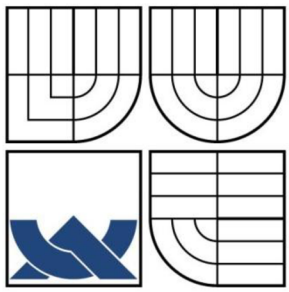
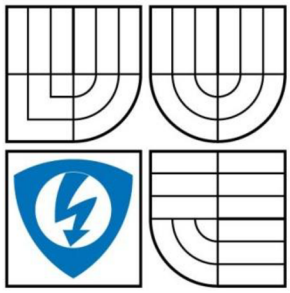


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKACNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

# SMĚROVACÍ PROTOKOL OLSR PRO MANET SÍTĚ V SIMULAČNÍM PROSTŘEDÍ OPNET MODELER

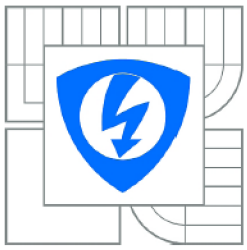
OLSR ROUTING PROTOCOL FOR MANET NETWORKS IN OPNET MODELER SIMULATION ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. PETR HOŠEK
AUTHOR

VEDOUCÍ PRÁCE — ING. LUKÁŠ RŮČKA
SUPERVISOR

BRNO 2011



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Diplomová práce

magisterský navazující studijní obor
**Telekomunikační a informační technika**

***Student:*** Bc. Petr Hošek ***ID:*** 73027
***Ročník:*** 2 ***Akademický rok:*** 2010/2011

**NÁZEV TÉMATU:**

## Směrovací protokol OLSR pro MANET sítě v simulačním prostředí OPNET Modeler

**POKYNY PRO VYPRACOVÁNÍ:**

Seznamte se s problematikou MANET sítí. Zejména se zaměřte na procesy směrování v těchto sítích. Proveďte podrobný teoretický rozbor směrovacího protokolu OLSR. Seznamte se se simulačním prostředím OPNET Modeler a jeho možnostmi konfigurace a simulace modelu MANET sítě. Prostudujte a zdokumentujete procesní model směrovacího protokolu OLSR v prostředí OPNET Modeler. Identifikujte místa a možnosti úpravy obsahu směrovacích zpráv tohoto protokolu. Rozšiřte původní zprávy směrovacího protokolu o další pole, pomocí kterého může bezdrátová stanice informovat své okolí o míře využití dostupné přenosové kapacity. Upravte původní mechanismy zpracování zpráv směrovacího protokolu OLSR o vyhodnocení a ukládání zatížení sousedních uzlů, pravidelnou aktualizaci těchto údajů a aktualizaci údajů o stanicích, které jsou v dosahu. Všechna svá zjištění a provedené úpravy podrobně zdokumentujte a přehledně prezentujte.

**DOPORUČENÁ LITERATURA:**

[1] Doyle, J., Carroll, J.: Routing TCP/IP. Indianopolis: Cisco Press, 2005. 936 s. ISBN: 1-58705-202-4.
[2] Zhang, Yan; Luo, Jijun; Hu, Honglin. Wireless Mesh Networking: Architectures, Protocols and Standards. New York: Auerbach Publications, 1. vydání, 2007. 592 s. ISBN 0-8493-7399-9.
[3] OPNET Technologies, OPNET Modeler Documentation Rel. 16.0, OPNET Technologies Inc., 2010.

***Termín zadání:*** 7.2.2011 ***Termín odevzdání:*** 26.5.2011

***Vedoucí práce:*** Ing. Lukáš Růčka

**prof. Ing. Kamil Vrba, CSc.**
*Předseda oborové rady*

# ANOTACE

První část práce se zaměřuje obecně na problematiku směrování v počítačových sítích. Popisuje význam směrování, základní prvky, algoritmy a protokoly. Dále popisuje tzv. MANET (Mobile Ad-hoc Network) sítě, kde jsou rozebrány směrovací protokoly v těchto sítích. Další kapitola se věnuje rozboru směrovacích protokolů OSPF (Open Shortest Path First ) verze 3 a protokolu OLSR (Optimized Link State Routing). U obou protokolů jsou popsány základní algoritmy, použití a jejich struktura. U protokolu OSPF je mimo jiné popsán i historický vývoj a rozdíly mezi verzí 2 a 3. U protokolu OLSR je popsán princip tzv. MPR (Multi Protocol Routing) uzlu, které tvoří jednu z nejdůležitějších součástí tohoto protokolu a kterými se odlišují od ostatních. Dále je krátce představen výkonný simulační nástroj OPNET modeler, pomocí kterého je možné simulovat provoz a chování téměř libovolné počítačové sítě s velmi rozsáhlými možnostmi nastavení parametrů jak pro fungování celé sítě, tak uzlů samotných. Praktická část práce je rozdělena do pěti částí. V první části se popisuje struktura procesních modelů, základních prvků, proměnných a bloků editorů směrovacího protokolu OSLR v programu OPNET modeler. Druhá část se věnuje datové struktuře ICI (Interface Control Information), která slouží pro mezi-procesovou komunikaci a ověřuje vytváření a příjem zpráv pomocí této funkce. Třetí část se věnuje procesnímu modelu směrovacího protokolu OLSR. Jsou zde popsány jednotlivé bloky a funkce, které se podílejí na funkci protokolu, a jejich význam. Ve čtvrté kapitole je popsáno rozšíření datové jednotky HELLO zprávy protokolu OLSR o další pole, které je schopné přenášet číselnou hodnotu mezi okolními stanicemi a její výpis do konzole. Poslední blok praktické části se věnuje vytvoření vlastní zprávy, která je odesílána mezi stanicemi jako součást OLSR. Tato zpráva obsahuje parametr aktuální přenosové rychlosti vzájemně komunikujících stanic. Tyto hodnoty jsou pro každou stanici zvlášť ukládány do externího souboru pro pozdější zpracování.

**KLÍČOVÁ SLOVA**: MANET, OLSR, OSPFv3, OPNET Modeler, QoS, směrovací protokol

## ABSTRACT

The first part focuses on general routing of computer networks. It describes the importance of routing, basic elements, algorithms and protocols. It also describes the so-called MANET networks, where there are discussed routing protocols in these networks. The next chapter deals with the analysis of routing protocols OSPF version 3 and OLSR protocol. For both protocols basic algorithms, their use and structure are described. In OSPF protocol there is also described historical development and differences between versions 2 and 3. The OLSR protocol properly describes the principle of MPR nodes, which is one of the most important parts of the protocol which differs it from the others. Next there is a brief introduction of a powerful simulation tool OPNET Modeler which allows simulating the operation and behavior of almost any computer network with a very extensive possibility of options for the functioning of the entire network, as well as nodes it selves. The practical part is divided into five parts. The first section describes the structure of process models, the basic elements, variables, and block editors OSLR routing protocol in OPNET Modeler program. The second part is devoted to the ICI data structure, which is used for inter-process communication and verify the creation and reception of messages using this function. The third part deals with the process model of OLSR routing model. There is a description of various blocks and functions involved in its function of protocol and significance. The fourth chapter shows an extension of data unit protocol OLSR HELLO messages to other fields, which is able to transmit a numeric value between neighboring stations, and these values print it in the console. The last block of the practical part is dedicated to creating its own message which is broadcast between stations like part of the OLSR packet messages. This message includes the parameter of actual data traffic rate of each communicating stations. These values are for each station exported to an external file for later processing.



**KEYWORDS**: MANET, OLSR, OSPFv3, OPNET Modeler, QoS, routing protocol

HOŠEK, P. *Směrovací protokol OLSR pro MANET sítě v simulačním prostředí OPNET Modeler*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 64 s. Vedoucí diplomové práce Ing. Lukáš Růčka.

## PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma "Směrovací protokol OLSR pro MANET sítě v simulačním prostředí OPNET Modeler" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ……..  ……..……..

(podpis autora)

**PODĚKOVÁNÍ**

V prvé řadě bych chtěl poděkovat vedoucímu diplomové práce Ing. Lukáši Růčkovi za odborné vedení, cenné rady a připomínky ke zpracování práce. Dále bych chtěl poděkovat celému týmu MANET, pod vedením doc. Ing. Karola Molnára, Ph.D., za příkladný přístup ke studentům a za vytvoření výborného kolektivu, který mi díky možnosti spolupráce se studenty zpracovávajících podobně zaměřenou problematiku byl nápomocen při řešení daného problému.

V Brně dne …….. ……..……..

Bc. Petr Hošek

# OBSAH

# ÚVOD

V dnešní době neustále se rozvíjejících technologií mohou být už i bezdrátové sítě natolik kvalitní, že se jejich využívání stává postupně více a více oblíbené před sítěmi pevnými. Hlavním důvodem stále rostoucí popularity je možnost mobility účastníka nebo zařízení, jež v bezdrátové síti funguje. Je třeba si ale uvědomit, že se svobodou pohybu to není tak zcela přesné, neboť uzel je sice mobilní a stále připojený v síti, ale může se pohybovat pouze ve vymezené oblasti, která je dána rozsahem vysílaného signálu přístupového bodu. Právě proto vznikly po několikaletém výzkumu takzvané MANET (Mobile Ad-hoc Network) sítě, které nepotřebují pro vzájemnou komunikaci žádný centrální prvek, a samotné směrování je řešeno přímo v mobilních uzlech. Zpočátku byla tato technologie vyvinuta pro vojenské účely a později začala pronikat i do komerční sféry [7]. Takže cestu měla tato technologie stejnou jako velké množství ostatních Hi-Tech technologií, které se nejprve vyvinou pro armádní potřeby a později, pokud je poptávka, se dostanou i do komerční sféry.

Jelikož směrování v sítích je poměrně složitý a rozsáhlý proces, bylo nutné vytvořit více druhů tzv. směrovacích protokolů neboli souhrn pravidel a postupů, dle kterých se směrování v sítích řídí. Mezi takové patří i protokoly OSPF (Open Shortest Path First ) a OLSR (Optimized Link State Routing), které jsou v teoretické části této práce podrobně rozebrány a popsány.

Kromě problematiky složitosti směrovacích protokolů, kde je kladen důraz na rychlost hledání a najití té správné cesty k cíli, se v posledních letech stále více řeší problematika zajištění požadované úrovně kvality přenosu pro dané spojení, tzv. QoS (Quality of Service). V sítích MANET, kde jsou uzly mobilní, je zjištění těchto služeb mnohem náročnější proces, než v sítích s pevnou infrastrukturou. Praktická část práce se kromě popisu protokolu OLSR (protokol byl zvolen po dohodě s vedoucím práce), jeho strukturou, vytvářením a funkcí zabývá možností rozšíření OLSR zprávy o pole, které bude údaj QoS přenášet mezi vzájemně komunikujícími uzly.

# 1. SMĚROVÁNÍ

Směrování je tou nejkomplikovanější a zároveň nejdůležitější funkcí všech počítačových sítí a první náznaky se začaly objevovat už v 50. letech minulého století. Byly to ale spíše předpovědi do budoucna, jelikož v té době jen málokterá organizace měla vůbec jeden jediný počítač.

Největším úskalím spojeným s vytvářením a používáním globální internetové sítě je vývoj vhodných prostředků pro vyhledávání vzdálených hostitelů, přístup a komunikaci mezi nimi. Je zřejmé, že jedna globální internetová síť musí poskytovat redundanci, tzn., že mezi libovolnou dvojicí hostitelských systémů musí být několik různých fyzických cest v případě, že by u jedné z cest došlo k výpadku.

A nakonec je potřeba do celého mechanismu zapojit určitá logická nebo matematická pravidla. Pokud do určitého cíle vede několik různých cest, je logické, že si nemohou být úplně rovny. Některé z nich budou třeba kratší nebo rychlejší než jiné. Bylo by tedy logické, všechny možné cesty vzájemně porovnat a vybrat z nich tu nejlepší nebo ty nejlepší. Postupem času se z těchto mechanismů vyvinuly takzvané směrovače a proces rozpoznávání, výpočtů a porovnávání cest do vzdálených sítí a hostitelských systémů pak nazýváme směrováním [14].

## 1.1 Význam směrování

Princip logické přilehlosti systémů funguje dobře nejen mezi dvěma počítači zapojenými ke stejné síti LAN (Local Area Network), ale i mezi počítači vzájemně propojenými přes internetovou síť a vzdálenými třeba tisíce kilometrů. Je přitom zřejmé, že mezi těmito dvěma extrémy musí být významné rozdíly. Tím největším je zde navazování spojení mezi zdrojovým a cílovým počítačem. V síti LAN tak oběma počítačům stačí při vzájemné komunikaci pouze odvysílat potřebné datové rámce do přenosového média. V internetové síti jsou oproti tomu oba systémy odděleny blíže nezjištěným počtem síťových hardwarových zařízení a přenosových prostředků. A zaplavit všechny tyto přenosové prostředky s tím, že snad pakety nakonec do cíle nějak dorazí, samozřejmě není řešení.

Jediným logickým řešením je identifikovat cestu v internetové síti do cíle. Nalezení cesty, té nejlepší cesty, je tedy náročný úkol.

## 1.2 Základní směrovací prvky

Mezi nejzákladnější směrovací prvky patří směrovače. Tato zařízení pracují na třetí vrstvě referenčního modelu OSI (Open Systems Interconnection) a musí mít dvě nebo více fyzických rozhraní, která jsou připojena k sítím LAN anebo k přenosovým zařízením sítě WAN (Wide Area Network). Během své činnosti zjišťují adresy počítačů a sítí, připojených k jednotlivým rozhraním, a jejich seznam ukládají do tabulky, která definuje vztah mezi adresami vrstvy tři a čísla portů, k nimž jsou příslušné systémy přímo nebo nepřímo připojeny.

Směrovač pracuje se dvěma typy síťových protokolů, z nichž oba působí ve vrstvě tři. Jsou to směrovatelné a směrovací protokoly. Směrovatelné protokoly, označované také jako směrované protokoly, zapouzdřují uživatelské informace a data do podoby paketů. Příkladem směrovaného protokolu je IP (Internet Protocol), jehož úkolem je zapouzdření aplikačních dat pro síťový přenos do příslušného cíle. Směrovací protokoly běží oproti tomu jen mezi směrovači, které podle nich stanovují dostupné cesty, vyměňují si o nich informace a po těchto cestách pak přeposílají pakety směrovaného protokolu. Úkolem směrovacího protokolu je pak poskytnout směrovači o síti veškeré informace, které potřebuje ke stanovení trasy, neboli cesty datagramů (paketů) [6].

**Výpočty cest**

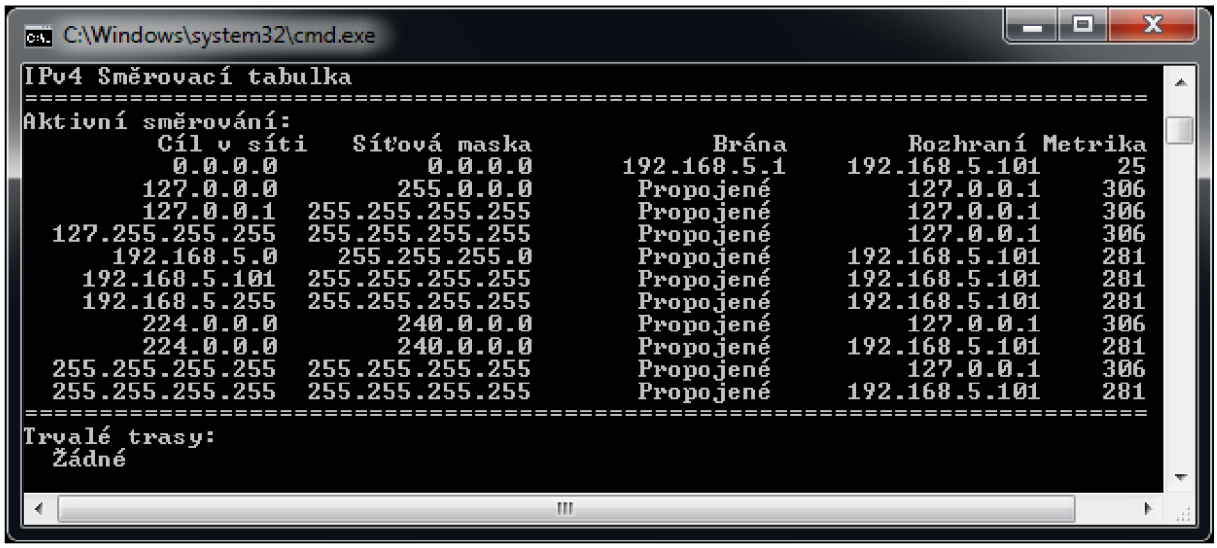
Úkol směrovače je najít k cíli vždy tu nejoptimálnější cestu z několika možných. Směrovač musí rozlišit různé cesty do stejného cíle a vybrat z nich tu nejlepší. Z této jednoduché charakteristiky vyplývá, že při stanovování cest do vzdálených cílů se uplatňuje určitá matematická logika. Schopnost aplikovat tuto logiku a provádět zmíněné matematické výpočty je tou nejdůležitější vlastnosti každého směrovacího prvku.

Jak už bylo zmíněno v předešlém odstavci, souhrnu pravidel, která umožňují směrovačům potřebné výpočty cest, říkáme směrovací protokol. Ve skutečnosti existuje směrovacích protokolů celá řada, z nichž většina je hojně podporována, takže můžeme internetové sítě sestavovat i ze směrovačů od různých výrobců.

Prostřednictvím směrovacích protokolů si směrovače zapojené do sítě mohou vzájemně vyměňovat informace o potenciálních cestách do konkrétních hostitelských systémů v této síti [14].

## 1.3 Směrovací tabulky

Směrovací tabulka je základní datová struktura při procesu směrování. Je to určitý datový soubor uložený v paměti RAM (Random-Access Memory) a uchovává v sobě informace o přímo a vzdáleně připojených sítích. Její obsah napovídá směrovacímu prvku (směrovači), přes které rozhraní je možno nejoptimálněji dosáhnout cílové sítě. Příklad směrovací tabulky je znázorněn na obr. 1.1.




*Obr. 1.1: Směrovací tabulka v operačním systému Windows 7*

Směrovací tabulka obsahuje tyto základní údaje:

**Cíl** – cílem může být hostitel, adresa podsítě, adresa sítě nebo výchozí trasa.

**Síťová maska** – 32-bitová adresa (stejně jako IP), která popisuje rozdělení sítě do podsítí.

**Brána** – IP adresa nejbližšího směrovače, na který mají být pakety odesílány.

**Rozhraní** – tento údaj značí, které rozhraní síťového prvku má být použito pro přenos paketů na další směrovač.

**Metrika** – vyjadřuje relativní cenu použití dané trasy pro přenos dat k cíli [1].

## 1.4 Směrovací algoritmy a protokoly

Proces vzniku, udržování a aktualizace směrovacích tabulek mají na starosti směrovací algoritmy. Pokud definujeme pro daný algoritmus přesná pravidla pro komunikaci a formát zpráv nesoucích směrovací informace, vznikne směrovací protokol.

Směrovací algoritmy můžeme rozdělit do dvou základních skupin: *statické* a *dynamické*.

### 1.4.1 Statické směrování

Jedná se o směrování na základě statických záznamu ve směrovací tabulce. Tyto záznamy přidává zpravidla ručně administrátor pomocí příkazů na konkrétním zařízení. Naproti tomu to, že konfigurace statického směrování je velmi jednoduchá a rychlá, má samotné směrování několik nevýhod:

- Administrátor musí znát kompletní topologii sítě,
- jakoukoli změnu v topologii sítě je nutné aplikovat ručně na každém směrovači, což je v případě velkých sítí velmi náročné,
- a tím pádem složitá správa už při malých sítích [6].

Právě kvůli těmto nevýhodám je statické směrování využívané jen ve velmi specifických případech a většinou v kooperaci se směrováním dynamickým.

### 1.4.2 Dynamické směrování

Dynamické směrování je proces směrování odlišný od statického v mnohých aspektech. Jeho základ tvoří použití směrovacích protokolů, jako jsou RIP (Routing Inforamation Protocol), IGRP (Interior Gateway Routing Protocol), OSPF (Open Shortest Path First), apod. Každý z těchto protokolu tvoří komunikaci mezi směrovači a umožňuje šíření informací týkající se sítí. Každá z těchto informací se využívá při aktualizování směrovací tabulky a celý proces se děje plně automaticky bez nutnosti zásahu administrátora. Oproti statickému směrování má dynamické následující hlavní výhody:

- Administrátor nemusí znát aktuální topologii sítě,
- jakékoli změny v topologii sítě se automaticky okamžitě šíří prostřednictvím směrovacích protokolu na všechna zařízení, které následně flexibilně na tyto změny zareagují,
- naproti složitější počítačové konfiguraci (v závislosti na zvoleném směrovacím protokolu) je potom administrace o mnoho jednodušší jako v případě statického směrování.

