VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMNUICATION
DEPARTMENT OF TELECOMMUNICATIONS

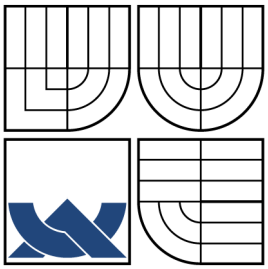
NÁVRH A REALIZACE PROTOKOLU V PROSTŘEDÍ NS2

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

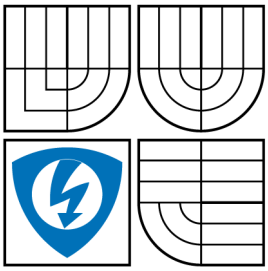
AUTOR PRÁCE
AUTHOR

Bc. DANIEL ZVOLENSKÝ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMNUICATION
DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH A REALIZACE PROTOKOLU V PROSTŘEDÍ NS2

DESIGN AND IMPLEMENTATION OF PROTOCOLS IN THE NS2

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DANIEL ZVOLENSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN KOUTNÝ

BRNO 2010

Abstrakt

Cílem této práce je podrobně se seznámit se simulačním prostředím Network Simulator 2, nastudovat problematiku senzorových sítí a jejich podporu v tomto simulátoru a dle výběru implementovat příslušný protokol a realizovat jeho funkčnost v praktické simulaci.

Abstract

The aim of this work is to become familiar with simulation environment Network Simulator 2, study the problems of sensor networks and their support in the simulator and implement the selected protocol and verify its funcitonality in a practical simulation.

Klíčová slova

Network Simulator 2, senzorové sítě, bezdrátové sítě, multicast, IP Multicast, WSN, ZigBee, MAODV, AODV, OTcl, C++, NS2

Keywords

Network Simulator 2, senzor network, wireless network, multicast, IP Multicast, WSN, ZigBee, MAODV, AODV, OTcl, C++, NS2

Citace

Daniel Zvolenský: Návrh a realizace protokolu v prostředí NS2, diplomová práce, Brno, FEKT VUT v Brně, 2010

Návrh a realizace protokolu v prostředí NS2

Prohlášení

Prohlašuji, že svou diplomovou práci na téma "Návrh a realizace protokolu v prostředí NS2"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 diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostní a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledů vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

.....
Daniel Zvolenský
21. května 2010

Poděkování

Chci poděkovat Ing. Martinovi Koutnému za užitečnou metodickou pomoc a cenné rady při spracovávaní této diplomové práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Bezdrôtová komunikácia sa v poslednom desaťročí začala radikálne presadzovať takmer vo všetkých oblastiach ľudskej činnosti, teda i v oblasti automatizácie. Rozvoj vysoko–rýchlostných bezdrôtových komunikácií a bezdrôtových sietí, ktorý nastal koncom deväťdesiatych rokov 20. storočia, bol umožnený rozvojom techniky, ktorá dovolila využiť UHF (Ultra High Frequency) a mikrovlnnú oblasť, teda frekvencie nad približne 1 *GHz*.

Z počiatku boli bezdrôtové komunikácie využívané najmä pre prenos hlasu. S nástupom moderných digitálnych prostriedkov nadobúdal na význame prenos dát, pretože významnými aplikáciami sa stávali ako bezdrôtový prístup k lokálnym počítačovým sieťam, tak i dátová komunikácia medzi autonómnymi zariadeniami. Najvýznamnejšie technológie, ktoré mali a majú význam pre automatizáciu sú GSM, GPRS, Wi–Fi, HSCSD, UMTS, Bluetooth, ZigBee a iné. Väčšina týchto štandardov nebola pôvodne navrhnutá pre použitie v priemysle, avšak postupom času boli priebežne inovované a rozširované tak, aby čo najlepšie vyhovovali aplikáciám, ktoré kladú zvýšené nároky na bezpečnosť, spoľahlivosť a prenos dát v reálnom čase [11].

Mobilné zariadenia ako také majú obmedzené výpočtové schopnosti. Dynamické vyhľadávanie ciest, udržiavanie smerovacích tabuliek, zaistenie odolnosti proti slučkám a iné, sú výpočtovo náročné operácie (najmä ak je sieť naozaj veľká). Preto správna voľba smerovacieho protokolu je veľmi dôležitá. Táto práca sa zaoberá návrhom a implementáciou smerovacieho protokolu MAODV. Jedná sa o multicast smerovací protokol, ktorý umožňuje dynamické, samočinné *multihop* smerovanie medzi zúčastnenými mobilnými uzlami. Umožňuje vytvárať multicast stromy, čím sa zjednodušuje smerovanie v sieti, a tým šetrí výpočtové prostriedky bezdrôtových uzlov.

Protokol MAODV je založený na smerovacom protokole AODV a ako taký s ním zdieľa mnoho podobností. V tejto diplomovej práci bude definovaný vlastný návrh a riešenie implementácie protokolu MAODV (a tým pádom aj AODV) do simulačného prostredia Network Simulator 2. Simulačné prostredie NS2 umožňuje simulovať rôzne sieťové protokoly, topológie a umožňuje i vlastnú implementáciu protokolov a agentov.

Kapitola 2

Bezdrôtové siete

Bezdrôtové siete sú založené na štandarde IEEE 802.11. Základná bezdrôtová sieť je vytvorená prepojením niekoľko prístupových bodov, ktoré medzi sebou komunikujú za pomoci rádiových vĺn (frekvenčné pásmo 2.4 GHz - 5 GHz). Prístupové body (mobilné uzly) fungujú ako smerovače, ktoré zasielajú správy najlepšou možnou cestou (*multihop*[1]). Pri komunikácii medzi dvoma uzlami je potrebné, aby tieto uzly boli vo svojom dosahu. Ak tomu tak nie je, potrebujú pre smerovanie paketov pomoc medziľahlých uzlov. Spojenie medzi zariadeniami sa vytvára až v momente jeho potreby. Pri odosielaní správy, medzi zariadeniami v sieti, vybrané uzle vytvoria komunikačný kanál, ktorý sa po prenose dát opäť rozpojí. Takáto sieť je plne distribuovaná a dokáže pracovať na akomkoľvek mieste bez pomoci akejkoľvek infraštruktúry [7, 8, 9].

Siete mesh pracujú na princípe *peer-to-peer*, teda všetky zariadenia v sieti disponujú rovnakou sadou služieb. Táto topológia prináša nasledovné výhody [7]:

- *redundancia spojenia* – spojenie v sieti *mesh* sa ustanoví len v prípade potreby a iba nevyhnutne potrebnú dobu, čo podstatne menej zaťažuje komunikačné pásmo,
- *nízke náklady na údržbu* – minimálne požiadavky na nastavovanie uzlov siete, pretože komunikácia je predovšetkým obsluhovaná transportným protokolom,
- *zvýšenie dosahu siete* – najmä vďaka väčšiemu počtu adaptérov využiteľných ku komunikácii (všetky uzle siete môžu byť využívané okrem svojej základnej funkcie k prenosu dát z iných zariadení).

Obecne, uzly v sieti sú mobilné a to robí topológiou distribuovanej siete časovo premennú. Dynamická povaha topológie siete je veľkou výzvou pre návrh *ad-hoc* sietí. Každý uzol siete je zvyčajne napájaný energeticky obmedzeným zdrojom (dobíjacie batérie). Spotrebu energie uzlov môžeme rozdeliť na tri časti [8]:

- spotreba energie pri spracovávaní dát uzlom,
- spotreba energie pri vysielaní vlastných informácií,
- spotreba energie pri použití uzla ako smerovača, to je zasielanie informácií ďalším uzlom v sieti.

[1] *multihop* – je sieť zložená z niekoľko segmentov, kde je funkčnosť siete podmienená smerovaním medzi jednotlivými bunkami

Z uvedeného vyplýva, že spotreba energie je kritickým bodom pri návrhu *ad-hoc*[2] sietí.

Mobilné zariadenia majú zvyčajne obmedzenú pamäťovú kapacitu a nízke výpočtové schopnosti. Sú závislé na ostatných uzloch a zdrojoch pre prístup dát a spracovanie informácií. Preto musí byť pre *ad-hoc* sieť zabezpečená spoľahlivá sieťová topológia a musia byť zaistené bezpečné smerovacie protokoly [8].

2.1 Výhody použitia topológie mesh

Hlavným prínosom použitia topológie *mesh* je redundancia spojenia, ktorá v tomto prípade vychádza priamo z podstaty topológie - je daná iba hustotou uzlov v sieti. Topológia *mesh* nie je obmedzujúca v štruktúre sieti, a preto zjednodušuje automatické zostavovanie spojenia a obnovenie siete po poruche. Spojenie medzi dvomi bodmi v topológii úplnej *mesh* možno zostaviť vždy, keď sú tieto body schopné komunikácie, a u topológie čiastočnej *mesh* je možné zostaviť takmer vždy. Existencia alternatívnych ciest taktiež umožňuje rovnomernejšie rozdeliť záťaž predávaných dát medzi jednotlivé uzly siete (*traffic balancing*). Pre bezdrôtovú komunikáciu je najviac významné, že uzly môžu komunikovať so svojimi blízkymi susedmi priamo, teda na kratšie vzdialenosti.

Iné topológie navrhované pre bezdrôtové siete prinášajú rôzne špecifické výhody, najmä zjednodušené smerovanie (topológia typu strom) alebo definovanú dobu odozvy (kruhová topológia), zvýšenie priepustnosti, atď.

V sieťach *mesh* sa za spoľahlivosť a redundancia spojenia platí náročným smerovaním. Náročné je predovšetkým dynamické vyhľadávanie ciest, udržovanie smerovacích tabuliek, zaistenie odolnosti proti vyhľadaniu ciest, poprípade udržovanie smerovacích tabuliek, zaistenie odolnosti proti smerovacím slučkám atď. Azda najrozšírenejším protokolom pre dynamické bezdrôtové siete je smerovací protokol AODV (Ad-hoc On-Demand Distance Vector - RFC 3561), vyvinutý na univerzite v kalifornskej Santa Barbare. Je zameraný na použitie s IP protokolom, ale jeho princíp je využitý pre smerovanie napríklad i u komunikačného štandardu *ZigBee*.

Topológia *mesh* so sebou nesie okrem spomenutého zložitejšieho smerovania i ďalšie nevýhody, ktoré súvisia s veľkým množstvom spojení, napríklad veľké nároky na pamäť zariadenia (je nutné v nej uložiť rozsiahle smerovacie tabuľky), zvýšenú spotrebu, možnosť interferencie a kolísajúcu priepustnosť [6].

2.2 Využitie bezdrôtových sietí

Realizácia siete *mesh* v praxi je značne závislá na použitej metóde komunikácie, technických prostriedkoch a požiadavkách aplikácie. Za prvé ide o siete tvorené v podstate rovnocennými a spravidla jednoduchými zariadeniami. To môžu byť napr. siete senzorov pre monitorovanie a zber dát v automatizácii. Zber dát je mimochodom hlavnou oblasťou využitia sietí *mesh* a často je jediné vhodné riešenie - napr. pre monitorovanie dát v oblastiach, kde sa pevné spoje inštalujú obtiažne [6].

[2] *ad-hoc* – znamená náhodne alebo improvizovane; v oblasti bezdrôtových sietí sa tak označujú siete, kde jednotliví účastníci nevyžadujú žiadnu predom vytvorenú infraštruktúru (pre komunikáciu medzi sebou) a sami zaisťujú funkcie potrebné pre riadenie siete

2.3 Systém ZigBee

ZigBee, u IEEE je vedený pod označením 802.15.4 WPAN Low Rate. Ide o špecifikáciu určenú pre vysoko úrovňové komunikačné protokoly, ktoré sú orientované na malé zariadenia s nízkou spotrebou a obsluhované na malé vzdialenosti do 75 *m*. Štandard *ZigBee* bol vyvíjaný s cieľom, aby zariadenia vydržali v prevádzke s bežnými tužkovými batériami i niekoľko mesiacov až rokov a aby bol nový štandard jednoduchší a lacnejší, než niektoré konkurenčné siete WPAN (napr. *Bluetooth*). Protokol je primárne určený pre využitie v priemysle, najmä pre senzorové siete.

