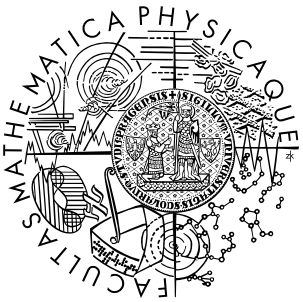
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Kouba

Simulace směrování v bezdrátových sítích

Katedra aplikované matematiky (KAM)

Vedoucí bakalářské práce: Prof. RNDr. Luděk Kučera, DrSc.
Studijní program: Informatika, obor obecná informatika

2006

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 12. 6. 2006 Jan Kouba

Obsah

Název práce: Simulace směrování v bezdrátových sítích
Autor: Jan Kouba
Katedra (ústav): Katedra aplikované matematiky (KAM)
Vedoucí bakalářské práce: Prof. RNDr. Luděk Kučera, DrSc.
e-mail vedoucího: ludek@kam.mff.cuni.cz

Abstrakt: Práce se zabývá vytvořením systému pro simulaci směrování v bezdrátových sítích. Systém umožňuje uživateli ručně zadávat parametry přenosu (vysílací výkony uzlů a trasy kanálů) a z nich provádí analýzu sítě. Systém umožňuje automatické určování parametrů přenosu tak, že se nejprve určí vysílací výkony a z nich pak trasy kanálů. Systém byl navržen s důrazem na snadné přidávání algoritmů. Program je zaměřen na uživatelskou přívětivost, snadné ovládání a dobrou rozšiřitelnost. Úkolem práce nebylo navrhnout co nejpřesnější model skutečné bezdrátové sítě, ale vytvořit systém, v kterém by bylo možné vyzkoušet nové algoritmy a zjistit, které by mohly být perspektivní a které ne. V práci je popsána architektura programu tak, aby mohl někdo další pokračovat v jeho vývoji. Součástí práce je návod jak program zkompilovat a používat.

Klíčová slova: bezdrátové sítě, směrování, multihop, simulace

Title: Routing simulation in wireless networks
Author: Jan Kouba
Department: Department of Applied Mathematics
Supervisor: Prof. RNDr. Luděk Kučera, DrSc.
Supervisor's e-mail address: ludek@kam.mff.cuni.cz

Abstract: This work deals with creation of system for routing simulation in wireless networks. System allows the user to enter relevant parameters for simulation and analyses the network. System can assign simulation parameters automaticaly based on chosen algorithm. First it counts node output powers and second communication routes. It is easy to add new algorithms. System is focused on user friendliness and good extensionality. The objective was not to implement realistic model of real network, but to create system where could be new algorithms tested. There is described architecture of the system in the work, so one can continue with development.

Keywords: wireless network, routing, multihop, simulation

Kapitola 1

Úvod

V dnešní době je bezdrátovým „ad hoc“ sítím věnována stále větší pozornost. Vývoj směřuje ke snadno nasaditelným (za žádné, nebo minimální konfigurace), dobře škálovatelným a hlavně distribuovaným bezdrátovým sítím s komunikací stylu „multihop“ (komunikace mezi dvěma účastníky může být přenášena přes jeden nebo více dalších uzlů sítě). Tyto sítě by měly být schopny pracovat v podmínkách, kde je nasazení pevně budovaných sítí náročné, nebo nemožné. Ke komunikaci je mohou využívat jak multimediální zařízení v domácnosti, tak i bojové jednotky na bitevním poli.

Protože síť nemá žádný centrální prvek, je třeba veškerou konfiguraci provádět pomocí distribuovaných algoritmů. Základním parametrem, který bude ovlivňovat fungování sítě, je vysílací výkon uzlů. Při vysokých vysílacích výkonech je možné komunikovat se vzdálenějšími uzly, ale zase dochází k většímu rušení vysílání ostatních uzlů. Také je možné, že uzel bude napájen z baterií, takže by se mělo energii šetřit. Naopak uzel vysílající s nízkým výkonem je více rušen vysíláním okolních uzlů, z čehož plyne nižší přenosová rychlost mezi ním a příjemcem signálu. Na druhou stranu zase tolik nerušíme komunikaci ostatních, takže mohou zároveň vysílat i jiné uzly.

1.1 Cíl práce

Cílem práce je vytvořit systém pro simulaci směrování v bezdrátových „multihop“ sítích. Systém bude umožňovat nastavení parametrů komunikace (polohy uzlů, vysílací výkony uzlů, trasy kanálů) a analýzu vlastností přenosu (úroveň rušení a z toho vyplývající dosažitelný komunikační výkon spojů a kanálů).

Zadat vysílací výkony uzlů a nastavit trasy kanálů půjde buď přímo, nebo půjde určit algoritmus, který je vypočítá.

Analýza přenosu bude probíhat tak, že se nejprve určí rušení spojů a

z nich maximální přenosový výkon. Nakonec se určí přenosové výkony kanálů.

Systém bude navržen tak, aby jej bylo možné snadno rozšířit o nové algoritmy pro určování vysílacích výkonů.

Zadávání parametrů simulace a zobrazování výsledků bude realizováno pomocí grafického rozhraní, které bude umožňovat uložení práce do souboru a opětovné načtení, stejně jako operace Undo/Redo.

Systém bude naprogramován v jazyce C++.

Kapitola 2

Model bezdrátové „multihop“ sítě

Jako první je třeba si položit otázku, jak vlastně bezdrátové „multihop“ sítě fungují a jak takovou síť budeme v programu modelovat.

V reálných bezdrátových sítích je k přenosu dat mezi uzly používáno sdílené médium „éter“. Je tedy třeba určit pravidla pro přistupování k tomuto médiu, která musí dodržovat všichni účastníci sítě. Protože odsimulovat reálný provoz v síti je velice obtížné, použijeme v programu značné zjednodušení.

Budeme předpokládat, že všechny uzly vysílají ve stejném frekvenčním pásmu, a tedy u vysílajících uzlů nás bude zajímat pouze vysílací výkon.

2.1 Určování možných příjemců signálu

Pro určování vysílacích výkonů a tras kanálů potřebujeme vědět, jak z vysílacího výkonu vysílače určit, s kterými přijímači může komunikovat přímo. Budeme předpokládat, že intenzita vysílaného signálu se snižuje s α mocninou vzdálenosti, kde $2 <= \alpha <= 4$. Vysílač u může přímo vysílat k přijímači v pokud $d^{\alpha}(u, v) <= P(u)$ (kde $d(u, v)$ je vzdálenost uzlů u a v a $P(u)$ udává vysílací výkon u.

2.2 Výpočet maximálního přenosového výkonu spoje

Dále musíme být schopni určit maximální přenosový výkon, kterého lze mezi dvěma přímo komunikujícími uzly dosáhnout. Pro potřeby programu použijeme následující zjednodušení. Maximální dosažitelný přenosový výkon $m_{u,v}$

mezi vysílačem u a přijímačem v určíme jako $m_{u,v} = T(\gamma)$, kde hodnota γ je odstup signálu od šumu vypočítaná jako:

$$\gamma = \frac{P(u) \cdot d^{-\alpha}(u,v)}{\sum_{m \in V \setminus \{u,v\}} P(m) \cdot d^{-\alpha}(u,v) + N}$$

V je množina všech vysílajících uzlů a N je konstanta určující šum prostředí.

T je funkce, která ze znalosti odstupu signálu od šumu určí maximální přenosový výkon. V programu jsou implementovány dvě takové funkce. První vychází z klasické Shannonovy kapacity Gaussovského kanálu s bílým šumem (v programu označována jako AWGN) a vypadá takto:

$$T_1(\gamma) = log_2(1 + \gamma)$$

druhá vychází z kapacity Rayleigh-Riceovského kanálu (v programu se označuje jako Rayleigh):

$$T_2(\gamma) = log_2(e) \cdot e^{1/\gamma}\Big(-E + ln(\gamma) - \sum_{k=1}^{\infty} \frac{(-1/\gamma)^k}{k \cdot k!}\Big)$$

kde $E = 0,5772$.

Druhá funkce dává o něco menší hodnoty než první.

