VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

OVĚŘENÍ MOŽNOSTÍ SMĚROVACÍCH PROTOKOLŮ PRO SÍTĚ MANET V PROSTŘEDÍ OPNET MODELER

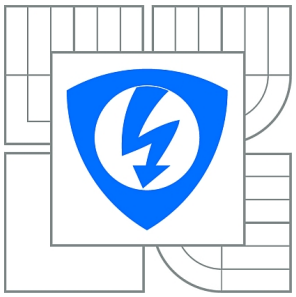
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE DANIEL KONEČNÝ
AUTHOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

OVĚŘENÍ MOŽNOSTÍ SMĚROVACÍCH PROTOKOLŮ PRO SÍTĚ MANET V PROSTŘEDÍ OPNET MODELER

VERIFICATION OF MANET ROUTING PROTOCOL POTENTIALS IN OPNET MODELER ENVIRONMENT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

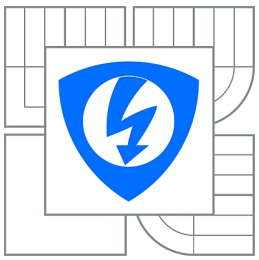
AUTOR PRÁCE — DANIEL KONEČNÝ

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ HOŠEK

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Daniel Konečný	***ID:***	115203
Ročník:	3	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Ověření možností směrovacích protokolů pro sítě MANET v prostředí OPNET Modeler

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce prostudujte problematiku MANET sítí a zaměřte se zejména na procesy směrování. Proveďte teoretický rozbor směrovacího protokolu DSR. Seznamte se se simulačním prostředím OPNET Modeler a zaměřte se zejména na možnosti tvorby modelu MANET sítě a konfigurace výše zmíněného směrovacího protokolu. V rámci praktické části vytvořte v simulačním prostředí scénář, v kterém nakonfigurujte směrovací protokol DSR. Následně proveďte rozbor procesního modelu protokolu DSR v prostředí OPNET Modeler, zaměřte se zejména na proces tvorby zpráv protokolu DSR a možnost definování nového pole do hlavičky vybraného typu zprávy. Úpravu procesního modelu ověřte pomocí simulace. Všechny dosažené výsledky uveďte přehledně v závěrečné práci.

DOPORUČENÁ LITERATURA:

[1] ILYAS, M.: The Handbook of Ad Hoc Wireless Networks. Boca Raton: CRC Press, 2003, ISBN: 0-8493-1332-5.
[2] SARKAR, S.: Ad Hoc Mobile Wireless Networks: Principles, Protocols and Applications. Boca Raton: Auerbach Publications, 2008, ISBN: 978-1-4200-6221-2.

Termín zadání:	7.2.2011	***Termín odevzdání:***	2.6.2011

Vedoucí práce: Ing. Jiří Hošek

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

UPOZORNĚNÍ:

ABSTRAKT

Tato bakalářká práce s názvem „Ověření možností směrovacích protokolů pro sítě MANET v prostředí OPNET Modeler“ je zaměřena na sítě MANET. První část představuje nejčastěji použité protokoly v prostředí OM. Druhá část je striktně zaměřena na směrovací protokol DSR, jeho parametry a konfiguraci v OM. Je zde popsán také procesní model DSR. Nejdůležitější část této práce je rozšíření datové jednotky protokolu DSR a využití této jednotky pro přenos informací, které jsou uloženy uvnitř jeho struktury. Pro rozšíření byl zvolen paket Route Request, který obsahuje novou proměnnou. Simulace dokazují úspěšný přenos rozšířeného paketu mezi stanicemi se schopností ukládat/číst ze struktury RREQ.

KLÍČOVÁ SLOVA

MANET, OPNET Modeler, DSR, Dynamic Source Routing, RREQ rozšíření, Směrovací protokol, Ad hoc.

ABSTRACT

This bachelor's thesis called „Verification of MANET routing protocol potentials in OPNET modeler enviroment“ is focused on MANET network. The first part is introducing routing protocols that are most used in OM. The second part of that work is strictly aimed to DSR routing protocol, its parametrs and configuration in OM. There is also discribed DSR process model. The most important part of that thesis is extension of data unit of DSR protocol and use that unit for transmision of information that are stored inside of its structure. As the extended unit was chosen Route Request packet that include new variable. The simulations are proving succesfull traffic of extension packet between nods with ability to store/read from RREQ extended structure.

KEYWORDS

MANET, OPNET Modeler, DSR, Dynamic Source Routing, RREQ Extension, Routing protocol, Ad Hoc.

KONEČNÝ, Daniel *Ověření možností směrovacích protokolů pro sítě MANET v prostředí OPNET Modeler*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 60 s. Vedoucí práce byl Ing. Jiří Hošek

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Ověření možností směrovacích protokolů pro sítě MANET v prostředí OPNET Modeler“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské 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.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Jiřímu Hoškovi, za vstřícný přístup a cenné rady při zpracování práce. Poděkování také patří Doc. Ing. Karolovi Molnárovi, Ph.D., a jeho týmu. V neposlední řadě chci poděkovat firmě OPNET Technologies Inc. za možnost pracovat s jejich produkty.

V Brně dne ....

(Podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

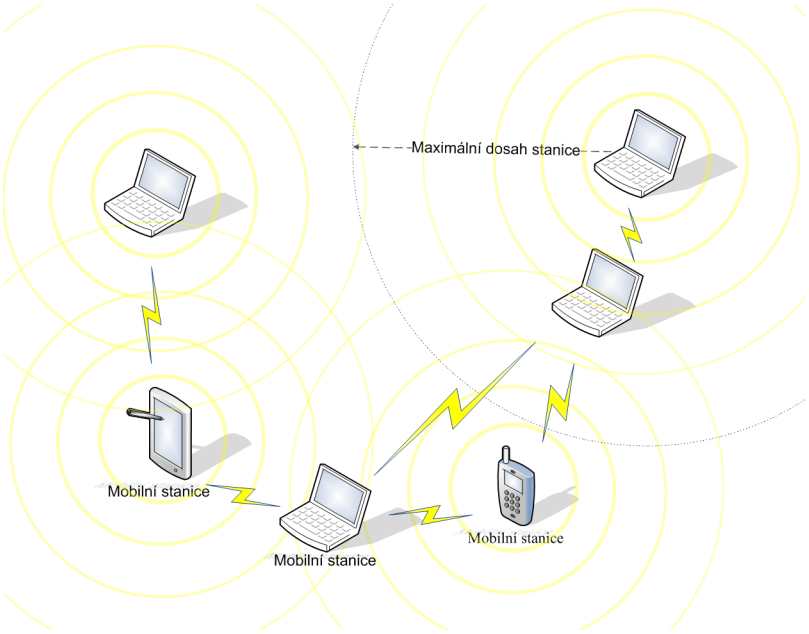
ÚVOD

Tento dokument se zabývá dynamickým směrováním v oblasti radiové komunikace sítí s rovnocennými prvky. Jeho nezbytnou součástí je snaha o vysvětlení problematiky a její praktické předvedení.

Cílem této bakalářské práce je prostudovat problematiku MANET (Mobile Ad Hoc Netwtork) sítí a zaměřit se zejména na procesy směrování. Proto má být proveden teoretický rozbor směrovacího protokolu DSR (Dynamic Source Routing). Dále je třeba se v této práci seznámit se simulačním prostředím OPNET Modeler a zaměřit se zejména na možnosti tvorby modelu MANET sítě a implementace výše zmíněného protokolu. V rámci praktické části bude v simulačním prostředí vytvořen scénář, ve kterém bude nakonfigurován směrovací protokol DSR. Následně se provede rozbor procesního modelu protokolu DSR v prostředí OPNET Modeler se zaměřením především na procesy tvorby zpráv protokolu a možnost definování nového pole do hlavičky vybraného typu zprávy. Upravený model bude ověřen pomocí simulace. Dosažené výsledky budou přehledně zhodnoceny.

1 MANET

V dnešní době jsou MANET sítě velmi rychle rostoucí doménou. Zkratka MANET nám sděluje, že se jedná o komunikaci rovnocenných prvků sítě. Přičemž jednotlivé prvky, obecně mobilní stanice, se mohou náhodně pohybovat v síti a zároveň zprostředkovávat přenos informací, přesměrovávat nebo samozřejmě tyto informace přijímat. Každá mobilní stanice může být ve skutečnosti jak klient, tedy přijímač, tak také server respektive odesílatel poskytující data napříč sítí. Strukturu takovéto sítě lze vidět na obr. 1.1.



Obr. 1.1: Mobilní stanice v síti Ad Hoc.

Sítě MANET jsou využívány díky své schopnosti velmi rychle a dynamicky měnit strukturu. Pro funkci sítě, mobilní stanice nepotřebují být v dosahu všech uzlů. Pokud stanice bude mít radiový dosah alespoň na jeden prvek sítě, může se stát plnohodnotným členem. Přesto je za všech situací kladen důraz, aby byla zajištěna plná konektivita v celém rozsahu sítě. Topologie takovéto sítě je mnohdy náhodná a to přináší do systému různé problémy, které je třeba řešit.

MANET sítě potřebují efektivní algoritmy k zabezpečení správy sítě, spojovacích, řídících, energetických, distribučních a směrovacích úkolů. V mobilní síti vždy neplatí, že nejkratší cesta mezi zdrojem a příjemcem dat je ta nejlepší [7].
Požadavky na algoritmy pro MANET jsou často odlišné podle parametrů sítě. Proto existuje sada protokolů, kde každý protokol má své specifické řešení komunikace.

1.1 Základní typy směrovacích protokolů pro Ad-Hoc sítě

O sítě Ad-Hoc se stará mnoho rozdílných směrovacích protokolů. V této práci budou zmíněny jejich dva základní typy a celkem pět protokolů samotných. Tato selekce je provedena především kvůli podpoře sítí MANET v simulačním prostředí OPNET Modeler, jehož funkce bude rozebírána později v kapitole 3.1.

1.1.1 Proaktivní (Proactive)

Proaktivní protokoly si periodicky vyměňují informace o svých směrovacích cestách mezi uzly nezávisle na tom, jestli zrovna přenáší pakety nebo ne. Každý uzel uchovává nezbytné informace ke směrování a každý uzel je zodpovědný při změně topologie za aktualizaci sítě.

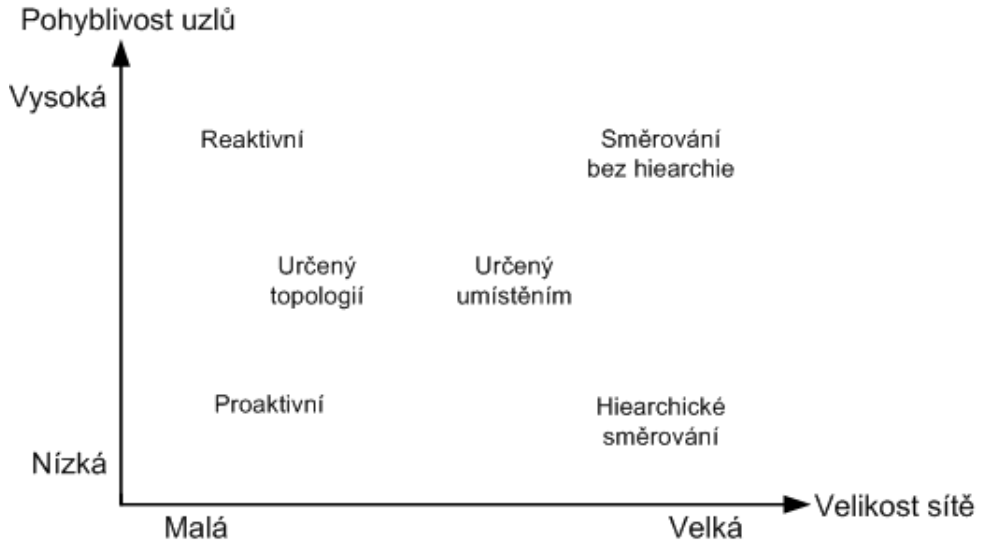
- **OLSR** (Optimized Link State Routing Protocol) – K administraci sítě používá tento protokol HELLO a TC (Topology Control) pakety. Jedná se o link-state směrovací protokol, který realizuje vysílání paketů přes předem vybrané uzly. Je vhodný pro velké mobilní sítě, díky optimalizaci, kterou řeší MPR (Multi Point Relay). [2]
- **GRP** (Geographic Routing Protocol) – Všechny uzly jsou určeny GPS (Global Positioning System) souřadnicemi. Komunikace samotná prochází přes kvadranty ve třech různých vrstvách.
- **DSDV** (Destination Sequenced Distance Vector) – Vynalezen roku 1994. Je jedním z prvních protokolů pro Ad-Hoc sítě. Tento protokol je velice efektivní v sítích s malým počtem uzlů. Jeho směrovací algoritmus je velmi dobře známý. Protože není nijak standardizován, není komerčně využíván. V průběhu let bylo vytvořeno mnoho vylepšení založených na směrovacím protokolu DSDV, např.: AODV (Ad Hoc On-demand Distance Vector).

1.1.2 Reaktivní (On-demand)

Reaktivní protokoly naopak sestavují trasu mezi uzly pouze v případě samotného požadavku na přenos paketů. Jako výsledek procesu objevení správné směrovací cesty je nezbytně nutné sestavit komunikaci mezi dvěma uzly a udržet spojení po dobu přenosu.

- **DSR** (Dynamic Source Routing)[1] – Je protokolem na který, se tento dokument přímo zaměřuje a podrobně je popsán v následující kapitole číslo 2.
- **TORA** (Temporally Ordered Routing Algorithm) – Tento protokol patří do rodiny „link reversal“ algoritmů. TORA je vytvořen tak, aby byl schopný reagovat na změny struktury sítě. Poskytuje více cest ke spojení zdroje s cílem. Sestavuje a udržuje si DAG (Direct Acyclic Graph) graf. Funkce protokolu je rozdělena do tří fází:
  - Vytvoření cesty,
  - Údržba,
  - Smazání cesty. [2]
- **AODV** (Ad Hoc On-demand Distance Vector) – Je kombinací protokolů DSR a DSDV. Tento protokol je navržen pro sítě s velkým počtem uzlů. Podporuje vícenásobné brány [7]. Směrování Hop-by-Hop a sekvenční číslo přebírá od DSDV. Proces údržby trasy od DSR.

Obrázek 1.2 zobrazuje obecnou představu, jaký typ protokolu Ad-Hoc by bylo třeba volit podle parametrů sítě, která je navrhována.



Obr. 1.2: Obecná vhodnost různých směrovacích Ad-Hoc protokolů. [4]

[1] RFC (Request for Comments) 4728 [3].

2 PROTOKOL DSR

Je reaktivním protokolem, který směruje na žádost (On-demand). Tento přístup je charakteristický tím, že nepoužívá tradiční směrovací tabulky v uzlech ani nežádá periodicky o jejich aktualizaci při změně topologie sítě. On-demand směrující protokoly, nejprve vyhledají směrovací cestu a teprve poté zahájí přenos dat. Směrovací aktivita je nulová, pokud přenos dat není generován uzly. Mechanika hledání takovýchto tras je podrobně popsána v kapitole 2.1. Dále je třeba zmínit, že směrovací protokol DSR používá, k úspěšnému přenosu dat, dvě techniky:

- Směrování od zdroje (Source Routing).
- Komunikaci z uzlu na uzel (Hop-by-Hop).

