VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

BEZDRÁTOVÉ SENZOROVÉ SÍTĚ S VYUŽITÍM MOBILNÍCH ZAŘÍZENÍ

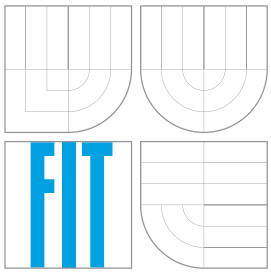
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN DORAZIL

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

BEZDRÁTOVÉ SENZOROVÉ SÍTĚ S VYUŽITÍM MOBILNÍCH ZAŘÍZENÍ

WIRELESS SENSOR NETWORKS BASED ON MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN DORAZIL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN SAMEK, Ph.D.

BRNO 2013

Abstrakt

Diplomová práce se zabývá využitím mobilních zařízení jako senzorových uzlů bezdrátové senzorové sítě. Jsou prozkoumány možnosti bezdrátové komunikace a typy senzorů na mobilních zařízeních se zaměřením na platformu Android. V rámci práce je navržena a implementována bezdrátová senzorová síť s využitím mobilních zařízení s operačním systémem Android. Ta v reálném čase monitoruje veličiny všech dostupných senzorů a přehledně je vizualizuje na základnové stanici. Volitelně mohou být snímána také data z GPS. Základnovou stanici tvoří desktopová aplikace na platformě Java Standard Edition. Komunikace v síti může probíhat pomocí Wi-Fi nebo Bluetooth, případně přes mobilní internet. Uzly lze vzdáleně konfigurovat SMS zprávami.

Abstract

This master's thesis deals with the idea of using mobile devices as sensor nodes in wireless sensor network. Focused mostly on Android platform, we explore possibilities of wireless communication, and describe various types of sensors on mobile devices. We design and implement wireless sensor network based on mobile devices running Android operating system. The network performs real-time capturing of data from all sensors available and optionally from GPS. All measurements are visualized at the base station, which is Java Standard Edition desktop application. Wi-Fi, Bluetooth or even cellular internet can be used for communication within the network. Nodes can be remotely configured via SMS messages.

Klíčová slova

Bezdrátové senzorové sítě, WSN, mobilní zařízení, Android, Java, mobilní aplikace, senzory, Wi-Fi, Bluetooth, GPS, SMS

Keywords

Wireless sensor networks, WSN, mobile devices, Android, Java, mobile application, sensors, Wi-Fi, Bluetooth, GPS, SMS

Citace

Jan Dorazil: Bezdrátové senzorové sítě s využitím mobilních zařízení, diplomová práce, Brno, FIT VUT v Brně, 2013

Bezdrátové senzorové sítě s využitím mobilních zařízení

Prohlášení

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

.....
Jan Dorazil
16. května 2013

Poděkování

Rád bych poděkoval panu Ing. Janu Samkovi, Ph.D, vedoucímu diplomové práce, za odbornou pomoc, ochotu a čas, který mi při tvorbě práce věnoval.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Ačkoliv to nemusí být na první pohled patrné, bezdrátové senzorové sítě jsou významnou součástí našich životů. Uplatnění nacházejí nejen ve zdravotnictví, armádě či ochraně životního prostředí, ale i ve firmách a domácnostech. Cílem této diplomové práce je prozkoumat možnosti spojení oblasti bezdrátových senzorových sítí se stále více se rozrůstajícím segmentem mobilních zařízení, konkrétně chytrých telefonů a tabletů. Mobilní zařízení s operačním systémem (např. Android) jsou dnes masově rozšířená a velmi snadno dostupná. Umožňují bezdrátovou komunikaci přes Wi-Fi, Bluetooth nebo mobilní sítě, a obsahují čím dál více typů senzorů. Stále více lidí tak dnes ve své kapse nosí akcelerometr, gyroskop, magnetometr, senzory snímající intenzitu osvětlení, teplotu, tlak, přiblížení, ...

V rámci práce je navržena a implementována bezdrátová senzorová síť s využitím mobilních zařízení jako uzlů. Síť v reálném čase monitoruje veličiny z dostupných senzorů a odesílá je na základnovou stanici, která je přehledně vizualizuje. Jelikož je práce zaměřená na průzkum možností, je kladen důraz na to, aby byla zachována aplikační nezávislost vytvořené sítě. S jejím využitím by mohla mobilní zařízení představovat, především díky nízké ceně a velkému rozšíření, levnější alternativu některých specializovaných senzorových uzlů.

Kapitola 2 obsahuje seznámení s bezdrátovými senzorovými sítěmi jako takovými. Popisuje, co jsou to bezdrátové senzorové sítě, jaké mají charakteristické vlastnosti a jakým způsobem je můžeme klasifikovat.

V kapitole 3 se zaměříme na prozkoumání možností využití mobilních zařízení jako bezdrátových uzlů. Popíšeme dostupné technologie bezdrátového spojení a možnosti, které nám z pohledu bezdrátových senzorových sítí nabízejí. Dále zdůvodníme výběr platformy Android pro implementaci uzlů a popíšeme možnosti a techniky programování bezdrátové komunikace na této platformě.

V kapitole 4 jsou popsány různé typy senzorů, kterými bývají vybavena mobilní zařízení. Je uvedeno, jaké veličiny měří a k čemu je lze typicky využít. Nakonec jsou uvedeny senzory podporované na platformě Android a způsob práce s nimi.

Kapitola 5 se zabývá návrhem vytvářené bezdrátové senzorové sítě. Je navržena architektura sítě, způsob adresování uzlů a přeposílání zpráv, komunikační protokol a struktura klíčových částí aplikací, které budou implementovat funkčnost uzlu a základnové stanice.

V kapitole 6 popíšeme implementaci vytvořené sítě. Zaměříme se zvlášť na aplikaci uzlu a základnové stanice. Nakonec provedeme testování sítě a jeho vyhodnocení.

Kapitola 2

Bezdrátové senzorové sítě

Jedním z hlavních cílů diplomové práce je prostudovat a popsat možnosti využití mobilních zařízení jako uzlů bezdrátové senzorové sítě. Abychom tak mohli učinit, je nutné seznámit se s těmito sítěmi jako takovými.

Kapitola popisuje, co jsou to bezdrátové senzorové sítě (podkapitola 2.1) a jaké mají charakteristické vlastnosti (podkapitola 2.2), jež je odlišují od klasických počítačových sítí typu LAN či WLAN.[1] Dále, jelikož je spektrum využití bezdrátových senzorových sítí velmi široké, je uvedena klasifikace různých typů těchto sítí (podkapitola 2.3) spolu s příklady použití a nastíněním odlišných problému, které je nutné při jejich návrhu řešit. Informace uvedené v této kapitole byly převážně čerpány z knihy [35], ale také z [28, 31, 32, 42, 43].

2.1 Co jsou to bezdrátové senzorové sítě

Bezdrátové senzorové sítě (Wireless Sensor Networks, WSN) mají široké pole použití od zdravotnictví po armádu, domácí i podnikové. Skládají se obvykle z velkého množství bezdrátově komunikujících *uzlů*, jež jsou rozmístěny tak, aby společně monitorovaly a šířily informace o požadovaných jevech v okolí. Uzly WSN spolupracují ve snaze dosáhnout společného cíle.

V každé WSN se typicky nachází alespoň jedna *základnová stanice* (base-station, sink). Jejím hlavním úkolem je sběr dat od uzlů sítě. Jinými slovy, základnové stanice jsou rozhraním, přes které může WSN komunikovat s okolním světem.

Jak již bylo zmíněno, rozsah použití WSN je veliký. Bohaté využití nacházejí nejen ve vojenském, civilním a podnikovém sektoru, ale nemalou roli hrají také v ochraně životního prostředí. Příkladem mohou být senzory pro vojenský dohled, senzory sledující a ovládající výrobní procesy v továrnách, bio-senzory ve zdravotnictví, senzorové sítě monitorující počasí nebo biotopy a senzorové sítě implementující „chytrou domácnost“. Za povšimnutí také stojí, že WSN dokážou pracovat na místech, kde je přítomnost člověka velmi obtížná, ne-li nemožná. Například senzory pro detekci zemětřesení, vln tsunami, záplav nebo měření extrémně nízkých teplot na pólech zeměkoule.

Snaha o návrh WSN a jejich uzlů tak, aby výsledek podporoval všechny možnosti aplikace, je velmi složitý a náročný proces. Výzkum probíhá na několika úrovních:

- *Úroveň komponent* — zaměřena na vylepšování uzlů jako bezdrátových senzorových zařízení.

[1] Local Area Network, resp. Wireless Local Area Network.

- *Systémová úroveň* — zaměřena na škálovatelnost a energetickou účinnost síťového spojení a komunikace uzlů.
- *Aplikační úroveň* — zaměřena na zpracování dat produkovaných senzory (např. lokalizace cíle podle dat naměřených několika senzory).

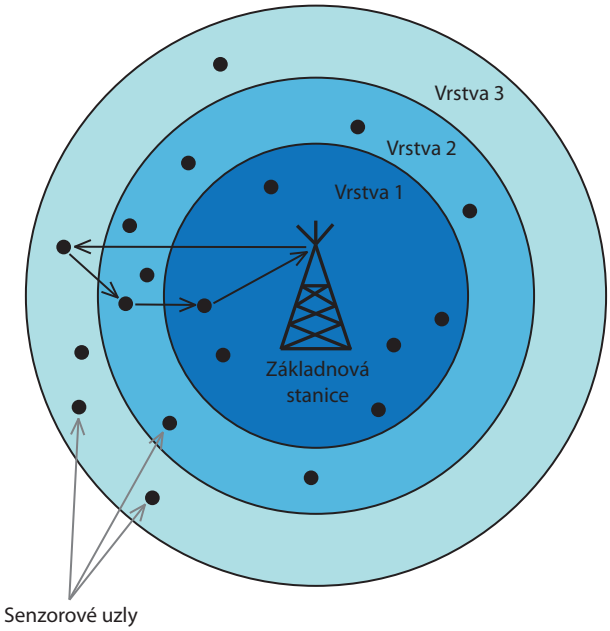
Obsah této kapitoly spadá především do systémové úrovně.

Uzly sítě

Uzel je bezdrátové senzorové zařízení (Wireless Sensor Device). Typicky se jedná o akumulátorem napájené zařízení, které, kromě schopnosti snímat fyzikální veličiny, disponuje možností bezdrátové komunikace, ukládáním menšího množství dat a omezeně také výpočetním výkonem a zpracováním signálů. Akumulátor uzlu často nelze dobíjet.

Pokroky v oboru integrovaných obvodů přinášejí postupné snižování velikosti, hmotnosti a ceny senzorových uzlů a současně zvyšování jejich rozlišení a přesnosti měření. Z tohoto důvodu je mnohdy výhodnější použít velké množství relativně jednoduchých uzlů, než pouze několik málo drahých a důmyslných. Tímto způsobem lze za nižší cenu dosáhnout lepšího pokrytí, přesnosti a spolehlivosti.

Rozmístění uzlů se liší dle konkrétní aplikace a často bývá náhodné. Autoři v [28] pak rozdělují uzly podle jejich polohy na statické a pohyblivé. S rozmístěním uzlů souvisí také volba architektury WSN. Rozlišujeme dvě základní architektury – vrstvenou a seskupnou.



Obrázek 2.1: Vrstvená architektura.

Vrstvená architektura

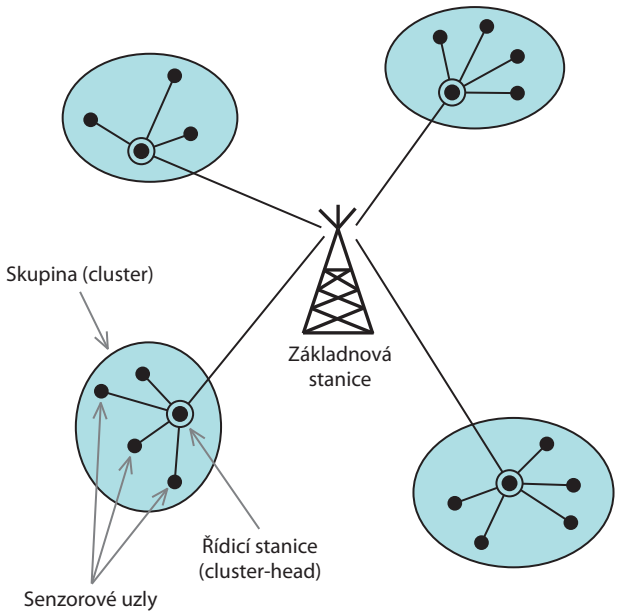
Uzly sítě jsou soustředěny v okolí základnové stanice ve vrstvách podle vzdálenosti. Na obrázku 2.1 je zobrazena vrstvená architektura se třemi vrstvami a je zde naznačen způsob, jakým probíhá komunikace. Základnová stanice může komunikovat se všemi uzly přímo a je typicky dále propojena s klasickou sítí. Uzel komunikuje se základnovou stanicí přímo pouze tehdy, nachází-li se v první vrstvě. Jinak může komunikovat pouze s uzly ze sousední vrstvy.

Jelikož uzly mohou komunikovat pouze na krátkou vzdálenost, je komunikace méně energetický náročná a šetří se tak akumulátor uzlu. Na druhou stranu, i když uzel nepotřebuje s nikým komunikovat, může být využit jako prostředník pro spojení jiného uzlu se základnovou stanicí, což může při nerovnoměrném rozložení zátěže naopak vést k rychlému vybití jeho akumulátoru.

Seskupná architektura

Uzly sítě jsou organizovány do skupin (clusters), kde každá skupina má právě jeden řídicí uzel (cluster-head). Uzly komunikují pouze s řídicím uzlem a ten pak se základnovou stanicí. Situace je znázorněna na obrázku 2.2.

Uzly obvykle shromažďují data a zasílají je přes řídicí uzel základnové stanici. Kromě pouhého přeposílání může řídicí uzel nad daty provádět agregační či jiné výpočty. Základnová stanice tak může získávat již ucelená hlášení z oblastí pokrytých různými skupinami.



Obrázek 2.2: Seskupná architektura.

2.2 Charakteristické vlastnosti WSN

Jednou ze základních vlastností klasických počítačových sítí (např. LAN) je podpora velmi různorodé množiny uživatelů. Každý uživatel má obvykle svůj *vlastní cíl*, ať už je to prohlížení webu, emailů, sledování „streamovaného“ videa či online hraní her. Tento způsob použití vede k vrstvené protokolové architektuře, která podporuje fungování jakékoliv nové aplikace (např. architektura TCP/IP, případně referenční model ISO/OSI).

Výše popsaný přístup ale nelze s výhodou použít pro návrh WSN, neboť ty mají odlišné charakteristické vlastnosti a mnohdy zcela protichůdné požadavky než klasické sítě. Cílem této podkapitoly je popsat nejdůležitější vlastnosti a požadavky na WSN.

Spolupráce uzlů

Pravděpodobně nejdůležitější hledisko, které senzorové sítě odlišuje od klasických, je jejich *cíl*. Senzorová síť jako celek slouží ke konkrétnímu účelu. Například častým úkolem senzorových uzlů je monitorování požadované fyzikální veličiny a informování základnové stanice o situaci v měřené oblasti. Ve snaze dosáhnout tohoto globálního cíle *uzly uvnitř WSN spolupracují*. Tímto se WSN zcela odlišují od ostatních sítí (např. WLAN), kde se každý uzel (uživatel) snaží získat ze sítě co nejvíce pro sebe.

Velikost sítě

Bezdrátové senzorové sítě mohou obsahovat od malého počtu (cca 10–20) až po velmi mnoho uzlů (cca 100–100 000). Díky sériové výrobě a technologickým pokrokům jsou uzly čím dál levnější, což usnadňuje vznik sítí s mnoha senzorovými uzly. Jejich hlavní výhodou je *robustnost*, využijeme-li uzly navíc ke zvýšení jejich hustoty. *Hustota uzlu* udává, kolik dalších uzlů se nachází v jeho komunikačním dosahu. Z tohoto pohledu může být rozložení uzlů ve WSN buď rovnoměrné (všechny uzly mají stejnou hustotu) nebo různorodé (uzly s různou hustotou). Vyšší hustota uzlů dělá síť více robustní vůči poruchám, jelikož existuje více alternativních cest mezi uzlem a základnovou stanicí.

Autoři v práci [43] uvádějí, že zvýšením hustoty uzlů blízko základnové stanice lze dosáhnout prodloužení životnosti[2] sítě oproti WSN s rovnoměrným rozložením. Tyto uzly jsou totiž obvykle první, které vyčerpají kapacitu svých akumulátorů, neboť kromě své vlastní komunikace přeposílají také hodně zpráv od vzdálenějších uzlů. Zvýšením jejich hustoty můžeme lépe rozložit jejich zátěž a tím ušetřit cennou energii.

Model komunikace

V internetu je tradiční způsob komunikace typu klient-server. Nejvíce provozu zde probíhá ve směru od serveru ke klientům. V peer-to-peer sítích pak komunikace probíhá přímo mezi libovolnými dvěma uzly (any-to-any).

Situace v bezdrátových senzorových sítích je ale odlišná. Hlavní úlohou senzorových uzlů je sběr dat, která jsou pak vyhodnocována základnovou stanicí a případně využita pro rozhodování o dalších akcích. Datový provoz v síti tedy probíhá ve dvou směrech:

- *Upstream (many-to-one)* — naměřená data putují od senzorových uzlů směrem k základnové stanici (viz sekce 2.3.2).

[2] Životnost může podle konkrétní aplikace znamenat např. dobu: za kterou se vybije akumulátor prvnímu uzlu; za kterou se vybijí akumulátory určité části WSN; nebo za kterou klesne procento konektivity nebo pokrytí pod danou úroveň. [35]

- *Downstream (one-to-many)* — základnová stanice posílá dotazy nebo příkazy vybraným uzlům sítě (viz sekce 2.3.3).

Ve většině WSN výrazně převládá provoz typu upstream. Komunikace any-to-any se zde nevyskytuje. Výjimkou může být pouze situace, kdy si uzly navzájem vypomáhají při přeposílání paketů směrem k základnové stanici. Typicky jsou ale využity pouze uzly ležící ve směru k ní.

Hardware uzlů

Hardware uzlů je další významnou věcí, kterou se WSN odlišují od jiných typů sítí. V mobilních nebo WLAN sítích jsou uzly velmi složitá a důmyslná zařízení, např. notebooky či chytré telefony. V porovnání s nimi jsou senzorové uzly poměrně jednoduché.

Autoři v práci [31] uvádějí přehled vývoje hardware senzorových uzlů v letech 1998–2007. Zkoumanými parametry byly především frekvence CPU, velikost pamětí RAM a Flash a rychlost bezdrátové komunikace. Různé aplikace WSN mají odlišné nároky na hardware uzlů. Parametry použitých CPU se pohybovaly od nejúspornějších 8 bitových mikrokontrolérů taktovaných na 31 kHz, až po relativně výkonné 32 bitové ARM procesory s taktem 200 MHz. Velikost pamětí RAM se lišila od 64 B po 2 MB. Senzory dále nabízely Flash úložiště s kapacitami od 512 B (vestavěné) až po 2 GB (slot pro SD kartu). Rychlost bezdrátové komunikace se pohybovala v rozmezí 10–1 000 kbps.

Dnes, zejména uvažujeme-li také chytré telefony jako uzly bezdrátové senzorové sítě (viz kapitola 3), můžeme pozorovat nárůst počtu jader a taktu CPU do řádu jednotek gigahertz. Ani paměti nezůstávají pozadu, RAM dnes mohou mít stovky megabajtů až jednotky gigabajtů a kapacity Flash pamětí se počítají na desítky a stovky gigabajtů. Podstatně také narostly maximální teoretické přenosové rychlosti bezdrátových technologií. Zejména Wi-Fi, resp. Bluetooth, kde se již pohybujeme v řádu stovek, resp. desítek, megabitů za sekundu (viz sekce 3.2.2, resp. 3.2.3).

Nedílnou součástí senzorového zařízení bývá akumulátor, od kterého je očekávána výdrž v rámci týdnů, měsíců nebo dokonce let. Možnosti dobíjení jsou často omezené nebo žádné. Podle polohy uzlu je možné např. využít fotovoltaické články pro dobíjení. Proto je při vývoji zařízení kladen důraz na co největší efektivitu algoritmů ve snaze snížit spotřebu energie na minimum. Nutnost efektivních algoritmů se netýká pouze hardware, je třeba optimalizovat také komunikační protokoly.

Škálovatelnost

Bezdrátová senzorová síť může sestávat z velkého množství uzlů. Komunikační protokoly by měly být škálovatelné tak, aby zůstaly efektivní i s narůstajícím počtem senzorových uzlů v síti. Jedním z možných řešení je rozdělení WSN na menší podsítě neboli skupiny (viz Seskupná architektura na str. 8). Každá skupina je pak spravována autonomně, což má za následek menší režii při směrování zpráv.

Rozmístění uzlů

Podle potřeb konkrétní aplikace mohou být uzly rozmístěny buď náhodně v zájmové oblasti (např. detekce lesních požárů), nebo deterministicky na předem zvolené pozice (např. měření teplot nebo seizmických aktivit). Zajištění spojení mezi deterministicky umístěnými uzly je pak zřejmě jednodušší než v případě náhodného rozmístění.

Mobilita uzlů a měnící se topologie sítě

V mnoha případech je pohyblivost uzlů ve WSN omezená nebo vůbec žádná. Na druhou stranu, některé aplikace vyžadují velkou mobilitu (např. monitorování pohybu divoké zvěře nebo tažného ptactva). Jsou-li uzly pohyblivé, topologie sítě se může dynamicky měnit. Je tedy nutné tyto změny reflektovat a správně aktualizovat směrovací informace.

Ke změnám topologie může docházet i v případě statických uzlů, díky změnám počasí nebo jiných okolních vlivů způsobujících útlum či stínění signálu (teplota, vlhkost, ...). To může vést k dočasným výpadkům uzlů ze sítě. K trvalému výpadku pak dojde při poruše uzlu.

V senzorových sítích, kde je zapotřebí jen malá frekvence komunikace, mohou uzly přecházet do úsporného režimu, aby šetřily energii akumulátoru. Topologie sítě se tedy změní i ve chvíli, kdy jeden nebo více uzlů vstoupí do úsporného režimu.

Z výše uvedených příčin má mobilita uzlů největší dopad na změnu topologie bezdrátové senzorové sítě.

2.3 Klasifikace WSN

Každá bezdrátová senzorová síť má unikátní množinu cílů a požadavků. Ty vedou k produkci různého toku dat. Uvažme dva příklady.

Mějme WSN monitorující přírodní podmínky, které ovlivňují růst plodin a chov dobytka. Síť je zaměřená na sběr informací – senzory měří fyzikální veličiny a hodnoty zasílají základnové stanici. Generovaný datový provoz je tedy rovnoměrný (upstream, many-to-one). Nekladou se zde velké nároky na zpoždění (latenci) zasílaných zpráv. Je ale nutné nastavit a udržovat směrovací informace tak, aby přeposílání zpráv základnové stanici bylo co nejméně energeticky náročné.

Dále mějme WSN pro detekci lesních požárů. Síť generuje datový provoz ve shlucích a jen velmi zřídka (množina senzorů detekuje požár). Jsou kladeny velké nároky na zpoždění paketů, zpráva o požáru se musí dostat na základnovou stanici „okamžitě“. Cena režie udržování efektivních směrovacích informací je příliš vysoká, data musí být přeposlána ihned, bez ohledu na to, zda je zvolená cesta energeticky optimální.

Všimněme si, že výše popsané aplikace jsou zcela odlišné. Musíme tedy navrhnout a implementovat aplikačně specifická řešení komunikace pro každou z nich. Zde nastává riziko, že náš návrh bude až příliš specifický a později, i pro velmi podobnou aplikaci, bude potřeba vytvářet zcela nové komunikační protokoly.

Z toho vyplývá, že návrh komunikačních protokolů v bezdrátových senzorových sítích nesmí být moc obecný (malá úroveň optimalizace) ani moc specifický (nové protokoly i pro velmi podobné aplikace). Proto autoři v knize [35] zavádí klasifikaci WSN, která, byť není striktní ani vyčerpávající, nám umožní návrh patřičných protokolů pro každou třídu WSN. Klasifikace je založena na zkoumání následujících hledisek:

- Účel/cíl aplikace.
- Požadavky na doručení dat (latence).
- Vlastnosti toku dat (směr, frekvence, ...).

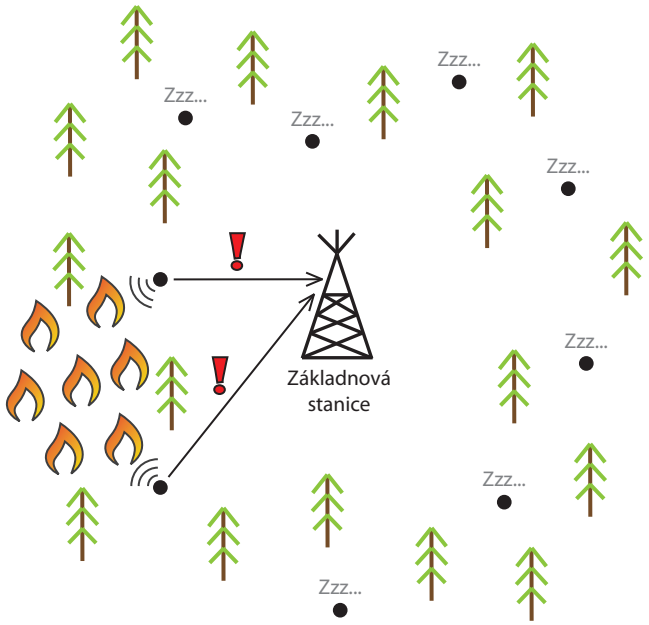
Většina stávajících aplikací WSN spadá do některé z následujících tříd. Kromě popisu vlastností jsou u každé třídy nastíněny časté problémy, se kterými se můžeme potýkat při návrhu WSN spadajících do dané třídy.

2.3.1 Detekce a hlášení událostí

Příklady aplikací WSN spadajících do této třídy mohou být sítě pro vojenský dohled, detekci narušitelů, detekci anomálií ve výrobním procesu nebo detekci lesních požárů. Jejich typickou vlastností je *velmi malá frekvence* výskytu událostí. Proto mohou být většinu času neaktivní a probouzet se z režimu spánku až při detekování události. Hlášení o události musí být poté zasláno základnové stanici co nejrychleji to jen jde. Zpráva obvykle obsahuje identifikaci události spolu s polohou, kde k události došlo.

Typickým problémem na aplikační úrovni je možnost *falešných poplachů*. Zřejmě požadujeme, aby byl jejich počet co nejmenší. Dobrým řešením je nespoléhat se na hlášení pouze jednoho uzlu, ale považovat událost za platnou až při jejím ohlášení více senzory. Shodne-li se více senzorů na tom, že událost nastala, je velmi pravděpodobné, že se nejedná o falešný poplach.

Na obrázku 2.3 je znázorněn příklad chování WSN pro detekci lesního požáru. Zatímco komunikační moduly většiny uzlů jsou neaktivní, dva z nich detekovaly požár a urychleně posílají hlášení základnové stanici.



Obrázek 2.3: WSN pro detekci lesního požáru.

Jednoduché adresování

Tradiční přístup k adresování v počítačových sítích, kdy je každému uzlu přiřazen jedinečný identifikátor (adresa), zde může vést k plýtvání přenosového pásma a výpočetního výkonu. Větší velikost adresy znamená větší packety, které musí uzly zpracovávat a přeposílat. Jelikož WSN této třídy dodržují many-to-one model komunikace, je nepravděpodobné, že budou někdy potřebovat přímo komunikovat mezi sebou (adresovat se navzájem). Tím vzniká prostor pro optimalizaci s cílem zmenšit velikost adresy na minimum.

Proaktivní versus reaktivní směrování

Obecně existují dva přístupy ke směrování ve WSN – *proaktivní* a *reaktivní*. Principem proaktivního směrování jsou periodické aktualizace směrovacích tabulek prostřednictvím kontrolních zpráv zasílaných mezi sousedními uzly. Reaktivní směrování, na druhou stranu, zjišťuje směrovací informace na vyžádání až ve chvíli, kdy jsou potřeba (nedochází k pravidelným výměnám informací mezi uzly). Z popisu výše lze snadno odvodit, že pro WSN sloužící k detekci a hlášení událostí je reaktivní směrování vhodnější.

Stav nečinnosti a úspora energie

Značné množství energie uzlu bývá spotřebováno během pouhého naslouchání na příchozí spojení. To může mít neblahý dopad na životnost celé WSN, protože čas, který uzly stráví smysluplnými výpočty je v porovnání s tímto velmi malý. Nabízí se tedy vypnout bezdrátové moduly uzlů a ponechat aktivní pouze senzory. Pak ale nastává problém v sítích, kde mohou být uzly využity pro přeposílání zpráv od vzdálenějších uzlů směrem k základnové stanici. Řešením může být periodické probouzení uzlů z režimu spánku, buď nezávisle na sobě nebo např. synchronně v rámci skupiny (clusteru). Synchronní probouzení ale vyžaduje dodatečnou režii v komunikačním protokolu, což spotřebu energie opět mírně zvyšuje. Alternativním řešením je použití speciálního úsporného bezdrátového modulu pouze pro naslouchání.

2.3.2 Sběr dat a periodická hlášení

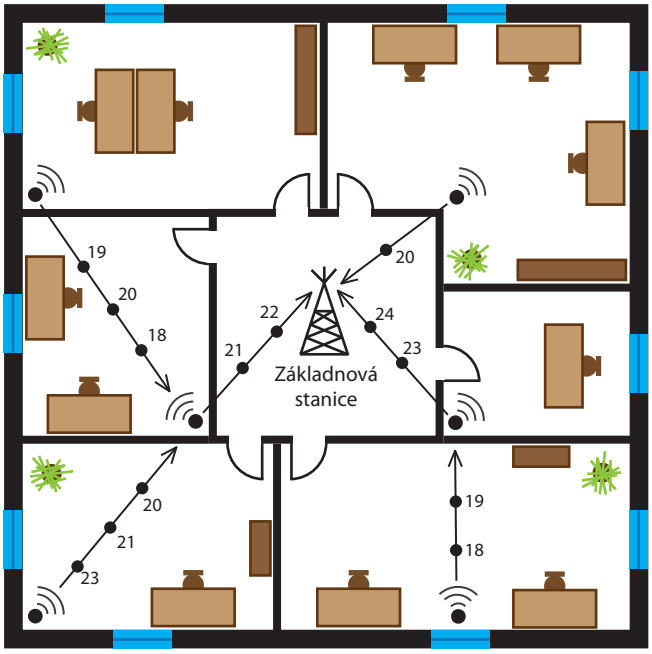
Aplikací WSN spadajících do této třídy je mnoho, např. monitorování přírodních podmínek ovlivňujících růst plodin a chov dobytka nebo monitorování teploty, vlhkosti a osvětlení v kancelářské budově. Hlavní činností uzlů je nepřetržitý sběr dat, která jsou pak *periodicky zasílána* základnové stanici. Mimo agregačních výpočtů může základnová stanice potřebovat vytvořit prostorový model z naměřených hodnot. Pak by měly zprávy obsahovat také informaci o místě, kde došlo k měření.

Kromě senzorů mohou být uzly vybaveny servopohonem. Na základě vyhodnocení distribuovaného měření pak mohou uzly regulovat např. množství hnojiva v půdě nebo vytápění v místnosti.

Senzorové sítě z této třídy typicky generují rovnoměrný datový tok typu many-to-one (upstream). Jelikož měření sousedících uzlů spolu často souvisí, nabízí se využití agregace dat těchto uzlů ještě před jejich odesláním základnové stanici. To vede k redukci počtu přenášených dat a tedy i množství energie vynaložené na bezdrátovou komunikaci. Tím se zvyšuje životnost sítě.

Agregaci dat můžeme obecně provádět dvěma způsoby. Prvním je postavení WSN se seskupnou architekturou (viz str. 8). O agregaci dat se pak stará řídicí uzel každé skupiny. Druhou možností, uvažujeme-li vrstvenou architekturu (viz str. 8), je postupná agregace dat v každém uzlu ve směru přeposílání paketů jednotlivými vrstvami směrem k základnové stanici.

Na obrázku 2.4 je zobrazena WSN monitorující teplotu v kancelářské budově. V každé místnosti je umístěn senzorový uzel, který měří teplotu a odesílá ji základnové stanici (hodnoty uvažujeme ve stupních Celsia). Vzdálenější uzly využívají pro přeposílání zpráv nejbližšího souseda ve směru k základnové stanici. Uzly, které přeposílají data jiných uzlů mohou zároveň provádět jejich agregaci.



Obrázek 2.4: WSN pro monitorování teploty v kancelářské budově (hodnoty uvažujeme ve stupních Celsia).

Úsporné směrování

Jelikož data po celou dobu života WSN neustále proudí od uzlů k základnové stanici, nejvíce energie je spotřebováno na komunikaci. Proto je důležité zvolit strategii směrování tak, aby se životnost sítě co nejvíce prodloužila. Nabízí se řešení v podobě posílání paketů po energeticky nejméně náročných cestách. Tento přístup ale vede k nadměrné zátěži a rychlému vybití akumulátorů uzlů na těchto cestách. Problém lze vyřešit rozdělením zátěže do několika cest.

Vyvažování zátěže v seskupné architektuře

Vyvažování zátěže (load-balancing), popsané v této části, je zaměřeno na seskupnou architekturu. Umožňuje prodloužit životnost WSN. Principem vyvažování je rozdělení zátěže uzlu, která by za normálních okolností znamenala rychlé vybití jeho akumulátoru, mezi více uzlů. Kritickým uzlem v seskupné architektuře je zřejmě uzel řídicí, jelikož má na starosti sběr dat od všech uzlů ve skupině, jejich agregaci a následné přeposílání na vzdálenou základnovou stanici.

Autoři v práci [32] navrhují algoritmus LEACH[3], který je použitelný pro homogenní WSN se seskupnou architekturou. Homogenní znamená, že všechny uzly sítě mají stejné vybavení a schopnosti. Jinými slovy, každý uzel ve skupině se může stát řídicím uzlem. Principem algoritmu LEACH je náhodné a periodické střídání uzlů v roli uzlu řídicího. Zřejmou nevýhodou je, že *každý* uzel sítě musí obsahovat důmyslný hardware řídicího uzlu. Pro WSN s velkým počtem uzlů to může znamenat příliš vysokou pořizovací cenu.

[3]Low-Energy Adaptive Clustering Hierarchy.

2.3.3 Dotazování základnovou stanicí

Dotazování základnovou stanicí je rozšiřující funkcí předcházejících dvou tříd, spíše než třídou čistě samostatnou. Namísto toho, aby uzly periodicky zasílaly svá měření, se základnová stanice dotazuje vybrané množiny senzorových uzlů na jejich aktuálně naměřená data. Může tak ovládat *granularitu měření* v různých částech WSN.

Například mějme WSN pro monitorování výrobního procesu. Pokud uzel detekuje nějakou anomálii, základnová stanice se dotáže jiných senzorových uzlů pro potvrzení a získání více informací o problému.

Dotazování je výhodné, požaduje-li základnová stanice data z různých oblastí sítě v různou dobu. Periodické zasílání naměřených hodnot by v tomto případě bylo zbytečným plýtváním energie jak na komunikaci, tak na výpočetní výkon. Princip dotazování tedy vede k efektivnějšímu využití zdrojů ve WSN.

Adresování uzlů

Pro dotazování základnovou stanicí je potřeba efektivní způsob adresování uzlů. Ve většině aplikací není nutné, aby základnová stanice znala adresy všech uzlů sítě. Obvykle stačí adresovat pouze *určité regiony*, protože když základnová stanice posílá dotaz, má většinou zájem o naměřené hodnoty z konkrétní oblasti WSN.

Dotazovací jazyk

Protože dotazování na data senzorových uzlů má podobný charakter jako dotazování nad databázemi, je použitý dotazovací jazyk založen na standardu SQL[4]. Dotazy pak obsahují dobře známá klíčová slova jako FROM, WHERE, HAVING AVG(), HAVING MAX() a další.

Vzdálená správa uzlů

Možnost komunikace základnové stanice s množinou senzorových uzlů nemusí být využita pouze k dotazování. Základnová stanice může vzdáleně aktivovat a deaktivovat uzly sítě nebo nastavovat jejich parametry. Například frekvenci aktualizací, typ požadovaných dat (má-li uzel více typů senzorů) nebo kritéria a prahy pro detekci události. Z tohoto důvodu je požadována možnost ovládat co nejvíce nastavení hardwaru prostřednictvím softwaru uzlů.

2.3.4 Sledování a dohled

Bezdrátové senzorové sítě této třídy mají široké uplatnění. V armádě mohou sloužit k dohledu nad hranicemi a sledování narušitelů. Při ochraně životního prostředí pomáhají k detekci pravidelných vzorů v pohybu hmyzu nebo ptactva. V silniční dopravě se pak používají statistiky ze sledování dopravy pro plánování stavby dálnic.

Tato třída kombinuje vlastnosti a funkce všech předchozích. Náplní práce těchto WSN je typicky detekce, lokalizace a následné sledování pohybu objektu/cíle. Je-li *detekován* objekt, základnová stanice se to musí dozvědět *co nejdříve*. Poté se může *dotázat* dalších uzlů pro získání časovými razítky označených pozic objektu, které využije pro výpočet jeho trajektorie pohybu.

[4]Structured Query Language.

Určování polohy

Problém určování polohy uzlů se týká skoro všech typů bezdrátových senzorových sítí. U sítí sloužících pro sledování a dohled je ale rozšířen o další rovinu. Kromě polohy uzlů musí totiž WSN určovat také polohu pohybujícího se objektu.

Od určování polohy požadujeme *robustnost* a *úspornost*. Robustností máme na mysli přesné určení polohy objektu i v případech kolísání rádiového signálu nebo změn topologie sítě vlivem selhání uzlu. Úsporností myslíme určení polohy s minimálními možnými nároky na spotřebu energie. Požadavky na robustnost a úspornost jdou proti sobě.

