```

Po tomto příkazu vznikne v aktuální složce naší aplikace adresář `sim` případně v novějších verzích adresář `simbuild`, obsahující přeložené zdrojové soubory pro simulaci.

5.5.2 Simulace prostřednictvím Python

Jak bylo uvedeno v úvodu této kapitoly o simulaci, pomocí konzole jazyka Python můžeme interaktivně řídit a sledovat průběh simulace. V takovém případě je zapotřebí nejprve přeložit zdrojové kódy TOSSIM pro danou aplikaci (viz předchozí podkapitola). Poté spustíme interpreter jazyka Python prostým příkazem `python` v terminálu. Nástroje tohoto jazyka byste měli mít nainstalované již od počátku, kdy jsme probírali instalaci vývojového prostředí TinyOS. Pokud se Vám přesto nedaří spustit interpreter, zkontrolujte nastavení TinyOS pomocí příkazu `tos-checkenv`. Ten Vám podá bližší informace o tom jaké programy potřebujete doinstalovat nebo jak změnit nastavení např. u některých systémových proměnných.

Následující obsah čerpá informace převážně z [15,16] :

Po úspěšném spuštění příkazové řádky jazyka Python byste měli vidět následující řádky, lišící se pouze ve verzi instalovaného software :

```
Python 2.3.4 (#1, Nov  4 2004, 14:13:38)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Následující postupy předpokládají základní znalost jazyka Python. Pro pochopení některých zmiňovaných konstrukcí doporučuji nahlédnout do online dokumentace[15].

Nyní již můžeme začít se simulací. Co však ještě budeme potřebovat je import knihovny TOSSIM a vytvoření objektu simulace. Tomu odpovídají následující dva příkazy:

```
>>> from TOSSIM import *
>>> t= Tossim([])
```

V tomto okamžiku můžeme začít simulaci. Jak bylo dříve zmíněno simulace je založena na diskrétních událostech. Právě v této oblasti je obecně známý algoritmus *next event* a příkaz, kterým provedeme další událost v TOSSIM obsahuje právě tato dvě slova.

```
>>> t.runNextEvent()
```

Jistě si všimnete, že pracujeme s vytvořeným objektem `t` jak je zvykem u objektových jazyků a volání metod (funkcí) je realizováno obdobným způsobem jako např. v jazyce Java (tzn. přes tečkovou notaci). V tuto chvíli volání `runNextEvent()` vrací hodnotu 0, protože ještě nebyla naplánována žádná událost, která by mohla být provedena.

Jedním typem událostí může být např. start (boot) zařízení, který se v TOSSIM provede touto posloupností příkazů:

```
>>> n = t.getNode(5)
>>> n.bootAtTime(10000)
```

Zavoláme-li po provedení těchto příkazů funkci `runNextEvent()` znovu, pak simulace přejde na tuto událost, která je prozatím jedinou událostí v plánovači a vrací hodnotu 1. Pro zjišťování stavu zařízení pak slouží základní funkce `isOn()`, `isOff()`. Pro ukončení činnosti běžícího zařízení můžeme použít funkce `turnOff()`.

Na předchozím příkladu jsme vytvořili objekt `n`, který reprezentuje zařízení s adresou 5. Spuštění tohoto zařízení jsme naplánovali na čas 10 000 simulačních kroků. Pokud chceme pracovat s časem ve formátu, který je pro člověka srozumitelnější, máme k dispozici např. funkci `ticksPerSecond()`, jež vrací počet simulačních tiků za jednotku času – sekundu.
Pro více informací doporučuji přečtení manuálových stránek knihovny TOSSIM [15,16].

---

15 http://docs.python.org/ - webové stránky s dokumentací jazyka Python.

5.5.3 Ladění aplikací pomocí příkazu `dbg`

Pro ladění programu nebo jen pro kontrolní výpisy při simulaci poskytuje TOSSIM nástroj umožňující výpis dat na standardní výstup. K tomu slouží příkaz `dbg`. Tento příkaz je možný vkládat do zdrojového kódu aplikace jako většinu ostatních konstrukcí jazyka nesC. Dojde-li vyhodnocení programu k místu výskytu příkazu `dbg`, pak jej interpretuje podobně jako např. v jazyce C funkce `printf()`.

Pro tyto ladící výpisy je možné nastavit různé módy pro specifikaci konkrétních stavů kdy se mají výpisy zobrazovat. Tyto módy se nastavují pomocí příznaků, jejichž bitové hodnoty se vzájemně dají kombinovat a tím lze docílit různých nastavení. Detailní popis všech dostupných módů naleznete v manuálu [15].

Příklad použití dbg:

```
dbg(DBG_BOOT, "Application booted");
```

Takto zapsaný příkaz vypíše na standardní výstup větu „Application booted“ v případě, že je nastaven příznak povolující zobrazení ladících výpisů při startu zařízení (DBG_BOOT).

5.5.4 Simulace síťové topologie

Co se týče konfigurace síťové topologie a testování síťového provozu, simulátor TOSSIM nabízí opět řadu užitečných nástrojů, jejichž pomocí lze simulovat a testovat různé parametry sítě, jako zatížení uzlů, rušení radiového signálu, apod.