2.3 Určování tras kanálů a vysílacích výkonů uzlů

Reálná síť by měla umět uspokojit požadavek na komunikaci mezi dvěma libovolnými uzly v síti. Není vhodné, aby při každém novém požadavku síť nějak razantně měnila svá nastavení (vysílací výkony). Používá se tedy strategie, kdy se nastaví vysílací výkony uzlů pevně tak, aby v celé síti byly dostupné všechny uzly a pak již komunikace probíhá po vzniklých spojích. Takový přístup je zvolen i v tomto programu.

Na uzly sítě se můžeme dívat jako na množinu bodů $M \subseteq \mathbb{R} \times \mathbb{R}$. Vysílací výkony pak určuje funkce $P : M \longrightarrow [0, +\infty)$. Funkce P již jednoznačně určuje orientovaný graf $G = (M, E)$, kde $(u, v) \in E$ právě když $d^{\alpha}(u, v) \leq P(u)$ ($d(u, v)$ je Eukleidovská vzdálenost mezi uzly u a v). Pokud hrana $(u, v) \in E$ znamená to, že uzel u může vysílat přímo k uzlu v. Běžným požadavkem v bezdrátových sítích je, aby graf G byl silně souvislý. Potom mohou komunikovat každé dva uzly sítě. To ale v programu požadováno není, protože nesouvislost grafu může být záměr uživatele.

Nyní se dostáváme k tomu, jak v programu probíhá automatické určování vysílacích výkonů a tras kanálů. Na počátku máme údaje o polohách uzlů a

množinu uspořádaných dvojic uzlů, které představují požadavky na komunikaci (tj. první a poslední uzly kanálu). Jako první se spustí algoritmus pro určování vysílacích výkonů. Pak se vezmou ručně nastavené uzly a změní se jejich výkony na uživatelem zadané hodnoty. Tedy nejprve se automaticky určí výkony *všech* uzlů a až potom se některé přepíší uživatelem zvolenými hodnotami. V programu jsou implementovány algoritmy *K-NNG* a statický algoritmus (viz. níže). Program byl navržen tak, aby přidání dalších algoritmů do programu bylo snadné (viz. 3.9).

Další krok je určení tras kanálů. Trasa kanálu začínajícího v u a končícího ve v se určí jako nejkratší cesta (co do počtu hran) z u do v v grafu G. Pokud taková cesta neexistuje (může se stát pokud G není silně souvislý), budou koncové uzly kanálu komunikovat přímo. (To by v reálném případě nebylo možné a i v programu bude vycházet maximální přenosový výkon velice malý, ale aspoň to upozorní uživatele, že něco není v pořádku.) Pokud trasu kanálu zadal uživatel ručně, použije se místo výše popsaného postupu ručně zadaná trasa. (Tady se také netestuje, zda kanál prochází pouze přes hrany grafu G.)

Algoritmus K-NNG

K-NNG (K nearest neighbour graph) je graf, ve kterém z každého vrcholu vede K orientovaných hran ke K nejbližším uzlům (různým od sebe sama). Algoritmus pracuje tak, že nejprve určí nejmenší n takové, že odpovídající n-NNG S už je silně souvislý. Vysílací výkon uzlu se pak určí jako $P(u) = y^\alpha$, kde y je délka nejdelší hrany vycházející z uzlu u v grafu S. Graf S je vlastně graf G (popsaný výše), který vznikne z vypočtených vysílacích výkonů. Z toho vyplývá, že pro takto vypočtené výkony je odpovídající graf G souvislý, tedy v něm lze najít cestu pro libovolný kanál.

Statický algoritmus

Tento algoritmus nastaví vysílací výkony všech uzlů na hodnotu zadanou uživatelem.

2.4 Analýza vlastností přenosu

Úkolem analýzy vlastností přenosu je určit maximální přenosové výkony kanálů.

Nejprve se určí maximální přenosové výkony spojů postupem uvedeným v 2.2. Pro naše potřeby budeme předpokládat, že se na rušení přenosu podílejí (patři do množiny V) všechny uzly, přes které vede nějaký kanál, nebo

které jsou zdrojovým uzlem kanálu (tedy uzly přes které žádné kanály neprocházejí, nebo cílové uzly kanálů žádné rušení nezpůsobují). Jinými slovy řečeno, uzel $u \in V$ právě když existuje aktivní spoj, ve kterém vystupuje uzel u jako vysílač. (Označení *aktivní spoj* budeme používat pro dvojice uzlů (u, v) takové, že uzel u předává zprávy nějakého kanálu přímo uzlu v.)

Maximální přenosový výkon kanálu se pak určí jako prosté minimum z maximálních přenosových výkonů aktivních spojů tvořících daný kanál.

Kapitola 3

Implementace

Tato část by měla pomoci pochopit strukturu programu do té míry, aby byl čtenář programátor schopen program upravit. Záměrně zde nejsou přesně uváděny parametry funkcí, typy návratových hodnot a detaily tříd. To všechno lze najít v dokumentaci vygenerované systémem Doxygen, nebo přímo ve zdrojových kódech.

Pokud program nainstalujete, najdete zdrojové kódy v adresáři `mus_package/project/`. V tomto adresáři je také soubor *mus.kdevelop* s projektem pro vývojové prostředí *KDevelop* (je třeba verze 3.2.1 nebo vyšší).

3.1 Platforma

Jako cílová platforma pro aplikaci MuS byla zvolena pracovní stanice typu PC s procesorovou architekturou IA-32 a s operačním systémem Linux. Systém musí mít grafické rozhraní (typicky x.org server) schopné provozovat aplikace používající knihovnu *Qt* a podporující OpenGL (alespoň v software modu). Jiné požadavky na hardware nepřekračují standard dnešních PC.

3.2 Použité knihovny

Program používá knihovnu Qt, některé knihovny ze sady knihoven *boost* (graph, serialization, type_traits, smart_ptr, property_map, static_assert) a knihovny OpenGL a Glut.

3.3 Architektura programu

Program je rozdělen do čtyř modulů: *env*, *ui*, *stat* a *ngl*.

Modul *env* tvoří jádro programu. Zde jsou uchovávána všechna uživatelem zadaná data o simulované síti, která jsou přístupná přes dobře definované a snadno použitelné rozhraní.

Modul *ui* představuje uživatelské rozhraní, které umožňuje zobrazovat, měnit a zpracovávat data uložená v modulu *env*. Tento modul je vystavěn na Qt.

V modulu *stat* jsou implementovány algoritmy pro počítání vysílacích výkonů uzlů a maximálních přenosových výkonů spojů. Obsahuje i některé pomocné třídy použité v algoritmech nebo v modulu *ui*.

Poslední je modul *ngl*. Zde je pouze několik funkcí a tříd pro snadnější práci s OpenGL.

Pokud nainstalujete program, naleznete zdrojové kódy v adresáři `mus_package/mus/project/src/`. Každý modulu se nachází ve stejnojmenném podadresáři (pouze modul *env* je v adresáři `environment/`). Všechny třídy a funkce patřící k modulu jsou schovány v namespace se jménem modulu.

3.4 Základní objekty simulace

V simulaci vystupují čtyři základní typy objektů:

- Uzly
- Vrstvy
- Kanály
- Aktivní spoje

Uzly představují účastníky sítě. Každý uzel může chtít komunikovat s jiným uzlem a umí přenášet zprávy jiných uzlů. Všechny uzly jsou stejné, tedy všechny mohou komunikovat a všechny musí přenášet komunikaci ostatních uzlů.

Kanál je posloupnost uzlů, která vymezuje, kudy putují zprávy mezi dvěma komunikujícími účastníky. Kanály jsou jednosměrné, tedy přenáší data jen v jednom směru, od prvního uzlu k poslednímu.

Aktivní spoj představuje přímé (orientované) spojení dvou uzlů. Mezi dvěma uzly je aktivní spoj, pokud leží v nějakém kanálu bezprostředně za sebou, tedy pokud mezi nimi není žádný mezilehlý uzel. Na kanál se tedy lze dívat také jako na posloupnost aktivních spojů (jeden aktivní spoj může používat i více kanálů). (viz. 2.4)

Program pracuje ještě s jedním typem objektů, s *vrstvami*. Vrstva nepředstavuje nic, co by bylo součástí simulované sítě. Umožňuje uživateli při

stejném rozestavení uzlů zkoumat různě nastavené přenosy. Každý kanál patří do nějaké vrstvy. V každé vrstvě lze mít vybrané jiné algoritmy pro určování parametrů přenosů a pro každou vrstvu se provádí analýza vlastností přenosu zvlášť. Co je nastaveno v jedné vrstvě neovlivňuje to, co je nastaveno ve vrstvě jiné. Jediné co vrstvy sdílejí jsou polohy uzlů a jejich názvy.