Nízka spotreba je dosahovaná predovšetkým využitím metódy priamo bezprostredného spektra (DSSS), na rozdiel od rozprestretého spektra s frekvenčnými preskokmi (FHSS) pri štandarde *Bluetooth*. Zariadenia na viac môžu prejsť do stavu hibernácie, kedy modul iba načúva a neúčastní sa prenosu signálu. Tým je možné znížiť spotrebu len na 1 až 2 μA (zariadenia *Bluetooth* odoberajú asi 100 μA) [7].

2.3.1 Sieťová topológia ZigBee

Zapojené zariadenia sa delia do troch kategórii:

- *sieťový koordinátor* – najzložitejšie zariadenie, udržuje informácie o celej sieti,
- *plne funkčné zariadenie FFD (Full Functionality Device)* – podporuje kompletné funkcie podľa štandardu IEEE 802.15.4 a veľmi dobre môže pracovať ako smerovač alebo v bode spojenia s inou sieťou,
- *zariadenie základných funkcií RFD (Reduced Functionality Device)* – obsahuje iba obmedzenú sadu funkcii pre použitie s koncovými zariadeniami.

Jednotlivé zariadenia sú prepojené buď priamo cez koordinátora alebo cez zariadenie FFD. O tom, do akej kategórie bude dané zariadenie (nazývané *mote*) v sieti zaradené, prevažne rozhoduje software implementovaný na čipe. Väčšina firiem distribuujúcich čipy *ZigBee* poskytuje k týmto súčiastkam i jednoduché operačné systémy, poprípade súbory funkcii umožňujúcich bezdrôtový prenos získaných dát.

Doba potrebná pre pridanie nového člena do siete je 30 *ms*, zariadenie je schopné sa prebudiť z hibernácie do 15 *ms* a doba prístupu k aktívnemu zariadeniu je taktiež 15 *ms*. Sieť umožňuje prepojenie až 65 536 zariadení. I takto rozsiahlu sieť spravuje jediný koordinátor, ku ktorému sú zariadenia pripojené v pozícii FFD alebo RFD. Celá sieť môže pracovať v niekoľkých topologických režimoch podľa typu zariadenia, ktoré sa zúčastní prenosu dát (hviezda, zhlukové stromy, zhlukové hviezdy). Protokol *ZigBee* má tieto parametre [7]:

- v pásme 2 450 *MHz* (pásmo ISM) je možné bezdrôtovo prenášať dáta rýchlosťou 250 *kbps* a v pásme 868 *MHz* (iba pre Európu) rýchlosťou 20 *kbps*, v pásme 915 *MHz* (ISM, len pre Ameriku) rýchlosťou 40 *kbps*,
- šestnásť kanálov v pásme 2 450 *MHz*, desať kanálov v pásme 915 *MHz* a jeden kanál v pásme 868 *MHz*,
- dosah 30 *m* v uzatvorených budovách, 100 *m* na voľnom priestranstve
- pre zabezpečenie je možné voliteľne použiť kódovanie komunikácie za pomoci AES 128 (Advanced Encryption Standard).

Kapitola 3

IP Multicast

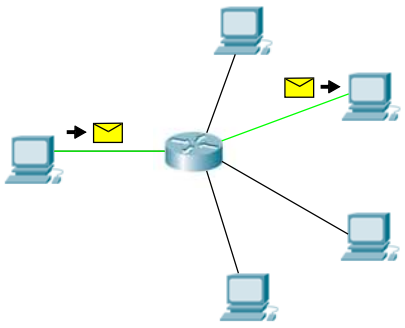
Pri písaní tejto kapitoly bolo čerpané zo zdroja [10].

IP multicast predstavuje efektívny spôsob využitia prenosovej šírky pásma pri odosielaní dát viacerým užívateľom. Aplikácie, ktoré využívajú *IP multicast* sú videokonferencie, firemná komunikácia, diaľkové štúdium, atď. Pri obyčajnom unicast odosielaní správ k viacero príjemcom dochádza k veľkej spotrebe šírky pásma. Naproti tomu sú multicast pakety rozposielané do siete v bode kde sa cesty rozdeľujú. To má za následok efektívnejší spôsob využitia sieťových prostriedkov.

IP multicast je teda ideálny pre skupiny užívateľov prijímajúcich rovnaký obsah dát. Skupinu multicast môže zakladať ktorýkoľvek užívateľ a kdekoľvek (či už v internej sieti alebo mimo nej). A taktiež sa môže ku skupine pripojiť ktorýkoľvek užívateľ z ktoréhokoľvek miesta (či už z Internetu alebo privátnej siete).

3.1 Porovnanie unicast, broadcast a multicast

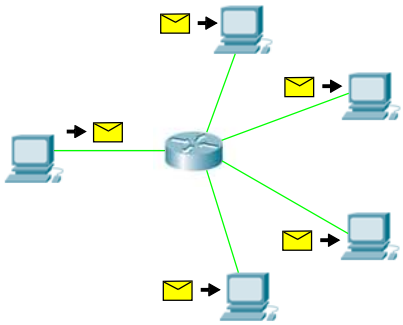
- **Unicast** – Aplikácie rozposielajú kópiu každého paketu zo zdroja správ smerom k užívateľovi. Ak chcú viacerí užívatelia prijímať rovnaký obsah dát, sú pakety rozposielané ku každému užívateľovi súčasne, čo má za následok zahltenie systémových zdrojov. To znamená, že ak 30 užívateľov žiada o príjem rovnakého obsahu dát, je nutné súčasne preposlať 30 takýchto kópii dát.



Obrázek 3.1: Typ prenosu unicas.

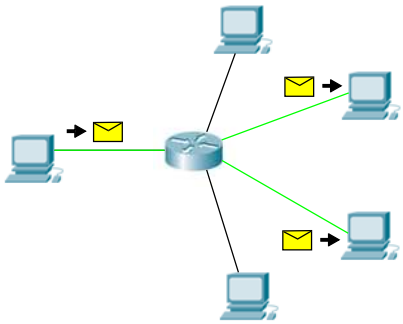
- **Broadcast** – Aplikácie pošlú jednu kópiu každého paketu na broadcast adresu. Týmto

spôsobom je informácia preposlaná každému užívateľovi v sieti. Takto je zachovaná priepustnosť siete, pretože rovnaký obsah dát je smerovaný ku všetkým užívateľom.



Obrázek 3.2: Typ prenosu broadcast.

- **Multicast** – Aplikácie pošlú jednu kópiu paketu a smerujú ju ku skupine vybraných príjemcov. Multicast je závislý od siete, ktorá rozosiela pakety do siete a smerom k užívateľom. Týmto spôsobom je kontrolovaná vyťaženosť siete a je redukované množstvo spracovaní uskutočnené jednotlivými užívateľmi. To znamená, že ak 30 užívateľov žiada o príjem rovnakého obsahu dát (a sú členmi multicast skupiny), prepošle sa jedna kópia dát, ktorá sa potom kopíruje ku všetkým užívateľom.



Obrázek 3.3: Typ prenosu multicast.

Multicast má množstvo výhod oproti unicastu a broadcastu. Zatiaľ čo unicast predstavuje efektívny spôsob ako doručiť obsah k jednému užívateľovi, pri odosielaní rovnakého obsahu dát viacerým užívateľom nastáva problém zahltenia šírky prenosového pásma. Broadcast, na druhej strane predstavuje dobrý spôsob šetrenia sieťových zdrojov (jediná kópia dát je preposlaná každému užívateľovi v sieti). Avšak tento spôsob prenosu vyvoláva otázky spotreby šírky pásma, pretože je nepraktický, ak iba hŕstka užívateľov potrebuje vidieť danú informáciu.

IP multicasting rieši problém "*bottleneck*[1]", kedy sú dáta prevedené od jedného odosielateľa k viacerým príjemcom. Odoslaním jedinej kópie dát do siete a povolením sieti replikovať pakety k príjemcom je zachovaná šírka pásma ako pre odosielateľa, tak i pre príjemcu.

[1] *bottleneck* – jav, pri ktorom výkon alebo kapacita celého systému je obmedzená počtom zdrojov alebo komponentov

3.2 IP Multicast adresy

Multicast adresa určuje ľubovoľnú skupinu IP užívateľov, ktorý sa chcú pripojiť do skupiny multicast a chcú prijímať dáta posielané do tejto skupiny. Ako náhle sa klient rozhodne byť členom skupiny, je obsah preposielaný tejto skupine odosielaný na jednu konkrétnu IP adresu. Takže dáta odoslané na túto adresu sú preposlané každému členovi multicast skupiny.

Organizácia IANA (*Internet Assigned Numbers Authority*) spravuje rozdelenie IP adries. Pre aplikácie *IP multicast* vymedzila adresný priestor triedy IP adries D (224.0.0.0 – 239.255.255.255). IP adresy adresného priestoru *triedy D* sú ďalej rozdelené na špeciálne potreby. Nasledujúci text zobrazuje dané rozdelenie adresného priestoru adries *triedy D*:

- 224.0.0.0 – 224.0.0.255 – iba pre použitie sieťových protokolov na lokálnom segmente siete. Pakety s touto adresou by nemali byť preposielané smerovačmi, tzn. že pakety ostávajú v LAN segmente a sú vždy prenášané s hodnotou TTL - 1,
- 224.0.1.0 – 238.255.255.255 – prezývané *adresy z globálnym rozsahom*, sú tieto adresy použité pre preposielanie dát medzi zdrojom dát a naprieč Internetom,
- 239.0.0.0 – 239.255.255.255 – prezývané *adresy s obmedzeným rozsahom* alebo *adresy s administratívnym rozsahom*, sú viazané na organizáciu. Smerovače majú prednastavené filtre, ktoré predchádzajú preposlaniu dát s IP adresami tohto rozsahu mimo internú sieť. V rámci organizácie, môže byť tento adresný priestor rozdelený na menšie časti (tzv. podsiete), čo umožňuje opakované použitie adries pre menšie domény.

Kapitola 4

MAODV

Pri písaní tejto kapitoly bolo čerpané zo zdroja [2].

MAODV (*Multicast Adhoc On-Demand Distance Vector*) protokol umožňuje dynamické, samočinné, *multihop* smerovanie medzi zúčastnenými mobilnými uzlami, ktoré chcú vstúpiť alebo zúčastniť sa multicast skupiny v rámci *ad-hoc* siete. Členstvo v týchto multicast skupinách je možné meniť za behu siete. MAODV umožňuje mobilným uzlom vytvoriť multicast strom spájajúci členov multicast skupiny. Mobilné uzly sú schopné rýchlo reagovať na zmeny topológie multicast siete. V prípade rozdelenia siete na menšie časti, sú jednotlivé multicast stromy stanovené samostatne a stromy pre rovnakú multicast skupinu sa v prípade zlúčenia pripoja veľmi rýchlo.

Jedným z charakteristických znakov protokolu MAODV je použitie *sekvenčných čísel* pre multicast skupiny. Každá multicast skupina má svoje *sekvenčné číslo*, ktoré je inicializované *lídrom multicast skupiny* a je pravidelne inkrementované. Pomocou týchto *sekvenčných čísiel* je zabezpečené, že cesty nájdené k multicast skupinám sú vždy tie najaktuálnejšie. Z daných možností voľby medzi dvomi cestami k multicast stromu, si žiadajúci uzol vyberie vždy len tú s najvyšším *sekvenčným číslom*.

MAODV je multicast protokol spojený so smerovacím protokolom AODV a ako taký, zdieľa s ním mnoho podobností. Napríklad typy správ "*Route Request*"a "*Route Reply*"sú práve založené na paketoch protokolu AODV. Podobne ako aj veľa parametrov konfigurácii používaných MAODV je definovaných AODV protokolom.

4.1 Princíp činnosti protokolu MAODV

RREQ (*Route Request*), RREP (*Route Replies*), MACT (*Multicast Activation*) a GRPH (*Group Hello*) sú správy použité multicast protokolom MAODV. Za pomoci týchto správ môžu uzly siete medzi sebou komunikovať pre potreby multicast spojenia.