Kapitola 3

Mobilní zařízení jako bezdrátové uzly WSN

Tato kapitola se zabývá myšlenkou využití mobilních zařízení jako *bezdrátových uzlů* WSN. Nejprve se zamyslíme nad motivací pro tento způsob využití. Poté se podíváme na mobilní zařízení z pohledu uzlů WSN a uvedeme přehled požadovaného vybavení (podkapitola 3.1). Následně popíšeme dostupné technologie pro bezdrátovou komunikaci, které jsou dnes v mobilních zařízeních běžně podporovány, a k čemu by se daly využít při návrhu WSN (podkapitola 3.2). Nakonec zdůvodníme výběr mobilní platformy Android, na kterou se při zkoumání blíže zaměříme a uvedeme její možnosti a způsob práce s bezdrátovými technologiemi (podkapitola 3.3). Informace uvedené v této kapitole byly převážně čerpány z knih [36, 41] a z [1, 2, 4, 5, 29, 33].

Klasický uzel WSN je malé akumulátorem napájené zařízení se schopností bezdrátové komunikace, obsahující senzory pro měření různých fyzikálních veličin (viz podkapitola 2.1). Jelikož mobilní zařízení vyhovují této definici, mohla by představovat, především díky nízké ceně a velkému rozšíření, levnější alternativu některých specializovaných senzorových uzlů. Dobrý poměr výbava vs. cena je tedy hlavní motivací myšlenky využití mobilních zařízení jako uzlů WSN. Navíc, díky existenci SDK[1] mobilních platforem, je jejich programování obvykle velmi pohodlné a na poměrně vysoké úrovni (např. v jazycích Java nebo C#).

Pojmem mobilní zařízení zde máme na mysli *chytré telefony* a *tablety*, které disponují otevřeným operačním systémem (např. Android, iOS či Windows Phone). V textu se dále, bez ztráty na obecnosti, zaměříme převážně na chytré telefony (dále jen telefony). To proto, že svou velikostí lépe vyhovují definici uzlu WSN. Navíc bývají z technologického hlediska lépe vybaveny než tablety, které často postrádají např. GSM nebo 3G modul. Z pohledu programovacích technik bychom mezi nimi mohli nalézt odlišnost pouze v přístupu k tvorbě a rozvržení grafického uživatelského rozhraní. Jedná se ale o rozdíl, který z hlediska jejich využití jako uzlů WSN můžeme zanedbat. Velký displej tabletů naopak nahrává jejich možnému využití jako základnové stanice pro vizualizaci měřených hodnot.

3.1 Vybavení a parametry mobilních zařízení

V této podkapitole je uveden přehled vybavení a parametrů důležitých v kontextu využití chytrých telefonů jako uzlů WSN.

[1] Software Development Kit.

Bezdrátová komunikace

Wi-Fi modul je dnes již nedílnou součástí telefonů. Samozřejmostí je schopnost připojení se k Wi-Fi hotspotu. Některá zařízení přidávají možnost vytvořit ad-hoc síť nebo zapnout funkci přenosný Wi-Fi hotspot, kdy je jiné síťové připojení (typicky mobilní internet) poskytnuto dalším zařízením – telefon se chová jako hotspot. Další běžnou součástí telefonů je *Bluetooth* modul, který umožňuje bezdrátově přenášet data mezi dvěma spárovanými zařízeními. Také může být podporován tzv. tethering, který slouží ke sdílení internetového připojení přes Bluetooth.

GSM telefony, které v ČR dominují, mohou dále komunikovat s okolím prostřednictvím *mobilního internetu*, resp. technologií GPRS, EDGE, 3G, případně LTE, které ale není běžně přítomno ve všech zařízeních, a jehož provoz je v ČR zatím pouze testovací a jen ve velmi malých oblastech.

Některá zařízení nabízejí také technologii NFC (Near Field Communication) umožňující bezdrátově přenášet data mezi dvěma k sobě přiloženými zařízeními. V kontextu komunikace uzlů WSN však není podpora NFC dobře využitelná.

Shrneme-li možnosti komunikace, zvolená mobilní platforma by měla podporovat alespoň Wi-Fi nebo Bluetooth. V případě Wi-Fi je velmi žádoucí podpora ad-hoc sítí. Pro přídavné funkce vzniklé bezdrátové senzorové sítě by šlo využít mobilní internet nebo dokonce SMS – například pro nahlášení důležité události nebo vzdálenou aktivaci/deaktivaci uzlu.

Senzory

Telefony jsou obvykle vybaveny základní sadou senzorů – akcelerometr, gyroskop, kamera, mikrofon, světelný senzor a další. Každá aplikace WSN může vyžadovat odlišné typy senzorů. Vybraná platforma by tedy měla poskytovat co nejvíce senzorů pro pokrytí co možná nejširšího spektra aplikací. Senzory telefonů se blíže zabývá kapitola 4.

Výdrž akumulátoru

Uzly WSN by měly vydržet fungovat na akumulátor co možná nejdéle. Zřejmě tedy požadujeme, aby fyzické zařízení na zvolené platformě dokázalo fungovat bez nabití co nejdelší dobu. Výdrž dnešních telefonů se pohybuje podle zátěže v rámci jednotek dnů, což z hlediska WSN není mnoho. Výhodou by mohla být možnost připojení přídavného externího akumulátoru nebo možnost jeho výměny, a to nejlépe za běhu přístroje.

Cena

Jelikož dobrý poměr výbava/cena je hlavní motivací, je potřeba zvolit takovou platformu, jejíž zařízení nejsou drahé tak, že by se jejich použití oproti klasickým uzlům WSN nevyplatilo. Obecně se ceny chytrých telefonů pohybují zhruba od tří do dvaceti tisíc korun.[2]

Rozměry

Všechny mobilní telefony jsou „zařízení do kapsy“. Proto jim z hlediska velikosti nic ve využití jako uzlů WSN nebrání.

[2] Přibližný stav v prosinci 2012.

3.2 Technologie pro bezdrátovou komunikaci

Tato podkapitola popisuje bezdrátové technologie, kterými je typicky vybaven dnešní telefon, a které jsou důležité z hlediska jeho existence jako uzlu WSN. Je uveden stručný popis každé technologie a příklad jejího využití ve WSN. Nakonec je provedeno srovnání spotřeby energie uvedených technologií.

3.2.1 Mobilní síť

Mobilní síť je poměrně abstraktní pojem, který v sobě zapouzdřuje celou řadu konkrétních technologií. Z hlediska WSN nás zajímají ty, které umožňují komunikaci typu stroj ↔ stroj. Jedná se tedy o služby poskytující internetové připojení, a případně také SMS. Uvažujeme v ČR dominantní mobilní síť GSM.

GPRS a EDGE

General Packet Radio Service (GPRS) je rozšířením standardu GSM, které umožňuje přenos paketových dat v této mobilní síti. Jsou-li k dispozici data pro zaslání, GPRS využívá momentálně volné sloty v TDMA rámci GSM komunikace. Time Division Multiple Access (TDMA) rámec se skládá z osmi za sebou jdoucích časových slotů. GPRS může na vyžádání pro sebe alokovat 1–8 těchto slotů. Kolik slotů je možné v danou chvíli alokovat závisí na momentálním vytížení sítě a politice mobilního operátora.

GPRS využívá čtyři kódovací schémata – CS-1 až CS-4. Liší se mírou redundance pro opravu chyb vzniklých během přenosu. CS-1 obsahuje nejvíce redundance, zatímco CS-4 žádnou (v reálném prostředí není použitelné). Systém volí kódování podle aktuální chybovosti. Teoretická maximální rychlost přenosu je 171,2 kbps při využití všech 8 slotů a kódování CS-4. Reálné rychlosti jsou podstatně menší, s kódováním CS-2 typicky 53,6 kbps pro příchozí data (4 sloty) a 26,8 kbps pro odchozí (2 sloty). Kromě malé rychlosti je další nevýhodou GPRS velmi velké RTT (Round Trip Time)[3] paketu přesahující 1 s.

Enhanced Data rates for Global Evolution (EDGE) je modifikace GPRS spočívající v náhradě GMSK modulace signálu za 8-PSK. To přináší nárůst rychlosti. U GSMK modulace nese jeden přenášený symbol informaci o hodnotě jednoho bitu. V případě modulace 8-PSK se v jednom symbolu přenáší hodnoty rovnou tří bitů. Více o modulacích viz [41].

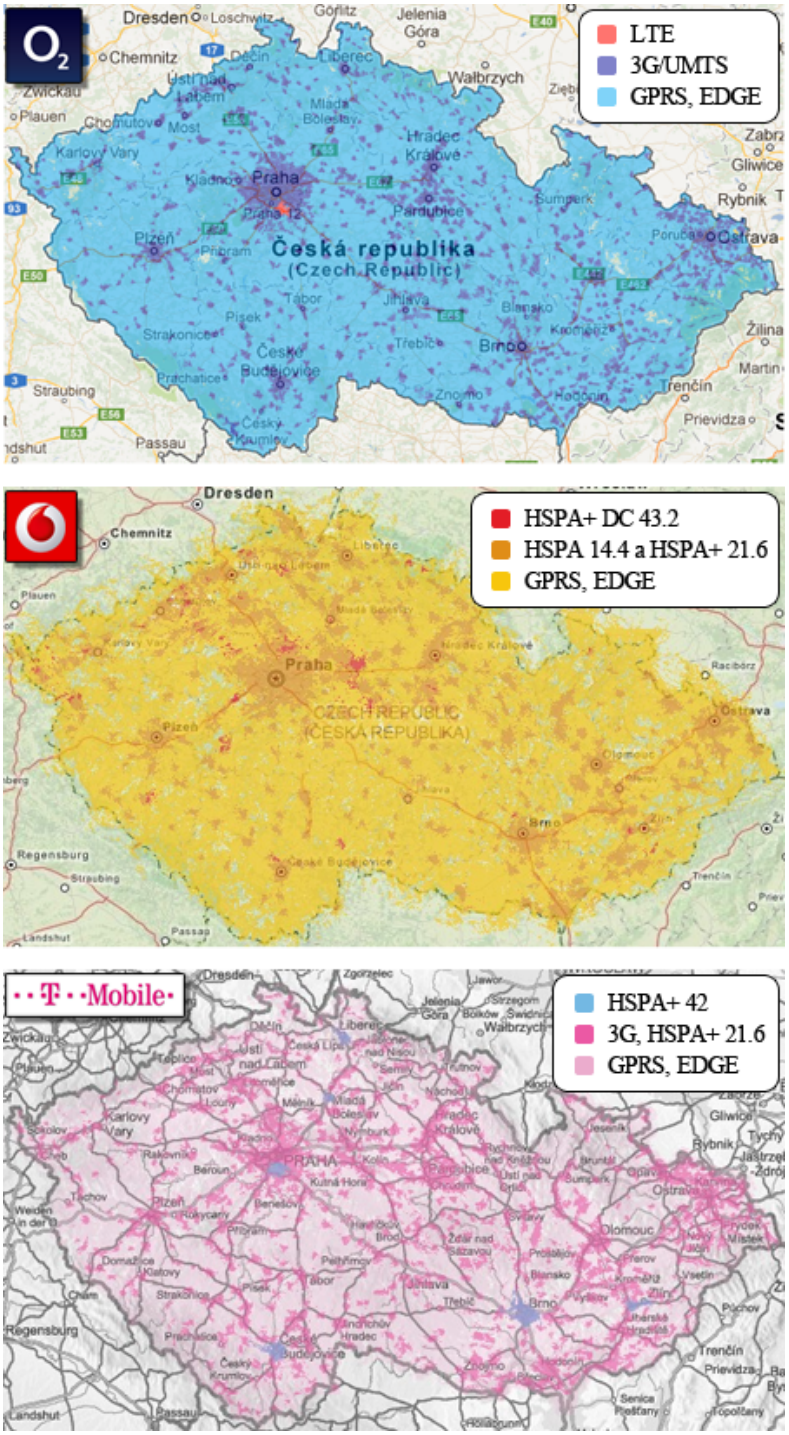
Na obrázku 3.1 jsou zobrazeny mapy pokrytí mobilního internetu operátorů v ČR. Byť jsou mapy, jak uvádějí operátoři, pouze orientační, je vidět, že pokrytí signálem GPRS a EDGE je velmi dobré. Stále však není dostupný úplně všude, problém muže způsobovat například členitý horský terén nebo sklepení budov.

Obě technologie mohou být s výhodou použity pro připojení a komunikaci uzlů přes internet. Díky dobrému pokrytí mohou uzly poměrně spolehlivě komunikovat na větší vzdálenosti mezi sebou nebo se vzdálenou základnovou stanicí. GPRS a EDGE nejsou příliš vhodné, požadujeme-li rychlou odezvu (např. pro detekční WSN) nebo klademe-li vyšší nároky na rychlost přenosu dat. Využití obou služeb je zpoplatněno příslušným mobilním operátorem.

3G a 4G

Označení sítě *3. generace* (3G) nesou nové technologie (UMTS, HSPA, HSPA+, ...), jejichž cílem je další zvýšení rychlosti mobilního internetu. Cílem bylo dosažení 384 kbps, přičemž

[3] Doba potřebná k přenosu signálu od zdroje k cíli a zpět.



Obrázek 3.1: Mapy pokrytí mobilního internetu v ČR (převzato z webových stránek operátorů dne 27. prosince 2012).

dnes je dosahováno typicky již větších rychlostí. Jelikož EDGE poskytuje nejvyšší rychlost jakou lze na GSM dosáhnout, standardy 3G a 4G již vyžadují nové frekvence, nové rádiové rozhraní, atd. – je tedy nutné vybudování nové sítě.

Na obrázku 3.1 je zobrazeno pokrytí ČR signálem 3G podle mobilních operátorů a technologií, které používají. Celá ČR zdaleka není pokryta, signál je soustředěn především na obydlené oblasti. Lze říci, že 3G je dnes dostupné již ve všech velkým, středních a ve většině menších měst.

Sítě *4. generace* (4G) jsou nástupci 3G, které opět navyšují přenosové rychlosti. Komerčně jsou ve světě nasazeny dvě technologie – WiMAX a LTE. Ani jedna z nich se v ČR zatím nedostala do plného provozu. První „vlaštovkou" je malý ostrůvek LTE v okolí Prahy, který slouží pro testování této technologie společností Telefónica O_2, viz obrázek 3.1.

Z hlediska WSN lze 3G a 4G konektivitu v zásadě využít stejným způsobem jako GPRS a EDGE, oproti kterým poskytují lepší RTT, větší rychlosti přenosu, ale podstatně horší pokrytí.

SMS a MMS

Jedná se o služby standardu GSM. *Short Message Service* (SMS) poskytuje přenos krátkých textových zpráv s maximálním počtem 160 znaků. Služba SMS nepoužívá standardní datové kanály GSM. Data jsou přenášena po signalizačních kanálech v době, kdy nejsou využity. Z tohoto důvodu je možné přijmout nebo odeslat SMS i během hovoru. SMS bývá jediná možnost, jak může mobilní síť komunikovat s telefonem. Lze ji využít např. pro aktualizaci softwaru telefonu nebo implementaci tzv. „push" služeb. Služba je dostupná všude tam, kde dosáhne signál GSM.

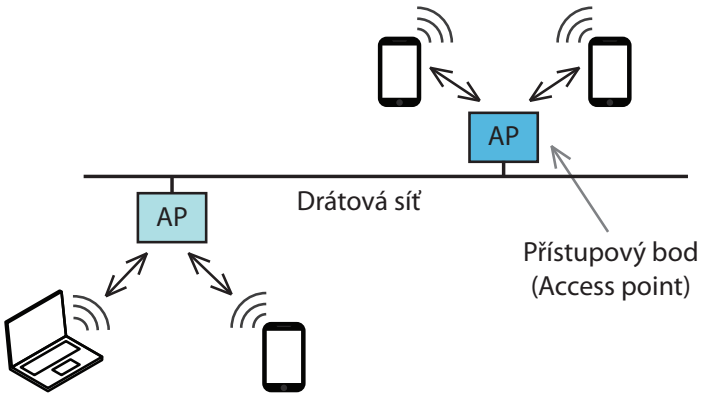
Nástupcem SMS je služba *Enhanced Message Service* (EMS), která navyšuje maximální délku zprávy na 760 znaků a umožňuje standardizovaným způsobem přenášet formátovaný text, malé obrázky a vyzvánění. Nikdy se plně neujala díky nástupu služby *Multimedia Message Service* (MMS), která již nabízí přenos větších obrázků (GIF, JPEG, WBMP), krátkých videoklipů (AMR) aj. Ačkoliv horní hranice není stanovena, velikost zprávy bývá omezena operátorem nebo schopnostmi telefonu. Pohybuje se zhruba v rozmezí 30–300 kB. Počet multimediálních příloh zprávy bývá zpravidla také omezen.

Služba SMS/MMS může být v kontextu WSN využita různě podle potřeby konkrétní aplikace. Například při detekci významné události může mimo jiné uzel poslat správci sítě SMS/MMS notifikaci. Naopak správce sítě může využít SMS ke sdělení uzlu, že se má aktivovat v případě, kdy má uzel v deaktivovaném stavu vypnuta ostatní rádia (Wi-Fi, Bluetooth) a nelze s ním tedy komunikovat běžnou cestou. Jedná se spíše o doplňkové a rozšiřující funkce. Nelze předpokládat, že by služba SMS či MMS byla dobře využitelná jako hlavní prostředek komunikace ve WSN.

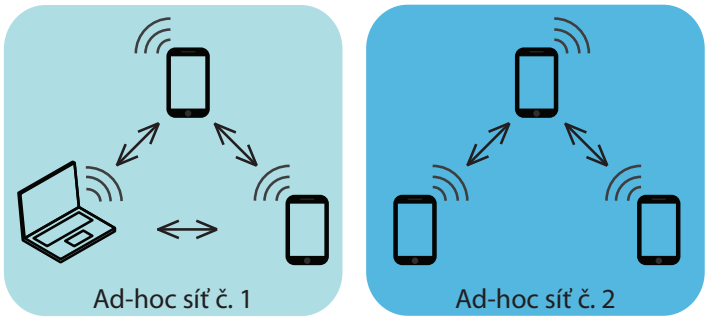
3.2.2 Wi-Fi

Wi-Fi (WLAN, Wireless Local Area Network) je standard 802.11 organizace IEEE, který využívá rádiový přenos dat pro vytvoření lokální počítačové sítě, typicky v rámci jedné instituce. Má několik verzí lišících se použitým frekvenčním pásmem a rychlostmi přenosu. Dnes stále nejpoužívanější verze 802.11b pracuje v pásmu 2,4 GHz a umožňuje přenos dat rychlostí 1–11 Mbps (uživatelská rychlost cca 6 Mbps). Verze 802.11a využívá pásmo 5 GHz a poskytuje rychlosti 6–54 Mbps. Verze 802.11g je výsledkem snahy o zrychlení 802.11b. Je velmi podobná 802.11a, ale pracuje v pásmu 2,4 GHz. Podporuje rychlosti 1–54 Mbps a zachovává zpětnou kompatibilitu s 802.11b.

Dalším vylepšením standardu je verze 802.11n. Může pracovat v pásmech 2,4 GHz i 5 GHz a zvyšuje maximální rychlost na 450 Mbps. Organizace IEEE již pracuje na nové verzi 802.11ac, která rozšiřuje 802.11n a navyšuje maximální rychlost až na 1 300 Mbps.



(a) Infrastrukturní architektura.



(b) Ad-hoc architektura.

Obrázek 3.2: Architektury Wi-Fi.

Architektura sítě

Standard Wi-Fi podporuje dvě architektury komunikace – infrastrukturní a ad-hoc. *Infrastrukturní* architektura umožňuje přístup i do jiných sítí. Bezdrátové uzly nekomunikují přímo mezi sebou, veškerá komunikace prochází přes přístupový bod (AP, Access Point), který zajišťuje přeposílání/směrování zpráv. Situace je znázorněna na obrázku 3.2a. Dva přístupové body, každý se třemi připojenými uzly, jsou propojeny drátovou sítí a tvoří tak jednu větší síť.

Ad-hoc architektura dovoluje přímou komunikaci dvou uzlů. Není tedy potřeba žádná další infrastruktura nebo přístupový bod. Aby mohly dva uzly komunikovat, musí být navzájem v rádiovém dosahu. Pokud nejsou, musí existovat jiné uzly v dosahu, které zprostředkují přeposílání zpráv. Na obrázku 3.2b jsou zobrazeny dvě ad-hoc sítě. Uzly sítě č. 1 jsou všechny navzájem v dosahu, a tedy mohou komunikovat každý s každým. V síti č. 2 zprostředkovává prostřední uzel přeposílání zpráv dvěma uzlům, které nejsou ve vzájemném rádiovém dosahu. Komunikace mezi sítí č. 1 a 2 probíhat nemůže, protože žádné dva z uzlů nejsou v přímém dosahu.

Dosah

Dosah Wi-Fi signálu je dán především použitým přenosovým pásmem a prostředím, ve kterém je signál vysílán. Nejčastější překážkou způsobující útlum signálu jsou stěny budov.

Pásmo 5 Ghz je díky vyšší frekvenci více náchylné k útlumům než pásmo 2,4 GHz. Dosah signálu v pásmu 2,4 GHz je 300 m mimo zástavbu a 30 m v budově. Maximální rychlosti přenosu lze dosáhnout jen do vzdálenosti 10 m. Dosah signálu v pásmu 5 GHz je 100 m mimo zástavbu nebo 10 m v budově. Maximální rychlosti přenosu lze docílit pouze na vzdálenost 5 m.

Využití

Technologii Wi-Fi lze velmi dobře využít pro komunikaci telefonů jako bezdrátových senzorových uzlů ve WSN. Volba způsobu komunikace závisí na konkrétních potřebách a struktuře WSN. Například je možné zvolit infrastrukturní síť pro komunikaci uvnitř skupin v seskupné architektuře WSN, kdy se řídicí uzel skupiny bude chovat jako přístupový bod. Toho lze na telefonu docílit aktivováním funkce přenosného Wi-Fi hotspotu, případně lze využít ad-hoc architekturu pro simulaci tohoto chování. Ad-hoc architekturu také s výhodou využijeme v případech, kdy je potřeba komunikovat přímo mezi uzly, kupříkladu ve vrstvené architektuře WSN.

3.2.3 Bluetooth

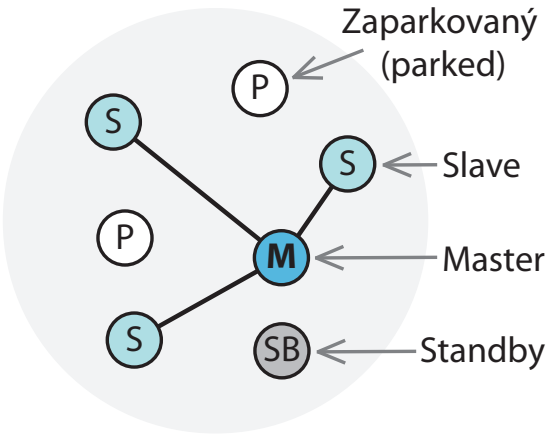
Technologie Bluetooth je standardem 802.15.1 organizace IEEE a spadá do kategorie sítí WPAN (Wireless Personal Area Network). Jde o univerzální rádiový prostředek pro bezdrátové propojení zařízení, typicky jednoho majitele, do ad-hoc sítě (laptop, telefon, tablet, handsfree, ...). Vydávání nových standardů a certifikaci produktů má na starosti organizace Special Interest Group. Pro bezdrátový přenos dat je využito pásmo 2,4 GHz.

V červenci 2010 byla vydána zatím poslední verze specifikace 4.0. Již od verze 1.2 jsou specifikace rozděleny na části popisující klasickou podobu Bluetooth a rozšíření. Verze 4.0 obsahuje dvě rozšíření: pro nízkou spotřebu a pro vysokou rychlost. Rozšíření pro nízkou spotřebu (HF, Hallmark Feature) umožňuje integraci Bluetooth do zařízení jako jsou hodinky, hračky, krokoměry, atd. To vše díky levné implementaci a výdrži provozu na akumulátor v řádu let. Rozšíření pro vysokou rychlost (HS, High Speed) je ve specifikaci přítomno již od předchozí verze 3.0. Hlavním vylepšením je zde nárůst teoretické maximální rychlosti přenosu na 24 Mbps, oproti předchozím 3 Mbps verze 2.1 s rozšířením EDR (Enhanced Data Rate). Rozšíření je možné kombinovat nebo používat samostatně, podle potřeb konkrétního zařízení.

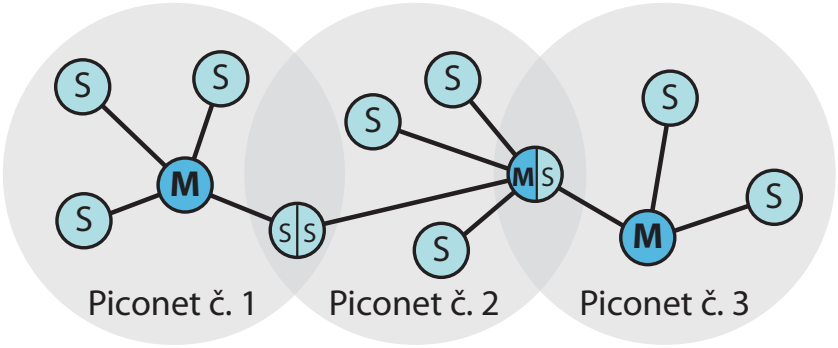
Bluetooth podporuje datové i hlasové přenosy. Datové přenosy (ACL, Asynchronous Connection Less) používají rychlé potvrzování doručení paketů. V případě poškození nebo ztráty je paket znovu zaslán. Naopak hlasové přenosy (SCO, Synchronous Connection Oriented) došlé pakety nepotvrzují, k znovuzasílání tedy nedochází. Na druhou stranu ale garantují horní hranici zpoždění a šířku přenosového pásma. Proto se hodí zejména pro real-time přenosy typu telefonování s handsfree nebo „streamování“ hudby, kde lze občasný výpadek paketu tolerovat.

Piconet a scatternet

Uzly komunikující pomocí Bluetooth tvoří ad-hoc síť zvanou *piconet*. Na obrázku 3.3a je uveden příklad struktury piconetu a role, které v něm mohou uzly zastávat. Vždy právě jeden uzel je v roli *master*. K němu jsou připojeny ostatní aktivní uzly v roli *slave*. Piconet má tedy hvězdicovou topologii. Jelikož je pro aktivní uzly použita pouze 3 b adresa (AMA,



(a) Piconet.



(b) Scatternet.

Obrázek 3.3: Architektura sítě Bluetooth.

Active Member Address), může piconet obsahovat maximálně 8 aktivních zařízení (1 master, až 7 slave uzlů). V síti se také mohou nacházet tzv. *zaparkované uzly* (parked). Tyto sice nejsou aktivní (nemohou komunikovat), ale ostatní uzly v piconetu o nich vědí a v případě potřeby se mohou velmi rychle aktivovat – přepnout do role slave. Pro zaparkované uzly je použita 8 b adresa (PMA, Parked Member Address), může jich být tedy více než 200. Pokud při aktivaci zaparkovaného uzlu není v piconetu místo (7 aktivních slave uzlů), musí se některý slave zaparkovat. Poslední role *standby* pouze označuje uzly, jež se piconetu nijak neúčastní, ačkoliv jsou v komunikačním dosahu.

Síť vytvořená spojením několika piconetů se nazývá *scatternet*. Na obrázku 3.3b jsou zobrazeny tři piconety tvořící jeden scatternet. Spojení může probíhat dvěma způsoby. Jeden uzel je připojen jako slave do dvou piconetů, viz spojení piconetů č. 1 a 2. Případně může být uzel v roli master v jednom piconetu a zároveň slave ve druhém, viz spojení piconetů č. 2 a 3. Nikdy však nemůže být uzel v roli master ve více než jednom piconetu. Komunikace probíhá tak, že se spojující uzel přepíná mezi piconety (obvykle periodicky). Tvorba scatternetů nemusí být podporována ve všech Bluetooth zařízeních.

Dosah

Ačkoliv dosah se může lišit podle výrobce a účelu zařízení, standard definuje jeho minimum. Bluetooth rádia jsou rozdělena do třech tříd. Třída 1 má dosah až 100 m. Třída 2 je nejčastější a poskytuje dosah do 10 m. Signál třídy 3 pak dosáhne pouze do 1 m.

Bezpečnost

Jelikož přes Bluetooth mohou být přenášena citlivá data, je nutné, aby bylo spojení zabezpečeno. Prvním krokem je *párování*, ke kterému dochází, pokud spolu dvě zařízení ještě nikdy předtím nekomunikovala. Uživatel zadá do obou zařízení tajný PIN (1–16 B). Na základě PINu, MAC adresy Bluetooth modulu a náhodně vygenerovaného čísla je vytvořen autentizační klíč (128 b). Ten může být v zařízení uložen trvale. *Autentizace* pak probíhá mechanismem výzva-odpověď (challenge-response), kdy zařízení požadující autentizaci pošle data k zašifrování (výzvu) druhému zařízení, které se chce autentizovat. Zařízení odpoví zašifrovanými daty, jejichž správnost zašifrování první zařízení ověří. Je-li vše v pořádku, autentizace je úspěšná. Nakonec obě zařízení vygenerují *šifrovací klíč* (maximálně 128 b), pomocí kterého je generována pseudonáhodná posloupnost bitů sloužící pro šifrování přenášených dat uživatele (bitovou operací XOR). Pro každý přenos může být generován nový šifrovací klíč.[4]

Využití

Stejně jako Wi-Fi lze i technologii Bluetooth velmi dobře využít pro komunikaci telefonů jako bezdrátových senzorových uzlů WSN. Telefony můžou komunikovat buď přímo mezi sebou (point-to-point), nebo mohou tvořit složitější struktury v podobě piconetů či scatternetů. V případě seskupné architektury WSN se nabízí využití piconetů a přiřazení role master řídícímu uzlu každé skupiny. Uvažujeme-li vrstvenou architekturu WSN, hodí se naopak spíše komunikace point-to-point.

Telefony mohou také umožňovat sdílení internetového připojení prostřednictvím Bluetooth (tethering). To se může hodit např. v případě, kdy chceme mít v rámci skupiny pouze jeden uzel s připojením k internetu (šetření energie, jeden datový tarif, ...) a rádi bychom k němu umožnili přistup i ostatním uzlům.

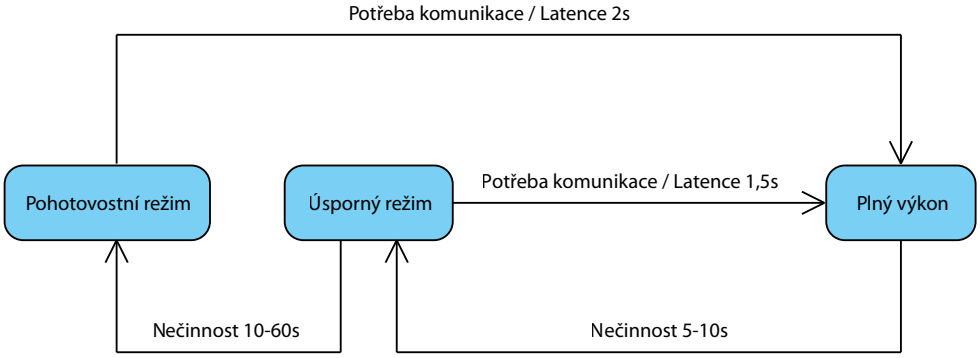
3.2.4 Spotřeba energie

Autoři v práci [4] provedli měření a vyhodnocení spotřeby energie technologií GSM (GPRS, EDGE), 3G a Wi-Fi na mobilních zařízeních. Zjistili, že spotřeba energie je úzce spjata s rozložením přenosů, nejen s celkovou velikostí přenesených dat. Například pár stovek bajtů přenášených nepravidelně přes 3G spotřebuje celkově více energie než jednorázový přenos 1 MB dat. To proto, že ve 3G je velké množství energie (skoro 60 %) vyplýtváno ve stavech s vysokou spotřebou energie již po skončení přenosu dat – vysoká spotřeba přetrvává ještě cca 12 s po skončení přenosu (*tail time*). Tato energie je označována jako *tail energy* a je konstantní – nezáleží na velikosti přenášených dat.

Na obrázku 3.4 je uveden zjednodušený stavový automat 3G rádia z pohledu spotřeby energie, viz [37]. Úsporný režim a plný výkon jsou stavy, ve kterých dochází ke spotřebě energie i po tom, co je komunikace ukončena, viz výše zmíněný pojem tail energy. V závislosti na politice mobilního operátora dochází k přepnutí z plného výkonu na úsporný režim

[4] Více viz [41].

až po 5-10 sekundách nečinnosti. Dále dojde k přepnutí z úsporného režimu na pohotovostní až po uplynutí 10-60 sekund. Tyto časy odpovídají výše zmíněnému pojmu tail time. Například mobilní operátor AT&T v USA má nastaveny hodnoty 5 a 12 sekund. Vodafone pak využívá 5 a 30 sekund. Za povšimnutí také stojí, že na začátku komunikace je zde přítomna latence 2 nebo 1,5 sekundy v závislosti na délce nečinnosti.



Obrázek 3.4: Zjednodušený stavový automat 3G rádia. [37]

V GSM je přítomen obdobný jev jako ve 3G. Rozdíl je ale v tom, že tail time je o polovinu menší než v případě 3G (cca 6 s). Nižší rychlosti přenosu dat GSM dále implikují, že více energie je spotřebováno při samotném přenosu dat. V případě Wi-Fi je spotřeba režie spojení srovnatelná s tail energy ve 3G. Samotný přenos dat je ale výrazně efektivnější než ve 3G, a to pro všechny velikosti přenášených dat.

Autoři v práci [33] měřili spotřebu energie technologií Bluetooth, Wi-Fi a 3G. Spotřebu porovnali také s množstvím přenesených dat. Měření probíhalo na platformě Android souvislým přenosem velkého objemu dat. Ukázalo se, že testovací zařízení[5] dokáže fungovat na akumulátor o cca 4 hodiny déle při použití Bluetooth než v případě Wi-Fi a 3G. Ačkoliv zařízení vydrželo při použití Wi-Fi a 3G zhruba stejně dlouho, technologie Wi-Fi za tuto dobu dokázala přenést dvakrát více dat než 3G. Celkově technologie 3G dokázala přenést nejméně dat, zhruba o polovinu méně než Wi-Fi a Bluetooth. Přestože Wi-Fi a Bluetooth přenesli přibližně stejné množství dat, Bluetooth na to potřebovalo dvakrát více času.

3.3 Bezdrátová komunikace na platformě Android

Na základě analýzy požadavků na vybavení cílové platformy a fyzických zařízení bylo zjištěno, že nároky nejsou moc velké (viz podkapitola 3.1). Musí být přítomen Wi-Fi nebo Bluetooth modul a alespoň základní množina senzorů. Tyto požadavky splňují všechny moderní široce rozšířené mobilní platformy – Android, iOS a Windows Phone. V oblasti výdrže na akumulátor a fyzických rozměrů jsou na tom zařízení těchto platforem velmi podobně. Zařízení s iOS a Windows Phone ale neumožňují jednoduchou výměnu vestavěného akumulátoru. Tento trend se již projevuje i na některých zařízeních s OS Android.

Dále se blíže zaměříme pouze na *platformu Android*, a to především z následujících důvodů. Platforma je nejvíce otevřená pro programátory. Platformy iOS a Windows Phone například neumožňují z kódu aplikace zapnutí a vypnutí Wi-Fi a Bluetooth modulů. To jsou

[5] HTC Desire HD, Android 2.3.

možnosti, které by se nám pro uzel WSN velmi hodily a na Androidu nepředstavují problém, viz níže. Z pohledu přítomnosti různých typů senzorů je na tom Android velmi dobře, viz kapitola 4. Platforma Android také nabízí nejširší výběr zařízení různých cenových hladin, od velmi levných po drahé (cca 2 500–15 000 Kč). Windows Phone je na tom podobně, ale zatím nenabízí tak velký počet zařízení (cca 4 000–15 000 Kč). Telefony iPhone s iOS jsou pak velmi drahé a momentálně jsou k dispozici pouze tři modely (cca 9 500–21 500 Kč).[6]

Hlavním programovacím jazykem platformy Android na aplikační úrovni je Java. Výrazné zastoupení má ale také jazyk XML, např. pro manifest aplikace nebo rozvržení prvků uživatelského rozhraní. Níže uvedené výpisy kódů jsou zapsány právě v těchto jazycích. V této podkapitole je uveden především výčet možností platformy Android. Detaily bezdrátové komunikace a technikami programování se dále podrobně zabývá příloha B. Informace uvedené v této části byly čerpány z knihy [36] a z [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23].

Monitorování připojení

Jelikož má mobilní zařízení k dispozici více možností jak se připojit k internetu (Wi-Fi, GPRS/EDGE, 3G, ...), je užitečné vědět, která z nich je právě aktivní. Třída `ConnectivityManager` umožňuje zjišťovat parametry aktivního připojení a monitorovat jeho změny. Ve výpisu 3.1 je uveden kód pro získání instance této třídy. Pro úspěšné získání jsou zapotřebí dvě oprávnění uvedená ve výpisu 3.2. V příloze B.1 je podrobně uveden postup, jak zjistit detailní informace o typu připojení a jak detekovat jeho změny.