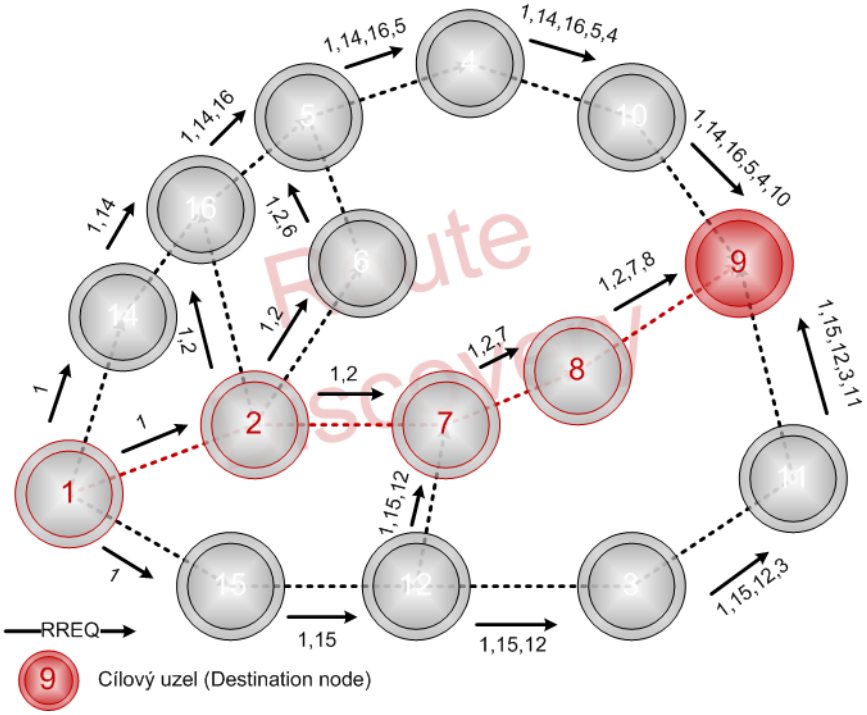
2.1 Základní mechanismus DSR

Protokol DSR je vytvořen na základních dvou mechanikách. Nalezení směrovací cesty (Route Discovery) a Údržba trasy (Route Maintance).

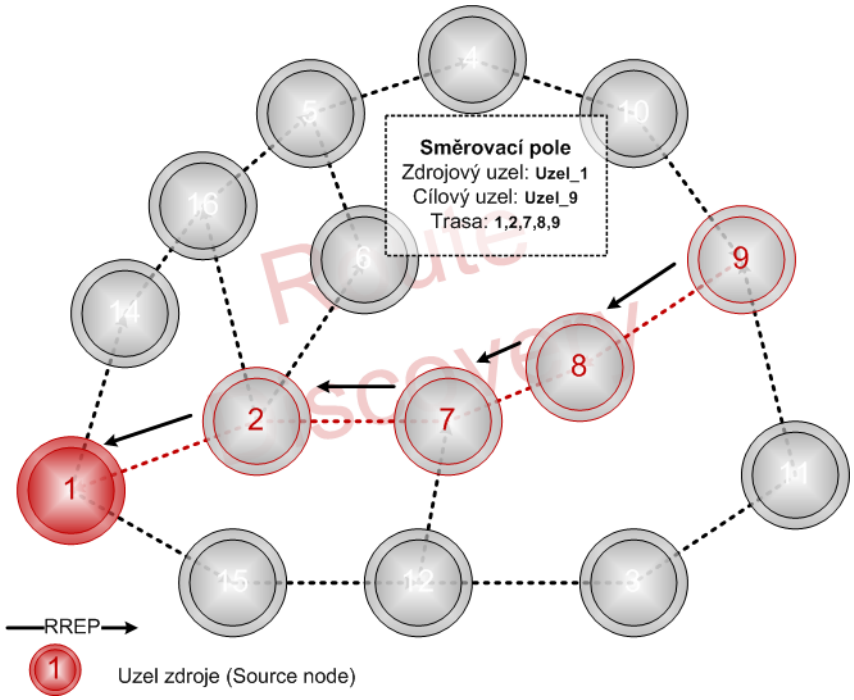
2.1.1 Hledání směrovací trasy (Route Discovery)

Route discovery využívá RREQ (Route Request) a RREP (Route Reply) kontrolní pakety. Slouží k vyhledávání trasy a potvrzení cesty v mezilehlém uzlu. Nalezení směrovací cesty je vybuzeno v případě, když zdrojový **S** uzel (Source) se pokusí poslat paket na místo určení, tedy **D** uzlu (Destination) a zároveň nemá tuto cestu již uloženou v paměti (Cache). Nalezení trasy je založeno na záplavě sítě (Flooding) RREQ pakety. Tyto pakety obsahují směrovací pole (Route Request Table) uvnitř kterého se nachází: adresa zdroje, adresa cílové destinace, unikátní identifikační číslo a záznam trasy.
Tohoto poznatku bude později využito v praktické části, především při pokusech o rozšíření datové jednotky RREQ. Visuální popis a hledání směrovací cesty je znázorněno na Obr. 2.1 a 2.2. Mezilehlý uzel se muže po příchodu RREQ kontrolního paketu zachovat podle tří možností:

- Pokud je cesta k **D** uzlu již uložená v paměti, odpoví mezilehlý uzel, uzlu **S** pomocí RREP (V tomto případě se jedná o spojení záznamu cesty (Concatenation of Route Record) od uzlu **S** s cestou, kterou má v paměti mezilehlý uzel, tedy až po **D** uzel),
- Pokud tento paket uzel už obdržel, tak jej zahodí,
- Může přidat jeho vlastní ID do záznamu směrování (Route Record) a všem jeho sousedním uzlům rozeslat pakety broadcastem.



Obr. 2.1: Princip hledání trasy protokolu DSR, pomocí RREQ paketu.



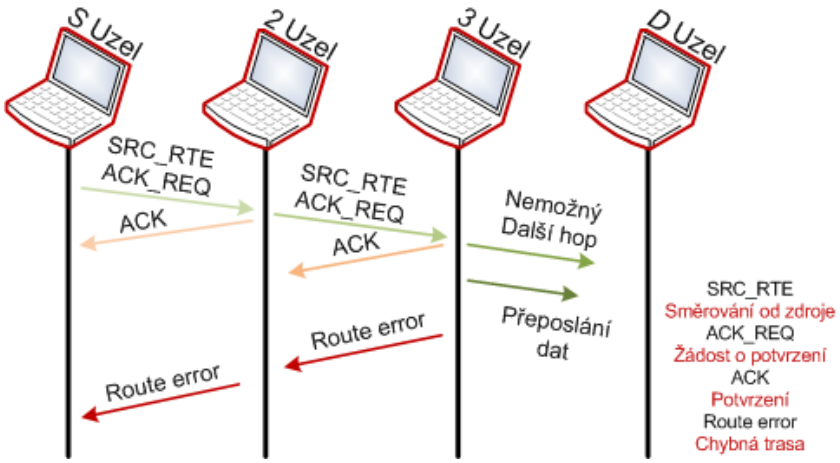
Obr. 2.2: RREP paket mířící do zdroje jako potvrzení nalezené trasy.

Když kontrolní paket RREQ dorazí na místo určení, **D** uzel vyšle odpověď ve formě dalšího RREQ paketu. Tentokrát ale trasou zpět podle cest, které má uložené v paměti. Spojení je tedy obousměrné. Dále je tu rozšíření protokolu DSR, nazvané RODA, které podporuje nesymetrické linky. V tomto případě dokáže cílový uzel vyvolat další hledání směrovací cesty a tak může rozšířit naakumulované cesty uvnitř RREQ paketu.

2.1.2 Údržba trasy (Route Maintance)

Základní mechanismus Route maintance je následující. Když mezilehlý uzel zjistí, že spojení na další stanici (Next-Hop) směrem k **D** uzlu je nefunkční, tak tuto trasu vyjme ze směrovací paměti a vrátí **S** uzlu takzvaný Route Error – kontrolní správu, která obsahuje informaci o chybné trase. Zdrojový uzel **S** potom vyšle novou žádost o nalezení cesty. Tyto principy přehledně zobrazuje obr. 2.3. Do tohoto základního mechanismu bylo navrženo několik vylepšení:

- **Promiskuitní mód** (Promiscious Mode Operation) – Uzel zapne jeho radiové komunikační rozhraní v promiskuitním módu, aby mohl zachytit kontrolní pakety, které nemusejí být adresovány přímo jemu. Tímto může získat užitečné informace, které mu dopomohou k lepší správě paměti (Cache Management),
- **Záchrana dat** (Salvaging) – Když spojení na dalším přeskoku není možné. Na rozdíl od posílání Error Route zpráv zpátky na **S** uzel může mezilehlý uzel použít cestu k cíli použitím jeho vlastní paměti,
- **Náhodné zpoždění** (Random delays) – Tento mechanismus se používá, aby se předešlo kolizi při posílání RREP paketů.[2]



Obr. 2.3: Ukázka režijní komunikace v případě chybné trasy.

Pro dobrou funkci protokolu DSR je Cache Management naprosto kritický parametr. Zvláště pod velkou zátěží sítě. Pokud staré trasy nejsou okamžitě vymazány z paměti, tak mohou být použity pro další směrování a rychle zaznamenány do paměti jiných uzlů. Pro tyto situace obsahuje protokol DSR několik nastavení (Tyto nastavení jsou v prostředí OPNET Modeler pevně dány uvnitř zdrojového kódu protokolu DSR. V této práci nebudou měněny, přesto je vhodné je pro kompletní funkci směrovacího protokolu DSR uvést.) např.:

- **Pevná doba životnosti** (Fixed Lifetime) – Každý zápis do směrovací tabulky má konstantní dobu vypršení. Ovšem je třeba si při volbě tohoto parametru dát obzvláště záležet, protože nevhodně zvolená konstanta může mít za následek ještě horší vlastnosti protokolu, než kdyby se správa paměti nepoužívala vůbec.
- **Přizpůsobivá doba životnosti** (Adaptive Lifetime) – Ustanovuje hodnotu konstanty pro dobu životnosti podle významných, sledovaných událostí (počet chybových ohlášení, přerušení spojení, atd.).
- **Dočasný zákaz zápisu** (Negative chache) – pokud uzel často ukládá vadné trasy nebo narazí na větší nárůst Route Error zpráv, potom přidělí uzel této trase interval, po který nebude do paměti z tohoto směru zapisovat žádný nový záznam.
- **Šíření zpráv o chybách** (Wider Error Notification) – Používá se v případě nalezení vadného spojení. Tento mechanismus je schopný efektivně vymazat záznamy cest ze sousedních a zdrojových uzlů. Šíření těchto oznámení je pomocí broadcastu od místa chyby komunikace.

Výsledky simulace ukazují, že zapnutí Adaptive Lifetime dosahuje stejných výsledků jako velmi dobře zvolená časová konstanta pro Fixed Lifetime. Nicméně, výkonové výsledky ukazují, že kombinace Adaptive Lifetime a aktivní mazání pomocí Wider Error Notification je lepší, než jejich použití samostatně. [2]

2.2 Energetická náročnost DSR

Energetická náročnost Ad-Hoc sítí není přímo závislá na používané šířce pásma. Stanice těchto sítí se obvykle pohybují ve třech stavech. Nečinné stanice (Idle), Stanice aktivní (příjem/přenos dat) nebo v Režimu spánku (Sleep mode). Protože stanice i ve stavu „nečinnosti“ mezi sebou komunikují kvůli údržbě sítě, je tento stav samotný dost energeticky náročný. Přesto energetická spotřeba narůstá výrazně i v případě příjmu dat. Nejhorší případ nastane, když stanice začne všesměrově vysílat tzv. Broadcastovat ve velice husté síti.

Protokol DSR je z pohledu energetické náročnosti mezi ostatními protokoly Ad-Hoc sítí poměrně velmi efektivní. Mohou za to mechanismy protokolu samotného. Velkou energetickou ztrátou pro DSR je promiskuitní mód stanic.[1]

3 SMĚROVACÍ PROTOKOL DSR V PROSTŘEDÍ OPNET MODELER

Tato kapitola nejprve představí program OPNET Modeler, poté procesní model protokolu DSR a nakonec nastavitelné parametry tohoto směrovacího protokolu v OM (OPNET Modeler).

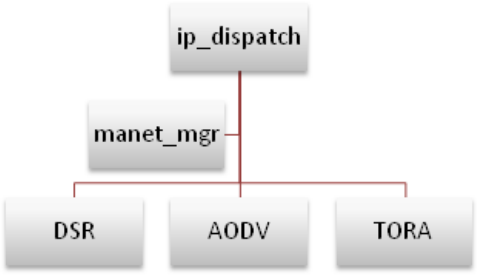
3.1 Simulační prostředí OPNET Modeler

Program OPNET Modeler je simulační prostředí, které bylo vyvinuto 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. Základním kamenem OM je jeho grafické prostředí, díky kterému je práce v něm efektivnější a rychlejší. Velmi důležitou vlastností OM je široká možnost tvorby různých statistik z dané simulace. Tato vlastnost nabádá k použití OM všude tam, kde je třeba ověřit chování reálného objektu v různých extrémních podmínkách (např. chování serveru při vysoké zátěži apod.). S tím také souvisí, že někdy nemůžeme na reálném objektu ověřit chování, které ani nemusí nastat, ale díky OM si jej můžeme nasimulovat, abychom znali výsledek chování reálného objektu v určité situaci a mohli takto předcházet nežádoucím stavům.

Výsledné statistiky můžeme generovat do zprávy ve formátech XML (Extensible Markup Language) nebo HTML (Hyper Text Markup Language), nebo uložit data do tabulek. Opačně aplikace umí z těchto formátů načíst vstupní data. Dále OM obsahuje prohlížeč animací, díky kterému můžeme názorně vidět průběh simulace. Simulace probíhá s určitým zrychlením, takže je možné třeba nasimulovat např. měsíční chování sítě v řádu hodin. Značná výhoda OM tkví v jeho objemných knihovnách, které mají dostupný zdrojový kód, z čehož plyne, že kód můžeme dále upravovat. Další vlastností OM je mutliplatformost (Windows, Linux). [6]

3.2 Procesní model protokolu DSR v OM

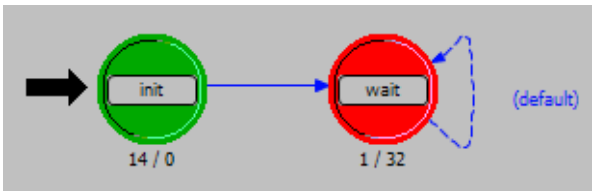
Struktura modelu uzlů MANET v OM má implementován směrovací protokol DSR na vrstvě IP (Internet Protocol). Vrstva IP obsahuje dceřiný proces `ip_dispatch` a ten se dále dělí na dceřiný proces `manet_mgr`. `Manet_mgr` pracuje jako hlavní řídící proces, který vyvolává požadovaný Ad Hoc směrovací protokol (DSR, GRP, AODV, TORA). Na obr. 3.1 lze vidět, jak se jednotlivé procesy dělí na další dceřiné procesy až k samotným protokolům MANET.



Obr. 3.1: Struktura implementace Ad Hoc protokolů v programu OM [8].

3.2.1 Proces manet_mgr

Obr. 3.2 zobrazuje procesní model `manet_mgr`, který obsahuje dva prvky. Počáteční nevynucený stav **init** a vynucený stav **wait**.



Obr. 3.2: Procesní model manet_mgr.

Nevynucený stav – Vstup: init

Funkce `manet_mgr_sv_init();` zajišťuje inicializaci stavových proměnných. Proměnné obsahují například vlastní identifikační číslo modelu, ID uzlu. Probíhá zde registrace procesu samotného a atributů protokolu. Dále jsou proměnné naplněny vlastnostmi příbuzných procesů.

Funkce `manet_mgr_routing_protocol_determine();` určí, který daný směrovací protokol MANET běží na konkrétním uzlu. Všechny rozhraní, které mají MANET protokoly povoleny, by měly mít nastaveno stejné směrování. Ve aktuální verzi OM 16. není možné mít rozdílné MANET protokoly na rozdílných rozhraních.

Funkce `manet_mgr_routing_process_create();` vytvoří a zároveň vyvolá nastavený směrovací proces MANET. [8]

Vynucený stav – Výstup: wait

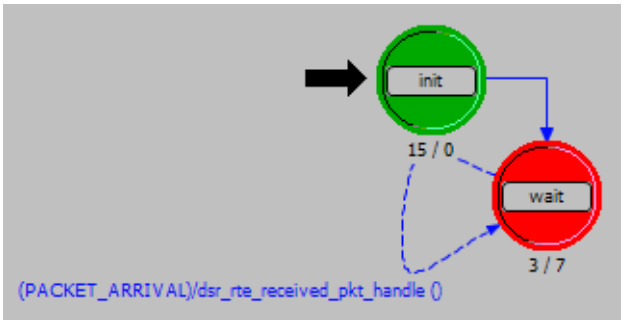
Funkce `invoker_phandle = op_pro_invoker (own_prohandle, &inv_mode);` a `invoker_id = op_pro_id (invoker_phandle);` čekají na příchozí výzvu, poté se z dalšího procesu ověřuje, jestli byl proces volán z dceřiného procesu nebo z jednoho z CPU (Central Processing Unit).