Každý uzel, kanál a vrstva mají jedinečný název (uzel mezi uzly, vrstva mezi vrstvami, kanál mezi kanály ve stejné vrstvě), které slouží ke snadné identifikaci objektu uživatelem.

3.5 Modul *env*

Modul *env* uchovává veškeré uživatelem zadané informace o simulaci. To zahrnuje uzly, vrstvy, kanály ve vrstvách, trasy kanálů a všechny jejich atributy. Umožňuje také efektivně procházet aktivní spoje.

Modul umožňuje tyto informace číst a měnit. Umožňuje ukládat simulované prostředí do souboru a opět ho načítat, vracet a znovu provádět změny (Undo/Redo).

Jako rozhraní slouží třída `Facade`. Tato třída umožňuje nastavovat parametry simulace a opět je číst. Je to jediná třída, ke které přistupují ostatní části programu používající modul *env*.

3.5.1 Datove struktury

Třída `Layer`

Zde jsou uloženy informace o vrstvě. Simulovaná síť je uchovávána v orientovaném grafu (`boost::adjacency_list`). Vrcholy představují uzly, hrany aktivní spoje. Ve vrcholech jsou uloženy lokální atributy uzlu (vysílací výkon) a ukazatel na globální atributy uzlu, ve hranách pak kanály, které jimi procházejí.

Kanály jsou uloženy v seznamu typu `std::list<ChannelProperties>`. V položkách jsou uloženy posloupnost ukazatelů na vrcholy, přes které kanál prochází, a atributy kanálu (název, barva, příznak zda se trasa určuje automaticky nebo ručně).

V třídě jsou také uloženy atributy vrstvy (název, typ a nastavení algoritmu který počítá vysílací výkony uzlů, typ a nastavení algoritmu který počítá maximální přenosové rychlosti, parametr α a šum prostředí).

Třída `Layer` se stará o to, aby byly tyto datové struktury konzistentní (například aby kanály uložené v hraně grafu opravdu touto hranou procházely).

Třída `Environment`

Ta uchovává data o celé simulaci. Ve vektoru typu `std::vector<Layer*>` jsou uloženy všechny vrstvy, v seznamu typu `std::list<NodeProperties*>` jsou uchovávány globální atributy uzlů (název a poloha).

Třída zajišťuje konzistenci datových struktur. Pokud je přidán vrchol, postará se, aby byl přidán i do všech již existujících vrstev. Pokud je přidána vrstva, vytvoří v ní všechny vrcholy a naváže ukazatele. Když je požadováno odebrání vrcholu, otestuje nejprve, zda přes vrchol neprochází kanál v nějaké vrstvě, a pak uzel odebere ze všech vrstev.

Třída se umí serializovat a deserializovat, čehož využívá třída `Facade` pro ukládání simulace do souboru a opětovné načítání.

3.5.2 Deskriptory

K identifikaci objektů v simulovaném prostředí se používá deskriptorů. Deskriptory nemají žádné atributy ani žádné metody, jsou to pouze hodnoty identifikující konkrétní objekt. Například pokud chceme odebrat uzel, je třeba mít jeho deskriptor `desc` a pak zavolat funkci `fac.RemoveNode(desc)` (`fac` je objekt typu `Facade`). Programátor používající modul `env` nepotřebuje znát vnitřní strukturu deskriptorů a ani by ho neměla zajímat.

Pro každý typ objektu máme jeden typ deskriptoru.

- `Facade::NodeDesc` – deskriptor uzlu
- `Facade::LinkDesc` – deskriptor spoje
- `Facade::LayerDesc` – deskriptor vrstvy
- `Facade::ChannelDesc` – deskriptor kanálu

Je třeba poznamenat, že spoj, kanál nebo atribut uzlu lokální ve vrstvě (viz 3.5.4) jednoznačně identifikuje dvojice deskriptor vrstvy, deskriptor spoje/kanálu/uzlu. To znamená, že například pro smazání kanálu je třeba znát nejen jeho deskriptor, ale také deskriptor vrstvy, ve které se kanál nachází.

Získávání deskriptorů

Pokud chceme získat například deskriptory všech uzlů, zavoláme

```
std::pair<Facade::NodeIter, Facade::NodeIter> ret;
ret = fac.Nodes();
```

funkce	typ iterátoru	návratová hodnota
`Nodes()`	`NodeIter`	Interval s deskriptory všech uzlů.
`Layers()`	`LayerIter`	Interval s deskriptory všech vrstev
`Links(...)`	`LinkIter`	Interval s deskriptory všech spojů ve vrstvě.
`Channels(...)`	`ChannelIter`	Interval s deskriptory všech kanálu ve vrstvě.
`ChannelNodes(...)`	`ChannelNodesIter`	Interval s posloupností vrcholů v kanálu.
`LinkChannels(...)`	`LinkChannelsIter`	Interval s deskriptory kanálů, které procházejí aktivním spojem.

Tabulka 3.1: Funkce pro získávání deskriptorů.

kde `fac` je objekt typu `Facade`. `Facade::NodeIter` je iterátor odkazující na hodnoty typu `Facade::NodeDesc`. Z iterátoru lze pouze číst (tj. `desc = *it` lze, ale `*it = desc` nelze) a iterátor lze posouvat dopředu i dozadu. Posouvání iterátoru mimo interval `<ret.first, ret.second)` vede k nedefinovanému chování.

Nejdůležitější funkce pro získávání deskriptorů jsou uvedeny v tabulce 3.1

Platnost deskriptorů

Deskriptory nemusí zůstávat (a opravdu nezůstávají) platné po celou dobu běhu programu. Je celkem samozřejmé, že deskriptor smazaného objektu by se neměl používat. Jestli však smazání nebo přidání objektu zneplatní i ostatní deskriptory, není vůbec jasné. V tabulce 3.2 je uvedeno, která operace zneplatní které deskriptory.

3.5.3 Manipulace s objekty

S objekty simulace se manipuluje pomocí členských funkcí třídy `Facade`. Všechny funkce jsou popsány ve zdrojových kódech, takže patřičné informace lze najít tam, nebo v dokumentaci vygenerované systémem Doxygen, která je uložena na CD.

Operace	Zneplatněné deskriptory
Přidání uzlu	žádné
Odebrání uzlu	všechny uzly
Přidání vrstvy	žádné
Odebrání vrstvy	všechny vrstvy
Přidání kanálu	žádné
Odebrání kanálu	odebraný kanál a deskriptory všech aktivních spojů, které přestaly kvůli odebrání kanálu existovat
Změna trasy kanálu	všechny spoje, které přestali kvůli změně trasy existovat
Undo/Redo	všechny
Load	všechny
Změna hodnoty atributu	žádné

Tabulka 3.2: Zneplatňované deskriptory při manipulaci s objekty.

3.5.4 Atributy

S každým objektem simulace jsou asociovány některé atributy (např. s uzlem poloha, název a vysílací výkon, s vrstvou nastavení algoritmů, ...). K atributům se přistupuje jednotně pomocí property map. Tento přístup má dvě výhody. Jednak umožňuje unifikovaný (z pohledu programátora) přístup k různým atributům a jednak jdou tyto property mapy použít v generických algoritmech knihovny *boost/graph* [3].

Property mapy

Zde je stručně popsáno co to vlastně property mapy jsou a jak se používají. Podrobnější informace lze najít v [2].

Property mapa (dále jen PM) je obecný koncept, který s klíči asociuje nějaké hodnoty. Property mapa může být libovolný typ, který splňuje jisté požadavky. Nejdůležitější z pohledu toho, kdo PM používá, jsou funkce pro přístup k hodnotám. Podle toho, co PM splňují, se rozdělují do čtyř druhů: *readable*, *writeable*, *read-write* a *lvalue*. Každá *lvalue* PM je zároveň *read-write* a každá *read-write* PM je zároveň *readable* i *writeable*.