Pro simulaci síťové komunikace je nejprve zapotřebí vytvořit model topologie sítě. Obvykle se nastavení parametrů sítě ukládá do souboru, ve kterém specifikujeme vždy na samotném řádku zdrojový uzel, cílový uzel a nakonec následuje síla signálu (dBm[16]) mezi těmito uzly. Příklad takového souboru (specifikujícího topologii o dvou uzlech) ukazuje následující řádek.

```
1   2   -56.0
2   1   -54.0
```

Takto můžeme specifikovat topologii pro všechny uzly v naší testované síti. Tato data pak lze velmi jednoduše zpracovat pomocí funkce `add()` objektu `radio` knihovny TOSSIM, která přímo vytváří ze zadaných dat topologii sítě pro simulaci [16].

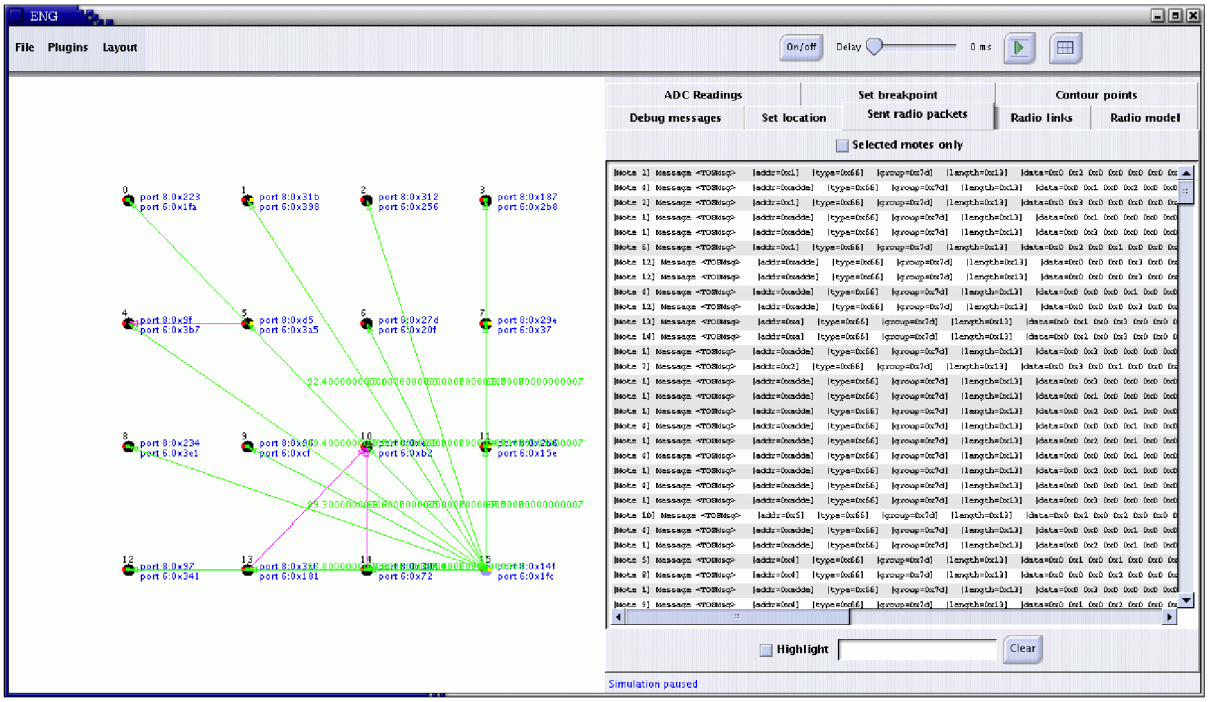
Podmínky pro bezdrátový přenos však nejsou téměř nikdy naprosto ideální, a proto je nutné specifikovat také charakteristiku šumu (rušení). Pro naše účely můžeme použít existující charakteristiky, nacházející se ve složce `tos/lib/tossim/noise/`.

Doposud jsme se bavili pouze o příkazech prováděných skrz příkazovou řádku interpreteru Python. V tomto momentě se však již jedná o složitější konstrukce a přibývá nám čím dál více příkazů a funkcí a bylo by přinejmenším nepřehledné a možná i únavné provádět příkazy jeden po druhém. Nic nám tedy nebrání v tom (a zpravidla se tak děje), abychom si vytvořili přímo celý soubor v jazyce Python, který poté spustíme a provedeme najednou. Kompletní příklad vytvoření takovéto síťové topologie a celkového modelu se všemi parametry uložený v samostatném souboru naleznete v příloze (Příloha B).

16 dBm – v těchto jednotkách se uvádní výkon (síla signálu) v poměru k výkonu 1 mW.

5.5.5 Vizualizace topologie

Kromě výše zmíněných možností je možné použít i vizualizačních nástrojů pro grafické zobrazení simulované sítě. Jedním z nich je grafické rozhraní TinyViz vyvinuté v programovacím jazyce Java. Jedná se o nástroj umožňující ladění a zobrazení stavu sítě za běhu simulace. Poskytuje řadu pluginů, které mohou poskytovat například tzv. debug výpisy, o nichž byla řeč v kapitole 5.5.3. Dalšími funkcemi jsou různé grafické zobrazení datových paketů včetně jejich směřování v síti.
Tento nástroj mimo jiné také umožňuje měnit řadu parametrů během simulace – např. hodnoty výsledků měření nebo polohu a vzdálenosti jednotlivých uzlů [17].



Obr. 17: Grafický vizualizační nástroj TinyViz – ukázka topologie [17].