Výpis 3.1: Získání instance třídy `ConnectivityManager`.

```
// Předpokládáme, že volání metody getSystemService() je provedeno ve třídě,
// která je potomkem abstraktní třídy Context (např. aktivita nebo služba).
String service = Context.CONNECTIVITY_SERVICE;
ConnectivityManager cm = (ConnectivityManager) getSystemService(service);
```

Výpis 3.2: Oprávnění potřebná pro získání instance třídy `ConnectivityManager`.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
```

Mobilní data

Mobilní zařízení s operačním systémem Android umožňují připojení k internetu skrze síť mobilního operátora. Sítě 2G (GPRS, EDGE) a 3G (UMTS, HSPA, ...) jsou běžně podporovány, s výjimkou některých tabletů. Podpora 4G sítí (LTE nebo WiMAX) je pak čím dál častější. V nastavení systému Android je možné zvolit *použití pouze 2G sítí*. Internetové připojení tak bude pomalejší, ale méně náročné na spotřebu energie. Toto nastavení bohužel není součástí veřejného aplikačního rozhraní (Application Programming Interface, API) Andoridu, nelze ho tak ovlivňovat z kódu aplikace. Dalším užitečným nastavením je možnost *povolení a zakázání mobilních dat* jako celku (od GPRS až po LTE). Ačkoliv i toto nastavení není součástí veřejného API, lze ho obejít použitím reflexe jazyka Java.

[6] Stav v prosinci 2012.

Toto řešení však není standardní a nemusí fungovat na všech zařízeních nebo budoucích verzích Androidu. Z tohoto důvodu nebude v této práci uvedeno.

Další funkcí, která se týká mobilní sítě, je režim *V letadle*. Jeho zapnutím dojde k vypnutí všech modulů souvisejících s mobilní sítí – GSM, 3G a 4G modul, je-li přítomen. Ve verzích předcházejících Androidu 4.2 bylo možné zapínat a vypínat režim V letadle z kódu aplikace, viz příloha B.2. Od verze 4.2 již ale tento postup nefunguje. Spolu se dvěma předchozími volbami je i tato považována za čistě uživatelskou, do které by aplikace neměly zasahovat.

Komunikace přes internet

Pokud je zařízení úspěšně připojeno k internetu, je možné přes něj komunikovat s okolím klasickým způsobem – typicky klient ↔ server. V příloze B.3 je uveden příklad ustavení HTTP spojení se serverem a je naznačen postup, jakým by probíhala výměna dat.

Za zmínku stojí, že nejnovější verze Androidu požadují, aby síťová komunikace probíhala v samostatném vlákně, tedy odděleně od hlavního vlákna, které je zodpovědné za odezvu grafického uživatelského rozhraní (Graphical User Interface, GUI). Pokus o síťovou komunikaci v GUI vlákně způsobí vyvolání výjimky `NetworkOnMainThreadException`.

Posílání a příjem SMS/MMS

Možnosti zasílání a příjmu SMS a MMS jsou na platformě Android následující. Obojí lze posílat, použitím intentů[7], přes vestavěnou aplikaci (je vyžadována interakce uživatele). Zprávy SMS lze, bez vědomí uživatele, odesílat přímo z aplikace za pomoci třídy `SmsManager`. Přímé odesílání MMS zpráv ale API Androidu nepodporuje. Lze také naslouchat na příchozí SMS, a to i tím způsobem, že je přijatá zpráva „zkonzumována“ a k uživateli se tak nikdy nedostane. Výše uvedené možnosti jsou detailně popsány v příloze B.4.

Výpis 3.3: Získání instance třídy `WifiManager`.

```
// Předpokládáme, že volání metody getSystemService() je provedeno ve třídě,
// která je potomkem abstraktní třídy Context (např. aktivita nebo služba).
String service = Context.WIFI_SERVICE;
WifiManager wifi = (WifiManager) getSystemService(service);
```

Výpis 3.4: Oprávnění pro použití správce Wi-Fi.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
```

Wi-Fi

Správce Wi-Fi, reprezentovaného třídou `WifiManager`, můžeme využít pro zapínání a vypínání Wi-Fi adaptéru, monitorování stavu připojení, hledání okolních hotspotů nebo pro tvorbu a správu konfigurací Wi-Fi sítí. Získání instance třídy `WifiManager` je uvedeno ve výpisu 3.3. Pro jeho použití musí manifest aplikace obsahovat oprávnění pro přístup a změnu stavu Wi-Fi, viz výpis 3.4. Práce s tímto správcem je dále podrobně popsána v příloze B.5. Pokud je zařízení připojeno k internetu pomocí Wi-Fi, může komunikovat

[7] Mechanismus pro spouštění a předávání informací mezi komponentami aplikací, viz [27].

s okolím např. stejným způsobem, jak je uvedeno výše v sekci Komunikace přes internet (str. 28).

Od verze Androidu 2.2 obsahuje nastavení systému položku pro aktivaci sdílení mobilních dat přes Wi-Fi. Z mobilního zařízení se tak stane hotspot poskytující internet připojeným klientům. Veřejné API Androidu ale bohužel neposkytuje žádnou možnost, jak tuto funkci ovládat z kódu aplikace. Nastavení je tedy čistě na uživateli. Android bohužel neobsahuje podporu ad-hoc architektury Wi-Fi.

Wi-Fi Direct

Technologie Wi-Fi Direct je dostupná od Andoridu 4.0. Slouží pro peer-to-peer spojení dvou zařízení pomocí Wi-Fi. Oproti Bluetooth je rychlejší, spolehlivější a funguje na větší vzdálenost. Veřejné API poskytuje prostředky pro hledání dostupných zařízení, jejich připojování a vzájemnou komunikaci. Připojované zařízení podporující Wi-Fi Direct nemusí být nutně jiné mobilní zařízení s OS Android, může se jednat např. o tiskárnu, skener, kameru či televizi.

Inicializace funkce Wi-Fi Direct v aplikaci je uvedena na výpisu 3.5. Hlavní funkčnost poskytuje peer-to-peer správce, reprezentovaný třídou `WifiP2pManager`. Jeho instanci je nutné získat jako první. Před voláním jakékoliv jeho metody musíme vždy nejprve provést inicializaci metodou `initialize(Context, Looper, ChannelListener)`. Ta vytvoří a vrátí nový komunikační kanál reprezentovaný třídou `Channel`. Bude využit později při volání dalších metod. Pro použití funkce Wi-Fi Direct je nutné v manifestu aplikace deklarovat oprávnění uvedená ve výpisu 3.6.

Zapnutím funkce, hledáním dostupných zařízení, připojením a komunikací se detailně zabývá příloha B.6. Nevýhodou možného využití Wi-Fi Direct v rámci WSN je nutnost interakce s uživatelem při spojování dvou zařízení.

Výpis 3.5: Inicializace funkce Wi-Fi Direct.

```
// Předpokládáme, že kód je umístěn ve třídě, která je potomkem
// abstraktní třídy Context (např. aktivita nebo služba).
// Získání správce Wi-Fi Direct.
String service = Context.WIFI_SERVICE;
WifiP2pManager wifiDirect = (WifiP2pManager) getSystemService(service);

// Inicializace Wi-Fi Direct - vrátí nový komunikační kanál.
Channel channel = wifiDirect.initialize(this, getMainLooper(),
    new ChannelListener() {
        public void onChannelDisconnected() {
            // Kanál byl odpojen - zde by se typicky provedl
            // opětovný pokus o inicializaci ...
        }
    }
);
```

Výpis 3.6: Oprávnění pro použití funkce Wi-Fi Direct.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

Bluetooth

Technologie Bluetooth slouží na Andoridu pro komunikaci typu peer-to-peer. Veřejné API umožňuje správu Bluetooth adaptéru, hledání dostupných zařízení, připojování se a přenos dat. Práce probíhá přes správce Bluetooth, jenž je reprezentován třídou `BluetoothAdapter`. Její instanci získáme statickou metodou `getDefaultAdapter()`, viz výpis 3.7. Pokud zařízení nepodporuje Bluetooth, vrací metoda `null`. Pro využití všech metod správce Bluetooth jsou zapotřebí dvě oprávnění, viz výpis 3.8.

Výpis 3.7: Získání instance třídy `BluetoothAdapter`.

```
BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();
boolean isBtSupported = (btAdapter != null);
```

Výpis 3.8: Oprávnění pro použití Bluetooth.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Správce obsahuje užitečné metody, např. `getAddress()` pro zjištění hardwarové adresy adaptéru nebo `getName()` a `setName(String)` pro zjištění a nastavení názvu zařízení. Tyto ale vrací platné hodnoty pouze v případě, že je Bluetooth zapnuto. V opačném případě vrací `null`. Zapnutí můžeme ověřit pomocí metody `isEnabled()`, případně můžeme zjistit stav adaptéru metodou `getState()`, která vrací hodnoty konstant `STATE_OFF`, `STATE_TURNING_ON`, `STATE_ON` a `STATE_TURNING_OFF`.

V příloze B.7 je podrobně popsáno, jak zapnout a vypnout Bluetooth adaptér, jak nastavit a monitorovat viditelnost zařízení, jak vyhledat dostupná Bluetooth zařízení v okolí, jak ustavit spojení na straně klienta a na straně serveru a jak se případně vyhnout nutnosti párování.

Od Androidu 2.2 můžeme využít tethering mobilního internetu přes Bluetooth. Stejně jako v případě Wi-Fi hotspotu lze funkci aktivovat pouze ručně v nastavení systému. Veřejné API manipulaci s touto funkcí neumožňuje.

GCM

GCM pro Android (Google Cloud Messaging for Android) je služba, kterou poskytuje společnost Google. Jde v podstatě o implementaci push notifikací. Služba umožňuje posílat krátké zprávy ze serveru přímo na zařízení s Androidem 2.2 nebo vyšším. Zde je uvedeno pouze pro úplnost, podrobnosti lze nalézt v [23].

Kapitola 4

Mobilní zařízení jako senzorové uzly WSN

V následující kapitole se zaměříme na různé typy senzorů, které lze nalézt v mobilních zařízeních. Nejprve uvedeme obecný přehled typů senzorů a jejich popis (podkapitola 4.1). Poté se zaměříme na senzory podporované platformou Android a na způsob práce s nimi (podkapitola 4.2). Pro srovnání uvedeme také typy senzorů podporované konkurenčními platformami iOS a Windows Phone. Informace uvedené v této kapitole byly čerpány z knih [36, 39] a z [6, 7, 8, 9, 29, 34, 40].

4.1 Typy senzorů

V této části se zaměříme na různé typy senzorů, které lze nalézt v mobilních zařízeních. Některé z nich jsou dnes již nedílnou součástí každého mobilního zařízení (např. akcelerometr nebo magnetometr), jiné na široké rozšíření teprve čekají (např. senzor teploty vzduchu nebo relativní vlhkosti). Pro lepší přehlednost rozdělíme senzory do tři skupin – pohybové, polohové a snímající okolní prostředí (viz [9]). Uvedeme, jaké fyzikální veličiny měří a k čemu je lze typicky využít.

Všechny senzory nemusí být nutně přítomny v zařízení jako hardwarová komponenta. Některé mohou být emulovány softwarově a své hodnoty počítat z dat jiných hardwarových senzorů. Jedná se o tzv. *virtuální senzory*. Způsob implementace senzorů na konkrétním fyzickém zařízení je závislý na jeho výrobci a použité platformě.

4.1.1 Pohybové senzory

Pohybové senzory jsou užitečné pro monitorování pohybu zařízení, např. náklonu, zatřesení, rotace nebo švihnutí. Typicky měří zrychlení a rotační síly podél třech souřadných os. Pohyb zařízení bývá přímou odezvou na vstup uživatele, např. řízení letadla nebo ovládání kuličky ve hrách. S hodnotami se pak pracuje relativně vzhledem k referenční poloze zařízení. Tu je obvykle možné v aplikacích nebo hrách změnit prostřednictvím kalibrace v nastavení. Pohyb ale může být také odezvou na okolní prostředí, kde se zařízení nachází, např. v jedoucím autě. V tomto případě je referenčním bodem vztažná soustava zeměkoule.

Akcelerometr

Akcelerometr měří zrychlení (v m/s^2), které působí na zařízení podél všech tří souřadných os (x, y a z). Zrychlení (akcelerace) je definováno jako velikost změny rychlosti v čase. Jelikož jde o vektorovou veličinu, udává také směr této změny. Hodnoty jsou měřeny vzhledem ke vztažné soustavě země.

Do měření je navíc započítáno i gravitační zrychlení ($g = 9{,}81\,m/s^2$). Proto, je-li zařízení v klidu, vrací akcelerometr hodnotu g na ose kolmé k zemskému povrchu. Naopak padá-li zařízení volným pádem směrem k zemi se zrychlením g, vrací akcelerometr na této ose hodnotu $0\,m/s^2$. Pro získání reálné akcelerace zařízení musí být gravitační síla z dat odstraněna. Toho lze dosáhnout použitím filtru typu horní propust. Analogicky můžeme aplikováním filtru typu dolní propust získat velikost gravitační síly.

Používá se typicky k detekci pohybu jako je např. náklon nebo zatřesení. Musí být přítomen v podobě hardwarové komponenty.

Lineární akcelerace

Stejně jako akcelerometr i senzor lineární akcelerace měří zrychlení zařízení (v m/s^2). Do naměřených hodnot ale nezapočítává gravitační zrychlení. Principálně senzor vrací hodnoty dané vztahem 4.1 (viz [7]).

$$linearni\_akcelerace = akcelerace - akcelerace\_vlivem\_gravitace \tag{4.1}$$

Senzor může být přítomen v podobě hardwarové komponenty, častěji se ale vyskytuje pouze jako senzor virtuální, kdy počítá lineární akceleraci z dat naměřených klasickým akcelerometrem za pomoci filtru typu horní propust. Používá se pro detekci pohybu, např. zatřesení, pohyb nahoru a dolů, atp.

Gravitace

Senzor gravitace měří gravitační zrychlení (v m/s^2), které působí na zařízení ve všech třech osách (x, y a z). Udává tedy směr a velikost gravitace. Ačkoliv může být hardwarový, převážně se vyskytuje jako virtuální, kdy získává gravitační zrychlení aplikací filtru typu dolní propust na výstup akcelerometru. Je-li zařízení v klidu, výstup senzoru by měl být shodný s výstupem akcelerometru. Používá se pro detekci pohybu, např. náklon do stran.

Gyroskop

Gyroskop je hardwarový senzor, který poskytuje údaje o natočení okolo třech souřadných os zařízení (x, y a z). Naměřené hodnoty jsou typicky v radiánech za sekundu (rad/s). Hodnoty jsou, na rozdíl od akcelerometru, měřeny vzhledem ke vztažné soustavě daného zařízení. Lze jej využít především pro detekci rotace, např. natočení, obrácení, atp.

Jelikož gyroskop ve skutečnosti měří rychlost (ne směr), abychom detekovali aktuální natočení, musí být výsledky jeho měření integrovány v čase. Výsledná hodnota pak reprezentuje změnu orientace podle dané souřadné osy. Z tohoto důvodu je třeba gyroskop buď kalibrovat, nebo použít další senzory ke zjištění počáteční orientace.

Také je třeba zmínit, že budeme-li se spoléhat pouze na data z gyroskopu, mohou se chyby získaných hodnot integrací v čase stále více zvyšovat (tzv. drift). To především díky chybám v kalibraci a přítomnosti šumu. Aby se tomuto předešlo, bývá gyroskop používán v kombinaci s dalšími senzory, obzvláště s akcelerometrem.

4.1.2 Polohové senzory

Polohové senzory je možné využít ke zjištění fyzické pozice zařízení vzhledem ke vztažné soustavě zeměkoule, např. určení polohy vůči magnetickému severnímu pólu (kompas). S hodnotami lze také pracovat vzhledem k referenčnímu bodu, který je dán konkrétní aplikací. Ačkoliv GPS přijímač není senzorem v pravém slova smyslu, je přidán do této kategorie, neboť primárně také poskytuje informace o poloze zařízení.

Magnetometr

Magnetometr je hardwarový senzor měřící velikost magnetického pole (v μT) pro všechny tři souřadné osy (x, y a z). Slouží k monitorování magnetického pole Země, avšak ve skutečnosti měří jakékoliv magnetické pole. V naměřených hodnotách se tedy může vyskytovat šum vlivem přítomnosti jiných zmagnetizovaných komponent, např. vibrační motorek, reproduktor, mikrofon nebo Bluetooth čip. Typickou aplikací magnetometru je kompas, což je také důvod, proč je magnetické pole měřeno ve všech třech osách. Dovoluje to uživateli držet kompas v jakékoliv poloze (ne jen vodorovné). Pro tvorbu kompasu je používán magnetometr spolu s akcelerometrem.

Virtuální senzor rotace

Existuje více způsobů, jak reprezentovat aktuální natočení zařízení. První využívá relativní změny úhlu otočení okolo každé z os x, y a z. Jednoduše definujeme, o kolik stupňů se zařízení pootočilo kolem svých os od předchozí pozice. Druhým způsobem jsou tzv. Eulerovy úhly, které natočení reprezentují následující trojicí:

- úhel naklopení (pitch, elevation) – natočení kolem příčné osy zařízení.
- úhel naklonění (roll) – natočení kolem podélné osy zařízení.
- úhel vybočení (yaw, azimuth) – natočení kolem svislé osy zařízení.

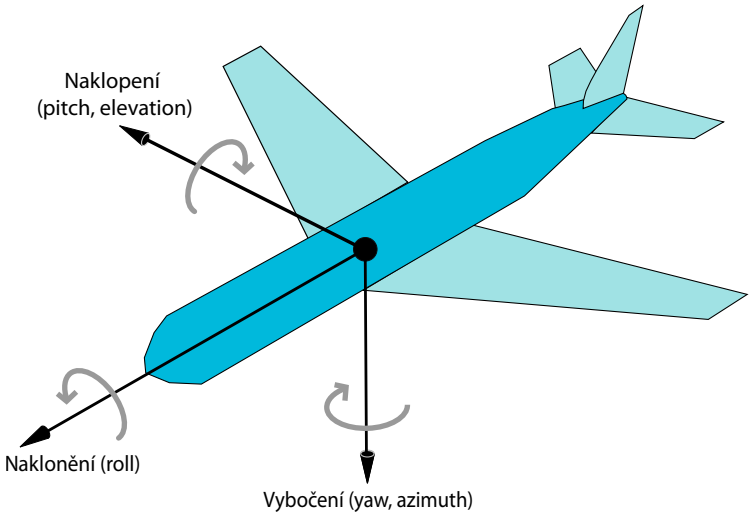
Na obrázku 4.1 jsou výše popsané úhly znázorněny. Problém nastává, natočíme-li zařízení kolmo nahoru nebo dolů (např. naklopení 90°). V této pozici totiž nerozeznáme naklonění od vybočení. Ve spoustě aplikací to ale nevadí, např. při použití v automobilech, ponorkách nebo dopravních letadlech se kolmé natočení nahoru či dolů nepředpokládá.

Třetím způsobem reprezentace natočení zařízení jsou rotační matice. Jedná se o matice velikosti 3×3, které reprezentují natočení podle tří souřadných os. Otočený bod dostaneme vynásobením původního bodu rotační maticí. Poslední způsob vyjádření je pomocí úhlu otočení kolem jedné osy definované třemi čísly. Toto lze reprezentovat např. pomocí kvaternionů[1]. Jednou z výhod použití kvaternionů je možnost snadné interpolace/extrapolace mezi dvěma danými pozicemi.

Výše popsané reprezentace natočení lze získat spojením dat naměřených kombinací jiných senzorů (typicky akcelerometr, gyroskop a magnetometr). Každý z nich má sice své nevýhody, ale naštěstí každý jiné. Dohromady si tedy mohou vypomoci a vzájemně své nedostatky minimalizovat (viz Sensor fusion [40]). Využití nacházejí při detekci rotačního pohybu a aktuálního natočení zařízení ve všech šesti stupních volnosti (6 Degrees Of Freedom, 6DOF)[2].

[1] Více o kvaternionech a reprezentaci rotace viz [30].

[2] Tři stupně pro posun na osách x, y a z, plus tři stupně definující otočení na každé z nich [40].



Obrázek 4.1: Eulerovy úhly.

Přiblížení

Senzor přiblížení (proximity sensor) je hardwarová komponenta měřící vzdálenost objektu od zařízení (obvykle relativně vzhledem k displeji). Některé senzory vrací přiblížení v centimetrech, jiné poskytují pouze binární informaci, zda je objekt blízko (near) nebo daleko (far). Objekt je daleko, překročí-li vzdálenost 5 cm (práh se ale může senzor od senzoru lišit). Senzor je typicky umístěn na přední straně zařízení, konkrétně nad displejem, vedle sluchátka pro telefonování. Používá se ke zjištění, zda je telefon během hovoru držen u ucha. Pokud ano, zařízení zhasne obrazovku, aby se předešlo interakci kapacitního displeje s tváří.

GPS

Global Positioning System (GPS) je satelitní navigační systém s celosvětovým pokrytím. Poskytuje informace o pozici, rychlosti, směru pohybu a přesném čase (satelity obsahují atomové hodiny nutné pro přesnou synchronizaci). Směr pohybu a rychlost se počítají z více po sobě jdoucích měření.

Přesnost získané polohy se pohybuje v jednotkách metrů. Pro reprezentaci pozice je použit globální geocentrický referenční systém (WGS-84, World Geodetic System of 1984), jehož počátek leží v těžišti zeměkoule. Konkrétní pozice zařízení je určená pomocí zeměpisné šířky a délky udávané ve stupních[3].

V mobilních zařízeních bývá obvykle implementována asistovaná GPS (A-GPS). Jejím hlavním účelem je urychlit nalezení rozmístění viditelných satelitů na obloze při zapnutí GPS modulu po delší době nečinnosti. Pro tento účel je využito internetové připojení ke stažení přesných dat o polohách satelitů (tzv. almanach a efemeridy).

Není-li dostupný GPS signál (např. uvnitř budov), můžeme na mobilních zařízeních obvykle zjistit polohu s menší přesností pomocí signálu z dostupných základnových stanic GSM (BTS, Base Transceiver Station) nebo přístupových bodů Wi-Fi (AP, Access Point) v okolí.

[3]Velikost jednoho stupně je pro šířku a délku odlišná.

4.1.3 Senzory snímající okolní prostředí

Senzory této kategorie umožňují snímat parametry okolního prostředí, např. teplotu vzduchu, tlak, osvětlení nebo vlhkost. Ve všech případech se jedná o hardwarové senzory. Jejich výhodou je jednoduchost použití, jelikož typicky nevyžadují kalibraci, filtrováni či jinou modifikaci naměřených hodnot. Do kategorie jsou také přiřazeni kamera, fotoaparát a mikrofon, neboť také umožňují snímat okolí.

Ambientní osvětlení

Světelný senzor měří hodnotu intenzity osvětlení v okolí (ambientní). Jednotkami jsou luxy (lx). Používá se převážně pro automatickou regulaci jasu displeje.

Teplota vzduchu

Senzor měří ambientní pokojovou teplotu obvykle ve stupních Celsia (°C). Využívá se k monitorování okolní teploty vzduchu.

Teplota zařízení

Senzor měří teplotu uvnitř zařízení obvykle ve stupních Celsia (°C). Konkrétní implementace závisí na daném zařízení, např. senzor může být přítomen v akumulátoru a vracet jeho teplotu.

Tlak vzduchu

Senzor atmosférického tlaku (barometr) měří ambientní tlak vzduchu v hektopascalech (hPa) nebo milibarech (mbar, 1 hPa = 1 mbar). Může být využit k monitorování změn tlaku vzduchu pro meteorologické účely nebo k výpočtu nadmořské výšky porovnáním aktuálního tlaku s hodnotou u hladiny moře.

Relativní vlhkost

Senzor měří relativní vlhkost okolí, která udává poměr množství vodní páry obsažené ve vzduchu vzhledem k množství páry ve stejném objemu vzduchu o stejné teplotě při jeho plném nasycení. Udává se v procentech.

Kromě monitorování relativní vlhkosti lze senzor využít, v kombinaci se senzorem teploty vzduchu, k výpočtu a monitorování rosného bodu a absolutní vlhkosti. *Rosný bod* je teplota, při které je vzduch plně nasycen vodními parami (relativní vlhkost je 100 %). Klesne-li teplota pod rosný bod, nastává kondenzace (vodní pára se mění na vodu). Výpočet rosného bodu t_d (v °C) je uveden v rovnici 4.2 (viz [6]), kde RH je relativní vlhkost (v %) a t je teplota vzduchu (v °C).

$$t_d = 243,12 \times \frac{ln\left(\frac{RH}{100}\right) + \frac{17,62 \times t}{243,12+t}}{17,62 - \left[ln\left(\frac{RH}{100}\right) + \frac{17,62 \times t}{243,12+t}\right]} \tag{4.2}$$

Absolutní vlhkost udává hmotnost vodní páry obsažené v jednotce objemu vzduchu. Počítá se v gramech na metr krychlový (g/m^3). Výpočet absolutní vlhkosti d_v (v g/m^3) je uveden v rovnici 4.3 (viz [6]), kde opět RH je relativní vlhkost (v %) a t je teplota vzduchu (v °C).

$$d_v = 216,7 \times \frac{6,112 \times \frac{RH}{100} \times e^{\frac{17,62 \times t}{243,12+t}}}{273,15+t} \tag{4.3}$$

Kamera, fotoaparát a mikrofon

V dnešní době bychom asi jen těžko hledali chytrý telefon, který neobsahuje kameru, fotoaparát a mikrofon. Fotoaparát a kamera bývají spojeny do jednoho čipu, který umožňuje jak pořizování fotografií (např. 5, 8 nebo 12 Mpx), tak i nahrávání video sekvencí (až ve Full HD rozlišení – 1080 px na výšku). Často bývá přítomna dodatečná přední kamera s nižším rozlišením pro video hovory.

Mikrofon je osazen samostatně a využívá se např. k telefonním hovorům, pro funkci diktafonu či pro zaznamenání zvukové stopy videa. Někdy může být přítomno více mikrofonů na různých stranách zařízení, které jsou pak využity pro redukci šumu a hluku z okolí.

4.2 Senzory na platformě Android

Platforma Android podporuje všechny typy senzorů uvedené v podkapitole 4.1 výše. Ne všechny jsou ale podporovány ve všech verzích systému. Tabulka 4.1 obsahuje přehled podporovaných typů senzorů, spolu s odpovídajícími konstantami pro jejich získání v kódu aplikace, a verzí Androidu, od které jsou zde přítomny (viz [9]).

Senzor	**Konstanta**	**Od verze**
Pohybové senzory		
Akcelerometr	`TYPE_ACCELEROMETER`	1.5
Lineární akcelerace	`TYPE_LINEAR_ACCELERATION`	2.3
Gravitace	`TYPE_GRAVITY`	2.3
Gyroskop	`TYPE_GYROSCOPE`	2.3
Polohové senzory		
Magnetometr	`TYPE_MAGNETIC_FIELD`	1.5
Virtuální senzor rotace	`TYPE_ROTATION_VECTOR`	2.3
Senzor orientace	~~`TYPE_ORIENTATION`~~	1.5
Přiblížení	`TYPE_PROXIMITY`	1.5
Senzory snímající okolní prostředí		
Ambientní osvětlení	`TYPE_LIGHT`	1.5
Teplota vzduchu	`TYPE_AMBIENT_TEMPERATURE`	4.0
Teplota zařízení	~~`TYPE_TEMPERATURE`~~	1.5
Tlak vzduchu	`TYPE_PRESSURE`	2.3
Relativní vlhkost	`TYPE_RELATIVE_HUMIDITY`	4.0

Tabulka 4.1: Přehled senzorů platformy Android.

Přeškrtnuté konstanty v tabulce značí typy senzorů, které byly označeny jako zastaralé (deprecated) a neměly by se již používat. Senzor orientace je zastaralý od Androidu 2.2, kvůli jeho výpočetní náročnosti a velikým nepřesnostem. Senzor teploty zařízení je zastaralý

od Androidu 4.0, kde byl, díky rozdílnosti v implementacích napříč různými zařízeními, nahrazen senzorem teploty vzduchu.

Pro srovnání nyní uvedeme senzory podporované platformami iOS a Windows Phone. Na iOS (viz [3]) můžeme využít trojici akcelerometr, magnetometr a gyroskop. Core Motion framework slučuje data z akcelerometru a gyroskopu pro snadnější použití. Ačkoliv fyzická zařízení mohou být vybavena senzorem osvětlení a přiblížení, neexistuje veřejné API, které by umožnilo získat jimi měřená data. Samozřejmostí je pak přítomnost a možnost práce s GPS, kamerou, fotoaparátem a mikrofonem.

Na Windows Phone (viz [38]) je situace obdobná, máme k dispozici akcelerometr, magnetometr i gyroskop. Nad nimi je přítomno tzv. Motion API, které slučuje virtuální senzory pro pohodlnější práci (lineární akcelerace, rotace, ...). Ve verzi normal využívá akcelerometr a magnetometr. Verze enhanced přidává navíc gyroskop, což má za následek zlepšení přesnosti. Stejně jako na iOS, ani zde není přítomno veřejné API pro jinak běžně osazované senzory osvětlení a přiblížení. Opět můžeme využít GPS, kameru, fotoaparát i mikrofon.

Z výčtu senzorů je patrné, že platforma Android je na tom nejlépe. Podporuje bez omezení všechny typy senzorů přítomné na konkurenčních platformách a přidává ještě několik navíc (tlak, vlhkost, ...).

Surová data všech senzorů uvedených v tabulce 4.1 lze na platformě Android snímat jednotným způsobem. Pro práci se senzory je zde třída `SensorManager`. Ta umožňuje získávat instance senzorů (třída `Sensor`), registrovat a odregistrovat listener na události senzoru (rozhraní `SensorEventListener`) a poskytuje konstanty a pomocné statické metody, které můžeme využít např. při interpretaci hodnot virtuálního senzoru rotace.

Třída `Sensor` slouží k reprezentaci konkrétního fyzického senzoru. Poskytuje metody, kterými lze zjistit typ a parametry senzoru (např. rozsah hodnot a rozlišení), jeho název, verzi, výrobce a spotřebu energie. Při registraci listeneru na události senzoru musíme určit senzor, vzorkovací frekvenci a vlastní listener.

Android poskytuje čtyři předdefinované hodnoty vzorkovací frekvence, z nichž každá se hodí k jinému účelu, viz tabulka 4.2. Od Androidu 3.0 lze kromě těchto specifikovat také konkrétní hodnotu v mikrosekundách. Jakákoliv hodnota vzorkovací frekvence je ale systémem vnímána pouze jako doporučení. Systém si v praxi frekvenci upravuje dle potřeby. Je doporučeno nastavit co možná nejmenší frekvenci použitelnou pro danou aplikaci, neboť systém obvykle poskytuje data rychleji. Navíc vyšší frekvence znamená také větší zatížení procesoru, a tím i větší spotřebu energie.

Konstanta	**Prodleva [ms]**	**Příklad použití**
`SENSOR_DELAY_FASTEST`	0	—
`SENSOR_DELAY_GAME`	20	Ovládání her
`SENSOR_DELAY_UI`	60	Aktualizace GUI
`SENSOR_DELAY_NORMAL`	200	Natočení obrazovky

Tabulka 4.2: Vzorkovací frekvence senzorů na platformě Android.

Listener na události senzoru musí implementovat rozhraní `SensorEventListener`, které obsahuje dvě metody. Metoda `onSensorChanged(SensorEvent)` je volána vždy ve chvíli, kdy jsou k dispozici nová data ze senzoru. Její tělo by mělo být co nejkratší, protože je pravděpodobné, že v průběhu měření bude volána velmi často. Instance třídy `SensorEvent`, předaná přes parametr, reprezentuje událost na senzoru. Obsahuje senzor,

na kterém došlo k události, časové razítko[4], přesnost senzoru a především pole reálných čísel, ve kterém jsou uloženy nově naměřené hodnoty. Délka tohoto pole se liší podle typu senzoru, např. pro akcelerometr obsahuje tři položky odpovídající zrychlení podél osy x, y a z. Druhou metodou rozhraní je `onAccuracyChanged(Sensor, int)`, která informuje o změně přesnosti daného senzoru, viz tabulka 4.3.

Konstanta	**Přesnost**
`SENSOR_STATUS_ACCURACY_HIGH`	Maximální
`SENSOR_STATUS_ACCURACY_MEDIUM`	Průměrná
`SENSOR_STATUS_ACCURACY_LOW`	Nízká, potřeba kalibrace
`SENSOR_STATUS_UNRELIABLE`	Nespolehlivé hodnoty, potřeba kalibrace

Tabulka 4.3: Úrovně přesnosti senzorů na platformě Android.

Odregistraci listeneru je potřeba provést co nejdříve je to možné, aby nedocházelo k plýtvání zdroji systému a energie. Typická je registrace v metodě `onResume()` a odregistrace v metodě `onPause()` životního cyklu aktivity. Ukázka práce se senzory na platformě Android je uvedena v příloze C.

Na závěr této kapitoly uvedeme možnosti a způsob práce s GPS na platformě Android. Vzhledem k rozsahu práce se zaměříme pouze na získání polohy ze satelitů GPS. Bližší informace o zjišťování polohy zařízení, také pomocí mobilních či Wi-Fi sítí, lze nalézt v knihách [36, 39] nebo na webu [24].

Výpis 4.1: Získání instance třídy `LocationManager`.

```
// Předpokládáme, že volání metody getSystemService() je provedeno ve třídě,
// která je potomkem abstraktní třídy Context (např. aktivita nebo služba).
String service = Context.LOCATION_SERVICE;
LocationManager lm = (LocationManager) getSystemService(service);
```

Výpis 4.2: Oprávnění potřebné pro práci s GPS.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Práci s polohou na platformě Android má na starosti správce polohy. Získání jeho instance je uvedeno ve výpisu 4.1. Pro práci s GPS potřebujeme také oprávnění k získávání přesné polohy, viz výpis 4.2. Pomocí správce pak můžeme registrovat a odregistrovat listener (rozhraní `LocationListener`) pro získávání aktualizací o poloze zařízení. Poloha je reprezentována instancí třídy `Location` a obsahuje aktuální zeměpisnou šířku a délku, nadmořskou výšku, rychlost, čas, přesnost polohy a další.

Při registraci listeneru můžeme zvolit, zda si přejeme jednorázové zjištění polohy nebo chceme-li dostávat periodické aktualizace. Ve druhém případě pak musíme zadat mimo jiné také minimální interval (v milisekundách) a minimální vzdálenost (v metrech) mezi aktualizacemi. Dosazení nuly za obě hodnoty má za následek získávání polohy jak nejrychleji to jen jde. V praxi je třeba volit frekvenci aktualizací obezřetně, jelikož pracující GPS modul

[4]Počet nanosekund od startu operačního systému (tzv. uptime).

má velmi negativní dopad na spotřebu energie. Bližší ukázka práce s GPS na platformě Android je uvedena v příloze D.

V podkapitole 4.1 byly mezi senzory snímající okolní prostředí uvedeny také kamera, fotoaparát a mikrofon. Popis způsobu práce s nimi na platformě Android je již nad rámec této práce. Bližší informace na toto téma je možné nalézt např. v knihách [36, 39] nebo na webu [25].

Kapitola 5

Návrh WSN

Součástí práce je vytvoření bezdrátové senzorové sítě s využitím mobilních zařízení. Tato kapitola se zabývá jejím návrhem. Nejprve se zaměříme na architekturu celé sítě (podkapitola 5.1). Poté navrhneme způsob adresace uzlů a přeposílání zpráv napříč sítí (podkapitola 5.2) a vytvoříme komunikační protokol mezi uzly a základnovou stanicí (podkapitola 5.3). Nakonec provedeme návrh klíčových částí dvou aplikací, které budou implementovat funkčnost uzlu sítě (podkapitola 5.4) a základnové stanice (podkapitola 5.5).

Upřesnění zadání

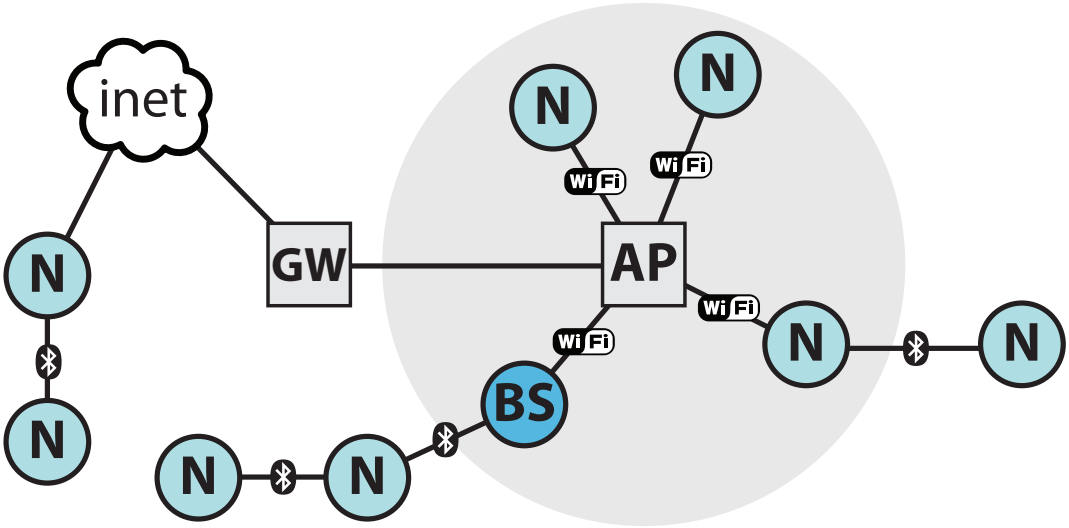
Vytvořená WSN bude v reálném čase monitorovat veličiny ze všech senzorů dostupných na daném mobilním zařízení. Bude také poskytovat data ze systému GPS tak, že frekvenci GPS aktualizací bude možné volit za běhu ze základnové stanice. Ta bude tvořena vizuální aplikací, která bude přehledně zobrazovat monitorovaná data.

Z bezdrátových technologií bude využito Bluetooth, Wi-Fi a případně také mobilní internetové připojení. Uzel bude moci mezi těmito technologiemi dle potřeby automaticky přepínat. Nejdůležitější nastavení uzlu (např. vypnutí a zapnutí) bude možné provádět také vzdáleně prostřednictvím SMS.

5.1 Architektura sítě

Na obrázku 5.1 je znázorněna architektura vytvářené bezdrátové senzorové sítě. Legenda jednotlivých entit je uvedena v tabulce 5.1. Jádrem je síť WLAN tvořená přístupovým bodem, do níž je připojena základnová stanice. Je-li uzel v dosahu přístupového bodu, komunikuje se základnovou stanicí prostřednictvím Wi-Fi. Pokud není v dosahu, nedisponuje Wi-Fi modulem nebo jej má např. z důvodu úspory energie vypnutý, může se spojit se základnovou stanicí pomocí Bluetooth, a to buď přímo, nebo skrze jiný uzel, jenž zajistí přeposílání zpráv. Má-li základnová stanice veřejnou IP adresu, mohou se k ní přes internet a výchozí bránu připojit vzdálené uzly, např. z 3G sítě nebo jiné sítě WLAN.

Jedná se v podstatě o vrstvenou architekturu, viz str. 8 a obrázek 2.1. Jednotlivé vrstvy lze tvořit definováním množiny zařízení, ke kterým se může uzel pomocí Bluetooth připojit. Uzly náležící do jedné vrstvy pak budou mít nastavenu stejnou množinu tzv. *next-hop* uzlů.



Obrázek 5.1: Architektura vytvářené WSN.

Entita	Význam
BS	Základnová stanice
N	Senzorový uzel
AP	Přístupový bod
GW	Výchozí brána
Wi Fi	Spojení přes Wi-Fi
	Spojení přes Bluetooth
inet	Internet

Tabulka 5.1: Legenda entit obrázku 5.1 a 5.2.