**Dělení dynamických protokolů:**

- **IGP (Interior Gateway Protocol)**

  - Používá se uvnitř autonomních systémů (sítě spojené z nějakého logického nebo geografického důvodu),
  - zástupci: RIP, IGRP, EIGRP (Enhanced Interior Gateway Routing Protocol), OSPF, IS-IS (Intermediate System To Intermediate System).

- **EGP (Exterior Gateway Protocol)**

  - Používá se mezi autonomními systémy,
  - zástupci: BGP (Boarder Gateway Protocol), EGP3 (Exterior Gateway Protocol).

**IGP můžeme ještě dále rozdělit:**

**a) Protokol pracující s vektorem vzdálenosti (Distance-Vector)**

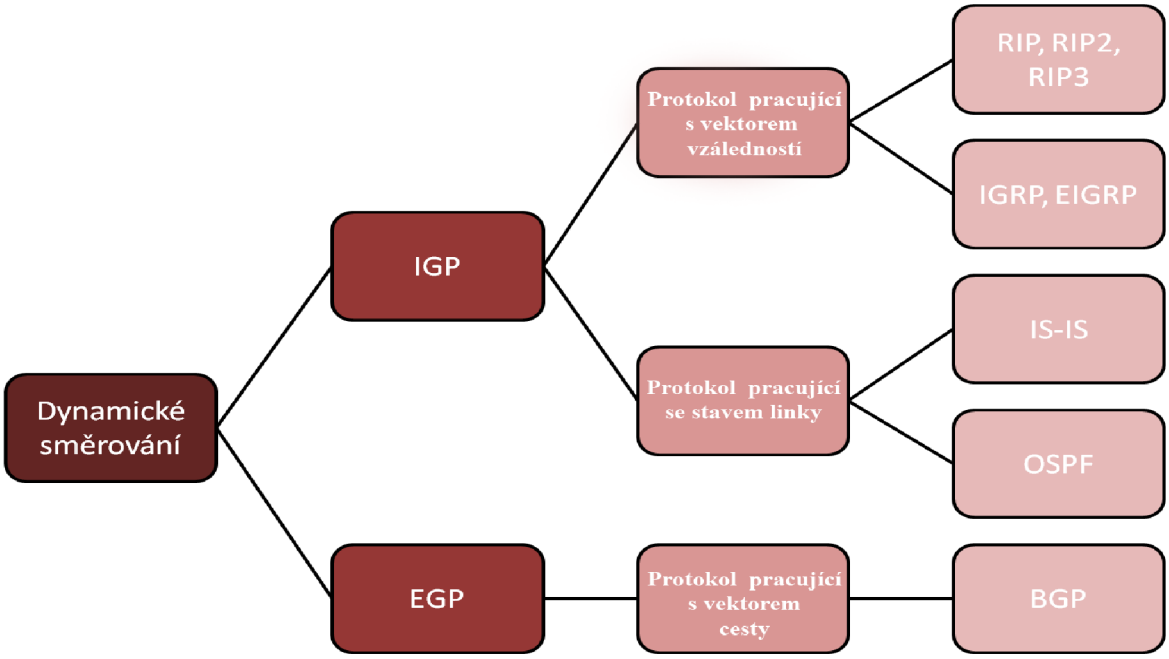
Protokoly z této rodiny fungují tak, že sousední zařízení si v pravidelných intervalech nebo při topologické změně (obvykle výpadek zařízení) vyměňují svoje kompletní kopie

směrovacích tabulek. Na základě obsahu těchto aktualizací doplňují nové informace a inkrementují určitou hodnotu, která je obvykle i metrikou udávající počet skoků k dané síti. Nevýhodou je, že zařízení znají topologii sítě jen na základě informací od svých bezprostředních sousedů.

**a) Protokol pracující se stavem linky (Link-State)**

Protokoly z této rodiny udržují kompletní informace o topologii dané sítě – zařízení jsou si vědoma všech ostatních zařízení v síti. Díky tomu je nutné udržovat podstatně více informací než jen směrovací tabulku. Typicky ještě například topologickou databázi, která je vystavena pomocí tzv. LSAs (Link-State Advertisments), jež si mezi sebou směrovače vyměňují. Tato dodatečná režie se projevuje tak, že směrovače potřebují více paměti a procesorového času a šířka pásma je na začátku značně vytížena inicializační tvorbou směrovacích a topologických tabulek. Na druhou stranu po této fázi je provoz na síti již minimální [6].

Pro přehlednost směrovacích protokolů výborně poslouží následující obr. 1.2.:




*Obr. 1.2: Schéma rozdělení protokolů pro dynamické směrování*

# 2. MANET SÍTĚ

MANET (Mobile Ad-hoc NETworks) jsou systémy samo-organizujících se sítí, které neobsahují žádnou pevnou strukturu. Prvky (mobilní stanice) takových sítí se často přemísťují a přesto je vyžadováno jejich permanentní připojení k dané síti. Je to řešeno tím, že všechny mobilní stanice v ad-hoc sítích mají v sobě mechanismy pro směrování a předávání paketů. Každý mobilní uzel MN (Mobile Node) buď generuje datový tok nebo je příjemce dat od jiného MN nebo pracuje jako směrovač přenášející datový tok k jinému MN. Z důvodu nepředpokládatelného umístění a pohybu MN v sítích MANET, běžné směrovací protokoly, používané pro bezdrátové sítě, nejsou pro tyto sítě vhodné [7]. Existuje mnoho variant sítí MANET a dělíme je dle rozdílných charakteristik, jako jsou:

- Velikost sítě (geografický rozsah a počet uzlů),
- pohyb uzlů,
- četnost topologických změn,
- požadavky na komunikaci,
- druh přenášených dat.

Z důvodu dynamické povahy těchto sítí je návrh komunikace a síťových protokolů poměrně náročný proces. Jeden z nejdůležitějších aspektů komunikačních procesů je návrh směrovacích protokolů použitých pro navázání a řízení více-skokových cest pro přenos dat mezi uzly. Mobilní a ad-hoc sítě mají několik význačných vlastností jako proměnná topologie, omezená šířka pásma, proměnná kapacita linek, energeticky proměnné algoritmy, omezená fyzická bezpečnost a další [7].

## 2.1 Třídění směrovacích protokolů v MANET sítích

Hlavní technický problém v MANET sítích je výběr dostatečného směrovacího protokolu, který si poradí s rychlými topologickými změnami. Směrovací protokol v ad-hoc sítích se mění v závislosti na typu sítě. V závislosti na způsobu aktualizací směrování, které jsou prováděny protokoly, je můžeme rozlišit do dvou kategorií:

- *Zaplavovací algoritmus*: Tyto algoritmy používají metodu zaplavení sítě směrovacími aktualizacemi. Následkem těchto aktualizací nutně dochází k nárůstu režií v síti. Na této metodě je založeno nejvíce protokolů jako například: AODV (Ad-hoc On-Demand Distance Vector), DSR (Dynamic Source Routing), GSR (Global State Routing), a další.
- *Algoritmus zpětné linky*: Tyto algoritmy pomáhají šetřit šířku pásma. Všechny záznamy o výpadcích linek jsou v tomto algoritmu použity pro obnovy cest**.** Dále vytváří k cíli spoustu alternativních cest, takže nedochází pouze k šetření šířky pásma v aktualizacích, ale také poskytuje alternativní cesty v případě výpadků.

V závislosti na účelu použití, protokoly mohou být děleny:

- *Jednosměrové (unicast)* – zahrnuje protokoly, které mohou být použity pro komunikaci na úrovni bod-bod.
- *Vícesměrové (multicast)* – zahrnuje protokoly, které mohou být použity pro komunikaci, kdy jeden účastník rozesílá data celé skupině nebo obráceně.
- *Geografické (geocast)* – zahrnuje protokoly, které mohou být použity pro zeměpisné účely.

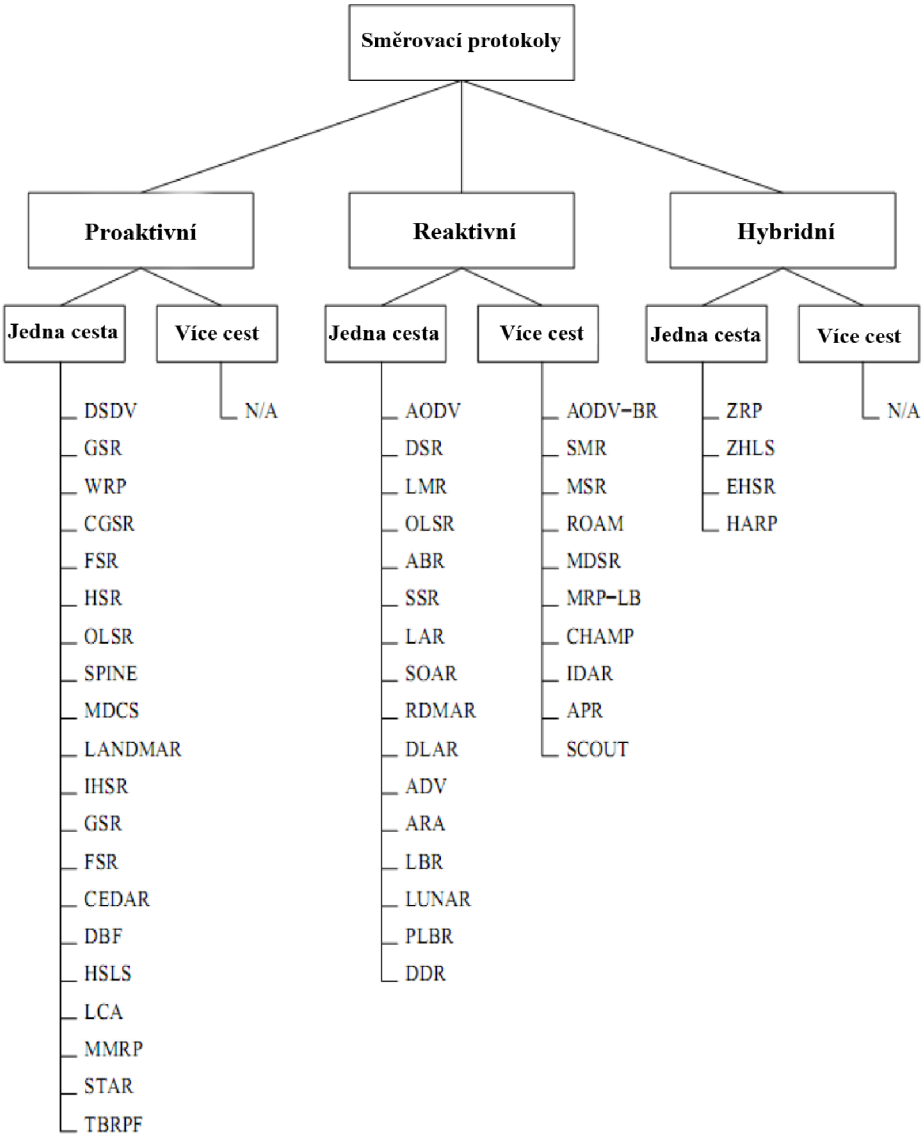
Jiný způsob pro třídění směrovacích protokolů může být založen na síťové hierarchii:

- *Rovinný* – všechny uzly jsou na stejné úrovni.
- *Hierarchický* – uzly jsou logicky nebo fyzicky rozděleny na rozdílnou úroveň dle hierarchie.

V závislosti na vícesměrové topologii jsou směrovací protokoly spojovány do dvou typů:

- *Stromová struktura* – existuje pouze jedna možná cesta mezi spojením zdroje a cíle. Tento protokol je dále dělen na topologie sdíleného a zdrojového stromu.
- *Síťová struktura* – zde existuje více možných cest mezi zdrojem a cílem a tyto protokoly jsou více rezistentní vůči změnám v síti [7].

Je mnoho kategorií, dle kterých mohou být MANET směrovací protokoly tříděny. Hlavní kategorie protokolů může být dále dělena na *proaktivní* (řízen tabulkou), *reaktivní* (na vyžádání) a *hybridní* (kombinace obou předchozích). Na obr. 2.1 je znázorněno přehledné schéma směrovacích protokolů sítí MANET.




*Obr. 2.1: Možnost dělení topology-based směrovacích protokolů v MANET sítích.*

### 2.1.1 Proaktivní směrovací protokoly

U proaktivních protokolů uzly periodicky vyhledávají směrovací informace uvnitř sítě. Kontrola režie těchto protokolů je předpokládaná, protože je nezávislá na přenosových profilech a má fixní horní mez. Toto je hlavní výhoda proaktivních směrovacích protokolů.

### 2.1.2 Reaktivní směrovací protokoly

Reaktivní protokoly reprezentují pravou povahu ad-hoc sítí, které jsou mnohem více dynamické než běžné infrastrukturní sítě. Místo opakujících se aktualizací směrovacích informací, reaktivní směrovací protokoly aktualizují směrovací informace pouze v momentě, když požadavek na směrování nastane, čímž se redukuje zatížení sítě. To má význam zejména v sítích s velkou mobilitou uzlů, kde by časté periodické aktualizace měly značný vliv na množství režijního provozu v síti.

### 2.1.3 Hybridní směrovací protokoly

Hybridní protokoly jsou směsicí vlastností obou výše zmíněných skupin. Strategie proaktivních protokolů je nalézat a řídit cesty v blízkosti uzlů, zatímco cesty do vzdálených uzlů jsou nalézány reakcemi. Následně režie a zpoždění, které je představováno proaktivními a reaktivními protokoly (v tomto pořadí), je minimalizováno. Hybridní protokoly byly známy pro jejich lepší škálovatelnost v menších sítích.

Další velice důležitou oblastí MANET sítí jsou směrovací protokoly založené na QoS (Quality of Service, tzv. QoS-based). QoS směrování vyžaduje nalezení cesty od zdroje k cíli, ale takovou cestu, která uspokojí QoS požadavky na spojení typu bod-bod, často uváděny výrazy šířka pásma nebo zpoždění. QoS je mnohem obtížnější zaručit v ad-hoc sítích než v naprosté většině sítí ostatních, protože bezdrátová šířka pásma je sdílena mezi přiléhající uzly a topologie sítě se neustále mění, jak se uzly pohybují. To vyžaduje rozsáhlou spolupráci mezi uzly při vytváření cest a při zabezpečení prostředků potřebných pro poskytování QoS [7].

# 3. ROZBOR SMĚROVACÍCH PROTOKOLŮ OSPFV3 A OLSR

## 3.1 OSPF (verze 3)

Tento směrovací protokol vychází z poměrně úspěšné řady proprietárních směrovacích protokolů nejkratších cest SPF (Shortest Path First), které se na trhu rychle prosazovaly. Všechny směrovací protokoly nejkratších cest, včetně protokolu OSPF, byly přitom postavené na matematickém algoritmu, známém jako Dijkstrův algoritmus. Ten provádí výběr cest podle stavů linek, nikoli jen podle vektorů vzdáleností.

### 3.1.1 Historie

Sdružení IETF (Internet Engineering Task Force) vytvořilo směrovací protokol OSPF koncem osmdesátých let. OSPF byl doslova a do písmene otevřenou verzí protokolů s nejkratšími cestami. Původní verze, označována jako OSPF verze 1, byla specifikována v dokumentu RFC 1131. Velice rychle ji ale nahradila značně rozšířená, zdokonalená verze, popsaná v dokumentu RFC 1247. Vzhledem k podstatnému zlepšení stability a funkčnosti protokolu byla označena jako OSPF verze 2. Tato verze protokolu se dočkala četných rozšíření, která byla na fóru IETF vytvořena jako otevřený standard. Následné specifikace byly publikovány v dokumentech RFC 1583, 2178 a poslední aktualizací RFC 2328 [10],[14].

Z důvodu neodvratně se blížícího vyčerpání adresního prostoru IPv4[1], se už postupně začíná zavádět adresní prostor IPv6, který mimo jiné potřebuje upravit protokolové zprávy, neboť ponesou adresu čtyřikrát delší, než v případě IPv4. Z těchto a i jiných důvodů vznikla poslední současně aktuální verze protokolu OSPF a to verze 3. Tato verze není ani tak zdokonalená verze 2, jako to bylo v případě přechodu z verze 1 na verzi 2, jako spíše nově vyvinutá. OSPF verze 3 je popsána v dokumentu RFC 2740 a aktualizace IPv6 v tomto protokolu je popsána v samostatném dokumentu RFC 5340 [3],[4].

### 3.1.2 Princip protokolu OSPF

Protokol OSPF byl navržen výslovně jako směrovací protokol pro sítě IP v rámci autonomních systémů. Jako takový nedokáže tedy přenášet pakety ostatních směrovatelných síťových protokolů, jako je IPX (Internetwork Packet Exchange) nebo AppleTalk. K výpočtu cest používá OSPF pouze cílové IP adresy, načtené z hlaviček paketů IP. Pro výpočty cest do jiných cílů než IP zde existují zvláštní opatření. Navíc jednotlivé zprávy OSPF jsou zapouzdřeny přímo do protokolu IP, pro doručování těchto informací tak nejsou potřeba žádné další protokoly, např. TCP (Transmission Control Protocol), UDP (User Datagram Protocol), apod.

[1] Dne 2. 3. 2011 na konferenci v Miami byly rozděleny poslední bloky IPv4 adres. Nicméně rezervy pro dané regiony (světadíly) jsou různé. V Asii budou vyčerpány do konce roku 2011, ale například v Africe postačí ještě na několik let.

Díky své konstrukci dokáže protokol OSPF rychle detekovat veškeré změny v topologii autonomního systému a konvergovat dle nové topologie. Rozhodnutí i směrování vychází přitom ze stavu linek, které propojují jednotlivé směrovače uvnitř autonomního systému. Všechny takovéto směrovače si udržují identickou databázi, která sleduje stav linek v síti. Součástí této databáze je také stav samotného směrovače, tedy informace o použitelnosti rozhraních, známých dosažitelných sousedech a informace o stavu linek.

Aktualizace směrovací tabulky, popisované jako oznámení o stavu linky LSA, se přímo rozesílají všem sousedům v oblasti daného směrovače. Formálně se tento aktualizační proces označuje jako *záplava*, i když tento nelichotivý pojem popírá skutečné výkonové charakteristiky protokolu OSPF.

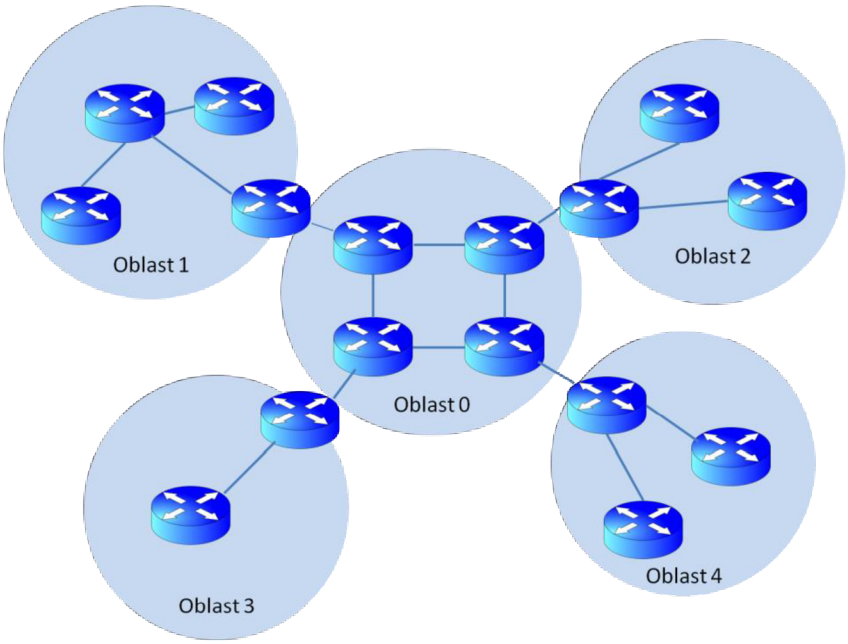
V praxi totiž sítě OSPF konvergují velice rychle, všechny směrovače v síti mají v provozu stejný směrovací algoritmus a potřebné aktualizace směrovacích tabulek si vzájemně odesílají přímo. Z vyměněných informací pak směrovač konstruuje obraz aktuální podoby sítě a jejích linek. Tento obraz podoby sítě ve směrovači má stromovou strukturu, kde vlastní směrovač je kořenem stromu. Celý strom označujeme jako strom nejkratších cest a sleduje nejkratší cesty do každého cíle v autonomním systému. Cesty do cílů umístěných mimo autonomní systém se zajišťují prostřednictvím tzv. výchozích bran (směrovačů) do příslušných externích sítí.