6 Testování a zpracování dat

Tato kapitola se věnuje jedné z posledních fází vývoje. Jsou zde popsány testovací postupy použité při realizaci této práce. Následující text pak obsahuje i návrh uživatelského rozhraní použitého pro zobrazení a vizualizaci naměřených dat. V závěru kapitoly je popsán konkrétní způsob vizualizace a použité nástroje.

6.1 Průběh testování aplikací

K domácímu testování a vývoji byly z fakulty zapůjčeny tři uzly MICAz, jedno programovací rozhraní MIB520, USB kabel pro připojení k počítači a dvě senzorové desky MTS400CA. Na základě tohoto vybavení tedy bylo možné získat dva plnohodnotné senzorové uzly a jeden uzel jako base station. Od toho se také odvíjel postup při vývoji aplikací a jejich testování.

Prvním bodem při prvotním setkání s těmito zařízeními a po nastudování dostupných materiálů, jimiž byla dokumentace a návody uveřejněné na internetu, bylo zprovoznění základních aplikací umístěných v repozitáři zdrojových souborů TinyOS. Na těchto aplikacích mezi nimiž byli např. `Blink`, `BlinkToRadio`, `BaseStation` nebo `EasyCollection` (všechny umístěné v adresáři `tinyos-2.1.1/apps/`) proběhlo ověření základní funkčnosti důležitých komponent zařízení. Jednou takovouto podstatnou komponentou je bezdrátový komunikační modul, který byl testován právě na aplikaci `BlinkToRadio` (bližší popis naleznete v souboru této aplikace na přiloženém CD/DVD) . Pomocí tohoto programu instalovaného na dvou uzlech lze velmi snadno zjistit zda jsou tyto dvě zařízení v dosahu radiového signálu a dokážou spolu komunikovat. Tímto způsobem byly vyměřeny jednotlivé vzdálenosti mezi sousedními uzly a byla stanovena síťová topologie definovaná v kapitole 4.

Pro testování a ladění bylo často použito LED diod umístěných přímo na těle senzorových uzlů. Nejčastější použití těchto tří diod různých barev bylo právě pro signalizaci stavu zařízení při bezdrátové komunikaci – odeslání paketu, příjem, přeposlání, apod. Další technikou použitou pro ladění aplikací bylo použití tzv.debug výpisů, o nichž byla zmínka v kapitole o simulaci a nástroji TOSSIM. Tento způsob ladění byl použit pouze při testování síťové komunikace a v pozdějších etapách vývoje již nebylo zapotřebí jeho použití.

Ve finální fázi testování jsme se přesunuli do prostor fakulty, kde měla být tato práce otestována a odzkoušena. Ukázalo se, že původně zamýšlené rozmístění uzlů, které bralo v úvahu i oficiální údaje o dosahu signálu, nebylo možné realizovat. Největším problémem byl právě špatný dosah bezdrátového komunikačního zařízení. Tato skutečnost je s nevětší pravděpodobností způsobena tím, že na fakultě existuje mnoho rušivých elementů v podobě signálu z Wi-fi přístupových bodů, kterých je zde opravdu hodně. To však není jediný probém. Dalším uskálím co se signálu týče jsou robustní železobetonové konstrukce a široké zdivo, kterých je v budově nespočet.

Po stanovení konečné síťové topologie spolu s dalšími kolegy, jsme naše aplikace a další programové a hardwarové vybavení přesunuli do kanceláře vedoucího práce. Odtud probíhala analýza sbíraných dat a zde byly umístěny base station připojené k hostitelskému PC. Vyvíjená bezdrátová senzorová síť byla testována po dobu přibližně dvou dní. Výsledky měření z jednotlivých uzlů byly ukládány do připravené databáze.

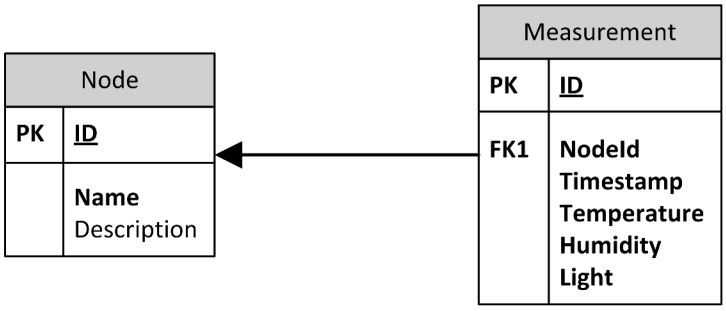
Při analýze tohoto testu bylo zjištěno, že i přes počáteční potíže s dosahem signálu se nakonec všem uzlům podařilo navázat spojení a k dispozici tak byla data ze všech těchto zařízení. Nicméně výsledky nebyly úplně stoprocentní a čas od času se vyskytla místa, kdy v zadaném intervalu data z určitého uzlu nebyla správně doručena. Tato skutečnost si vyžádala, aby byl původní protokol, v němž byly cesty zadány staticky pro každý uzel přepracován. Namísto prvně použitého adresovaného

zasílání paketů byl zvolen CTP[17] protokol, který je tzv. adress-free. To znamená, že není potřeba zadávat konkrétní cíl paketu ručně pomocí adresy.