Dále se zaměříme přímo na procesní model protokolu DSR s názvem `dsr_rte`, který obsahuje většinu nastavitelných funkcí tohoto směrovacího protokolu. Funkce protokolu DSR dále pracují s externími soubory, které jsou uvnitř samotného OM. Jejich výpis pro každý procesní model protokolu zobrazíme pomocí **File** ↦ **Generate Text Report**. Tyto soubory se obvykle nachází ve složce *C:\Program Files\OPNET\version\models\std\manet (ip)*.

Soubor funkcí modelu `dsr_rte` je dále zapsán v hlavičkových souborech, které obsahují časové konstanty a struktury režijních funkcí. Velikost polí paketů DSR. Struktury, ve kterých jsou zapsány základní nastavení komunikace protokolu DSR jako žádost, potvrzení nebo chyba směrování. Dále také funkce pro tvoření paketu nebo funkce pro hledání, ukládání nebo šíření trasy. Tyto soubory se obvykle nachází ve složce *C:\Program Files\OPNET\version\models\std\include* [8]

3.2.2 Proces dsr_rte

Procesní model protokolu DSR vidíme na obr. 3.3. Obsahuje celkem dva prvky, jenž volají především funkce uvnitř FB (Function Block). Prvním je stav **init**, který zajišťuje především inicializaci proměnných a oznámení nebo registraci statistik či nastavení paměti. Následně ve druhém stavu **wait** se čeká na příchod paketu z vyššího procesu.



Obr. 3.3: Procesní model dsr_rte.

Nevynucený stav – Vstup: init

Funkce `dsr_rte_sv_init();` zajišťuje inicializaci proměnných. Mezi výčet těchto proměnných si můžeme například představit proměnné zajišťující registraci procesů nebo identifikační číslo (ID) uzlů, vlastních procesů nebo procesů, které je možno zdědit z jiných funkcí. Dále také název procesu a procesního modelu. Inicializace řetězce pro tisk dat. Nastavení, zda exportovat směrovací cesty do souboru. Proměnné pro globální statistiky a další...

Funkce `dsr_rte_stats_reg();` zajišťuje registraci statistik a vymezuje paměť pro struktury, se kterými pracuje.

Další funkce `dsr_notification_log_init();` slouží pro inicializaci různých oznámení protokolu DSR.

Funkce `dsr_rte_attributes_parse_buffers_create();` pracuje jako parser/třídič atribut nastavený v procesním modelu a zároveň nastavuje prostor (buffer) pro tříděná data. Pracuje tedy téměř se všemi proměnnými, jenž je možné nastavit v parametrech protokolu DSR.

Funkce která registruje směrovací proces DSR jako vyšší vrstvu v uzlu IP se nazývá `Ip_Higher_Layer_Protocol_Register("dsr",&higher_layer_proto_id);`.

Nevynucený stav – Výstup: wait

Uvnitř tohoto stavu probíhá práce především s dočasnými proměnnými, jež jsou obsahem funkcí vyvolaných z jiných procesů například z CPU. O tuto práci se stará funkce `invoker_prohandle=op_pro_invoker(own_prohandle,&invoke_mode);`.

Proměnné a Bloky – Variables and Blocks

Stavový automat respektive procesní model `dsr_rte` rozhodně nevystihuje celkovou podstatu fungování protokolu DSR. Jeden z cílů je rozšířit datovou jednotku směrovacího protokolu DSR. V prostředí OM lze nahlédnout přímo do zdrojových kódů. Prvním vodítkem jsou komentáře uvnitř funkčního bloku.

V oblasti **Packet Arrival Functions** se nachází funkce, jenž se starají o **příchod paketů**, jejich rozdělení podle TLV (Type, Lenght, Value) uvnitř headrů. V případě, že jsou pakety rozděleny a patří do správné vrstvy budou přiděleny další funkci obvykle s koncovkou `process` jenž se postará o další zpracování.

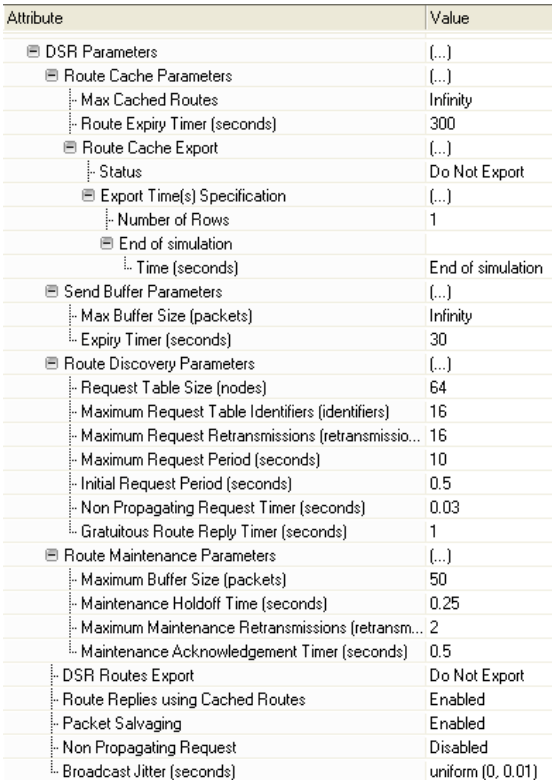
Funkce `dsr_rte_received_pkt_handle();` se dá považovat za hlavní, protože jako první zpracovává příchozí paket a rozhoduje a o jeho postupu. Podřízenou funkcí, která dělí pakety podle typů je `dsr_rte_app_pkt_arrival_handle();`. Je několik možností kam může tato funkce poslat paket dále ke zpracování. Může se jednat například o chybnou zprávu, zprávu údržby, ale častější o zprávu režie sítě. Jako příklad bude zvolena zpráva Route Request. Při příchodu zprávy tohoto typu se o její zpracování postará funkce `dsr_rte_received_route_request_process();`

Oblast funkčního bloku **Packet Creation Functions** obsahuje hlavní zprávy, které protokol DSR zasílá. Lze tu nalézt funkce pro **tvorbu zpráv** starající se o režii sítě pomocí RREQ a RREP paketů, zprávu paměti nebo zaslání zpráv o chybné trase, či možnosti o jejím zkrácení.

Funkce `dsr_rte_route_request_send();` má několik podřízených funkcí, jedna z nich `dsr_pkt_support_route_request_tlv_create();` vytváří TLV „možnosti“ paketu RREQ. Pro každou zprávu protokolu DSR je ve FB funkce tohoto typu. Parametry, které vytváří jsou podstatné protože určují rozměry paketu. Jedním z cílů práce je rozšíření datové jednotky a proto je toto zásadní model funkce. [8]

3.3 DSR parametry v prostředí OPNET Modeler

Simulační prostředí OPNET Modeler umožňuje nastavit protokol DSR do sítí Ad-Hoc. Program navíc podporuje možnost změny parametrů protokolu samotného. V kapitole 2.1 jsme popisovali základní mechanismy protokolu DSR. Jejich nastavení, je pro praktickou část dokumentu zásadní. Na obr. 3.4 vidíme rozbalenou strukturu možných nastavení protokolu, jejich názvy (Attribute) a hodnoty (Value). Parametry na obrázku jsou nastaveny jako defaultní.



Attribute	Value
DSR Parameters	(...)
Route Cache Parameters	(...)
Max Cached Routes	Infinity
Route Expiry Timer (seconds)	300
Route Cache Export	(...)
Status	Do Not Export
Export Time(s) Specification	(...)
Number of Rows	1
End of simulation	
Time (seconds)	End of simulation
Send Buffer Parameters	(...)
Max Buffer Size (packets)	Infinity
Expiry Timer (seconds)	30
Route Discovery Parameters	(...)
Request Table Size (nodes)	64
Maximum Request Table Identifiers (identifiers)	16
Maximum Request Retransmissions (retransmissio...	16
Maximum Request Period (seconds)	10
Initial Request Period (seconds)	0.5
Non Propagating Request Timer (seconds)	0.03
Gratuitous Route Reply Timer (seconds)	1
Route Maintenance Parameters	(...)
Maximum Buffer Size (packets)	50
Maintenance Holdoff Time (seconds)	0.25
Maximum Maintenance Retransmissions (retransm...	2
Maintenance Acknowledgement Timer (seconds)	0.5
DSR Routes Export	Do Not Export
Route Replies using Cached Routes	Enabled
Packet Salvaging	Enabled
Non Propagating Request	Disabled
Broadcast Jitter (seconds)	uniform (0, 0.01)

Obr. 3.4: Defaultní parametry protokolu DSR v prostředí OPNET Modeler.

- **Route Cache Parameters**: Parametry směrovací paměti uzlů
  - *Max Cached Routes*: Maximální počet tras v paměti uzlů
  - *Route Empiry Timer (second)*: Čas kdy vyprší směrovací trasa
  - *Route Cahce Export*: Export směrovací paměti uzlů
    - Status: Možnost vybrat export
    - Export Time(s) Specification
      - *Time (seconds)*: Definovaný čas kdy exportovat směrovací paměť

- **Send Buffer**: Vyrovnávací paměť dat
  - *Max buffer Size (packets)*: Maximální velikost fronty dat (V případě, že mechanismus Route discovery ještě nesestavil směrovací cestu, tak se začne vytvářet tato fronta).
  - *Expiry Timer (seconds)*: Časový interval zahození paketů (Každý paket ve frontě obsahuje čas vypršení (zahození), který udává Expiry timer)

- **Route Discovery Parameters**: Parametry hledání tras
  - *Request Table Size (nodes)*: Tato hodnota určuje maximální počet stanic (destinací), které „směrovací pole“ udrží v aktivní žádosti.
  - *Maximum Request Table Identifiers (identifiers)*: Každé směrovací pole si drží záznamy tras k uzlům. Tato hodnota určuje maximální počet ID záznamů ke specifické cílové adrese.
  - *Maximum Request Retransmissions (retransmissions)*: Hodnota udává maximální počet pokusů o znovu zaslání žádosti (Request Route) pro specifickou cílovou adresu.
  - *Maximum Request Period (seconds)*: Když se mechanismu Route Discovery nepodaří najít směrovací trasu k cíli, tak se interval dalšího hledání zdvojnásobí až do maximální hodnoty Request Period.
  - *Initial Request Period (seconds)*: V případě, že mechanismus Route Discovery nedosáhne poprvé svého cíle, tak parametrem Initial Request Period určíme čas, za který se pokusí znovu vyhledat správnou trasu. (Platí pouze pro první pokus. Další pokusy pokaždé dvojnásobně rostou až do Maximum Request Period).
  - *Non Propagating Request Timer (seconds)*: Časový interval rozesílání non-propagating žádostí, které mají nastevený TTL (Time to Live) na hodnotu 1.
  - *Gratuitous Route Reply Timer (seconds)*: Časový limit pro dobrovolný zápis, zachyceného paketu, do směrovacího pole.

- **Route Maintance Parameters**: Parametry údržby cest obsahující zprávy žádosti (Maintance Request) a potvrzení (Maintance Acknowledgements).
  - *Maximum Buffer Size (packets)*: Maximální počet paketů sloužící k údržbě tras, které je vyrovnávací paměť schopna udržet.
  - *Maintance Holdoff Time (seconds)*: Při přeposílání paketu, musí uzel zjistit, jestli je následující uzel dostupný. Jestliže mu od sousedního uzlu přišlo potvrzení v době dřívější než je čas zdržení (Holdoff Time), tak při dopravě paketu nemusí znovu zjišťovat dostupnost. Uzel tedy může obejít potvrzení dostupnosti do doby, než vyprší čas zdržení (Holdoff time) a v

tomto intervalu posílat pakety bez omezení.
    - *Maximum Maintance Retransmissions (retransmissions)*: Abychom zjistili, jestli je sousední uzel dosažitelný, vysíláme žádosti Route Request. Tato hodnota udává maximální počet opakovaných žádostí.
    - *Maintance Acknowledgement Timer (seconds)*: Pokud za dobu tohoto časového intervalu nepřijde ACK (Acknowledgment) potvrzení, tak uzel přepošlu paket jinému uzlu než je uzel cílový.

- **DSR routes Export**: Export směrovacích cest přímo pro stanici
- **Route Replies using Cached Routes**: Místo přeposlání Route Request paketu dál. Uzel přijímající route request, který mu není adresován, rozbalí tento paket a začne hledat v jeho vlastní směrovací paměti cestu k cíli žádosti. Pokud uzel tuto cestu najde a tento parametr je aktivní, potom uzel navrátí route reply ke zdrojovému uzlu s cestou z jeho vlastní směrovací paměti.
- **Packet Salvaging**: Oprava trasy. Když mezilehlý uzel pomocí Route maintance zjistí, že následující hop je nedostupný, nemusí paket rovnou zahodit, ale díky tomuto nastavení může paket zachránit tím, že do paketu zapíše alternativní cestu z Route cache.
- **Non Propagating Request**: Povolením tohoto parametru, uzel implementuje „non-propagating“ route request jako počáteční volbu Route Discovery. Nastavení umožní, aby uzel poslal Route Request s hodnotou hop limit 1, čili uzly které tento RREQ přijmou, jej nemohou dále všesměrově rozeslat, protože je paket zahozen. Pokud se uzlu nevrátí žádná Route Reply vyšle uzel Propagation Route Request, který není omezen TTL a může být dál přeposílán (pro možnost poslání non-propagating Route Request je třeba mít toto nastavení aktivováno).
- **Broadcast Jitter** (second): Na všesměrové zasílání režijních paketů je aplikováno male zpoždění, které je rovnoměrně rozloženo v rozmezí mezi nulovou hodnotou a Broadcast jitterem. Platí pro tyto případy:
    1. Když uzel vytvoří, posílá Route Reply (neplatí pro uzly, které Route Reply jen přeposílají).
    2. Když uzel znovu všesměrově zašle Route Request (neplatí pro uzly, ze kterých je originálně route request zaslán).

4 PRAKTICKÁ ČÁST

Tato kapitola se zabývá konfigurací protokolu DSR v prostředí OPNET Modeler. Dále simulacemi testující možnosti protokolu. Především bude testována funkčnost implementace, vlastnosti sítě při různých rychlostech pohybu mobilních MANET stanic a možné optimalizace. Dále orientační simulace na generování paketů a především simulace jenž bude testovat rozšíření hlavičky datové jednotky RREQ směrovacího protokolu DSR. Jednotlivým simulacím bude předcházet konfigurace modelu. Na konci každá podkapitoly bude vyhodnocení výsledků. Veškeré provedené simulace jsou přiloženy na CD.

4.1 Simulace zaměřená na funkci protokolu DSR

Vytvoříme scénář s názvem **DSR_function** o rozměrech 4x4 km model prostředí Campus s celkovým počtem šestnácti stanic, vizuálně rozmístěných do mřížky v na ploše 3x3 km. Každá stanice má teoretický dosah 1 km dle doporučení OPNET Technologies Inc. Do všech stanic je implementován protokol DSR. Všechny stanice obdrží IP adresu verze 4. Přenos dat bude probíhat ze stanice **node_1** do stanice **node_16** a naopak. Zátěž daty je podle nastavení konstantní jak v čase, tak ve velikosti paketů. Scénář by měl ilustrovat poměrně ideální komunikaci a bezztrátový přenos dat s velice malou odezvou. Na tomto modelu máme také zájem zobrazit výběr ideální směrovací trasy, sestavené základními mechanismy protokolu DSR.