V tabulce 3.3 jsou uvedeny funkce pro přístup k property mapám, jejich sémantika a druh, ve kterém je daná funkce vyžadována. V tabulce `Map` představuje typ property mapy, `map` je instance typu `Map`, `Key` je typ klíče, `k` je instance typu `Key`, `Val` je typ hodnot a `v` je instance typu `Val`.

Funkce	Popis	Druh
`v=get(map,k)`	Vrací hodnotu asociovanou s klíčem `k`.	*readable*
`put(map,k,v)`	Přiřadí ke klíči `k` hodnotu `v`.	*writeable*
`Val& v=map[k]`	`map[k]` vrací referenci na hodnotu asociovanou s klíčem `k`.	*lvalue*

Tabulka 3.3: Funkce pro přístup k property mapě.

Property mapy v programu

V PM atributů fungují deskriptory objektů jako klíče. Například pokud `posMap` je objekt typu PM pro polohy uzlů, `nDesc` je platný deskriptor uzlu a `newPos` jsou souřadnice, pak se změnění polohy uzlu provede jako `put( posMap, nDesc, newPos)`.

Složitější je situace u atributů kanálů nebo u lokálních (vzhledem k vrstvě) atributů uzlů (třeba uživatelem zadaný vysílací výkon). Zde k určení hodnoty atributu jeden deskriptor nestačí, tady už musí být dva (deskriptor vrstvy a deskriptor uzlu/kanálu).

Zde se nabízí dvě možná řešení. Buď používat jako klíč property mapy dvojici (`pair`) deskriptorů, nebo použít „dvouúrovňové“ PM (viz. níže). V programu byl zvolen druhý přístup, protože to umožňuje získat třeba property mapu s vysílacími výkony uzlů v jedné vrstvě (v té už jsou klíče pouze deskriptory uzlů). To způsobuje snadnější použití PM pro programátora (není nutné stále opakovat deskriptor vrstvy) a umožňuje to i přímé použití PM v boost knihovně.

V tomto odstavci je vysvětleno, co je to „dvouúrovňová“ PM. „Dvouúrovňová“ PM pro deskriptor vrstvy vrátí PM druhé úrovně, která už pro deskriptor kanálu/uzlu vrátí hodnotu atributu. Takže například pokud `powMap` je objekt typu PM s uživatelem zadanými vysílacími výkony, `lDesc` je platný deskriptor vrstvy a `nDesc` je platný deskriptor uzlu, pak uživatelem zadaný vysílací výkon by se získal jako

```
double power = get( get( powMap, lDesc), nDesc);
```

Obdobně změna by se provedla voláním

```
put( get(powMap, lDesc), nDesc, 10.0);
```

Na posledním příkladu je vidět jedna důležitá vlastnost PM. Zde se volá funkce `put` na mapu získanou z volání `get` (které vrací hodnotou). Mohlo by se tedy zdát, že po destrukci *dočasné* PM se výsledek akce zahodí. To se však nestane. Všechny PM atributů se chovají tak, že po zavolání copy konstruktoru sdílí nově vytvořená PM data se starou. Přiřazení se chová se stejnou sémantikou.

Zápis příkazů v předchozích dvou příkladech jde zkrátit na

```
double power = get( powMap, lDesc, nDesc);
```

respektive

```
put( powMap, lDesc, nDesc, 10.0);
```

Získání property mapy

Již bylo popsáno, co to property mapy atributů jsou a jak se používají, takže se konečně dostáváme k tomu, jak je získat.

K identifikaci typu PM používáme takzvané *tagy* (podobně jako v knihovně *boost/graph* [3]). Tag je typ, který neobsahuje žádné datové položky ani metody. Objekt typu tag se používá jako parametr funkce vracející PM tak, aby se zavolala správná přetížená funkce, jejíž návratová hodnota je PM požadovaného typu. Tag se používá také jako parametr šablony třídy `PropertyMap` (viz. níže).

Šablona třídy `PropertyMap` slouží ke získání typu požadované PM. Jako parametr šablony se použije tag. Ve vzniklé třídě jsou pak definice dvou typů:

- `PropertyMap<Tag>::type` - Typ read-write resp. lvalue PM.
- `PropertyMap<Tag>::const_type` - Typ readonly PM.

PM se získá voláním `get( tag, fac )`, kde `tag` je tag určující PM a `fac` je objekt typu `Facade`. Deklarace funkcí `get` vypadá takto:

```
template<class Tag>
PropertyMap<Tag>::type       get( Tag, Facade& fac );
template<class Tag>
PropertyMap<Tag>::const_type get( Tag, const Facade& fac );
```

Je vidět, že z konstantního objektu třídy `Facade` získáme pouze readonly PM.

Je důležité, aby životnost `fac` přesahovala životnost vrácené PM. Používání vrácené PM po zániku objektu `fac` vede k nedefinovanému chování.

Tagy a s nimi asociované PM jsou uvedeny v tabulce 3.4. V posledním sloupci je uvedeno, které deskriptory určují daný atribut (n–uzlu, l–vrstvy, ch–kanálu, i–spoje). Pokud jsou zde uvedeny dvě hodnoty, znamená to, že PM je dvouúrovňová.

Výčet tagů není úplný, jsou zde uvedeny pouze ty, které by mohl použít programátor implementující další algoritmy. Všechny tagy lze najít v souborech *envpropertymaps.h* a *layerpropertymaps.h*.

Pro přístup k hodnotě atributu není nutné nejdříve property mapu získat a pak až z ní číst nebo do ní zapisovat. Lze použít zkratky podobně jako u dvouúrovňových PM.

Tag	Asociovaný atribut	Typ hod.	Ident.
NodePropPositionTag	Poloha uzlu.	Vector2	n
NodePropIndexTag	Index uzlu.	int	n
LayerPropIndexTag	Index vrstvy.	int	l
LayerPropAlphaTag	Hodnota α (viz. 2.1).	double	l
LayerPropBackNoiseTag	Šum prostředí (viz. 2.1).	double	l
ChannelPropIndexTag	Index kanálu.	int	l, ch
LinkPropIndexTag	Index spoje.	int	l, i

Tabulka 3.4: Důležité tagy property map

```
v = get( tag, facade, lDesc, nDesc )
```

je ekvivalentní zápisu

```
v = get( get( tag, facade), lDesc, nDesc )
```

Názvy objektů

Názvy objektů slouží ke snadné identifikaci objektů uživatelem. Protože názvy musí být jednoznačné, není možné je měnit pomocí property map (při `put` není jak oznámit, že takový název je již použit). K tomu je třeba použít funkce

- `Facade::RenameNode()`
- `Facade::RenameChannel()`
- `Facade::RenameLayer()`

3.5.5 Serializace/deserializace

Aby šla simulace uložit, umí se třída `Environment` serializovat a deserializovat. K tomu je použita knihovna *boost/serialization* [4]. Pro uložení prostředí stačí zavolat funkci `Facade::Save()`, načtení se provede voláním `Facade::Load()`.

3.5.6 Undo/Redo

Při běhu programu dochází ke změnám v simulovaném prostředí (vytváří a ruší se objekty, upravují se atributy). Umožnit vrátit každou jednotlivou změnu je nešikovné a zbytečné. Proto je třeba explicitně označovat stavy, do kterých má jít simulace vrátit. K tomu slouží funkce `PushState()` a `SaveState()`.

`Facade::PushState()` vytváří nový stav. Funkce `Facade::SaveState()` přepíše předchozí uložený stav aktuálními hodnotami.

Voláním `Facade::Undo()` se vrátíme k předchozímu uloženému stavu a posunutí na další uložený stav se provede pomocí funkce `Facade::Redo()`.

Undo/Redo funkcionalita je implementována pomocí serializace. Při požadavku na vytvoření nového stavu se celá simulace uloží do paměti a při Undo/Redo se pak zase načte. To se může zdát jako pomalé, ale není, protože ukládání prostředí, nebo načítání se děje pouze na podnět uživatele.

3.6 Modul *ui*

Modul *ui* slouží k zobrazování simulované sítě, umožňuje uživateli manipulovat s objekty simulace, vybírat algoritmy, které se mají použít, a zobrazuje výsledky analýzy přenosu.