Poslední fází, avšak pro uživatele tou nejpodstatnější z hlediska zdroje informací je zpracování dat na koncovém zařízení. V našem případě se jedná o osobní počítač se systémem Windows, na němž je prostřednictvím programu Oracle VM VirtualBox spuštěn systém Xubuntu 10.04 LTS. Data na tomto počítači budou zpracována a uložena pro pozdější analýzu a vizualizaci.

6.2 Návrh webového rozhraní

O čtení dat ze sériového rozhraní USB již byla řeč dříve. Nyní přichází na řadu zpracování těchto dat, jejich uložení a následně zobrazení. Ukládání dat do databáze bylo také již několikrát zmíněno a konkrétní ukázku naleznete v příloze (Příloha A), pro nás je však v tento moment důležité v jakém formátu jsou data do databáze vkládána.



Obr.18: ER diagram databázových tabulek.

Existují zde dvě tabulky, z nichž jedna obsahuje záznamy o senzorových uzlech. Do této tabulky vložíme informace o všech zařízeních v naší senzorové síti, o kterých budeme chtít později zobrazit záznamy jimi pořízené. Sloupce tabulky zahrnují informaci o adrese uzlu a krátký popis, který může být použit např. k identifikaci zařízení podle jeho umístění, viz příklad níže (Tab.4).

ID	Name	Description
1	1	Chodba budovy A
2	2	Přednáškovámístnost

Tab.4: Příklad záznamu v databázové tabulce obsahující informace o uzlech.

Druhá z tabulek pak slouží pro shromažďování výsledků měření. Její struktura také není složitá. Obsahuje index na sloupci ID ve formě primárního klíče, poté sloupec NodeId, který je cizím klíčem a odkazuje se do tabulky zařízení zmíněné v předchozím příkladu. A nakonec pro měřené veličiny zde máme sloupce Temperature, Humidity a Light.

Do této tabulky jsou data vkládána již v člověku srozumitelné podobě. Převod na jednotky v nichž jsou data prezentována se děje ještě před zápisem do databáze a to konkrétně v Java aplikaci naslouchající a zpracovávající data na sériovém rozhraní. Příklad takovéto databázové tabulky opět můžeme demonstrovat na následujícím příkladu (Tab.5). Obě tabulky jsou znázorněny na obrázku (Obr.18).

17 CTP = Collection Tree Protokol – jedná se o protokol směřující data k base station pomocí stromové struktury skládající se z obecně z několika větví.

ID	NodeId	Timestamp	Temperature	Humidity	Light
1	1	2012-04-05 14:35:03	24.03	54.35	315.20
2	2	2012-04-05 14:35:11	23.89	58.76	316.00

Tab.5: Příklad záznamu v databázové tabulce obsahující data z měření.

Vzhledem k tomu, že máme k dispozici počítač, na kterém je spuštěna MySQL databáze a v ní jsou naše uložená data, jeví se jako vhodné řešení realizovat vizualizaci a analýzu pomocí webového rozhraní. Výhodou je jednoduchost implementace a navíc možnost zpřístupnění této webové aplikace přes internet, což může být v řadě případů žádoucí. Pro implementaci tohoto rozhraní byl zvolen jazyk PHP. Zajisté existuje mnoho jiných, možná i lepších řešení, nicméně tato varianta byla zvolena pro zkušenosti s tímto jazykem. Nebudu zde popisovat do detailů vývoj webového rozhraní od grafického návrhu, stylování a podobně, protože to není pro tuto práci až tak podstatné. Co je naopak důležité zmínit je způsob vizualizace. Pro tento účel bylo použito knihovny JPGraph[18]. Tento nástroj je open-source a pro nekomerční využití a studijní potřeby je šířen pod licencí QPL 1.0 (Qt Free)[19]. Pro případ komerčního užití existuje k této volné verzi profesionální alternativa s dalšími rozšířeními. Více informací naleznete na stránkách projektu.

6.3 Vizualizace výsledků měření

Webové rozhraní určené k prezentaci naměřených dat zahrnuje grafickou vizualizaci pomocí grafů a textovou s využitím tabulek. Webová aplikace poskytuje analýzu dat v několika kategoriích. Jsou to následující:

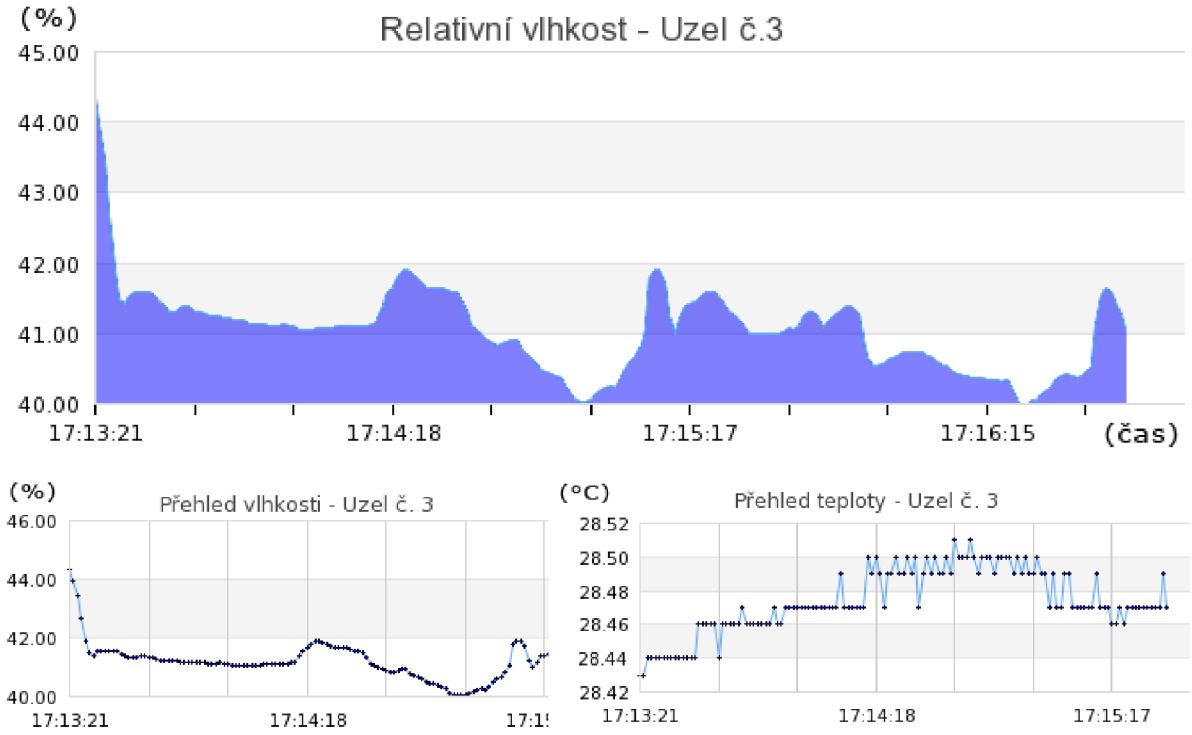
- celkový přehled (přehledně zobrazuje aktuální informace ze všech uzlů)
- detail zařízení (zaměřuje se na vybraný uzel a zobrazuje data ve vybraném období)
- detail veličiny (zaměřuje se na vybranou veličinu a zobrazuje informace ze všech uzlů)
- statistiky (poskytuje základní statistické údaje ze všech uzlů v daných obdobích)

Obr.19: Celkový pohled na uživatelské webové rozhraní.

18 http://jpgraph.net – domovská stránka projektu JPGraph pro kreslení grafů v PHP.
19 http://www.opensource.org/licenses/qtpl.php – definice licence QPL.

Při zpracování dat pro zobrazení formou tabulek dochází k prostému výběru pomocí MySQL příkazu SELECT. Poté se pro každý řádek databáze splňující zadaná kritéria generuje v PHP příslušný HTML kód vytvářející výslednou tabulku.

O něco zajímavější je vytváření grafů, které jsou do stránek vkládány jako obrázky. V HTML to běžně znamená použití tagu `<img>`. Ten však jako parametr `src` nebude mít zadanou cestu k obrázku, ale cestu k PHP skriptu, který bude náš graf generovat. Tento skript obsahuje řadu importovaných knihoven z použitého nástroje JPGraph a také množství nastavení, týkající se grafického vzhledu, typu, popisů a dalších parametrů požadovaného grafu. Nejdůležitějším prvkem však je načtení zdrojových dat. Obecně můžeme data načítat odkudkoliv, ze souboru, uložená v proměnných, atd. My je budeme načítat z MySQL databáze v podstatě stejným způsobem jako u předchozích tabulkových výpisů. V každém případě, ať už data načítáme odkudkoliv, jejich zpracování je stejné. Jsou vytvořeny dvě proměnné typu pole, do kterých jsou naše záznamy uloženy. S těmito proměnnými pak lze různě manipulovat a nastavovat parametry zobrazení pomocí zmíněných nastavení, jejichž výčet nalezneme v dokumentaci na oficiálních stránkách, či v materiálech na přiloženém CD/DVD.



Obr.20: Ukázka grafické vizualizace dat pomocí grafu knihovny JPGraph.

Další ukázky grafické vizualizace dat jsou k nahlédnutí v příloze (Příloha C).

6.4 Zhodnocení výsledků

Na základě testování v budově FIT VUT v Brně se ukázalo, že při použití statické topologie s protokolem, v němž mají všechny uzly v síti pevně danou cestu kudy směřovat data, není tím nejvhodnějším řešením. Přestože návrh takového vlastního protokolu byl původně zamýšlen a testován, při analýze testovacích dat bylo dosaženo závěru, že tento způsob sbírání dat je příliš nespolehlivý a pro použití v praxi tedy neužitečný. Hlavním problébem tohoto prvně navrženého protokolu byla neschopnost zotavení se z výpadku. Dalším bodem, ve kterém tento protokol nesplňoval zadané požadavky byla ztrátovost dat.

Obě výše zmíněné slabiny jsou vyřešeny částečným využitím již zavedeného a používaného protokolu CTP. *Collection tree protokol*, který stojí za touto zkratkou, poskytuje řešení problému výpadku některého z uzlů přes něhož jsou směrována data z jiných zařízení. Způsob jakým se s touto potíží vyrovnává spočívá ve vytvoření stromu na základě nejlepších cest, jejichž výpočet je realizován pravidelným zasíláním tzv. beacon paketů. Tyto jsou všesměrově rozesílány na okolní uzly a podle zpětně získaných informací jsou určeny cesty ve stromu, kde kořenovým uzlem je base station sbírající data z celé sítě [18].