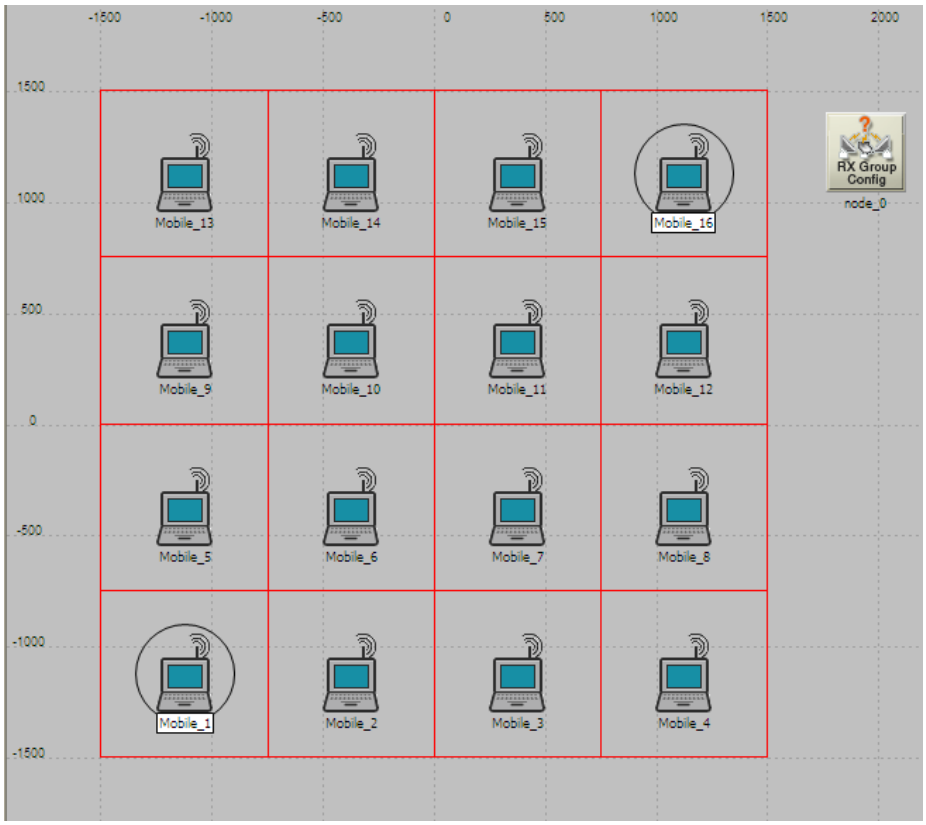
4.1.1 Konfigurace modelu

Založení projektu

1. Spustíme **OPNET Modeler**.
2. Vybereme položku **File** $\mapsto$ **New**, potvrdíme project.
3. Zadáme jméno projektu **project_DSR** a jméno scénáře **DSR_function**.
4. S pomocí Startup Wizard vytvoříme prázný scénář Create empty scenario.
5. Dále vybereme pro nás vhodné prostředí Campus.
6. Rozměry prostředí nastavíme na 4x4 kilometrů.
7. V tabulce Model Family vyhledáme položku MANET a změníme hodnotu na YES. Tím jsme vybrali základní nastavení palety pro MANET sítě.
8. Stisknutím Finish dokončíme konfiguraci prostředí.

Vytvoření modelu

1. Otevřel se nám Object pallete tree. Klikneme na záložku MANET a dvojklikem vybereme **manet_station**. Těchto stanic rozmístíme celkem 16, do sítě připomínající mříž jako na obr. 4.1.
2. Dále vybereme model **Rxgroup Config**, který umístíme ke stanicím.
3. Nastavíme dosah stanic. Pravým tlačítkem klikneme na **Rxgroup Config** ↦ **Edit Attributes** ↦ **Reciever Selection Parameters** ↦ **Distance Treshold** (meters) nastavíme na hodnotu 1000 metrů.
4. Nastavíme, aby vypočet dosahu se opakoval v intervalu pěti sekund **Rxgroup Config** ↦ **Edit Attributes** ↦ **Duration** ↦ **Refresh Interval** zvolíme 5 sekund.
5. Výběrem **Protocols** ↦ **IP** ↦ **Addresing** ↦ **Auto-Assign IPv4 Addresses** nakonfigurujeme na všech stanicích modelu, IP adresu verze 4.
6. Vybereme všechny stanice. Pravé tlačítko myši na **node_X** ↦ **Sellect Similar Node** ↦ **Edit Attributes** ↦ **AD-HOC Routing Parameters** ↦ **AD-HOC Routing Protocol** nastavíme na **DSR** a nesmíme zapomenout zaškrtnout **Apply to selected objects**.



Obr. 4.1: Síť stanic 3x3 km pro scénář DSR_function.

Nastavení přenosu dat

1. Klikneme pravým tlačítkem myší na uzel **node_16** ↦ **Edit Attributes** ↦ **MANET Traffic Generation Parameters** volbu **Number of Rows** zvolíme na 1. Tím se nám otevře další tabulka nastavení. **Start Time** (seconds) dáme 110 sekund. **Packet Inter-Arrival Time** (seconds) zvolíme **constant** (0.03). Velikost paketu **Packet Size** (bits) zvolíme **constant** (1024). Cílovou IP adresu nastavíme podle uzlu jedna (**node_1** ↦ **Edit Attributes** ↦ **IP** ↦ **IP Host Parameters** ↦ **Interface Information**). Pro nás tedy **Destination IP Address** 192.0.1.1 a potvrdíme OK.
2. Stejný postup opakujeme pro stanici **node_1** (rozdíly v nastavení **node_16** a **node_1** jsou vidět na obr. 4.2).

Attribute	Value
DSR Parameters	(...)
Route Cache Parameters	(...)
Max Cached Routes	Infinity
Route Expiry Timer (seconds)	300
Route Cache Export	(...)
Status	Do Not Export
Export Time(s) Specification	(...)
Number of Rows	2
End of simulation	
Time (seconds)	End of simulation
150	
Time (seconds)	150
Send Buffer Parameters	Default
Route Discovery Parameters	Default
Route Maintenance Parameters	Default
DSR Routes Export	Do Not Export
Route Replies using Cached Routes	Enabled
Packet Salvaging	Enabled
Non Propagating Request	Disabled
Broadcast Jitter (seconds)	uniform (0, 0.01)
GRP Parameters	Default
OLSR Parameters	Default
TORA/IMEP Parameters	Default
VPN	
DHCP	
Reports	
IP	
MANET Traffic Generation Parameters	(...)
Number of Rows	1
110	
Start Time (seconds)	110
Packet Inter-Arrival Time (seconds)	constant (0.03)
Packet Size (bits)	constant (1024)
Destination IP Address	192.0.1.1
Stop Time (seconds)	End of Simulation
Wireless LAN	
Wireless LAN MAC Address	Auto Assigned
Wireless LAN Parameters	(...)
BSS Identifier	0
Access Point Functionality	Disabled
Physical Characteristics	Direct Sequence
Data Rate (bps)	11 Mbps
Channel Settings	Auto Assigned
Transmit Power (W)	0.005
Packet Reception-Power Threshold...	-95

Attribute	Value
DSR Parameters	(...)
Route Cache Parameters	(...)
Max Cached Routes	Infinity
Route Expiry Timer (seconds)	300
Route Cache Export	(...)
Status	Export
Export Time(s) Specification	(...)
Number of Rows	2
End of simulation	
Time (seconds)	End of simulation
150	
Time (seconds)	150
Send Buffer Parameters	(...)
Route Discovery Parameters	(...)
Route Maintenance Parameters	(...)
DSR Routes Export	Do Not Export
Route Replies using Cached Routes	Enabled
Packet Salvaging	Enabled
Non Propagating Request	Disabled
Broadcast Jitter (seconds)	uniform (0, 0.01)
GRP Parameters	Default
OLSR Parameters	Default
TORA/IMEP Parameters	Default
VPN	
DHCP	
Reports	
IP	
MANET Traffic Generation Parameters	(...)
Number of Rows	1
120	
Start Time (seconds)	120
Packet Inter-Arrival Time (seconds)	constant (0.03)
Packet Size (bits)	constant (1024)
Destination IP Address	192.0.1.16
Stop Time (seconds)	End of Simulation
Wireless LAN	
Wireless LAN MAC Address	Auto Assigned
Wireless LAN Parameters	(...)
BSS Identifier	0
Access Point Functionality	Disabled
Physical Characteristics	Direct Sequence
Data Rate (bps)	11 Mbps
Channel Settings	Auto Assigned
Transmit Power (W)	0.005
Packet Reception-Power Threshold...	-95

Obr. 4.2: Kompletní nastavení node_16 a node_1 pro DSR_function.

Export směrovací paměti uzlů

1. Vybereme uzel, na němž máme MANET Traffic např. **node_16** ↦ **Edit Attributes** ↦ **AD-HOC Routing Parameters** ↦ **DSR Parameters** ↦ **Route Cache Parameters** ↦ **Route Cache Export** ↦ **Export Time Specification** zvolíme **Number of Rows** na hodnotu 2. Čas exportu **Time** (seconds) nastavíme na 150.

Individuální statistiky

1. Označíme stanice, pro které chceme udělat individuální charakteristiky např. **node_1**, **node_16** ↦ **Choose Individual DES Statistics** ↦ **Node Statistic** ↦ **DSR, MANET, Wireless Lan** a potvrdíme OK.

Spuštění simulace

1. Na panelu nástrojů vybereme **DES** ↦ **Configure/Run DES** (nebo **ctrl**+**R**)
   (a) Nastavíme dobu simulace **Duration** na 250 seconds.
   (b) Necháme přesnost 100 hodnot na statistiku – **Values per statistic**.
   (c) **Simulation Kerne**l definuje způsob překladu modelu do spustitelného kódu. Nastavíme na **Optimized**.
2. Po spuštění simulace tlačítkem **Run** se objeví další okno Simulation progress. V případě, že simulace proběhne v pořádku, uvidíte Simulation complete nyní můžete kliknout na Close.

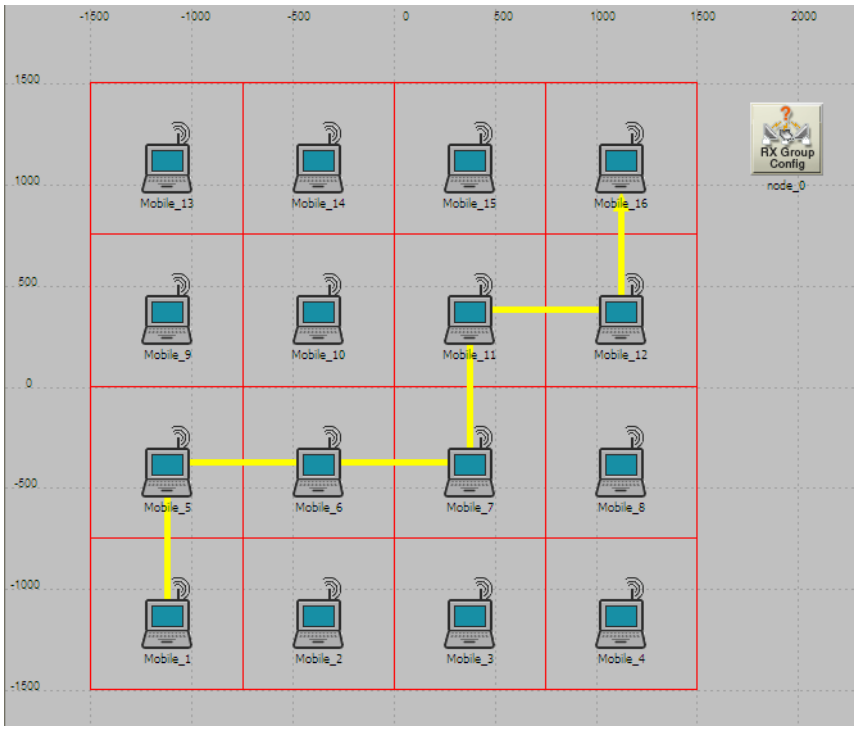
Zobrazení výsledků

1. Grafy režijních paketů. Klikneme na ikonu **View results** ↦ **Global Statistics** ↦ **DSR** a zaškrtneme **Routing Traffic Recieved** (bits/sec) + **Routing Traffic sent** (bits/sec) dále pod grafem vybereme volbu **Presentation** ↦ **Overlaid Statistics**
2. Zpoždění sítě MANET. Klikneme na ikonu **View results** ↦ **Global Statistics** ↦ **MANET** zaškrtneme **Delay**(secs) dále pak volbu **Presentation** ↦ **avarage**.
3. Pro zobrazení směrovací paměti stanic. Přepneme na záložku **DES Run (1) Tables** kde máme vyexportované směrovací cesty. **Object Tables** ↦ **Campus Network** ↦ **Node1 nebo Node 16** ↦ **DSR** ↦ **Route Cache at 150 seconds**.
4. Pro všechny vyhodnocení obecně platí zobrazení statistik tlačítkem Show, kde můžeme dále upravovat graf (osy, vhled, možnost zobrazení atd.)

5. Vizuální zobrazení směrování lze nastavit pomocí **Protocols** ↦ **MANET** ↦ **DSR** ↦ **Display DSR Routes**.

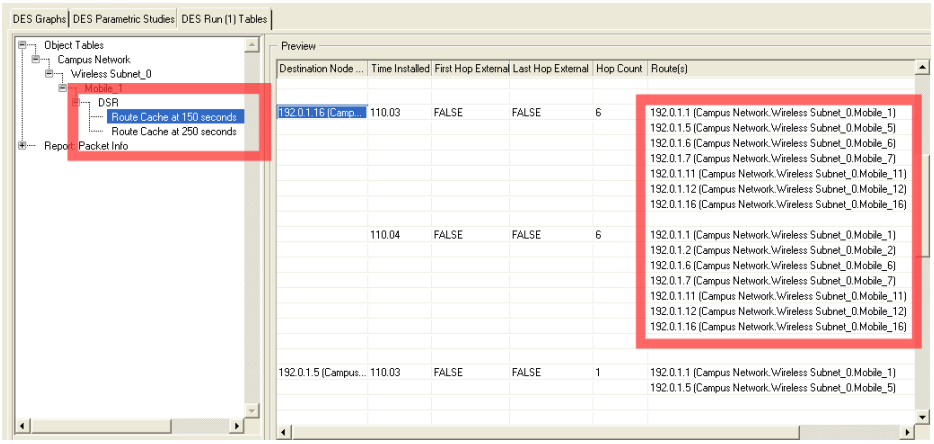
4.1.2 Vyhodnocení výsledků

Na obr. 4.3 vidíme, že směrování ze stanice 1 do stanice 16 proběhlo vizuálně v pořádku. Z teorie víme, že reaktivní protokol DSR fungující na systému žádosti tzv. On-demand by měl vždy vybrat co nejkratší trasu podle metriky dané počtem hopů na sousední stanice. Jako nástroj pro potvrzení teorie využijeme paměť samotných stanic. Proto jsme exportovali směrovací paměť šestnácté stanice stanice v čase 150 vteřin.



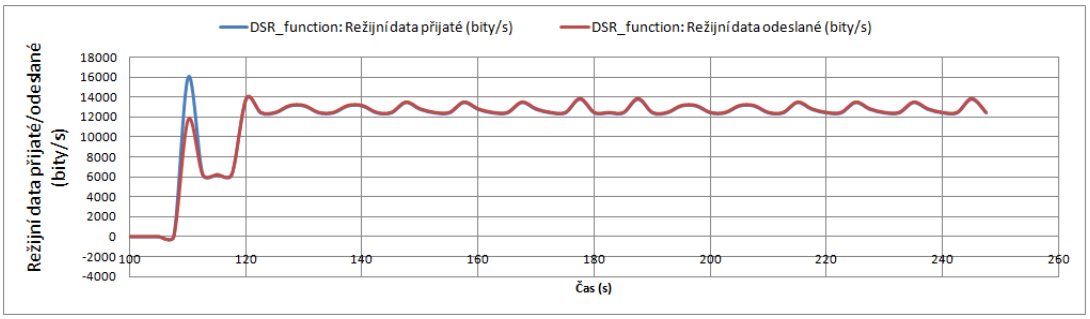
Obr. 4.3: Směrování stanic znázorněno vizuálně.

Na obr. 4.4 lze vidět dvě směrovací cesty se stejným počtem hopů. Jednu protokol DSR vyhodnotil jako rychlejší díky dřívějšímu příchodu RREP paketů. Přesto druhou cestu vnímá jako alternativu, kdyby nastaly na první trase problémy. Dá se tedy konstatovat, že implementace protokolu DSR je funkční po stránce směrování. Protože parametry simulace, ale i samotný model byly stavěny jako ideální, aby mohla být ověřena funkčnost protokolu DSR v prostředí OPNET modeler.



Obr. 4.4: Směrovací paměť (Route Cache) uzlu 16.