Pokým užívatelia multicast skupiny zostávajú spojený (v multicast strome), protokol MAODV nehrá žiadnu rolu. Ak však sa chce uzol pripojiť k multicast skupine alebo chce nájsť cestu k multicast skupine, použije uzol vysielanie správ RREQ pre objavenie cesty k danej skupine. Cesta je určená vtedy, keď RREQ dosiahne uzol, ktorý je už členom multicast stromu a záznam uzla o sekvenčnom čísle multicast skupiny je prinajmenšom rovnako veľký ako číslo obsiahnuté v správe RREQ. Na žiadosti správ RREQ, ktoré nie sú charakteru "pripojiť sa"(*join requests*) môže odpovedať každý uzol s aktuálnou cestou k multicast stromu. Aktuálna cesta je definovaná ako platná cesta v multicast tabuľke,

ktorá obsahuje sekvenčné číslo pre multicast skupinu prinajmenšom rovnako veľké ako číslo obsiahnuté v správe RREQ. Cesta k multicast stromu je sprístupnená unicast správou RREP späť ku zdroju správ RREQ. Vzhľadom k tomu, každý uzol prijímajúci žiadosť si pamätá cestu späť ku zdroju žiadostí, správa RREP môže byť poslaná späť ku zdroju od ľubovoľného uzla. Ako náhle zdrojový uzol obdrží správy RREP, vyberie z nich tú najlepšiu cestu ku multicast stromu. Touto cestou potom vyšle správu MACT, ktorou sa daná trasa aktivuje.

Uzly v sieti monitorujú svoje linky smerom ku multicast stromu. Ak sa spojenie na niektorej z liniek preruší, daná vetva sa okamžite inicializuje ako "spadnutá"a uzle pracujú na jej oprave použitím správ RREQ/RREP/MACT.

Líder skupiny multicast je priradený ku každej skupine multicast. Základnou úlohou tohto uzla je inicializácia a udržiavanie sekvenčného čísla multicast skupiny. Správa "*Group Hello*"je pravidelne vysielaná po sieti lídrom multicast skupiny. Táto správa obsahuje sekvenčné číslo multicast skupiny a zodpovedajúcu IP adresu lídra skupiny. Správa GRPH teda sa používa na šírenie aktualizácie sekvenčného čísla muticast skupiny po sieti. Využitie správy GRPH nastáva napríklad v prípadoch, kedy je potreba znovu pripojiť časti multicast stromov, ktoré boli odpojené od stromu multicast prerušením spojenia. Zaslaním práve správy GRPH do odrezanej vetvy je táto vetva znovu dosažiteľná.

4.2 Udržiavanie záznamov multicast stromu

Pre každý multicast strom, ku ktorému uzol patrí, a to buď preto, že je členom skupiny, alebo preto, že je smerovačom multicst stromu, uzol udržiava zoznam najbližších susedov (*next hop*) - to je tých uzlov, ktorí sú tiež časťou multicast stromu. Tento zoznam susedných uzlov (*next hop*) sa používa pre prenos správ prijatých pre multicast skupinu. Uzol šíri multicast správu ku každému susednému uzlu, avšak okrem toho, od ktorého správa prišla. Ak existuje viacero susedných uzlov, správa môže byť šírená za pomoci multicast paketu. Iba tí susedia, ktorí patria do multicast stromu a ešte neprijali paket, pokračujú v šírení multicast paketu.

4.3 Generovanie správy Route Request

Uzol posiela správu RREQ v prípadoch keď zistí, že by mal byť súčasťou multicast skupiny, ak však tomu ešte tak nie je, alebo ak má správu pre multicast skupinu, ale ešte nepozná cestu k nej. Ak sa chce uzol stať členom multicast skupiny, odošle správu RREQ s príznakom `'J'`. V opačnom prípade necháva príznak správy nenastavený. Cieľová adresa správy RREQ je vždy nastavená na adresu multicast skupiny. Ak uzol pozná lídra multicast skupiny a cestu k nemu, môže uzol vložiť adresu lídra do tzv. "*Multicast Group Leader*"políčka a poslať správu RREQ do zodpovedajúceho cieľa. V opačnom prípade, ak uzol nepozná cestu k lídrovi alebo nepozná samotného lídra multicast skupiny, vyšle správy RREQ (z nevyplneným políčkom "*Multicast Group Leader*") po sieti všesmerovo (tzv. "*broadcast*").

Po odoslaní správy RREQ uzol čaká na príjem správy RREP. Uzol môže opätovne preposlať správu RREQ až pokým neprijme správu RREP (medzu opätovného preposielania určuje parameter RREQ_RETRIES). Ak je RREQ správa odoslaná lídrovi skupiny multicast a správa RREP nie je prijatá do času RREP_WAIT_TIME, uzol následne vyšle všesmerovo správy RREQ pre danú multicast skupinu. Ak správa RREP nie je prijatá ani po ďalších žiadostiach (a medza RREQ_RETRIES je dosiahnutá), uzol môže predpokladať,

že neexistujú ďalší iný členovia tejto skupiny a uzol následne inicializuje sekvenčné číslo danej multicast skupiny. V opačnom prípade, ak chce uzol len poslať pakety, ktoré má pre túto multicast skupinu, ale nechce sa stať jej členom, zahodí pakety ktoré má a preruší komunikáciu.

Ak sa chce uzol pridať ku multicast skupine alebo odoslať správu multicast skupine, musí najskôr skontrolovať svoje záznamy v tabuľke "*Group Leader Table*". Na základe informácii v tejto tabuľke uzol formuluje a pošle správu RREQ ako je popísané na začiatku tejto podkapitoly.

4.4 Prijatie správy Route Request

Ak uzol prijme správu RREQ, skontroluje najskôr či má správa nastavený príznak `'J'`. Ak je príznak `'J'` nastavený, uzol môže odpovedať iba v prípade, keď je členom multicast stromu pre danú multicast skupinu a má záznam o sekvenčnom čísle tejto skupiny, ktoré je prinajmenšom rovnako veľké ako číslo obsiahnuté v správe RREQ. Ak príznak `'J'` nie je nastavený, potom uzol môže odpovedať na správu RREQ v prípade, ak má nastavenú platnú cestu k multicast skupine a spĺňa vyššie spomínané kritérium *sekvenčného čísla*.

Ak uzol nespĺňa ani jednu z uvedených podmienok, môže zaslať správu RREQ z jeho rozhrania (rozhraní), avšak do IP hlavičky vloží svoju IP adresu. Následne uzol aktualizuje sekvenčné číslo cieľu v správe RREQ na maximálnu hodnotu a sekvenčné číslo multicast skupiny vo svojej smerovacej tabuľke. Pole *TTL* alebo *hop limit* zníži svoju hodnotu o jeden. Pole *počet skokov* v odoslanej správe RREQ je inkrementované o jednu.

Uzol vždy tvorí alebo aktualizuje spätné trasy ktoré sú potrebné v prípade, ak uzol dostane správu RREP späť od uzla, ktorý zaslal správu RREQ (rozpoznaného za pomoci IP adresy).

Okrem vytváraní alebo aktualizácii záznamov v smerovacej tabuľke pre zdrojový uzol, uzol prijímajúci správy RREQ taktiež vytvára záznamy o susedných uzloch (*next hop*) pre multicast skupinu do jej smerovacej tabuľky. Ak neexistuje žiadny záznam pre multicast skupinu, uzol záznam vytvorí a potom umiestni uzol, od ktorého prijal správu RREQ ako susedný uzol pre túto skupinu.

4.5 Generovanie správy Route Replies

Ak uzol prijme správu RREQ pre pripojenie sa ku skupine multicast a je už členom multicast stromu pre danú skupinu, uzol aktualizuje multicast smerovacie tabuľky a vygeneruje RREP správu. Zdrojová a cieľová IP adresa v správe RREQ sú skopírované do príslušných polí v správe RREP. Správa RREP obsahuje aktuálne sekvenčné číslo pre multicast skupinu a IP adresu lídra skupiny multicast. Okrem toho, uzol inicializuje pole "počet skokov"(*hop count*) správy RREP na nulu. Ďalšie informácie o multicast skupine sú vložené do poľa "*Multicast Group Information*". Správa RREP je odoslaná späť do uzla, od ktorého bola prijatá správa RREQ.

Uzol môže odpovedať na správu RREQ "pripojiť sa"(*join request*) iba v prípade, ak je členom multicast stromu. Ak uzol dostane žiadosť, ktorá neobsahuje správu "pripojiť sa", môže odpovedať iba ak má aktuálnu cestu k multicast stromu. V opačnom prípade pokračuje v preposielaní žiadosti RREQ. Ak uzol prijme správu RREQ "pripojiť sa"pre multicast skupinu a nie je ešte členom multicast stromu danej skupiny multicast, prepošle správu RREQ svojim susedným uzlom.

Ak uzol prijme správu RREQ, ktorá má v políčku "zdrojová IP adresa"IP adresu tohto uzlu, zdrojový uzol očakáva, že tento uzol je lídrom multicast skupiny. V tomto prípade, ak uzol v skutočnosti nie je lídrom skupiny multicast správu RREQ jednoducho ignoruje. Zdrojovému uzlu tak vyprší čas RREP_WAIT_TIME a uzol prepošle všesmerovo (*broadcast*) správu RREQ bez stanovenia adresy lídra skupiny multicast.

Bez ohľadu na to či správu RREP generuje líder skupiny multicast alebo člen stromu multicast, políčka správy RREP sú nastavené nasledujúco:

- *počet skokov*: 0,
- *cieľová IP adresa*: IP adresa skupiny multicast,
- *sekvenčné číslo cieľa*: aktuálne sekvenčné číslo multicast skupiny,
- *životnosť*: doba, po ktorú uzly obdržia RREP s ohľadom na platnosť cesty (používa sa iba v prípade ak uzol neobdrží správu RREQ "pripojiť sa").

Pole "*Multicast Group Information*"je taktiež obsiahnuté v správe RREQ "pripojiť sa". Ak uzol generujúci RREP nie je členom multicast stromu (pretože správa RREQ nebola charakteru "pripojiť sa"), vloží uzol jeho vzdialenosť od multicast stromu do poľa "počet skokov"(*hop count*), namiesto pôvodnej nuly.

4.6 Šírenie správy Route Replies

Ak medziľahlý uzol prijme správu RREP ako odozvu na správu RREQ (ktorú predtým preposlal), uzol vytvorí záznam o "nasledujúcom skoku"(*next hop*) ku multicast skupine pre uzol, od ktorého prijal správu RREP. Okrem toho, uzol aktualizuje záznam "životnosť"(*Lifetime*) v smerovacej tabuľke, ktorý je združený s uzlom, od ktorého získal správu RREP. Následne sa inkrementujú hodnoty polí "počet skokov"(*hop count*) a "počet skokov ku multicast skupine"(*Multicast Group Hop Count*). Takto upravený paket sa pošle ďalej ku zdroju správy RREQ.