Jedním z podstatných důvodů rychlé konvergence protokolu OSPF jsou jeho oblasti. IETF stanovilo jako dva nejdůležitější cíle tyto následující [14]:

- Lepší škálovatelnost sítě,
- krátká doba konvergence.

Klíč k úspěšnému vyřešení obou úkolů spočívá v rozdělení sítí do menších částí označovaných jako *oblasti* (*areas*). Pod pojmem oblast zde rozumíme soubor síťově propojených koncových systémů, směrovačů a přenosových prostředků. Každou oblast definuje jednoznačné číslo oblasti, stanovené v konfiguraci směrovače. Rozhraní směrovačů, k nimž jsou definována stejná čísla, jsou tedy součástí stejné oblasti. Ideální je nedefinovat tyto oblasti zcela nahodile, ale jasně mezi nimi stanovit takové hranice, které by vedly k minimálním objemům provozu mezi různými oblastmi. Jinými slovy, jednotlivé oblasti by měly odrážet spíše skutečné vzorky provozu než geografické či politické hranice. Tato myšlenka je samozřejmě spíše teoretická a v konkrétním síťovém prostředí se může ukázat jako prakticky nepoužitelná. Počet oblastí podporovaných v síti OSPF je limitován velikostí pole s ID (Identification) oblasti. Pokud bude řeč o směrování mezi jednotlivými oblastmi, tak zde platí podmínka, že veškeré směrování musí probíhat přes oblast 0 a jednotlivé oblasti o nenulovém čísle spolu nesmí přímo komunikovat. Schematické rozdělení oblastí je znázorněno na obr. 3.1.

Toto hierarchické omezení zajišťuje elegantní škálovatelnost sítí OSPF bez složitého proplétání různých linek a směrovačů [14].




*Obr. 3.1: Schematické znázornění OSPF oblastí.*

### 3.1.3 Rozdíly mezi OSPFv2 a OSPFv3

Jak už bylo řečeno, tato aktuálně poslední zdokumentovaná verze protokolu OSPF je popsána v dokumentu RFC 2740. Existují některé významné podobnosti mezi vztahy protokolů RIPng - RIPv2 a OSPFv3 - OSPFv2. Mezi ty nejdůležitější patří, že OSPFv3 používá stejné základní mechanismy jako OSPFv2, a to SPF algoritmus. Dále zaplavovací zprávy, volba DR (Designated Router), použití oblastí a jiné. Konstanty a proměnné jako časovače a metrika jsou také stejné.

Jelikož OSPFv3 není zpětně kompatibilní s OSPFv2, tak pokud se budou používat v OSPF cesty IPv4 a IPv6, musí běžet současně jak protokol OSPFv2, tak i OSPFv3.

Kromě změn v LSA zprávách, které budou popsány dále, je několik změn v samotných OSPF procedurách a v této části si představíme ty nejdůležitější z nich:

- *Linkové protokolové procesy*

  Rozhraní k jedné lince může mít i více než jednu IPv6 adresu. Ve skutečnosti jedna linka může být součástí více podsítí a dvě rozhraní, připojených ke stejné lince ale patřících do rozdílných IPv6 podsítí, mohou stále komunikovat. OSPFv3 mění z OSPFv2 pojem „podsíť“ na „linku“ a dovoluje výměnu paketů mezi dvěma sousedy na stejné lince, ale patřící do jiné IPv6 linky.

- *Sousedi jsou vždy identifikování podle Router ID*

  OSPFv2 sousedi ve všesměrových a NBMA (Non-broadcast Multiple Access) linkách jsou identifikováni jejich adresami na rozhraních, zatímco sousedi na jiných typech linek jsou identifikováni RID (Routing Identification). OSPFv3 odstraňuje tuto

nesrovnalost a to tak, že všichni sousedi na všech typech linek jsou identifikováni pouze RID.

- *Rozsah zaplavování na lokálních linkách*

  OSPFv3 udržuje doménu AS (Autonomous System) a oblast zaplavovanou v rozsahu OSPFv2, ale s přidáním rozsahu pro lokální linky. Tento údaj se objevuje v nových linkových LSA zprávách, které slouží pro přenos informací sousedům propojených jednou linkou a tyto zprávy se už dál za hranice připojených směrovačů nešíří.

- *Použití místních linkových adres*

  OSPFv2 pakety mají rozsah na úrovni místních linkových adres a nejsou přenášeny žádným směrovačem. OSPFv3 používá místní linkové IPv6 adresy jako adresu zdroje a adresy dalších skoků.

- *Odstranění specifické OSPF autorizace*

  IPv6 má již v sobě zakomponován mechanizmus pro autorizaci. Z toho důvodu OSPFv3 už nepotřebuje další vlastní ověřování OSPFv3 paketů, stačí použití protokolu IPv6.

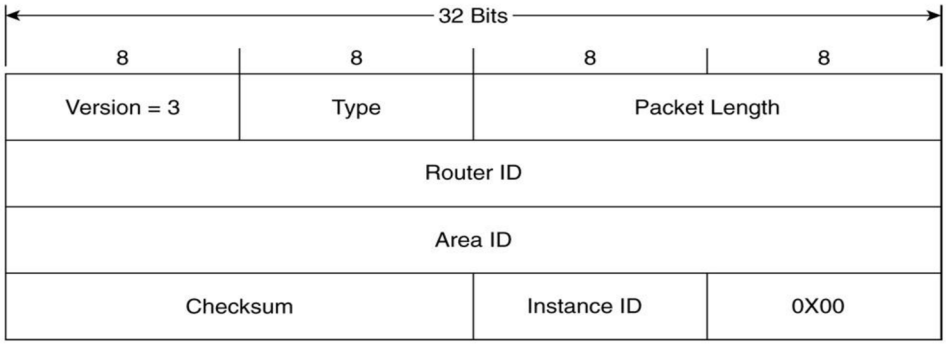
- *Více přizpůsobivé nakládání s neznámými typy LSA*

  Pokud OSPFv2 obdrží neznámý typ LSA, zahodí jej. OSPFv3 s nimi může zacházet buď jako se zaplavovacími pakety místní linky nebo je uchovat a rozeslat, pokud jsou rozpoznány, ačkoliv ignoruje jejich vlastní SPF algoritmus. To může znamenat jednodušší síťové změny a jednodušší integraci nových možností v OSPFv3 oproti OSPFv2 [4].

**OSPFv3 zprávy**

OSPFv2 a OSPFv3 mají oba stejné protokolové číslo 89 a jako OSPFv2 i OSPFv3 používá vícesměrové adresy kdykoliv je to možné. IPv6 vícesměnová adresa pro AllSPFRoutery je FF02::5 a AllDRoutery je všesměrová adresa FF02::6. Oba mají rozsah místních linek. Zde zcela zřejmě vidíme podobnost u OSPFv2 adres 224.0.0.5 a 224.0.0.6.

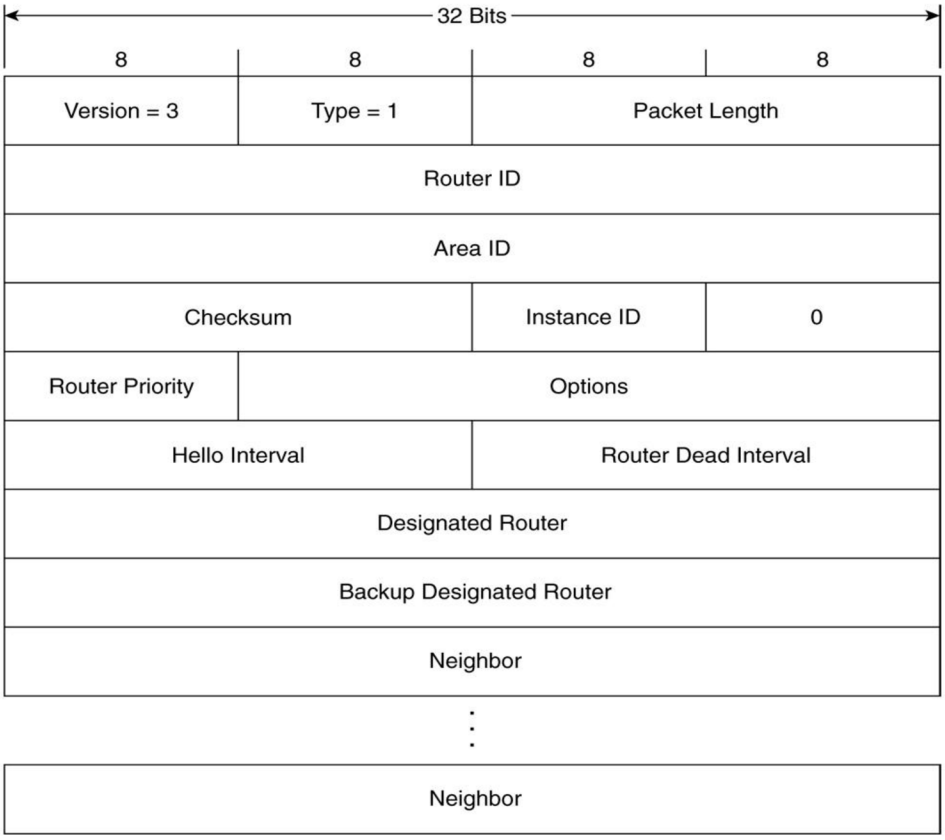
OSPFv3 používá stejných 5 typů zpráv jako OSPFv2, tedy HELLO, DD (Database Description), LS Database Request, LS Database Update a LS Acknowledgement a čísluje je také stejně. Hlavička zpráv, zobrazena na obr. 3.2, je už oproti OSPFv2 rozdílná. Verze je samozřejmě 3 místo 2, ale více důležité je, že chybí pole pro autorizaci. Jak už bylo popsáno v předchozí podkapitole, OSPFv3 používá pro autorizaci rozšíření v hlavičce IPv6 paketu samotného. Dále je tu Instance ID, které dovoluje současný běh více OSPFv3 instancí na jedné lince. Toto ID má význam pouze na lokální lince, protože OSPFv3 zprávy nejsou posílány mimo linku, kde vznikly.




*Obr. 3.2: Hlavička paketu OSPFv3*

Kromě hlavičky paketu je potom rozdílný formát zprávy v OSPFv3 oproti OSPFv2 jen v„HELLO“ zprávách a ve zprávách popisujících databáze.

Obr 3.3 znázorňuje formát OSPFv3 HELLO zprávy. Oproti OSPFv2 zde není pole pro síťovou masku, protože IPv6 ji nepotřebuje. Kromě toho stejné pole (pod hlavičkou) se objevuje v obou paketech. Pole *Options* (Nastavení) vzrostlo na 24 bitů a *Router Dead Interval* se snížil z 32 na 16 bitů. Důsledkem toho se teoretické maximum této hodnoty snížilo ze 4.3 miliardy na 65,535 sekund. Tato změna má ale malý až téměř žádný vliv na funkci sítí. Maximální konfigurovatelná hodnota pro Router Dead Interval u většiny OSPF implementací už stejně byla dlouhou dobu po maximálně tuto hodnotu a změna v tomto poli je pouze úspora nevyužitého prostoru.

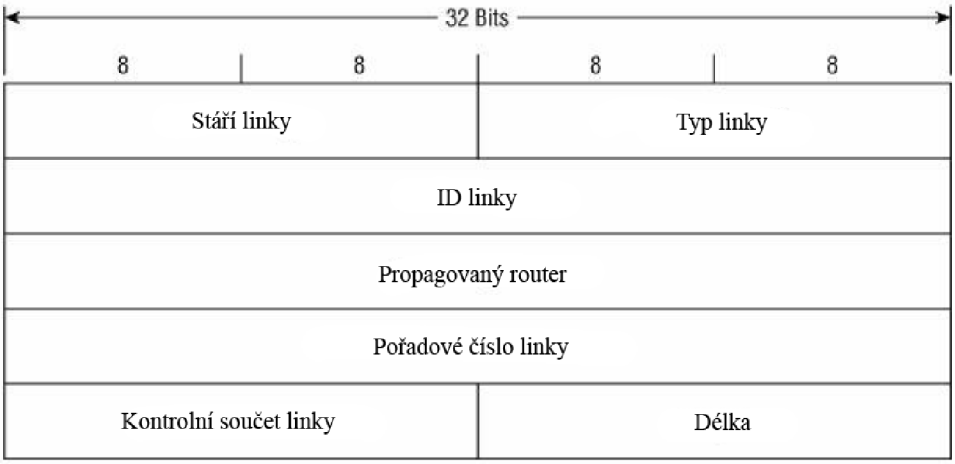



*Obr. 3.3: Paket HELLO zprávy OSPFv3.*

Ostatní tři typy zpráv – Link State Request, Link State Update a Link State Acknowledgement mají téměř stejný formát jako OSPFv2 [4].

**LSA zprávy**

Link State Advertisement (LSA) je základ komunikace prostřednictvím OSPF směrovacího protokolu pro IP protokol. Oznamuje místní směrovací topologii směrovače ostatním směrovačům, které jsou ve stejné OSPF oblasti. OSPF je navržen pro škálovatelnost, tedy některé LSA zprávy nejsou rozesílány na všechny rozhraní, ale pouze na ty, které patří do dané oblasti. Tímto způsobem mohou být informace detailně udržovány, zatímco hromadné informace jsou rozesílány do zbytku sítě. Zprávy LSA jsou rozděleny do 11 kategorií. Na obr. 3.4 vidíme tvar hlavičky paketu LSA zprávy, který má 128 bitů. Po 128. bit jsou všechny zprávy stejné a liší se až těmi následnými, které mají teoreticky neomezenou délku.




*Obr. 3.4: Hlavička LSA zprávy.*

## 3.2 OLSR

OLSR je proaktivní algoritmus patřící do kategorie protokolů sledujících stav linek (link-state). Jejich charakteristická vlastnost je koncept takzvaných vícebodových přenosových spojení MPR (Multi-protocol Routing). Protokol OLSR je nejčastěji používán v sítích, které jsou tvořeny pouze mobilními uzly, a tedy neobsahují žádný centrální řídicí prvek.

### 3.2.1 Algoritmus

OLSR používá 2 typy zpráv: „HELLO“ a „TC“ (Topology Change). Zprávy typu „HELLO“ jsou používány všemi uzly pro určování jejich přímých sousedů a pro dva skoky dále (pozn.: přímí sousedi sousedů, kteří nejsou sousedy uzlu). Tedy na tomto základu jsou navrženy MPR uzly, jinými slovy podmnožina uzlů, kde každý uzel byl oddělen nejvíce

dvěma skoky od jiného uzlu v dané podmnožině. Uzly jsou ekvivalenty MPR DR (Designated Router) v OSPF algoritmu.

TC zprávy jsou posílány pravidelně mezi uzly a informují uzly o potenciálních změnách v síťové topologii. Všechny uzly, které obdrží TC paket, aktualizují jejich databázi a optimální trasu [7].

### 3.2.2 Použití

OLSR je dobře škálovatelný algoritmus a je vhodný pro malé i velké bezdrátové sítě. Protože odpovídá protokolům pracujících na principu sledování linky, uzly vyžadují relativně velkou paměť a výpočetní výkon, takže nejsou příliš vhodné pro senzorové sítě.

### 3.2.3 Vlastnosti protokolu OLSR

Tento protokol je docela vhodný pro velké a husté sítě a to především díky optimalizaci dobře použitelného MPR algoritmu. Díky němu se může použitím tohoto protokolu dosáhnout lepších výsledků v porovnání s běžnými protokoly pracujících na principu sledování linek.

**Hlavní vlastnosti OLSR:**

- Základní předpoklad algoritmů protokolů na principu sledování stavu linek, je synchronizace databáze ve vybraných uzlech (DR v OSPF, OLSR MPR). Toto brání ve výskytu vzniku smyček v paketových cestách. V případě sítí MANET je náročné zaručit doručení paketu každému uzlu. Vzhledem k tomu OLSR předpokládá, že TC zprávy jsou zasílány dostatečně často a že potenciální proměnlivý stav databáze potrvá nejkratší dobu, jak jen to bude možné.
- Použitím pouze sady MPR uzlů pro vytvoření cest může redukovat množství paketů v síti.
- Vysoká mobilita sítě má za následek posílání násobků menších TC paketů a ve výsledku velký nárůst odesílaných paketů.
- Ve zvláštních případech rozpadu spojení mohou být dočasné smyčky vytvořeny ještě předtím, než se rozšíří aktualizace.
- Základní verze OLSR algoritmu nezajišťuje detekci kvality linky (linka funguje nebo ne). Protože v sítích MANET linky nejsou vždy funkční, existují vyvinuté rozšíření pro OLSR algoritmy, které přidávají funkčnost.

**Hlavní přednosti OLSR:**

- Má dostupné cesty uvnitř standardní směrovací tabulky, což může být užitečné pro různé systémy.
- Síťové aplikace, ve kterých není zpoždění vytvořené hledáním cest, které je běžně s tímto procesem spojené.
- Generované zátěže při směrování, které jsou obecně větší než u reaktivních protokolů, nenarůstají s počtem použitých cest.

- Hodnoty časů vypršení a platnosti informace jsou zahrnuty uvnitř zpráv přenášejících informaci umožňující použít rozdílné hodnoty časovače v rozdílných uzlech.

**Hlavní nedostatky OLSR:**

- Původní rysy OLSR nezahrnují žádná opatření pro snímání nebo dohled nad kvalitou sítě. Jednoduše se předpokládá, že linka je aktivní, pokud nějaký počet paketů byl nedávno doručen. To předpokládá, že linky jsou bi-modální (buď pracují, nebo mají výpadek), což není bezpodmínečný případ v bezdrátových sítích.
- OLSR používá paměť a síťové prostředky v upřednostňování šíření dat do potenciálně nevyužité cesty. Zatímco toto není problém pro pevné přístupové body a notebooky, OLSR se tím stává nevhodné pro senzorové sítě, které se snaží být většinu času v režimu spánku.
- Tím, že se jedná o protokol na principu sledování stavu linky, OLSR vyžaduje větší množství šířky pásma a procesorové paměti pro výpočet optimálních cest v síti. V typických sítích, kde OLSR je používáno (které vzácně překročí několik stovek uzlů), toto ovšem neznamená větší problém [7].

### 3.2.4 MPR (Multi Point Relaying)

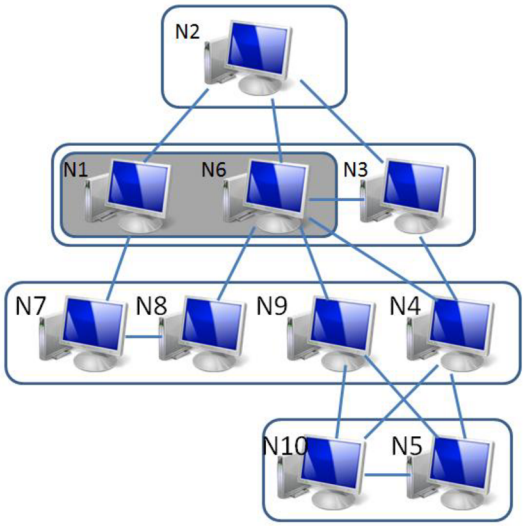
Hlavní rozdíl mezi OLSR a jeho předchůdcem LSR je ten, že OLSR navíc spoléhá na tzv. MPR uzly. Primární účel MPR je omezení zbytečného vysílání servisních, všesměrových zpráv v síti. Pakety se stejnou informací je zbytečné doručovat vícekrát v té samé oblasti sítě. MPR pomáhá tento problém redukovat. Pomocí MPR je navíc část topologie zjednodušena, jelikož směrování k danému uzlu probíhá jen přes jeho MPR.