Testováním přepracované verze aplikace s použitím CTP protokolu bylo dosaženo lepších výsledků jak ve zmiňovaných výpadcích sítě, tak i v otázce ztrátovosti paketů. Při simulaci výpadku jednoho z uzlů, který plní roli prostředníka (forwarder), bylo zjištěno, že přehodnocení a znovusestavení stromové struktury se provede během krátké chvíle – řádově sekundy až jednotky minut. Mezitím však mohou probíhat další měření v zařízeních, která svá data směřují právě přes uzel vyřazený z provozu. Do doby než se síť zotaví z tohoto výpadku je k dispozici fronta, do níž se automaticky řadí pakety, které nemohly být doručeny právě z důvodu nedostupnosti cílového uzlu. Tento mechanismus je přímo zabudován v implementci CTP protokolu a není tak zapotřebí realizace vlastní fronty [18].

Co se týče celkového shrnutí práce, je důležité se pozastavit nad přínosem ale i jejími nedostatky a kriticky tak zhodnotit dosažené cíle. V zadání se podařilo dostát všem jeho bodům a všechny stanovené cíle byly realizovány. Největší problémy byly v počátcích, kdy bylo zapotřebí zprovoznit vývojové prostředí a všechny potřebné knihovny, nástroje a programy pro provoz a vývoj aplikací v TinyOS. Všechny vyvinuté aplikace zajisté mají i své nedostatky. Tím hlavním, na který je možné poukázat je odolnost vůči okolním rušivým vlivům v podobě jiných senzorových sítí, čí jednotlivých zařízení pracujících na stejném nebo podobném principu. V této oblasti by se tedy dalo v budoucnu zapracovat na vývoji algoritmů zajišťujících lepší integritu dat a odolnost vůči rušení v rámci jedné senzorové sítě.

Druhý nedostatek, o kterém by bylo vhodné se zmínit, se týká části zpracování dat na sériovém rozhraní. Tuto fázi zpracování dat obstarává aplikace psaná v jazyce Java. Pomineme-li téma efektivity zdrojového kódu, na které by bylo také možné vést debatu, pak je hlavní nevýhodou této aplikace její specifičnost. Jednoduše řečeno je tento program šitý na míru pro nasazení v naší konkrétní síti. Pro tuto práci se samozřejmě nejedná o závažný problém, avšak pro širší použití by bylo vhodné přepracovat tuto aplikaci tak, aby bylo její využití možné v širším rozsahu a při různých parametrech senzorové sítě.

Jako pozitivní stránku je třeba zhodnotit způsob vizualizace a jednoduché ovládání uživatelského rozhraní realizovaného pomocí webových technologií (především PHP, HTML, MySQL). Díky tomu je možné výsledky monitorování pomocí senzorové sítě zpřístupnit on-line na internetových / intranetových stránkách, např. jako součást firemního systému či jiného informačního systému.

7 Závěr

Zadané body práce se v tomto projektu podařilo úspěšně splnit a realizovat tak systém monitorující stav budovy pomocí bezdrátové senzorové sítě. V této oblasti může nacházet různá uplatnění, jako např. monitorování podmínek životního prostředí, detekce požárů a sledování jejich šíření.

V současné době jsou tyto technologie ještě stále ve vývoji a troufnu si říct, že u nás ještě zdaleka nejsou zcela probádané. Výzkum těchto systémů má tedy zajisté svoji budoucnost, v níž se může zaměřovat na zlepšování a hledání nových lepších možností bezdrátové komunikace, zmenšování energetických nákladů, objevy nových mobilních technologií, apod.

Přínosem pro autora bylo v této práci poznání zcela nového odvětví informačních technologií, které nabízí řadu zajímavých otázek pro další studium a budoucí vývoj. Projekt, nebo chcete-li systém, vyvinutý v rámci této práce by mohl mít své další uplatnění při realizaci přídavných modulů pro komunikaci s uzly a jejich řízení. V současné době se jedná pouze o statický systém shromažďující data podle programu zadaného při instalaci na dané zařízení. Přidáním nových možností a funkcí by vzrostla operabilita a uplatnění tohoto systému v reálném nasazení. Další vývoj by pak mohl být směřován také na uživatelské rozhraní, které je jakožto komunikační brána mezi daty a uživatelem nedílnou součástí kvalitního systému jako celku.

Literatura

[1] HOLEŠINSKÝ, Pavel, KOMOSNÝ, Dan, ŠIMEK, Milan. *Lokalizační techniky bezdrátových senzorových sítí založené na triangulačním mechanismu*. Elektrorevue. [cit. 2012-01-12].

[2] RAGHAVENDRA, C.S, SIVALINGAM, Krishna M., ZNATI, Taieb. *Wireless sensor network*. 2006 [cit. 2012-01-24].

[3] *TinyOS Documentation Wiki.* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/Main_Page>.

[4] MÁŠA, Pavel. *Architektura operačního systému TinyOS*. Brno, 2009. Bakalářská práce. Masarykova Univerzita. [cit. 2012-01-17].

[5] MEMSIC Inc. *MICAz. Wireless measurement system* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz>.

[6] *Tymo:DYMO on TinyOS - TinyOS Documentation Wiki* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/Tymo>.

[7] *Network protocols - TinyOS Documentation Wiki* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/Network_Protocols>.

[8] *Mote-mote radio communication - TinyOS Documentation Wiki* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/Mote-mote_radio_communication>.