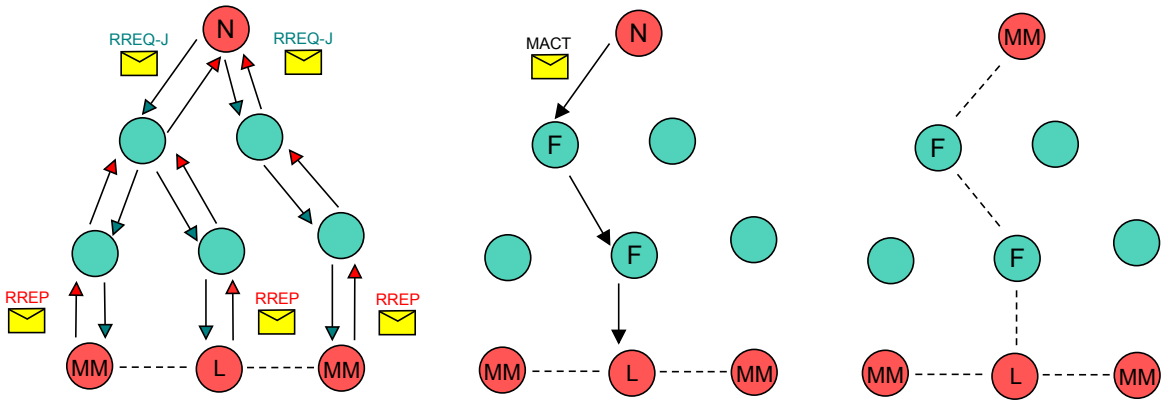
4.7 Aktivácia cesty

Keď uzol vysiela správu RREQ, je pravdepodobné, že prijme viac ako jednu odpoveď od uzlov nachádzajúcich sa v multicast strome. Správa RREP nastaví cestu ukazovateľov zatiaľ čo cestuje späť ku zdrojovému uzlu. Ak je správa RREQ typu "pripojiť sa"(*join request*), môžu tieto ukazovatele prípadne zahrnúť vetvu do multicast stromu. Pretože multicast dáta môžu byť vysielané ako "*broadcast*", cesta k multicast stromu musí byť jednoznačne vybraná. V opačnom prípade, každý uzol s cestou ku stromu, ktorý prijme multicast paket bude tento paket preposielať, čo vedie k neefektívnemu využitiu šírky pásma siete. Z toho dôvodu je potrebné aktivovať iba jednu cestu vytvorenú správou RREP. Správa RREP obsahujúca najväčšie sekvenčné číslo cieľa je vybraná a pridaná ako vetva multicast stromu (alebo ako cesta k multicast stromu, ak správa nebola charakteru "pripojiť sa"). V prípade, že uzol prijme viac ako jednu správu RREP s rovnako veľkým (najväčším) sekvenčným číslom, uzol vyberie tú, ktorá má najmenšiu hodnotu "počet skokov"(*hop count*), to je najkratšia cesta ku členom multicast stromu.

Po prejdení času RREP_WAIT_TIME, musí uzol vybrať cestu, ktorú bude chcieť použiť ako cestu ku multicast stromu. Toto sa dosiahne zaslaním správy MACT (*Multicast Activation*). Pole "IP adresa cieľa"správy MACT je nastavená na IP adresu multicast skupiny. Po nastavení uzol odošle túto správu na vybranú cestu. Takto sa nastaví záznam v smerovacej tabuľke (*Activated*), ktorý je združený s týmto uzlom. Po prijatí tejto správy uzlom, ktorému bola správa MACT odoslaná, aktivuje záznam o ceste vo svojej multicast smerovacej tabuľke, čím sa ukončí vytvorenie vetvy. Všetky susedné uzly, ktoré neprijali túto správu do svojich smerovacích tabuliek, nebudú mať nikdy aktivovaný záznam o tejto ceste.

Existujú dva scenáre ako môžu susedné uzle prijať správu MACT. Ak bol uzol predtým členom multicast stromu, nebude ďalej šíriť správu MACT. Avšak, ak je nasledujúci uzol vybraný zdrojovým uzlom správy MACT, ktorý nebol členom multicast skupiny, uzol bude šíriť pôvodnú správu RREQ ďalej po sieti, hľadajúc uzly siete, ktoré sú už členmi stromu. Tak je možné, že tento uzol prijme viac ako jednu správu RREP.

Keď uzol prijme správu MACT, vytvorí si záznam o nasledujúcom skoku, odošle vlastnú MACT správu uzlu, ktorý si vyberie ako susedný, a tak pokračuje ďalej až pokým nedosiahne uzol, ktorý bude časťou multicast stromu.



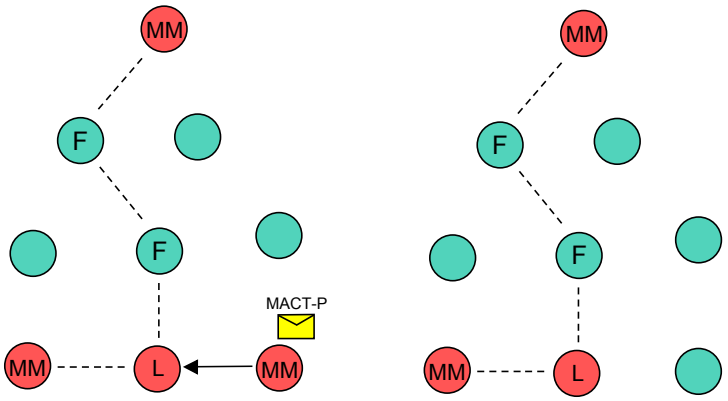
Obrázek 4.1: Pripojenie sa uzlu "N"ku multicast skupine.

4.8 Odpojenie sa od multicast stromu

Člen multicast skupiny môže zmeniť kedykoľvek svoj členský status. Avšak, opustiť multicast strom môže iba v prípade, ak nie je smerovačom v strome pre ostatné uzly v multicast skupine (t.j. ak je koncovým uzlom). Ak uzol, ktorý chce opustiť multicast skupinu, je koncovým uzlom, odošle svojmu susednému uzlu správu MACT, ktorá má nastavený príznak `'P'` a nastavenú cieľovú IP adresu ako adresu skupiny multicast. Až potom si môže uzol vymazať záznam o multicast skupine zo svojej smerovacej tabuľky. Ak susedný uzol prijme správu MACT uzla, ktorý sa chce odpojiť od multicast skupiny, vymaže si informácie o tomto uzle a smerovaní k nemu. Ak odstránením odosielajúceho uzla sa spôsobí, že tento uzol sa stane koncovým a ak tento uzol nie je členom multicast skupiny, môže uzol odrezať (*prune*) samého seba odoslaním vlastnej správy MACT smerom ku stromu multicast.

Keď líder skupiny multicast chce opustiť multicast skupinu, urobí tak spôsobom popísaným vyššie. Ak je líder skupiny práve koncovým uzlom, môže opustiť skupinu odoslaním správy "*prune*"ku svojmu susednému uzlu. V opačnom prípade, ak líder skupiny nie je

koncovým uzlom, nemôže odrezať samého seba z multicast stromu.



Obrázek 4.2: Odpojenie sa multicast člena "MM"od multicast skupiny.

4.9 Správa Group Hello

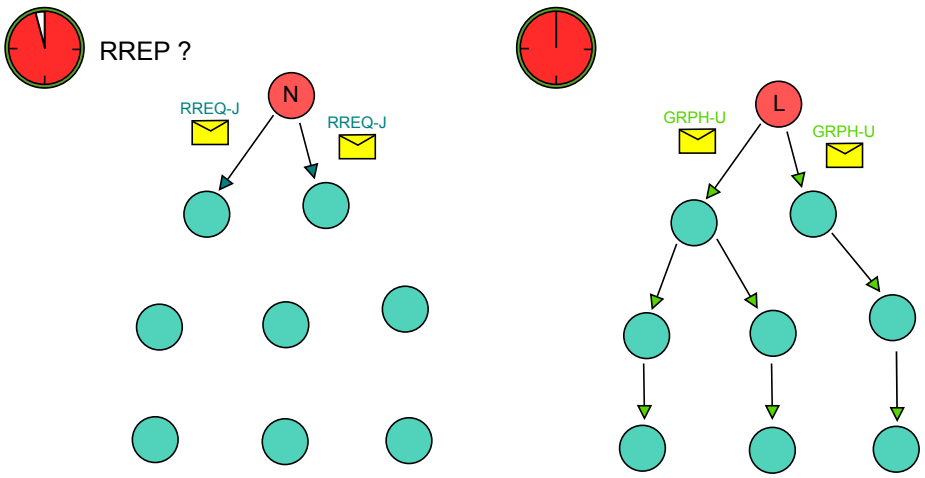
Ak uzol pošle správu RREQ pre pripojenie sa k multicast skupine (t.j. s príznakom nastaveným na `'J'`) a po pokusoch RREQ_RETRIES nedostane odpoveď, stane sa lídrom skupiny multicast. Uzol následne inicializuje sekvenčné číslo multicast skupiny a odošle všesmerove (*broadcast*) správy GRPH smerom do siete, aby informoval uzly v sieti, že on je novým lídrom skupiny multicast. Aby sa informácia o novom lídrovi skupiny šírila konzistentne, vysiela líder skupiny multicast správy GRPH naprieč sieťou v intervaloch (tzv. GROUP_HELLO_INTERAVAL). Obsah polí správy GRPH je nastavený nasledovne:

- *príznak U*: 0,
- *príznak M*: 0,
- *počet skokov*: 0,
- *IP adresa lídra skupiny*: IP adresa lídra skupiny multicast,
- *IP adresa multicast skupiny*: IP adresa multicast skupiny, pre ktorú je uzol lídrom skupiny,
- *sekvenčné číslo multicast skupiny*: o jednu viac ako posledné známe sekvenčné číslo multicast skupiny.

Uzly prijímajúce správu GRPH inkrementujú o hodnotu jedna pole "počet skokov"(*hop count*) pred samotným odoslaním správy. Ak uzol nie je v multicast skupine a prijme správu GRPH, nastaví príznak na `'M'`. To znamená, že táto forma správy cestovala multicast stromom, a preto nemôže byť použitá členmi skupiny za účelom overenia vzdialenosti od lídra skupiny. Príznak `'U'` je nastavený lídrom skupiny v prípade keď dôjde k zmene informácie lídra skupiny. Pri tejto zmene informuje uzly, ktoré by si mali aktualizovať informáciu o lídrovi súvisiacu s danou multicast skupinou.

Uzol, ktorý prijme správu GRPH, najskôr overí či náhodou už predtým neprijal správu GRPH s rovnakou IP adresou/sekvenčným číslom skupiny v čase BCAST_ID_SAVE. Ak

áno, prijatú správu GRPH jednoducho zahodí. V opačnom prípade uzol aktualizuje svoju tabuľku lídra skupiny nastavením aktuálnej IP adresy/IP adresy lídra multicast skupiny a inkrementuje políčko "počet skokov"(*hop count*) v správe. Ak uzol, ktorý prijme správu GRPH je členom multicast stromu, taktiež aktualizuje túto informáciu vo svojej multicast smerovacej tabuľke. Ak príznak `'M'` nie je v správe nastavený, potom tento uzol môže použiť pole "počet skokov"pre overenie aktuálnej vzdialenosti od lídra skupiny multicast. Po spracovaní správy GRPH si uzol uloží kombináciu "IP adresa/sekvenčné číslo", aby sa zabránilo viacnásobnému spracovaniu rovnakej správy GRPH. Uzol nakoniec prepošle správu ku svojim susedným uzlom.



Obrázek 4.3: Vytvorenie nového lídra "L"skupiny multicast.

Kapitola 5

Vlastná špecifikácia protokolu MAODV

V tejto kapitole bude popísaný vlastný návrh protokolu MAODV a budú ukázané zmeny oproti pôvodnej verzii [1, 3].

5.1 Hlavné rozdiely

Hádam najpodstatnejšou zmenou je úprava správ. Správy sa odlišujú od pôvodnej verzii štruktúrou, obsahom jednotlivých polí, použitými príznakmi, atď. Veľmi dôležitou zmenou je, že vlastný návrh protokolu MAODV sa nestará o opravu "spadnutých"spojení ako tomu je v pôvodnej verzii. Uzol, ktorý detekuje susedný uzol ako nečinný sa teda nepodieľa na jeho oprave (obnovení), ale informuje susedné uzle o nefunkčnosti príslušného uzlu. Po prijatí správy o nečinnom uzle a po exspirovaní príslušného časovaču si uzol vymaže smerovaciu informáciu o nečinnom uzle zo smerovacej tabuľky. Ďalšou dôležitou zmenou je, že po prijatí správy RREQ-J sa neporovnáva sekvenčné číslo multicast skupiny v správe s číslom uloženým v smerovacej tabuľke uzla. Na správu typu RREQ-J môže odpovedať každý člen príslušnej hľadanej multicast skupiny, ktorý má uložené informácie pre danú multicast skupinu vo svojej smerovacej tabuľke. Pre správne fungovanie protokolu MAODV boli definované vlastné smerovacie tabuľky.

5.2 Definovanie správ

Vlastná verzia protokolu MAODV využíva štyri hlavné typy správ: RREQ, RREP, MACT a GRPH. Každá zo správ obsahuje niekoľko príznakov. Príznaky umožňujú presnejšie špecifikovať ako sa má s danou správou zaobchádzať. Napríklad správa RREQ (bez príznaku) sa odosiela v prípade, že uzol nepozná cestu k inému uzlu, pričom správu RREQ-J (s príznakom *"join"*) odosiela uzol v prípade, ak sa chce pripojiť ku multicast skupine. Uzol na základe nastaveného príznaku v správe vie rozlíšiť, o aký konkrétny typ správy sa jedná. Vďaka príznakom nie je potrebné definovať v novej verzii protokolu MAODV ďalšie typy správ, čím sa podstatne zjednoduší komunikácia medzi uzlami.