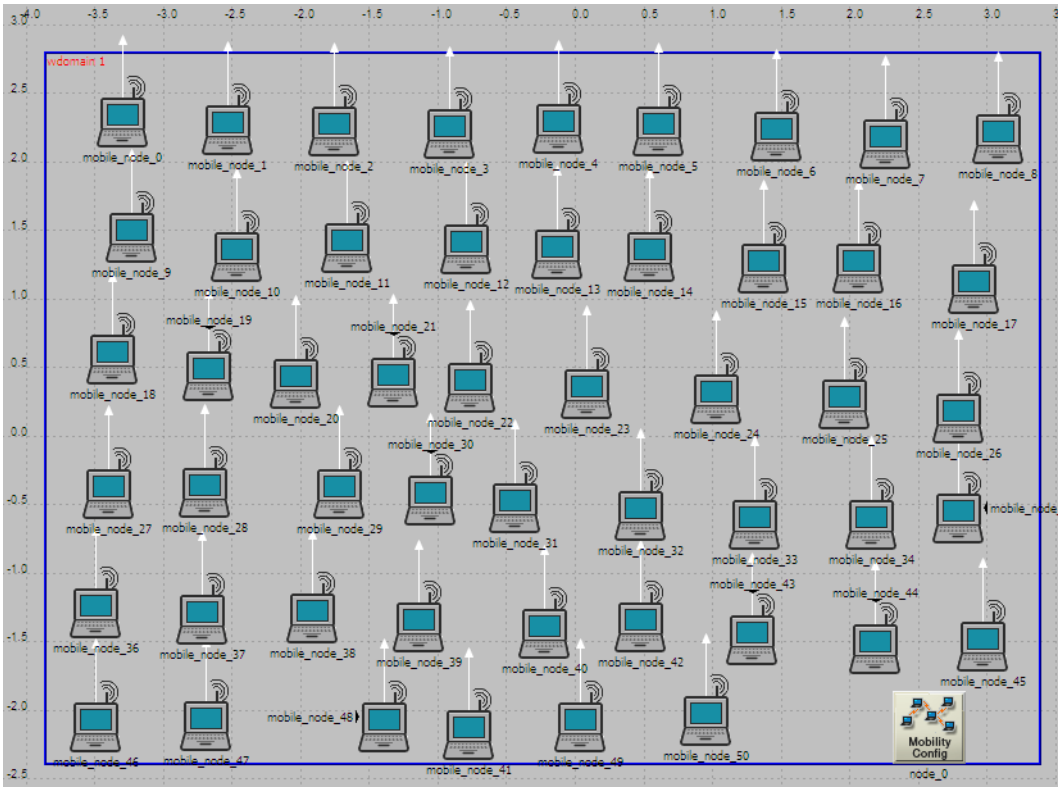
Lze očekávat ideální charakteristiky přenosu režijních dat, které přehledně zobrazuje graf na obr. 4.5. Modrá barva značí provoz režijních paketů přijatých a červená provoz odeslaných režijních paketů. Jejich charakteristiky jsou téměř identické. Z toho usuzujeme, že se jednalo o bezproblémový přenos dat a implementace protokolu DSR je funkční i po stránce přenosu dat. Dalším důležitým parametrem je bezesporu odezva, která se v tomto scénáři pohybuje v průměru pod hranicí 6ms, což je skvělý výsledek, který se ale dá u takto malé sítě předpokládat.



Obr. 4.5: Režijní přenos dat mezi node_0 a node_16.

4.2 Simulace různých rychlostí pohybu

Dalším simulačním modelem je znovu prostředí Campus, tentokrát s rozlohou 8x6 kilometrů a s počtem 50 stanic, které se budou náhodně pohybovat v tomto teritoriu s třemi různými rychlostmi. Pro každou rychlost náhodného pohybu bude vytvořen vlastní scénář, aby bylo možné porovnat výsledné charakteristiky. První rychlost pohybu bude 5 m/s (18 km/h), druhá 15 m/s (54 km/h) a nakonec třetí rychlost náhodného pohybu uzlů 25 m/s (90 km/h). Jednotlivé rychlosti byly voleny tak, aby se výsledné charakteristiky zobrazovaly s co největší vypovídací hodnotou. Základní schéma modelu lze vidět na obr. 4.6, nicméně trajektorie náhodného pohybu nejsou vykresleny z důvodu přehlednosti obrázku.

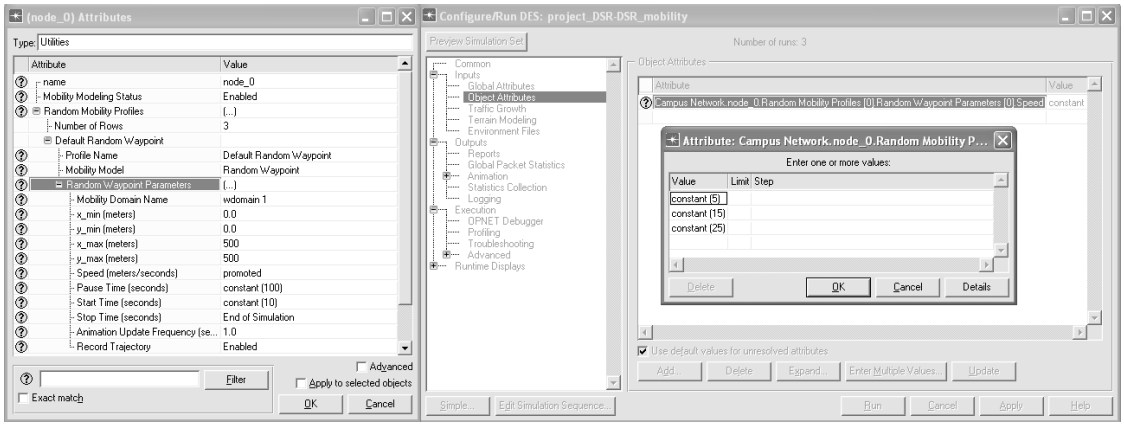


Obr. 4.6: Model padesáti stanic pro různé tři rychlosti náhodného pohybu.

4.2.1 Konfigurace modelu

1. Vytvoříme nový scénář s názvem **DSR_mobility**, znovu Campus model prostředí. Tentokrát s rozměry 8x6 kilometrů a počtem 50 uzlů.
2. Všechny stanice umístíme do prostoru **Domain Mobility**, který nám určí přesný prostor pohybu stanic.

3. Z palety MANET vložíme komponentu **Mobility Config**, kterou nastavíme příslušnou pohyblivost stanic.
4. Do všech stanic implementujeme protokol **DSR** s jeho defaultním nastavením parametrů.
5. Přidělíme IP adresy verze 4.
6. Nastavíme tři různé rychlosti pohybu. Rychlost v Mobility Config povýšíme jako proměnou a pro simulaci ji zvolíme tři vstupní konstanty, toto nastavení lze přehledně vidět na obr. 4.7.
7. Nakonec nastavíme provoz (Traffic) mezi vybranými uzly (Pro náš případ se jednalo o celkem 3 provozy a to uzly: **1** $\mapsto$ **34, 5** $\mapsto$ **24 a 6** $\mapsto$ **14**.

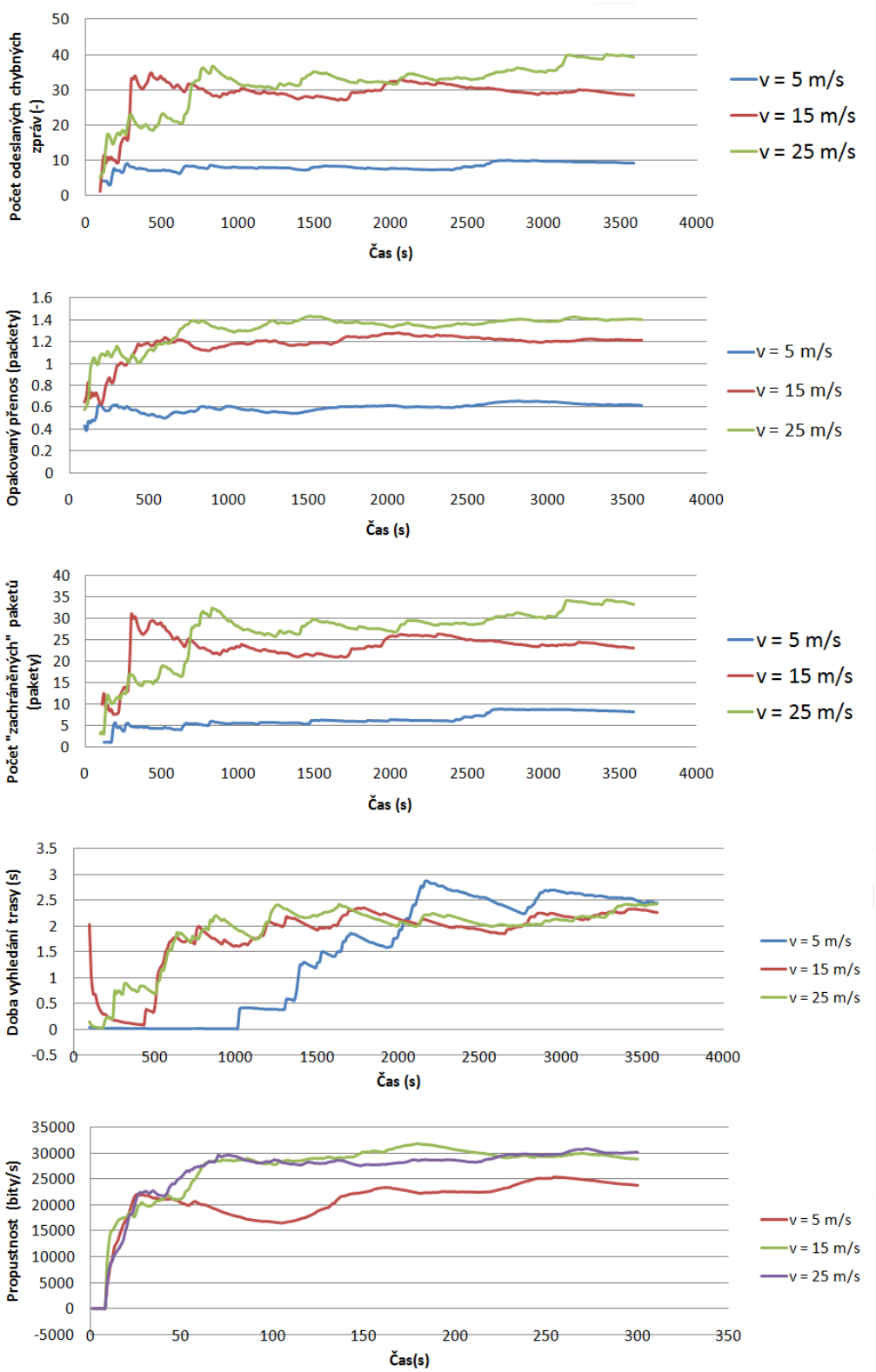


Obr. 4.7: Mobility config – Vstupní konstanty rychlosti.

4.2.2 Vyhodnocení výsledků

Obr. 4.7 sumarizuje výsledky tří simulací náhodného pohybu padesáti stanic. Veškeré grafy zobrazují průměrné hodnoty v čase. Celkem 5 grafů poukazuje na to, že směrovací kvality protokolu DSR s rostoucí rychlostí pohybu stanic klesají. Rychlosti jsou na grafech odstupňovány následovně: 5m/s - modrá barva DES-1, 15 m/s - červená barva DES-2, 12 m/s - barva zelená DES-3. Na grafu **Total Route Error**, lze pozorovat, že s rostoucí rychlostí pohybu stanice se rapidně zvyšuje počet zpráv signalizující chyby ve spojení. Tento problém musí protokol řešit častějším přeposíláním paketů viz graf **Retransmission Attempts**. Dále má protokol možnost pakety tzv. zachránit, tedy rovnou je nezahodit, ale přeposlat do alternativní cesty mechanismem **Packet Salvage**. Počet těchto „zachráněných“ paketů je zobrazen na grafu **Total Packets Salvaged**.

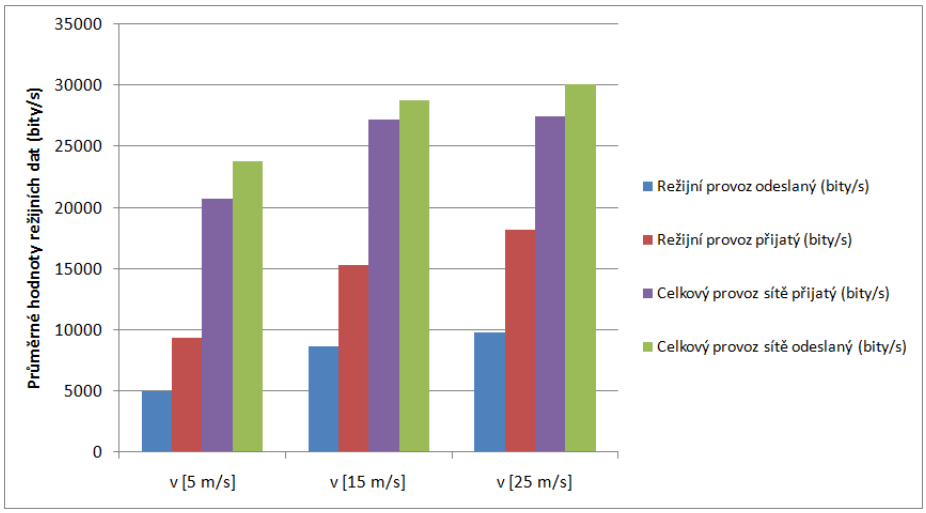
Rostoucí problémy v síti radikálně zvyšují počet režijních paketů, viz Obr. 4.10, což nepříjemně ovlivňuje odezvu. Charakteristika grafu **Route Discovery Time**,



Obr. 4.8: Výsledky simulace pohybu padesáti stanic.

popisuje zpoždění mezi procesem vyhledání trasy od zdroje k cíli. Obvyklé hodnoty se pohybují v okolí 2000 ms. Kritické hodnoty dosahují do výše 3000 ms, což je velice neuspokojivé. Síť se tedy se zvyšující rychlostí pohybu stanic potýká s problémy např.: Větší chybovost při vyhledávání tras a jejich neuspokojivá odezva, či zvýšený počet režijních dat. Přesto na grafu **Wireless LAN Throughput** (bits/sec) pozorujeme kvalitní přizpůsobení propustnosti sítě.

Obr. 4.9 vyžaduje podrobnější vysvětlení. Celkem čtyři barvy signalizují provoz sítě odeslaných (Sent) nebo přijatých (Recieved) bitů za sekundu. Tři sloupcové grafy značí tři různé rychlosti (viz výše).Lze vidět, že ve všech případech jsou přijatá režijní data větší než odeslaná. To je způsobeno promiskuitním módem stanic, které zachytávají režijní pakety provozu, i když jim nejsou adresovány. Tento mechanismus udržuje stanice lépe informované z pohledu směrování. Nevýhodou je však energetická náročnost.
Z grafu lze odečíst, že pakety zajišťující režii tvoří, pro první rychlosti náhodného pohybu stanic, asi 20%. Pro druhý průběh, tedy rychlost 15 m/s, je režie ještě vyšší a pohybuje se na hranici 30% z průměrného počtu všech odesílaných dat. Poslední průběh jen kopíruje předchozí zkušenost a vyšší rychlost mobility stanic kompenzuje větším počtem režijních paketů, které dosahují až 35% z průměrného přenosu všech dat.



Obr. 4.9: Provoz sítě pro padesát stanic v pohybu.

4.3 Simulace zaměřená na optimalizaci parametrů

V této kapitole budou měněny parametry protokolu DSR tak, aby bylo docíleno lepších vlastností sítě. Ze získaných teoretických a praktických znalostí o protokolu DSR byly vybrány parametry, které provoz sítě zasáhnou nejvíce. Tyto operace lze provést na současném modelu z předchozí simulace (viz kapitola 4.2), ale s výběrem pouze jedné rychlosti náhodného pohybu uzlů (25 m/s). Tato rychlost byla volena kvůli největší citlivosti na provedené změny parametrů a tedy i nejlépe rozpoznatelné výsledky.