MPR se stává vybraný uzel z okolí (vzdálenost na jeden skok) optimalizovaný podle daných mechanizmů. Každý uzel následně informuje své přímé sousedy v „HELLO“ zprávě (vzdálenost na jeden skok) kdo je jeho MPR. Pokud uzel obdrží informaci, že je pro jiný uzel MPR, tak si tuto informaci zapamatuje a začne tomuto uzlu poskytovat služby, které se od něj jako od MPR čekají – to je předávat dále všesměrové OLSR zprávy (hlavně zprávy TC) [12].

**Příklad použití MPR**

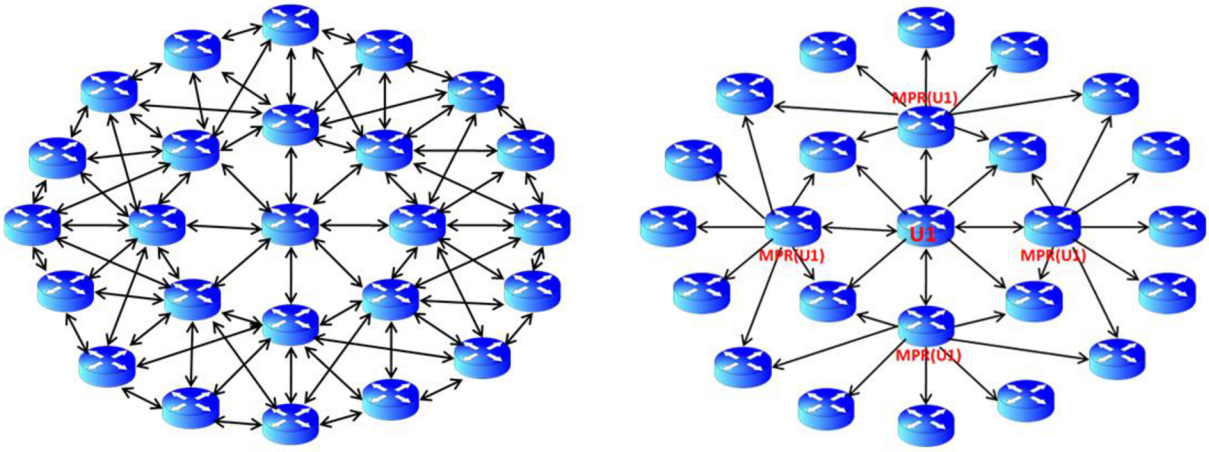
Princip MPR je znázorněn na obr. 3.5. Uzel v síti, v tomto případě uzel N2, si vybere potřebný počet sousedních uzlů v síti tak, aby přes ně měl dosažitelné všechny sousedy vzdálené na dva skoky. Těmto vybraným uzlům (N1 a N6) oznámí, že si je vybral jako MPR. Jsou tedy MPR pro uzel N2 a musí začít poskytovat služby MPR. Nyní se na uzly N1 a N6 uplatní důležité pravidlo a to, že jen uzel, který byl vybrán jako MPR, musí předávat dále všesměrové OLSR zprávy (od uzlu N2). Když tedy obdrží N1 nebo N6 všesměrovou zprávu, které je tvůrcem uzel N2, musí ji poslat dále. Ale uzel N3, který není MPR pro uzel N2, sice přijme všesměrovou zprávu od N2 a zpracuje ji pro svoji potřebu, ale dále ji už předávat nemusí.

Tím se šetří přenosová kapacita a zjednodušuje topologie. Uzel N2 je pro ostatní uzly dosažitelný jen přes uzly, které si vybral jako MPR tedy N1 a N6.




*Obr. 3.5: Příklad použití MPR v OLSR.*

Srovnání celkové topologie při normálním směrování a zjednodušení cest při použití *MPR* uvádí obr. 3.6. Na obr. 3.6 vlevo je znázorněna topologie, která by vznikla z pohledu uzlu U1 bez použití MPR. Na obr. 3.6 vpravo jsou znázorněny červeně *MPR*, které si uzel U1 vybral. Z obrázků je patrné, že se výrazně sníží počet cest, které jsou nutné k vytvoření celkové topologie, nutných k zachování komunikace se všemi uzly vzdálenými dva skoky od uzlu U1. Tím je redukována velikost rozesílaných informací a dále informace, které by jinak byly duplicitní, nejsou zbytečně a opakovaně vysílány v místní části sítě [12].



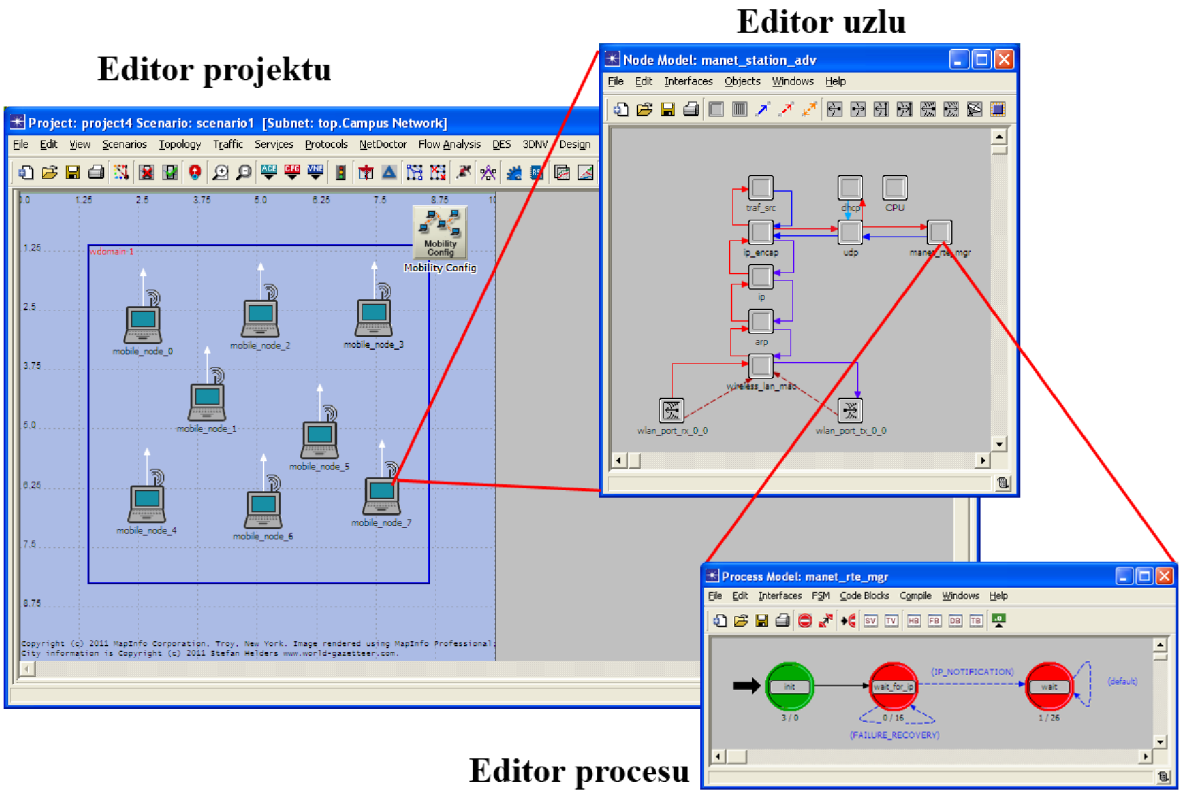

*Obr. 3.6: Směrovací cesty s a bez použití MPR.*

# 4. PROSTŘEDÍ OPNET MODELER

Program OPNET Modeler (OM) je simulační prostředí vyvinuté firmou OPNET Technologies Inc. a slouží pro návrh, simulaci a analýzu různých síťových technologií a mechanismů. Velice efektivně a podrobně dokáže modelovat chování rozsáhlých heterogenních sítí včetně komunikačních protokolů pracujících na různých úrovních modelu sítě [11].

## 4.1 Editory

OPNET Modeler nabízí možnost použití velkého množství editorů, které umožňují vytvářet modely sítí a nastavovat jejich parametry s různým stupněm abstrakce. Ve většině případů se používají tři základní editory a to editor projektu (Project Editor), editor uzlů (Node Editor) a editor procesů (Process Editor). Jejich hierarchická struktura je uvedena na obr. 4.1.




*Obr. 4.1: Základní struktura editorů v OM.*

### 4.1.1 Project Editor

Je hlavní grafický editor modelující topologii a komunikaci v síti. Síť obsahuje jednotlivé uzly a odkazy na objekty konfigurovatelné přes dialogový box. Je možné použít objekty z knihovny OM Model Library, nebo si vytvořit vlastní uzly a modely. Projektový editor má v sobě implementovány mapy světa, díky nimž je struktura a fyzické rozložení sítě názornější.

Samotný projekt sítě je pak tvořen jedním nebo více scénáři, které umožňují vytvoření odlišných konfigurací sítě v rámci jednoho projektu.

### 4.1.2 Node Editor

Představuje rozhraní nižší úrovně než projektový editor. Ukazuje architekturu síťového zařízení nebo systému a jeho vzájemné vztahy mezi funkčními modely a volanými funkcemi. Model uzlu se sestává z modulů uzlů, které jsou vzájemně propojeny datovými cestami a to buď tokem paketů či statickými spoji. Každý modul může generovat, posílat a přijímat pakety od ostatních modulů uvnitř celého uzlu. Moduly typicky představují aplikace, protokolové vrstvy, algoritmy a fyzické prostředky jako jsou zásobníky, porty a sběrnice.

### 4.1.3 Proces Editor

Je rozhraní nejnižší úrovně. Procesní editor je konečný stavový automat přizpůsobený snaze specifikovat všechny úrovně modelu do detailu (protokolu, prostředku aplikaci a algoritmu). Stavy a přechody jsou definovány v grafickém diagramu. Každý stav a proces modelu obsahuje kód v C/C++ podporovaný rozsáhlou knihovnou s funkcemi vytvořenými pro protokolové programování. Každý stav obsahuje vstupní exekutivu a výstupní exekutivu. Každý automat může být definován odděleně ve stavu a může být volán z uživatelské knihovny. [9]

# 5. PRAKTICKÁ ČÁST PRÁCE

Pro praktickou část bylo přiděleno zpracování pouze směrovacího protokolu OLSR a práce je rozdělena na 5 částí.

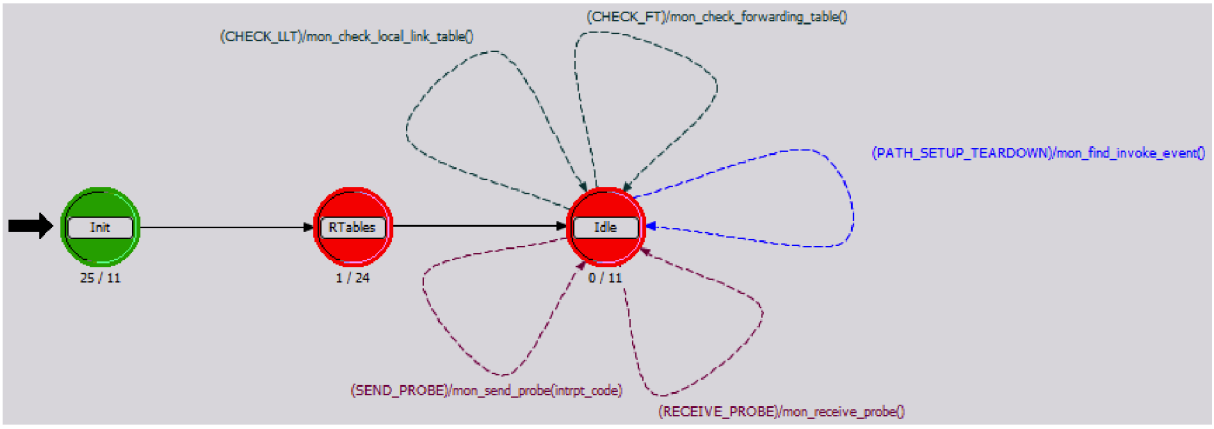
## 5.1 Struktura procesních modelů

V této části je probrána struktura procesních modelů v OPNETu, dále je rozebrán editor procesů a popsány základní prvky, stavy a přechody. V poslední části jsou popsány proměnné a bloky editoru procesů, které obsahují zdrojové kódy.

### 5.1.1 Editor procesů

Editor procesů je rozhraní nejnižší úrovně. Slouží pro tvorbu konečného stavového automatu FSM (Finite State Machines) přizpůsobeného snaze specifikovat všechny úrovně modelu do detailu.

Stavy a přechody jsou definovány v grafickém diagramu, viz obr. 5.1. Každý stav a proces modelu obsahuje kód v C/C++ podporovaný rozsáhlou knihovnou s funkcemi vytvořenými pro protokolové programování [9].




*Obr. 5.1: Ukázka stavového automatu editoru procesů*

### 5.1.2 Popis základních prvků editoru procesů

Mezi základní prvky editoru procesů patří:

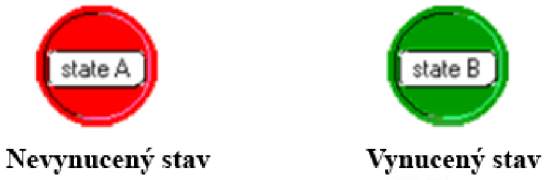
- **stav (state)** – představuje stav procesu. Proces může být např. ve stavu čekání na zprávu od vysílače. Graficky je znázorněn červeným nebo zeleným kruhem (viz obr. 5.1).

- **přechod (transition)** - představuje změnu mezi jednotlivými stavy. Může být spojený s určitou událostí (např. přijetí paketu). Je označen tenkou šipkou (viz obr. 5.1).
- **počátek (initial)** – označuje počáteční stav. Tento stav procesu bude vykonaný jako první. Je označen tlustou černou šipkou (viz obr. 5.1) [13].

Dále dělíme *stav (state)* na dva základní typy:

- **Vynucený (Forced)** – v editoru procesů je znázorněn zeleným kruhem. Při přechodu do tohoto procesu se vykoná kód, který tento stav obsahuje ve vstupní (Enter Executive) i výstupní (Exit Executive) pozici a automaticky dojde k přechodu do dalšího stavu. U tohoto stavu je jedno, jestli kód bude vložen do vstupní nebo výstupní pozice, protože vždy budou vykonány obě části kódu. Pro přehlednost je ale doporučeno vkládat kód u vynuceného stavu pouze do vstupní pozice.

- **Nevynucený (Unforced)** – v editoru procesů je znázorněn červeným kruhem. Po přechodu do tohoto stavu v něm proces zůstává tak dlouho, dokud nedojde k další události. Každá událost je definována přerušením. Při přechodu do nevynuceného stavu, se provede pouze kód ze vstupní pozice a poté se vyčká na určité přerušení. Pokud přerušení přijde, vykoná se kód z výstupní pozice [15].




*Obr. 5.2: Dva základní stavy (vynucený a nevynucený)*

I další prvek *přechod (transition)* je možné dále dělit na dva základní typy:

- **Podmíněný (Conditional)** – označuje přechod, kdy je stanovena podmínka, která určuje pravidla přechodu do nového stavu. Je-li tato podmínka splněna, uskuteční se přechod do dalšího stavu. Podmíněný přechod se v editoru procesu vyznačuje přerušovanou čarou a v blízkosti této čáry se nachází jméno podmínky.

- **Nepodmíněný (Unconditional)** - označuje přechod, který přechází okamžitě do dalšího stavu. Nepodmíněný přechod se v editoru procesu vyznačuje plnou čarou [15].

### 5.1.3 Proměnné a bloky editoru procesů

V editoru procesů se dále deklarují názvy jednotlivých proměnných a bloků. Proměnné lze deklarovat třemi následujícími způsoby:

- **SV – State Variables (stavové proměnné)** – zde se deklarují proměnné, které mají platnou hodnotu, i když proces přechází z jednoho do druhého stavu. Tyto hodnoty jsou dostupné také pro blok FB (Function Block).
- **TV – Temporary Variables (dočasné proměnné) –** zde se deklarují proměnné, které mají svou hodnotu platnou vždy jen v rámci jednoho procesu. Na rozdíl od SV nejsou dostupné pro blok FB.
- **HB – Header Block (hlavičkový blok) –** tento blok definuje aktuální hlavičku daného procesu. Proměnné zde zapsané lze využít pro blok FB. V bloku HB jsou obvykle dále definována jednotlivá přerušení pro přechody do různých procesů.

Dále lze definovat bloky:

- **FB – Function Block (funkční blok)** – do tohoto bloku je možné zapsat funkce napsané v jazyce C/C++, které následně mohou být použity v kódu jednotlivých procesů.
- **DB – Diagnostic Block (diagnostický blok)** – tento blok obsahuje funkce, které zasílají diagnostické informace na standardní výstup zařízení. Často je využívána funkce *printf()*. Diagnostický blok má plný přístup ke stavovým proměnným (SV), ale nemá přístup k dočasným proměnným (TV).
- **TB – Termination Block (ukončovací blok) –** blok slouží pro dealokaci paměti zabrané dynamicky vytvořenými proměnnými [11].

## 5.2 Generování a příjem zpráv pomocí funkce ICI

V této části je probráno vytvoření procesního modelu umožňujícího vygenerování a příjem zpráv pomocí funkce ICI (Interface Control Information). Výstupem bude minimalistický projekt v OPNET Modeleru vytvořený na základě studentských prací z předchozích let, v tomto případě převážně z prací [2] a [8].

### 5.2.1 Datová struktura ICI

ICI je datová struktura, která slouží jako druh určité mezi-procesové komunikace. ICI využívá určitého typu přerušení, pokud proces uvedl ICI do funkce předtím, než se udála akce, která přerušení způsobila. Hlavní použití ICI zpráv je v rozhraní vrstvového protokolu, ale mohou být také použity ke spojení informace s propracovanými vlastními přerušeními nebo peer-to-peer vzdálenými přerušeními. ICI jsou dynamické simulační objekty, protože jsou vytvářeny a zase rušeny podle potřeby během provádění některého z procesů. Základní příkaz pro jejich vytvoření je *op_ici_create()*.

ICI je formát, který se vytváří pomocí *ICI Format Editoru*, jenž obsahuje seznam atributů a datových typů podporovaných nově vytvořenými ICI. Pokud není existující ICI používáno po delší dobu, může být zrušeno pomocí příkazu *op_ici_destroy()*. Takové zrušení znamená uvolnění paměti, kterou si ICI pro sebe zabralo. Často jsou ICI zrušeny pomocí

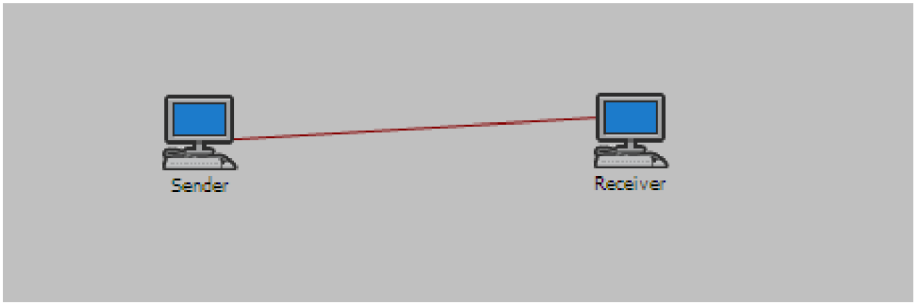
procesu, který je přijímá, protože přijímající proces je ten nejvíce informovaný o tom, kdy už ICI splnily svoji funkci [13].

### 5.2.2 Generování a příjem zpráv pomocí funkce ICI

Jelikož cílem tohoto úkolu a vůbec celé diplomové práce není přímo vytvořit model pro přenos zpráv pomocí funkce ICI, bude se postup vytváření odkazovat na práci [2], dle které bude tato problematika probrána.

#### ➢ Vytvoření projektu

- Do nově vytvořeného projektu se vloží dva uzly z rodiny Ethernet a to **ethernet_wkstn** (Fixed Node), které se vzájemně propojí pomocí duplexní linky **100BaseT**.
- Vytvořené uzly se pojmenují například **Sender** a **Receiver**. Nyní bude scénář vypadat přibližně jako na obr 5.3.
- Na závěr se stanicím automaticky přiřadí IP adresy pomocí menu **Protocols -> IP -> Addressing -> Auto-Assign IPv4 Addresses**.