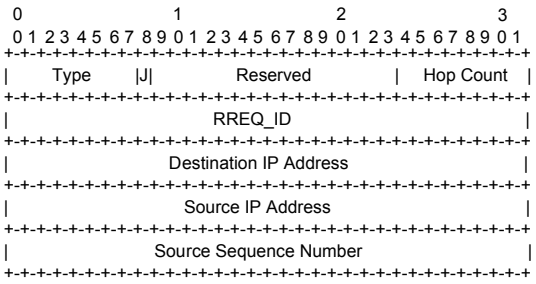
Významnou zmenou prešli aj štruktúry správ. Keďže sú definované iba štyri druhy správ, polia správ musia byť všeobecnejšieho charakteru. Pri odosielaní sú nepoužité polia v správe implicitne nastavené na hodnotu FALSE. Týmto sa predíde nežiaducemu spracovaniu správ.

Aktuálnosť ciest medzi uzlami je zabezpečená pomocou sekvenčných čísiel. Uzol si inkrementuje svoje sekvenčné číslo vždy ako vyšle správu RREQ, RREP, MACT alebo GRPH. Výnimku tvorí iba správa RREP-H (s príznakom *"hello"*), ktorej odoslaním informuje uzol svoje susedné uzle o svojej existencii a o svojom aktuálnom sekvenčnom čísle. Na správu RREQ-J (správa o pripojení sa do multicast skupiny) môže odpovedať iba člen skupiny multicast. Ostatné uzle túto správu iba preposielajú ďalej. Správu GRPH (*"group hello"*) môže odoslať iba líder skupiny multicast.

5.2.1 Štruktúra paketov RREQ

Formát správy RREQ zobrazenej na obr.5.1 obsahuje nasledujúce polia:

- *Type* – označuje o aký typ paketu sa jedná, v tomto prípade RREQ,
- *bez príznaku* – správa bez príznaku je odosielaná ako žiadosť o vyhľadanie uzla,
- *J* – s príznakom *"join"* ako žiadosť o pripojenie sa do multicast skupiny,
- *Hop Count* – počet skokov od zdroja správy,
- *RREQ_ID* – poradové číslo, ktoré jednoznačne definuje správu RREQ v spojení s IP adresou zdroja správy,
- *Destination IP Address* – IP adresa uzla, ktorému je správa určená,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREQ vytvoril),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREQ vytvoril).



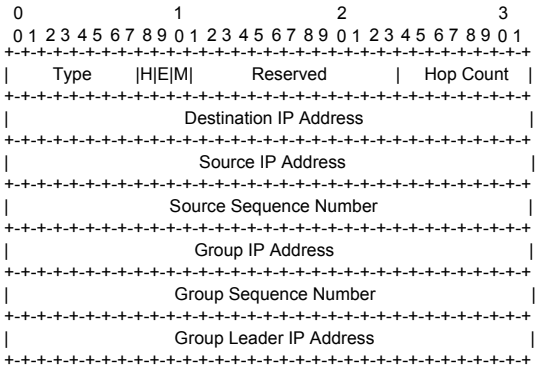
Obrázek 5.1: Štruktúra správy RREQ.

5.2.2 Štruktúra paketov RREP

Formát správy RREP zobrazenej na obr.5.2 obsahuje nasledujúce polia:

- *Type* – označuje o aký typ paketu sa jedná, v tomto prípade RREP,
- *bez príznaku* – správa bez príznaku je odosielaná ako odpoveď na správu RREQ,
- *H* – s príznakom "hello", uzol informuje susedné uzle o svojej existencii a o svojom aktuálnom sekvenčnom čísle,

- *E* – s príznakom "error", uzol informuje susedné uzle o výpadku v sieti,
- *M* – s príznakom "multicast", indikuje odpoveď pre potreby multicast komunikácie,
- *Hop Count* – počet skokov od zdroja správy,
- *Destination IP Address* – IP adresa uzla, ktorému je správa určená,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREP vytvoril),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREP vytvoril),
- *Group IP Address* – IP adresa multicast skupiny,
- *Group Sequence Number* – sekvenčné číslo multicast skupiny,
- *Group Leader IP Address* – IP adresa lídra multicast skupiny.

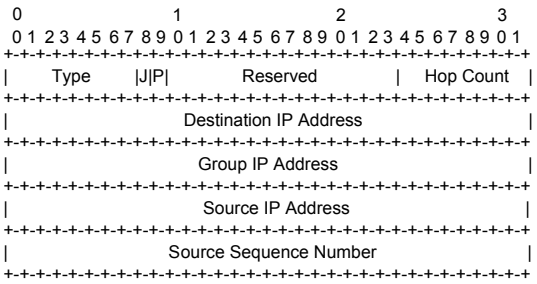


Obrázek 5.2: Štruktúra správy RREP.

5.2.3 Štruktúra paketov MACT

Formát správy MACT zobrazenej na obr.5.3 obsahuje nasledujúce polia:

- *Type* – označuje o aký typ paketu sa jedná, v tomto prípade MACT,
- *J* – s príznakom "join", uzol potvrdzuje svoje pripojenie sa k multicast skupine a aktivuje cestu (vetvu) k multicast skupine,
- *P* – s príznakom "prune", uzol sa odpája od multicast skupiny,
- *Hop Count* – počet skokov od zdroja správy,
- *Destination IP Address* – IP adresa uzla, ktorému je správa určená,
- *Group IP Address* – IP adresa multicast skupiny ku ktorej sa uzol pripája,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu MACT vytvoril),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu MACT vytvoril).

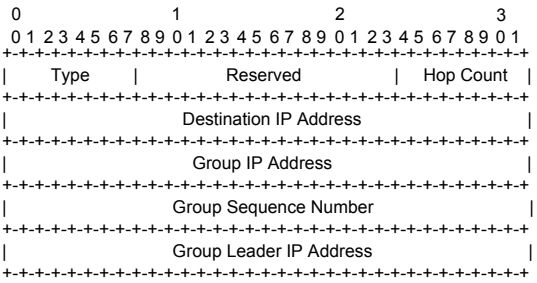


Obrázek 5.3: Štruktúra správy MACT.

5.2.4 Štruktúra paketov GRPH

Formát správy GRPH zobrazenej na obr.5.4 obsahuje nasledujúce polia:

- *Type* – označuje o aký typ paketu sa jedná, v tomto prípade GRPH,
- *Hop Count* – počet skokov od zdroja správy,
- *Destination IP Address* – IP adresa uzla, ktorému je správa určená (poväčšinou broadcast),
- *Group IP Address* – IP adresa multicast skupiny,
- *Group Sequence Number* – sekvenčné číslo multicast skupiny,
- *Group Leader IP Address* – IP adresa lídra multicast skupiny.



Obrázek 5.4: Štruktúra správy GRPH.

5.3 Smerovacie tabuľky

Každý uzol si vedie svoje záznamy v štyroch tabuľkách: *RREQ Buffer Table*, *Multicast Membership Table*, *Multicast Routing Table* a *Routing Table*. Tabuľka *Routing Table* obsahuje záznamy pre smerovanie správ medzi uzlami, tabuľka *Multicast Routing Table* obsahuje záznamy o multicast členoch a *Multicast Membership Table* je tabuľka so záznamami, do ktorých multicast skupín daný uzol patrí.

Pre prijímanie správ RREQ má každý uzol vytvorenú osobitú tabuľku *RREQ Buffer Table* obsahujúcu polia *"zdrojová adresa"* a *"RREQ_ID"*, ktoré sú pre každý uzol jedinečné.

Pri každom odoslaní správy RREQ si uzol inkrementuje hodnotu *"RREQ_ID"* a vytvorí si záznam v tabuľke *RREQ Buffer Table*. Uzol prijímajúci správu RREQ si porovná údaje v správe s údajmi uloženými v tabuľke *RREQ Buffer Table*. Ak sa daný záznam už v tabuľke nachádza, správa sa ignoruje. V opačnom prípade je správa spracovávaná ďalej a je vytvorený nový záznam do tabuľky.

Nadmerný počet správ blúdiacich medzi uzlami by mal negatívny a až kritický dopad na sieť. Tá by sa stala zahltenou, pretože uzle by neboli schopné spracovávať nadmerné množstvo správ. Aby správy neblúdili medzi uzlami chaoticky, každý uzol prijímajúci správu, si vždy kontroluje od koho správa prišla. Pre kontrolu prijatia/neprijatia správ RREQ slúži už spomínaná tabuľka *RREQ Buffer Table*. Ostatné správy sú porovnávané s údajmi v tabuľkách *Multicast Routing Table* a *Routing Table*. Uzol vo svojich smerovacích tabuľkách porovnáva trojicu parametrov *"cieľová adresa – sekvenčné číslo – počet skokov"* s parametrami v prijatej správe. Ak sa všetky tri parametre rovnajú, znamená to, že uzol už túto správu predtým prijal a takúto správu jednoducho zahodí. V opačnom prípade bude správa spracovávaná ďalej a záznam sa aktualizuje.

5.3.1 Štruktúra RREQ Buffer Table

Tabuľka obsahuje nasledovné polia:

- *RREQ_ID* – poradové číslo, ktoré jednoznačne definuje správu RREQ v spojení s IP adresou zdroja správy, pri každom odoslaní správy RREQ si uzol svoje *"RREQ_ID"* inkrementuje,
- *Source IP Address* – IP adresa zdrojového uzlu odosielajúceho správu RREQ.

5.3.2 Štruktúra Multicast Membership Table

Tabuľka obsahuje nasledovné polia:

- *Group IP Address* – IP adresa multicast skupiny, ktorej je uzol členom,
- *Group Sequence Number* – sekvenčné číslo skupiny, ktorej je uzol členom,
- *Group Leader IP Address* – IP adresa lídra skupiny multicast.

5.3.3 Štruktúra Routing Table

Tabuľka obsahuje nasledovné polia:

- *Destination IP Address* – IP adresa cieľového uzlu,
- *Destination Sequence Number* – sekvenčné číslo cieľového uzlu,
- *Hop Count* – počet skokov od cieľového uzlu,
- *Cost* – cena je implicitne nastavená na hodnotu 255, ak cieľový uzol nevysiela periodicky správu *"hello"* alebo ak sa nepodieľa na preposielaní správ, je cesta k uzlu označená ako nespoľahlivá a každý uzol, ktorý obsahuje záznam o tomto uzle si dekrementuje políčko *"Cost"* o jednu,

- *Flag* – je nastavený buď na *"TRUE"* alebo *"FALSE"*, závisí od toho či je cesta k danému uzlu aktívna (či exspiroval časovač *"Lifetime"*),
- *Lifetime* – časovač, vyjadruje platnosť cesty do určitého času.