Tento modul byl vystaven na knihovně *Qt* (Qtoolkit). Volba Qt má dva důvody. První je, že má nejlepší dokumentaci mezi podobnými knihovnami pro Linux [5]. Druhým důvodem je přenositelnost. Qt je dostupné nejen pro Linux, případně jiné UNIX-like systémy, ale i pro Windows a MacOS. To by mělo umožnit snadný přenos programu na jinou platformu.

Dále popíšeme čtyři nejdůležitější třídy. V dalším textu je používáno anglické slovo *widget*. Jeho význam je stejný jako v dokumentaci ke Qt. Tímto termínem se myslí třída (nebo objekt třídy), která má jako předka (třeba nepřímého) třídu `QWidget` (viz. [5]). Protože neznám žádné ekvivalentní české slovo, budu používat anglický originál.

Widget `LayerView`

Je poděděn od třídy `QListView`. Jeho úkolem je zobrazovat hierarchii vrstev a kanálů a provádět manipulaci s nimi (vytváření, rušení, přejmenovávání).

Pomocí signálů informuje ostatní části programu o přidání nového objektu, o úpravě názvu objektu, nebo o změně výběru. Vybraný může být nejvýše jeden objekt.

Widget `EnvironmentView`

Je poděděn od třídy `QGLWidget`. Je to grafický widget který, zobrazuje simulované prostředí a některé statistiky.

Umožňuje uživateli pomocí myši upravovat polohu uzlů, přidávat nové uzly a rušit existující. Jeho pomocí lze vybírat uzly a aktivní spoje (v jeden okamžik může být vybrán nejvýše jeden objekt).

O přesunutí/vytvoření/smazání uzlu, nebo změně výběru objektu informuje ostatní části programu pomocí signálů.

Widget `PropEdtiStack`

Je poděděn od třídy `QWidgetStack` (je to zásobník s widgety, kdy viditelný je pouze ten na vrcholu zásobníku).

Umožňuje měnit parametry vybraných objektů a zobrazuje statistiky o nich. Pro každý typ objektu obsahuje zvláštní podřazený widget. Zobrazuje pak ten, který je nastaven pomocí funkcí `Set*()`. O jeho správné nastavení se stará widget `MainWindow`.

Podřazené widgety informují ostatní části programu o změně parametrů simulace, nebo o změnách takových, které nepotřebují přepočítání statistik (změna názvu uzlu).

Widget `MainWindow`

Tento widget tvoří hlavní okno programu. Jeho úkolem je řídit spolupráci tří výše popsaných widgetů.

Pokud některý widget změní parametr simulace, postará se o znovuurčení automaticky určovaných parametrů, přepočítání statistik a překreslení okna `EnvironmentView`. Při změně výběru v `LayerView` nebo `EnvironmentView` se postará o zviditelnění správného okna v `PropEditStack`.

Pokud dojde ke změně některého parametru simulace, zajistí tato třída přepočítání všech statistik, kterých se změna týká. To se provádí ve funkci `MainWindow::RecountStat()`. Odtud jsou volány algoritmy a pomocné funkce z modulu *stat*.

3.7 Modul *stat*

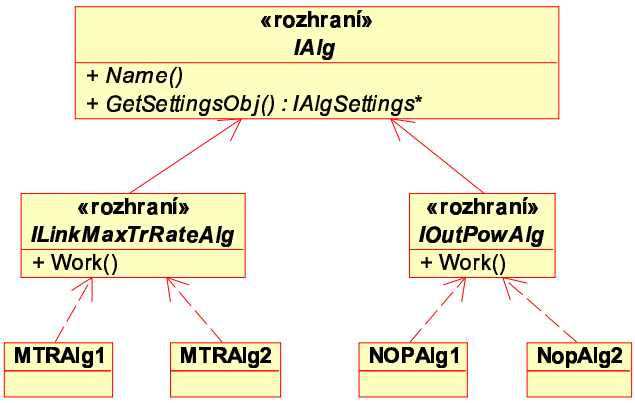
Tento modul zajišťuje automatické určování vysílacích výkonů a analýzu přenosů. Definuje rozhraní, které musí algoritmy splňovat, a jsou v něm implementovány dva algoritmy pro určování vysílacích výkonů uzlů a dva pro počítání maximálních přenosových výkonů spojů (viz. 2.2 a 2.3).

3.7.1 Rozhraní algoritmů

Program používá dva druhy algoritmů:

- algoritmy pro určení vysílacího výkonu uzlu (`IOutPowAlg`)
- algoritmy pro vypočítání maximální přenosové rychlosti spojem (`IMaxTrRateAlg`)

S každým druhem algoritmu je spojena třída (uvedena v závorce) definující rozhraní, které musí třída implementující algoritmus splňovat. Hierarchie tříd je zachycena v diagramu 3.1.



Obrázek 3.1: Diagram dědičnosti tříd

Protože se předpokládá, že v programu bude implementováno více algoritmů pro každý druh, musí být uživateli umožněno mezi nimi vybírat. K tomu je třeba, aby měly pro uživatele srozumitelné názvy.

Některé algoritmy mohou ke svému běhu potřebovat určitá nastavení. Tato nastavení se jednak musí ukládat a uživatel musí mít možnost je měnit. Algoritmy potřebující nastavení tedy musí definovat ještě třídu splňující rozhraní `IAlgSettings` (více viz. 3.7.2). Také musí existovat způsob, jak algoritmus sdělí, jestli nastavení potřebuje a pokud ano tak jaké.

Splnění požadavků v předchozích dvou odstavcích zajišťuje rozhraní `IAlg`. `Name()` vrací jméno algoritmu, které se bude zobrazovat v programu. `GetSettingsObj()` vrací ukazatel na nově vytvořený objekt typu nastavení daného algoritmu (potomek třídy `IAlgSettings`) (viz. 3.7.2). Tato funkce se volá, pokud je třeba vytvořit nový objekt s nastaveními pro daný algoritmus. Pokud vrací hodnotu `0`, pak to znamená, že tento algoritmus žádná nastavení nepotřebuje. Vlastnictví vráceného objektu přechází na toho, kdo funkci volal (tedy třída objekt nikdy neruší!).

Pro každý algoritmus je v programu vytvořen právě jeden objekt, který se používá k výpočtu parametrů všech vrstev. V každé vrstvě může být nastavení algoritmu jiné, takže je důležité, aby běh algoritmu závisel pouze na stavu simulovaného prostředí a na hodnotách v případném nastavení algoritmu. Tedy výsledek algoritmu nesmí záviset na vnitřním stavu objektu algoritmu!

Protože každý druh algoritmu počítá něco jiného, existují dvě různá rozhraní (`IOutPowAlg` a `ILinkMaxTrRateAlg`) poděděná od `IAlg`. V těch je

definována jediná funkce `Work()`. Ta je volána pokaždé, když je třeba vypočítat hodnotu nějakého parametru(ů) simulace. Obě funkce přebírají jako parametry ukazatel na nastavení (je nulový, pokud alg. nastavení nepotřebuje), referenci na `env::Facade` a deskriptor vrstvy, ve které algoritmus běží. Poslední dva parametry se předávají proto, že algoritmus může potřebovat některou informaci ze simulovaného prostředí (např. parametr α, nebo počet vrcholů, počet kanálů, ...).

3.7.2 Rozhraní nastavení

Všechny třídy nastavení algoritmů musí být poděděny od `IAlgSetting`. Hodnoty nastavení v nich uchovávané musí jít upravit uživatelem a musí jít uložit a opět načíst.

O první požadavek se stará funkce `Setup()`. Ta je volána kdykoli uživatel chce změnit nastavení. Tato funkce by měla zobrazit okno, kde uživatel může zadat potřebné hodnoty, a počkat až ho uživatel zavře. Pokud došlo ke změně některého parametru (a tedy je nutné přepočítat statistiky) vrací funkce `true`, jinak vrací `false`.

Možnost uložení a načtení zajišťuje členská šablonovaná funkce `serialize()`. Detaily jsou popsány v [4]. Příklad je uveden v 3.9.

3.8 Modul *ngl*

Tento modul obsahuje typy a funkce pro snadnější práci s OpenGL.

Detaily naleznete v Doxygen dokumentaci na CD nebo ve zdrojových kódech.