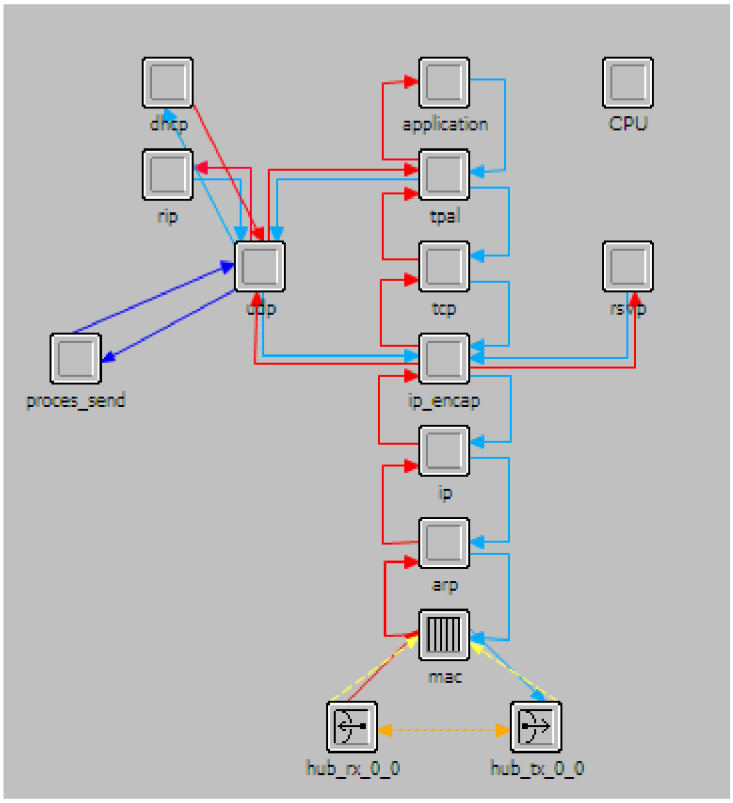

*Obr. 5.3: Schéma scénáře*

#### ➢ Vytvoření uzlu a procesního modelu

Nyní je zapotřebí vytvořit nový uzel, který bude vycházet ze stávajícího, ale bude doplněný o další procesní model.

- Dvojklikem na jeden z uzlů (začne se uzlem **Sender**) se otevře editoru uzlů. Pomocí menu **Objects -> Create Processor** se vloží nový proces někde poblíž procesu **udp** a ten se pojmenuje například **proces_send**.
- Nyní je potřeba mezi procesy **udp** a **proces_send** vytvořit komunikaci a to pomocí menu **Objects -> Create Packet Stream**. Komunikace bude v obou směrech (viz obr. 5.4)

- Editor uzlů se pojmenuje například **ICI_proces_send**.[2]
- Nyní je potřeba se vrátit do editoru projektu a stanici **Sender** přiřadit nově vytvořený uzel. To se provede kliknutím pravým tlačítkem na stanici a výběrem **Edit Attributes (Advanced)**. V nově otevřeném okně změnit hodnotu **model** na nově vytvořený **ICI_proces_send**, který lze nalézt (po kliknutí na **Edit…**) ve složce **ICI**.
- Změnu potvrdit a obdobným způsobem se upraví editor uzlů pro stanici **Receiver** pouze s tím rozdílem, že proces se bude jmenovat **proces_receive** a model bude uložen jako **ICI_proces_receive**.



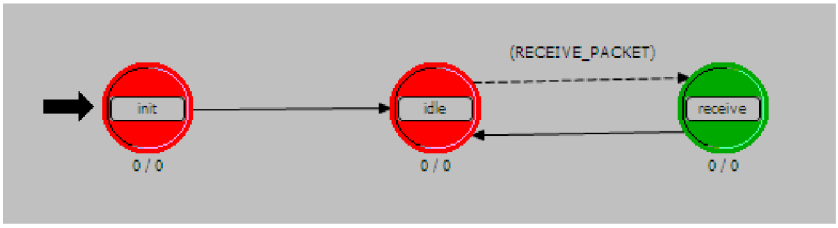

*Obr. 5.4: Model uzlu „Sender“*

## ➢ Vytvoření procesního modelu

- Dvojklikem na nově vytvořený proces **proces_send** se zobrazí prázdné okno editoru procesů.
- Budou potřeba 3 stavy, které se vloží pomocí **FSM -> Create State**. Stavy se pojmenují popořadě **init, idle a send.** Dále je potřeba mezi nimi vytvořit přechody pomocí **FSM -> Create Transition** (mezi **idle** a **send** bude přechod tam i zpět). Stav **send** bude vynucený a to se docílí kliknutím pravým tlačítkem na stav a volbou **Make State Forced**.

[2] Je vhodné si veškeré upravené modely ukládat pod jinými názvy, aby nedocházelo k přepisováním výchozích souborů OPNETu.

- Přechod, který vede od **idle** k **send** se bude jmenovat **SEND_PACKET** a to se docílí tak, že se klikne pravým tlačítkem na přechod, zvolí se **Edit Attributes** a název se vepíše do pole **condition.**
- Nyní se model uloží jako například **process_model_sender**. Ještě je zapotřebí nově vytvořený procesní editor přiřadit k procesu **proces_send**. To se provede kliknutím pravým tlačítkem na daný proces, volbou **Edit Attributes** a v kolonce **proces_model** se stávající nahradí nově vytvořeným **process_model_sender**. Aby se inicializace spustila ihned od začátku, změní se parametr **begsim intrpt -> enable**.
- Obdobně se vytvoří procesní model pro stanici **Receiver** a proces **proces_receive**. Stavy se budou jmenovat **init, idle** a **receive**. Přechodu **idle -> receive** se přiřadí stav **RECEIVE_PACKET**.
- Procesní model se uloží jako **process_model_receiver** a je potřeba jej opět přiřadit modelu **proces_receive** a nezapomenout opět nastavit pole **begsim intrpt -> enable**. Výsledek může vypadat jako na obr. 5.5.



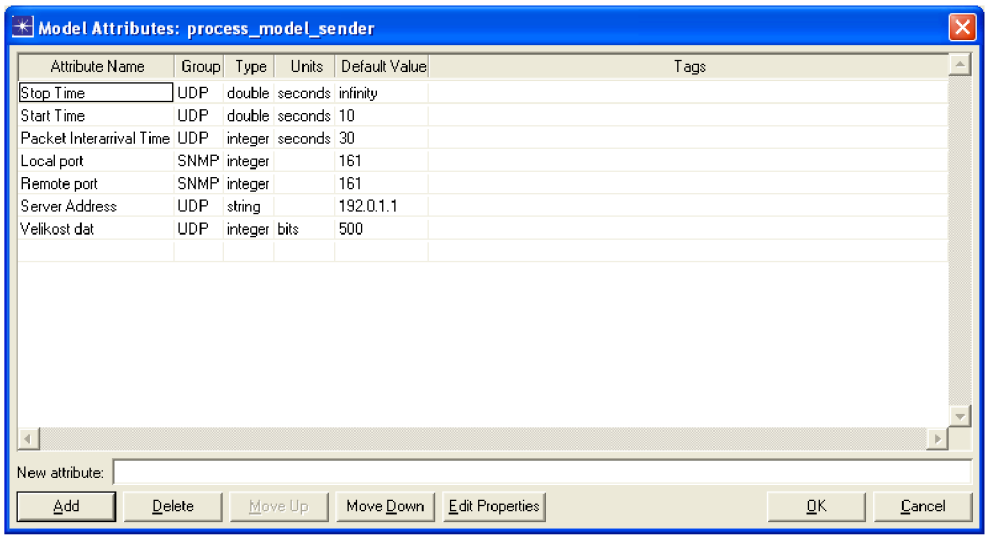

*Obr. 5.5: Procesní model Receiveru*

### ➢ Zdrojový kód procesního modelu uzlu Receiver

Nyní se musí doplnit zdrojový kód pro jednotlivé vstupní a výstupní stavy a stavové proměnné. Samotný zdrojový kód v této práci uveden není, ale je součástí přílohy práce [2], kde je zdrojový kód uvedený kompletní a v práci byl použit téměř beze změn.

- Nejprve se doplní zdrojový kód na straně vysílací stanice (Sender) a to přímo v nově vytvořeném procesním modelu **process_model_sender**.
- Stavové proměnné (blok **SV**) – zde se deklarují proměnné, které dále při vytváření zprávy používají a jsou dostupně také pro blok FB. Například místní a vzdálený port, ID uzlu, start a konec přenosu a další.
- V proměnné **TV** se deklarují proměnné, které se používají pouze v rámci jednoho procesu, například pro odesílání a přijímání paketu.
- Proměnná **HB** obsahuje aktuální hlavičku daného procesu. Proměnné, co jsou deklarované zde, jsou taktéž dostupné i v bloku FB. Definují se zde například přechody a přerušení
- Nyní se doplní zdrojový kód do vstupních a výstupních pozic jednotlivých stavů a to vždy kliknutím pravým tlačítkem kurzoru na daný stav a volbou **Edit Enter Execs** nebo **Edit Exit Execs** podle toho, jestli je cílem vložit kód pro vstupní nebo výstupní pozici.

- Vstupní pozice stavu **init** provádí načítání a získávání různých ID (vlastní, rodič, uzel) a získávání atributu uzlu. Ve výstupní pozici se provádí registrace nového portu, na kterém bude komunikace probíhat a při úspěšném dokončení se port vypíše do konzole.
- Ve vstupní pozici stavu **idle** se zjišťuje, jestli se bude paket posílat nebo ne a na základě výsledku se provede či neprovede přerušení. Ve výstupní pozici se pouze uloží kód přerušení do proměnné.
- Ve vstupní pozici stavu **send** se provede vytvoření nového paketu, nastaví se velikost a vytvoří se nová zpráva ICI, které se následně odešle do UDP streamu.
- Na závěr je potřeba doplnit atributy modelu pomocí **Interfaces -> Model Attributes** a to podle obr. 5.6.
  Zde se nastaví hodnoty startu a konce vysílání zpráv ICI, dále v jakých intervalech se budou posílat, port místního a cílového uzlu, na kterém budou komunikovat, zdrojová IP adresa a velikost přenášených dat.




*Obr. 5.6: Atributy modelu „process_model_sender“*

## ➢ Zdrojový kód procesního modelu uzlu Receiver

Naplnění proměnných a pozic u procesního modelu **process_model_receiver** probíhá opět obdobně.

- Blok **SV** obsahuje deklaraci proměnných pro zpracování ICI paketu a uvolnění paměti.
- Bloky TV a HB mají velice podobné parametry, jako tomu bylo v případě odesílajícího uzlu.
- Ve vstupní pozici stavu **init** se opět provádí načítání a získávání různých ID (vlastní, rodič, uzel), dále získávání atributů o vysílajícím uzlu. Ve výstupní pozici se otevírá nová relace a registruje se nový port pro ICI zprávu. Následně se kontroluje, za jakých podmínek byla ICI zpráva přijata a dle výsledku vypíše tuto informaci do konzole.

- Dále se už jen doplňuje vstupní pozice stavu **receive**. Zde dochází ke kontrole, zdali paket obsahuje požadovaná data a vyčítání paketu ze streamu. V případě kladného přijetí, vypíše tuto zprávu do konzole.
- Na závěr je opět potřeba upravit atributy modelu **Interfaces -> Model Attributes** dle obr. 5.7.

| Attribute Name | Group | Type | Units | Default Value |
|---|---|---|---|---|
| Local port | UDP | integer | | 161 |

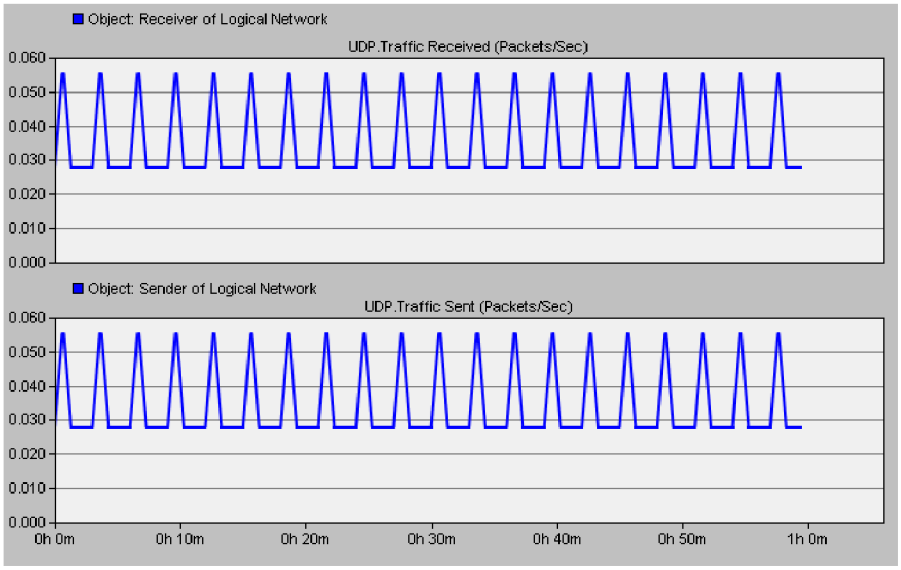
*Obr. 5.7: Atributy modelu „process_model_receiver“*

## ➢ Spuštění simulace a výsledky

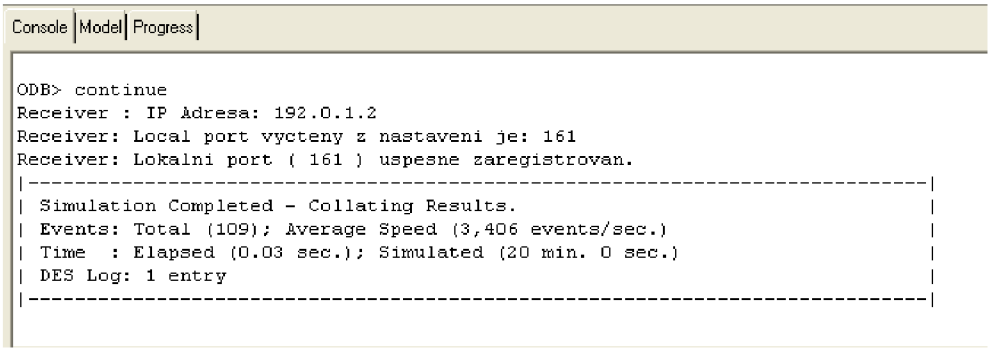
Dříve než se spustí simulace, se ještě musí nastavit statistiky, které nás zajímají. To se provede kliknutím pravého tlačítkem na pracovní plochu a volbou **Choose Individual DES Statistic**. V tomto případě se označí položka **Node Statistic -> UDP** a potvrdí se tlačítkem **OK**.

Simulace se spustí kliknutím na ikonu běžce v panelu nástrojů nebo přes menu **DES -> Configure/Run Discrete Event Simulation**. Pole **Simulation Kernel** se nastaví na **Development** a pokud je třeba spustit simulaci v debuggeru, vybere se **Execution -> OPNET Debugger** a zatrhne se pole **Use OPNET Simulation Debugger (ODB)**. Nyní se může projekt spustit stiskem tlačítka **Run**.

Po dokončení simulace se mohou zobrazit výsledky pravým kliknutím na pracovní ploše a zvolením položky **View Results**. Výsledný graf (viz obr. 5.8) dokazuje, že mezi stanicemi skutečně UDP provoz probíhá. Provoz lze zobrazit i v již zmiňovaném debuggeru (viz obr. 5.9).




*Obr. 5.8: Graf provozu mezi stanicemi*




*Obr. 5.9: Příklad výpisu informací o provozu ICI zpráv v konzole debuggeru.*

## 5.3 Procesní model směrovacího protokolu OLSR

V této části je prostudován procesní model směrovacího protokolu OLSR. Výstupem je textový popis stavů, funkcí a přechodů definovaných pro procesy s OLSR souvisejícími.

### 5.3.1 Základní nastavení

Jak již bylo uvedeno, tak je OPNET modeler rozdělen do jednotlivých úrovní a my si rozebereme nastavení a chování protokolu OLSR ve všech následujících úrovních:

- Úroveň uzlu,
- rodičovský procesní model (Parent Model),
- dceřiný procesní model (Child Model).

Nejprve je vytvořen velice jednoduchý projekt, kde jediné, co se při vytváření nového projektu nastaví, bude zaškrtnutí používání palety objektů **MANET** a ostatní nastavení se může nechat výchozí. Jakmile je scénář hotov, vloží se na pracovní plochu objekt **wlan_wkstn** (nebo **manet_station**).

### 5.3.2 Úroveň uzlu

V této úrovni se pouze se nastaví, aby stanice pracovala podle protokolu **OLSR**. Toho se docílí pravým kliknutím kurzoru na stanici a výběrem **Edit Atributes.** Zde se nastaví **AD-HOC Routing Parameters -> AD-HOC Routing Protocol -> OLSR** a potvrdí se tlačítkem **OK**.

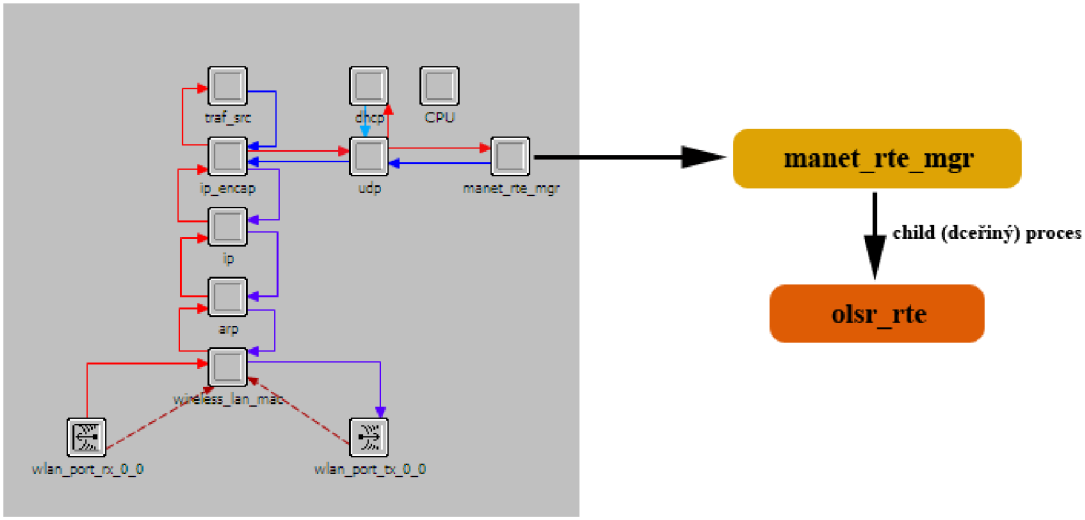
Pokud se na stanici klikne pravým tlačítkem a zvolí se **Edit Node Model** nebo se na stanici poklepe dvakrát kurzorem myši, objeví se okno editoru uzlu, kde je možné vidět jednotlivé vnitřní bloky, ze kterých se model uzlu skládá (viz například obr. 5.4). Ten nejdůležitější je **manet_rte_mgr**, který bude popsán dále.

### 5.3.3 Úroveň manažera (rodičovský procesní model)

Tento procesní model se nachází v modelu uzlu pod blokem **manet_rte_mgr**. Jedná se o tzv. kořenový (root) proces, který je přímo připojený přes UDP. OLSR je v tomto případě tzv. dceřiný proces tohoto procesu a komunikuje s UDP přes port 698. Pakety přicházející z nižších vrstev jsou předány procesu **manet_rte_mgr** přes UDP a jsou v pořadí předávány OLSR dceřinému-procesu (child procesu). Řídící pakety (HELLO a zprávy TC) jsou posílány přímo na UDP na specifický port OLSR dceřiného-procesu. Tuto hierarchii nám více přiblíží obr. 5.10.