5.3.4 Štruktúra Multicast Routing Table

Tabuľka obsahuje nasledovné polia:

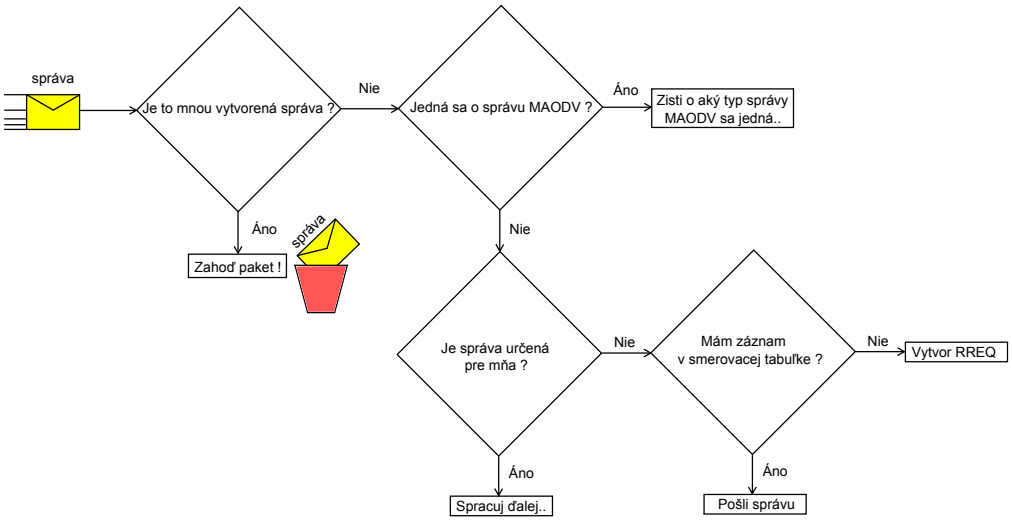
- *Group IP Address* – IP adresa multicast skupiny, ktorej je uzol členom,
- *Group Sequence Number* – sekvenčné číslo skupiny, ktorej je uzol členom,
- *Group Leader IP Address* – IP adresa lídra skupiny multicast,
- *Member IP Address* – IP adresa člena skupiny multicast,
- *Hop Count* – počet skokov od cieľového uzlu,
- *Flag* – je nastavený buď na *"TRUE"* alebo *"FALSE"*, závisí od toho či je cesta k danému uzlu aktívna (či exspiroval časovač *"Lifetime"*),
- *Lifetime* – časovač, vyjadruje platnosť cesty do určitého času.

Kapitola 6

Fungovanie vlastného protokolu MAODV

Každý uzol pri prijatí paketu sa snaží zistiť o aký typ paketu sa jedná. Ak vyhodnotí prijatý paket ako MAODV, následne určuje aký typ správy MAODV prijal (RREQ, RREP, MACT alebo GRPH). Detailnejší popis spracovania jednotlivých správ bude popísaný v nasledujúcich podkapitolách.

Po prijatí paketu uzol najskôr dekrementuje pole TTL v hlavičke IP. Ak hodnota TTL nie je rovná nule, je paket spracovaný ďalej. V ďalšom kroku uzol porovnáva zdrojovú IP adresu zo svojou IP adresou. Ak sa adresy rovnajú, znamená to, že uzol prijal sebou odoslanú správu a správu zahodí. Následne uzol identifikuje typ paketu. Po identifikácii typu paketu si uzol porovnáva svoju IP adresu s cieľovou IP adresou nachádzajúcej sa v prijatej správe. Ak sa IP adresy zhodujú alebo ak IP adresa v prijatej správe je označená ako *"broadcast"*, uzol spracováva prijatú správu ďalej. V opačnom prípade sa uzol snaží zistiť či má vo svojej smerovacej tabuľke záznam o uzle, ktorému je správa určená. Ak záznam má, správu prepošle danému uzlu. Ak sa záznam o príslušnom uzle v tabuľke nenachádza, uzol vytvorí a odošle správu RREQ. Celý postup spracovania správy zobrazuje obr.6.1:



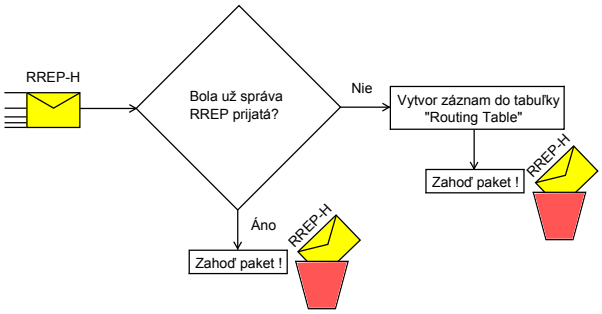
Obrázek 6.1: Proces spracovania správy.

6.1 Vytvorenie a prijatie správy RREP-H

Prvá komunikácia uzlov prebieha nastavením a odoslaním správy RREP-H (s príznakom *"hello"*). Uzol informuje susedné uzle o svojej existencii a o svojom aktuálnom sekvenčnom čísle. Správa bude nastavená nasledovne:

- *Type* – RREP,
- *Flag* – s príznakom H (*"hello"*),
- *Hop Count* – 1,
- *Destination IP Address* – broadcast,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREP vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREP vytvorí),
- *Group IP Address* – FALSE,
- *Group Sequence Number* – FALSE,
- *Group Leader IP Address* – FALSE.

Všetky správy typu RREP-H majú nastavenú hodnotu TTL = 2. Predíde sa tak nekontrolovateľnému šíreniu správ medzi uzlami. Susedné uzly teda správu RREP-H po spracovaní nepreposielajú ďalej, ale správu jednoducho zahodia.



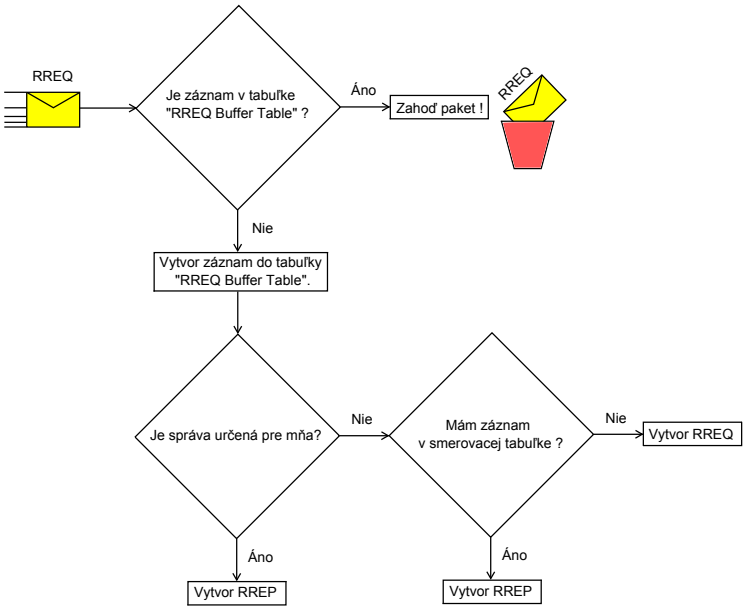
Obrázek 6.2: Proces spracovania správy RREP-H.

Na obr.6.2 je zobrazené spracovanie správy RREP-H. Uzol pred vytvorením záznamu skontroluje či už danú správu neprijal a či sa už záznam v jeho tabuľke nenachádza. V správe porovnáva polia *"zdrojová IP adresa"*, *"sekvenčné číslo"* a *"počet skokov"*. Ak záznam o danom susednom uzle neexistuje, vytvorí si ho. Ak správa obsahuje záznam s lepším sekvenčným číslom alebo počtom skokov, uzol si aktualizuje záznam vo svojej tabuľke *Routing Table*. V opačnom prípade sa prijatím správy nič neaktualizuje a uzol správu zahodí.

6.2 Vytvorenie a prijatie správy RREQ

Ak uzol nepozná cestu k inému uzlu, napríklad prijal správu pre takýto uzol a nemá pre neho záznam vo svojej smerovacej tabuľke, vytvorí správu RREQ bez príznaku. Správa bude nastavená nasledovne:

- *Type* – RREQ,
- *Flag* – bez príznaku,
- *Hop Count* – 1,
- *RREQ_ID* – identifikačné číslo správy RREQ,
- *Destination IP Address* – IP adresa hľadaného uzla,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREQ vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREQ vytvorí).



Obrázek 6.3: Proces spracovania správy RREQ.

Pred odoslaním správy si uzol inkrementuje svoje sekvenčné číslo a hodnotu RREQ_ID (tieto hodnoty už zapísal aj do poľa *"RREQ_ID"* a *"Source Sequence Number"* v tele správy) a vytvorí si záznam do tabuľky *RREQ Buffer Table* pre dvojicu *"RREQ_ID"*, kde bude zaznamenaná aktuálna hodnota identifikačného čísla správy RREQ a *"Source IP Address"*, kde zapíše svoju IP adresu. Takto sa zabráni opakovanému spracovaniu správy pri jej opätovnom prijatí.

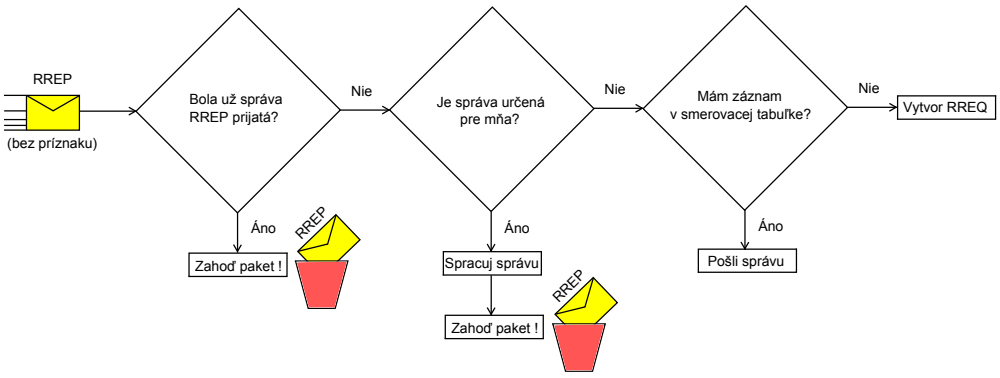
Uzol prijímajúci správu RREQ si skontroluje v tabuľke *"RREQ Buffer Table"* či už danú správu neprijal. Ak správu predtým ešte neprijal, vytvorí si o nej záznam do tabuľky a bude pokračovať v spracovaní správy, v opačnom prípade správu zahodí.

Po vytvorení si záznamu si uzol aktualizuje alebo vytvorí záznam o uzle vo svojej smerovacej tabuľke *Routing Table*, od ktorého správa prišla. Následne uzol porovnáva svoju IP adresu s cieľovou IP adresou nachádzajúcej sa v prijatej správe. Ak sa IP adresy zhodujú alebo ak IP adresa v prijatej správe je označená ako *"broadcast"*, uzol spracováva prijatú správu ďalej. V opačnom prípade sa uzol snaží zistiť či má vo svojej smerovacej tabuľke záznam o uzle, ktorému je správa určená. Ak záznam má, odpovedá uzlu, ktorý vytvoril správu RREQ. Odpoveď bude v podobe správy RREP. Ak sa záznam o danom uzle v tabuľke nenachádza, uzol vytvorí a odošle vlastnú správu RREQ. Celý postup spracovania správy zobrazuje obr.6.3.

6.3 Vytvorenie a prijatie správy RREP

Uzol vytvára správu RREP ako odpoveď na správu RREQ. Odpovedať môže i uzol, ktorému nie je správa určená, ale má uložený záznam o hľadanom uzle vo svojej smerovacej tabuľke. Správa bude nastavená nasledovne:

- *Type* – RREP,
- *Flag* – bez príznaku,
- *Hop Count* – hodnota počtu skokov uložená v smerovacej tabuľke plus jedna,
- *Destination IP Address* – IP adresa uzlu, ktorý vytvoril správu RREQ,
- *Source IP Address* – IP adresa hľadaného uzla,
- *Source Sequence Number* – sekvenčné číslo hľadaného uzla,
- *Group IP Address* – FALSE,
- *Group Sequence Number* – FALSE,
- *Group Leader IP Address* – FALSE.



Obrázek 6.4: Proces spracovania správy RREP.

Spracovanie správy RREP bez príznaku je zobrazené na obr.6.4. Uzol si najskôr skontroluje či už danú správu neprijal na základe troch parametrov: *"zdrojová IP adresa"*,

"sekvenčné číslo" a *"počet skokov"*. Ak správa nebola ešte prijatá, to znamená, že uzol si pridá alebo aktualizuje záznam vo svojej smerovacej tabuľke, pokračuje v spracovaní správy. V opačnom prípade správu zahodí. Následne si uzol kontroluje komu je správa určená. Porovnáva svoju IP adresu s cieľovou IP adresou uloženou v správe RREP. Ak sú IP adresy zhodné a uzol si už aktualizoval alebo pridal záznam o hľadanom uzle, správu zahodí. V prípade, že sa IP adresy nezhodujú bude uzol vo svojej smerovacej tabuľke hľadať záznam o cieľovej IP adrese obsiahnutej v správe RREP. Ak záznam nájde, správu prepošle príslušnému uzlu. V opačnom prípade vytvorí správu RREQ.