3.9 Přidávání nových algoritmů

Program byl navržen tak, aby do něj bylo možné snadno přidávat další algoritmy. Detaily o implementaci algoritmů najdete v 3.7.

K přidání nového algoritmu do programu je třeba vytvořit následující třídy:

- Třídu provádějící samotný výpočet daného parametru.
- Třídu uchovávající nastavení algoritmu.
- Třídu reprezentující dialog, pomocí kterého lze nastavovat parametry algoritmu.

První třída je vždy třeba. Druhou a třetí třídu není třeba vytvářet, pokud algoritmus nepotřebuje žádná nastavení. Popis rozhraní, které musí první dvě třídy splňovat, je v 3.7.

Přidání algoritmu bude demonstrováno na příkladu algoritmu nastavujícího vysílací výkony uzlů na uživatelem zadanou hodnotu.

Třída algoritmu

V této třídě ve funkci `Work()` probíhá samotný výpočet vysílacích výkonů.

```
/* soubor outpowalg_static.h */
namespace stat {
class OutPowAlg_Static : public IOutPowAlg
{
    public:
        OutPowAlg_Static() {}
        ~OutPowAlg_Static();
        virtual void Work( const IAlgSettings* settings,
                       const env::Facade& env,
                       env::Facade::LayerDesc layerDesc,
                       NodeOutPow::value_type nodeOutPow );
        virtual QString Name();
        virtual IAlgSettings* GetSettingsObj();

};
}

/* soubor outpowalg_static.cpp */
namespace stat {
OutPowAlg_Static::~OutPowAlg_Static() {}

void OutPowAlg_Static::Work( const IAlgSettings* settings,
                         const env::Facade& env,
                         env::Facade::LayerDesc,
                         NodeOutPow::value_type nodeOutPow )
{
    const OutPowAlgSettings_Static* set =
      dynamic_cast<const OutPowAlgSettings_Static*>(settings);
    double powVal = set->GetOutPow();
    env::Facade::NodeIter nSt, nEn;
    for ( tie( nSt, nEn) = env.Nodes(); nSt != nEn; nSt++ )
        put( nodeOutPow, *nSt, powVal );
```

```
}

QString OutPowAlg_Static::Name()
{   return QObject::tr( "Static", "NodeOutPowAlg_Static" ); }

IAlgSettings* OutPowAlg_Static::GetSettingsObj()
{   return new OutPowAlgSettings_Static(); }
}
```

Funkce `Name()` vrací řetězec identifikující algoritmus. `QObject::tr()` zajistí, že se řetězec zobrazí ve správném jazyce.

Funkce `GetSettingsObj()` vrací ukazatel na nově vytvořený objekt typu `OutPowAlgSetting_Static`.

Funkce `Work()` provádí samotný výpočet. Jediné co udělá je, že nastaví vysílací výkony všech uzlů na uživatelem zadanou konstantu. Funkce dostává v parametru `settings` ukazatel na objekt stejného typu, jaký vrací funkce `GetSettingsObj()`.

Aby bylo možné v programu nový algoritmus vybrat, je třeba do funkce `MainWindow::Init_()` přidat řádek

```
uiData_.outPowAlgSet().AddAlg( new stat::OutPowAlg_Static() );
```

Třída nastavení

```
/* soubor outpowalgsettings_static.h */
namespace stat {
class OutPowAlgSettings_Static : public IAlgSettings
{
    public:
        OutPowAlgSettings_Static( ) {}
        ~OutPowAlgSettings_Static();
        int Setup();
        double GetOutPow() const
        {   return outPow; }
    private:
        friend class boost::serialization::access;
        template<class Archive>
        void serialize(Archive & ar, const unsigned int)
        {
            namespace s = boost::serialization;
/* 1) */    ar & s::make_nvp( "base_class",
                    s::base_object<IAlgSettings>(*this) );
            ar & s::make_nvp( "outPow", outPow );
```

```
        }
    private:
        double outPow;
};
}
```

Šablonovaná funkce `serialize` *musí* mít tuto hlavičku, *musí* být privátní a *musí* obsahovat deklaraci spřátelené třídy `access`. Tělo funkce *musí* jako první serializovat předka, přesně jako v `1)`. Pak může následovat serializace dalších hodnot (více v [4]).

```
/* soubor outpowalgsettings_static.cpp */
namespace stat {
OutPowAlgSettings_Static::~OutPowAlgSettings_Static() { }

int OutPowAlgSettings_Static::Setup()
{
    ui::OutPowAlgSettingsDlg_Static
            dlg( outPow, 0, "outPowSettingsDlg" );
    int ret = dlg.exec();
    if ( ret == QDialog::Accepted ) {
        outPow = dlg.OutPow();
        return 1;
    } else
        return 0;
}
}
```

Funkce `Setup()` nejdříve vytvoří dialog, počká až ho uživatel zavře a pak vrátí 1 pokud došlo ke změně, nebo 0 pokud ke změně nedošlo.

Pro správnou funkci načítání a ukládání nastavení přes ukazatel na `IAlgSettings` je ještě nutno přidat do souboru *algorithmsettings.h* následující řádky:

```
#include "outpowalgsettings_static.h"
BOOST_CLASS_EXPORT( stat::OutPowAlgSettings_Static )
```

Jméno třídy na druhém řádku musí být včetně namespace!

Třída dialogu

Na třídě reprezentující dialog pro úpravy nastavení není nic k vysvětlování, takže její kód zde neuvádíme.

3.10 Přenos na jinou platformu

Všechny knihovny, které program používá, jsou k dispozici na všech nejhojněji používaných operačních systémech (Windows, Linux, Mac OS X). Knihovny jsou toho charakteru, že pro ně napsaný kód by měl jít přeložit na všech podporovaných platformách. V programu nejsou použity žádné nepřenositelné věci, takže by měl jít s minimálním úsilím přeložit i jinde než na Linuxu.

Program ukládá simulaci do xml souborů, které by měly být přenositelné i mezi různými platformami.

Kapitola 4

Uživatelská příručka

4.1 Spuštění programu

Po nainstalování programu se nachází spustitelný soubor (*mus*) a soubory s překladem do českého jazyka (*mus_cs.qm* a *mus-stat_cs.qm*) v adresáři `mus_package/mus/project/bin/`. Příklady s uloženou simulací jsou na CD v adresáři `doc/test/`.

Program se spouští příkazem `./mus`. Aby program používal češtinu, je třeba mít nastavené české prostředí; například provedením shellového příkazu

```
export LANG=cs_CZ
```

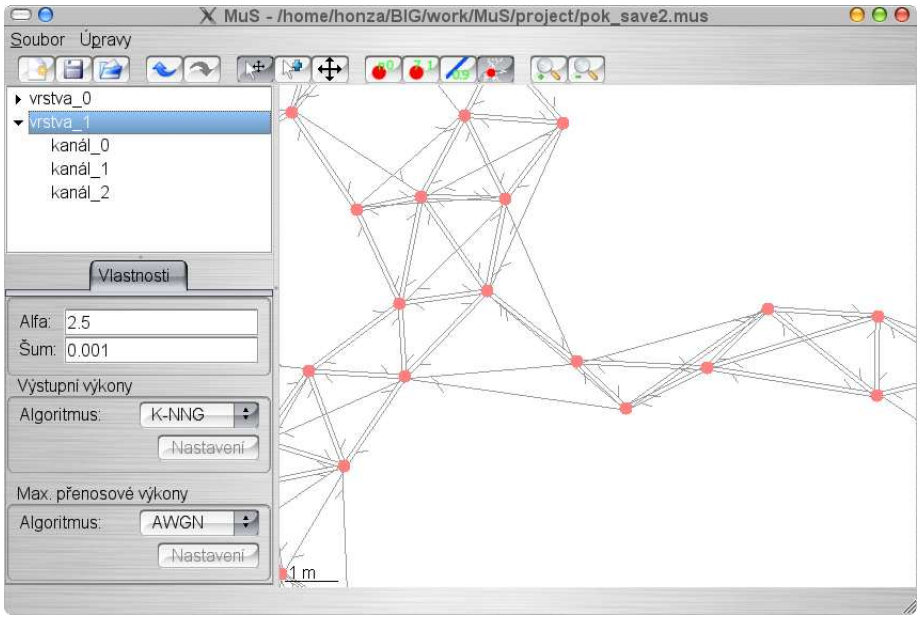
Pokud je aktuální jazyk různý od češtiny, bude program v angličtině. Ke správné funkci češtiny je třeba mít v adresáři programu dva soubory s překladem (více viz. 5).