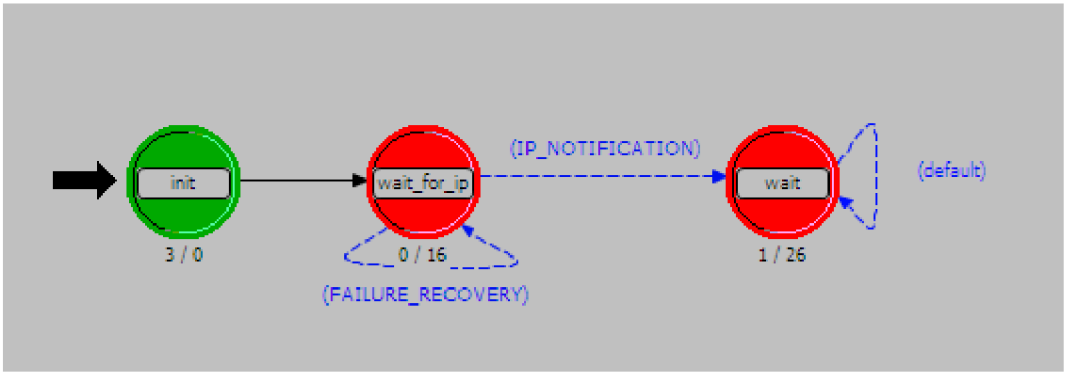
OLSR používá pro komunikaci s UDP ICI zprávu **udp_command_inet**. Tato zpráva je umístěna v adresáři *<instalační_složka>\<verze_OPNETu>\models\std\rip* a používá následující položky:

- *interface received (adresa rozhraní)*
- *local_port (místní port)*
- *rem_port (vzdálený port)*
- *rem_addr (vzdálená adresa)*




*Obr. 5.10: Schéma architektury OLSR v modelu uzlu*

Samotný procesní model manažera otevřeme dvojklikem na procesor **manet_rte_mgr**. Objeví se nám model, který se skládá za 3 stavů (jeden vynucený a dva nevynucené) a 4 přechodů (viz obr. 5.11).




*Obr. 5.11 Procesní model procesu **manet_rte_mgr**.*

**Stav *init***

Po spuštění přejde simulace nejprve do vynuceného stavu **init**, což je začátek procesu a kde se pouze inicializují stavové proměnné. Programový kód, který je v tomto stavu obsažený (**manet_rte_mgr_sv_init**), odkazuje na inicializaci ve funkčním bloku.

**Stav *wait_for_ip***

Poté přejde do nevynuceného stavu **wait_for_ip**. Jak je vidět na obr. 5.11, na vstupní pozici je nula řádků, tzn., že proces bude čekat na přerušení. Dále proběhne přechod do následujícího stavu **wait**, ale s podmínkou **IP_NOTIFICATION**. Tato podmínka určí směrovací protokol, v tomto případě OLSR. Pokud podmínka proběhne úspěšně, spustí se OLSR protokol jako dceřiný-proces a dokončí se přechod do stavu **wait**.

**Stav *wait***

V tomto stavu se čeká na přijetí zprávy z procesoru UDP, kde dojde k její identifikaci a předání dceřinému-procesu. Poté se vrátí stav opět na začátek a čeká na další přerušení.

### 5.3.4 Funkční blok rodičovského procesu – popis vybraných funkcí

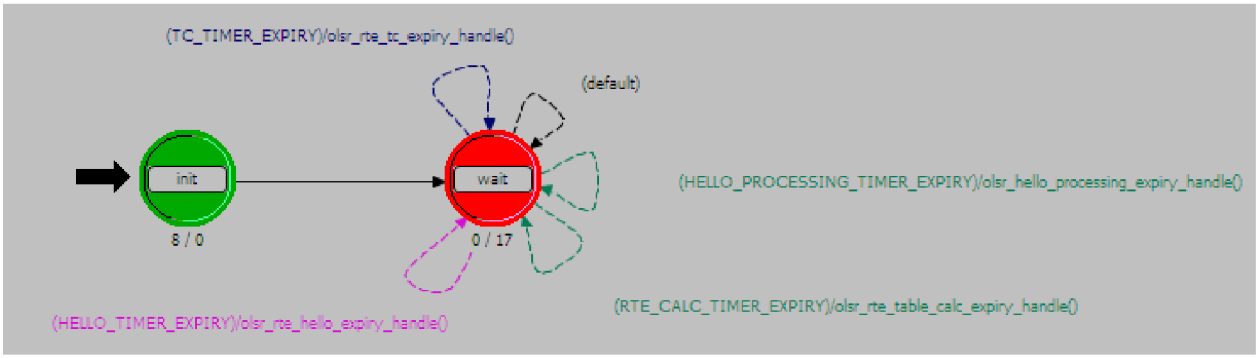
- **manet_rte_mgr_sv_init** - zde probíhá jednak deklarace proměnných, dále také nastavení jména uzlu, získání ID objektu a uzlu, získání vlastního handleru, a další. Tato funkce je volána ve stavu **init.**

- **manet_rte_mgr_routing_protocol_determine** – tato funkce je inicializována ve stavu **wait_for_ip** a jak název napovídá, provádí určení a spuštění směrovacího protokolu pro tento uzel.

- **manet_rte_packet_arrival** – tato funkce je inicializována ve stavu **wait** a má za úkol přijmout zprávu (paket), přiřadit jí ID a předat ji dceřinému-procesu OLSR. Pokud není dceřiný-proces aktivní, zpráva je zničena.

### 5.3.5 Úroveň směrovacího protokolu dceřiného procesu (child model)

Jelikož byl na začátku zvolen směrovací protokol OLSR, manažer v tomto případě spouští jako **dceřiný-proces** proces **olsr_rte**. Procesní model protokolu OLSR můžeme vidět na obr. 5.12. Skládá se ze dvou stavů – **init** a **wait**. Tyto stavy budou popsány dále.




*Obr. 5.12: Procesní model protokolu OLSR*

Tento procesní model je možno editovat dvěma způsoby. Může se vytvořit nový proces, kterému potom nastavíme atribut **process model** na **olsr_rte** a druhým způsobem je, že se otevře procesní model **manet_rte_mgr** a následně pomocí menu **File -> Open Child Process Model -> olsr_rte**.

**Stav *init***

Jedná se o vynucený stav, který opět proběhne jako první. Jsou zde inicializovány sdílené globální proměnné, stavové proměnné, paměťové pole a registrovány statistiky. Zde probíhá opět pouze inicializace a samotná deklarace výše zmíněných proměnných a funkcí je opět ve funkčním bloku.

**Stav *wait***

Jedná se o nevynucený stav, ve kterém se provádí všechny úkoly spojené se směrováním protokolem OLSR, například výpočet směrovacích tabulek. Dále je tu podmínka, která kontroluje typ přerušení a na základě toho dochází k dalším krokům. Kolem tohoto stavu je 5 podmíněných přechodů, které po splnění podmínky vykonají danou funkci a vrací se zpět do výchozího stavu.

### 5.3.6 Funkční blok dceřiného procesu – popis vybraných funkcí

- **olsr_rte_sv_init** – tato funkce je inicializována ve stavu **init**. Zde se kromě deklarace a volání několika proměnných inicializuje a zpracovává rozhraní a komunikace ICI – UDP. Dále se tu nastavuje cílová adresa a port. Nastavení cílové adresy je důležité pro funkčnost všesměrového (broadcast) vysílání v IPv4. V IPv6 se z broadcast vysílání stává multicast. Pro správnou funkci tohoto multicast vysílání

musíme nastavit pole **strm_index** v UDP ICI. Toto nastavení bude případně zapsáno i do IP vrstvy ICI.

- **olsr_rte_pkt_arrival_handle** – jak už z názvu vyplývá, tato funkce slouží ke zpracování příchozích paketů. Jakmile řídící paket dorazí na port, kde OLSR proces poslouchá, je tento proces zavolán a paket je přijat. Proces je vyvolán rodičovským procesem **manet_rte_mgr**, který oznamuje příchod paketu.

- **olsr_rte_process_hello_pk** – zde je vytvářen HELLO paket. Je mu přidělována zdrojová a cílová adresa, je prováděna kontrola duplicity adres (jinak by byl paket zahozen), dále je tu přidělována a uvolňována potřebná paměť, a další.

- **olsr_rte_process_hello** – zpracovává přijaté HELLO zprávy, aktualizuje nastavení linek, sousedů a MPR statusů.

- **olsr_rte_send_hello** – v této funkci dochází k vytváření a pravidelnému vysílání HELLO zpráv. Dále se tu alokuje potřebná paměť a definuje velikost zprávy. Ta má velikost 32 bitů a její struktura je následující:
  - *Samotná zpráva* (16 bitů)
  - *Hold time* (8 bitů) – ten se dělí na tzv. Neighbor Hold Time (časovač vypršení linky), Topology Hold Time (časovač pro záznamy v tabulce spojů) a Duplicate Message Hold Time (časovač, který se používá pro omezení duplikace zpráv)
  - *Willingness* (8 bitů) – tento atribut se dá popsat jako ochota uzlu přesměrovat provoz na další uzly.

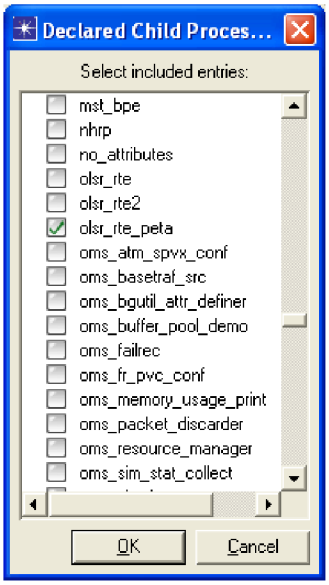
## 5.4 Rozšíření datové jednotky HELLO zprávy protokolu OLSR

V této části je popsáno rozšíření HELLO zprávy směrovacího protokolu OLSR o nové pole, kde výchozí pakety směrovacího protokolu jsou upraveny tak, že je přidána vlastní hodnota do paketu, paket je odeslán sousedním uzlů, sousední uzly identifikují modifikovaný paket a zobrazí hodnotu, která byla přidána. Výstupem je simulační model MANET sítě (včetně postupu vytváření a zdrojových kódů).

### 5.4.1 Vytvoření záložních souborů a nového pole

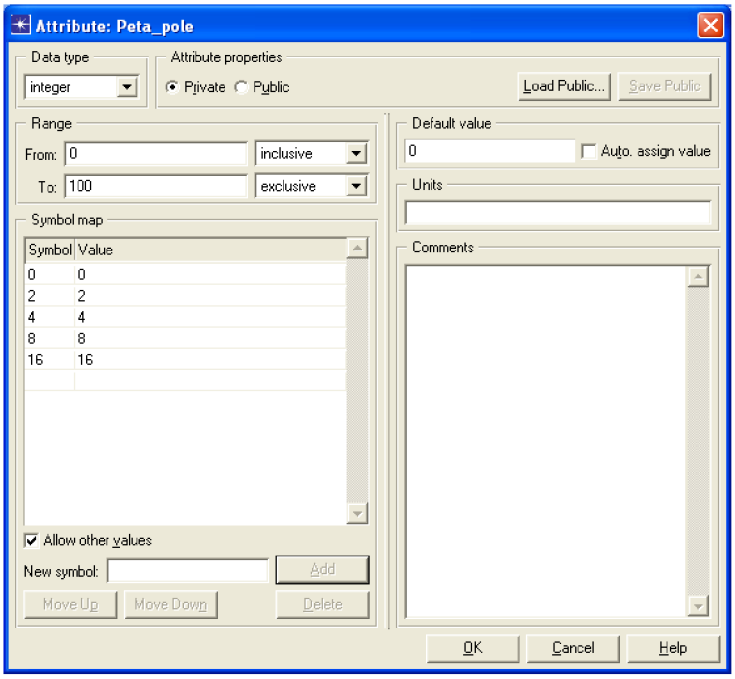
Nejprve je vhodné si vytvořit zálohy souborů OM, které se budou editovat, aby nedošlo k přepisování soubory defaultních. Nejprve se ve složce *<instalační_složka>\<verze_OPNETu>\models\std\include* vytvoří kopie souboru **olsr_pkt_support.h**, který se pojmenuje například **olsr_pkt_support_peta.h**. Dále se vytvoří záloha modelu uzlu **wlan_wkstn**, která se pojmenuje například **wlan_wkstn_adv_peta.nd.m**. Nesmí se zapomenout tento nově vytvořený model uzlu přiřadit všem stanicím, se kterými se pracuje. Postup byl již popsán dříve. Dále se vytvoří záloha procesního modelu uzlu **manet_rte_mgr**,

která se pojmenuje například **manet_rte_mgr_peta** a opět se musí k procesu **manet_rte_mgr** přiřadit. Poslední, co se bude zálohovat je (dceřiný) *child_proces*. Nejprve se v procesním modelu otevře a uloží jako například **olsr_rte_peta** a poté se přiřadí pomocí menu **File -> Declare Child Process Models…**, kde se právě nově uložený dceřiný proces vybere (viz obr. 5.13).




*Obr. 5.13: Přiřazení nově vytvořeného (dceřiného) child_procesu.*

Nyní se vytvoří nové pole v HELLO zprávě protokolu OLSR. Po otevření manažera procesu a přes menu **Interfaces -> Model Attributes** se zobrazí okno, kde se označí řádek **OLSR Parameters** a dále se klikne pro editaci na tlačítko **Edit Properties**. Nyní se zobrazilo okno, kde je možné editovat jednotlivé parametry OLSR HELLO zprávy. Do posledního volného řádku se napíše název nového pole, například **Peta_pole**. Poté se toto nové pole označím a pro další úpravy se klikne na **Edit Properties**. Zde se nastaví datový typ **integer**, defaultní hodnota a rozsah hodnot, které bude možné nastavit a pro povolení možnosti nastavit i jiné než výchozí hodnoty je potřeba zaškrtnout pole **Allow other values**, viz obr. 5.14.




*Obr. 5.14: Nastavení atributů nového pole*

Na závěr je potřeba provést kompilaci a pokud proběhne bez chyb, tak přidání nového pole lze ověřit tak, že na některou ze stanic se klikne pravým tlačítkem, zvolí se **Edit Attributes -> AD-HOC Routing Protocol -> OLSR Paramaters** a zde by mělo být vidět nově vytvořené pole **Peta_pole** a mělo by být možné mu vybrat buď z přednastavených anebo přiřadit i vlastní hodnotu.

Nyní je vytvořené nové pole v OLSR HELLO zprávě a zbývá nastavit vysílání a příjem zpráv, které budou toto pole obsahovat.

### 5.4.2 Nastavení nového pole OLSR zprávy

- Nejprve se otevře ve vhodném editoru soubor **olsr_pkt_support_peta.h**, který byl v úvodu vytvořen jako kopie souboru **olsr_pkt_support.h**. Do části označené *„/* HELLO Message*/“* se doplněním následujícího kódu vytvoří nové 32-bitové datové pole. Velikost tohoto pole byla zvolena s ohledem na velikost ostatních polí OLSR HELLO zprávy:

```
OpT_uInt32     MojeData;
```

- Nyní až v samotném dceřiném procesu **olsr_rte_peta** se v bloku stavových proměnných vloží následující řádek, kterým se deklaruje nová proměnná, do které se později budou vkládat hodnoty z nového pole.

```
int      \moje_data;
```

- Dále je třeba v hlavičkovém bloku změnit, že bude používán jiný hlavičkový soubor, než výchozí. Tzn. nahradit řádek se záznamem **olsr_pkt_support.h** následujícím řádkem:

```
#include <olsr_pkt_support_peta.h>
```

- Ve funkčním bloku se musí upravit nebo doplnit funkcí více.
  - **olsr_rte_atributes_parse** – tato funkce vyčítá nastavené atributy OLSR a následujícím řádkem se naplní nově vytvořená proměnná **moje_data** hodnotou z nového pole **Peta_pole**.

```
op_ima_obj_attr_get (olsr_parms_child_id, "Peta_pole",
&moje_data);
```

  - **olsr_rte_process_hello** - v této funkci dochází ke zpracování přijaté zprávy HELLO a k aktualizaci linek a sousedů. Nejprve se deklamuje proměnná **mojedata**

```
int mojedata;
```

    Dále se vyčte informace z HELLO zprávy a vloží se do proměnné, což zaručuje následující řádek.

```
mojedata = hello_msg_ptr->mojedata;
```

    Následující řádek slouží k zobrazení aktuální hodnoty:

```
printf ("Moje data: %i ", mojedata);
```

- Jelikož se přidáním nového pole zvětšila HELLO zpráva o dalších 32 bitů. Musí se tento údaj ještě upravit (řádky přibližně 1700 a 1790) dle níže uvedeného kódu:

```
olsr_hello_message_size += 64;
```

- Pro doplnění HELLO zprávy se ještě musí přidat informace do hlavičky zprávy a to následujícím řádkem (ve funkčním bloku přibližně řádek 1850):

```
hello_msg_ptr->mojedata = moje_data;
```

- **olsr_rte_olsr_message_print** – tato funkce, nacházející se ve funkčním bloku, je podpůrná funkce, která umožňuje vypisování hodnot v debuggeru. Nejprve se opět inicializuje proměnná *mojedata*:

```
int mojedata;
```

  Dále se musí proměnná naplnit a doplnit do výpisu (řádek přibližně 4400):

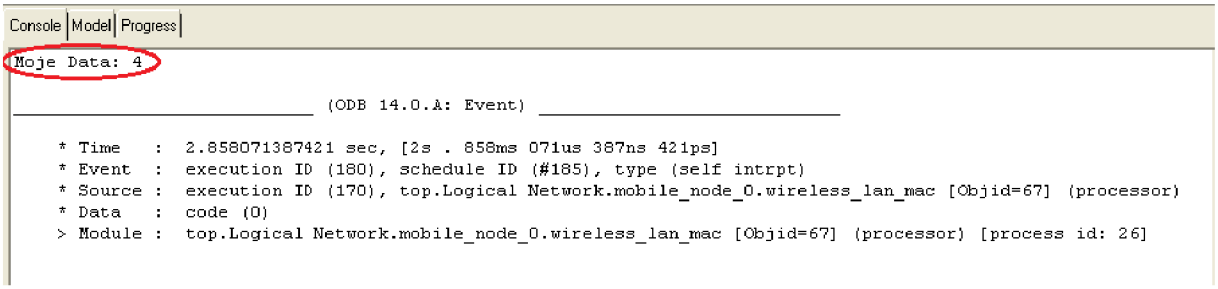
```
mojedata = hello_msg_ptr->mojedata;

  printf("Htime: %d Willingness:%d moje_data:%d", htime,
  willingness, mojedata);
```

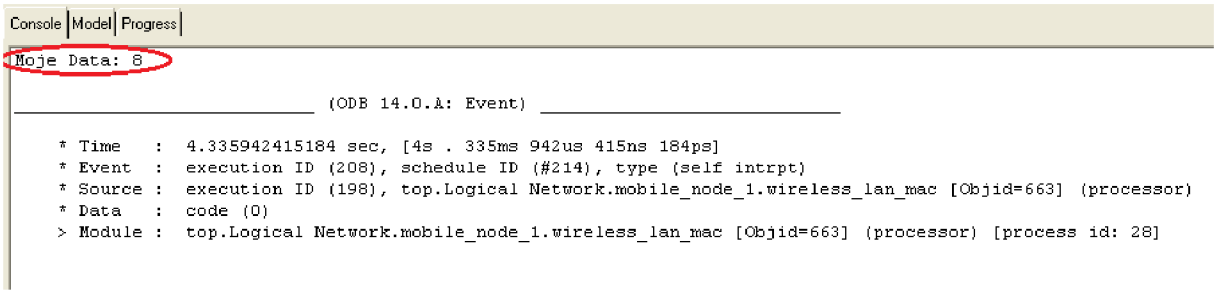
### 5.4.3 Zobrazení výsledků

Spustí-li se projekt v debuggeru, při sledování konzole je možné zaznamenat, že skutečně oba 2 uzly (vloženy byly pouze 2) si vyměňují mezi sebou hodnotu pole Peta_pole

(údaj Moje Data), která byla vložena v nastavení OLSR zprávy v daném uzlu, viz obr. 5.15 a 5.16.




*Obr. 5.15: Zobrazení údaje Moje Data uzlu „Network.mobile_node_0" v debuggeru.*




*Obr. 5.16: Zobrazení údaje Moje Data uzlu „Network.mobile_node_1" v debuggeru.*

## 5.5 Vytvoření nové zprávy protokolu OLSR obsahující údaj pro QoS

V tomto posledním bloku praktické části je popsáno vytvoření nové zprávy, která přenáší v rámci OLSR paketu údaj související se zajištěním kvality služeb a to aktuální přenosovou rychlost. Příjemci zprávy jsou všechny stanice v okolí (přesněji v dosahu vysílající stanice). Údaj o zatížení sousedních uzlů je vložen do externího souboru, kde zůstanou hodnoty uloženy i po skončení simulace. Výstupem je funkční simulační model a podrobný popis postupu jeho vytváření včetně zdrojových kódů.