5.2 Adresace uzlů a přeposílání zpráv

V této podkapitole se zaměříme na způsob adresování uzlů a princip přeposílání zpráv ve vytvářené síti. Jelikož uzel může komunikovat se základnovou stanicí pomocí více technologií (Wi-Fi, Bluetooth, mobilní data), není vhodné využít jen např. IP adresu ve WLAN síti nebo MAC adresu Bluetooth adaptéru pro adresování uzlů. Jako lepší způsob se jeví vytvoření vrstvy, která z hlediska adresování abstrahuje použitou technologii bezdrátové komunikace. Pro tento účel nacházíme *inspiraci* v již existujícím mechanismu MPLS (Multiprotocol Label Switching, viz [42]). Každému uzlu sítě bude nastaven v rámci WSN *jedi-*

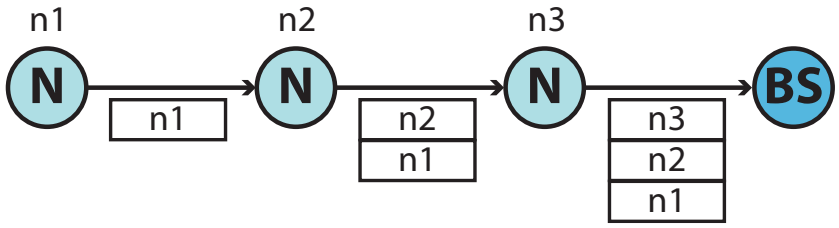
nečný název (tzv. *návěští* či *label*). Ten bude sloužit k adresaci (identifikaci) daného uzlu.

Při návrhu principu přeposílání zpráv vezměme v úvahu směry, jakými budou v síti data přenášena. Většina zpráv bude zasílána z uzlu na základnovou stanici (hlášení aktuálně naměřených hodnot). Budeme ale také potřebovat, byť obecně méně často, zasílat zprávy ze základnové stanice na zvolený uzel (např. při volbě frekvence GPS aktualizací).

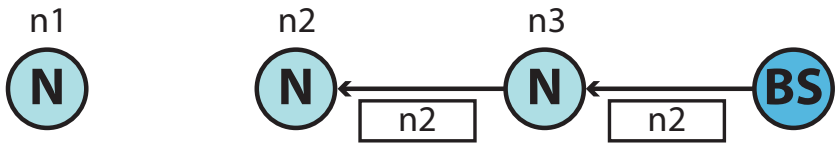
Směr přeposílání v prvním případě je zřejmý, zprávy budou vždy směrovány k základnové stanici. Tento směr je výchozí, nemusíme tedy základnovou stanici nijak explicitně adresovat. Základnová stanice ale bude potřebovat zjistit, ze kterého uzlu zpráva přišla. Proto bude každá zpráva pro základnovou stanici obsahovat zásobník návěští, který odesílatel inicializuje svým jedinečným jménem. Každý uzel, který případně bude zprávu přeposílat, pak přidá na vrchol tohoto zásobníku své vlastní návěští. Základnová stanice tedy bude moci snadno zjistit nejen odesílatele zprávy, ale také všechny uzly, které ji po cestě přeposílaly.

Ve druhém případě, kdy posílá základnová stanice zprávu na zvolený uzel, již zásobník po cestě plnit nepotřebujeme. Uzel totiž nemusí přesně znát cestu zprávy, jelikož směrem k základnové stanici se zprávy posílají implicitně. Stačí tedy, aby základnová stanice inicializovala zásobník jedinečným názvem cílového uzlu a odeslala zprávu směrem, kterým je tento uzel připojen.

Oba výše zmíněné směry toku dat ilustrují obrázky 5.2. Obrázek 5.2a ukazuje příklad zaslání zprávy z uzlu n1 na základnovou stanici. Je vidět, že každý uzel, který zprávu přeposílá (n2 a n3), přidává na vrchol zásobníku návěští své jedinečné jméno. Na obrázku 5.2b je pak uveden příklad komunikace v opačném směru, kdy základnová stanice posílá zprávu pro uzel n2. Zásobník návěští již obsahuje pouze jméno cílového uzlu.



(a) Od uzlu n1 k základnové stanici.



(b) Od základnové stanice k uzlu n2.

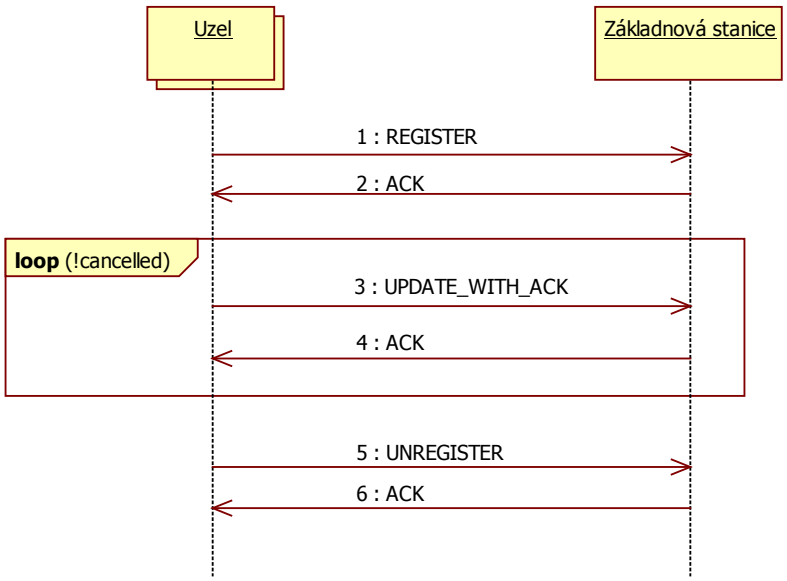
Obrázek 5.2: Přeposílání zpráv pomocí návěští.

5.3 Komunikační protokol

V následující podkapitole navrhneme protokol pro komunikaci mezi uzly a základnovou stanicí. Ze způsobu adresování uzlů a přeposílání zpráv v síti vyplývá (viz podkapitola 5.2),

že potřebuje-li základnová stanice odeslat zprávu uzlu, musí znát jeho jméno a směr, kterým je připojen, není-li připojen k základnové stanici přímo. Komunikaci tedy musí zahájit uzel, který je zde v roli klienta, zatímco základnová stanice je server. Dále vezměme do úvahy, že během komunikace bude moci uzel přepínat mezi různými technologiemi připojení.

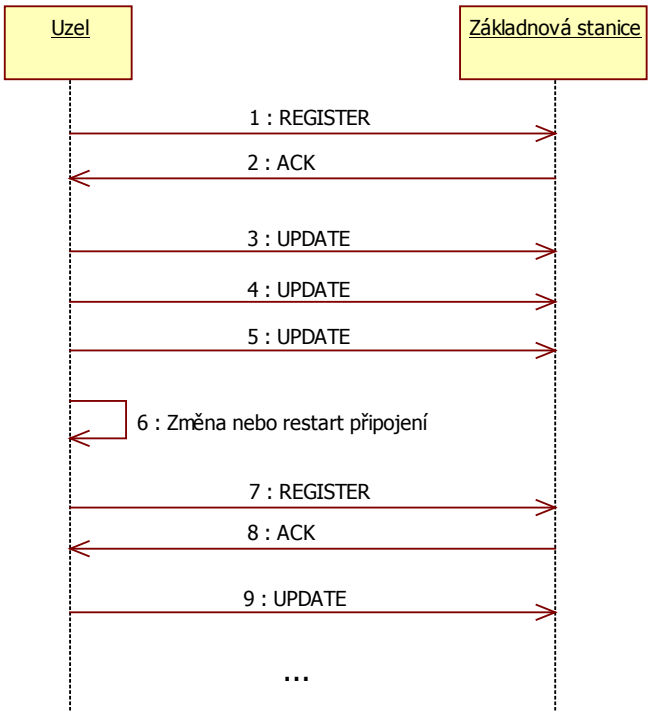
Základem komunikačního protokolu jsou čtyři typy zpráv, viz diagram sekvence na obrázku 5.3. Zpráva `REGISTER` slouží pro *registraci uzlu* na základnové stanici. Touto uzel zahajuje komunikaci. Zpráva nese také informace o uzlu, jako např. model a výrobce zařízení, verzi operačního systému nebo identifikaci technologie, kterou je uzel připojen k základnové stanici nebo next-hop uzlu. Zpráva `UPDATE_WITH_ACK` představuje *aktualizaci naměřených hodnot* a je zasílána periodicky z uzlu na základnovou stanici. Zpráva `UNREGISTER` slouží k *odregistraci uzlu* a ukončuje jeho komunikaci se základnovou stanicí. Poslední typ zprávy je `ACK`, kterým základnová stanice *potvrzuje úspěšné doručení* předchozích typů zpráv.



Obrázek 5.3: Jádro komunikačního protokolu.

Důležitou vlastností protokolu je, že registrační zpráva nemusí být zaslána pouze na začátku komunikace. Pokud dojde k novému připojení již registrovaného uzlu, např. vlivem změny bezdrátové technologie nebo dočasné ztráty signálu, je zaslána nová registrační zpráva. Základnová stanice tak obdrží aktuální informace o způsobu a směru připojení uzlu (muže se změnit cesta přeposílání zpráv). Tato situace je naznačena v diagramu sekvence na obrázku 5.4. Diagram také ukazuje další typ zprávy, kterým je `UPDATE`. Jedná se o totožnou zprávu jako `UPDATE_WITH_ACK` s tím rozdílem, že ji není třeba potvrzovat zprávou `ACK`. To muže být výhodné například při připojení přes větší počet next-hop uzlů, kdy by požadavek na potvrzování způsoboval již nepřijatelné zpoždění.

Jádro komunikačního protokolu (popsané výše) nyní rozšíříme o další potřebné zprávy. Ty slouží pro ovládání uzlu ze základnové stanice. Jedná se o dvě asynchronní zprávy, které mohou být zaslány kdykoliv, je-li uzel zaregistrován. Prvním typem je `GPS_RATE`, jenž má za úkol říci uzlu, jakou minimální frekvenci snímání hodnot z GPS má nastavit. V počátečním stavu budou aktualizace GPS vypnuty z důvodu šetření energie. K jejich



Obrázek 5.4: Vícenásobná registrace uzlu.

zapnutí, a později případně k vypnutí nebo změně frekvence, dojde až ručním nastavením ze základnové stanice pomocí tohoto typu zprávy. Druhým, a již posledním typem zprávy, je `CLOSE_FWD_REQUEST`. Tuto použijeme ve chvíli, kdy chceme ze základnové stanice ručně přerušit komunikaci se zvoleným uzlem. Jedná se v podstatě o odregistraci uzlu ze základnové stanice. Rozdíl je ale v tom, že práce uzlu tímto nekončí, bude se snažit o znovu-připojení k základnové stanici. V praxi by se mělo jednat o přerušení komunikace např. ve smyslu uzavření TCP/IP socketu. Pro přímo připojené uzly žádnou zprávu nepotřebujeme. S těmi, jejichž zprávy jsou přeposílány jinými uzly, ale komunikaci přímo na základnové stanici přerušit nemůžeme. Tím bychom totiž přerušili také spojení s uzly, jež zprávy přeposílají. Místo toho vyšleme tento typ zprávy na uzel, který je prvním next-hop uzlem ve směru od zvoleného uzlu k základnové stanici. Ten pak provede uzavření spojení.

Použití posledního typu zprávy pravděpodobně nebude příliš časté, nicméně může se nám hodit například v situaci, kdy na základnové stanici není spuštěn Bluetooth server. Uzly, které by jinak byly v jeho dosahu, se tak musejí připojit jiným způsobem, např. přes next-hop uzel, který je připojen pomocí Wi-Fi. Po zapnutí Bluetooth serveru pak můžeme využít tento typ zprávy pro přerušení spojení se zmíněnými uzly, čímž jim dáme nový pokus o připojení se k Bluetooth serveru základnové stanice, který již uspěje.

Zprávy typu `GPS_RATE` a `CLOSE_FWD_REQUEST`, které jsou posílány ze základnové stanice na zvolený uzel, nejsou potvrzovány zprávami `ACK`. To proto, že jejich efekt je patrný i bez nich. Například při nastavení nové frekvence GPS aktualizací je informace o její úspěšné změně obsažena ve zprávě typu `UPDATE`, resp. `UPDATE_WITH_ACK`.

Na závěr této podkapitoly ještě uvedeme shrnutí formou tabulky 5.2. Jsou v ní uvedeny všechny typy zpráv, jejich krátký popis, kdy jsou zasílány a jakým směrem. Zápis $\mathrm{N} \rightarrow \mathrm{BS}$

značí posílání zpráv od uzlu směrem k základnové stanici, opačný směr je pak vyjádřen zápisem BS → N. Poslední sloupec tabulky informuje o tom, zda zpráva vyžaduje potvrzení úspěšného doručení (ACK).

Typ zprávy	**Popis**	**Kdy**	**Směr**	**Vyžaduje potvrzení**
REGISTER	Zaregistrování uzlu na základnové stanici	Po ustavení spojení	N → BS	Ano
UNREGISTER	Odregistrování uzlu ze základnové stanice	Při ukončení práce uzlu	N → BS	Ano
UPDATE_WITH_ACK	Aktualizace hodnot	Nastavitelný interval	N → BS	Ano
UPDATE	Aktualizace hodnot	Nastavitelný interval	N → BS	Ne
GPS_RATE	Nastavení frekvence snímání dat z GPS	Ručně	BS → N	Ne
CLOSE_FWD_REQUEST	Přerušení komunikace s uzlem	Ručně	BS → N	Ne
ACK	Potvrzení úspěšného doručení zprávy	Po zprávách vyžadujících potvrzení	BS → N	—

Tabulka 5.2: Přehled všech typů zpráv ve vytvářené WSN.

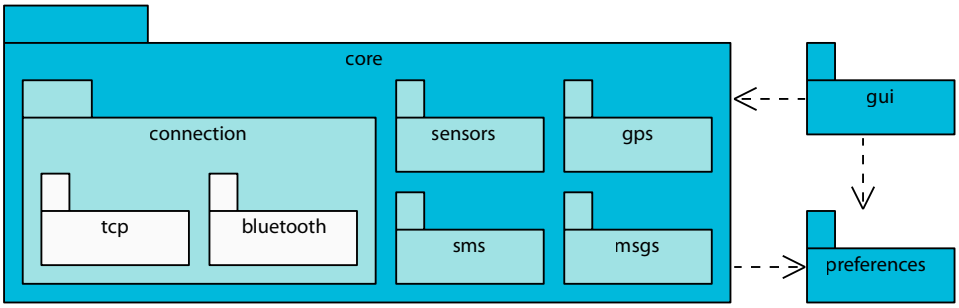
5.4 Návrh uzlu

V této podkapitole se zaměříme na návrh aplikace, která bude implementovat funkčnost bezdrátového senzorového uzlu vytvářené WSN. Nejprve si rozdělíme aplikaci do vrstev a popíšeme funkčnost každé z nich. Poté si podrobněji navrhneme způsob fungování bezdrátové komunikace uzlu s okolím.

Diagram balíčků na obrázku 5.5 zobrazuje hierarchii vrstev aplikace. Vrstva *gui* bude implementovat grafické uživatelské rozhraní, pomocí kterého bude možné spouštět nebo ukončovat práci uzlu a provádět jeho nastavení. Vrstva *preferences* pak bude toto nastavení uchovávat a spravovat.

Nejdůležitější je zde vrstva *core*, která v sobě zapouzdřuje všechny klíčové součásti aplikace. Balík *sensors* bude sloužit k měření veličin z dostupných senzorů. S tím souvisí také balík *gps*, který bude provádět snímání dat z GPS. Dále balík *sms* bude obsahovat Broadcast Receiver pro příjem a zpracování konfiguračních SMS zpráv. Zbývají ještě balíky související s komunikací. Balík *msgs* bude implementovat a zapouzdřovat práci s různými typy zpráv zasílaných v rámci komunikace (viz podkapitoly 5.2 a 5.3). Balík *connection* bude abstrahovat použité technologie pro síťové spojení a přepínání mezi nimi. Konkrétně se bude možné spojit pomocí TCP/IP nebo Bluetooth, viz balíky *tcp* a *bluetooth*.

V diagramu na obrázku 5.6 je zobrazen návrh způsobu fungování bezdrátové komunikace uzlu. Nechť jsou data měřená senzory a GPS modulem vzorkována v nastaveném intervalu. V každém intervalu je pak generována *nová odchozí zpráva* typu UPDATE nebo UPDATE_WITH_ACK. Ta je poté vložena do fronty k odeslání, viz *outQueue*. Fronta má ne-



Obrázek 5.5: Diagram balíčků uzlu.

blokující zápis a blokující čtení. Je-li plná, přidání nové zprávy způsobí zahození té na jejím začátku. Síť má totiž za úkol monitorovat veličiny v reálném čase a první položka fronty obsahuje nejstarší hodnoty. Fronta *outQueue* zde slouží k abstrakci aktuálně použité technologie připojení *směrem k základnové stanici*, odesílatel (vyšší vrstva aplikace) tak nemusí znát aktuální stav a jiné detaily spojení.

Obdélník *TCP* na obrázku představuje připojení pomocí TCP/IP, které je využito pro Wi-Fi nebo mobilní data. Obdélník *BT* pak znázorňuje spojení přes Bluetooth. K základnové stanici (viz BS) může být uzel připojen oběma způsoby, aktivní je ale vždy maximálně jeden. Alternativně se může uzel spojit s některým z jeho množiny next-hop uzlů (viz N vpravo), a to jedině prostřednictvím Bluetooth.

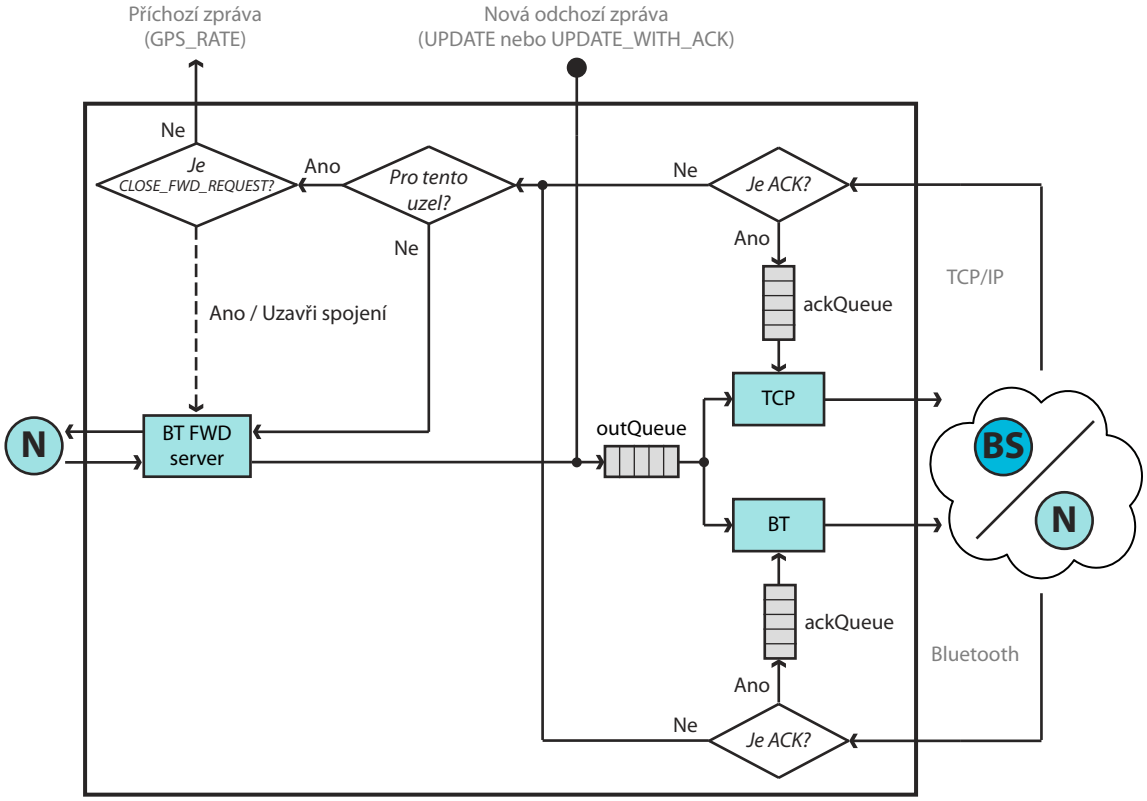
Jelikož také potřebujeme asynchronně přijímat zprávy ze základnové stanice, mají oba typy připojení speciální vlákno, které má toto na starost. Ne vždy se ale musí jednat o zprávu pro uzel, kterou je nutné propagovat do vyšších vrstev aplikace. Tuto výjimku tvoří zprávy typu `ACK`, které je třeba doručit tam, odkud byla potvrzovaná zpráva odeslána. Pro synchronizaci mezi přijímačem a vysílačem jsou zde fronty potvrzení, viz dvě *ackQueue*. Mají neblokující zápis a blokující čtení. Je-li fronta potvrzení plná, je nově vkládané `ACK` zahozeno. Po odeslání zprávy vyžadující potvrzení musí vysílač (TCP nebo BT) počkat, dokud se v příslušné ackQueue neobjeví odpovídající `ACK`. Případně pak po vypršení časového limitu odeslání zopakovat.

Zdali je obdržená zpráva vložena do ackQueue nebo propagována dále, záleží na výsledku vyhodnocení rozhodovacího uzlu „*Je ACK?*“. Všimněme si, že postačující podmínkou je zde pouze rozpoznání typu zprávy. Nedíváme se na to, kam zpráva směřuje. Toto si můžeme dovolit díky zvolené strategii potvrzování zpráv, která není v diagramu přímo zobrazena. Potvrzení totiž nebudeme nikdy přeposílat. Jinými slovy, je-li uzel připojen k základnové stanici přes některý z jeho next-hop uzlů, je to právě tento next-hop uzel, který mu bude odeslané zprávy potvrzovat. Tím dává najevo, že převzal zodpovědnost za doručení zprávy. Poté ji stejným způsobem přepošle dál směrem k základnové stanici.

Funkci next-hop uzlu poskytuje v diagramu obdélník *BT FWD server*. Jedná se o Bluetooth server, ke kterému se může připojit jiný uzel (viz N vlevo). Zprávy, které server obdrží, vkládá do fronty odchozích zpráv. Vyžaduje-li zpráva potvrzení, odešle zpět `ACK`.

Pokud zpráva obdržená ze směru základnové stanice není `ACK`, je dále propagována do rozhodovacího uzlu „*Pro tento uzel?*“. Ten se již podívá na to, jaký je cílový uzel zprávy. Není-li zpráva určena pro tento uzel, je předána do BT FWD serveru, který ji přepošle směrem k jejímu cíli. Naopak, je-li příchozí zpráva určena tomuto uzlu, je předána do rozhodovacího uzlu „*Je CLOSE_FWD_REQUEST?*“. Ten se znovu podívá na typ zprávy.

Jedná-li se o `CLOSE_FWD_REQUEST`, znamená to, že základnová stanice požaduje ukončení spojení s uzlem, pro něhož je tento prvním next-hop uzlem. V takovém případě je vyslán příkaz BT FWD serveru, aby uzavřel spojení s právě připojeným uzlem. V diagramu je toto naznačeno přerušovanou šipkou. Pokud je typ zprávy jiný, je propagována do vyšší vrstvy aplikace jako nově *příchozí zpráva*. Aktuálně již může jít pouze o zprávu typu `GPS_RATE`, v budoucnu je ale možné komunikační protokol rozšířit o nové příkazy. Jejich příchod pak bude také zpracováván výše popsaným způsobem.



Obrázek 5.6: Jádro bezdrátové komunikace uzlu.

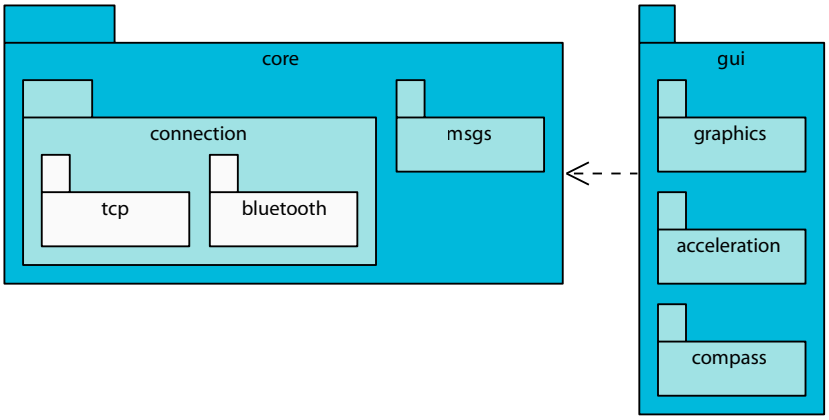
5.5 Návrh základnové stanice

V této podkapitole se zaměříme na návrh aplikace, která bude implementovat funkčnost základnové stanice ve vytvářené WSN. Nejprve si rozdělíme aplikaci do vrstev a popíšeme funkčnost každé z nich. Poté navrhneme způsob fungování bezdrátové komunikace základnové stanice s uzly.

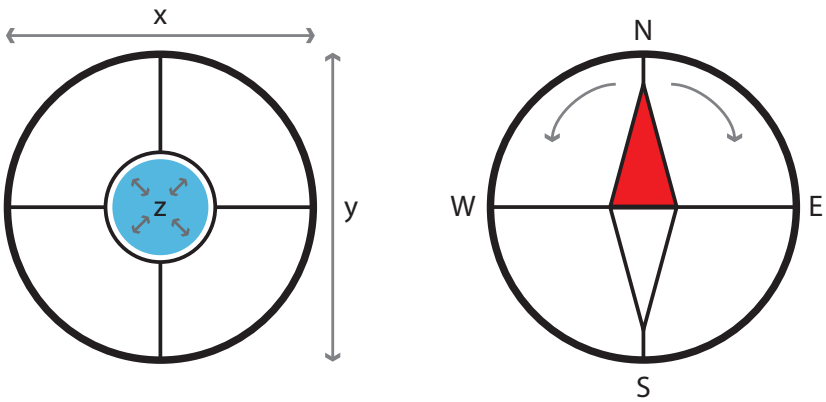
Diagram balíčků na obrázku 5.7 zobrazuje hierarchii vrstev aplikace. Vrstva *gui* bude implementovat grafické uživatelské rozhraní, které je zde velmi důležité, neboť bude sloužit k vizualizaci naměřených dat. Balík *graphics* bude obsahovat zdroje ve formě rastrové grafiky (např. ikony). Dále pro vizualizaci hodnot akcelerometru si vytvoříme vlastní grafickou komponentu. Ta bude zobrazovat hodnoty zrychlení podél všech tří souřadných os (x, y a z). Na obrázku 5.8a je uveden její návrh. Zrychlení na ose x a y je vizualizováno pohy-

bem kruhu v horizontálním a vertikálním směru. Akcelerace podél osy z je pak znázorněna zvětšováním a zmenšováním jeho poloměru. Práci s komponentou bude zapouzdřovat balík *acceleration*. Pro názornější vizualizaci natočení zařízení bude, kromě zobrazení úhlů podél všech tři os (viz Eulerovy úhly na str. 33), vytvořena také vlastní grafická komponenta kompasu. Na obrázku 5.8b je zobrazen její návrh. Obsahuje klasickou střelku, jež rotací kolem svého středu ukazuje, na kterou ze světových stran směřuje horní hrana zařízení.

Důležitou součástí je také vrstva *core*, která má na starosti především komunikaci s uzly senzorové sítě. Balík *msgs* bude implementovat a zapouzdřovat práci s různými typy zpráv. Balík *connection* bude abstrahovat použité technologie pro připojení uzlů, čímž umožní vyšším vrstvám aplikace pracovat s připojenými uzly jednotným způsobem. Obsahuje také TCP/IP a Bluetooth servery naslouchající na připojení nových uzlů sítě, viz balíky *tcp* a *bluetooth*.



Obrázek 5.7: Diagram balíčků základnové stanice.



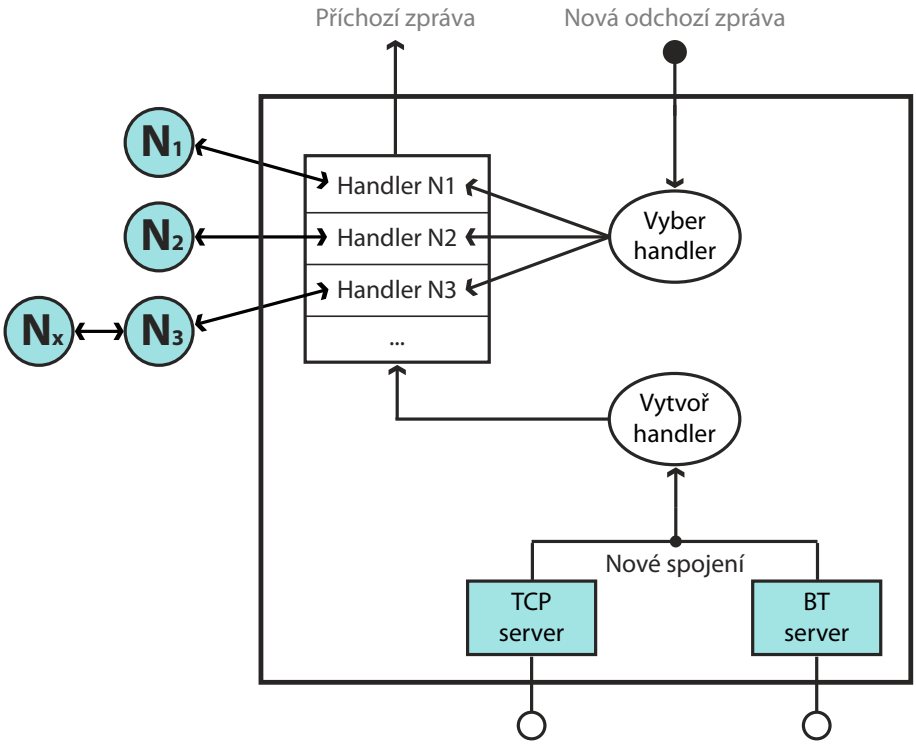
(a) Zrychlení. (b) Kompas.

Obrázek 5.8: Návrh vlastních komponent GUI.

V diagramu na obrázku 5.9 je zobrazen návrh způsobu fungování bezdrátové komunikace základnové stanice. Práce začíná spuštěním serverů. Obdélník *TCP server* reprezentuje

TCP/IP server a obdélník *BT server* představuje Bluetooth server. Oba naslouchají na příchozí spojení. Připojí-li se nový uzel k některému ze serverů, je *vytvořen handler*, který dále spravuje komunikaci s uzlem a abstrahuje použitou technologii spojení. Handler je poté přidán do seznamu handlerů všech aktivních spojení. V diagramu například vidíme, že *Handler N1* slouží pro komunikaci s uzlem N_1 a *Handler N2* komunikuje s uzlem N_2. Dále pak *Handler N3* komunikuje nejen s uzlem N_3, ale potažmo také s uzlem N_x, jelikož N_3 zprostředkovává přeposílání. Obdrží-li kterýkoliv z aktivních handlerů zprávu od uzlu (např. `REGISTER` nebo `UPDATE`), je propagována do vyšší vrstvy aplikace jako *příchozí zpráva*. Daný handler také odešle zpět `ACK`, vyžaduje-li zpráva potvrzení.

Pokud základnová stanice potřebuje zaslat uzlu asynchronní zprávu (např. `GPS_RATE`), je vytvořena *nová odchozí zpráva*. Podle cílového uzlu zprávy je *vybrán handler*, který zajistí její odeslání. Zde je důležité, aby základnová stanice znala cesty připojení všech uzlů. Pokud totiž není uzel připojen k základnové stanici přímo, je nutné vybrat handler, který odešle zprávu správným směrem.



Obrázek 5.9: Jádro bezdrátové komunikace základnové stanice.

Kapitola 6

Implementace WSN

Tato kapitola se zabývá implementací vytvářené bezdrátové senzorové sítě. Nejprve popíšeme způsob implementace odesílání a příjmu zpráv v síti (podkapitola 6.1). Dále se blíže zaměříme na popis implementace dvou aplikací, které zajišťují funkčnost uzlu sítě (podkapitola 6.2) a základnové stanice (podkapitola 6.3). Nakonec zmíníme alternativní architekturu sítě (podkapitola 6.4) a provedeme testování vytvořené WSN (podkapitola 6.5).

Platforma pro implementaci uzlů

Na základě průzkumu možností využití mobilních zařízení jako bezdrátových uzlů a senzorových zařízení bylo zjištěno (viz kapitola 3 a 4), že platforma *Android* je pro tento účel nejvýhodnější. Její aplikační rozhraní umožňuje více možností manipulace s bezdrátovými technologiemi než konkurenční platformy iOS a Windows Phone. Také z hlediska měření fyzikálních veličin podporuje nejvíce typů senzorů. Další výhodou Androidu je jeho rozšíření a značný počet zařízení různých cenových hladin.

Jedním z úskalí platformy Android je jeho fragmentace (roztříštěnost) – odlišné verze operačního systému na spoustě zařízení různých výrobců. Čím dál více se ale tato situace zlepšuje.[1] Pro implementaci byla zvolena podpora *verze 2.3.3* (API 10) a vyšší, a to především z následujících důvodů:

- Byla přidána podpora tzv. insecure Bluetooth socketů, které umožňují navázat spojení bez nutnosti párovat zařízení.
- Byla přidána podpora nových typů senzorů, viz tabulka 4.1 na str. 36.
- Dle statistik ze 14 denního období končícího 2. dubna 2013 pokryjeme zhruba 94 % stávajících zařízení, což je více než dostatečné množství (viz [21]).[2]
- Zvažujeme-li nákup nových zařízení, stěží bychom hledali takové, kde je přítomna verze Androidu nižší než 2.3.x.[3]

Platforma pro implementaci základnové stanice

Základnová stanice bude vytvořena jako desktopová aplikace. Pro její implementaci byla zvolena platforma *Java Standard Edition 7*. Důvodem této volby je přenositelnost vytvořené

[1] Statistiky lze sledovat na webu [21].
[2] Statistika je založená na přístupech do obchodu Google Play.
[3] Zjištěno průzkumem nabídky internetových obchodů Sunnysoft a Alza.cz dne 28. dubna 2013.

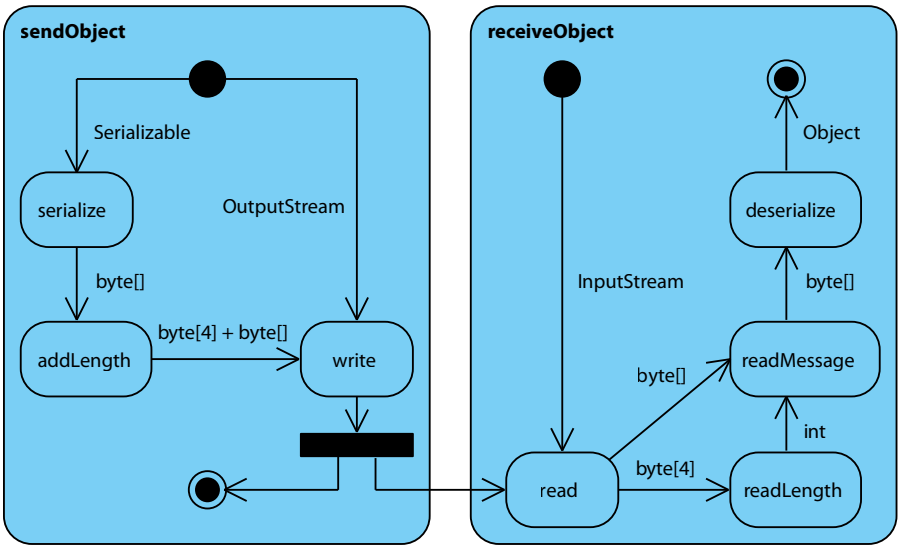
aplikace a využití stejného programovacího jazyka pro uzel i základnovou stanici. K vytvoření grafického uživatelského rozhraní využijeme knihovnu *Swing*.

6.1 Odesílání a příjem zpráv

V této podkapitole se zaměříme na implementaci odesílání a příjmu zpráv. Ať už se jedná o spojení pomocí TCP/IP nebo Bluetooth, v jazyce Java probíhá výměna dat na nejnižší úrovni vždy pomocí input a output streamů bajtů, resp. standardních tříd `InputStream` a `OutputStream` jazyka Java. Proto můžeme využít na uzlu i základnové stanici shodný algoritmus, a to pro komunikaci přes TCP/IP i Bluetooth. Jelikož se práce snaží zachovat aplikační nezávislost pro snadnou modifikaci do budoucna, vytvoříme algoritmus, který umožní jednotně odesílat i přijímat všechny stávající i budoucí typy zpráv.

Prostředkem, který nám toto umožní implementovat, je serializace a deserializace objektů v jazyce Java. Bude tedy možné odesílat a přijímat libovolný serializovatelný objekt. Požadavek na serializovatelnost objektu je přitom snadno splnitelný, stačí v definici příslušné třídy uvést, že implementuje rozhraní `Serializable`.

Na obrázku 6.1 je uveden diagram aktivity algoritmu pro odeslání a příjem zprávy. Na straně odesílatele je použita metoda *sendObject*. Jejím vstupem je serializovatelný objekt (zpráva k odeslání) a output stream získaný po navázání spojení. Nejprve dojde k serializaci objektu na pole bajtů (datový typ `byte[]`), viz akce *serialize*. Abychom na straně příjemce poznali, kde přijímaný objekt končí a případně začíná nový, přidáme na začátek pole 4 bajty, které reprezentují celé číslo typu `int` nesoucí počet bajtů serializovaného objektu, viz akce *addLength*. Poté již můžeme akcí *write* zapsat výsledné pole bajtů do output streamu, čímž provedeme jeho odeslání. Metoda je neblokující a tímto končí.



Obrázek 6.1: Diagram aktivity algoritmu pro odeslání a příjem zprávy.