4.2 Hlavní okno

Hlavní okno je rozděleno do tří částí: grafické okno, seznam vrstev a kanálů a okno s vlastnostmi a statistikami vybraného objektu.

4.3 Grafické okno

Grafické okno se nachází vpravo. Zobrazuje prostředí, umožňuje přidávat, odebírat, posouvat uzly a vybírat uzly a spoje. (viz. 4.8)



Obrázek 4.1: Hlavní okno programu.

4.4 Seznam vrstev a kanálů

Okno vlevo nahoře je strom s názvy vrstev a kanálů. Vrstvy jsou položky v hloubce 1, kanály představují položky v hloubce 2.

Po stisknutí pravého tlačítka mimo položky se zobrazí nabídka, skrze kterou jde přidat novou vrstvu.

Stisknutím pravého tlačítka nad vrstvou se zobrazí nabídka, pomocí které lze přidat kanál do vrstvy, odstranit vrstvu, nebo zkopírovat vrstvu.

Při stisknutí pravého tlačítka nad kanálem se zobrazí nabídka, pomocí které lze kanál smazat.

Pokud v okně vyberete položku a kliknete na ní levým tlačítkem, budete moci upravit její název. Ten musí být jedinečný mezi objekty stejného typu.

4.5 Vlastnosti a statistiky

Okno vlevo dole zobrazuje informace o právě vybraném objektu (viz. 4.6). Pokud je vybrán spoj, je v okně pouze záložka *Statistiky*; Pokud je vybrána vrstva nebo uzel, je tam pouze záložka *Vlastnosti*). Pokud je vybrán kanál, jsou zobrazeny obě záložky.

V záložce *Statistiky* se zobrazují vypočtené statistiky dostupné pro daný objekt.

V záložce *Vlastnosti* se upravují vlastnosti vybraného objektu (název,

poloha, trasa kanálu, ...).

Vlastnosti vrstvy

U vrstvy se nastavuje algoritmus pro určování vysílacích výkonů uzlů a algoritmus, který z odstupu signálu od šumu určí maximální přenosový výkon spoje. U obou algoritmů je ještě tlačítko *Nastavení*. Pokud není zašedlé, tak kliknutím na něj zobrazíte okno s nastaveními algoritmu. Pokud je zašedlé, znamená to, že se u algoritmu nedá nic nastavovat.

Zde jdou ještě nastavit hodnota α (viz. 2.1) a šum prostředí (viz. 2.2).

Vlastnosti uzlu

Zde lze upravit název, polohu a vysílací výkon uzlu. Vysílací výkon je možné nastavit buď ručně, nebo lze přepnout na určování vysílacího výkonu automaticky.

Vlastnosti kanálu

U kanálu je možné nastavit název, barvu a trasu kanálu. Trasa kanálu se určuje buď automaticky (pak lze vybrat pouze první a poslední uzel), nebo ručně (potom se nastavují i mezilehlé uzly).

Pokud ve vlastnostech kanálu vyberete ruční určování trasy kanálu, zobrazí se v dolní části záložky *Vlastnosti* tlačítko *Upravit trasu* a seznam s posloupností vrcholů, přes které probíhá komunikace.

Pokud chcete upravit kanál, stiskněte tlačítko *Upravit trasu*. To přepne program do režimu upravování kanálu.

Přidání uzlu do kanálu se provede tak, že vyberete (v seznamu, nebo v grafickém okně) uzel, za který se má nový uzel přidat, a pak v grafickém okně vyberete ten nový uzel. Pokud chcete přidat uzel na začátek kanálu, pak nejprve zrušte výběr uzlů (např. kliknutím do grafického okna mimo uzly) a potom vyberte uzel, který chcete přidat.

Odstranit uzel z kanálu lze buď stisknutím pravého tlačítka nad názvem uzlu v seznamu a vybráním *odstranit z kanálu*, nebo vybráním uzlu v seznamu a stisknutím klávesy `Delete`.

Pokud je vybráno automatické určování trasy, zobrazí se v dolní části dvě tlačítka pro nastavení zdrojového a cílového uzlu kanálu. Vybrání uzlu se provede tak, že kliknete na odpovídající tlačítko a v grafickém okně vyberete novy uzel.

4.6 Vybírání objektů

V jednu chvíli může být vybrán nejvýše jeden objekt (vrstva, kanál, uzel, spoj). K výběru objektů slouží *grafické okno* a *seznam vrstev a kanálů*. Kanály a vrstvy se vybírají v seznamu vrstev a kanálů. Uzly a spoje se vybírají v grafickém okně. Informace o právě vybraném objektu se zobrazují v okně vlevo dole.

4.7 Nabídka

Soubor

Umožňuje ukládání a načítání simulace, vytvoření nové simulace a ukončení programu.

Úpravy

Zpět

Vrátí naposledy provedenou změnu.

Znovu

Znovu provede naposledy vrácenou změnu.

Hromadné přidání vrcholů

Umožňuje přidat náhodně rozmístěné vrcholy do prostředí. Při vybrání této položky se otevře nové okno, ve kterém uživatel zadá *počet* vrcholů a *prefix* názvu.

Každý nově přidaný vrchol bude mít název `<prefix><číslo>`, kde `číslo` je nejmenší nezáporné číslo takové, že uzel se stejným názvem ještě neexistuje. Poloha vrcholu bude někde ve viditelné části prostředí.

4.8 Nástrojová lišta

Tlačítka v nástrojové liště umožňují rychlejší práci s programem. Tlačítka jsou rozdělena do pěti skupin.

Soubor

První skupina tlačítek jsou zkratky pro některé položky z nabídky *Soubor* (viz. 4.7).

Úpravy

Druhá skupina tlačítek jsou zkratky pro *zpět* a *znovu* (viz. 4.7).

Režim myši

Třetí skupina tlačítek umožňuje vybrat režim myši pro *grafické okno*.

První tlačítko přepne do režimu vybírání. Vybrání objektu se provede stisknutím levého tlačítka nad objektem. Výběr se zruší stisknutím tlačítka mimo objekty. Stisknutí pravého tlačítka nad uzlem zobrazí nabídku, pomocí které lze uzel odebrat. V tomto režimu lze také posouvat uzly. Stačí najet kurzorem nad uzel, stisknout levé tlačítko, přetáhnout uzel na novou pozici a tlačítko uvolnit.

Druhé tlačítko přepne do režimu přidávání vrcholů. Pokud je tlačítko aktivováno a uživatel stiskne levé tlačítko nad grafickým oknem, přidá se nový uzel na místo pod kurzorem. Název uzlu se nastaví na `n_<číslo>`, kde `číslo` je nejmenší nezáporné číslo takové, že uzel se stejným názvem ještě neexistuje.

Třetí tlačítko přepne do režimu pohybu prostředím. Stisknutím levého tlačítka se prostředí „uchopí“ a pohybem myši se posunuje. Po uvolnění tlačítka se prostředí „upustí“ a k žádnému dalšímu pohybu prostředí již nedochází.

Ve všech režimech lze pomocí prostředního tlačítka posouvat prostředím a kolečkem myši oddalovat a přibližovat prostředí.

Zobrazení

Čtvrtá skupina tlačítek nastavuje, co se má v grafickém okně zobrazovat.

První tlačítko zapíná zobrazování názvů uzlů, druhé zobrazování vysílacích výkonů uzlů a třetí zapíná zobrazování maximálních přenosových výkonů aktivních spojů. Posledním tlačítkem se vybírá, zda se mají zobrazovat aktivní spoje a kanály nebo všechny dostupné spoje (Tj. mezi uzly se vykreslí orientovaná hrana, pokud mohou uzly daným směrem komunikovat (viz. 2.1)).

Přiblížení

V poslední skupině jsou dvě tlačítka na přibližování a oddalování prostředí.