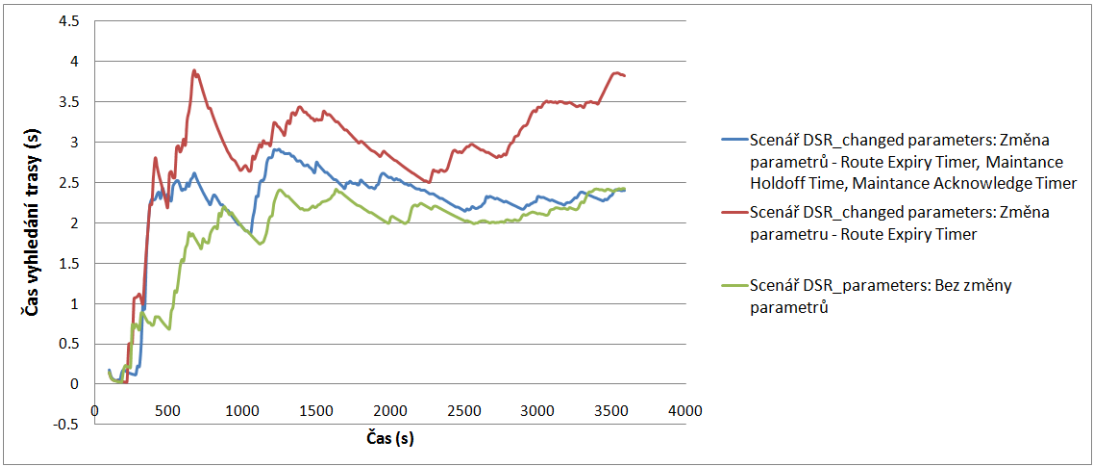
Je třeba se soustředit na kvalitní spojení s co nejmenším počtem chybových zpráv, **Route Error Messages**. Dále také na zpoždění při hledání nové trasy Route Dicovery time a především na počet odeslaných režijních paketů v poměru s celkovým počtem odeslaných dat.

4.3.1 Konfigurace modelu

1. Scénář **DSR_mobility** duplikujeme a pojmenujeme **DSR_parameters**.
2. **DSR_parameters** upravíme na jednu konstantní rychlost náhodného pohybu stanic a v Config Mobility a vymažeme vstupní proměnné z položky **Object Attributes**.
3. Dále tento scénář duplikujeme ještě dvakrát s názvy **DSR_changed parameters** a **DSR_changed parameters 2**.
4. Ve scénáři **DSR_changed parameters** změníme parametr **Route Expiry Timer** (Neboli čas kdy vyprší směrovací trasa), z 300 na 30 sekund (Tuto hodnotu volíme na základě rychlosti pohybu uzlů. Předpokládáme, že při rychlosti 25 m/s se stanice za tento čas ztratí z dosahu ostatních stanic jen s malou pravděpodobností).
5. Ve scenáři **DSR_changed parameters 2** změníme parametr **Route Expiry Timer**, z 300 na 30 sekund, dále **Maintance Holdoff Time** (Zvýšíme propustnost dat tím, že snížíme počet zpráv, které jsou potřeba k ověření dostupnosti souseda při posílání dat) na 600 ms (Dvojnásobná hodnota defaultního nastavení. Tímto snížíme počet kontrolních zpráv o polovinu. Hodnota se dá ještě zvyšovat, ale na úkor spolehlivosti přenosu dat) a nakonec **Maintance Acknowledgement Timer** (Kontrola spojení mezi uzly) z hodnoty 500 na 150 ms (Sice zvýšíme počet režijních zpráv, ale docílíme tím velké stability přenosu a zajistíme plnou funkci parametru **Holdoff Time**, bez zbytečných komplikací).

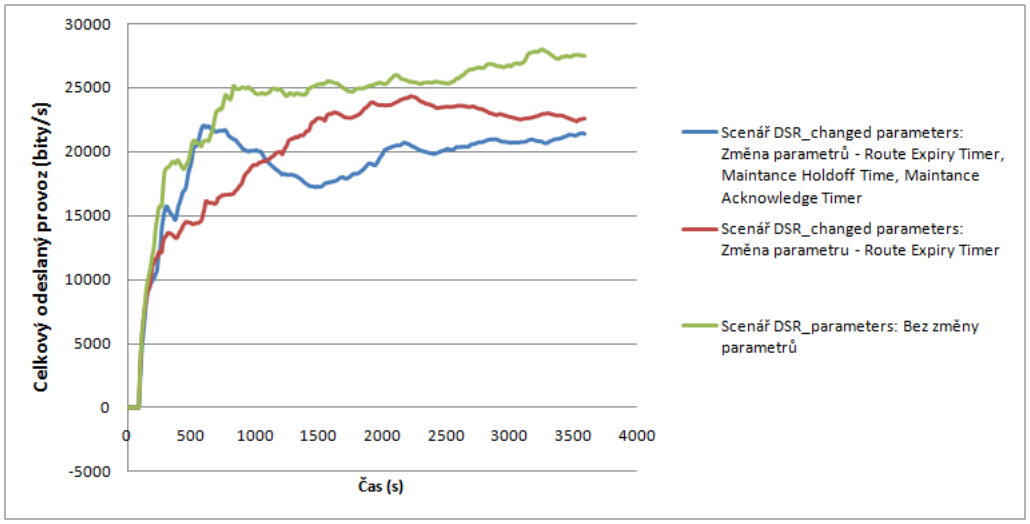
4.3.2 Vyhodnocení výsledků

Je třeba zobrazit grafy všech 3 scénářů **DSR_(changed)parameters (2)**, tím, že se v prohlížeči výsledků (**Result Browser**) vybere ze záložky **Resul for ↦ Current project** kde lze zaškrtnout všechny tři scénáře související s optimalizací parametrů protokolu DSR. Na obr. 4.10 a 4.11 lze vidět celkem 2 různé grafy, které obsahují vždy 3 barevné charakteristiky. Zelená barva, pro defaultní parametry protokolu DSR, červená barva po změně prvního parametru a nakonec modrá barva pro všechny tři změny parametrů. Z hodnot víme **Total Route Error Sent**, že úpravou parametrů protokolu **DSR** bylo docíleno mnohem menšího počtu chybových zpráv a to v nemalém měřítku. Bohužel úprava hodnoty časového intervalu **Route Expiry Timer** nebyla pro scénář **DSR_changed parameters** dostačující. Následek tohoto zásahu byl velký nárůst zpoždění mechanismu na hledání trasy (Route Discovery), to lze pozorovat na červené charakteristice **Route Discovery Time**.



Obr. 4.10: Zpoždění vyhledávacího mechanismu protokolu DSR.

Proto byly zavedeny pro scénář **DSR_changed parameters 2** další dvě změny parametrů **Maintance Holdoff Time Maintance** a **Acknowledgement Timer**. Tyto změny lze pozorovat jako zelenou charakteristiku, která zpožděním odpovídá původnímu nastavení parametrů DSR. Největší změnou je průměrný počet odeslaných paketů, které jsou znázorněny na obr. 4.11 v grafu **Total Traffic Sent** v bitech za sekundu. Protože je simulace tvořena za konstantního toku dat, můžeme odvodit, že bylo tímto krokem ušetřeno velké množství režijních dat. Tyto rozdíly jsou patrné hlavně ke konci simulace, při porovnání zelené a modré charakteristiky.



Obr. 4.11: Výsledný datový tok pro upravené parametry DSR.

4.4 Simulace zaměřená na generování paketů

V tomto modelu se pokusíme ověřit schopnost komunikace mezi dvěma duplexně spojenými stanicemi v OM. Komunikace bude zajištěna s pomocí protokolu UDP (User Datagram Protocol). Cílem je znovu podle předlohy vytvořit procesní model, který je schopen uskutečnit přenos datové jednotky, přijmout ji na straně přijímače a zobrazit její obsah. Tento model slouží jako kvalitní základ informací pro rozšíření datové jednotky směrovacího protokolu DSR. Základem pro tento model bude práce V. Mikulici [5].

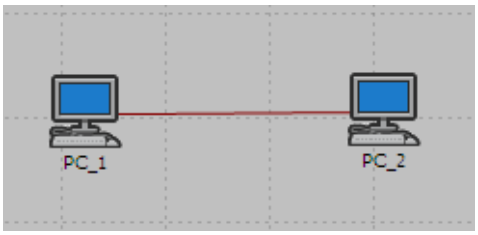
4.4.1 Konfigurace modelu

Založení projektu

1. Spustíme **OM**.
2. S pomocí Startup Wizard vytvoříme prázdný scénář Create empty scenario s názvem **Client_Server**.
3. Dále vybereme pro nás vhodné prostředí Office Network.
4. Rozměry prostředí nastavíme na 5x5 metrů.
5. Další krok přeskočíme tlačítkem Next a konfiguraci dokončíme pomocí Finish.

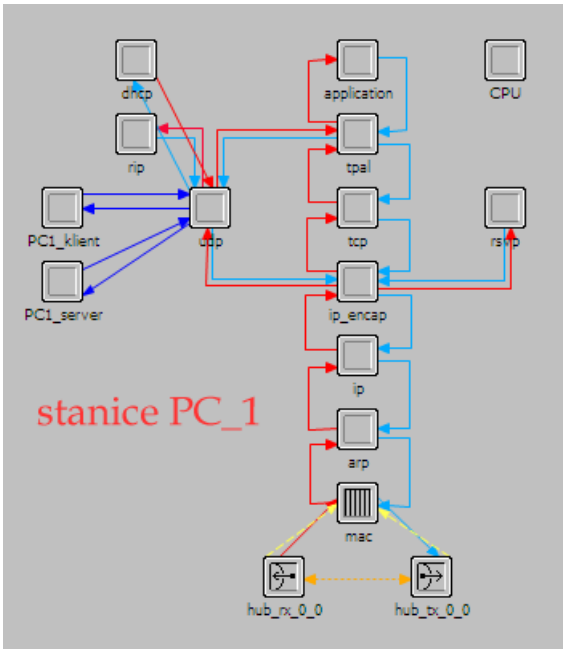
Vytvoření modelu

1. Automaticky se otevře Object pallete tree. Dvojklikem vybereme stanici **ethernet_wkstn**. Pro tento scénář nám budou stačit dvě stanice rozmístěné v Office Network.
2. Dále vybereme v záložce Link Models virtuální kabel **100BaseT** a tyto dvě stanice vzájemně propojíme.
3. Na jeden z uzlů klikneme pravým tlačítkem myší a vybereme možnost **Edit Attributes (Advanced)** Viz obr. 4.12.



Obr. 4.12: Model sítě pro generování paketů.

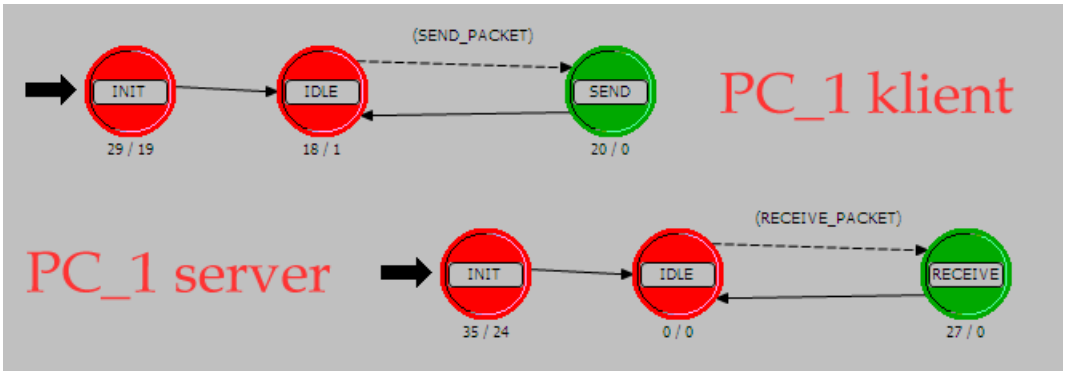
4. Dále v záložce **IP** ↦ **IP host parameters** ↦ **Interface Information** do položky Adress nastavíme IP adressu pro každý uzel **PC_1 klient** 192.168.1.15 a **PC_2 klient** 192.168.1.10.
5. Dvojklikem levého tlačítka myši na **PC_1(2)** se dostaneme o „úroveň“ níž, na model uzlů.
6. Zde vytvoříme processory s názvy klient a server, které budou tam a zpět spojeny s processorem **UDP** modrou šipkou **Create Packet Stream**. Dále **Objects** ↦ **Create Processor** viz obr.4.13.



Obr. 4.13: Upravený model uzlů stanice PC_1.

7. Pro přenos paketů nastavíme pravým talčítkem myši na processory klient a server a vybereme Edit Attributes. V položce process models zvolíme **server_packet_gen** pro server a **klient_packet_gen** tím se nám nastaví vnitřní struktura procesů podle předlohy viz obr. 4.14.
8. Proccesní modely dále upraveny pro duplexní přenos mezi dvěma stanicemi. Především se jedná o správné nastavení portů, kontrolu IP adress, intervaly přenosu paketů a jejich velikosti viz obr. [4.15, 4.16, 4.17, 4.18]. Hodnoty atribut byly převzaty z práce [5].
9. Pro výpis do konzole obousměrné komunikace je třeba nezapomenou přepsat proměnné procesorů server, aby se vzájemně nepřepisovali.

```
  /* Ziskani vlastniho ID */
my_obj_id_2 = op_id_self();
```



Obr. 4.14: Model procesů klient/server.

Attribute Name	Group	Type	Units	Default Value
Stop Time	UDP	double	seconds	Infinity
Start Time	UDP	double	seconds	10
Packet Interarrival Time	UDP	integer	seconds	30
Local port	UDP	integer		121
Remote port	UDP	integer		121
Server Address	UDP	string		192.168.1.15
Velikost dat	UDP	integer	bits	1024

Obr. 4.15: Attributy PC_1 klient.

Attribute Name	Group	Type	Units	Default Value
Local port	UDP	integer		120

Obr. 4.16: Attributy PC_1 server.

Attribute Name	Group	Type	Units	Default Value
Stop Time	UDP	double	seconds	Infinity
Start Time	UDP	double	seconds	15
Packet Interarrival Time	UDP	integer	seconds	30
Local port	UDP	integer		120
Remote port	UDP	integer		120
Server Address	UDP	string		192.168.1.10
Velikost dat	UDP	integer	bits	512

Obr. 4.17: Attributy PC_2 klient.

Attribute Name	Group	Type	Units	Default Value
Local port	UDP	integer		121

Obr. 4.18: Attributy PC_2 server.

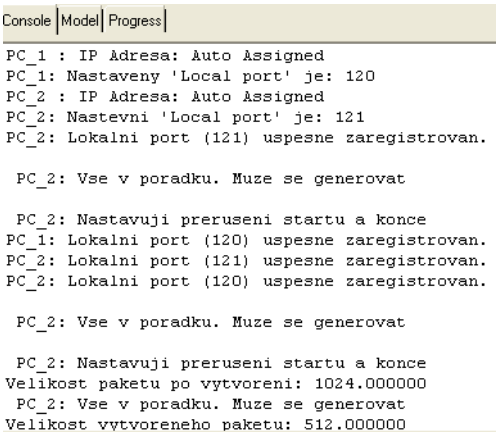
```
/* Ziskani ID rodice */
parent_obj_id_2 = op_topo_parent(my_obj_id);

/* Ziskani UDP ID */
udp_obj_id_2 = op_id_from_name (parent_obj_id, OPC_OBJTYPE_PROC
    , "udp");
```

10. Výpis přenosu UDP paketů získáme z proměnné `parent_obj_name`.

```
/* UDP packet */
printf ("%d. %s: UDP datagram dorazil do cile \n", counter++,
    parent_obj_name_2); // Debug Info
```

4.4.2 Simulace a výsledky

Konfiguraci **DES** spustíme zkratkou **ctrl+R**. **Duration** – čas simulace nastavíme na 10 minut. Protože výsledky simulace budou vypsány do konzole, je vhodné zaškrtnout v záložce **Execution** ↦ **OPNET Debugger** ↦ **Use OPNET Simulation Debugger**. Průběh simulace se nám do konzole vypíše po stisknutí tlačítka Continue. Zobrazení obsahu přeneseného paketu lze snadno zobrazit stejným postupem. Místo záložky console, ale vybereme záložku model, ve které zaškrtneme Show Animation. Ještě je třeba se dostat na správnou úroveň modelu. Stromovou strukturou v levé části okna se proklikáme přes Office Network a Client k UDP. Kliknutím na paket si zobrazíme obsah datové jednotky ve fázi, kterou požadujeme. Jak lze vidět z obr. 4.19 simulace zaměřená na generování a příjem datových jednotek se uskutečnila úspěšně.