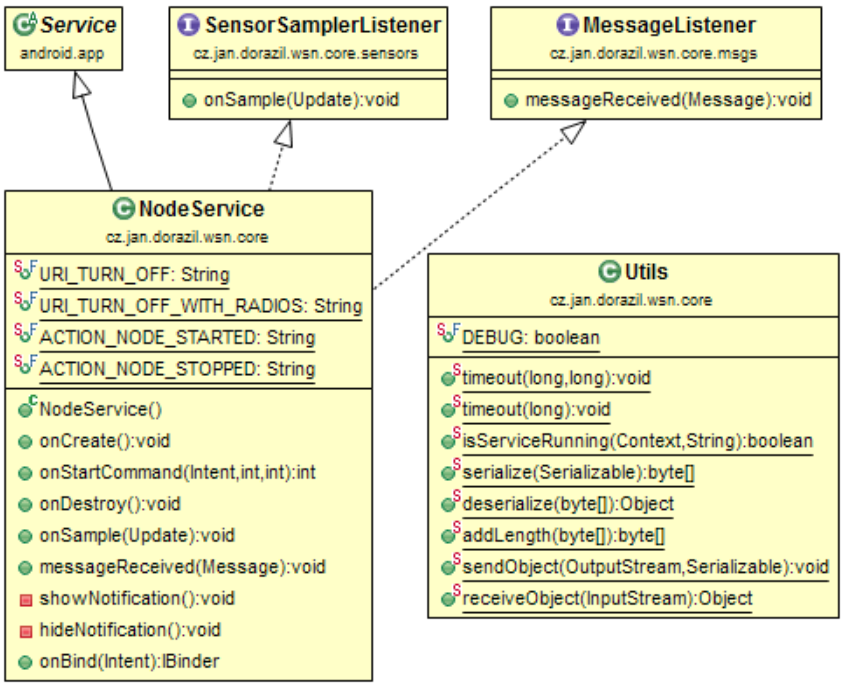
Na straně příjemce je pro příjem zprávy použita blokující metoda *receiveObject*. Jejím vstupem je input stream získaný po navázání spojení. Metoda čeká, dokud v něm nejsou přítomny data ke čtení. Jakmile jsou data k dispozici, jsou nejprve načteny první 4 bajty,

které interpretujeme jako délku nadcházející zprávy v bajtech, viz akce *readLength*. Následně můžeme z input streamu načíst právě tolik bajtů, kolik je velikost zprávy, viz akce *readMessage*. Poté akcí *deserialize* provedeme deserializaci přijatého pole bajtu na původní objekt. Ten je pak předán jako návratová hodnota, čímž metoda končí. Pro efektivní práci s poli bajtů je v implementaci využita třída `ByteBuffer`.

Výhodou výše popsaného řešení je dobrá úroveň zapouzdření a možnost pozdějšího rozšíření typů zpráv bez nutnosti modifikovat tuto vrstvu aplikace. Také je možné libovolně upravovat, přidávat či odebírat atributy tříd reprezentujících zprávy, opět bez nutnosti jakékoliv změny tohoto kódu. Právě proto je toto řešení vhodné pro vytvářenou WSN. Nevýhodou zvoleného postupu mohou být v podstatě konstantní velikosti zpráv. Vždy se posílá celý objekt a nebere se v úvahu, zda se např. změnily všechny nebo jen některé hodnoty od předchozí aktualizace naměřených dat. Pro účely této sítě, která monitoruje veličiny v reálném čase, to ale příliš nevadí. Většina hodnot se totiž mezi aktualizacemi opravdu mění, a to díky vyšší frekvenci snímání dat ze senzorů, než je interval potřebný pro vizualizaci v reálném čase.

6.2 Implementace uzlu

V následující podkapitole se budeme zabývat popisem implementace aplikace na platformě Android, která zajišťuje funkčnost uzlu vytvářené WSN. Postupně budou uvedeny a popsány diagramy tříd klíčových vrstev aplikace. Jejich hierarchie je zobrazena ve formě diagramu balíčků na obrázku 5.5 v podkapitole 5.4. Pro větší přehlednost a skrytí některých implementačních detailů, jejichž popis není již v rozsahu této práce, nebudou diagramy tříd obsahovat vždy úplně všechny atributy a metody. Nakonec popíšeme GUI aplikace.



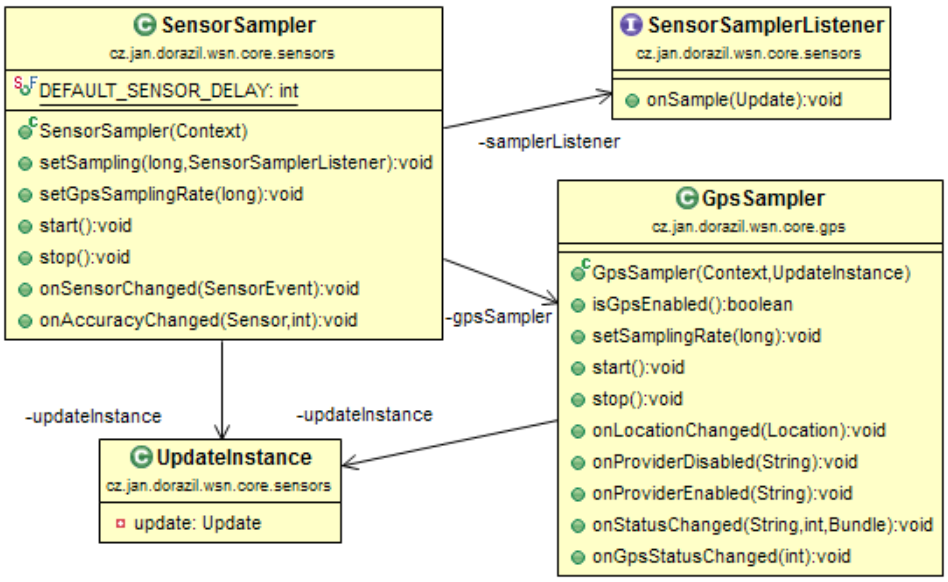
Obrázek 6.2: Diagram tříd na nejvyšší úrovni balíku core.

Vrstva core

Na obrázku 6.2 je uveden diagram tříd na nejvyšší úrovni balíku core. Velmi důležitou roli zde hraje třída `NodeService`, která implementuje službu platformy Android. Jejím spuštěním a zastavením začíná a končí práce uzlu. Ve stavové liště je zobrazována notifikace informující o jejím běhu. Třída má dva klíčové privátní atributy, kterými jsou `sensorSampler` a `connection`. První zajišťuje měření veličin z dostupných senzorů a snímání dat z GPS. Výsledné hodnoty pak ve formě instance třídy `Update` periodicky předává do služby v uživatelem nastaveném intervalu. K předávání slouží rozhraní `SensorSamplerListener` s metodou `onSample(Update)`.

Zde přichází na řadu druhý zmíněný atribut, a sice `connection`. Ten abstrahuje a zapouzdřuje spojení se základnovou stanicí. Jeho prostřednictvím služba odesílá naměřené hodnoty, bez ohledu na aktuální stav spojení a použitou technologii. Zároveň, pokud základnová stanice pošle asynchronní zprávu pro uzel (např. `GPS_RATE`), je příchozí zpráva propagována do služby pomocí rozhraní `MessageListener` a jeho metody `messageReceived(Message)`. V případě již zmíněné zprávy `GPS_RATE` pak služba nastaví objektu `sensorSampler` novou frekvenci snímání dat z GPS.

Ve vrstvě je dále přítomna třída `Utils`, která obsahuje důležité pomocné statické metody, jež jsou využívány napříč celým balíkem core. Především je zde umístěna implementace odesílání a přijímání zpráv popsaná v podkapitole 6.1.



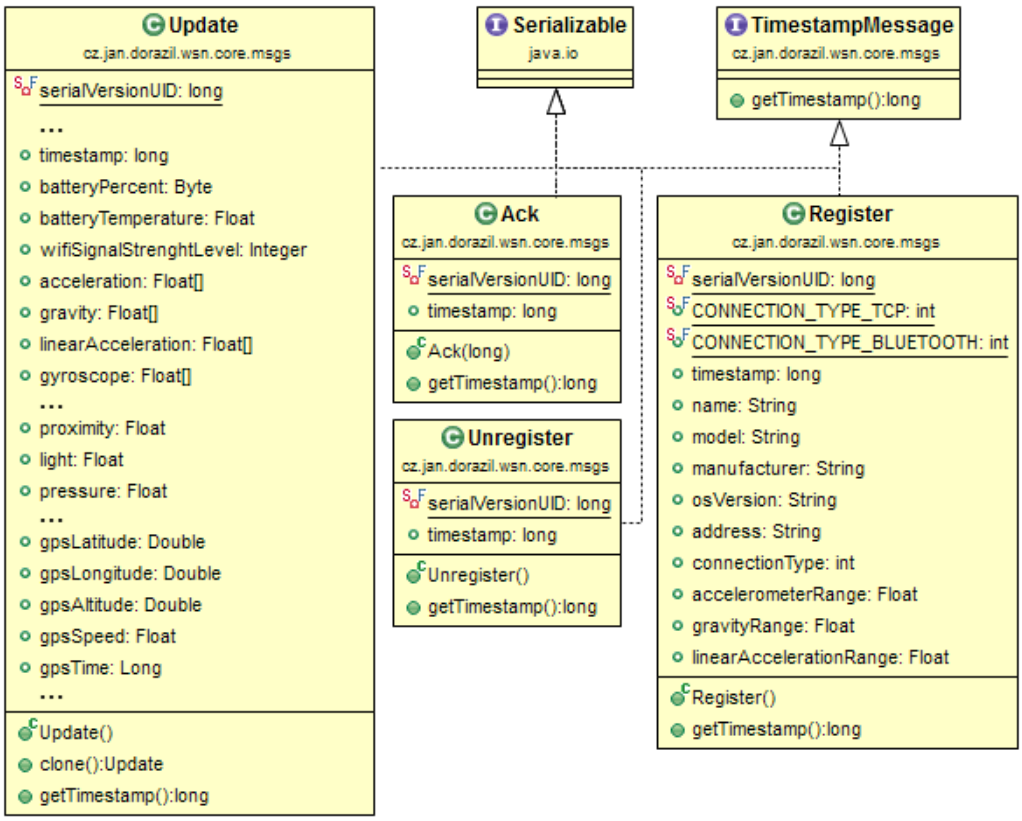
Obrázek 6.3: Diagram tříd v balících sensors a gps.

Vrstvy sensors a gps

Na obrázku 6.3 je uveden diagram tříd balíků sensors a gps. Klíčové jsou zde třídy `SensorSampler` a `GpsSampler`. Třída `SensorSampler` provádí vzorkování hodnot měřených všemi senzory, které jsou na daném zařízení dostupné (viz tabulka 4.1). Navíc měří sílu Wi-Fi signálu a stav a teplotu akumulátoru. Navzorkované hodnoty pak propaguje do vyšší

vrstvy aplikace přes již zmíněné rozhraní `SensorSamplerListener`. Pro zajištění synchronizace mezi získáváním dat ze senzorů a jejich vzorkováním, je použita třída `UpdateInstance`. Ta obsahuje jednu privátní instanci třídy `Update`, se kterou je možné pracovat pouze metodami, které jsou synchronizovány pomocí monitorů jazyka Java. Hodnoty senzorů jsou snímány s přibližnou frekvencí `DEFAULT_SENSOR_DELAY` nastavenou na hodnotu `SENSOR_DELAY_UI`, což je dle tabulky 4.2 cca 60 ms. Reálná rychlost snímání se ale liší podle konkrétního zařízení a potřeb systému, typicky je rychlejší.

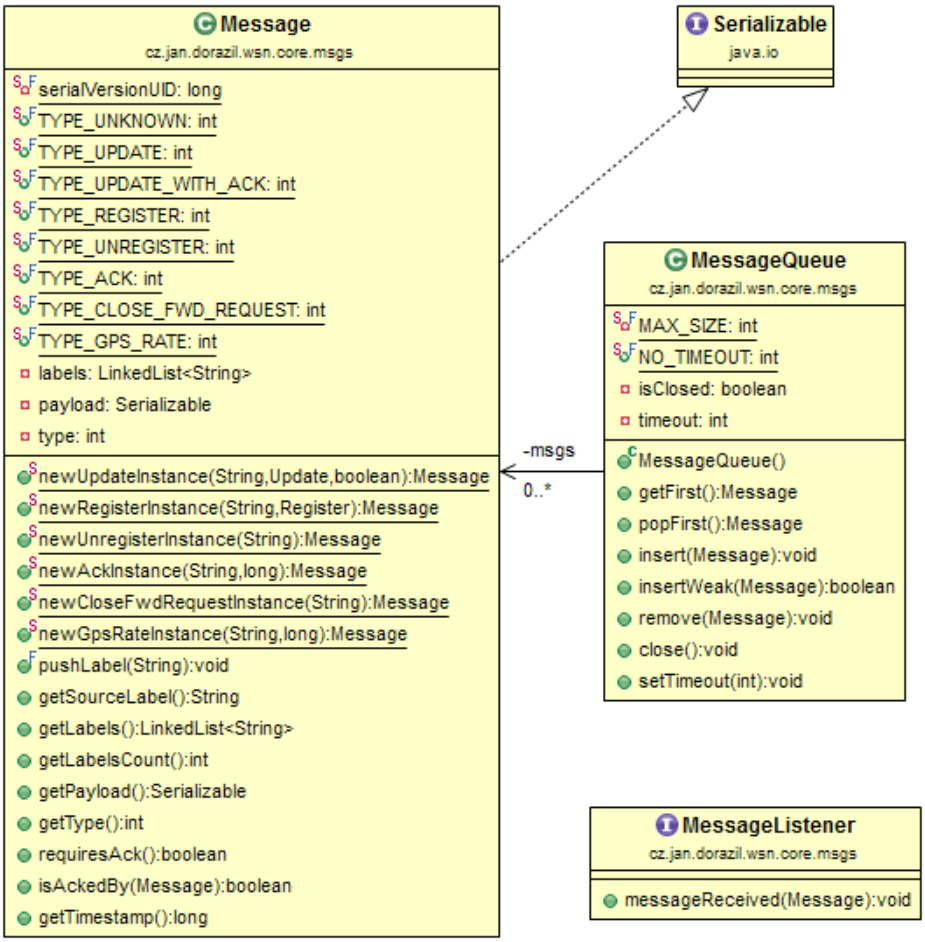
Třída `SensorSampler` obsahuje také objekt třídy `GpsSampler`. Ta slouží pro získávání dat z GPS, které ukládá do sdílené instance třídy `UpdateInstance`. Data obsahují zeměpisnou šířku a délku, nadmořskou výšku, rychlost a čas. Metodou `setSamplingRate(long)` je možné nastavit minimální prodlevu mezi měřeními v milisekundách. Jak je vidět, třída `SensorSampler` zapouzdřuje nejen práci se senzory, ale potažmo také s GPS.



Obrázek 6.4: Diagram tříd reprezentujících typy zpráv.

Vrstva msgs

Balík msgs obsahuje třídy pro reprezentaci zpráv a práci s nimi. Na obrázku 6.4 je uveden diagram tříd, které představují obsah zasílaných zpráv. Všechny jsou serializovatelné a implementují rozhraní `TimestampMessage` pro získání časového razítka zprávy bez ohledu na její obsah. To slouží k potvrzování tak, že instance třídy `Ack` nese časové razítko potvrzované zprávy. Zpráva `Register`, sloužící k registraci uzlu na základnové stanici, nese



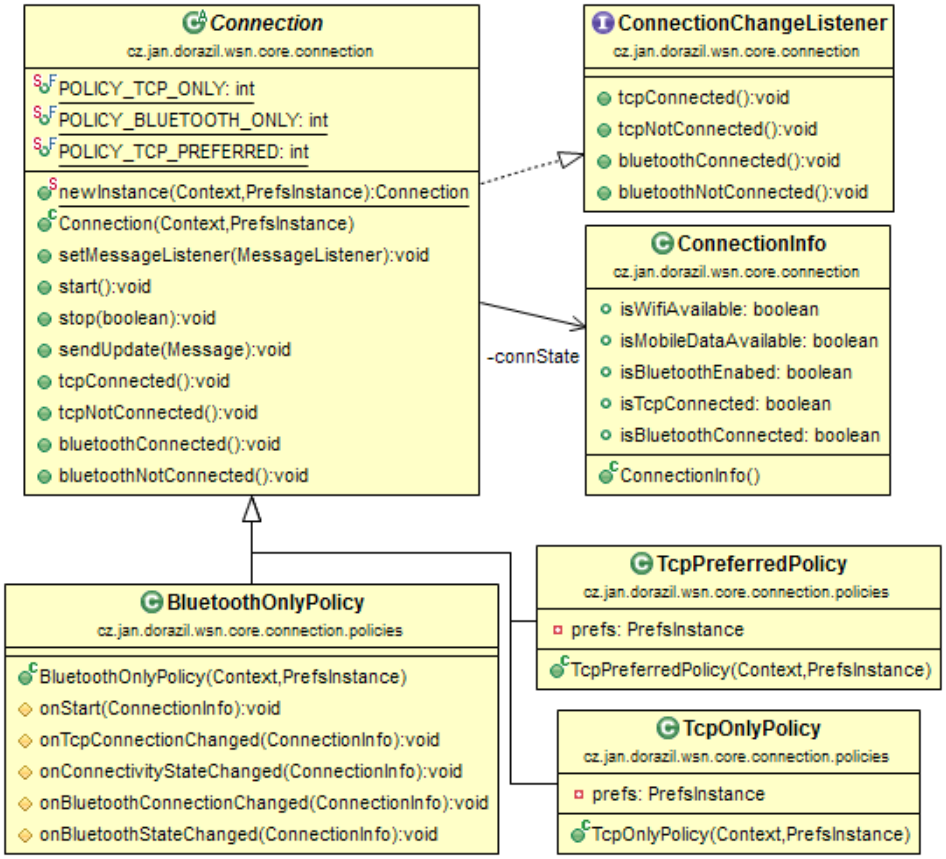
Obrázek 6.5: Diagram tříd pro práci se zprávami.

informace o uzlu. Nejdůležitější je zde zpráva `Update`, která obsahuje všechny aktuálně naměřené hodnoty ze senzorů a GPS. Kromě toho informuje také např. o síle Wi-Fi signálu či aktuálním stavu akumulátoru. Výčet jejích atributů je v diagramu zkrácen, jelikož je jich větší počet. Zpráva `Unregister`, pro odregistraci uzlu ze základnové stanice, pak nese pouze časové razítko, aby mohlo být její úspěšné doručení potvrzeno.

Na obrázku 6.5 je uveden diagram tříd pro práci se zprávami. Třída `Message` představuje serializovatelnou zprávu, která je zasílána napříč sítí. Zapouzdřuje všechny typy zpráv definované v komunikačním protokolu (viz podkapitola 5.3) a přidává navíc mechanismus pro adresování uzlů a přeposílání zpráv (viz podkapitola 5.2). Instance konkrétních typů zpráv jsou tvořeny pomocí statických továrních metod. Zpráva pak obsahuje zásobník návěští uzlů (atribut `labels`), vlastní obsah zprávy (atribut `payload`) a identifikaci jejího typu (atribut `type`). Za zmínku ještě stojí metoda `isAckedBy(Message)`, která slouží k ověření, zda obdržené `ACK` potvrzuje danou zprávu.

Třída `MessageQueue` implementuje frontu zpráv. Má blokující čtení a neblokující zápis. Chování při vkládání zprávy do zaplněné fronty se liší dle použité metody. Metoda `insert(Message)` odstraní zprávu ze začátku fronty, je využita pro frontu zpráv k odeslání (viz outQueue v diagramu na obrázku 5.6). Metoda `insertWeak(Message)` se nechová tak razantně, místo toho zprávu do fronty nepřidá. Úspěch vkládání signalizuje svou

návratovou hodnotou. Je použita pro fronty příchozích potvrzení (viz ackQueue v diagramu na obrázku 5.6). Jak již bylo řečeno, čtení zprávy z fronty je blokující operace. Můžeme ale nastavit časový limit čekání. Ten je využit u front příchozích potvrzení. Součástí balíku msgs je také rozhraní `MessageListener`, které napříč aplikací slouží k propagaci přijatých zpráv do vyšších vrstev aplikace.



Obrázek 6.6: Diagram tříd pro ovládání spojení.

Vrstva connection

Na obrázku 6.6 je uveden diagram tříd na nejvyšší úrovní balíku connection. Slouží pro ovládání bezdrátového spojení. Jádrem je zde abstraktní třída `Connection`. Od ní jsou pak odvozeny konkrétní politiky spojení (tzn. různé přístupy, viz popis politik v dalších odstavcích), které již pouze implementují její abstraktní metody. Ty jsou v diagramu tříd uvedeny pouze u třídy `BluetoothOnlyPolicy`. Mají za úkol informovat politiku o změně stavu konektivity nebo spojení se základnovou stanicí. Jsou vždy volány z nadtřídy `Connection`, která detekci těchto změn zapouzdřuje.

Stav konektivity je zjišťován pro platformu Android standardním způsobem, a sice odchytáváním příslušných systémových akcí Broadcast Receiverem (viz příloha B). Aktuální stav spojení se základnovou stanicí pak prostřednictvím rozhraní `Connection-ChangeListener` poskytují třídy, jež implementují spojení přes TCP/IP a Bluetooth.

Třída `ConnectionInfo` tyto informace uchovává a slouží k jejich předání do abstraktních metod implementovaných v politikách. Politika tak může zjistit, zda je dostupné připojení přes Wi-Fi či mobilní data, zda je zapnuto Bluetooth a zda je uzel spojen se základnovou stanicí přes TCP/IP nebo Bluetooth.

Například metoda `onBluetoothConnectionChanged(ConnectionInfo)` je volána při navázání nebo přerušení Bluetooth spojení se základnovou stanicí nebo next-hop uzlem. Politika pak může zareagovat podle své definice např. zastavením serveru pro přeposílání zpráv od jiných uzlů v případě, že není aktivní žádné spojení ve směru k základnové stanici. Všechny potřebné metody pro ovládání spojení jsou již implementovány ve třídě `Connection`, politiky již jen podle aktuálního stavu reagují jejich voláním.

Výše popsaná struktura je navržena tak, aby bylo v budoucnu možné co nejvíce jednoduchým způsobem přidat nové nebo upravit stávající politiky. Pro přidání politiky stačí pouze vytvořit třídu odvozenou od `Connection` a implementovat několik metod, do kterých se podle vytvářené politiky umístí reakce na události týkající se konektivity a spojení. Máme tři politiky spojení, mezi kterými může uživatel přepínat:

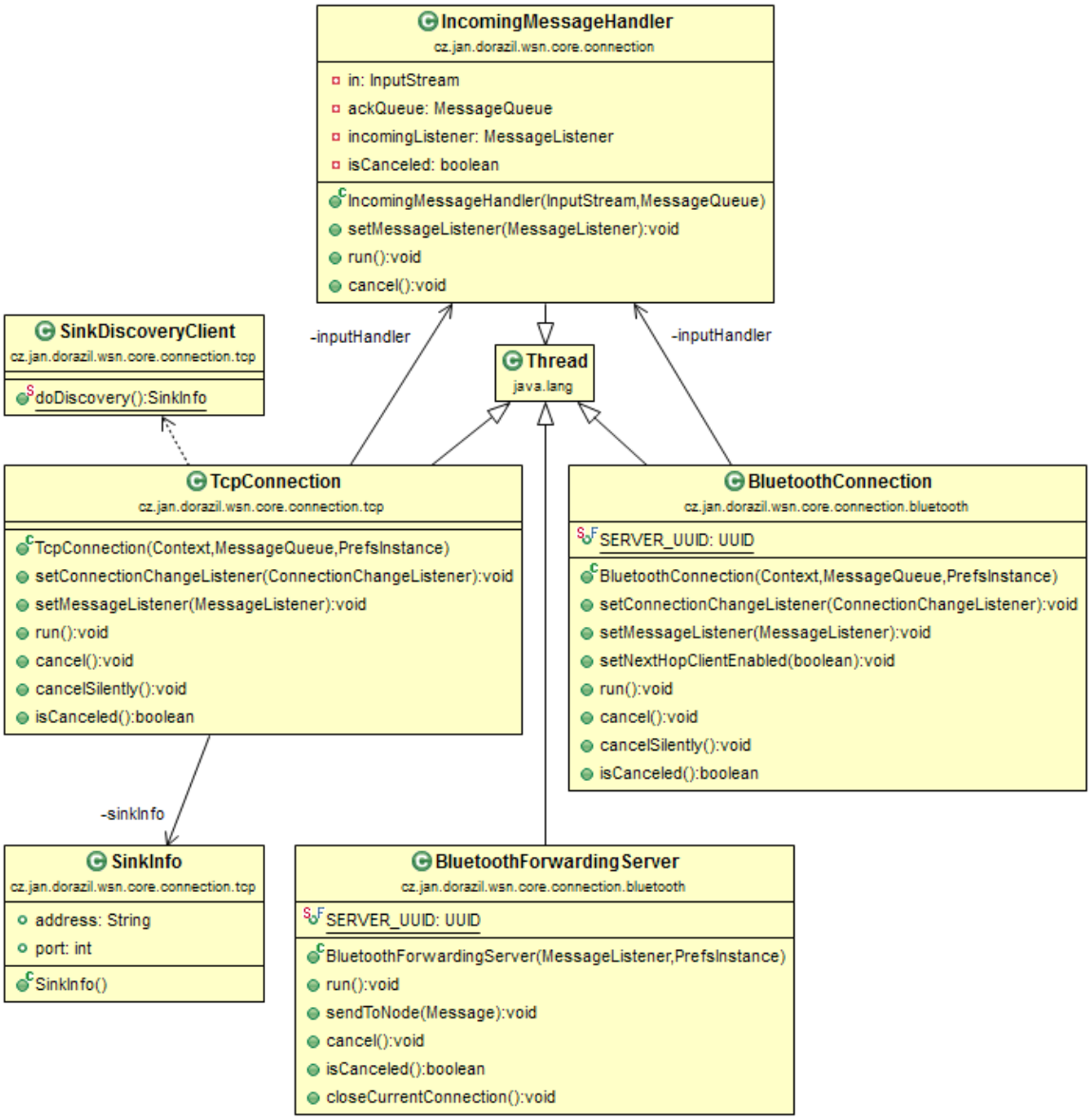
- *Pouze Bluetooth* (třída `BluetoothOnlyPolicy`) — umožňuje připojení směrem k základnové stanici pouze pomocí Bluetooth (buď přímo nebo přes next-hop uzel).
- *Pouze TCP/IP* (třída `TcpOnlyPolicy`) — umožňuje připojení k základnové stanici pouze prostřednictvím TCP/IP, tedy za pomoci Wi-Fi nebo mobilních dat.
- *TCP/IP preferováno* (třída `TcpPreferredPolicy`) — umožňuje připojení směrem k základnové stanici s využitím TCP/IP i Bluetooth. Mezi nimi přepíná, přičemž TCP/IP spojení je preferováno. Bluetooth je využito pouze v případě, kdy není TCP/IP dostupné (viz návrh architektury sítě v podkapitole 5.1).

Třída `Connection` také uchovává frontu odchozích zprávy a předává ji třídám, které implementují spojení. Proto můžeme metodou `sendUpdate(Message)` odeslat zprávu bez ohledu na technologii a stav připojení. Zpráva je vložena do fronty a odeslána při nejbližší možné příležitosti. Dále třída zajišťuje propagaci příchozích zpráv od základnové stanice do vyšší vrstvy aplikace. K tomu je použito rozhraní `MessageListener`. Jelikož spravuje také server pro přeposílání zpráv jiných uzlů, provádí jejich přidávání do fronty odchozích zpráv a obdobně v opačném směru. Opět je využito rozhraní `MessageListener`, tentokrát pro získávání zpráv z nižších vrstev aplikace (tcp a bluetooth, viz dále).

Vrstvy tcp a bluetooth

Balíky tcp a bluetooth obsahují třídy zajišťující bezdrátové připojení na nejnižší úrovni. Mají na starosti navázání a správu spojení a implementaci komunikačního protokolu – zde např. dochází k odesílání zpráv pro registraci a odregistraci uzlu na základnové stanici. Na obrázku 6.7 je uveden diagram těchto tříd.

Spojení ve směru k základnové stanici zajišťují třídy `TcpConnection` a `Bluetooth-Connection`. Ačkoliv první využívá technologii TCP/IP a druhá Bluetooth, mají velmi podobnou strukturu. Obě jsou vlákny jazyka Java a obsahují konečný automat. Ten má na starosti správu spojení tak, že i při výpadku se snaží o znovu-připojení. Obě třídy čtou zprávy z totožné fronty odchozích zpráv a odesílají je směrem k základnové stanici. Aby obě nepracovaly zároveň je zajištěno politikami připojení ve vyšší vrstvě aplikace. Protože je možné mezi nimi přepínat, obsahují, kromě metody `cancel()`, která odregistruje uzel



Obrázek 6.7: Diagram tříd pro bezdrátové spojení.

a ukončí vlákno, také metodu `cancelSilently()`, jež pouze ukončí vlákno, ale neprovádí odregistraci. Jinými slovy, k ukončení s odregistrací dojde pouze při ukončení práce uzlu zastavením služby `NodeService`. Jinak se pouze přepínají technologie připojení a provádí se vícenásobná registrace, aby základnová stanice vždy věděla, jak a kudy je uzel připojen.

TCP/IP spojení má navíc možnost, kterou jsme zatím nezmínili. Jedná se o automatické nalezení serveru základnové stanice v síti WLAN. Slouží k tomu, aby nebylo nutné nastavovat jeho IP adresu a port zvlášť na každém uzlu. To je výhodné, zejména je-li na sítí aktivní DHCP server pro automatické přidělování IP adres. Tuto funkci implementuje třída `SinkDiscoveryClient` a její statická metoda `doDiscovery()`. Ta při úspěšném nalezení vrátí instanci třídy `SinkInfo`, která nese IP adresu a port serveru. Mechanismus hledání je jednoduchý, klient odešle *broadcastem* zprávu na server, jehož port je dán do-

předu. Podporuje-li lokální síť zasílání všesměrových zpráv, a nachází-li se v ní základnová stanice se spuštěnými servery, odpoví zprávou nesoucí požadovanou IP adresu a port.

Poznamenejme také, že třída `BluetoothConnection` nezprostředkovává pouze přímé spojení se základnovou stanicí. Je-li to metodou `seNextHopClientEnabled(boolean)` povoleno, a nepodaří-li se přímé spojení se základnovou stanicí, snaží se třída postupně připojit k některému z uživatelem definované množiny next-hop uzlů. Adresování zde probíhá na základě MAC adres Bluetooth adaptérů.

Jelikož potřebujeme přijímat asynchronní zprávy ze základnové stanice (např. `GPS_RATE`), musíme implementovat mechanismus, který nám to umožní. V návrhu způsobu fungování komunikace uzlu se jedná o rozhodovací uzel „Je ACK?“ a frontu potvrzení ackQueue, viz diagram na obrázku 5.6. Toto chování je implementováno třídou `IncomingMessageHandler`, která tvoří samostatné vlákno. Při vytvoření je předán `InputStream`, ze kterého bude handler číst, a fronta zpráv, do které budou vkládána potvrzení o doručení. Takto mohou být zprávy pro vyšší vrstvu aplikace nebo připojený uzel přijímány a předávány přes rozhraní `MessageListener` k dalšímu zpracování. Naopak potvrzení jsou ukládána do fronty `ackQueue`. Vlákna `TcpConnection` a `BluetoothConnection` pak po odeslání zprávy, která vyžaduje potvrzení, čekají na doručení `ACK` blokujícím čtením z této fronty. Ze zřejmých důvodů si každé vytvořené spojení tvoří vlastní instanci třídy `IncomingMessageHandler`, fronta `ackQueue` tedy není sdílená, jako je tomu u fronty zpráv k odeslání.

Poslední nepopsanou entitou diagramu je třída `BluetoothForwardingServer`. Ta obsahuje Bluetooth server, který naslouchá na připojení jiného uzlu, pro který pak tento uzel zajistí přeposílání zpráv. Příchozí zprávy propaguje dále rozhraním `MessageListener`. Pro odeslání zprávy na připojený uzel je zde metoda `sendToNode(Message)`. V případě, že na uzel přijde zpráva typu `CLOSE_FWD_REQUEST`, je zde metoda `closeCurrentConnection()`, která uzavře spojení s právě připojeným uzlem. Server poté začne opět naslouchat na nové spojení.

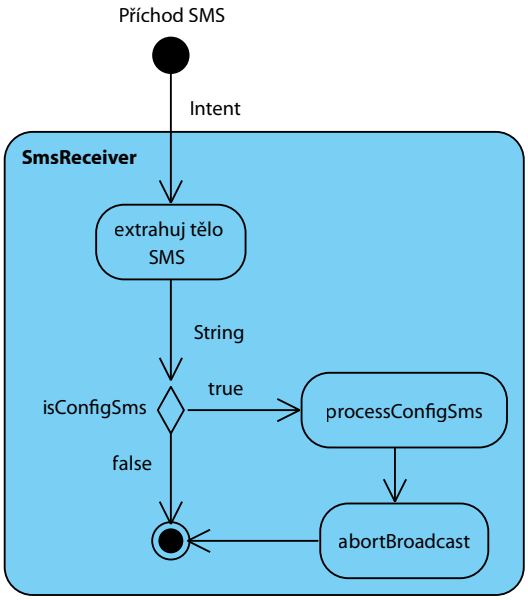
Pojem Bluetooth server je z hlediska implementace Bluetooth komunikace na Androidu chápán trochu jinak než je obvyklé. Nejedná se o server v pravém slova smyslu, jde v podstatě jen o určení rolí pro navázání peer-to-peer spojení. Jakmile je spojení ustaveno, UUID[4], na kterém server naslouchá, již nemůže být využito pro připojení jiného klienta (viz [26]). Díky tomuto omezení platformy Android je v jednu chvíli povoleno poskytování přeposílání zpráv pouze pro jeden uzel. Všimněme si, že toto nevylučuje řetězení uzlů za sebe, jen se nemůže více uzlů připojit ke stejnému next-hop uzlu zároveň. Toto omezení nám ale příliš nevadí, jelikož podporuje princip, kdy se dosahuje větší životnosti sítě zvýšením hustoty uzlů blízko základnové stanice, viz práce [43] a velikost sítě na str. 9. Zabrání se tak rychlému vybití akumulátoru uzlů, které by navíc přeposílaly data několika jiných.

Vrstva sms

Balík sms je tvořen třídou `SmsReceiver`. Jedná se o Broadcast Receiver, který přijímá SMS zprávy. Jedná-li se o konfigurační SMS pro uzel, je zpráva zkonzumována bez vědomí uživatele. Detaily tvorby takového receiveru jsou uvedeny v příloze B.4. Způsob práce třídy je uveden v diagramu aktivity na obrázku 6.8. Nastavením priority se podařilo umístit receiver na takové místo v systému, kde obdrží broadcast informující o nové SMS jako první. Může tak z objektu třídy `Intent` extrahovat tělo příchozí SMS a rozhodnout, zda se jedná o konfigurační SMS pro uzel (viz rozhodovací uzel *isConfigSms*). V kladném případě je pak

[4] Universally Unique Identifier, 128 bitový řetězcový identifikátor kanálu pro spojení.

provedena příslušná akce (viz *processConfigSms*) a je zabráněno další propagaci broadcastu (viz *abortBroadcast*). Jiné aplikace se tak o příchozí SMS nedozví, např. nezobrazí se notifikace. Nejedná-li se o konfigurační SMS, není její propagace zastavena. Může být tedy doručena např. výchozí systémové aplikaci pro práci s SMS a MMS zprávami.



Obrázek 6.8: Diagram aktivity třídy `SmsReceiver`.

Pomocí SMS můžeme vzdáleně nastavovat politiku spojení a zapínat či vypínat práci uzlu. To se může hodit v případech, kdy je uzel umístěn na vzdáleném nebo špatně dostupném místě. Funkce může být ale užitečná také pro šetření energie. Pomocí SMS můžeme totiž uzel vypnout tak, že se nejen zastaví jeho práce, ale také se vypnou Wi-Fi a Bluetooth rádia. Pak je nutné řešit problém, jak uzel opět vzdáleně zapnout. Řešením je vyčkat na spuštění přes SMS, a tím ušetřit energii, jež by se spotřebovala např. při periodickém probouzení uzlu.

Zda se jedná o konfigurační SMS je ověřeno v metodě `isConfigSms(String)` za pomoci regulárního výrazu. Konfigurační SMS musí obsahovat tělo s textem ve formátu

`wsn-config:<cmd>[,<cmd>];`,

kde `<cmd>` může nabývat hodnot z množiny příkazů

{`bt`, `tcp`, `tcp-pref`, `start`, `stop`, `restart`}.

Například SMS s tělem

`wsn-config:tcp-pref,start;`

nastaví politiku spojení na TCP/IP preferováno a spustí práci uzlu. Kompletní výčet příkazů a jejich funkcí je uveden v tabulce 6.1. Zpráva může také obsahovat prefix „`www:`“, který je automaticky přidáván při odesílání SMS z webových služeb. Vyžadovaný formát je *case-insensitive* a umožňuje také benevolenci v podobě přidávání bílých znaků mezi jednotlivé části zprávy.

Příkaz	Funkce
`bt`	Nastaví politiku na Pouze Bluetooth
`tcp`	Nastaví politiku na Pouze TCP/IP
`tcp-pref`	Nastaví politiku na TCP/IP preferováno
`start`	Spustí práci uzlu
`stop`	Zastaví práci uzlu
`restart`	Restartuje práci uzlu (např. pro aplikaci nově nastavené politiky)

Tabulka 6.1: Příkazy konfigurační SMS.

Vrstva preferences

Balík preferences obsahuje třídu `Prefs`, která slouží pro práci s nastavením aplikace. Najdeme v ní statické metody pro získávání/ukládání nastavení uživatele z/do persistentní paměti. Třída `Prefs` dále obsahuje vnořenou třídu `PrefsInstance`, jejíž instanci lze získat tovární metodou `currentInstance(Context)`. Ta pak reprezentuje snímek nastavení v době tvorby této instance. Hodnoty z ní lze pouze číst.

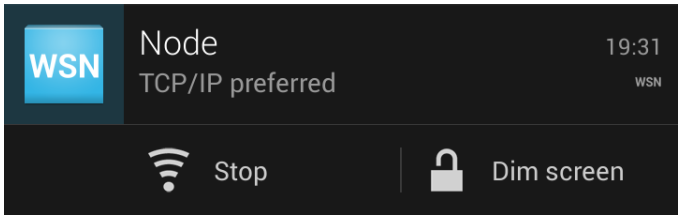
Snímek je využit ve třídě `NodeService`. Ta při spuštění získá aktuální instanci nastavení aplikace a propaguje ji dále do všech tříd souvisejících s prací uzlu, které provádí nějaké rozhodnutí na základě nastavení uživatele. Tímto mechanismem je zajištěno, že nastavení lze pomocí grafického uživatelského rozhraní nebo konfiguračních SMS měnit i za běhu uzlu. Projeví se pak až při jeho restartování.