Kapitola 5

Instalace

5.1 Příprava

Na přiloženém CD se nachází adresář `mus_package`. V tom jsou obsaženy zdrojové kódy tohoto programu, zdrojové kódy knihoven Qt a boost a skripty pro snadnější instalaci. Následující návod předpokládá, že jde do adresáře zapisovat, tedy jako první zkopírujte adresář `mus_package/` z CD na oddíl pevného disku, kde máte právo zapisovat.

5.2 Kompilátor

Ke kompilaci je potřeba kompilátor gcc verze 3.4.* nebo 3.3.*.

Při vývoji byla použita verze 3.4.5. Program se podařilo úspěšně přeložit i s verzí 3.3.6. Měl by tedy jít přeložit s gcc verze 3.4.* a 3.3.*, z nichž alespoň některá je na většině systémů dostupná. Jiné kompilátory nebyly testovány. gcc lze stáhnout na adrese *http://gcc.gnu.org/*.

5.3 Knihovny

OpenGL a Glut

Program používá knihovny OpenGL a Glut, které jsou standardní součástí běžných systémů s grafickým prostředím.

boost

Další knihovna potřebná ke kompilaci je knihovna (nebo spíše sada knihoven) *boost*. Pokud máte v systému nainstalovanou tuto knihovnu verze 1.33.1 nebo vyšší, můžete přejít na další sekci.

Při vývoji programu byla použita verze 1.33.1. S nižší verzí se program nepřeloží! Použití vyšší verze než 1.33.1 nebylo testováno (ale mělo by být možné).

Pokud tato knihovna v systému není nainstalována, tak ji buď nainstalujte způsobem obvyklým v dané distribuci (gentoo: `emerge boost`), nebo ji postavte pouze lokálně. To se provede tak, že se přepnete do adresáře `mus_package` a zavoláte skripty

```
./scripts/build_boost.sh src/boost_1_33_1.tar.bz2
source ./scripts/env_boost.sh
```

To vytvoří adresář `build/boost/` a v něm potřebné hlavičkové soubory a knihovny a nastaví prostředí tak, aby je kompilátor našel. V tomto případě se postaví pouze ty knihovny, které program potřebuje.

Qt

Poslední knihovna kterou program používá je knihovna *Qt* verze 3. Pokud ji v systému nemáte, nainstalujte ji způsobem obvyklým na dané distribuci, nebo ze zdrojových kódů (jsou přiloženy na CD). U knihovny je podrobný návod instalace, takže nemá smysl ho zde opisovat.

Může se ještě stát, že v systému je více verzí knihovny Qt. V tom případě je třeba zajistit, aby při kompilaci byla použita ta správná. Příkazem

```
qmake -v
```

zjistíte verzi Qt, kterou používáte. Pokud je verze různá od 3, nastavte proměnnou prostředí `PATH` tak, aby cesta k verzi 3 předcházela verzím ostatním. Pokud jsou spustitelné soubory Qt verze 3 v adresáři `/usr/qt/3/bin`, provedete to takto:

```
PATH=/usr/qt/3/bin:$PATH
```

Jako další je třeba se ujistit, že jsou správně nastaveny proměnné prostředí `QMAKESPEC` a `QTDIR`. Aby `qmake` používal překladač gcc proveďte:

```
export QMAKESPEC=linux-g++
```

a proměnnou `QTDIR` nastavte na adresář s knihovnou Qt například takto:

```
export QTDIR=/usr/qt/3/
```

5.4 Kompilace

Nyní je již všechno připraveno ke kompilaci.

Předpokládejme, že aktuální adresář je `mus_package`. Kompilaci provede následující posloupnost příkazů:

```
tar -xvjf src/mus.tar.bz2
cd mus/project
qmake mus.pro -o Makefile
make
make -f Makefile_trans
```

Poslední příkaz vytvoří soubory s překladem programu do češtiny a uloží je ke spustitelnému souboru. Zkompilovaný program (včetně souborů s překladem) naleznete v `mus_package/mus/project/bin`.

5.5 Možné problémy

Konflikt verzí knihovny `libstdc++.so`

Je potřeba, aby program, knihovna *Qt* i knihovna *boost* byly zkompilovány pomocí stejné verze gcc, jinak hrozí, že dojde ke konfliktu sdílených knihoven `libstdc++.so`. Kompilátor při linkování vypíše sice jen warning, ale může to způsobovat pád programu.

Chybějící hlavičkové soubory Qt

Pokud kompilace skončí s chybovou hláškou, že se nepodařilo najít některý hlavičkový soubor Qt (např. *qpopupmenu.h*), je to pravděpodobně způsobeno tím, že kompilátor používá jinou verzi Qt než 3.*. Ujistěte se, že používáte správnou verzi Qt (jak je to uvedeno v 5.3), a proveďte instalaci znovu. Pozor, zavolat znova pouze `make` nestačí! Některé soubory `Makefile` jsou již vytvořeny a v nich se používá špatná verze Qt. Nejjistější je smazat celý adresář `mus_package/mus/` a provést znovu postup uvedený v 5.4.

Kapitola 6

Závěr

Zadáním práce bylo vytvořit systém pro simulaci směrování v bezdrátových sítích, který umožňuje určovat parametry přenosů automaticky. Hlavní problém, který bylo třeba vyřešit, byl jak provádět automatické určování kanálů a vysílacích výkonů.

První nápad bylo nejdříve zavolat algoritmus, který určí trasy kanálů, a z nich určovat vysílací výkony. Takový přístup ale neodpovídá realitě. Aby se určila trasa kanálu ve skutečné síti, je třeba aby spolu uzly již komunikovaly. Z tohoto přístupu také vyplývalo, že vysílací výkon uzlů závisí na požadavcích na komunikaci, což určitě není běžné chování skutečných sítí.

Skutečné sítě se chovají přesně obráceně. Tam se nejdříve nastaví vysílací výkony uzlů, každý uzel zjistí s kým může přímo komunikovat a pak již komunikace probíhá pouze po těchto dostupných spojích. Takový přístup byl nakonec zvolen i v programu.

6.1 Shrnutí dosažených výsledků

Byl vytvořen uživatelsky přívětivý a snadno použitelný systém pro testování směrování v bezdrátových sítích. Systém umožňuje ručně zadat všechny relevantní parametry, nebo provádí jejich automatické určování způsobem, který je blízký fungování skutečných bezdrátových sítí. Výstup systému jsou maximální přenosové rychlosti spojů a kanálů.

Systém je navržen tak, že umožňuje snadné přidávání dalších algoritmů. Na ukázku byly implementovány dva algoritmy pro určování vysílacích výkonů a dva algoritmy pro počítání maximálních přenosových rychlostí. Předpokládá se, že do systému budou přidávány podle potřeby další algoritmy.

Obrovský přínos měla práce pro mě jako pro řešitele. Prohloubil jsem si znalosti o programování v C++ (hlavně v oblasti šablon), naučil jsem se používat knihovnu Qt a objevil jsem řadu nových možných použití knihovny

boost. Práce mě naučila soustavné a trpělivé práci. Získal jsem také řadu nových zkušeností a poznatků o programování uživatelského rozhraní.

6.2 Nápady na vylepšení

Hlavní náplní této práce bylo vytvořit „infrastrukturu“ na testování směrování. To, aby dosažené výsledky simulace přesně odpovídaly realitě, nebylo prvořadým úkolem. Další práce by tedy měla směřovat tímto směrem.

Určování maximálních přenosových výkonů probíhá nyní podle velmi zjednodušeného modelu sítě a na základě jednoduchých výpočtů. Další práce by tedy měla směřovat k zpřesnění modelu sítě tak, aby se více přiblížil realitě. Místo provádění statistických výpočtů by bylo přesnější simulovat skutečný provoz sítě (např. jako v [1]).

Literatura

[1] Kučera Luděk, Kučera Štěpán: *Wireless Communication in Random Geometric Topologies*, Praha, Kyoto, 2006

[2] Boost.org: *Property map library documentation*, http://www.boost.org/libs/property_map/property_map.html

[3] Boost.org: *Graph library documentation*, http://www.boost.org/libs/graph/doc/table_of_contents.html

[4] Boost.org: *Serialization library documentation*, http://www.boost.org/libs/serialization/doc/index.html

[5] Trolltech: *Qt assistant*, verze 3.3.4