Obr. 4.19: Výpis konzole pro simulaci zaměřené na generování paketů.

4.5 Simulace zaměřená na rozšíření původní datové jednotky protokolu DSR

Úkolem pro tuto simulaci bylo rozšířit původní datovou jednotku směrovacího protokolu DSR. Pro rozšíření byla zvolena režijní datová jednotka tzv. **Route Request** paket (RREQ). Protokol DSR využívá tohoto paketu při hledání trasy mezi uzly, tedy za využití mechaniky Route Discovery a šíření pomocí principu Flooding.

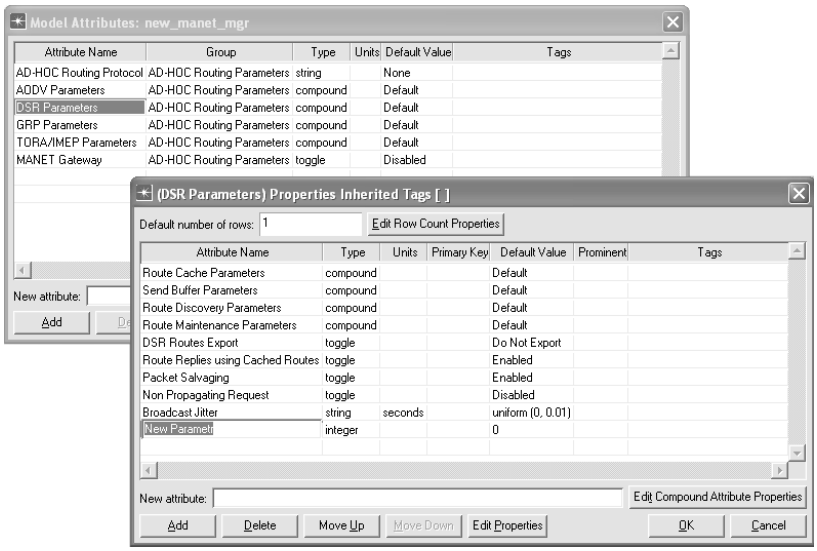
4.5.1 Úprava zdrojového kódu

V první části bude upraven samotný procesní model, tedy jeho bloky, a poté podle provedených změn budou přizpůsobeny vázané soubory protokolu DSR.

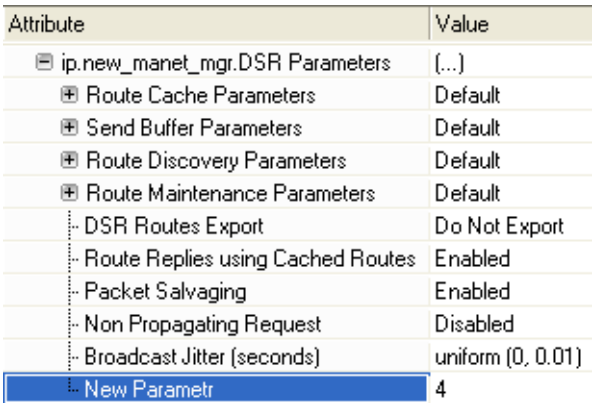
Blok stavových proměnných – SV (State Variable)

```
/* Nový parametr */
int \parametr;
```

Tímto se zadeklarujeme nová stavová proměnná. Která bude obsahovat data našeho nového atributu protokolu DSR. Nový atribut lze snadno vytvořit v grafickém prostředí OM tím, že se dostaneme na procesní model `manet_mgr` a na panelu nástrojů zvolíme **Interfaces** ↦ **Model Attributes**, myší vybereme **DSR Parameters** ↦ **Edit Properties**. Tento ostup je zobrazen na obr. 4.20. Obr. 4.21 ukazuje na nový attribut uvnitř DSR zvaný **New Parametr**.



Obr. 4.20: Přidání nového attributu protokolu DSR.



Obr. 4.21: Ukázka nově vytvořeného attributu New Parametr

Funkční Block – FB (Function Block)

Funkce `op_ima_obj_attr_get` vyčítá hodnotu z nastavených atributů. Jedná se o stavovou proměnnou `parametr` a atribut `New Parametr`.

```
/* Kontrola jestli je parametr nastaven */
op_ima_obj_attr_get (dsr_parms_child_id,"New Parametr", &parametr);
```

Aby bylo možné ze struktury RREQ paketu tisknout hodnoty do konzole, bylo třeba nejprve identifikovat v oblasti **Packet Arrival Functions** správou funkci, která pracuje uvnitř funkčního bloku, jako funkce vnořená dalších dvou funkcí, které jsou této nadřazeny. Níže je uvedena část úpravy zdrojového kódu této funkce, která nám zajistí tisk hodnot do konzole („pomocná“ funkce `inet_address` slouží k vyčítání proměnných, které obsahují například název uzlu nebo jeho IP adressu).

```
/* Hlavní funkce, jenž zpracovává příchozí paket */
dsr_rte_received_pkt_handle ();
->
/* Zde probíhá rozdělení paketů podle typu */
dsr_rte_app_pkt_arrival_handle ();
->
/* Funkce, jenž se stará o příchozí paket RREQ */
dsr_rte_received_route_request_process ();

inet_address_print (src_hop_addr_str, ip_dgram_fd_ptr->src_addr);
inet_address_to_hname (ip_dgram_fd_ptr->src_addr, src_node_name);
inet_address_print (dest_hop_addr_str, route_request_option_ptr->
   target_address);
inet_address_to_hname (route_request_option_ptr->target_address,
   dest_node_name);
```

```
printf  ("\n New Parametr %d from node %s destined to node %s",
    route_request_option_ptr->str_parametr, src_hop_addr_str,
    dest_hop_addr_str);
```

Další funkce, kterou je třeba upravit, se nachází v oblasti **Packet Creation Funcitons**. Naši stavovou proměnnou vložíme tímto do funkce, která se stará o vytváření parametrů uvnitř paketu RREQ. Funkce s názvem `dsr_rte_route_request_send();`.

```
  /* Funkce vytvoří TLV "možností" paketu RREQ */
dsr_tlv_ptr = dsr_pkt_support_route_request_tlv_create (
    route_request_identifier, dest_address, parametr);
```

V dalších krocích je třeba naše změny přizpůsobit souborům, které jsou k protokolu DSR vázány. Jejich umístění je popsáno v kapitole 3.2. Jedná se o dva hlavičkové soubory a jeden externí soubor s koncovkou `ex.c`. Uvnitř souboru `dsr_ptypes.h` rozšíříme vstupní hodnoty funkce o typ naší proměnné tedy integer.

```
/* dsr_packet_support vstupní proměnné "možností" RREQ paketu */
DsrT_Packet_Option* dsr_pkt_support_route_request_tlv_create (long
    int, InetT_Address, int);
```

Další hlavičkový soubor `dsr_pkt_support.h` obsahuje nadefinované struktury veškerých datových jednotek protokolu DSR. Zajímavá je pro nás především struktura `DsrT_Route_Request_Option`. Uvnitř struktury vytvoříme proměnnou, do které budeme zapisovat a dále z ní vyčítat hodnoty pro tisk do konzole.

```
/******** Struktury zpráv *********/
/* "Možnosti" RREQ paketu */
   typedef struct
  {
    long int       identification;
    int            str_speed;
    InetT_Address  target_address;
    List*          route_lptr;
  } DsrT_Route_Request_Option;
```

Úpravou souboru `dsr_pkt_support.ex.c` propojíme naši stavovou proměnnou a proměnnou uvnitř struktury, ze které máme zájem hodnoty vyčítat. První funkci `dsr_pkt_support_route_request_tlv_create`, která slouží jako podpora při vytváření paketu RREQ (rezervace paměti pro parametry uvnitř hlavičky paketu) a pracuje jen s přesně definovaným typem, délkou a hodnotou paketu (TLV – Type, Lenght, Value).

```
DsrT_Packet_Option*
```

```
dsr_pkt_support_route_request_tlv_create (long int request_id,
   InetT_Address dest_address, int speed)
  {
  DsrT_Route_Request_Option*      route_request_ptr = OPC_NIL;
  DsrT_Packet_Option*             dsr_tlv_ptr = OPC_NIL;
  int                             address_length;

  /** Funkce vytváří "možnosti" RREQ uvnitř DSR paketu **/
  FIN (dsr_pkt_support_route_request_tlv_create (<args>));

    address_length = (inet_address_family_get (&dest_address) ==
       InetC_Addr_Family_v4 ?
                                      IP_V4_ADDRESS_LENGTH :
                                         IP_V6_ADDRESS_LENGTH);
    route_request_ptr = dsr_pkt_support_route_request_mem_alloc ();
    route_request_ptr->identification = request_id;
    route_request_ptr->target_address = inet_address_copy (
       dest_address);

    /* Zápis nového parametru do struktury */
    route_request_ptr->str_parametr = parametr;

    dsr_tlv_ptr = dsr_pkt_support_option_mem_alloc ();
    dsr_tlv_ptr->option_type = DSRC_ROUTE_REQUEST;
    dsr_tlv_ptr->option_length = DSR_HEADER_OPTIONS +
       address_length;
    dsr_tlv_ptr->dsr_option_ptr = (void*) route_request_ptr;
    FRET (dsr_tlv_ptr);
    }
```

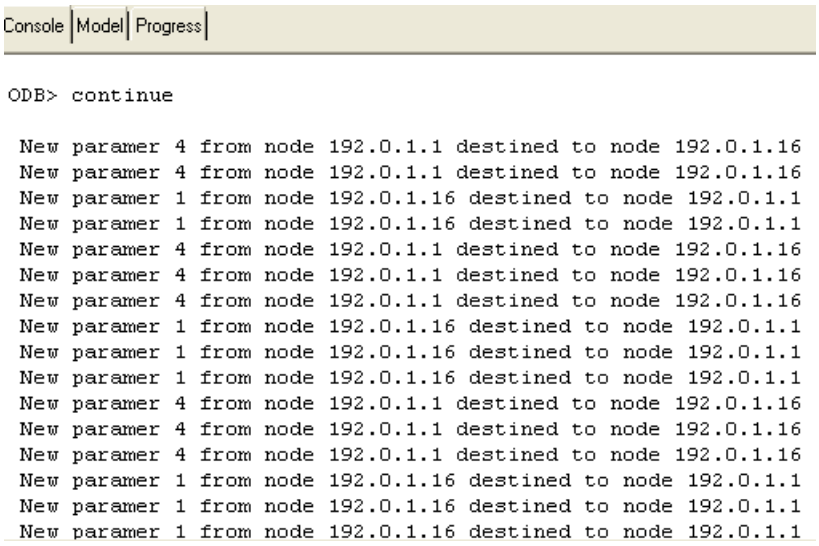
Druhá funkce `dsr_pkt_support_route_request_mem_copy` jen vytváří kopii parametru RREQ paketu pro další zpracování.

```
static DsrT_Packet_Option*
dsr_pkt_support_route_request_mem_copy (DsrT_Packet_Option*
   dsr_tlv_ptr)
  {
  DsrT_Route_Request_Option*    route_request_ptr = OPC_NIL;
  DsrT_Route_Request_Option*    copy_route_request_ptr = OPC_NIL;
  int                           num_hops, count;
  InetT_Address*                hop_address_ptr;
  InetT_Address*                copy_address_ptr;
  DsrT_Packet_Option*           copy_dsr_tlv_ptr = OPC_NIL;

  /** Funkce vytváří kopii RREQ "možností" **/
  FIN (dsr_pkt_support_route_request_mem_copy (<args>));
```

```
/* Funkce získá originální "možnosti" RREQ */
route_request_ptr = (DsrT_Route_Request_Option*) dsr_tlv_ptr->
    dsr_option_ptr;

/* Alokace paměti pro kopii "možností" RREQ */
copy_route_request_ptr = dsr_pkt_support_route_request_mem_alloc
    ();
copy_route_request_ptr->identification = route_request_ptr->
    identification;

/* Kopie zápisu nového parametru do struktury */
copy_route_request_ptr->str_parametr = route_request_ptr->
    str_parametr;

copy_route_request_ptr->target_address = inet_address_copy (
    route_request_ptr->target_address);
```

4.5.2 Vyhodnocení výsledků

Na jednoduchém modelu ze scénáře **DSR_function** byl nasimulován provoz dvou uzlů. V konzoli lze vidět hodnotu přenášené proměnné nastavené pro každou stanici (**node_1** má hodnotu parametru 1 a **node_16** hodnotu 4) uvnitř upraveného RREQ paketu, spolu s IP adresami uzlů, na kterých je nastaven provoz. Podařilo se uložit informaci dovnitř struktury datové jednotky RREQ a znovu ji vyčíst na koncové stanici. Výpis konzole této simulace lze vidět na obr. 4.22. Tento model je uložen na přiloženém CD pod názvem scénáře **parametrs_DSR_function**.



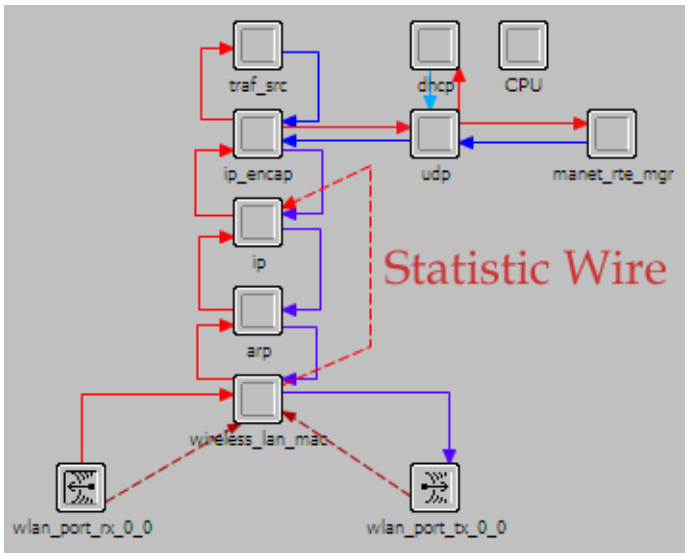
Obr. 4.22: Výpis konzole pro simulaci rozšiřující RREQ paket.

4.6 Simulace zaměřená na výpis přenosové rychlosti rozhraní MANET

Poslední simulace se jako předešlá zabývá rozšířením datové jednotky směrovacího protokolu DSR. Tentokrát nebude uvnitř struktury RREQ paketu přenášena hodnota parametru, ale přenosová rychlost rozhraní MANET. Tyto hodnoty se budou vypisovat jak do konzole, tak ukládat do souboru. Z výsledků by mělo být možné sledovat „zatížení“ stanic/sítě.

Pro rozšíření jsme zvolili režijní datovou jednotku tzv. Route Request paket (RREQ). Protokol DSR využívá tohoto paketu při hledání trasy mezi uzly, tedy za využití mechaniky Route Discovery a šíření pomocí principu Flooding. Jeho častá aktivita nám zaručí poměrně aktuální hodnoty, které máme zájem sledovat.

Uvnitř komunikace MANET stanic budou sledovány statistiky přenosové rychlosti mezi stanicemi. Budeme vyčítat nejvhodnější statistiku **Wireless Lan.Data Traffic Sent** (bits/sec). Protože se tato informace vytváří v uzlovém modelu uvnitř procesu `wireless_lan_mac` bude třeba pro vyčítání těchto hodnot vytvořit logické spojení k procesu `IP` zvané **Statistic Wire**. Na obr.4.23 je toto spojení vizuálně znázorněno. Nastavení tohoto spojení je vidět na obr. 4.24. Vytvořit spojení lze snadno pomocí záložky **Objects** ↦ **Create Statistic Wire** uvnitř uzlového modelu MANET stanice.



Obr. 4.23: Spojení pomocí Statistic Wire v modelu uzlů.