[9] LEVIS, Philip. *TEP 111 - Message_t* [online]. [cit. 2012-01-24]. Dostupný z WWW: <http://www.tinyos.net/tinyos-2.x/doc/html/tep111.html>.

[10] *Xubuntu 10.04,SVN T2.1,Step by step - TinyOS Documentation Wiki* [online]. [cit. 2012-04-14]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/Xubuntu_8.04,_CVS_T2.1,_Step_by_step>.

[11] LEVIS, Philip. *TinyOS Programming* [online]. 27.10.2006 [cit. 2012-04-20]. Dostupný z WWW: <http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf>.

[12] HORÁČEK, Jan. *Platforma pro mobilní agenty v bezdrátových senzorových sítích.* Diplomová práce, Brno, FIT VUT v Brně, 2009. [cit. 2012-04-29].

[13] CROSSBOW TECHNOLOGY Inc. *MTS/MDA Sensor and Data Acquisition Boards User's Manual*. 2004 [cit. 2012-04-27]

[14] SENSIRION The Sensor Company. *Humidity sensor SHT11.* 2012. [cit. 2012-04-27]. Dostupný z WWW: <http://www.sensirion.com/en/01_humidity_sensors/02_humidity_sensor_sht11.htm>

[15] LEVIS, Philip, LEE, Nelson: *TOSSIM: A Simulator for TinyOS Networks.* User's manual, in TinyOS documentation. [cit. 2012-04-28] Dostupný z WWW: <http://www.cs.berkeley.edu/~pal/pubs/nido.pdf>.

[16] *TOSSIM – TinyOS Documentation Wiki* [online] [cit. 2012-04-28]. Dostupný z WWW: <http://docs.tinyos.net/tinywiki/index.php/TOSSIM>.

[17] *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications.* In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).

[18] FONSECA, Rodrigo, GNAWALI, Omprakash, JAMIESON, Kyle, LEVIS, Philip. *TEP 119 – Collection* [online]. 2006 [cit. 2012-05-07]. Dostupný z WWW: <http://www.tinyos.net/tinyos-2.x/doc/html/tep119.html>.

Příloha A: Úryvky kódu Java aplikace

Ukázka zpracování dat ze sériového rozhraní

```
. . .

 try {
   reader.open(PrintStreamMessenger.err);
   for (;;) {
     byte[] packet = reader.readPacket();

     ByteArrayOutputStream b = new ByteArrayOutputStream();
     PrintStream prt = new PrintStream(b);
     Dump.printPacket(prt, packet);
     content = b.toString("utf-8");

     sub0 = content.substring(18, 23);
     sub1 = content.substring(24, 29);
     sub2 = content.substring(30, 35);
     sub3 = content.substring(36, 41);
     sub4 = content.substring(42, 44);


     am_id = convert.hexStringToInt(sub0);
     node = convert.hexStringToInt(sub1);
     temp = convert.getTemperature(sub2);
     hum  = convert.getHumidity(sub3,temp);
     light= convert.getLight(sub4);


     //vytvorime pozadavek na zapis dat do databaze MySQL
     mysql = new MyTest();
     mysql.saveMeasurement(node, temp, hum, light);




   }
 }

 . . .
```

Ukázka výpočtu relativní vlhkosti na základě použití vzorce z testovacího souboru pro senzorovou desku MTS400.

```
public double getHumidity(String hum_hex, double temperature) {

  if( (value = hexStringToInt(hum_hex)) == 9999) humidity = 9999;
  else
     humidity = (-2.0468 + 0.0367*(double)value - 0.0000015955 *
     Math.pow((double)value,(double )2))+(temperature - 25) *
     (0.01+0.00008*(double)value);

  return myRound(humidity,2);
}
```

Ukázka vytvoření příkazu pro zápis do MySQL databáze

```
public void saveMeasurement(int node, double temp, double hum,
double light) {

    try {

      preparedStatement = con.prepareStatement("INSERT INTO
      Measurement values (default, ?, ?, ?, ?, ?)");

      preparedStatement.setInt(1, node);
      preparedStatement.setString(2, getDateTime());
      preparedStatement.setDouble(3, temp);
      preparedStatement.setDouble(4, hum);
      preparedStatement.setDouble(5, light);

      //provedeme zapis
      preparedStatement.executeUpdate();

    } catch (SQLException e) {
          System.err.println("Cannot create prepared statement!");
          e.printStackTrace();
    }
```