Vrstva gui

Uživatelské rozhraní aplikace slouží pro ovládání a nastavení uzlu. Základem je řídící aktivita `ControlActivity`, která je zobrazena při spuštění aplikace. Její výsledná podoba je uvedena na obrázku E.1 v příloze E. Skládá se ze dvou částí. První je tvořena přepínacím tlačítkem pro spuštění a zastavení práce uzlu (s popiskem *Stopped*, resp. *Running*) a tlačítkem pro zobrazení aktivity `SleepActivity` (s popiskem *DIM screen*). Ta simuluje přechod do režimu spánku tím, že zobrazí černou obrazovku, skryje notifikační lištu, nastaví minimální jas displeje a zajistí, aby zařízení automaticky nepřešlo do opravdového režimu spánku. Slouží k tomu, že na některých zařízeních způsobuje automatický přechod do režimu spánku pokles výkonnosti Wi-Fi rádia, případně jeho úplné zastavení. Toto je primárně řešeno tzv. zámky[5], jež jsou aktivní po dobu běhu služby `NodeService`. Ty ale nemusí na všech zařízeních vždy zajistit plný výkon Wi-Fi, proto byla vytvořena tato alternativa.

Druhou část řídicí aktivity tvoří nastavení aplikace v podobě čtyř stránek, mezi kterými lze listovat tažením prstu do stran. Každá stránka je samostatný *fragment* platformy Android. Tyto fragmenty jsou umístěny do komponenty `ViewPager`, která zajišťuje listování. Pro indikaci aktuálně zobrazené stránky je využita volně dostupná a široce rozšířená komponenta `ViewPagerIndikator`[6]. Konkrétně je využita třída `TitlePageIndicator`, která přidává titulek nad aktivní stránku. Ten se animuje synchronně s listováním stránek. Pro přehlednější navigaci zobrazuje také titulky stránek, které se nachází nalevo a napravo od právě aktivní. Detailní popis jednotlivých stránek je uveden v příloze E.

[5] Konkrétně `WakeLock` pro plný výkon CPU a `WifiLock` pro plný výkon Wi-Fi v režimu spánku.
[6] Dostupné na http://viewpagerindicator.com/ pod licencí Apache License verze 2.0.



Obrázek 6.9: Rozbalená notifikace na Androidu 4.2.2.

Spuštěná práce uzlu může být indikována notifikací ve stavové liště zařízení, viz obrázek 6.9. Ta zobrazuje zvolenou politiku připojování a od Androidu 4.1 je v rámci ovládacích prvků notifikace umožněno zastavení práce uzlu a zobrazení výše popsané aktivity pro simulaci režimu spánku.

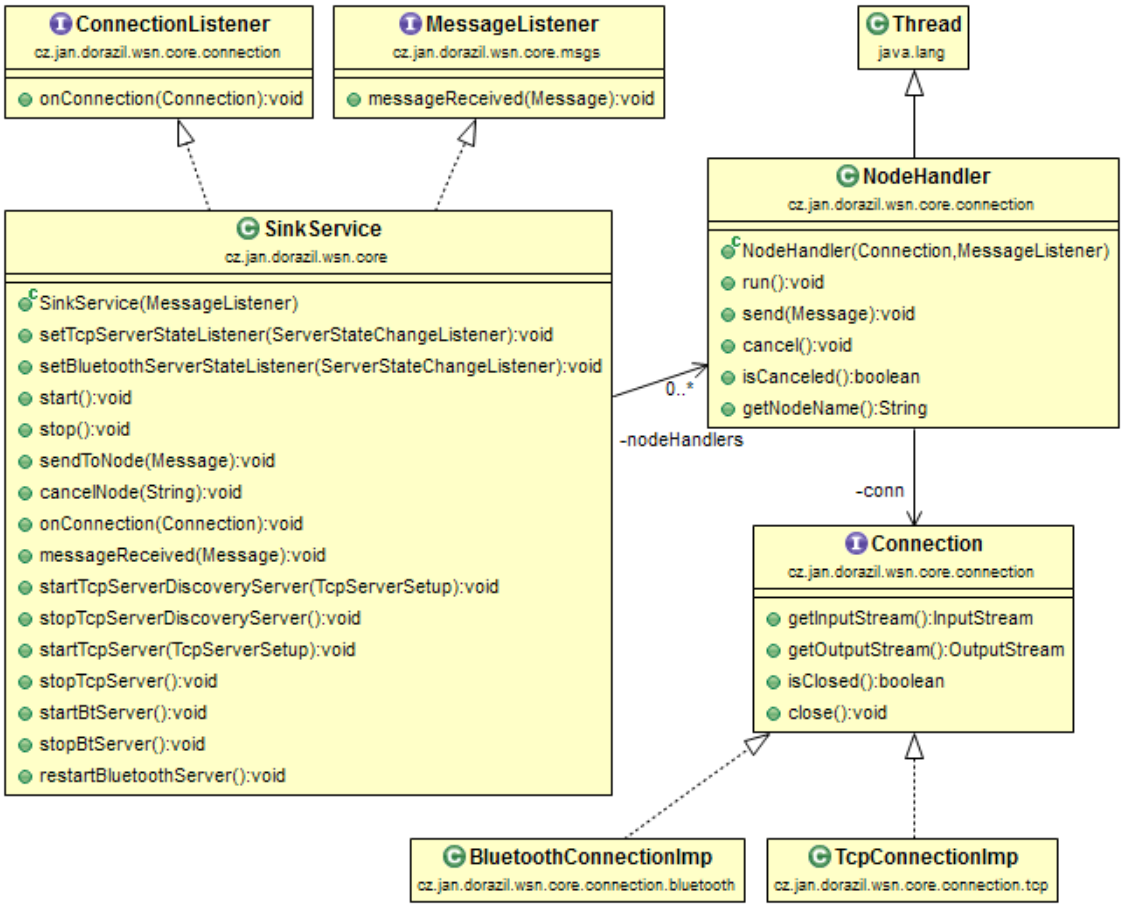
6.3 Implementace základnové stanice

V této podkapitole se zaměříme na popis implementace aplikace základnové stanice. Jedná se o program na platformě Java SE 7 využívající Swing toolkit pro tvorbu GUI. Budou uvedeny a popsány diagramy tříd klíčových vrstev aplikace, jejichž hierarchie je zobrazena ve formě diagramu balíčků na obrázku 5.7 v podkapitole 5.5. Pro větší přehlednost a skrytí některých implementačních detailů, jejichž popis není již v rozsahu této práce, nebudou diagramy tříd obsahovat vždy úplně všechny atributy a metody. Nakonec popíšeme uživatelské rozhraní aplikace, které je zde velmi důležité, neboť slouží k vizualizaci naměřených hodnot.

Jádro komunikace

V diagramu tříd na obrázku 6.10 je uvedeno jádro komunikace základnové stanice s uzly sítě. Je tvořeno především třídami z balíků core a connection, které implementují mechanismus navržený v diagramu na obrázku 5.9 v podkapitole 5.5. Základem je třída `SinkService`, jejíž běh reprezentuje spuštěnou práci základnové stanice, viz metody `start()` a `stop()`. Třída spravuje instance TCP/IP a Bluetooth serverů (viz Servery níže), které zde při připojení nového uzlu propagují instanci nového spojení prostřednictvím rozhraní `ConnectionListener`. Spojení je reprezentováno rozhraním `Connection`, které abstrahuje použitou technologii. Můžeme tak komunikovat se všemi uzly jednotným způsobem bez ohledu na to, zda je spojení implementováno třídou `TcpConnectionImp` nebo `BluetoothConnectionImp`.

Pro každé nové obdržené spojení třída `SinkService` vytvoří a spustí novou instanci handleru `NodeHandler` a přidá ji do seznamu `nodeHandlers`. Třída `NodeHandler` je vláknem jazyka Java, které zajišťuje komunikaci s přímo připojeným uzlem a potažmo také s těmi, jejichž zprávy tento uzel přeposílá. Handler implementuje komunikační protokol a příchozí zprávy dále propaguje rozhraním `MessageListener` do třídy `SinkService`. Odtud jsou pak propagovány stejným rozhraním do vrstvy gui, která implementuje uživatelské rozhraní a vizualizaci dat. Naopak, potřebujeme-li z GUI odeslat zprávu na uzel (např. `GPS_RATE`), využijeme metodu `sendToNode(Message)` třídy `SinkService`. Ta podle obsahu zásobníku návěští zprávy, kde se nachází název cílového uzlu a prvního uzlu v jeho směru, vyhledá příslušný handler a odešle zprávu jeho metodou `send(Message)`.



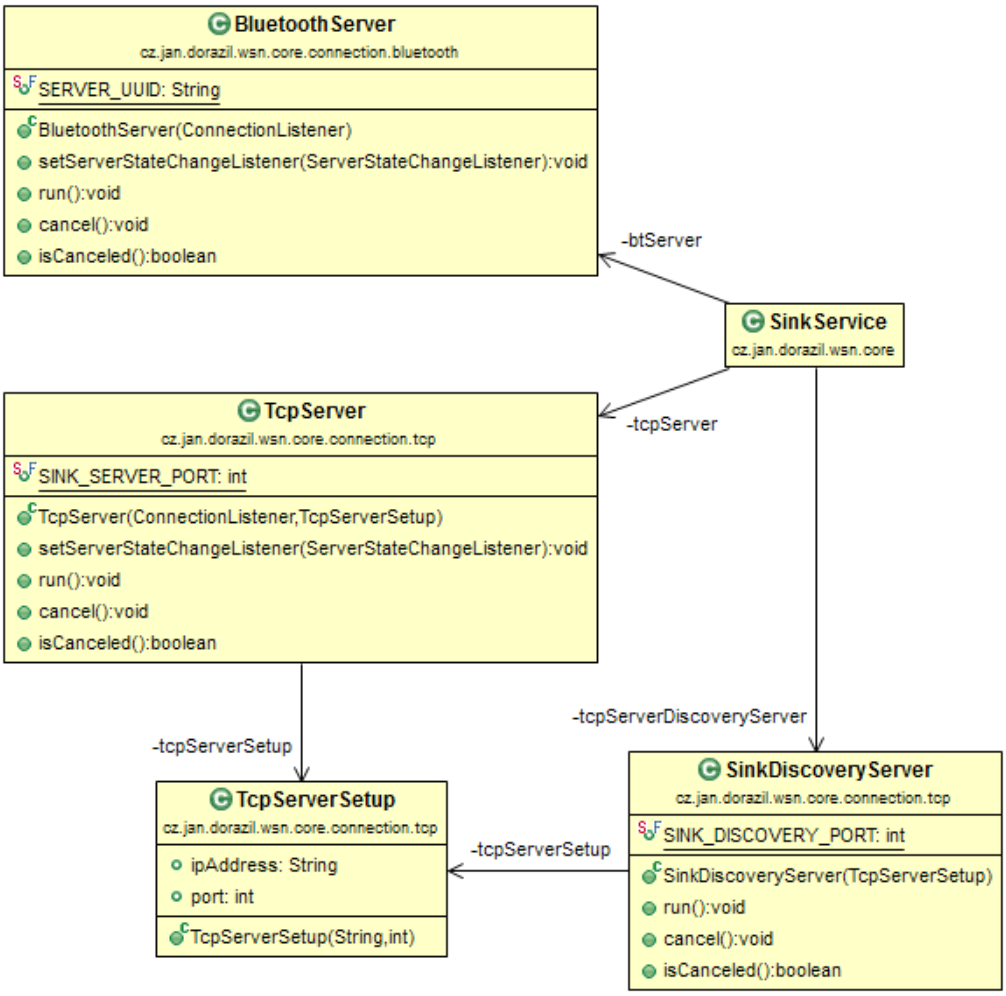
Obrázek 6.10: Diagram tříd jádra komunikace.

Poznamenejme, že balík msgs, který implementuje zprávy komunikačního protokolu a práci s nimi, je totožný s balíkem msgs v aplikaci implementující uzel sítě, viz str. 54.

Servery

V diagramu na obrázku 6.11 jsou uvedeny třídy, které implementují servery na základnové stanici. Jejich instance spravuje výše zmíněná třída `SinkService`, která také poskytuje metody pro jejich ovládání, např. z uživatelského rozhraní. Všechny servery jsou zároveň vlákny jazyka Java.

Třída `SinkDiscoveryServer` implementuje UDP server, který naslouchá a odpovídá broadcastovým dotazům na umístění TCP/IP serveru. Běží na dopředu zvoleném portu `SINK_DISCOVERY_PORT`, který nelze změnit a všechny uzly jej znají. Aktuální umístění TCP/IP serveru (jeho IPv4 adresa a číslo portu) je v aplikaci reprezentováno instancí třídy `TcpServerSetup`. Je uživatelsky nastavitelné při spuštění serveru, viz Vrstva gui níže. Konkurentní TCP/IP server základnové stanice je implementován třídou `TcpServer`, která všechny příchozí spojení propaguje rozhraním `ConnectionListener` do vyšší vrstvy aplikace. Třída `BluetoothServer` implementuje server pro přímé připojení uzlu přes Bluetooth. Nově příchozí spojení jsou rovněž předávána rozhraním `ConnectionListener` do vyšší vrstvy aplikace. Pomocí rozhraní `ServerStateChangeListener` mohou servery



Obrázek 6.11: Diagram tříd serverů.

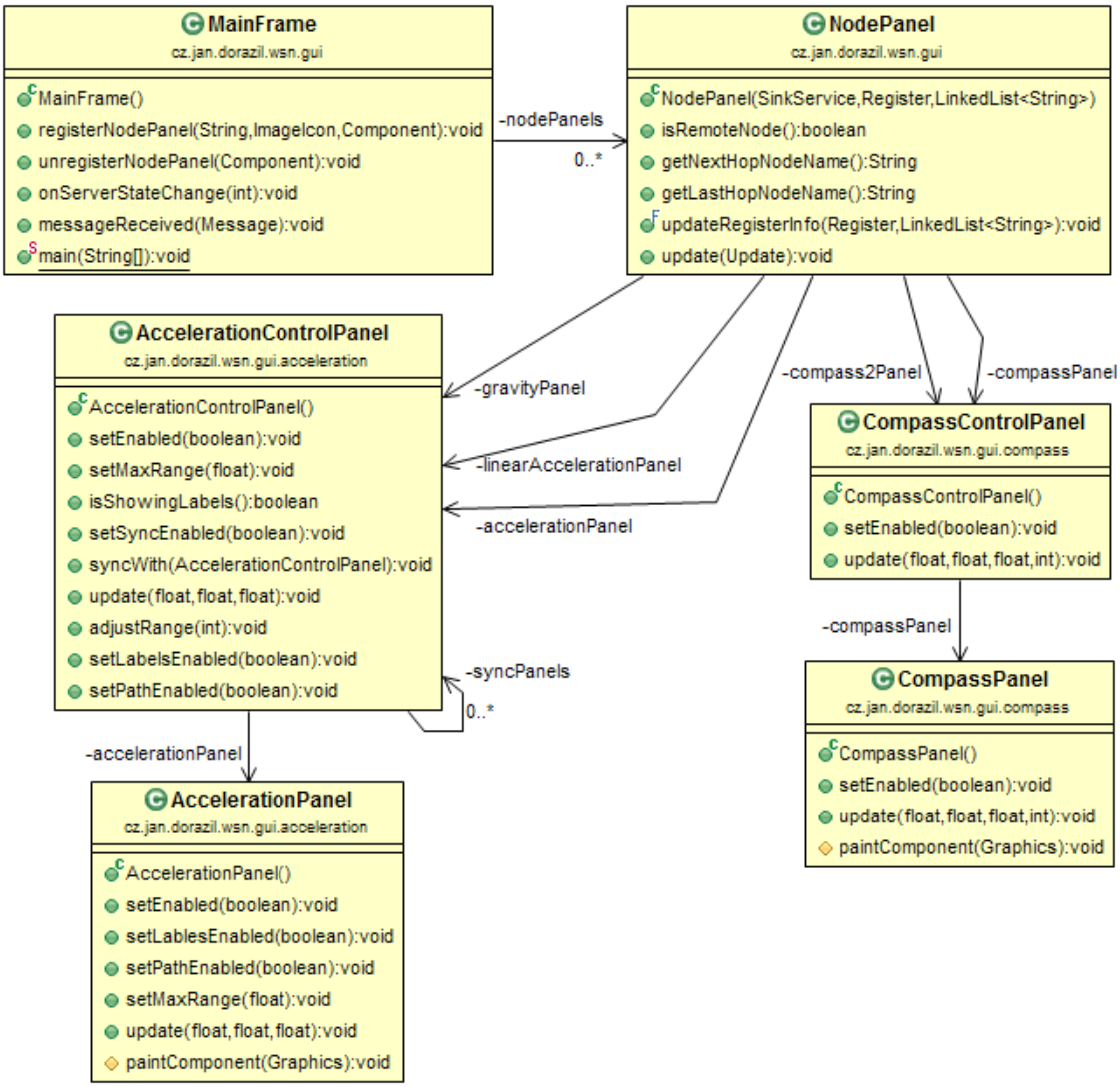
informovat vyšší vrstvy aplikace o svém aktuálním stavu.

Pro práci s Bluetooth je v aplikaci využita knihovna *BlueCove*[7]. Ta je neoficiální implementací rozhranní JSR-82, což je standardizované API jazyka Java pro ovládání Bluetooth. V aktuální verzi 2.1.0 neobsahuje podporu 64 bitových systémů, je tedy nutné spouštět aplikaci ve 32 bitovém JRE (Java Runtime Environment).

Vrstva gui

Nakonec této podkapitoly se zaměříme na grafické uživatelské rozhraní základnové stanice. To je tvořeno balíčkem gui a vnořenými balíčky acceleration, compass a graphics. Nejprve popíšeme třídy, jež GUI implementují, a poté představíme jeho výslednou podobu. Na obrázku 6.12 je zobrazen diagram tříd uživatelského rozhraní. Hlavní okno aplikace tvoří třída `MainFrame`. Ta obsahuje instanci třídy `SinkService` a ovládací prvky pro spouštění a zastavování serverů. Pro vizualizaci naměřených dat slouží třída `NodePanel`, která zobrazuje

[7]Dostupné na http://bluecove.org/ pod licencí Apache License verze 2.0. Modul pro linux je dostupný pod licencí GNU General Public License a může vyžadovat dodatečná nastavení, viz http://bluecove.org/bluecove-gpl/.



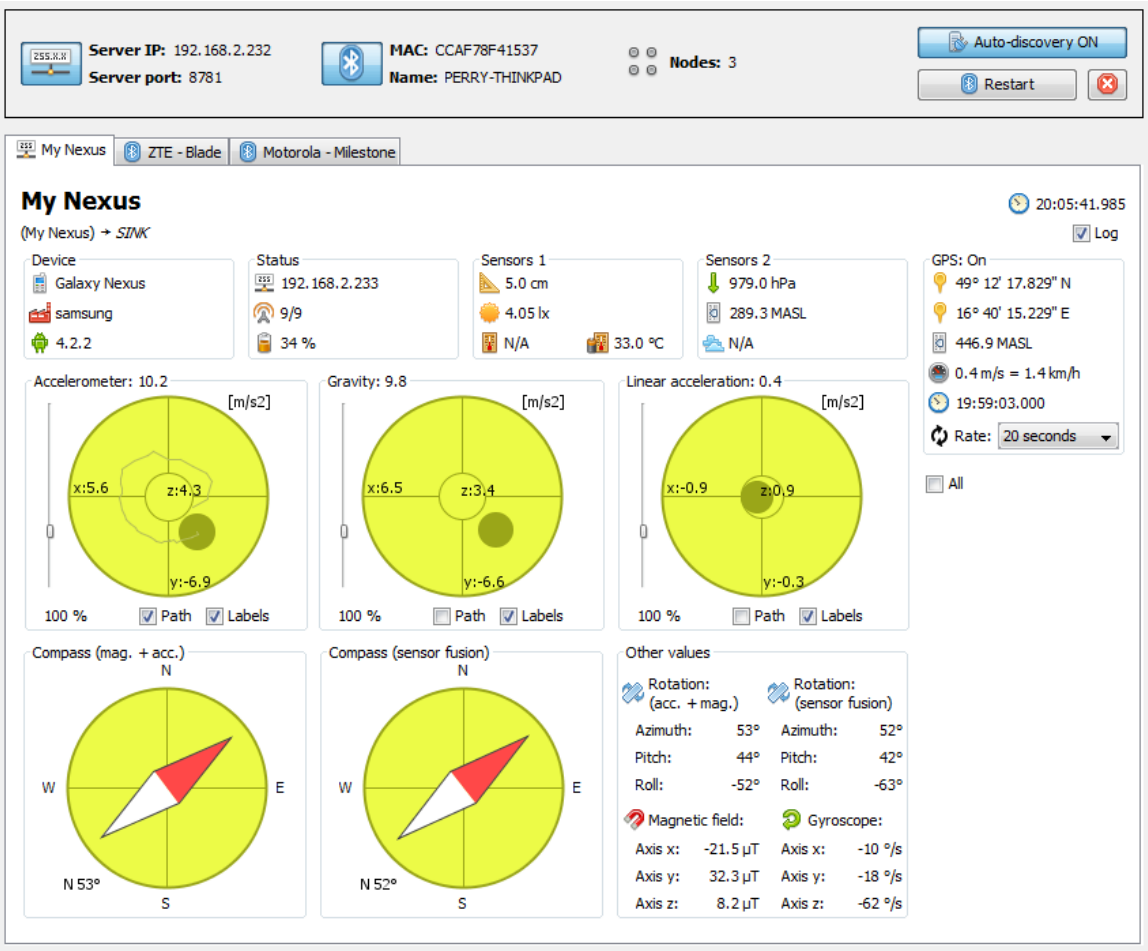
Obrázek 6.12: Diagram tříd uživatelského rozhraní.

informace a naměřené hodnoty jednoho uzlu. Třída `MainFrame` implementuje rozhraní `MessageListener` přes které získává příchozí zprávy ze třídy `SinkService`. Jedná-li se o registraci zatím neregistrovaného uzlu, je vytvořena nová instance třídy `NodePanel` a je uložena do seznamu panelů `nodePanels`. Přijde-li aktualizace nebo nová registrace již registrovaného uzlu, je zpráva předána odpovídajícímu panel, který aktuální hodnoty zobrazí.

Pro lepší vizualizaci zrychlení a natočení zařízení byly vytvořeny dvě vlastní komponenty, viz podkapitola 5.5 a obrázek 5.8. Komponenta zrychlení je implementována v balíčku acceleration. Ten obsahuje třídu `AccelerationPanel`, která zajišťuje vykreslování komponenty, a třídu `AccelerationControlPanel`, jež tu předchozí obaluje a přidává k ní dodatečné ovládací prvky, viz popis výsledné podoby dále. Pomocí metody `syncWith(AccelerationControlPanel)` můžeme definovat synchronizované nastavování ovládacích prvků pro více panelů zrychlení (úpravou jednoho se změní i ostatní). Uživatel tak může najednou ovládat panely zobrazující hodnoty akcelerometru (viz atribut

`accelerationPanel`), gravitaci (viz atribut `gravityPanel`) a lineární akceleraci (viz atribut `linearAccelerationPanel`).

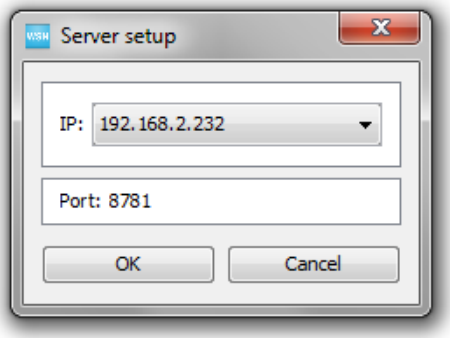
Druhou komponentou je kompas, jehož základ vykresluje třída `CompassPanel` z balíku compass. Obalová třída `CompassControlPanel` mu pak přidává popisky os. Zobrazujeme dva kompasy, jelikož máme dva způsoby získávání natočení zařízení. První je výsledkem spojení měření magnetometru a akcelerometru. Tato kombinace senzorů je na fyzických zařízeních častější, neposkytuje ale tak dobré výsledky jako rotace druhá. Ta je získaná z virtuálního senzoru rotace fůzí hodnot magnetometru, akcelerometru a gyroskopu (viz `TYPE_ROTATION_VECTOR` v tabulce 4.1). Všechny matematické operace, jež souvisí s výpočtem a zpracováním natočení zařízení z dat senzorů, za nás řeší pomocné metody, které jsou součástí API Androidu. Výpočet je tedy prováděn na uzlech a na základnovou stanici jsou v aktualizačních zprávách (`UPDATE` či `UPDATE_WITH_ACK`) zasílány již výsledné Eulerovy úhly, viz str. 33.



Obrázek 6.13: Uživatelské rozhraní základnové stanice.

Na obrázku 6.13 je uvedena výsledná podoba uživatelského rozhraní základnové stanice. Nahoře se nachází panel pro ovládání serverů. Obsahuje dvě přepínací tlačítka pro spuštění a zastavení TCP/IP a Bluetooth serveru. Při spuštění TCP/IP serveru je zobrazen dia-

log umožňující výběr jedné z dostupných IPv4 adres a nastavení čísla portu, viz obrázek 6.14. Ovládací panel dále zobrazuje IPv4 adresu a číslo portu TCP/IP serveru, MAC adresu a název lokálního Bluetooth adaptéru a počítadlo připojených uzlů. Napravo od něj se nachází přepínací tlačítko pro spuštění a zastavení discovery serveru pro automatické nalezení TCP/IP serveru základnové stanice v rámci WLAN. Pod ním je přítomno tlačítko pro restartování Bluetooth serveru, které může být užitečné, například dojde-li k restartu Bluetooth adaptéru. Vedle něj se nachází tlačítko pro ukončení komunikace s vybraným uzlem (buď přímo, nebo skrze zprávu `CLOSE_FWD_REQUEST`).



Obrázek 6.14: Dialog nastavení TCP/IP serveru.

Pod ovládáním serverů se nachází panely zobrazující naměřená data (instance třídy `NodePanel`). Panely jsou uspořádány do záložek. Titulek každé záložky zobrazuje jedinečný název uzlu. Ikona záložky pak udává, zda je uzel připojen technologií TCP/IP nebo Bluetooth. V levém rohu panelu se znovu nachází jedinečný název uzlu. Pod ním je zobrazena cesta jeho připojení. Jde o posloupnost uzlů, které postupně přeposílají data tohoto uzlu k základnové stanici. První je vždy daný uzel a poslední je základnová stanice, která je zde označena jako *SINK*. Na obrázku je tedy uzel My Nexus připojen přímo k základnové stanici. Vpravo nahoře se nachází časové razítko poslední doručené aktualizace. Pod ním je umístěn checkbox, jehož zatržením se spustí logování naměřených hodnot do CSV souboru s frekvencí maximálně jedenkrát za sekundu. Soubor je pojmenován podle názvu uzlu a je uložen do adresáře aplikace. Pokud již existuje, jsou nová data přidávána na jeho konec.

Nyní popíšeme jednotlivé části panelu, které zobrazují další informace o uzlu a jeho naměřené hodnoty. Část *Device* obsahuje (vždy shora) název zařízení, jméno jeho výrobce a verzi operačního systému Android. Dále *Status* zobrazuje IPv4 adresu uzlu, resp. Bluetooth MAC adresu uzlu, v případě TCP/IP spojení, resp. Bluetooth spojení. Při použití Wi-Fi je zobrazena síla signálu. Pod ní se nachází důležitá položka, jež informuje o stavu akumulátoru uzlu. Grafika ikon síly signálu a stavu akumulátoru se mění ve třech stupních podle aktuální hodnoty. Část *Sensors 1* zobrazuje hodnoty naměřené senzory přiblížení, osvětlení, okolní teploty a teploty akumulátoru. Na zařízení Galaxy Nexus není senzor okolní teploty přítomen, což je indikováno popiskem N/A. V části *Sensors 2* nalezneme data senzoru tlaku vzduchu, ze kterého je vypočtena nadmořská výška. Ta není v absolutní hodnotě příliš přesná, lze ale dobře využít relativně, např. pro detekci, v jakém poschodí budovy se uzel nachází. Posledním senzorem snímajícím okolní prostředí je senzor relativní vlhkosti, který ale na zařízení Galaxy Nexus není přítomen.

Napravo se nachází část *GPS*, která slouží nejen k zobrazení dat z GPS modulu, ale také pro jeho vzdálené ovládání. V záhlaví je indikátor povolení využití GPS (On/Off). Toto je volba, která musí být na Androidu provedena ručně v nastavení systému. Dále je zobrazena zeměpisná délka a šířka, nadmořská výška, rychlost a čas. Pod nimi se nachází combobox, jehož prostřednictvím lze volit minimální frekvenci snímání dat. Hledání polohy je indikováno animací (rotací) ikony dvou šipek.

Uprostřed panelu se nachází výsledná podoba komponenty pro vizualizaci zrychlení. Je zde přítomna třikrát, pro zobrazení surových dat akcelerometru a hodnot získaných z virtuálních senzorů gravitace a lineární akcelerace. Pomocí vertikálního táhla je možné nastavit maximální zobrazovaný rozsah komponenty. Ten se inicializuje při registraci uzlu hodnotou, kterou daný senzor poskytuje. Obvykle se jedná o $1\,g$ nebo $2\,g$, což na Zemi odpovídá $9{,}823\,\mathrm{m/s^2}$ a $19{,}646\,\mathrm{m/s^2}$. Ruční úpravou této hodnoty můžeme snížit nebo zvýšit citlivost komponenty na změny zrychlení. Checkboxem Path můžeme nechat zobrazit cestu pohyblivého kruhu ve formě lomené čáry, viz první komponenta pro akcelerometr. Druhým checkboxem Labels lze skrýt zobrazování popisků os. V záhlaví komponenty je uvedeno celkové zrychlení působící na zařízení. Napravo od komponent zrychlení se nachází checkbox All, jehož zatržení způsobí synchronizaci jejich ovládacích prvků, viz metoda `syncWith(AccelerationControlPanel)` výše.

Ve spodní části panelu najdeme výslednou podobu komponenty kompasu. Je zde přítomna dvakrát, díky výše zmíněným dvěma způsobům zjišťování natočení uzlu. První zobrazuje natočení (azimut) zjištěné z dat magnetometru a akcelerometru. Druhý, označený jako sensor fusion, vizualizuje natočení (azimut) získané virtuálním senzorem rotace, a to spojením měření magnetometru, akcelerometru a gyroskopu. Jelikož magnetometr snadno podléhá rušení (např. v přítomnosti magnetů nebo feromagnetických kovů), byl do aplikace zabudován mechanismus detekce ztráty jeho přesnosti. Aktualizační zprávy obsahují aktuální přesnost měření magnetometru ve čtyřech úrovních, jež odpovídají konstantám v tabulce 4.3. Nízká přesnost je vizualizovaná tučným oranžovým ohraničením a nespolehlivost naměřených hodnot tučným červeným ohraničením kompasu. Senzor pak můžeme zkalibrovat pohybem opisujícím číslici 8 postupně ve všech třech souřadných osách.

Poslední část *Other values* pro úplnost zobrazuje natočení okolo všech tří souřadných os získané oběma výše zmíněnými způsoby. Pod nimi se pak nachází surová data magnetometru a gyroskopu. Hodnoty gyroskopu byly pro lepší vizualizaci přepočteny z radiánů za sekundu na stupně za sekundu.

6.4 Alternativní architektura sítě

V podkapitole 2.1 jsme rozdělili architektury bezdrátových senzorových sítí na vrstvenou a seskupnou. Vytvořená WSN, tak jak byla navržena v podkapitole 5.1, spadá do vrstvené architektury. Změnou nastavení na uzlech (v aplikaci a systému Android) můžeme docílit také zapojení do architektury seskupné. Ta je tvořena skupinami uzlů, kde právě jeden uzel v každé skupině je řídicí. Ten má aktivovanou funkci přenosného Wi-Fi hotspotu a poskytuje tak ostatním uzlům skupiny své mobilní internetové připojení (chová se jako přístupový bod). Základnová stanice má veřejnou IP adresu a uzly sítě s ní komunikují přes internet prostřednictvím řídicího uzlu, ke kterému náleží (jsou v dosahu jeho Wi-Fi signálu).

Vzhledem k relativně vysoké ceně mobilních dat a energetické náročnosti sdílení internetového připojení, není tato topologie vhodná pro získávání naměřených hodnot v reálném čase. Své užití by ale mohla nalézt v detekční bezdrátové senzorové síti, kdy navíc můžeme skupiny rozmístit do odlehlých oblastí, daleko od sebe nebo od základnové stanice.

6.5 Testování vytvořené WSN

V této podkapitole se budeme zabývat testováním a vyhodnocením vytvořené WSN. Nejprve uvedeme, na jakých zařízeních byla síť testována. Poté popíšeme výsledky testů propustnosti a životnosti sítě. Na závěr změříme přibližný dosah Bluetooth adaptérů testovacích mobilních zařízení.

Zařízení pro vývoj a testování

Jako uzly sítě byly k dispozici tři telefony s operačním systémem Android: *Samsung Galaxy Nexus* s nejnovější verzí Androidu 4.2.2, *Motorola Milestone* s verzí 2.3.5 (alternativní firmware od CyanogenMod[8]) a *ZTE Blade* také s verzí 2.3.5.

Roli základnové stanice převzal notebook *Lenovo ThinkPad T520* s 64 bitovým operačním systémem Windows 7 Professional SP1, Bluetooth modulem ThinkPad Bluetooth 3.0 od společnosti Broadcom a Wi-Fi adaptérem Intel® Centrino® Advanced-N 6205. Základnová stanice byla testována také na laptopu Dell Latitude E550 s 32 bitovým operačním systémem Windows XP Professional SP3, Bluetooth modulem Dell Wireless 370 Bluetooth Mini-card od společnosti Broadcom a Wi-Fi adaptérem Intel® WiFi Link 5100 AGN. Dále pak na notebooku Compal FL90v2 s 32 bitovým operačním systémem Windows 7 Professional SP1, Bluetooth modulem Broadcom Bluetooth 2.0 EDR USB Dongle a Wi-Fi adaptérem Atheros AR5B91 Wireless Network Adapter.

Propustnost sítě

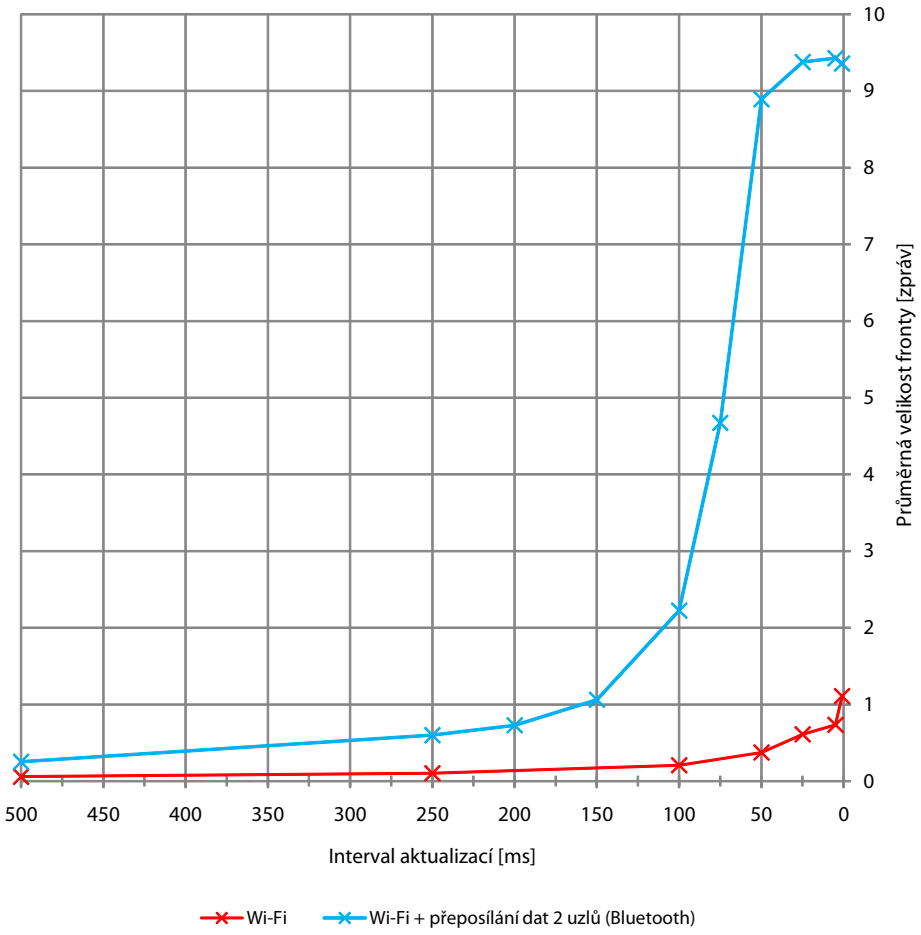
Pro určení rychlosti sítě byla změřena propustnost jejího uzlu. Měření bylo provedeno na referenčním zařízení Galaxy Nexus. Propustnost byla zjišťována jako závislost průměrné velikosti fronty zpráv k odeslání za jednu minutu provozu sítě na rychlosti zasílání aktualizací. Pro výpočet průměrné velikosti fronty $\bar{s}$ byl použit vzorec 6.1, kde c je kapacita fronty (v našem případě 10), t_i je čas, po který fronta obsahovala i položek, a T je celkový čas, přes který průměrnou velikost měříme. Každé měření bylo provedeno třikrát a z výsledků byl vypočten aritmetický průměr. Pro odměření jedné minuty byl využit časovač, který po vypršení ukončil běh služby `NodeService`.

$$\bar{s} = \sum_{i=0}^{c} \left(i \times \frac{t_i}{T} \right) \tag{6.1}$$

Celkem byly měřeny čtyři situace. Samotný uzel připojený přes Wi-Fi a uzel připojený stejným způsobem, ale navíc přeposílající data dvou dalších uzlů (zapojení ZTE Blade → Motorola Milestone → Galaxy Nexus → *SINK*). Obojí pak ve variantě s potvrzováním zpráv a bez něj. Interval aktualizací byl nastavován v rozmezí od 500 ms po 1 ms, což je minimum, které lze zvolit. Povolení potvrzování a interval aktualizací byl nastavován vždy shodně na každém zařízení. Výsledky byly vyneseny do grafů.

V grafu na obrázku 6.15 jsou zobrazeny výsledky měření s potvrzováním pomocí `ACK` zpráv. Na ose x jsou postupně nastavované intervaly aktualizací v milisekundách. Na ose y se pak nachází odpovídající průměrná velikost fronty (počet zpráv v intervalu 0–10). Červená křivka zobrazuje průběh testu pro samostatný uzel připojený přes Wi-Fi. Je vidět,

[8] Viz http://www.cyanogenmod.org/.