### 5.5.1 Nastavení pracovní plochy

Během jakýchkoliv úprav je vhodné vždy dané soubory nebo bloky zálohovat, tzn. nově upravené uložit pod jiný názvem, aby se nepřepisovaly výchozí soubory OPNETu.

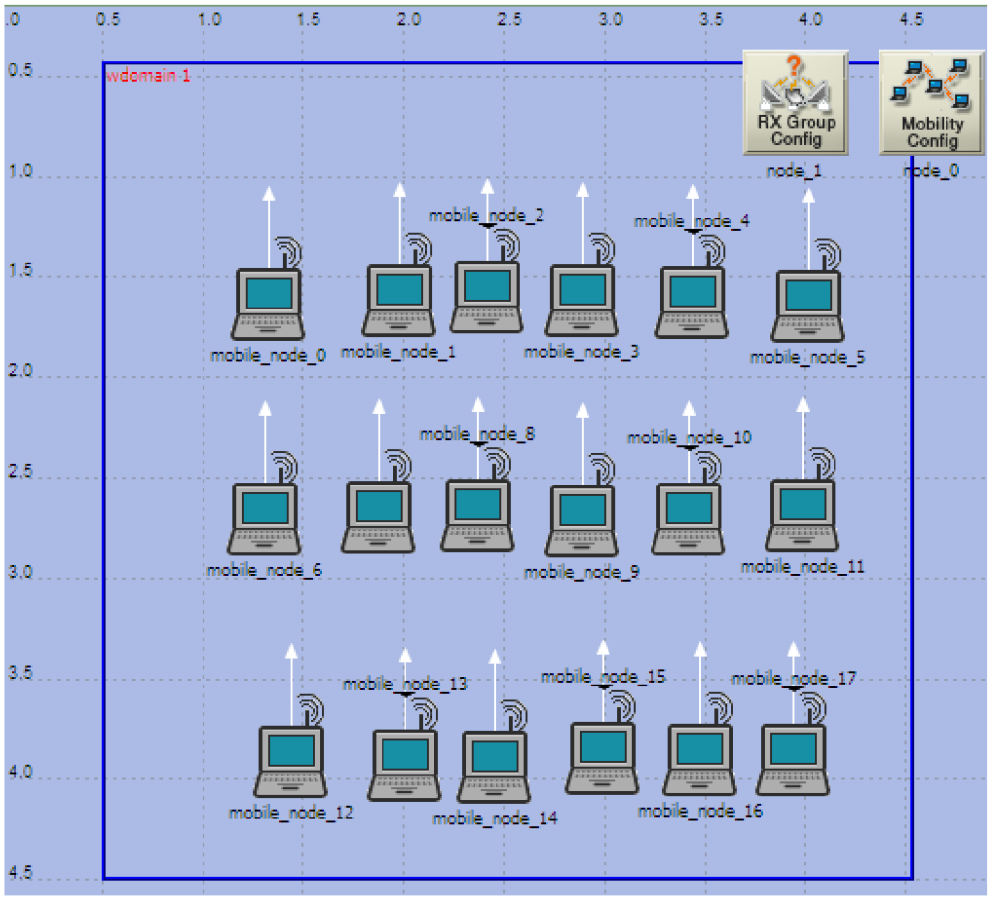
Při vytváření nového projektu se může použít například pracovní prostředí typu **Campus** s rozměry přibližně **5x5 km**. Dále se označí, že se bude pracovat s paletou typu **MANET**.

Nejprve se vloží **Mobility Domain**, což je doména (plocha), která ohraničuje pohyb bezdrátových stanic. Do ní se vloží několik stanic **manet_station** (Mobile Node). Zde se po kliknutí pravým tlačítkem a volby **Edit Attributes** zobrazí okno, kde se nastaví pracovní doména pomocí **Random Mobility Profiles -> Random Waypoint (Record Trajectory) ->**

**Random Waypoint Parameters -> Mobility Domain Name -> wdomain1**. Dále se nastaví **Pauze Time -> None**, aby se stanice pohybovaly neustále a **Start Time** se nastaví na **constant(0)**, aby pohyb začal ihned po startu simulace.

Dále se vloží prvek **Mobility Config**, který slouží k nastavení mobility uzlů a dále **RxGroup Config**, kde se nastavují nejrůznější obecné parametry pro chování stanic. Zde se nastaví **Duration -> Refresh Interval (seconds) -> 5**, což značí, jak často bude docházet k přepočítávání parametrů stanic a dále se nastaví parametr **Distance Treshold (meters)** na hodnotu **1,500**. Tato hodnota udává dosah stanic v metrech.

Jako pracovní stanice budou požity mobilní stanice **manet_station_adv**. Tato stanice se vloží na plochu například 18x. Nyní se nastaví stanicím náhodný pohyb a to tak, že pomocí kurzoru se označí všechny stanice a v menu **Topology -> Random Mobility -> Set Mobility Profile…** se vybere **Random Waypoint (Record Trajectory)**. Zvolený profil je symbolizován zobrazením bílé svislé šipky nad každou stanicí. Nyní pracovní plocha bude vypadat přibližně jako na obr. 5.17.




*Obr. 5.17: Pracovní plocha projektu*

Nyní ještě zbývá nastavit samotné stanice. Opět se označí pomocí kurzoru všechny stanice a zvolí se menu **Edit Attributes -> AD-HOC Routing Parameters -> AD-HOC Routing Protocol -> OLSR**. Ostatní parametry není potřeba měnit. Důležité je zatrhnout **Apply to selected object** a až poté změny potvrdit tlačítkem **OK**. Jen tak se toto nastavení

provede pro všechny označené uzly. Poslední nastavení je přiřazení IP adres a to pomocí menu **Protocols -> IP -> Addressing -> Auto-Assign IPv4 Addresses**. Tím se automaticky všem stanicím přiřadí IP adresy z rozsahu IPv4.

Nyní jsou stanice a pracovní plocha nastaveny a je možné se pustit do vytvoření samotné zprávy a nastavení jejího zasílání a zachytávání.

### 5.5.2 Hlavičkový soubor olsr_packet_support.h

Nejprve je zapotřebí upravit hlavičkový soubor **olsr_packet_support.h**. Zde se nejdříve definuje nová zpráva pomocí:

```
#define OLSRC_HOSEK_MESSAGE   5  /* číslo 5 značí pořadí této zprávy ze
všech zpráv OLSR */
```

Dále se vytvoří struktura nové zprávy **HOSEK_MESSAGE**. Ta obsahuje jednak proměnnou *ref_count*, která slouží jako počitadlo pro práci s vyhrazenou pamětí, dále novou proměnnou *pole* typu double (musí být použit tento datový typ, protože hodnoty, které se vrací ze statistik, jsou také typu double) a proměnná *hosek_msg_vptr*, která uchovává seznam všech vytvořených zpráv Hosek.

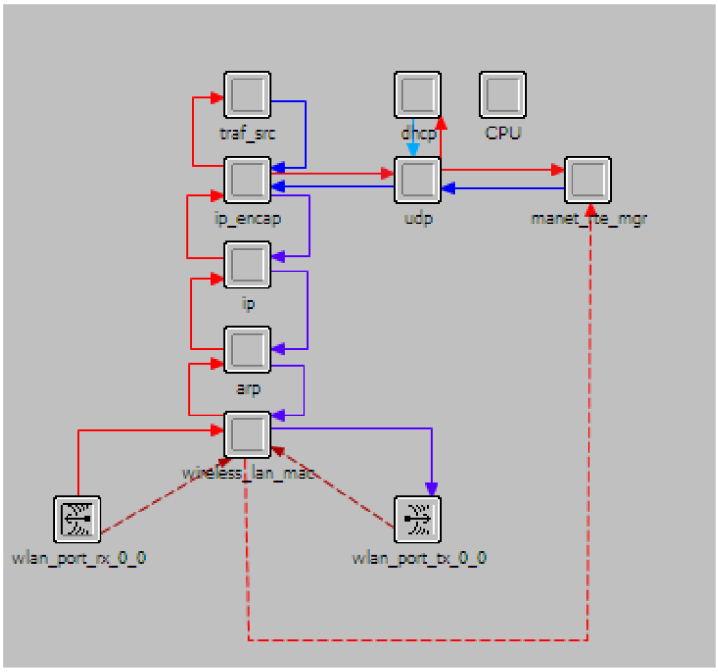
```
typedef struct
      {
#if defined (OPD_PARALLEL)
      PrgT_Mt_Spinlock  ref_count_lock;
#endif
      int   ref_count;
      double pole;
      PrgT_Vector*           hosek_msg_vptr;

      } OlsrT_Hosek_Message;
```

Hlavičkový soubor uložíme například jako *olsr_packet_support_ukol5.h*.

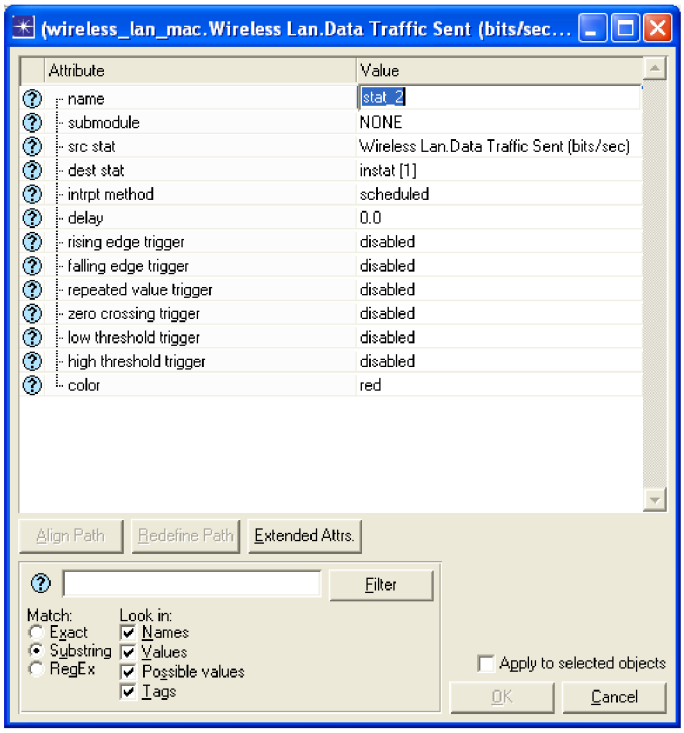
### 5.5.3 Nastavení zachytávání statistik

V modelu uzlu je zapotřebí nastavit zachytávání požadované statistiky. To se provede pomocí ikony červené šipky anebo přes menu **Objects -> Create Statistic Wire**. Linka je vedena z uzlu **wireless_lan_mac** do uzlu **manet_rte_mgr**. Uzel **wireless_lan_mac** je vybrán proto, jelikož tento uzel má na starosti přístup k fyzickému přenosovému médiu a údaje o přenosové rychlosti se vyčítají právě zde. Jakmile je linka hotova (viz obr. 5.18), je zapotřebí ještě nastavit vlastnosti statistiky.




*Obr. 5.18: Vytvořená linka pro zachytávání statistik Statistic Wire (červená přerušovaná čára)*

Na nově vytvořenou „linku statistiky“ se klikne pravým tlačítkem a zvolí se **Edit attributes**. V nově otevřeném okně se nejprve nastaví požadovaná statistika v poli **src stat**, dále se nastaví ID statistiky, se kterou se bude dále pracovat v poli **dest stat** a na závěr je potřeba vypnout všechny vyvolávače přerušení počínaje **rising edge trigger** a konče **high treshold trigger**. Výsledné nastavení je vidět na obr. 5.19.

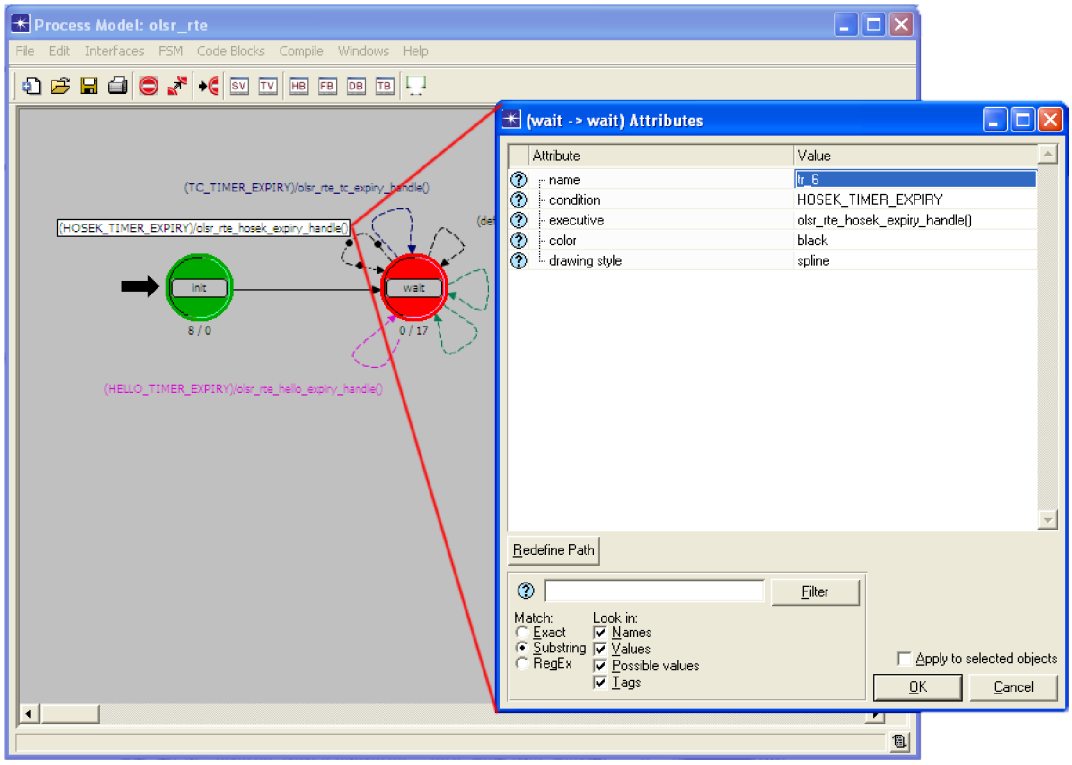



*Obr. 5.19: Nastavení linky pro zachytávání statistik*

### 5.5.4 Vytvoření časovače pro novou zprávu

Pro nově vytvořenou zprávu je zapotřebí definovat časovač, neboli při jakém přerušení se akce (odeslání zprávy) provede. Časovač pro tuto událost se nastavuje v procesním modelu **olsr_rte** pomocí menu **FSM -> Create Transition** a nebo pomocí šesté ikony zleva v panelu nástrojů. Jakmile je časovač vytvořen, pomocí menu **Edit Attributes** (po stisku pravého tlačítka) se objeví okno pro nastavení časovače. Zde se nastaví následující hodnoty (viz obr. 5.20):

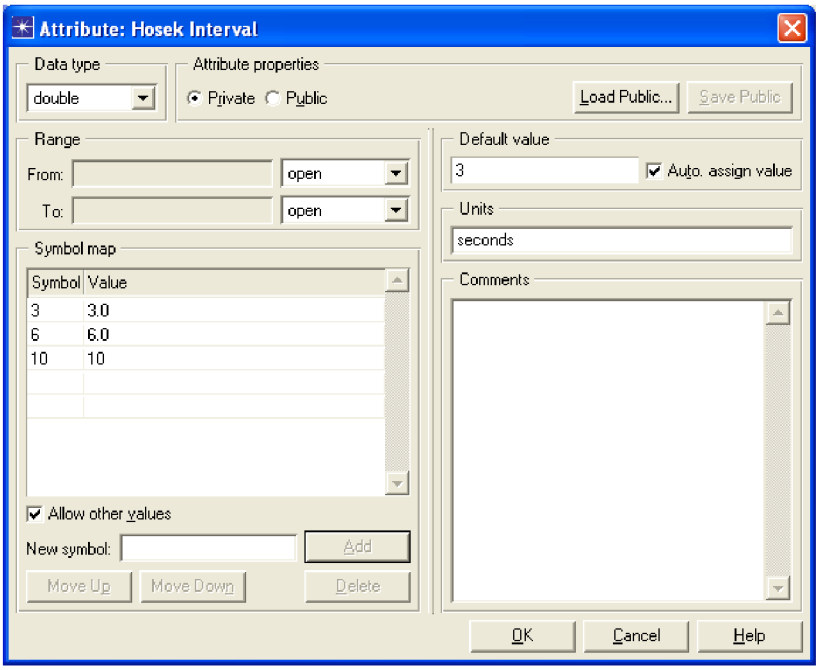
- *condition* (přerušení) – **HOSEK_TIMER_EXPIRY**
- *executive* (akce, která se provede po přerušení) – **olsr_rte_hosek_expiry_handle()**




*Obr. 5.20: Časovač a jeho nastavení pro novou zprávu*

### 5.5.5 Vytvoření nového pole pro nastavení OLSR zpráv

Proto, aby bylo možné definovat četnost vysílání nové zprávy v rámci OLSR zpráv, je potřeba vytvořit nejprve nový atribut, který bude umožňovat tuto hodnotu nastavit. V procesním modelu bloku **manet_rte_mgr** se zvolí menu **Interfaces -> Model Attributes**, kde se označí řádek **OLSR Parameters** a zvolí se **Edit Properties**. V nově otevřeném okně jsou vidět všechny parametry, které můžeme pro OLSR zprávu nastavit nebo změnit. Vloží se tedy nový atribut **Hosek Interval** a následně se zvolí **Edit Properties** a hodnoty v nově otevřeném okně se nastaví podle obr. 5.21. Nyní by tento nový atribut měl přibýt v možnostech nastavení OLSR zprávy stanic s defaultní hodnotou „3“.