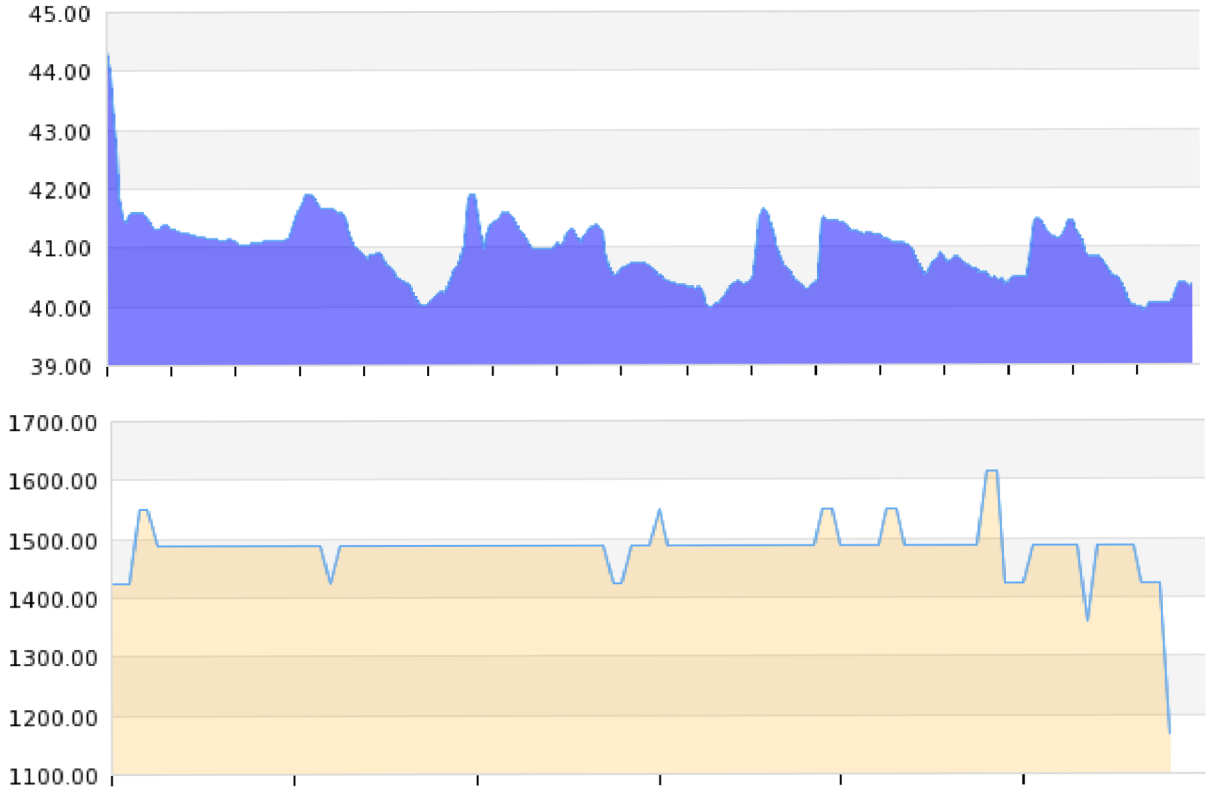
Příloha B: Konfigurace topologie v TOSSIM

Tato ukázka pochází ze zdroje [16].

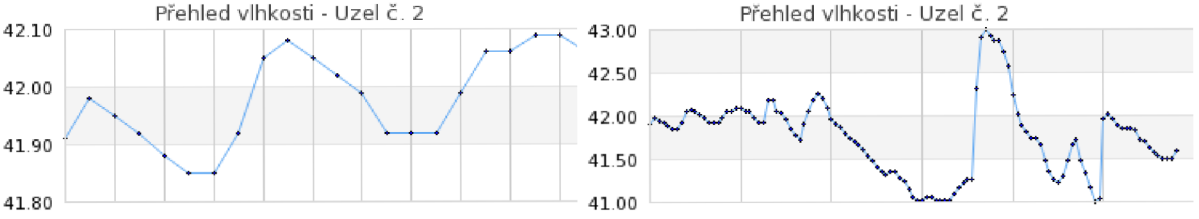
```
#! /usr/bin/python
import sys
from TOSSIM import *
from RadioCountMsg import *

t = Tossim([])
m = t.mac()
r = t.radio()

t.addChannel("RadioCountToLedsC", sys.stdout)
t.addChannel("LedsC", sys.stdout)

for i in range(0, 2):
  m = t.getNode(i)
  m.bootAtTime((31 + t.ticksPerSecond() / 10) * i + 1)

f = open("topo.txt", "r")
for line in f:
  s = line.split()
  if s:
    if s[0] == "gain":
      r.add(int(s[1]), int(s[2]), float(s[3]))

noise = open("meyer-heavy.txt", "r")
for line in noise:
  s = line.strip()
  if s:
    val = int(s)
    for i in range(4):
      t.getNode(i).addNoiseTraceReading(val)

for i in range(4):
  t.getNode(i).createNoiseModel()

for i in range(60):
  t.runNextEvent()

msg = RadioCountMsg()
msg.set_counter(7)
pkt = t.newPacket()
pkt.setData(msg.data)
pkt.setType(msg.get_amType())
pkt.setDestination(0)

print "Delivering " + str(msg) + " to 0 at " + str(t.time() + 3);
pkt.deliver(0, t.time() + 3)


for i in range(20):
  t.runNextEvent()
```

Příloha C: Ukázky vizualizace dat

Ukázka různých typů grafického zobrazení



Obr.21: Ukázka detailních grafů veličin – relativní vlhkost (%), intenzita osvětlení (lux).



Obr.22: Ukázka grafů z celkového přehledu senzorové sítě za poslední hodinu měření.

Ukázka tabulkového výpisu statistik

Maxima naměřených hodnot - teplota vzduchu

Uzel č.	Datum/čas	Teplota (°C)	Vlhkost (%)	Osvětlení (lux)
2	2012-05-10 17:16:56	28.85	41.42	1615.00
3	2012-05-10 17:16:40	28.54	41.59	1359.00

Obr.23: Ukázka tabulkového zobrazení dat (zde statistiky maximálních hodnot).