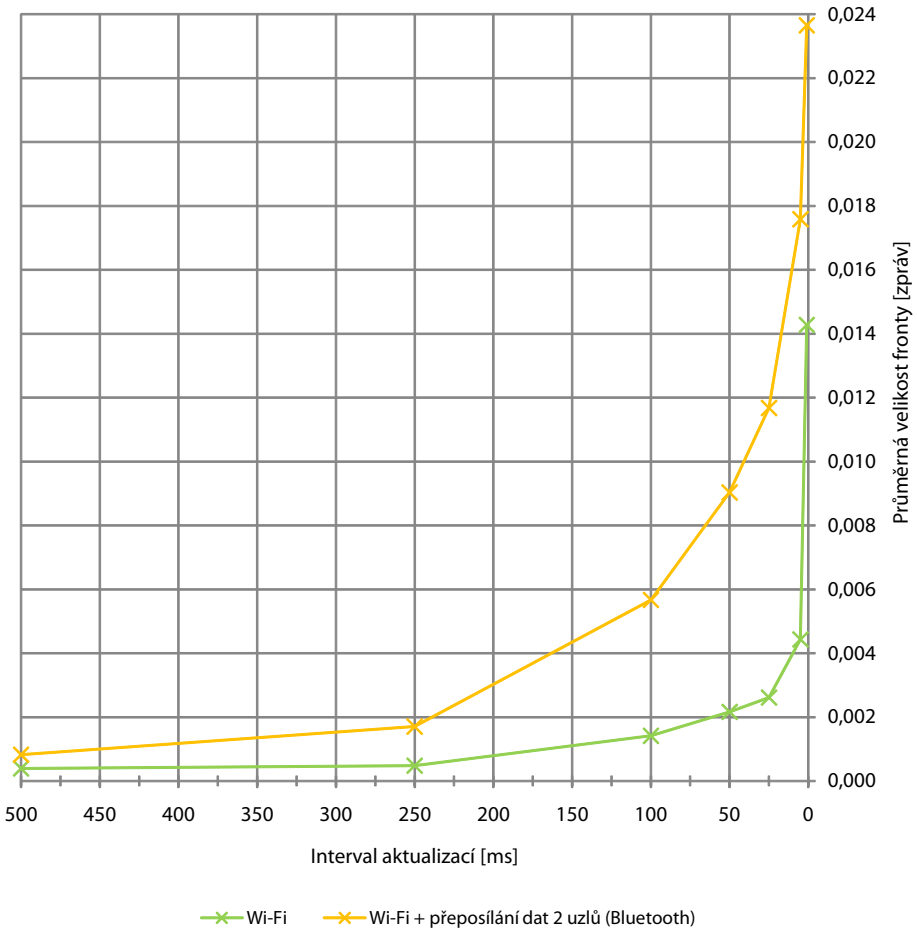
Obrázek 6.15: Graf propustnosti uzlu s potvrzováním.

že velikost fronty je ve většině případů menší než jedna. To značí, že uzel bez obtíží zvládá odesílat zprávy všemi rychlostmi.

Modrá křivka v grafu zobrazuje průběh varianty s přeposíláním dat dvou uzlů připojených přes Bluetooth. Zde je patrné, že po překročení hranice deseti zpráv za sekundu (interval 100 ms) se fronta začíná rychle plnit. Po dosažení dvaceti zpráv za sekundu (interval 50 ms) je již fronta většinu času zaplněná a dochází tak k zahazování nejstarších zpráv z jejího začátku. Na základnové stanici je pak patrné zhruba dvousekundové zpoždění ve vizualizaci. Díky volbě relativně malé kapacity fronty a způsobu zahazování nejstarších zpráv při jejím zaplnění, nemá zpoždění vizualizace tendenci se dále s narůstající rychlostí aktualizací zvětšovat.

V grafu na obrázku 6.16 jsou uvedeny výsledky měření bez potvrzování. Zelená křivka zobrazuje průběh pro samostatný uzel bez přeposílání a žlutá křivka průběh varianty s přeposíláním dat dvou dalších uzlů. I při nejvyšší rychlosti tisíc zpráv za sekundu (interval 1 ms) nepřekročí průměrná délka fronty hodnotu 0,024 zpráv. To značí v podstatě okamžité odeslání každé zprávy ihned po jejím přidání do fronty.

Grafy svým tvarem připomínají nepřímou úměrnost. To proto, že čím menší je interval aktualizací, tím jsou kladeny větší nároky na rychlost odesílání zpráv a tedy průměrná



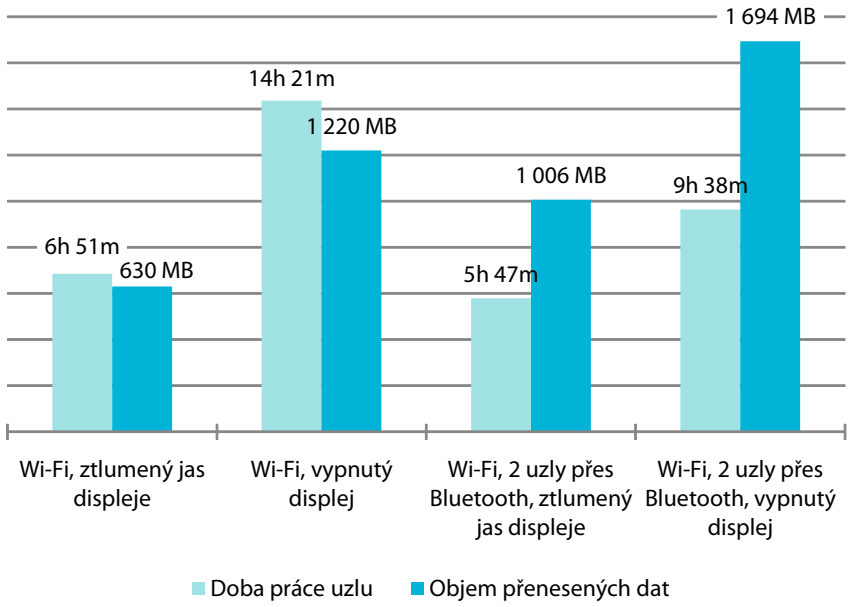
Obrázek 6.16: Graf propustnosti uzlu bez potvrzování.

velikost fronty je větší, a naopak.

Důvodem horší propustnosti při použití potvrzování je především fakt, že každá zpráva zůstává ve frontě k odeslání až do chvíle, dokud není potvrzeno její úspěšné doručení. Takto může být v případě neúspěchu odeslána později, např. jinou technologií, dojde-li u stávající k výpadku spojení. Kompromisem mezi rychlostí a spolehlivostí může být využití potvrzování pouze pro uzly, které jsou k základnové stanici připojeny přímo. Ostatní ať pro zachování vizualizace v reálném čase potvrzování nevyžadují. Musíme si ale dát pozor, aby tato konfigurace příliš nezatížila přímo připojený uzel. Například pokud víme, že uzel nebude přeposílat zprávy více než jednoho nebo dvou uzlů. Pak můžeme na všech uzlech využít interval 50 nebo 100 ms, který zajistí hladkou vizualizaci.

Životnost sítě

V podkapitole 2.2 jsme v souvislosti s velikostí sítě poprvé použili pojem životnost. Ta může znamenat například dobu, za kterou se vybije akumulátor prvnímu uzlu, za kterou se vybijí akumulátory určité části WSN, nebo za kterou klesne procento konektivity nebo pokrytí pod danou úroveň. Životnost vytvořené WSN byla změřena jako doba do vybití akumulátoru uzlu, vždy s různým nastavením nebo zapojením. Měření bylo opět provedeno na



Obrázek 6.17: Sloupcový graf životnosti uzlu sítě.

referenčním zařízení Galaxy Nexus s kapacitou akumulátoru 1 750 mAh. Telefon byl vždy nejprve nabit na 100 %. Poté byl změřen čas a celkový objem přenesených dat (vlastních i případně od jiných uzlů) do jeho úplného vybití, což se projevilo vypnutím zařízení. Interval aktualizací byl nastaven na 50 ms bez potvrzování a kromě použitých technologií byly všechny ostatní vypnuty (mobilní internet, NFC, GPS a ve dvou případech i Bluetooth). Připojení k mobilní síti bylo aktivní. Byly ukončeny pokud možno všechny ostatní běžící aplikace.

Výsledky byly zaneseny do sloupcového grafu na obrázku 6.17. Pro každý ze čtyř prováděných testů jsou vždy uvedeny dva sloupce s dobou práce uzlu a celkovým objemem přenesených dat. Při prvním testu (v grafu zleva) byl samostatný uzel připojen k základnové stanici přes Wi-Fi. Během celé doby jeho práce byla v popředí aktivita `SleepActivity` pro simulaci režimu spánku, viz popis implementace vrstvy gui na str. 61. Ukázalo se, že s touto konfigurací dokáže uzel pracovat 6 h 51 m, během kterých přenese 630 MB dat. Z přehledu využití akumulátoru v nastavení systému Android bylo zjištěno, že práce uzlu měla pouze 13% podíl na jeho vybití a celých 84 % bylo spotřebováno pro napájení displeje, byť s minimálním jasem.

Jelikož byl tento výsledek neuspokojující, byl proveden druhý test, který se od předchozího lišil v tom, že zařízení bylo během práce v úsporném režimu, kdy je displej zcela vypnut. O plný výkon CPU a Wi-Fi se staraly jen použité zámky. Uzel takto dokázal pracovat 14 h 21 m, během kterých přenesl 1 220 MB dat. Napájení displeje se tak na vybití akumulátoru nepodílelo vůbec, zatímco práce uzlu spotřebovala 75 % jeho kapacity. Zbylé procenta pak byly po malých částech rozděleny mezi aktivity jako pohotovostní režim, OS Android, Wi-Fi a další.

Zbylé dva testy byly provedeny stejným způsobem jako dva předchozí s tím rozdílem, že uzel zároveň přeposílal data dvou za sebou zapojených uzlů. Kromě Wi-Fi bylo tedy využito i Bluetooth. Schéma zapojení zařízení bylo shodné jako u měření propustnosti

(ZTE Blade → Motorola Milestone → Galaxy Nexus → *SINK*). Ve variantě se ztlumeným displejem vydržel uzel pracovat 5 h 47 m, během kterých přenesl 1 006 MB dat (počítala se i přeposílaná data). To je přibližně o jednu hodinu méně než bez přeposílání. Displej spotřeboval 81 % kapacity akumulátoru, zatímco samotná práce uzlu pouhých 12 %.

Situaci se opět podařilo zlepšit úplným vypnutím displeje v posledním čtvrtém testu. Uzel pracoval 9 h 38 m, během kterých přenesl 1 694 MB dat. Na vybití akumulátoru se podílel již ze 78 %. Z naměřených dat je zřejmé, že chceme-li maximalizovat životnost vytvořené WSN, je nutné uzly přepnout do režimu spánku, kdy je displej zařízení zcela vypnut. Poté záleží také na aktuálním propojení, jelikož uzly, které poskytují přeposílání jiným, vydrží pracovat kratší dobu.

Dosah Bluetooth

V tabulce 6.2 je uveden přibližný dosah Bluetooth adaptérů testovaných mobilních zařízení. Měření bylo prováděno v otevřeném prostoru bez překážek. Jelikož se ukázalo, že hranice dosahu není ostrá, za ztrátu signálu byly považovány, kromě úplné ztráty komunikace, také její výrazné výpadky, které se již neslučovaly s monitorováním v reálném čase. Proto jsou naměřené hodnoty brány pouze jako orientační. Pro zařízení s nejdelším dosahem (Galaxy Nexus) nelze navíc s jistotou tvrdit, že jde o dosah Bluetooth modulu tohoto zařízení. Výpadky komunikace mohly být teoreticky způsobeny ztrátou signálu ze strany základnové stanice (notebooku).

Zařízení	**Dosah [m]**
Samsung Galaxy Nexus	25
Motorola Milestone	17
ZTE Blade	3,5

Tabulka 6.2: Přibližný dosah Bluetooth testovaných zařízení.

Kapitola 7

Závěr

Cílem této diplomové práce bylo prozkoumat a popsat možnosti použití mobilních zařízení jako uzlů bezdrátové senzorové sítě. Jelikož typický uzel WSN disponuje schopností bezdrátové komunikace a je vybaven množinou senzorů, nahlédli jsme na mobilní zařízení právě z těchto dvou hledisek. Blíže jsme se zaměřili na platformu Android, především díky její otevřenosti a velkému rozšíření.

Na základě zjištěných informací byla navržena a implementována bezdrátová senzorová síť s využitím mobilních zařízení jako uzlů. Ta v reálném čase monitoruje veličiny všech dostupných senzorů a přehledně je vizualizuje na základnové stanici. Volitelně mohou být snímána také data z GPS. Komunikace v síti může probíhat přes infrastrukturní Wi-Fi, peer-to-peer Bluetooth nebo případně pomocí mobilního internetového připojení. Bluetooth spojením lze tvořit vrstvy sítě. Každý uzel má definovánu množinu next-hop uzlů, které pak zajistí přeposílání jeho zpráv. Ačkoliv komunikace probíhá primárně ve směru od uzlů k základnové stanici, v opačném směru mohou být uzlu zasílány asynchronní příkazy. Uzly lze také vzdáleně konfigurovat pomocí SMS zpráv.

Síť byla implementována s využitím zařízení s operačním systémem Android. Základnovou stanici tvoří desktopová aplikace na platformě Java Standard Edition. Obě aplikace mají grafické uživatelské rozhraní. Během testování vytvořené WSN jsme se zaměřili na její propustnost, životnost a dosah Bluetooth adaptérů testovaných zařízení.

V budoucnu je možné síť rozšířit o podporu více navzájem komunikujících decentralizovaných základnových stanic. V případě, že si nevystačíme se senzory dostupnými přímo v mobilním zařízení, můžeme využít např. platformu Arduino pro připojení externího senzoru. Práci nám zde ulehčí ADK (Android Development Kit) nebo můžeme využít protokol AOA (Android Open Accessory), více viz [39].

Literatura

[1] Bluetooth Special Interest Group: *Building with the Technology: Overview* [online]. [cit. 2013-01-06].
URL https://www.bluetooth.org/Building/overview.htm

[2] Bluetooth Developer Portal: *Bluetooth Core Specification v4.0* [online]. [cit. 2013-01-07].
URL http://developer.bluetooth.org/TechnologyOverview/Pages/v4.aspx

[3] Allan, A.: *Basic Sensors in iOS*. O'Reilly Media, 2011, ISBN 978-1-449-30846-9, 108 s.

[4] Balasubramanian, N.; Balasubramanian, A.; Venkataramani, A.: Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009, ISBN 978-1-60558-771-4, s. 280–293.

[5] Cisco: *802.11ac: The Fifth Generation of Wi-Fi, Technical White Paper* [online]. srpen 2012 [cit. 2012-12-30].
URL http://www.cisco.com/en/US/prod/collateral/wireless/ps5678/ps11983/white_paper_c11-713103.pdf

[6] Google: *Environment Sensors* [online]. 2013-01-04 [cit. 2013-01-08].
URL http://developer.android.com/guide/topics/sensors/sensors_environment.html

[7] Google: *Motion Sensors* [online]. 2013-01-04 [cit. 2013-01-08].
URL http://developer.android.com/guide/topics/sensors/sensors_motion.html

[8] Google: *Position Sensors* [online]. 2013-01-04 [cit. 2013-01-08].
URL http://developer.android.com/guide/topics/sensors/sensors_position.html

[9] Google: *Sensors Overview* [online]. 2013-01-04 [cit. 2013-01-08].
URL http://developer.android.com/guide/topics/sensors/sensors_overview.html

[10] Google: *ConnectivityManager* [online]. 2013-01-15 [cit. 2013-01-25].
URL http://developer.android.com/reference/android/net/ConnectivityManager.html

[11] Google: *NetworkInfo* [online]. 2013-01-15 [cit. 2013-01-25].
URL http://developer.android.com/reference/android/net/NetworkInfo.html

[12] Google: *TelephonyManager* [online]. 2013-01-15 [cit. 2013-01-25].
URL http://developer.android.com/reference/android/telephony/TelephonyManager.html

[13] Google: *HttpURLConnection* [online]. 2013-01-15 [cit. 2013-01-27].
URL http://developer.android.com/reference/java/net/HttpURLConnection.html

[14] Google: *WifiInfo* [online]. 2013-01-15 [cit. 2013-01-29].
URL http://developer.android.com/reference/android/net/wifi/WifiInfo.html

[15] Google: *WifiManager* [online]. 2013-01-15 [cit. 2013-01-29].
URL http://developer.android.com/reference/android/net/wifi/WifiManager.html

[16] Google: *Wi-Fi Direct* [online]. 2013-01-15 [cit. 2013-01-31].
URL http://developer.android.com/guide/topics/connectivity/wifip2p.html

[17] Google: *WifiP2pManager* [online]. 2013-01-15 [cit. 2013-01-31].
URL http://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html

[18] Google: *BluetoothAdapter* [online]. 2013-01-15 [cit. 2013-02-01].
URL http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html

[19] Google: *BluetoothClass* [online]. 2013-01-15 [cit. 2013-02-01].
URL http://developer.android.com/reference/android/bluetooth/BluetoothClass.html

[20] Google: *BluetoothDevice* [online]. 2013-01-15 [cit. 2013-02-01].
URL http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html

[21] Google: *Dashboards* [online]. 2013-04-02 [cit. 2013-04-27].
URL http://developer.android.com/about/dashboards/index.html

[22] Google: *Determining and Monitoring the Connectivity Status* [online]. [cit. 2013-01-25].
URL http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html

[23] Google: *Google Cloud Messaging for Android* [online]. [cit. 2013-02-02].
URL http://developer.android.com/google/gcm/index.html

[24] Google: *Location Strategies* [online]. [cit. 2013-04-15].
URL http://developer.android.com/guide/topics/location/strategies.html

[25] Google: *Media and Camera* [online]. [cit. 2013-04-15].
URL http://developer.android.com/guide/topics/media/index.html

[26] Google: *Bluetooth* [online]. [cit. 2013-05-01].
URL http://developer.android.com/guide/topics/connectivity/bluetooth.html

[27] Google: *Intents and Intent Filters* [online]. [cit. 2013-05-08].
URL http://developer.android.com/guide/components/intents-filters.html

[28] Gujjarlapudi, R.; Kakaraparthy, A.; Goli, K. M.: Integration of Android Operating System with Wireless Sensor Network. *International Jurnal of Computer Science And Technology*, ročník 3, č. 1, leden–březen 2012, ISSN 0976-8491.

[29] Hanáček, P.: Bezdrátové a mobilní sítě. Kurz na FIT VUT, 2012/2013.
URL http://www.fit.vutbr.cz/study/msc/course-l.php.cs?id=8563

[30] Hart, J. C.; Francis, G. K.; Kauffman, L. H.: Visualizing Quaternion Rotation. *ACM Transactions on Graphics*, ročník 13, č. 3, červenec 1994: s. 256–276, ISSN 0730-0301.

[31] Healy, M.; Newe, T.; Lewis, E.: Wireless Sensor Node Hardware: A Review. In *Sensors, 2008 IEEE*, říjen 2008, ISSN 1930-0395, s. 621–624.

[32] Heinzelman, W.; Chandrakasan, A.; Balakrishnan, H.: An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communications*, ročník 1, č. 4, říjen 2002.

[33] Kalic, G.; Bojic, I.; Kusek, M.: Energy Consumption in Android Phones when Using Wireless Communication Technologies. In *MIPRO, 2012 Proceedings of the 35th International Convention*, květen 2012, ISBN 978-1-4673-2577-6, s. 754–759.

[34] Kotwal, A.; Bray, T.; Chan, T.: *Google I/O 2012 – The Sensitive Side of Android* [online]. 2012-06-29 [cit. 2013-04-12].
URL http://www.youtube.com/watch?v=Q0V_ld7iNw4

[35] Li, Y.; Thai, M. T.; Wu, W.: *Wireless Sensor Networks and Applications*. Springer, 2008, ISBN 978-0-387-49591-0, 464 s.

[36] Meier, R.: *Professional Android^{TM} 4 Application Development*. John Wiley & Sons, 2012, ISBN 978-1-118-10227-5, 864 s.

[37] Meier, R.: *Google I/O 2012 – Making Good Apps Great: More Advanced Topics for Expert Android Developers* [online]. 2012-06-30 [cit. 2013-04-16].
URL http://www.youtube.com/watch?v=PwC1OlJo5VM

[38] Microsoft: *Sensors for Windows Phone* [online]. 2013-04-08 [cit. 2013-04-15].
URL http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202968(v=vs.105).aspx

[39] Milette, G.; Stroud, A.: *Professional Android^{TM} Sensor Programming*. John Wiley & Sons, 2012, ISBN 978-1-118-18348-9, 552 s.

[40] Sachs, D.: *Sensor Fusion on Android Devices: A Revolution in Motion Processing* [online]. 2010-08-02 [cit. 2013-01-08].
URL http://www.youtube.com/watch?v=C7JQ7Rpwn2k

[41] Schiller, J.: *Mobile Communications*. Addison-Wesley, 2003, ISBN 0-321-12381-6, 512 s.

[42] Švéda, M.: Přenos dat, počítačové sítě a protokoly. Kurz na FIT VUT, 2011/2012.
URL http://www.fit.vutbr.cz/study/msc/course-l.php.cs?id=8137

[43] Wang, D.; Cheng, Y.; Wang, Y.; aj.: Lifetime Enhancement of Wireless Sensor Networks by Differentiable Node Density Deployment. In *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on*, říjen 2006, ISBN 1-4244-0507-6, s. 546–549.

Příloha A

Obsah CD

K diplomové práci je přiloženo CD, které obsahuje:

- `/`
  - `thesis` ..... Pdf soubor s technickou zprávou
    - `tex` ..... Zdrojové kódy technické zprávy
  - `WSN-Node` ..... Zdrojové kódy pro uzel (Eclipse, Andorid)
    - `bin` ..... Instalační apk balíček
    - `doc` ..... Programová dokumentace
  - `WSN-Sink` ..... Zdrojové kódy základnové stanice (NetBeans, Java SE)
    - `dist` ..... Spustitelný jar soubor
    - `doc` ..... Programová dokumentace
    - `lib` ..... Potřebné knihovny
  - `ViewPagerIndicator` ..... Knihovna potřebná pro GUI uzlu

Příloha B

Detaily bezdrátové komunikace na platformě Android

B.1 Monitorování připojení

Máme-li již instanci správce připojení (viz výpis 3.1 na str. 27), můžeme zjistit bližší informace o aktivním připojení (je-li vůbec nějaké). K tomu slouží třída `NetworkInfo`, jejíž instanci popisující aktivní připojení získáme voláním metody `getActiveNetworkInfo()` správce připojení. Příklad získání instance a jejího využití je uveden ve výpisu B.1. Jsou zde získány informace o tom, zda je vůbec nějaké internetové připojení aktivní (`isConnected`), a zda jde o připojení přes Wi-Fi nebo mobilní síť (`isWifiConnected` a `isMobileDataConnected`). Dále, díky vysokým cenám mobilních dat v zahraničí, může být užitečné zjistit, jestli jsme připojení přes roaming (`onRoaming`). Můžeme také určit, kterou technologií mobilních dat jsme připojeni (`isGPRS`, `isEDGE`, `isUMTS`, ...). Tato informace může být užitečná například v případě, kdy chceme v aplikaci snížit objem přenášených dat na pomalejších technologiích.

Výpis B.1: Příklad získání informací o aktivním připojení.

```
// Informace o aktivním připojení.
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

// Připojeno?
boolean isConnected = (activeNetwork != null) &&
    activeNetwork.isConnectedOrConnecting();

// Připojeno přes Wi-Fi?
boolean isWifiConnected = isConnected && (activeNetwork.getType() ==
    ConnectivityManager.TYPE_WIFI);

// Připojeno přes mobilní data (GPRS/EDGE, 3G,...)?
boolean isMobileDataConnected = isConnected && (activeNetwork.getType() ==
    ConnectivityManager.TYPE_MOBILE);

// Mobilní data v zahraničí?
boolean onRoaming = isMobileDataConnected && activeNetwork.isRoaming();
```

```
// Zjištění, přes jakou technologii mobilních dat jsme připojeni.
// Pro jednoduchost uvažujeme pouze GPRS, EDGE a UMTS. Další konstanty
// viz třída TelephonyManager.
if (isMobileDataConnected) {
    int subType = activeNetwork.getSubtype();
    boolean isGPRS = (subType == TelephonyManager.NETWORK_TYPE_GPRS);
    boolean isEDGE = (subType == TelephonyManager.NETWORK_TYPE_EDGE);
    boolean isUMTS = (subType == TelephonyManager.NETWORK_TYPE_UMTS);
    // ...
}
```

Pouhé zjištění aktuálního stavu připojení mnohdy aplikaci nestačí. Často si přejeme být informováni, dojde-li ke změně tohoto stavu. Pro tento účel lze na platformě Android vytvořit tzv. Broadcast Receiver. Stručně řečeno, jde o kousek kódu aplikace, který bude vyvolán systémem ve chvíli, kdy se změní stav připojení. Broadcast Receiver musí být, stejně jako aktivita nebo služba, deklarován v manifestu aplikace, viz výpis B.2. V deklaraci je uveden Intent Filter, jehož atributy Action specifikují, na jaké události bude receiver reagovat. Zde máme zájem o monitorování pouze jedné události, kterou je změna stavu připojení (`CONNECTIVITY_CHANGE`). Příklad implementace Broadcast Receiveru je uveden ve výpisu B.3. Třída musí být odvozena od abstraktní třídy `BroadcastReceiver`. Přepsáním metody `onReceive(Context, Intent)` pak definujeme vlastní reakci na receiverem sledované události. V extra datech předaného intentu můžeme nalézt bližší informace o změně stavu připojení. V uvedeném příkladu implementace je z intentu zjištěno, zda je zařízení připojeno k internetu nebo ne. Bližší informace pak mohou být získány pomocí správce připojení, viz výše.

Výpis B.2: Deklarace Broadcast Receiveru na změnu stavu připojení.

```
<!-- Deklarováno v manifestu aplikace, uvnitř tagu "application". -->
<receiver android:name=".ConnectivityChangedReceiver" >
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

Výpis B.3: Broadcast Receiver na změnu stavu připojení.

```
public class ConnectivityChangedReceiver extends BroadcastReceiver {
    /** Metoda je volána systémem při změně stavu připojení. */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Zjištění stavu.
        String extraName = ConnectivityManager.EXTRA_NO_CONNECTIVITY;
        boolean isConnected = !intent.getBooleanExtra(extraName, true);

        // Je-li připojeno, získání více informací o aktivním připojení.
        if (isConnected) {
            ConnectivityManager cm = (ConnectivityManager)
                context.getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
            // ...
        }
    }
}
```

B.2 Režim V letadle

Ve verzích předcházejících Androidu 4.2 bylo možné zapínat a vypínat režim V letadle z kódu aplikace, viz výpis B.4. Potřebné oprávnění je uvedeno ve výpisu B.5. Od verze 4.2 již ale tento postup nefunguje, jelikož konstanta `AIRPLANE_MODE_ON` byla přesunuta ze třídy `Settings.System`[1] do `Settings.Global`. Tato nastavení lze z aplikace číst, ale ne zapisovat. Konstanta může být tedy využita jen pro zjištění, zda je režim V letadle aktivní.

Výpis B.4: Zapnutí/vypnutí režimu V letadle (před verzí 4.2).

```
void setAirplaneMode (Context context, boolean enable) {
    // Nastavení nové hodnoty v nastavení systému.
    Settings.System.putInt(context.getContentResolver(),
            Settings.System.AIRPLANE_MODE_ON, enable ? 1 : 0);

    // Vyslání broadcastu, který způsobí aplikaci nově nastavené hodnoty.
    Intent intent = new Intent(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    intent.putExtra("state", enable);
    context.sendBroadcast(intent);
}
```

Výpis B.5: Oprávnění potřebné pro změnu nastavení systému.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
```

B.3 HTTP komunikace

Ve výpisu B.6 je uveden příklad ustavení HTTP spojení se serverem a je naznačen postup, jakým by probíhala výměna dat (upload a download). K přístupu na internet je zapotřebí oprávnění uvedené ve výpisu B.7. Výměna dat probíhá zápisem do output streamu, resp. čtením dat z input streamu, což není pro programovací jazyk Java nijak výjimečné. Metodu `setDoOutput(boolean)` je nutné zavolat s argumentem `true`, požadujeme-li upload dat. Metoda `setChunkedStreamingMode(int)` s argumentem `0` povoluje fragmentaci uploadovaných dat a nastavuje výchozí velikost fragmentu. Slouží k tomu, aby se odesílaná data nehromadila v operační paměti mobilního zařízení, dokud nejsou kompletně zapsána do output streamu. Místo toho se data při zapisování do streamu již postupně odesílají v podobě fragmentů.

Výpis B.6: Ustavení HTTP spojení pro upload a download dat.

```
HttpURLConnection httpConn = null;
try {
    // URL serveru, ke kterému se připojujeme.
    URL url = new URL("http://www.android.com/");

    // Otevření spojení.
    httpConn = (HttpURLConnection) url.openConnection();
```

[1] Zde je označena jako Deprecated.

```
        // 1) Upload.
        httpConn.setDoOutput(true);
        httpConn.setChunkedStreamingMode(0);
        OutputStream out = httpConn.getOutputStream();
        // Zápis dat do "out" ...

        // 2) Download.
        // Ověření návratového kódu.
        int responseCode = httpConn.getResponseCode();
        if (responseCode = HttpURLConnection.HTTP_OK) {
            InputStream in = httpConn.getInputStream();
            // Načtení dat z "in" ...
        }

    } catch (MalformedURLException ex) {
        // Nevalidní URL ...

    } catch (IOException ex) {
        // Chyba při spojování nebo komunikaci ...

    } finally {
        // Uzavření spojení, pokud bylo vytvořeno.
        if (httpConn != null) httpConn.disconnect();
    }
```

Výpis B.7: Oprávnění pro přístup k internetu.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.INTERNET" />
```

B.4 Posílání a příjem SMS/MMS

Ve výpise B.8 je uveden příklad odeslání SMS zprávy pomocí intentu na vestavěnou SMS aplikaci mobilního zařízení. Nejprve je vytvořena nová instance třídy `Intent`, kde již v konstruktoru specifikujeme akci odeslání zprávy (`ACTION_SENDTO`) a nastavíme cíl (pomocí schématu `sms:`) spolu s telefonním číslem příjemce. Poté jako extra data intentu přidáme text SMS zprávy. Nakonec vyvoláme aplikaci pro obsloužení vytvořeného intentu metodou `startActivity(Intent)`, která je součástí abstraktní třídy `Context` a tříd od ní odvozených (typicky aktivita nebo služba). Je-li aplikací, které umí odesílat SMS zprávy více, a není-li nastavena výchozí, uživatel si bude moci vybrat, kterou z nich spustit. Ve spuštěné aplikaci pak bude již předvyplněno číslo příjemce a text SMS. Odeslání zprávy pak musí uživatel potvrdit stiskem tlačítka „Odeslat“.

Obdobný postup lze aplikovat i pro zaslání MMS zprávy, viz výpis B.9. Všimněme si, že akce `ACTION_SENDTO` musela být nahrazena obecnější `ACTION_SEND`. To má za následek, že kromě MMS aplikace se může uživateli ve výběru aplikací, které umí intent obsloužit, objevit i např. emailový klient nebo aplikace Gmail. Číslo příjemce musíme nastavit v extra datech intentu, stejně jako text zprávy a URI multimediální přílohy. MIME typ přílohy specifikujeme metodou `setType(String)`.

Výpis B.8: Odeslání SMS zprávy pomocí intentu.

```
// Vytvoření SMS intentu - specifikace akce a čísla příjemce.
Intent smsIntent = new Intent(Intent.ACTION_SENDTO,
    Uri.parse("sms:555123456"));

// Přidání textu zprávy.
smsIntent.putExtra("sms_body", "Text SMS zprávy ...");

// Vyvolání intentu - spuštění aplikace, která jej umí obsloužit.
startActivity(smsIntent);
```

Výpis B.9: Odeslání MMS zprávy pomocí intentu.

```
// Získání URI na multimediální přílohu MMS zprávy.
Uri attachedUri = Uri.parse("content://media/external/images/photo.jpeg");

// Vytvoření MMS intentu.
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attachedUri);
smsIntent.putExtra("address", "555123456");
smsIntent.putExtra("sms_body", "Text MMS zprávy ...");
smsIntent.putExtra(Intent.EXTRA_STREAM, attachedUri);
smsIntent.setType("image/jpeg");

// Vyvolání intentu - spuštění aplikace, která jej umí obsloužit.
startActivity(smsIntent);
```

Předchozí postup zahrnoval interakci uživatele, což pravděpodobně není příliš vhodné, uvažujeme-li použití mobilního zařízení jako uzlu WSN. Přímé odeslání SMS zprávy je možné provést prostřednictvím správce SMS, resp. třídy `SmsManager`. Ve výpise B.10 je uvedeno získání instance třídy `SmsManager` a příklad odeslání textové a datové SMS. Pro odeslání textové SMS obsahuje správce metodu `sendTextMessage(String, String, String, PendingIntent, PendingIntent)`. První parametr udává číslo příjemce, druhým lze specifikovat SMS centrum (`null` pro výchozí), třetí je text zprávy a poslední dva lze využít pro sledování a potvrzení úspěšného doručení zprávy[2].

Výpis B.10: Odeslání SMS zprávy přímo z aplikace.

```
// Získání instance třídy SmsManager.
SmsManager smsManager = SmsManager.getDefault();

// 1) Textová zpráva.
String sendTo = "555123456";
String text = "Text SMS zprávy ...";
smsManager.sendTextMessage(sendTo, null, text, null, null);

// 2) Datová zpráva (binární).
short destPort = 80;
byte[] data = [ ... vlastní data ... ];
smsManager.sendDataMessage(sendTo, null, destPort, data, null, null);
```

Pro odeslání datové (binární) SMS lze využít metodu `sendDataMessage(String, String, short, byte[], PendingIntent, PendingIntent)`. Rozdíl oproti tex-

[2] Více o potvrzení doručení SMS zprávy viz [36]

tové zprávě spočívá pouze v tom, že tělo zprávy má podobu pole bytů a navíc je specifikováno číslo cílového portu. Požadujeme-li přímé odesílání SMS zpráv pomocí třídy `SmsManager`, musíme do manifestu aplikace přidat patřičné oprávnění, viz výpis B.11.

Výpis B.11: Oprávnění pro odeslání SMS přímo z aplikace.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.SEND_SMS" />
```

Odesílání SMS zpráv máme již pokryto, proto se nyní podíváme na jejich příjem. Při příchodu nové SMS na mobilní zařízení je vyvolán Broadcast Intent s akcí `"android.provider.Telephony.SMS_RECEIVED"`. Poznamenejme, že se jedná o řetězcový literál, jelikož veřejné API neposkytuje příslušnou konstantu. Hodnota se tedy může v budoucích verzích Androidu změnit. Ačkoliv spoleh na takovéto hodnoty není považován za dobrý postup, chceme-li v aplikaci přijímat SMS zprávy, nemáme na výběr. Pro příjem SMS musíme tedy vytvořit Broadcast Receiver.

Ve výpisu B.12 je uvedena registrace příslušného Broadcast Receiveru v manifestu aplikace. Intent Filter obsahuje nepovinný atribut, se kterým jsme se dosud u receiveru nesetkali. Je jím `android:priority` s hodnotou nastavenou na 1 000. Jeho uvedením vytvoříme tzv. Ordered Broadcast (seřazený). Broadcast tak bude doručován receiverům postupně podle jejich priority od nejvyšší po nejnižší. Takto zajistíme, aby naše aplikace obdržela broadcast informující o nově příchozí SMS jako první. Jsme na půl cesty ke „zkonzumování" příchozích SMS bez vědomí uživatele, více viz níže.

Pro umožnění přijímání SMS zpráv v aplikaci je nutné deklarovat příslušné oprávnění v manifestu, viz výpis B.13.

Výpis B.12: Deklarace Broadcast Receiveru na příchozí SMS zprávy.

```
<!-- Deklarováno v manifestu aplikace, uvnitř tagu "application". -->
<receiver android:name=".SmsReceiver" android:exported="true" >
    <intent-filter android:priority="1000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Výpis B.13: Oprávnění pro naslouchání na příchozí SMS zprávy.

```
<!-- Deklarováno v manifestu aplikace. -->
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

Příklad implementace Broadcast Receiveru je uveden ve výpise B.14. Extra data intentu obsahují nově příchozí SMS zprávy. Ty jsou ještě zabaleny do svých PDU (Protocol Data Unit) a je třeba je z nich rozbalit. K tomu je využita statická metoda `createFromPdu(byte[])` třídy `SmsMessage`. Každá SMS zpráva je pak reprezentována instancí třídy `SmsMessage`. Ta obsahuje metody, které lze použít pro získání informací o SMS zprávě a jejím obsahu. Např. metodu `getOriginatingAddress()` pro telefonní číslo odesílatele, `getTimestampMillis()` pro čas přijetí a `getMessageBody()`, resp. `getUserData()`, pro obsah textové, resp. datové, SMS zprávy.

Na konci ukázkového kódu receiveru voláme metodu `abortBroadcast()`. Ta zabrání broadcastu v dalším šíření – nebude doručen žádnému dalšímu Broadcast Receiveru. Dohromady s nastavením priority Intent Filtru v manifestu aplikace (viz výše), která zajistí,

že naše aplikace obdrží broadcast jako první, docílíme „zkonzumování“ příchozích SMS zpráv bez vědomí uživatele. Informace o nově příchozí zprávě se totiž nedostane k vestavěné aplikaci, která by ji za normálních okolností uložila do databáze a zobrazila uživateli notifikaci.