*Obr. 5.21: Nastavení nového atributu OLSR zprávy Hosek Interval*

### 5.5.6 Úprava bloků stavových proměnných, funkčního a hlavičkového bloku a odstranění nulování statistik

V procesním modelu **manet_rte_mgr** se přejde do dceřiného procesu pomocí menu **File -> Open Child Process Model -> olsr_rte**. Zde se nejprve v bloku stavových proměnných deklarují nové proměnné pro získání ID a jména rodiče a také pro časovač, se kterými se bude dále pracovat ve funkčním bloku.

```
Objid \parent_id;
char  \parent_name[128];
double      \hosek_interval;
Evhandle    \hosek_timer_evhandle;
```

V hlavičkovém bloku se musí upravit 17. řádek, na kterém je definován hlavičkový soubor olsr_pkt_support.h, dle názvu tohoto nově pojmenovaného hlavičkového souboru.

```
#include <olsr_pkt_support_hosek.h>
```

Dále se deklaruje nový časovač pro vysílání zprávy a definuje se, při jakém přerušení se časovač spustí:

```
/*casovac pro vysilani zpravy*/
#define OLSRC_HOSEK_TIMER_EXPIRY    12

/* preruseni, při kterem se casovac spusti */
#define HOSEK_TIMER_EXPIRY                          \
           ((invoke_mode == OPC_PROINV_DIRECT) && (intrpt_type ==
OPC_INTRPT_SELF) && (intrpt_code == OLSRC_HOSEK_TIMER_EXPIRY))
```

Dále se musí deklarovat privátní sdílená proměnná, která slouží pro alokaci paměti objektů:

```
static Pmohandle        hosek_msg_pmh;
```

Na závěr se v tomto bloku musí deklarovat nové funkce pro odesílání a zpracování nové zprávy a pro správu časovače.

```
static void      olsr_rte_send_hosek (Boolean);
static void      olsr_rte_hosek_expiry_handle (void);
static void      olsr_rte_process_hosek (const OlsrT_Message*, int, int);
```

Nyní se ve funkčním bloku dělají následující úpravy:

- **olsr_rte_globals_init**
  V této funkci se pomocí následujícího příkazu alokuje paměť pro nově vytvořenou zprávu.

```
hosek_msg_pmh = op_prg_pmo_define ("OLSR Hosek Message",
                 sizeof (OlsrT_Hosek_Message), 32);
```

- **olsr_rte_sv_init**
  V této funkci se deklarují proměnné následně použité pro název uzlu přijímacího uzlu a vytvoření externího „*.txt“ souboru, který bude obsahovat data z nové zprávy.

```
char             nazev_uzlu[128];
FILE             *f;
```

Pro získání jména přijímacího uzlu se použijí následující dva řádky, kde se používají proměnné vytvořené v bloku stavových proměnných.

```
parent_id = op_topo_parent (own_module_objid);
op_ima_obj_attr_get_str (parent_id, "name", 128, parent_name);
```

Následující řádky slouží pro vytvoření externího souboru a vytvoření struktury dat, které v něm budou uloženy do tabulky.

```
strcpy(nazev_uzlu, parent_name);
strcat(nazev_uzlu, ".txt");

if ((f = fopen(nazev_uzlu,"w")) == NULL)
          {
printf("Soubor se nepodarilo otevrit\n");
}

fprintf(f, " |--------------|---------------------------------------|------
-----------------| \n |     Cas      |                Uzel
|  Prenosova rychlost   |\n |--------------|------------------------------
--------|----------------------| \n");

if (fclose(f) == EOF)
      {
```

```
printf("Soubor se nepodarilo zavrit\n");
}
```

Jako poslední v této funkci se přibližně ve dvou třetinách zdrojového kódu spustí časovač pomocí následujícího kódu.

```
hosek_timer_evhandle = op_intrpt_schedule_self (op_sim_time () +
op_dist_uniform (hosek_interval), OLSRC_HOSEK_TIMER_EXPIRY);
```

- **olsr_rte_attributes_parse**
  Tato funkce slouží k vyčítání nastavených parametrů (atributů) OLSR zprávy. V tomto případě se jedná o vyčtení nastaveného intervalu četnosti vysílání zpráv.

```
op_ima_obj_attr_get (olsr_parms_child_id, "Hosek Interval",
&hosek_interval);
```

- **olsr_rte_pkt_arrival_handle**
  Tato funkce slouží ke zpracování přijatého paketu. Následující funkce (podmínka) nejprve ověří, o jaký paket se jedná a poté provede vyčtení potřebných atributů a zpracování zprávy.

```
if (olsr_message_ptr->message_type == OLSRC_HOSEK_MESSAGE)

/*funkce olsr_rte_process_hosek musi vratit hodnoty ip adresu a adresu
lokalni rozhrani, */
            {
            int               local_intf_addr, ip_src_addr;
            InetT_Address     inet_local_intf_addr;
            InetT_Address     inet_ip_src_addr;

/* ziskani InetT_Address z ukazatelů */
            inet_local_intf_addr = *inet_local_intf_addr_ptr;
            inet_ip_src_addr = *inet_ip_src_addr_ptr;


/* převod InetT_Address adresy */
            local_intf_addr = inet_rtab_unique_addr_convert
(inet_local_intf_addr, &is_local_ip_duplicate);
            ip_src_addr = inet_rtab_unique_addr_convert (inet_ip_src_addr,
&is_remote_ip_duplicate);

/*zde se spusti zpracovavani hosek zpravy*/
            olsr_rte_process_hosek(olsr_message_ptr, ip_src_addr,
local_intf_addr);
            FOUT;
            }
```

- **olsr_rte_process_hosek**
  Toto je funkce, která přímo zpracovává novou zprávu, viz komentáře v kódu.

```
static void
olsr_rte_process_hosek (const OlsrT_Message* olsr_message_ptr, int
ip_src_addr, int local_intf_addr)
{
/*inicializace a deklarace proměnných*/
```

```
	double                                  pren_rychl;
	OlsrT_Hosek_Message*                    hosek_msg_ptr;
	double                                  cas_simulace;
	char                                nazev_uzlu[128];
	FILE                                *f;
	InetT_Address                           inet_tmp_addr;
	char                                tmp_str [256];
	char                    vysilac_name [OMSC_HNAME_MAX_LEN];
	int                                 originator_addr;


	FIN (olsr_rte_process_hosek (OlsrT_Message*, int, int));

/* ziskani ip adresy sender uzlu z olsr paketu */
	originator_addr = olsr_message_ptr->originator_addr;

/* ziskani hosek zpravy z olsr paketu */
	hosek_msg_ptr = (OlsrT_Hosek_Message*) olsr_message_ptr->message;

/* ziskani hodnoty pole, tzn. prenosova rychlost, z hosek zpravy */
	pren_rychl = hosek_msg_ptr->pole;

                    cas_simulace = op_sim_time();

/* ziskani a konverze ip adresy a jmena uzlu /sender/  */
	inet_tmp_addr = inet_rtab_index_to_addr_convert (originator_addr);
	inet_address_print (tmp_str, inet_tmp_addr);
	inet_address_to_hname (inet_tmp_addr, vysilac_name);
/* vypisovani do konzole */
	printf("%s: Aktualni prenosova rychlost v case %.3f uzlu %s je: %.2f
bitu/s \n",parent_name, cas_simulace, vysilac_name, pren_rychl);

/* vypisovani do souboru */
	strcpy(nazev_uzlu, parent_name);
	strcat(nazev_uzlu, ".txt");
	if ((f = fopen(nazev_uzlu,"a")) == NULL) {
		printf("Soubor se nepodarilo otevrit\n");
		}

	fprintf(f, " |    %0.3f |     %s    |     %0.2f bitu/s      | \n",
cas_simulace, vysilac_name, pren_rychl);
	fprintf(f, " |--------------|--------------------------------------
|----------------------| \n");

	if (fclose(f) == EOF) {
		printf("Soubor se nepodarilo zavrit\n");
		}

	FOUT;
	}
```

- **olsr_rte_send_hosek**
  Tato funkce se nachází v bloku, který obsahuje funkce pro odesílání paketů a zpráv, a v tomto případě se jedná přímo o odesílání nově vytvořené zprávy. Jednotlivé části jsou opět popsány ve zdrojovém kódu jako komentáře.

```
static void
olsr_rte_send_hosek (Boolean send_jittered)
	{
```

```
/* deklarace a inicializace proměnných */
OlsrT_Hosek_Message*         hosek_msg_ptr;

      int                                    originator_addr;
      OlsrT_Message*                    olsr_msg_ptr;

      double                                 akt_pren_rychlost;

      int                                    olsr_hosek_message_size;

      Packet*                                olsr_pkptr;

      FIN (olsr_rte_send_hosek (Boolean));

/* alokace pameti pro zpravu */
      hosek_msg_ptr = (OlsrT_Hosek_Message*) op_prg_pmo_alloc
(hosek_msg_pmh);

/* vycitani prenosove rychlosti ze statistiky */
      akt_pren_rychlost = op_stat_local_read (1); /*hodnota, která byla
nastavena v poli dest_adr v nastaveni linky statistiky */

/* zadani velikosti zpravy */
      olsr_hosek_message_size = 32;

/* vkladani hodnoty prenosove rychlosti do zpravy */
            hosek_msg_ptr->pole = akt_pren_rychlost;

/ *ziskani ip adresy */
            originator_addr = own_main_address;

/* vlozeni teto zpravy do OLSR Message */
            olsr_msg_ptr = olsr_pkt_support_olsr_message_create
(hosek_msg_ptr, originator_addr, olsr_hello_message_size, msg_seq_num++,
OLSRC_HOSEK_MESSAGE, neighbor_hold_time, 1, 0, is_ipv6_enabled);


/* Create olsr pkt by adding pkt_len and pkt_seq_num to this olsr_msg */
olsr_pkptr = olsr_pkt_support_pkt_create (olsr_msg_ptr, pkt_seq_num++);
      /* Send olsr pkt to udp */
      olsr_rte_pkt_send (olsr_pkptr, (is_ipv6_enabled?
InetI_Ipv6_All_Nodes_LL_Mcast_Addr: InetI_Broadcast_v4_Addr),
send_jittered);


      FOUT;
      }
```

- **olsr_rte_hosek_expiry_handle**
  Tato funkce se nachází v části pro řízení a nastavení časovačů a jednotlivé části jsou popsány jako komentáře ve zdrojovém kódu.

```
static void
olsr_rte_hosek_expiry_handle (void)
      {

/* ridi casovac a jeho vyprseni u zprav Hosek  a na zaklade toho posila
pravidelne Hosek zpravy */
```

```
      FIN (olsr_rte_hosek_expiry_handle (void));


/* vola funkci pro vytvoreni a odeslani zpravy Hosek */
      olsr_rte_send_hosek (OPC_FALSE);


/* planuje pristi kontrolu stavu pro odeslani zpravy */
      if (op_dist_uniform (2.0) > 1.0)
            {
            hosek_timer_evhandle = op_intrpt_schedule_self (op_sim_time ()
+ hosek_interval -  op_dist_uniform (0.5), OLSRC_HOSEK_TIMER_EXPIRY);
            }
      else
            {
            hosek_timer_evhandle = op_intrpt_schedule_self (op_sim_time ()
+ hosek_interval +  op_dist_uniform (0.5), OLSRC_HOSEK_TIMER_EXPIRY);
            }


      FOUT;
      }
```

- **olsr_rte_olsr_message_destroy**
  V této části dochází ke „zničení" zprávy pro uvolnění paměti. Nejprve se zavolá funkce pro dealokaci paměti.

```
OlsrT_Hosek_Message*          hosek_msg_ptr;
```

A poté se kontroluje, o jakou zprávu se jedná (řešeno pomocí *switch*) a poté se provádí akce (*case*), která splní požadavek.

```
case (OLSRC_HOSEK_MESSAGE):
                        {
                        hosek_msg_ptr = (OlsrT_Hosek_Message*)
olsr_message_ptr->message;
                        op_prg_mem_free (hosek_msg_ptr);
                              break;
```

- **olsr_rte_fail_rec_handle**
  Jako poslední funkce, kterou je potřeba ve funkčním bloku doplnit, je funkce olsr_rte_fail_rec_handle, která má na starosti funkce pro řízení akce, které se provedou v případě nějakého pádu nebo selhnání přerušení.

  Nejprve se vymažou všechny plánované přerušení a časovače.

```
op_ev_cancel_if_pending (hosek_timer_evhandle);
```

A dále se naplánuje další přerušení pro vysílání zpráv Hosek.

```
hosek_timer_evhandle = op_intrpt_schedule_self (op_sim_time () +
op_dist_uniform (hosek_interval), OLSRC_HOSEK_TIMER_EXPIRY);
```

Na závěr je potřeba otevřít v procesním modelu uzlu blok **wireless_lan_mac** a v něm zvolit dceřiný proces **wlan_mac**. Ve funkčním bloku tohoto dceřiného procesu dochází k nulování statistik vždy při přijetí dalšího přerušení. Kdyby se daný řádek neodstranil nebo nezakomentoval, tak by se ve výsledcích zobrazovaly jen samé nuly. Nulování provádí následující řádek, který se v celém funkčním bloku nachází třikrát a ve všech případech je potřeba jej tedy minimálně okomentovat nebo rovnou odstranit.

```
/* op_stat_write_t (data_traffic_sent_handle_inbits, 0.0, tx_end_time); */
```

### 5.5.7 Spuštění simulace a výsledky

V okně pro spuštění simulace (menu **DES -> Configure/Run Discrete Event Simulation**… nebo ikona běžce) se nastaví **Simulation Kernel -> Development** a v menu **Execution -> OPNET Debugger** se zatrhne **Use OPNET Simulation Debugger (ODB)**, pokud je požadováno debugger při spuštění simulace použít. Ostatní hodnoty se mohou ponechat dle výchozího natavení a projekt se spustí pomocí tlačítka **Run**.

V debuggeru je možné sledovat pro kontrolu hodnoty aktuálních přenosových rychlostí, a zároveň se tyto hodnoty přehledněji ukládají do nových souborů ve složce projektu. Příklad takového souboru je na obr. 5.22. Soubory se vytvoří vždy pro každou stanici jeden a obsahují po řádcích čas simulace, uzel, od kterého byla nová OLSR zpráva přijata, a aktuální přenosovou rychlost. Když se v debuggeru zobrazí pohyb stanic během simulace a srovná se s dosaženými výsledky, tak se potvrdí teoretické předpoklady, že čím dále jsou od sebe uzly, tím více přenosová rychlost klesá až do situace, kdy jsou od sebe uzly tak daleko, že se spojení nenaváže. V rámci tohoto projektu tato situace nastane spíše ojediněle, jelikož je použita relativně malá plocha pro pohyb stanic.

| Cas | Uzel | Prenosova rychlost |
|---|---|---|
| 0.844 | Campus Network.mobile_node_10 | 0.00 bitu/s |
| 0.891 | Campus Network.mobile_node_3 | 0.00 bitu/s |
| 1.638 | Campus Network.mobile_node_2 | 2816.00 bitu/s |
| 2.420 | Campus Network.mobile_node_5 | 2880.00 bitu/s |
| 2.610 | Campus Network.mobile_node_11 | 2848.00 bitu/s |
| 2.929 | Campus Network.mobile_node_9 | 3008.00 bitu/s |
| 4.150 | Campus Network.mobile_node_3 | 3072.00 bitu/s |
| 4.292 | Campus Network.mobile_node_10 | 2944.00 bitu/s |
| 4.995 | Campus Network.mobile_node_2 | 2784.00 bitu/s |

*Obr. 5.22: Příklad obsahu vytvořeného souboru nové OLSR zprávy.*

# 6. ZÁVĚR

Cílem této diplomové práce bylo seznámit se s problematikou sítí MANET, dále provést teoretický rozbor směrovacího protokolu OLSR. Následně se seznámit se simulačním prostředí OPNET Modeler a s možnostmi konfigurace a simulace modelu MANET sítě v něm. Hlavní část praktické práce je zaměřena na rozbor implementace protokolu OLSR a rozšíření tohoto směrovacího protokolu o novou zprávu, která by informovala okolní uzly (stanice) o míře využití dostupné přenosové kapacity.

Teoretická část této práce se nejprve zaobírá problematikou směrování v počítačových sítích. Popisuje význam, základní směrovací prvky, algoritmy a protokoly. Tato kapitola slouží jako úvod do problematiky následně probíraných témat. Další kapitola je věnována sítím MANET a obecně směrovacím protokolům, které se v těchto sítích používají. V následující kapitole jsou dva z těchto protokolů popsány. A to protokoly OSPFv3 a OLSR. U obou protokolů je proveden jejich rozbor, jsou popsány principy, funkce a jejich použití. Další kapitola se věnuje popisu simulačního prostředí OPNET Modeler, ve kterém se dají simulovat prakticky všechny typy počítačových sítí s bohatými možnostmi nastavení nejrůznějších parametrů.

Praktická část práce se zaměřuje pouze na protokol OLSR, který byl zvolen po dohodě s vedoucím práce a je rozdělena do pěti bloků. V prvních dvou blocích je rozebrána struktura procesních modelů a dále generování a příjem zpráv pomocí funkce ICI, což je datová struktura sloužící pro mezi-procesovou komunikaci. Následující blok se věnuje rozboru procesního modelu protokolu OLSR. Jsou zde rozebrány jednotlivé bloky a funkce, které jsou použity pro implementaci a chod tohoto protokolu v simulačním prostředí OPNET Modeler. Je podrobně popsán jejich význam a funkce. Předposlední část popisuje rozšíření datové jednotky HELLO zprávy protokolu OLSR, která slouží ke komunikaci se sousedními uzly a mimo jiné informuje o dostupnosti uzlu, který zprávu vysílá. Poslední část práce se zabývá vytvořením nové zprávy, která se jako součást OLSR protokolu pravidelně rozesílá všem okolním stanicím v dosahu. Podařilo se vytvořit zprávu, která přenáší údaj o maximální přenosové rychlosti, dle které jsou v danou chvíli uzly schopny mezi sebou komunikovat. Tyto hodnoty se jednak vypisují do konzole, a také se ukládají do externího souboru, kde zůstávají uloženy i po ukončení simulace. V práci je podrobně popsán celý postup vytváření této zprávy včetně funkcí, které je potřeba doplnit nebo upravit. Tento výsledek by se dal v budoucnu použít jako parametr QoS, dle kterého by se určovala nejlepší cesta pro komunikaci a směrování mezi uzly, což už ale přesahuje rámec této práce.

# 7. SEZNAM OBRÁZKŮ

# POUŽITÁ LITERATURA


1. BAKER, Fred. *Requirements for IP Version 4 Routers*. Santa Barbara, California, USA, červen, 1995. 175 s. Dostupné z WWW: <http://www.ietf.org/rfc/rfc1812.txt>.

2. BEDNÁRIK, Ján. *Modelování komunikace proprietárním protokolem, určeným pro výměnu informací o podporované technologii QoS, v prostredí OPNET Modeler*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, Brno 2007. 77 s., 15 s. příloh. Bakalářská práce.

3. COLTUN R. RFC 2740: *OSPF for IPv6*. IETF, Network Working Group, 1999. 80 s. [citováno 15. března 2011]

4. DOYLE, Jeff. *CCIE Professional Development Routing TCP/IP, Volume I*. Second Edition. Indianapolis, USA : Cisco Press, 2004. 1170 s. ISBN 1-58705-202-4.

5. ILYAS, Mohammad. *The Handbook of Ad-hoc Wireless Networks*. University Boca Raton, Florida : CRC Press, 2003. 560 s.

6. LOMNICKÝ, Marek. *Směrování a směrovací protokoly: Technologie sítí WAN (CCNA4)* [online]. Brno, 2007. 8 s. Absolventská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z WWW: <http://netacad.fit.vutbr.cz/texty/ccna-moduly/ccna2-6.pdf>.

7. Members of the INDECT Consortium. MANET Physical Layer Analysis, MANET MAC Layer analysis, MANET Routing Protocol Analysis, MANET Self Positioning algorithms analysis. *INDECT Consortium* [online]. 23. 2. 2010, WP9, [cit. 2010-12-15]. Dostupný z WWW: <http://www.indect-roject.eu/files/deliverables/public/INDECT_Deliverable_D9.8_v2010022 23a.pdf/view>.

8. MIKULICA, Vladimír. *Generování datových jednotek v prostředí OPNET Modeler*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 70 s., Diplomová práce

9. MOLNÁR, Karol. ZEMAN, O., SKOŘEPA, M., Moderní síťové technologie, Laboratorní cvičení [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2008. 101 s. Dostupné z URL: <http://www.utko.feec.vutbr.cz/~molnar/mmos/MMOS_lab.pdf>.

10. MOY J. RFC 2328: *OSPF Version 2*. IETF, Network Working Group, 1998. 54 s. [citováno 12. března 2011]

11. NOVONÝ, Vít. Komunikační prostředky mobilních sítí, Laboratorní cvičení č. 2. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 20 s.

12. NOVOTNÝ, Vojtěch. *Mobilní směrovací protokoly s podporou IPv6 (MANET)*. Brno, 2007. 89 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

13. OPNET Technologies, Inc. *OPNET Modeler Product Documentation Release 16.0.* OPNET Modeler, 2007.

14. SPORTACK, Mark A. *IP Routing Fundamentals*. Indianapolis, USA : Cisco Press, 2004. 352 s.

15. ZEMAN, Otto. *Implementace simulačního modelu zjednodušené databáze DiffServ-MIB*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. 61 s., Diplomová práce.

# SEZNAM POUŽITÝCH ZKRATEK

- AODV Ad-Hoc On Demand Distance Vector
- BGP Boarder Gateway Protocol
- DB Diagnostic Block
- DR Designated Router
- DSR Dynamic Source Routing
- EGP Exterior Gateway Protocol
- EIGRP Enhanced Interior Gateway Routing Protocol
- FB Function Block
- FSM Finite State Machines
- GSR Global State Routing
- HB Header Block
- ICI Interface Control Information
- ID Identification
- IETF Internet Engineering Task Force
- IGP Interior Gateway Routing Protocol
- IP Intenet Protocol
- IPv4 Internet Protocol version 4
- IPv6 Internet Protocol version 6
- IPX Internetwork Packet Exchange
- IS-IS Intermediate System To Intermediate System
- LAN Local Area Network
- LSA Link-State Advertisments
- MANET Mobile Ad-Hoc Network
- MPR Multi Protocol Routing
- NBMA Non-broadcast Multiple Access
- OLSR Optimized Link State Routing
- OM OPNET Modeler
- OSI Open Systems Interconnection
- OSPF Open Shortest Path First
- QoS Quality of Service
- RAM Random-access memory
- RID Routing Identification
- RIP Routing Inforamation Protocol
- SPF Shortest Path First
- SV State Variables
- TB Termination Block
- TC Topology Change
- TCP Transmission Control Protocol
- TV Temporary Variables

- UDP User Datagram Protocol
- WAN Wide Area Network