6.4 Vytvorenie a prijatie správy RREP-E

Správu RREP-E s príznakom *"error"* posiela uzol svojim susedným uzlom, ak zistí nečinnosť svojho susedného uzla. Nečinný uzol je taký uzol, od ktorého neprišla správa RREP-H s príznakom *"hello"* do určitého časového intervalu. Príčinou môže byť zlyhanie spojenia, výpadok uzlu, posun uzlu, atď. Pri objavení nečinného suseda nastaví uzol správu RREP-E nasledovne:

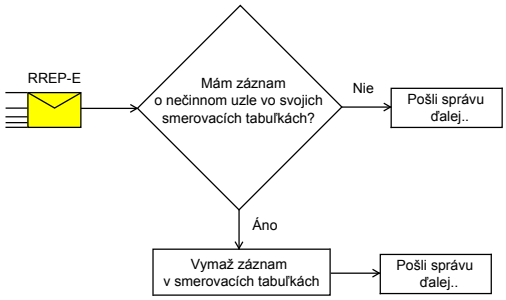
- *Type* – RREP,
- *Flag* – s príznakom E (*"error"*),
- *Hop Count* – hodnota počtu skokov uložená v tabuľke plus jedna,
- *Destination IP Address* – broadcast,
- *Source IP Address* – IP adresa nečinného uzla,
- *Source Sequence Number* – sekvenčné číslo nečinného uzla,
- *Group IP Address* – FALSE,
- *Group Sequence Number* – FALSE,
- *Group Leader IP Address* – FALSE.

Nastavenie počtu skokov v správe RREP-E nie je až tak dôležité, pretože uzly prijímajúce túto správu si budú vymazávať záznam o nečinnom uzle zo svojich smerovacích tabuliek. Naopak, dôležité je nastavenie poľa TTL, pretože nevhodnou voľbou tohto parametru bude správa zbytočne spracovaná aj uzlami, ktoré nemajú o nečinnom uzle záznam vo svojich smerovacích tabuľkách.

Na obr.6.5 je zobrazený postup spracovania správy RREP-E. Uzol prijímajúci túto správu si vyhľadá príslušný záznam o nečinnom uzle vo svojich smerovacích tabuľkách *Routing Table* a *Multicast Routing Table*. Ak záznam nájde, vymaže si ho a správu prepošle ďalej. Ak záznam pre nečinný uzol neexistuje ani v jednej zo smerovacích tabuliek, správa sa ignoruje a prepošle ďalej.

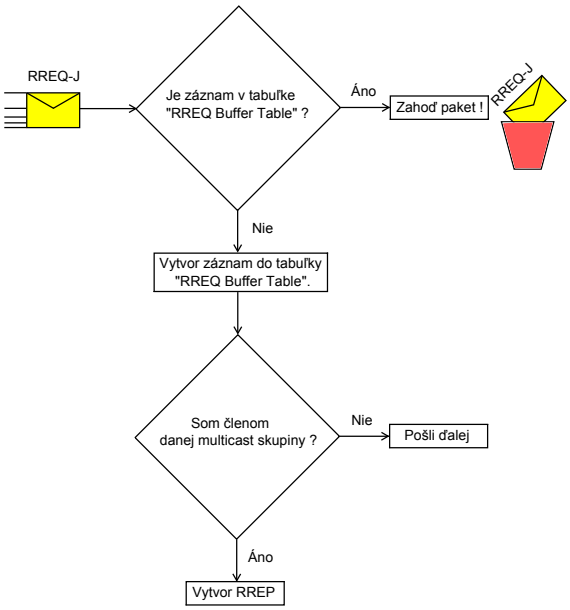
6.5 Vytvorenie a prijatie správy RREQ-J

Správu RREQ-J s príznakom *"join"* odosiela uzol v prípade ak sa chce pripojiť do multicast skupiny alebo má správu pre multicast skupinu. Správa bude nastavená nasledovne:



Obrázek 6.5: Proces spracovania správy RREP-E.

- *Type* – RREQ,
- *Flag* – s príznakom J (*"join"*),
- *Hop Count* – 1,
- *RREQ_ID* – identifikačné číslo správy RREQ,
- *Destination IP Address* – IP adresa hľadanej multicast skupiny,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREQ vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREQ vytvorí).



Obrázek 6.6: Proces spracovania správy RREQ-J.

Proces spracovania prijatej správy RREQ-J je rovnaký ako u správy RREQ bez príznaku. Uzol si porovnáva polia *"RREQ_ID"* a *"zdrojová adresa"* správy so záznamy v tabuľke

RREQ Buffer Table. Po vytvorení si záznamu si uzol aktualizuje alebo vytvorí záznam o uzle vo svojej smerovacej tabuľke *Routing Table*, od ktorého správa prišla. Keďže má správa nastavený príznak *"join"*, bude sa jednať o multicast správu. Uzol porovná IP adresu hľadanej multicast skupiny so záznamami uloženými v tabuľke *Multicast Membership Table*, pretože na správu RREQ-J môže odpovedať iba člen hľadanej skupiny multicast. Ak je uzol členom multicast skupiny odpovie na správu RREQ-J správou RREP-M. V opačnom prípade správu prepošle susedným uzlom. Celý postup spracovania správy zobrazuje obr.6.6.

6.6 Vytvorenie a prijatie správy RREP-M

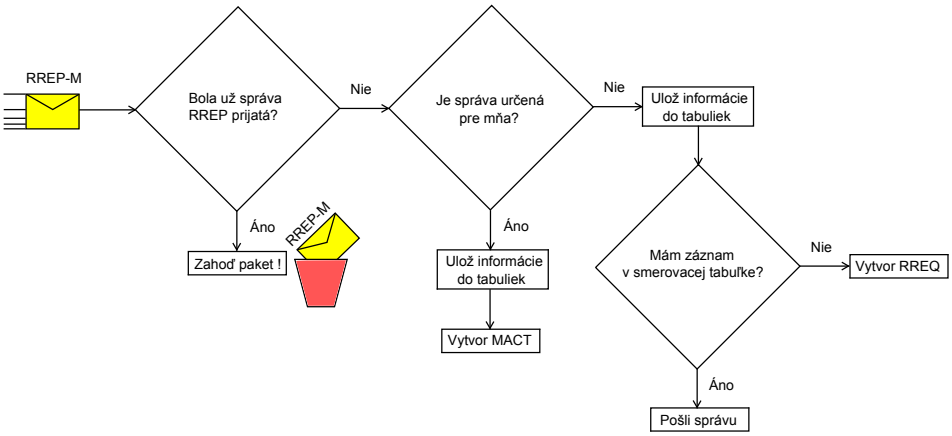
Každý uzol, ktorý obdrží správu RREQ-J a je členom hľadanej multicast skupiny, môže odpovedať správou RREP-M s príznakom *"multicast"*. Odpovedajúci uzol si najskôr uloží informácie o novom členovi do tabuľky *Multicast Routing Table*, pričom príslušný záznam bude označený ako neaktívny a to až do chvíle, pokým uzol neobdrží správu MACT-J.

Informácie o multicast skupine, ktoré budú vložené do správy RREP-M, sa zoberú z tabuľky *Multicast Membership Table*. Správa bude nastavená nasledovne:

- *Type* – RREP,
- *Flag* – s príznakom M (*"multicast"*),
- *Hop Count* – 1,
- *Destination IP Address* – cieľová IP adresa uzla, ktorý vytvoril správu RREQ-J,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu RREP-M vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu RREP-M vytvorí),
- *Group IP Address* – IP adresa hľadanej multicast skupiny,
- *Group Sequence Number* – sekvenčné číslo hľadanej multicast skupiny,
- *Group Leader IP Address* – IP adresa lídra multicast skupiny.

Pri prijatí správy RREP-M si uzol kontroluje či už danú správu neprijal. Aktualizuje si alebo vytvorí záznam o uzle, ktorý správu RREP-M vytvoril, do svojej smerovacej tabuľky. Ak nebola prevedená žiadna zmena, správa sa ignoruje. V nasledujúcom kroku uzol porovnáva svoju IP adresu s cieľovou IP adresou v správe RREP-M. Ak sú adresy rovnaké, vloží si uzol informácie zo správy do tabuľky *Multicast Membership Table*. V prípade, že uzol prijme viacero správ RREP-M pre danú multicast skupinu, vyberie si tú, ktorá má najväčšie sekvenčné číslo a do tabuľky *Multicast Routing Table* si pridá záznam o uzle, ktorý správu RREP-M vytvoril. Pre aktivovanie členstva v multicast skupine odošle správu MACT-J.

Ak sa od seba IP adresy odlišujú, vytvorí si uzol záznam do tabuľky *Multicast Routing Table* o novom členovi multicast skupiny. Daný záznam bude označený ako neaktívny, a to až pokým uzol neprijme aktivačnú správu MACT-J. Uzol si ešte aktualizuje alebo vytvorí záznam o členovi multicast skupiny, ktorý správu RREP-M vytvoril. Potom uzol vyhľadá vo svojej smerovacej tabuľke záznam pre uzol, ktorému je správa určená. Ak uzol daný záznam nemá, vytvorí a pošle správu RREQ. Celý postup spracovania správy zobrazuje obr.6.7.



Obrázek 6.7: Proces spracovania správy RREP-M.

6.7 Vytvorenie a prijatie správy MACT-J

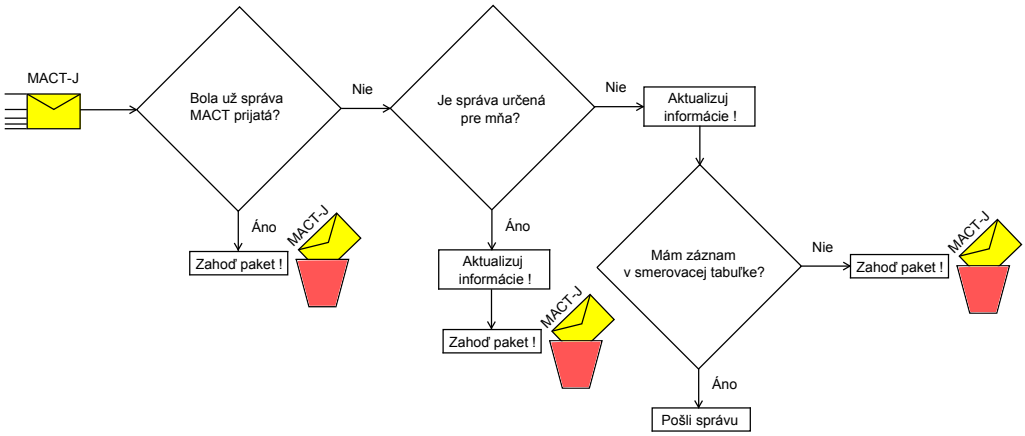
Po prijatí niekoľkých správ RREP-M si môže uzol vybrať tú, ktorá má najväčšie sekvenčné číslo a najmenší počet skokov. Danú cestu aktivuje za pomoci správy MACT-J s príznakom *"join"*. Obsah správy vyzerá nasledovne:

- *Type* – MACT,
- *Flag* – s príznakom J ("join"),
- *Hop Count* – 1,
- *Destination IP Address* – IP adresa uzla, od ktorého prijal správu RREP-M,
- *Group IP Address* – IP adresa multicast skupiny ku ktorej sa uzol pripája,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu MACT vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu MACT vytvorí).

Spracovanie správy MACT-J zobrazuje obr.6.8. Ak uzol prijme správu MACT-J, najskôr si skontroluje, či už danú správu neprijal predtým. Následne porovnáva svoju IP adresu s IP adresou v správe. Ak sa IP adresy zhodujú, aktualizuje si uzol informácie vo svojich smerovacích tabuľkách a správu zahodí. Ak sú IP adresy rôzne, aktualizuje si uzol informácie vo svojich smerovacích tabuľkách a správu prepošle cieľovému uzlu. Pokiaľ uzol nemá záznam o cieľovom uzle, správu zahodí.