Výpis B.14: Naslouchání na příchozí SMS zprávy.

```
public class SmsReceiver extends BroadcastReceiver {

    /**
     * Metoda je volána systémem při příchodu nové/nových SMS.
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Získání extra dat z intentu.
        Bundle extras = intent.getExtras();

        if (extras != null) {
            // Rozbalení příchozích SMS zpráv z jejich PDU.
            Object[] pdus = (Object[]) extras.get("pdus");
            SmsMessage[] msgs = new SmsMessage[pdus.length];

            for (int i = 0; i < pdus.length; ++i) {
                msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
            }

            // Přes všechny příchozí SMS zprávy.
            for (SmsMessage msg : msgs) {
                // Telefonní číslo odesílatele.
                String from = msg.getOriginatingAddress();

                // Čas přijetí zprávy.
                long when = msg.getTimestampMillis();

                // Text zprávy (pro textové SMS).
                String text = msg.getMessageBody();

                // Data zprávy (pro datové SMS)
                byte[] data = msg.getUserData();

                // ...
            }

            // Zkonzumování broadcastu.
            abortBroadcast();
        }
    }

}
```

B.5 Wi-Fi

Máme-li již instanci správce Wi-Fi (viz výpis 3.3 na str. 28), můžeme jednoduše docílit zapnutí nebo vypnutí Wi-Fi adaptéru jeho metodou `setWifiEnabled(boolean)`, viz

výpis B.15. Před jejím voláním je pomocí metody `getWifiState()` zjištěno, zda adaptér není již zapnutý nebo se právě nezapíná, resp. zda není již vypnutý nebo se právě nevypíná.

Výpis B.15: Zapnutí a vypnutí Wi-Fi adaptéru.

```
// Získání aktuálního stavu Wi-Fi adaptéru.
int state = wifi.getWifiState();

final int ENABLED   = WifiManager.WIFI_STATE_ENABLED;
final int ENABLING  = WifiManager.WIFI_STATE_ENABLING;
final int DISABLED  = WifiManager.WIFI_STATE_DISABLED;
final int DISABLING = WifiManager.WIFI_STATE_DISABLING;

// 1) Zapnutí Wi-Fi adaptéru.
if (state != ENABLED && state != ENABLING) {
    wifi.setWifiEnabled(true);
}

// 2) Vypnutí Wi-Fi adaptéru.
if (state != DISABLED && state != DISABLING) {
    wifi.setWifiEnabled(false);
}
```

Chceme-li monitorovat stav Wi-Fi připojení musíme vytvořit Broadcast Receiver. Jeho tvorba zde již nebude uvedena, jelikož je analogická k předchozím receiverům – na změnu stavu připojení (str. 81) a na příchozí SMS (str. 85). Místo toho popíšeme pouze nejpoužívanější akce, na které můžeme pomocí receiveru naslouchat:

- `WIFI_STATE_CHANGED_ACTION` — indikuje změnu stavu Wi-Fi adaptéru jako hardwarové komponenty. Extra data intentu obsahují dvě hodnoty: aktuální stav (`EXTRA_WIFI_STATE`) a předchozí stav (`EXTRA_PREVIOUS_STATE`). Možné stavy jsou: zapnuto, zapínání, vypnuto, vypínání (viz výpis B.15) a neznámý stav (`WIFI_STATE_UNKNOWN`).
- `SUPPLICANT_CONNECTION_CHANGE_ACTION` — indikuje ustavení a ztrátu spojení s přístupovým bodem. Extra data intentu obsahují příznak `EXTRA_SUPPLICANT_CONNECTED`, který má hodnotu `true`, je-li spojení ustaveno, nebo `false`, je-li ztraceno.
- `RSSI_CHANGED_ACTION` — slouží pro monitorování změn síly signálu přístupového bodu, ke kterému je zařízení právě připojeno. Extra data intentu obsahují hodnotu `EXTRA_NEW_RSSI` typu integer, která informuje o aktuální síle signálu. Před použitím této hodnoty je vhodné využít statickou metodu `calculateSignalLevel(int, int)` třídy `WifiManager`. Jejím prvním parametrem je získaná síla signálu a druhým je počet úrovní. Metoda vrací odpovídající úroveň od nuly po zvolený počet úrovní mínus jedna (viz také výpis B.16).

Detaily o aktivním připojení je možné získat metodou `getConnectionInfo()` třídy `WifiManager`, viz výpis B.16. Metoda vrací instanci třídy `WifiInfo`, která obsahuje např. aktuální sílu signálu, rychlost připojení, název přístupového bodu (SSID) nebo jeho IP a MAC adresu.

Výpis B.16: Detaily o aktivním Wi-Fi připojení.

```
// Získání informací o připojení.
WifiInfo info = wifi.getConnectionInfo();

// Není-li připojeno, je BSSID rovno null.
if (info.getBSSID() != null) {
    // Získání úrovně síly signálu (stupnice 0-4).
    int signalLvl = WifiManager.calculateSignalLevel(info.getRssi(), 5);

    // Rychlost připojení a její jednotky.
    int speed = info.getLinkSpeed();
    String speedUnits = WifiInfo.LINK_SPEED_UNITS;

    // Název přístupového bodu.
    String ssid = info.getSSID();

    // IP a MAC adresa.
    int ip = info.getIpAddress();
    String mac = info.getMacAddress();

    // ...
}
```

Správce Wi-Fi umožňuje spustit hledání přístupových bodů v okolí a získat jeho výsledky. Hledání je spuštěno metodou `startScan()`. Jelikož je hledání asynchronní, je třeba vytvořit a zaregistrovat Broadcast Receiver, pomocí něhož bude naše aplikace informována o dokončení vyhledávání a bude moci vyzvednut výsledky. Seznam nalezených hotspotů získáme metodou `getScanResults()`. Ukázkový kód je uveden ve výpisu B.17. Před spuštěním hledání je nejprve dynamicky zaregistrován (nepotřebuje deklaraci v manifestu aplikace) nový Broadcast Receiver na dokončení hledání (akce `SCAN_RESULTS_AVAILABLE_ACTION`), který vyzvedne jeho výsledky. V praxi by bylo třeba si instanci receiveru zapamatovat a později odregistrovat. Výsledky obsahují informace o nalezených přístupových bodech – síla signálu, rychlost, SSID a další.

Výpis B.17: Vyhledání Wi-Fi hotspotů v okolí.

```
// Tvorba a dynamická registrace Broadcast Receiveru.
// Metoda pro registraci je součástí abstraktní třídy Context.
registerReceiver(new BroadcastReceiver() {

    /**
     * Metoda je volána po dokončení hledání.
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Vyzvednutí výsledků hledání.
        List<ScanResults> results = wifi.getScanResults();
        // ...
    }

}, new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));

// Spuštění hledání.
wifi.startScan();
```

Správce Wi-Fi dále umožňuje spravovat nakonfigurované sítě. Jejich seznam lze získat metodou `getConfiguredNetworks()`. Každá konfigurace je reprezentována objektem třídy `WifiConfiguration`. Mezi sítěmi můžeme poté přepínat metodou `enableNetwork(int, boolean)`. První parametr je ID sítě, které lze získat z její konfigurace, druhý parametr je příznak, zda odpojit všechny ostatní sítě (musí být `true`, chceme-li přepnout na jinou síť). Můžeme kdykoliv vytvořit novou konfiguraci Wi-Fi sítě. Metodami `addNetwork(WifiConfiguration)`, `updateNetwork(WifiConfiguration)` a `removeNetwork(int)` lze konfigurace přidávat, upravovat a mazat.

B.6 Wi-Fi Direct

Pro použití Wi-Fi Direct na Androidu 4.0 (Ice Cream Sandwich) musí být tato funkce nejprve zapnuta. Jedinou možností, jak zapnutí provést z aplikace, je nechat uživateli zobrazit obrazovku nastavení bezdrátových sítí systému, kde může ručně funkci zapnout (viz výpis B.18). Funkce se pak po krátké době nečinnosti sama vypne. Aktuální stav (zapnuto/vypnuto) můžeme v aplikaci monitorovat pomocí Broadcast Receiveru na akci `WIFI_P2P_STATE_CHANGED_ACTION`. Od Androidu 4.1 (Jelly Bean) se ale situace změnila. Postačí již pouze zapnutá Wi-Fi jako taková. Tu můžeme zapnout/vypnout čistě programově, bez nutnosti interakce s uživatelem, viz výpis B.15. Jak inicializovat Wi-Fi Direct je uvedeno ve výpisu 3.5 na str. 29.

Výpis B.18: Zapnutí funkce Wi-Fi Direct na Androidu 4.0 (Ice Cream Sandwich).

```
String action = android.provider.Settings.ACTION_WIRELESS_SETTINGS;
startActivity(new Intent(action));
```

Výpis B.19: Wi-Fi Direct – hledání dostupných zařízení.

```
// 1) Tvorba a registrace receiveru pro získání dostupných zařízení.
registerReceiver(new BroadcastReceiver() {

    /**
     * Metoda je volána při změně seznamu dostupných zařízení.
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Získání výsledků hledání.
        wifiDirect.requestPeers(channel, new PeerListListener() {

            public void onPeersAvailable(WifiP2pDeviceList peers) {
                // Získání aktuálního seznamu.
                List<WifiP2pDevice> devs = peers.getDeviceList();
                // ...
            }

        });
    }

}, new IntentFilter(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION));

// 2) Spuštění hledání dostupných zařízení.
wifiDirect.discoverPeers(channel, null);
```

Hledání dostupných zařízení je uvedeno ve výpisu B.19. Pro jeho spuštění je využita metoda `discoverPeers(Channel, ActionListener)`. Prvním parametrem je kanál, pomocí druhého lze pak definovat `ActionListener`, kterým můžeme detekovat, zda operace skončila úspěchem či neúspěchem (pro stručnost a přehlednost není ve výpisu uveden). Jelikož je metoda asynchronní, je opět potřeba vytvořit Broadcast Receiver, přes který bude aplikace informována o možnosti získání seznamu dostupných zařízení. V tomto případě se jedná spíše o aktualizace, neboť akce je vysílána pokaždé, změní-li se seznam dostupných zařízení. Hledání je aktivní tak dlouho, dokud není ustaveno spojení. Na Androidu 4.1 a vyšším lze hledání explicitně ukončit metodou `stopPeerDiscovery(Channel, ActionListener)`. Pro získání nalezených zařízení slouží metoda `requestPeers(Channel, PeerListListener)`, která je opět asynchronní. Parametry jsou kanál a `PeerListListener`, přes jehož metodu `onPeersAvailable(WifiP2pDeviceList)` je vrácen seznam dostupných zařízení, jakmile je připraven. Metodou `getDeviceList()` získáme kýžený seznam, kde je každé zařízení reprezentováno instancí třídy `WifiP2pDevice`.

Po získání seznamu dostupných zařízení je dalším krokem připojení. Ve výpisu B.20 je uvedena metoda `connectTo`, která vyšle požadavek na spojení se zařízením předaným jako parametr. Nejprve je třeba vytvořit konfiguraci, což spočívá ve vytvoření nové instance třídy `WifiP2pConfig` a zkopírováním adresy připojovaného zařízení. Poté již může být zavolána metoda `connect(Channel, WifiP2pConfig, ActionListener)`. Na zařízení, ke kterému se připojujeme je zobrazen dialog vyžadující potvrzení spojení uživatelem.

Výpis B.20: Wi-Fi Direct – požadavek na spojení se zařízením.

```
void connectTo (WifiP2pDevice device) {
    // Vytvoření konfigurace (adresa připojovaného zařízení).
    WifiP2pConfig config = new WifiP2pConfig();
    config.deviceAddress = device.deviceAddress;

    // Požadavek na spojení.
    wifiDirect.connect(channel, config, null);
}
```

Při úspěšném připojení je na obou zařízeních broadcastována akce `WIFI_P2P_CONNECTION_CHANGED_ACTION`. Extra data intentu obsahují položku `EXTRA_NETWORK_INFO`, která uchovává instanci třídy `NetworkInfo`. Ta obsahuje metodu `isConnected()`, podle jejíž návratové hodnoty poznáme, zda jsme připojeni nebo odpojeni. Jsme-li připojeni, můžeme se dotázat na detaily spojení metodou `requestConnectionInfo(Channel, ConnectionInfoListener)`. Parametry jsou kanál a `ConnectionInfoListener` přes jehož metodu `onConnectionInfoAvailable(WifiP2pInfo)` je vrácena instance třídy `WifiP2pInfo`, jakmile je připravena. Jednu z nejdůležitějších informací udává veřejný atribut `isGroupOwner`. S jeho pomocí v aplikaci rozlišíme, které zařízení je v roli vlastníka skupiny, a které je pouze členem skupiny. Typicky pak platí, že vlastník skupiny funguje jako server (vytváří server socket) a člen jako klient (vytváří klientský socket). Výše popsaný postup je uveden ve výpisu B.21.

Komunikace (výměna dat) probíhá pomocí standardních socketů a input/output streamů jazyka Java. Server vytvoří server socket a čeká na příchozí spojení, viz výpis B.22. Klient vytvoří klasický socket a připojí se na server, viz výpis B.23. Výměna dat pak může probíhat oběma směry nezávisle na tom, kdo je v roli klienta a serveru. Je třeba poznamenat, že obě operace (`accept()` i `connect()`) jsou blokující a tudíž se musíme vyvarovat jejich volání v hlavním vlákně aplikace.

Výpis B.21: Wi-Fi Direct – změna stavu připojení.

```
// Tvorba a registrace receiveru pro detekci změny stavu připojení.
registerReceiver(new BroadcastReceiver() {

    /**
     * Metoda je volána při změně stavu připojení (připojeno/nepřipojeno).
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Zjištění aktuálního stavu.
        String extraKey = WifiP2pManager.EXTRA_NETWORK_INFO;
        NetworkInfo networkInfo =
            (NetworkInfo) intent.getParcelableExtra(extraKey);

        // Připojeno?
        if (networkInfo.isConnected()) {
            // Získání detailních informací o připojení.
            wifiDirect.requestConnectionInfo(channel,
                new ConnectionInfoListener() {

                public void onConnectionInfoAvailable(WifiP2pInfo info) {
                    // Znovu ověříme spojení.
                    if (info.groupFormed) {
                        // Rozlišení serveru a klienta.
                        if (info.isGroupOwner) {
                            // Server socket ...

                        } else {
                            // Klientský  socket ...
                        }
                    }
                }

            });
        }
    }

}, new IntentFilter(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION));
```

Výpis B.22: Wi-Fi Direct – spojení a komunikace na straně serveru.

```
// Vytvoření socketu a čekání na spojení na daném portu.
int port = 8666;
ServerSocket serverSocket = new ServerSocket(port);
Socket client = serverSocket.accept();

// Komunikace.
InputStream in = client.getInputStream();
OutputStream out = client.getOutputStream();
// ...
```

Výpis B.23: Wi-Fi Direct – spojení a komunikace na straně klienta.

```
// Parametry spojení - timeout, port a adresa serveru
int timeout = 10000;
int port = 8666;
String address = ... ;  // Například z WifiP2pInfo#groupOwnerAddress.
InetSocketAddress socketAddress = new InetSocketAddress(address, port);

// Vytvoření socketu a spojení.
Socket socket = new Socket();
socket.bind(null);
socket.connect(socketAddress, timeout);

// Komunikace.
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
// ...
```

B.7 Bluetooth

Získání správce Bluetooth je uvedeno ve výpisu 3.7 na str. 30. Chceme-li zapnout Bluetooth, máme dvě možnosti. První zahrnuje interakci s uživatelem. Je uvedena ve výpisu B.24. Zde vytvořená metoda `enableBluetoothRequest()` spustí systémovou aktivitu dotazující se uživatele na zapnutí Bluetooth (má podobu dialogu). Výsledek se můžeme dozvědět přepsáním metody `onActivityResult(int, int, Intent)`, která je zavolána jakmile uživatel provede volbu. Podle předaných hodnot `requestCode` a `resultCode` zjistíme, zda se Bluetooth podařilo zapnout. Druhý způsob zahrnuje možnost zapnutí i vypnutí Bluetooth a nevyžaduje přitom zásah uživatele. Spočívá ve využití správce Bluetooth a jeho metod `enable()` a `disable()`, viz výpis B.25.

Výpis B.24: Dotaz na zapnutí Bluetooth.

```
// Předpokládáme umístění kódu uvnitř aktivity.
private static final int BT_ENABLE_REQUEST_CODE = 1;

// Zobrazení žádosti o zapnutí Bluetooth (dialog).
void enableBluetoothRequest () {
    Intent i = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(i, BT_ENABLE_REQUEST_CODE);
}

// Zjištění, zda uživatel souhlasil se zapnutím Bluetooth.
@Override protected
void onActivityResult (int requestCode, int resultCode, Intent data) {
    if (requestCode == BT_ENABLE_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Bluetooth bylo zapnuto.
            // ...
        }
    }
}
```

Jelikož zapínání/vypínaní Bluetooth jsou asynchronní operace, jejichž délka provádění se může pohybovat v rámci jednotek sekund, můžeme vytvořit Broadcast Recei-

ver pro monitorování změn stavu adaptéru. Příslušná akce je reprezentována konstantou ACTION_STATE_CHANGED. Intent, který je při změně stavu broadcastován obsahuje extra data, jež informují o aktuálním (EXTRA_STATE) a předchozím (EXTRA_PREVIOUS_STATE) stavu adaptéru.

Výpis B.25: Zapnutí/vypnutí Bluetooth bez nutnosti interakce s uživatelem.

```
void setBtEnabled (boolean enabled) {
    if (enabled) btAdapter.enable();    // Zapnutí Bluetooth.
    else btAdapter.disable();           // Vypnutí Bluetooth.
}
```

Aby bylo možné zařízení vyhledat v ostatních zařízeních, musí být povolena jeho viditelnost. Zda je zařízení viditelné můžeme zjistit metodou `getScanMode()`, která vrací jednu z následujících konstant:

- SCAN_MODE_NONE (výchozí) — zařízení není viditelné, žádné jiné jej v seznamu dostupných zařízení nebude obsahovat.
- SCAN_MODE_CONNECTABLE — zařízení, která byla s tímto již dříve spárována, uvidí zařízení v seznamu dostupných. Jiná jej ale nenaleznou.
- SCAN_MODE_CONNECTABLE_DISCOVERABLE — zařízení je viditelné pro všechna Bluetooth zařízení.

Jedinou možností, jak veřejné API Andoridu dovoluje zapnutí viditelnosti z aplikace, je zobrazení systémové dialogové aktivity, která si vyžádá potvrzení od uživatele, viz výpis B.26. Volbu uživatele se můžeme dozvědět opět v přepsané metodě `onActivityResult(int, int, Intent)`.

Výpis B.26: Dotaz na zapnutí viditelnosti Bluetooth zařízení.

```
// Předpokládáme umístění kódu uvnitř aktivity.
private static final int BT_DISCOVERY_REQUEST_CODE = 2;

// Zobrazení žádosti o zapnutí viditelnosti (dialog).
void enableDiscoveryRequest () {
    Intent i = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    // Nastavení doby viditelnosti zařízení (v sekundách).
    i.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 120);
    startActivityForResult(i, BT_DISCOVERY_REQUEST_CODE);
}

// Zjištění, zda uživatel souhlasil se zapnutím viditelnosti.
@Override protected
void onActivityResult (int requestCode, int resultCode, Intent data) {
    if (requestCode == BT_DISCOVERY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {
            // Viditelnost bylo povolena.
            // ...
        }
    }
}
```

Přejeme-li si monitorovat stav viditelnosti, můžeme vytvořit Broadcast Receiver na akci ACTION_SCAN_MODE_CHANGED. Extra data intentu pak informují o aktuálním (EXTRA_SCAN_MODE) a předchozím (EXTRA_PREVIOUS_SCAN_MODE) stavu viditelnosti zařízení.

Nyní se zaměříme na vyhledávání dostupných Bluetooth zařízení v okolí. Proces hledání můžeme spustit metodou startDiscovery() a ukončit metodou cancelDiscovery(). Hledání je asynchronní operace, může trvat až 12 s a velmi zatěžuje Bluetooth adaptér. Během hledání by se tedy neměly provádět jiné operace, např. připojování nového zařízení nebo přenos dat. Zda právě probíhá hledání, můžeme ověřit metodou isDiscovering(). Začátek a konec hledání je navíc rozeslán broadcastem jako akce ACTION_DISCOVERY_STARTED a ACTION_DISCOVERY_FINISHED.

Každé Bluetooth zařízení je reprezentováno instancí třídy BluetoothDevice. Pro získání výsledků hledání okolních zařízení je třeba vytvořit Broadcast Receiver na akci BluetoothDevice.ACTION_FOUND. Každé nalezené zařízení je tímto broadcastem ohlášeno zvlášť. Extra data intentu obsahují položky EXTRA_DEVICE, EXTRA_CLASS, a jsou-li dostupné, také EXTRA_NAME a EXTRA_RSSI. Položka EXTRA_DEVICE obsahuje objekt třídy BluetoothDevice, jenž popisuje dané zařízení. Položka EXTRA_CLASS obsahuje instanci třídy BluetoothClass, která obsahuje obecnou charakteristiku zařízení a jeho schopností, např. se může jednat o headset se schopností přenášet audio signál a provádět telefonní hovory (příslušné konstanty lze nalézt ve třídách BluetoothClass.Device a BluetoothClass.Service). Položky EXTRA_NAME a EXTRA_RSSI pak udávají jméno Bluetooth zařízení a sílu signálu. Ukázka kódu pro vyhledání dostupných Bluetooth zařízení je uvedena ve výpisu B.27.

Výpis B.27: Vyhledání dostupných Bluetooth zařízení v okolí.

```
// Seznam pro nalezená zařízení.
ArrayList<BluetoothDevice> devs = new ArrayList<BluetoothDevice>();

void startBtDiscovery() {
    // Zaregistrování receiveru pro získání výsledků hledání.
    registerReceiver(discoveryResult,
        new IntentFilter(BluetoothDevice.ACTION_FOUND));

    // Spuštění hledání.
    btAdapter.startDiscovery();
}

// Vytvoření Broadcast Receiveru pro získání výsledků hledání.
BroadcastReceiver discoveryResult = new BroadcastReceiver() {

    /**
     * Metoda je volána při nalezení Bluetooth zařízení.
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        // Získání zařízení
        String extraName = BluetoothDevice.EXTRA_DEVICE;
        BluetoothDevice d = intent.getParcelableExtra(extraName);

        // Vložení zařízení do seznamu.
        if (d != null && d.getName() != null) devs.add(d);
    }

}
```

Implementace Bluetooth na Androidu využívá komunikační protokol RFCOMM, který umožňuje spojení a komunikaci dvou spárovaných zařízení. Pokud nejsou při pokusu o spojení zařízení spárována, je automaticky zobrazen systémový dialog, kterým musí uživatel na obou stranách potvrdit párování. Množinu spárovaných zařízení je možné získat metodou `getBondedDevices()`. Spojení probíhá přes sockety reprezentované třídami `BluetoothServerSocket` a `BluetoothSocket`. Třída `BluetoothServerSocket` se chová jako server a slouží pro naslouchání na příchozí spojení. Třída `BluetoothSocket` je využita na straně klienta pro připojení se k serveru. Jakmile je spojení ustaveno, může výměna dat probíhat oběma směry nezávisle na tom, kdo je v roli klienta a serveru.

Ukázka spojení a komunikace na straně serveru je uvedena ve výpisu B.28. Instanci třídy `BluetoothServerSocket` získáme voláním metody `listenUsingRfcommWithServiceRecord(String, UUID)`, které předáme jméno serveru a jedinečný identifikátor (UUID, Universally Unique Identifier). Toto UUID bude posléze využito na straně klienta pro připojení se k tomuto serveru. Čekaní na příchozí spojení započneme voláním metody `accept()`. Ta je blokující a je tedy nutné vyvarovat se jejího volání v hlavním vlákně aplikace. Po úspěšném připojení klienta vrátí metoda instanci třídy `BluetoothSocket`, ze které získáme input a output streamy pro výměnu dat mezi oběma zařízeními.

Výpis B.28: Bluetooth – spojení a komunikace na straně serveru.

```
// Příprava - název serveru a jedinečné UUID.
String name = "BluetoothServer";
UUID uuid = UUID.fromString("4f0bd510-6d46-11e2-bcfd-0800200c9a66");

// Vytvoření server socketu.
BluetoothServerSocket btServer =
    btAdapter.listenUsingRfcommWithServiceRecord(name, uuid);

// Naslouchání na příchozí spojení.
BluetoothSocket socket = btServer.accept();

// Komunikace.
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
// ...
```

Ukázka spojení a komunikace na straně klienta je uvedena ve výpisu B.29. Nejprve musíme získat instanci třídy `BluetoothDevice` reprezentující zařízení, ke kterému se chceme připojit. Získat ji můžeme buď pomocí hledání okolních zařízení (viz výše), nebo, známe-li MAC adresu Bluetooth zařízení, metodou `getRemoteDevice(String)`. Instanci třídy `BluetoothSocket` získáme voláním metody `createRfcommSocketToServiceRecord(UUID)`, které předáme UUID serveru. Spojení iniciujeme metodou `connect()`. Ta je opět blokující a je tedy nutné vyvarovat se jejího volání v hlavním vlákně aplikace. Po úspěšném připojení získáme input a output streamy pro výměnu dat.

Chceme-li se vyhnout nutnosti párovat zařízení, můžeme na Androidu 2.3.3 nebo vyšším využít tzv. *insecure* spojení. To vytvoříme voláním metod `listenUsingInsecureRfcommWithServiceRecord(String, UUID)` na straně serveru a `createInsecureRfcommSocketToServiceRecord(UUID)` na straně klienta.

Výpis B.29: Bluetooth – spojení a komunikace na straně klienta.

```
// Zařízení, ke kterému se připojujeme a UUID serveru.
String btServerMac = ... ;
BluetoothDevice device = btAdapter.getRemoteDevice(btServerMac);
UUID uuid = UUID.fromString("4f0bd510-6d46-11e2-bcfd-0800200c9a66");

// Vytvoření klientského socketu.
BluetoothSocket socket = device.createRfcommSocketToServiceRecord(uuid);

// Spojení se serverem.
socket.connect();

// Komunikace.
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
// ...
```

Příloha C

Ukázka práce se senzory na platformě Android

Ve výpisu C.1 je uvedena ukázka kódu, který pracuje se senzory na platformě Android. Nejprve musíme, voláním metody `getSystemService(String)`, vytvořit instanci správce senzorů. Pomocí něj pak můžeme získat instance senzorů, o jejichž data máme zájem. V ukázkovém kódu jsme pomocí metody `getDefaultSensor(int)` získali akcelerometr, gyroskop, senzor přiblížení a senzor osvětlení (konstanty viz tabulka 4.1 na str. 36). Pokud některý senzor není na zařízení přítomen, vrací metoda `null`.

Dále definujeme listener, který musí implementovat rozhraní `SensorEventListener` (viz podkapitola 4.2 na str. 36). Metoda `onSensorChanged(SensorEvent)` je volána vždy ve chvíli, kdy jsou k dispozici nová data ze senzoru. V ní rozpoznáme, o jaký senzor se jedná, a v případě akcelerometru přečteme aktuálně naměřené hodnoty zrychlení podél osy x, y a z. S těmi poté můžeme dále pracovat. Metodou `onAccuracyChanged(Sensor, int)` můžeme detekovat změnu přesnosti některého ze senzorů (konstanty viz tabulka 4.3 na str. 38).

Snímání dat spustíme zaregistrováním listeneru na daný senzor metodou `registerListener(SensorEventListener, Sensor, int)`. Pro první parametr použijeme dříve vytvořený listener, druhým parametrem je senzor a třetím vzorkovací frekvence, se kterou si přejeme data ze senzoru získávat. Frekvence je ale pouze orientační, v praxi si ji systém upravuje podle potřeby (více viz podkapitola 4.2 a tabulka 4.2 na str. 37).

Zastavení snímání dat ze senzorů provedeme odregistrací listeneru metodou `unregisterListener(SensorEventListener)`. Musíme tak učinit co nejdříve je to možné, abychom neplýtvali systémovými zdroji a energií. Typická je registrace v metodě `onResume()` a odregistrace v metodě `onPause()` životního cyklu aktivity.

Výpis C.1: Snímání dat ze senzorů.

```
// Získání správce senzorů. Předpokládáme volání metody getSystemService()
// ve třídě odvozené od třídy Context (např. aktivita nebo služba).
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

// Získání senzorů. Vrátí null, pokud není senzor na zařízení přítomen.
Sensor accSen   = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor gyroSen  = sm.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
Sensor proxSen  = sm.getDefaultSensor(Sensor.TYPE_PROXIMITY);
Sensor lightSen = am.getDefaultSensor(Sensor.TYPE_LIGHT);
// ...
```

```
// Vytvoření listeneru pro snímání dat ze senzorů.
SensorEventListener listener = new SensorEventListener () {

    /**
     * Callback pro obdržení nových dat ze senzoru.
     */
    @Override
    public void onSensorChanged(SensorEvent event) {
        switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            // Nové hodnoty naměřené akcelerometrem.
            float accX = event.values[0];
            float accY = event.values[1];
            float accZ = event.values[2];
            // Zpracování hodnot...
            break;

        // Další senzory...

        }
    }

    /**
     * Callback pro zjištění změny přesnosti senzoru.
     */
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        switch (accuracy) {
        case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
            // ...
            break;

        case SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM:
            // ...
            break;

        case SensorManager.SENSOR_STATUS_ACCURACY_LOW:
            // ...
            break;

        case SensorManager.SENSOR_STATUS_UNRELIABLE:
            // ...
            break;
        }
    }
};

// Spuštění snímání dat ze senzorů.
sm.registerListener(listener, accSen, SensorManager.SENSOR_DELAY_FASTEST);
sm.registerListener(listener, gyroSen, SensorManager.SENSOR_DELAY_GAME);
sm.registerListener(listener, proxSen, SensorManager.SENSOR_DELAY_UI);
sm.registerListener(listener, lightSen, SensorManager.SENSOR_DELAY_NORMAL);
// ...

// Zastavení snímání dat ze senzorů.
sm.unregisterListener(listener);
```

Příloha D

Ukázka práce s GPS na platformě Android

Ve výpisu D.1 je uvedena ukázka kódu, který pracuje s GPS na platformě Android. Přesněji slouží ke zjištění polohy zařízení za pomocí satelitů systému GPS. Nejprve musíme, voláním metody `getSystemService(String)`, vytvořit instanci správce polohy. Také nesmíme zapomenout na deklaraci patřičného oprávnění, viz výpis 4.2 na str. 38.

Dále definujeme listener, který musí implementovat rozhraní `LocationListener`. Jeho metoda `onLocationChanged(Location)` je volána vždy ve chvíli, kdy je k dispozici nová poloha zařízení. Z předané instance třídy `Location` můžeme zjistit kromě polohy také nadmořskou výšku, rychlost či přesný čas. Metody `onProviderDisabled(String)`, `onProviderEnabled(String)` a `onStatusChanged(String, int, Bundle)` můžeme využit pro detekci změny stavu poskytovatele polohy (v našem případě je poskytovatelem GPS).

Metodou `requestLocationUpdates(String, long, float, LocationListener)` spustíme periodické snímání polohy. Můžeme si ale podle potřeby vybrat z několika jejích přetížených variant. Prvním parametrem je specifikace poskytovatele. V našem případě máme zájem o polohu z GPS, proto využijeme konstantu `GPS_PROVIDER`. Jako druhý a třetí argument pak musíme zadat minimální interval (v milisekundách) a minimální vzdálenost (v metrech) mezi aktualizacemi. Dosazení nuly za obě hodnoty má za následek získávání polohy jak nejrychleji to jen jde, což ale v praxi není vhodné z hlediska spotřeby energie. Posledním parametrem je dříve vytvořený listener. Snímání polohy zastavíme metodou `removeUpdates(LocationListener)`, které opět předáme vytvořený listener.

Výpis D.1: Zjištění polohy pomocí GPS.

```
// Získání správce polohy. Předpokládáme volání metody getSystemService()
// ve třídě odvozené od třídy Context (např. aktivita nebo služba).
String service = Context.LOCATION_SERVICE;
LocationManager lm = (LocationManager) getSystemService(service);

// Vytvoření listeneru pro aktualizace z GPS.
LocationListener listener = new LocationListener () {
```

```
    /**
     * Callback pro obdržení nové polohy.
     */
    @Override
    public void onLocationChanged(Location location) {
        // Získání informací o poloze, nadmořské výšce, rychlosti a čase.
        double latitude  = location.getLatitude()
        double longitude = location.getLongitude()
        double altitude  = location.getAltitude()
        float speed      = location.getSpeed()
        long time        = location.getTime()
        // Zpracování hodnot...
    }

    /**
     * Voláno při zakázání poskytovatele polohy (GPS) uživatelem.
     */
    @Override
    public void onProviderDisabled(String provider) {
        // ...
    }

    /**
     * Voláno při povolení poskytovatele polohy (GPS) uživatelem.
     */
    @Override
    public void onProviderEnabled(String provider) {
        // ...
    }

    /**
     * Voláno při změně stavu poskytovatele polohy (GPS), např. může být
     * dočasně nedostupný.
     */
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras){
        // ...
    }

};

// Spuštění snímání polohy z GPS.
String provider = LocationManager.GPS_PROVIDER; // Poloha z GPS.
long minTime = 0;           // Frekvence (v ms).
float minDistance = 0;      // Vzdálenost mezi měřeními (v m);
lm.requestLocationUpdates(provider, minTime, minDistance, listener);

// ...

// Zastavení snímání polohy.
lm.removeUpdates(listener);
```

Příloha E

Uživatelské rozhraní a nastavení uzlu vytvořené WSN

V této příloze se blíže zaměříme na grafické uživatelské rozhraní aplikace implementující uzel vytvořené WSN, viz obrázek E.1. Popíšeme jednotlivé stránky nastavení a volby, které umožňují uživateli nastavit. První stránka, s titulkem *General*, obsahuje obecná nastavení uzlu:

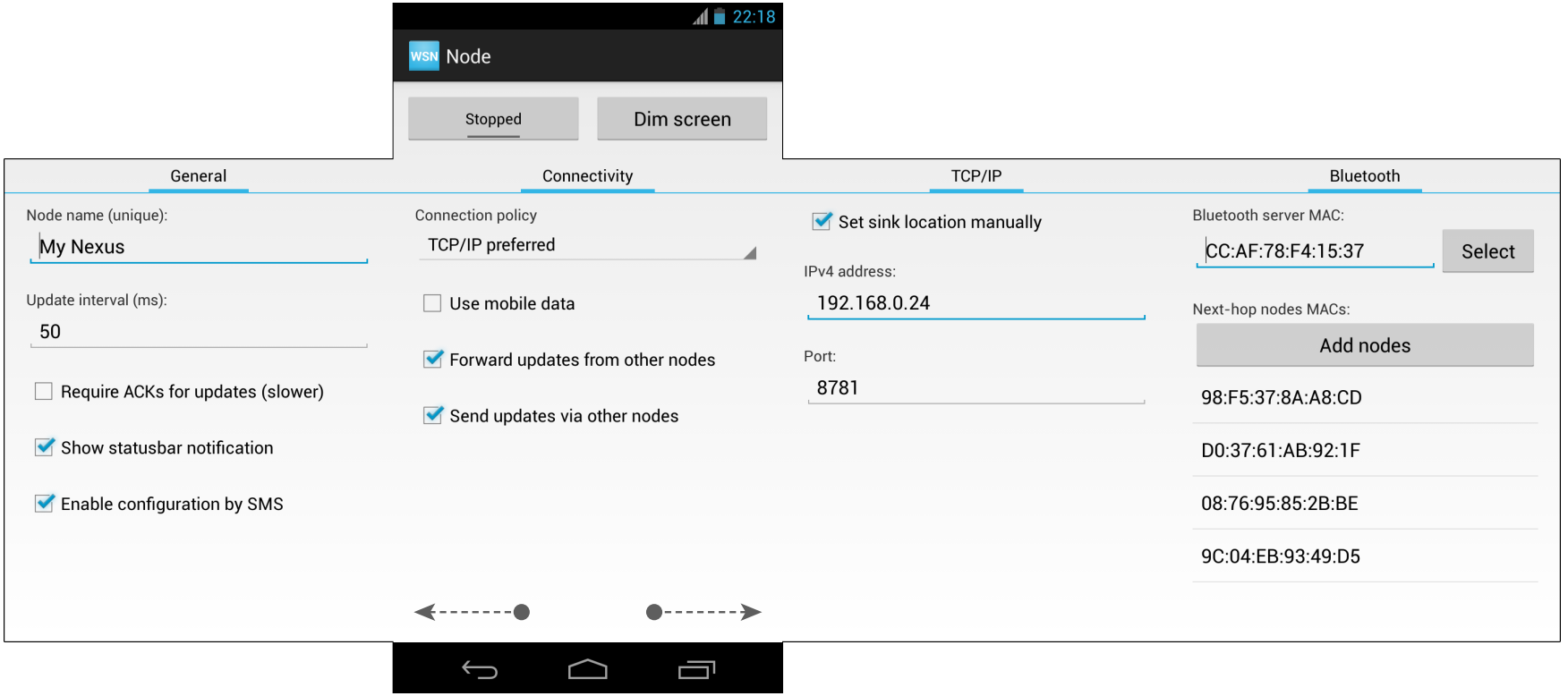
- Jedinečný název uzlu.
- Interval zasílání aktualizací v milisekundách.
- Zda má být úspěšné doručení aktualizací potvrzováno. Jde o volbu mezi typem zprávy `UPDATE` a `UPDATE_WITH_ACK`.
- Zda má být práce uzlu indikována notifikací ve stavové liště.
- Povolení vzdálené konfigurace přes SMS.

Druhá stránka, s titulkem *Connectivity*, obsahuje volby týkající se konektivity:

- Politika připojení — Pouze Bluetooth, Pouze TCP/IP nebo TCP/IP preferováno.
- Povolení využití mobilních dat.
- Povolení přeposílání zpráv od jiných uzlů (zda spouštět Bluetooth forwarding server).
- Zda umožnit připojení přes některý z nastavených next-hop uzlů.

Třetí stránka, s titulkem *TCP/IP*, umožňuje nastavit TCP/IP spojení se základnovou stanicí:

- Zda použít manuální nastavení serveru základnové stanice (jinak je využit mechanismus automatického nalezení). Je-li povoleno, pak je nutné zadat také následující dvě volby.
- IPv4 adresa základnové stanice.
- Číslo portu TCP/IP serveru základnové stanice.



Obrázek E.1: Uživatelské rozhraní uzlu.

Poslední čtvrtá stránka, s titulkem *Bluetooth*, umožňuje nastavit Bluetooth spojení se základnovou stanicí a next-hop uzly:

- Bluetooth MAC adresa základnové stanice. Nebo obecně MAC adresa zařízení, ke kterému se má uzel připojovat primárně.
- Množina next-hop uzlů ve formě seznamu Bluetooth MAC adres.

Bluetooth MAC adresy výše je možné zadat buď ručně, nebo výběrem ze spárovaných nebo aktuálně viditelných zařízení v okolí. Klepnutím na MAC adresu next-hop uzlu a potvrzením zobrazeného dialogu je možné ji ze seznamu odstranit.