Attribute	Value
name	stat_2
submodule	NONE
src stat	Wireless Lan.Data Traffic Sent (bits/sec)
dest stat	instat [1]
intrpt method	scheduled
delay	0.0
rising edge trigger	disabled
falling edge trigger	disabled
repeated value trigger	disabled
zero crossing trigger	disabled
low threshold trigger	disabled
high threshold trigger	disabled
color	red

Obr. 4.24: Nastavení Statistic Wire.

4.6.1 Úprava zdrojového kódu

Nejprve provedeme úpravu v samotném procesním modelu, tedy v jeho blocích, a poté podle provedených změn přizpůsobíme vázané soubory protokolu DSR.

Blok stavových proměnných - SV (State Variable)

Zde proběhne deklarace nové stavové proměnné typu **double** s názvem `speed`. Tento typ je vybrán především kvůli kompatibilitě hodnot se statistikami, které se taky ukládají do proměnné typu **double**.

```
/* Nová proměnná */
Double  \speed;
```

Funkční blok - FB (Funtion Block)

Uvnitř FB pokračujeme v úpravě funkce `dsr_rte_received_pkt_handle();`jako tomu bylo v předešlé simulaci v oblasti **Packet Arrival Function**, nicméně rozšíření se bude tentokrát týkat nejenom výpisu hodnot (statistik) do konzole, ale i uložení získané hodnoty do souboru.

```
/* Nové deklarace proměnných */
FILE*   f;
char    název_stanice[128];

/* Nová proměnná cas do které se načítá aktuální čas simulace */
  cas = op_sim_time();

/* Tisk do konzole */
```

```
printf(" |  %-7.3f  ||     %-20.30s     ||    %-10.2f bitu/s   |  \
   n", cas, src_node_name, route_request_option_ptr->str_speed);

/* Tisk do souboru */
    strcpy(nazev_stanice, "C:/");
    strcat(nazev_stanice, dest_node_name);
    strcat(nazev_stanice,".txt");
    if ((f = fopen(nazev_stanice, "a")) == NULL)  {
    printf("Soubor se nepodarilo otevrit\n");     }
    fprintf(f, " |  %-7.3f  ||     %-20.30s     ||    %-10.2f bitu/
       s   |  \n", cas, src_node_name, route_request_option_ptr->
       str_speed);
    if (fclose(f) == EOF) {printf("Soubor se nepodarilo zavrit\n");
          }
```

Další funkce, kterou je třeba upravit, se nachází v oblasti **Packet Creation Funcitons**. Naši stavovou proměnnou vložíme tímto do funkce, která se stará o vytváření parametrů uvnitř paketu RREQ. Těsně před vložením se nám do naší proměnné vloží hodnota vyčtená funkcí `op_stat_local_read` na čtení statistik. Níže jsou popsany úpravy ve funkci `dsr_rteq_route_request_send();`.

```
/* Vložení hodnoty statistiky do nové proměnné */
speed = op_stat_local_read(1);
/* Funkce vytvoří TLV "možností" paketu RREQ */
dsr_tlv_ptr = dsr_pkt_support_route_request_tlv_create (
   route_request_identifier, dest_address, speed);
```

Abychom mohli takto snadno vyčítat statistiky je třeba zakomentovat část kódu uvnitř procesu `wireless_lan_mac`, který se stará o nulování statistik. Podrobně ho lze nalézt uvnitř **FB** jeho dceřiného procesu `wlan_mac` pod funkcí pro tvorbu statistik přenosové rychlosti `data_traffic_sent_handle_inbits` (je třeba dát při zakomentování pozor, protože se tam nulovaní objevuje víckrát).

```
/*tx_end_time = current_time + total_pk_size / operational_speed;
    op_stat_write_t (data_traffic_sent_handle_inbits, 0.0,
       tx_end_time);
    op_stat_write_t (data_traffic_sent_handle, 0.0, tx_end_time);*/
```

V dalších krocích je třeba naše změny přizpůsobit souborům, které jsou k protokolu DSR vázány. Jejich umístění je popsáno v kapitole 3.2. Jedná se o dva hlavičkové soubory a jeden externí soubor s koncovkou `ex.c`.

Uvnitř souboru `dsr_ptypes.h` rozšíříme vstupní hodnoty funkce o typ naší proměnné **double**.

```
/* dsr_packet_support vstupní proměnné "možností" RREQ paketu */
```

```
DsrT_Packet_Option* dsr_pkt_support_route_request_tlv_create (long
    int, InetT_Address, double);
```

Další hlavičkový soubor `dsr_pkt_support.h` obsahuje nadefinované struktury všech datových jednotek protokolu DSR. Pro tento scénář je třeba se zaměřit především na strukturu `DsrT_Route_Request_Option`. Uvnitř struktury vytvoříme proměnnou, do které se bude zapisovat a vyčítat hodnoty pro tisk do konzole a souboru.

```
/******** Struktury zpráv *********/
/* "Možnosti" RREQ paketu */
/typedef struct
  {
  long int       identification;
  double         str_speed;
  InetT_Address  target_address;
  List*          route_lptr;
  } DsrT_Route_Request_Option;
```

Úpravou souboru `dsr_pkt_support.ex.c` propojíme naši stavovou proměnnou a proměnnou uvnitř struktury, ze které máme zájem hodnoty vyčítat.

```
DsrT_Packet_Option*
dsr_pkt_support_route_request_tlv_create (long int request_id,
    InetT_Address dest_address, double speed)
  {
  DsrT_Route_Request_Option*      route_request_ptr = OPC_NIL;
  DsrT_Packet_Option*             dsr_tlv_ptr = OPC_NIL;
  int                             address_length;

  /** Funkce vytváří "možnosti" RREQ uvnitř DSR paketu **/
  FIN (dsr_pkt_support_route_request_tlv_create (<args>));

    address_length = (inet_address_family_get (&dest_address) ==
       InetC_Addr_Family_v4 ?
                                      IP_V4_ADDRESS_LENGTH :
                                         IP_V6_ADDRESS_LENGTH);
    route_request_ptr = dsr_pkt_support_route_request_mem_alloc ();
    route_request_ptr->identification = request_id;
    route_request_ptr->target_address = inet_address_copy (
       dest_address);

    /* Zápis proměnné speed do struktury */
    route_request_ptr->str_speed = speed;

    dsr_tlv_ptr = dsr_pkt_support_option_mem_alloc ();
    dsr_tlv_ptr->option_type = DSRC_ROUTE_REQUEST;
```

```
    dsr_tlv_ptr->option_length = DSR_HEADER_OPTIONS +
       address_length;
    dsr_tlv_ptr->dsr_option_ptr = (void*) route_request_ptr;

    FRET (dsr_tlv_ptr);
    }
```

Druhá funkce `dsr_pkt_support_route_request_mem_copy` jen vytváří kopii parametru RREQ paketu pro další zpracování.

```
static DsrT_Packet_Option*
dsr_pkt_support_route_request_mem_copy (DsrT_Packet_Option*
   dsr_tlv_ptr)
  {
  DsrT_Route_Request_Option*    route_request_ptr = OPC_NIL;
  DsrT_Route_Request_Option*    copy_route_request_ptr = OPC_NIL;
  int                           num_hops, count;
  InetT_Address*                hop_address_ptr;
  InetT_Address*                copy_address_ptr;
  DsrT_Packet_Option*           copy_dsr_tlv_ptr = OPC_NIL;

  /** Funkce vytváří kopii RREQ "možností" **/
  FIN (dsr_pkt_support_route_request_mem_copy (<args>));

  /* Funkce získá originální "možnosti" RREQ */
  route_request_ptr = (DsrT_Route_Request_Option*) dsr_tlv_ptr->
     dsr_option_ptr;

  /* Alokace paměti pro kopii "možností" RREQ */
  copy_route_request_ptr = dsr_pkt_support_route_request_mem_alloc
     ();
  copy_route_request_ptr->identification = route_request_ptr->
     identification;

  /* Kopie zápisu proměnné speed do struktury */
  copy_route_request_ptr->str_speed = route_request_ptr->str_speed;
    copy_route_request_ptr->target_address = inet_address_copy (
       route_request_ptr->target_address);
```

4.6.2 Vyhodnocení výsledků

Při komunikaci stanic pomocí protokolu DSR se na žádost (on demand) vysílají a zpracovávají RREQ pakety, které přenáší režijní data uvnitř struktury. Pomocí prostředí OM se úspěšně povedlo rozšíření této struktury a následně vyčítání hodnot přenosové rychlosti. Pro každou stanici, která přijímá data resp. je na ní směrován provoz je automaticky vytvořen soubor s názvem zmíněné stanice (pro tuto simulaci `Campus Network.mobile_node_24.txt` viz obr. 4.25). Tento soubor obsahuje čas vytvoření statistiky, název stanice posílající data a přenosová rychlost v okamžiku vytvoření RREQ paketu. Soubor se automaticky ukládá na disk **C:** dle zdrojového kódu.

Ačkoliv je zpráva RREQ šířena záplavovou metodou, po nalezení cíle zpráva zaniká a vytváří se nová datová jednotka RREP. Ta se vrátí do stanice, jenž přenos iniciovala a tím vzniká virtuální spojení. Dále se RREQ pakety posílají jen k režii sítě nebo v případě kolapsu a žádosti o nové spojení. Proto není možné touto konstrukcí Broadcastovat hodnotu statistiky přenosové rychlosti. Nicméně do jednoho uzlu může být vedeno více provozů z několika stanic a díky tomu je možné zaznamenat statistiku přenosové rychlosti pro všechny tato spojení do jednoho společného souboru.

```
| 479.249 ||  Campus Network.mobile_node_5  ||  3680.00    bits/s |
| 479.249 ||  Campus Network.mobile_node_5  ||  3680.00    bits/s |
| 479.249 ||  Campus Network.mobile_node_5  ||  3680.00    bits/s |
| 479.253 ||  Campus Network.mobile_node_5  ||  3680.00    bits/s |
| 479.253 ||  Campus Network.mobile_node_5  ||  3680.00    bits/s |
| 554.025 ||  Campus Network.mobile_node_5  ||  2816.00    bits/s |
| 554.531 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 554.531 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 555.528 ||  Campus Network.mobile_node_5  ||  2848.00    bits/s |
| 557.533 ||  Campus Network.mobile_node_5  ||  3744.00    bits/s |
| 569.868 ||  Campus Network.mobile_node_5  ||  2816.00    bits/s |
| 569.868 ||  Campus Network.mobile_node_5  ||  2816.00    bits/s |
| 573.367 ||  Campus Network.mobile_node_5  ||  3648.00    bits/s |
| 573.867 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 574.868 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 576.870 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 589.048 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
| 589.550 ||  Campus Network.mobile_node_5  ||  2592.00    bits/s |
```

Obr. 4.25: Výpis přenosové rychlosti do souboru.

Tato simulace byla provedena na kopii scénáře **DSR_mobility**. Po mnoha testech se ukázala jako nejvhodnější díky velkému počtu stanic a jejich pohyblivosti. Tyto parametry zvyšují pravděpodobnost vyslání RREQ paketu a tím i zápis statistiky do souboru.

5 ZÁVĚR

Tato bakalářská práce odhaluje možnosti síťové komunikace směrovacího protokolu DSR v bezdrátovém prostředí mezi rovnocennými prvky. První část práce je soustředěna na teorii MANET sítí včetně rozdělení a popisu několika nejpoužívanějších protokolů v OM. V další části proběhl teoretický rozbor směrovacího protokolu DSR. Práce osvětluje jeho základní mechanismy směrování a údržby. V prostředí programu OPNET je zmapován procesní model DSR včetně pohledu na jeho umístění ve struktuře nadřazených a dceřiných procesů. Velkým přínosem pro praktickou část jsou detailně popsané nastavitelné parametry protokolu DSR v OM.

Praktická část tvořená čistě v prostředí OM začíná simulací, jenž ověřuje správnou funkci protokolu DSR s defaultními parametry. Výsledky ukazují, že na poměrně ideálním modelu protokol bezchybně směruje a přenáší data s minimální odezvou. Druhá simulace je zátěžovým testem protokolu DSR a zároveň ukázkou několika nejpodstatnějších parametrů. Testovalo se zde chování sítě ve třech různých rychlostech náhodného pohybu uzlů. Výsledky ukazují protokol DSR jako kvalitnější v sítích s menší pohyblivostí, které nejsou příliš husté. Další simulace navazuje na simulaci předešlou, tentokrát bylo třeba přizpůsobit nastavitelné parametry protokolu DSR tak, abychom snížily zátěž sítě a tím ušetřili prostředky. Výsledek této simulace můžeme považovat za úspěch, protože se pro zvolenou síť podařilo dlouhodobě snížit režijní náklady.

Ve čtvrté simulaci byla vyzkoušena síť s duplexně spojenými stanicemi pro generování paketů a byly tak položeny základy pro práci na rozšíření původní datové jednotky směrovacího protokolu DSR v páté simulaci. Zde byl vytvořen nový parametr, jenž se úspěšně povedlo přenášet mezi stanicemi prostřednictvím rozšířeného RREQ paketu o novou proměnnou ve struktuře. Výpis přenášených parametrů je přehledně tisknut do konzole. Poslední simulace se podobně jako předešlý scénář soustřeďuje na rozšíření RREQ paketu, nicméně tentokrát se do nové proměnné struktury této datové jednotky zapsala hodnota (statistika) přenosové rychlosti rozhraní MANET. Tato hodnota je tedy úspěšně vypisována jak do konzole, tak do souboru a proto můžeme považovat poslední cíl tohoto dokumentu za splněný.

LITERATURA

[1] FEENEY, L. *An Energy-comsuption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks*. Swedish Institute of Computer Science, Kista, Sweden, 2001. 27s. Dostupné z URL: <http://www.sics.se/ lmfeeney/publications/Files/monet01energy.pdf>.

[2] ILYAS, Mohammad. *THE HANDBOOK OF AD HOC WIRELESS NETWORKS*. Florida Atlantic. University Boca Raton: CRC PRESS, 2003. 624 s. ISBN 0-8493-1332-5.

[3] JOHNSON D., HU Y., MALTZ D. *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*. IETF Network Working Group, RFC 4728. February 2007. Dostupné z URL: <http://www.ietf.org/rfc/rfc4728.txt>.

[4] KUSOMANEN, P. *Classification of Ad Hoc Routing Protocols*. Naval Academy, Finish Defence Forces, Helsinky, Finland, 13s. Dostupné z URL: <http://www.netlab.tkk.fi/opetus/s38030/k02/Papers/12-Petteri.pdf>.

[5] MIKULICA, V. *Generování datových jednotek v prostředí OPNET Modeler*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 70 s.

[6] MOLNAR, K., ZEMAN, O., SKOŘEPA, M., *Moderní sítové technologie, Laboratorní cvičení*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ustav telekomunikaci, 2007. 101s.

[7] NOVOTNY, V. *Mobilní směrovací protokoly s podporou IPV6 (MANET)*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ustav telekomunikaci, 2007. 89s.

[8] OPNET Technologies. *OPNET Modeler Product Documentation Release 16.0.*, OPNET Technologies Inc., 2010.

[9] SARKAR, S. *Ad Hoc Mobile Wireless Networks: Principles, Protocols and Applications*. Boca Raton: Auerbach Publications, 2008. ISBN 978-1-4200-6221-2.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ACK Acknowledgment

AODV Ad Hoc On-demand Distance Vector

CPU Central Processing Unit

DAG Direct Acyclic Graph

DSDV Destination Sequenced Distance Vector

DSR Dynamic Source Routing

FB Function Block

GPS Global Positioning System

GRP Geographic Routing Protocol

IP Internet Protocol

MANET Mobile Ad Hoc Netwtork

MPR Multi Point Relay

OLSR Optimized Link State Routing Protocol

OM OPNET Modeler

RFC Request for Comments

RREP Route Reply

RREQ Route Request

SV State Variable

TORA Temporally Ordered Routing Algorithm

TLV Type, Lenght, Value

TC Topology Control

TTL Time to Live

UDP User Datagram Protocol

SEZNAM PŘÍLOH

A OBSAH DVD

- Bakalářská práce v elektronické podobě:
  - **Bachelor_thesis.pdf**
- Praktická část bakalářské práce v programu OPNET Modeler:
  - **Bc_projectDSR_ver16archiv.opcfa**
- Modely vázané se simulací, která je zaměřena na generování paketů [4.4]:
  - **Server_Client.rar**