6.8 Vytvorenie a prijatie správy MACT-P

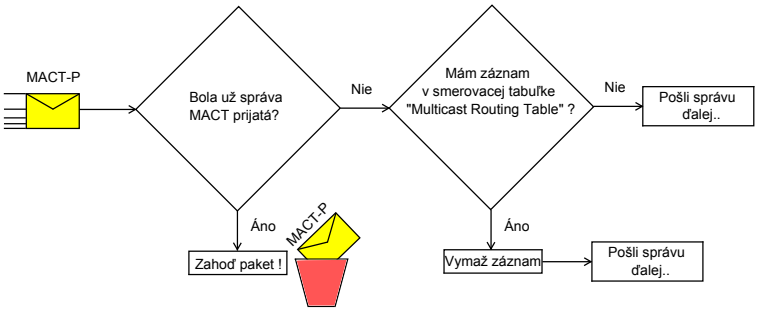
Uzol sa môže odpojiť od multicast skupiny iba v prípade ak nie je medzi uzlom pre ostatné uzly v multicast skupine a iba v prípade kedy by odpojením sa od multicast skupiny neobmedzil komunikáciu uzlov v multicast skupine. To znamená, že uzol je koncovým uzlom v danej vetve multicast skupiny. Svoje odpojenie sa od multicast skupiny uzol inicializuje



Obrázek 6.8: Proces spracovania správy MACT-J.

vytvorením a odoslaním správy MACT-P s príznakom *"prune"* okolitým susedným uzlom. Správa bude nastavená nasledovne:

- *Type* – MACT,
- *Flag* – s príznakom P ("prune"),
- *Hop Count* – 1,
- *Destination IP Address* – broadcast,
- *Group IP Address* – IP adresa multicast skupiny od ktorej sa uzol odpája,
- *Source IP Address* – IP adresa zdrojového uzla (uzol, ktorý správu MACT vytvorí),
- *Source Sequence Number* – sekvenčné číslo zdrojového uzla (uzol, ktorý správu MACT vytvorí).



Obrázek 6.9: Proces spracovania správy MACT-P.

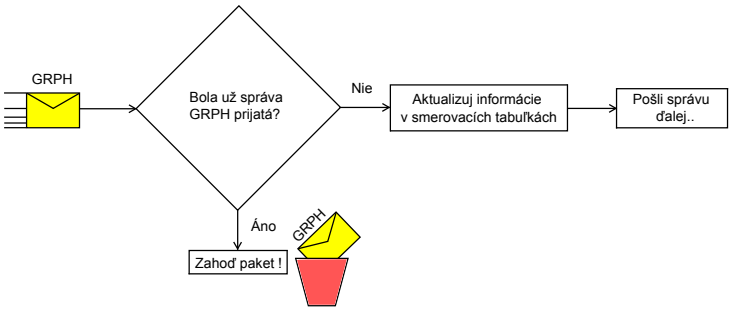
Obr.6.9 zobrazuje postup spracovania správy MACT-P. Po jej prijatí si uzol skontroluje či už danú správu neprijal. Keďže každá správa typu MACT-P má nastavenú cieľovú adresu ako *"broadcast"*, nemusí uzol porovnávať svoju IP adresu s IP adresou v správe a rovno prejde k spracovaniu správy. Vo svojej *Multicast Routing Table* tabuľke si vyhľadá záznam

o uzle, ktorý sa od multicast skupiny odpája. Ak záznam nájde, vymaže si ho a správu pošle ďalej. V opačnom prípade ak záznam nenájde, nenastane v smerovacej tabuľke žiadna zmena a uzol môže správu MACT-P poslať ihneď ďalej.

6.9 Vytvorenie a prijatie správy GRPH

Správa GRPH slúži pre šírenie informácii o multicast skupine okolitým uzlom v sieti. Nesie v sebe informácie o IP adrese multicast skupiny, o sekvenčnom čísle multicast skupiny a o IP adrese lídra skupiny multicast. Správu GRPH môže odosielať iba líder multicast skupiny. Pre zachovanie aktuálnosti ciest ku multicast skupine vysiela líder skupiny túto správu v periodicky sa opakujúcich intervaloch. Nastavenie správy je nasledovné:

- *Type* – GRPH,
- *Hop Count* – 1,
- *Destination IP Address* – broadcast,
- *Group IP Address* – IP adresa multicast skupiny,
- *Group Sequence Number* – sekvenčné číslo multicast skupiny,
- *Group Leader IP Address* – IP adresa lídra multicast skupiny (uzol, ktorý správu GRPH vytvorí).



Obrázek 6.10: Proces spracovania správy GRPH.

Ako znázorňuje obr.6.10, uzol si pred spracovaním správy GRPH skontroluje či danú správu už neprijal. V tomto prípade kontroluje IP adresu multicast skupiny a jej sekvenčné číslo. Následne si v tabuľke *Multicast Routing Table* vyhľadá všetkých členov príslušnej multicast skupiny a aktualizuje im informácie o sekvenčnom čísle získané zo správy GRPH. Ak je uzol členom príslušnej multicast skupiny, aktualizuje si informácie v tabuľke *"Multicast Membership Table"*. Po spracovaní pošle správu susedným uzlom.

Kapitola 7

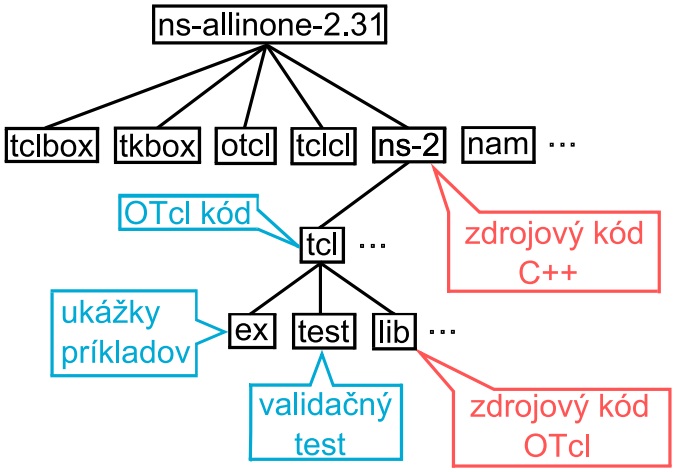
Network Simulator 2

Pri písaní tejto kapitoly bolo čerpané zo zdroja [5].

Pre *Network Simulator* 2 sú definované tieto štyri *ad-hoc* smerovacie protokoly, ktoré sú v ňom aj obsiahnuté: DSDV (*Destination Sequence Distanece Vector*), DSR (*Dynamic Source Routing*), TORA (*Temporally Ordered Routing Algorithm*) a AODV (*Ad-hoc On-demand Distance Vector*). Jedná sa o unicastové smerovacie protokoly, v ktorých preposielanie informácie (správy) sa uskutočňuje z jedného zdrojového uzla do jedného cieľového uzla. Takže v tomto prípade exituje iba jeden odosielateľ a jeden príjemca.

7.1 Skladba zdrojových kódov

Pred samotným rozšírením *Network Simulátoru* 2 o nový protokol, by bolo dobré si ukázať hierarchiu zdrojových kódov tohto simulátoru. Na obr.7.1 je možné vidieť časť adresnej štruktúry simulátora.



Obrázek 7.1: Adresná štruktúra simulátoru NS2.

Sub-adresár `ns-2` je miesto, ktoré obsahuje všetky implementácie simulátora (C++ alebo OTcl), validné testy skriptu OTcl a príklady OTcl skriptu. V tomto adresári sú všetky OTcl kódy a testovacie skripty umiestnené pod sub-adresárom `tcl` a väčšina z kódu C++,

ktorý implementuje udalosť plánovača a základné sieťové súčasti triedy objektu (s výnimkou WWW), sa nachádza na hlavnej úrovni. Napríklad, pred samotnou realizáciu implementácie agenta UDP, je potrebné sa dostať do adresára `ns-allinone-2.33/ns-2` a otvoriť súbory `udp.h`, `udp.cc` a súbory, ktoré obsahujú vykonávanie predchodcu tried UDP podľa potreby.

Tcl adresár obsahuje sub-adresáre, medzi ktoré patrí i `lib` adresár, v ktorom sa nachádzajú zdrojové kódy OTcl pre najzákladnejšie a najpodstatnejšie časti implementácie NS (agent, uzol, paket, adresa, smerovanie, atď.). Zdrojové kódy OTcl pre LAN, web alebo napríklad implementáciu Multicast sú umiestnené v samostatných sub-adresároch zložky `tcl`.

- `ns-lib.tcl`: Nachádzajú sa tu členské funkcie triedy objektu *simulátor* a väčšina z jeho triednych metód okrem tých pre LAN, web, Multicast a ich podobným.
- `ns-default.tcl`: Sú tu umiestnené predvolené hodnoty pre nastaviteľné parametre rôznych sieťových prvkov. Keďže väčšina sieťových prvkov je implementovaná v jazyku C++, nastaviteľné parametre sú vlastne C++ premenné dostupné pre OTcl pomocou funkcie prepojenia z OTcl (`C++_variable_name,OTcl_variable_name`).
- `ns-packet.tcl`: Nachádza sa tu formát prevedenia hlavičky paketu. Pri vytvorení novej paketovej hlavičky, je potreba ju zaregistrovať do tohto súboru tak, aby sa uskutočnil proces inicializácie hlavičky paketu, čím bude hlavička zahrnutá v zásobníku s hlavičkami a bude mať v ňom svoje miesto.
- ostatné OTcl súbory: Ostatné OTcl súbory v tomto adresári obsahujú implementáciu OTCL rôznych sieťových objektov alebo kontrolných častí sieťových objektov v C++. Napríklad aplikácia FTP je celkom implementovaná v OTcl a zdrojový kód je umiestnený v `ns-source.tcl`.

Sub-adresár `ex` obsahuje rôzne príklady simulačných scenárov a sub-adresár `test` obsahuje simulačné scenáre, ktoré overujú inštalovaný NS s rôznymi simuláciami a porovnávajú simulované výsledky s očakávanými výsledkami.

7.2 Príklad väzby jazykov OTcl a C++

Rozšírenie NS o nový objekt zvyčajne zahrňuje vytvorenie väzby medzi jazykmi OTcl a C++. V nasledujúcom príklade bude ukázané ako sa vytvoria nové objekty tried jazykov OTcl a C++ a následne ich vzájomná väzba.

Ako prvý sa musí vytvoriť nový objekt trieda v jazyku C++. V tomto prípade to bude trieda `MyAgent`, ktorá je dedená z triedy `Agent`, pre ktorú je potreba vytvoriť instanciu v jazyku OTcl a to nasledujúcim spôsobom: definuje sa objekt väzby `MyAgentClass`, ktorý je dedený z triedy `TclClass`, čím sa vytvorí OTcl objekt s menom `Agent/MyAgentOTcl` a väzba medzi objektom OTcl a objektom C++ `MyAgent`.

```
1: class MyAgent : public Agent {
2: public:
3:        MyAgent();
4: protected:
5:        int command(int argc, const char*const* argv);
6: private:
7:        int my_var1;
```

```
 8:        double my_var2;
 9:        void MyPrivFunc(void);
10: };
11:
12: static class MyAgentClass : public TclClass {
13: public:
14:        MyAgentClass() : TclClass("Agent/MyAgentOTcl") {}
15:        TclObject* create(int, const char*const*) {
16:            return(new MyAgent());
17:        }
18: } class_my_agent;
```

Pri prvom spustení NS sa vykoná konštruktor pre statické premenné `class_my_agent`, a tak je instancia `MyAgentClass` vytvorená. V tomto procese, trieda `Agent/MyAgentOTcl` a jej príslušné metódy (členské funkcie) sú vytvorené v OTcl. Kedykoľvek sa užívateľ snaží vytvoriť instanciu tohto objektu v OTcl pomocou príkazu `new Agent/MyAgentOTcl`, odvoláva sa na `MyAgentClass::create`, ktorý vytvorí instanciu `MyAgent` a vracia adresu. Avšak vytvorenie C++ instancie objektu z OTcl neznamená, že sa môžu vyvolať členské funkcie alebo prístupové premenné C++ instancie objektu z OTcl.

Nový C++ objekt `MyAgent` má dva parametre premenných a to `my_var1` a `my_var2`, ktoré je možné nastavovať (meniť) jednoducho z OTcl pomocou simulačného skriptu. K tomu slúži funkcia **binding**, ktorú je treba použiť pre každú premennú triedy C++, ktorá má byť exportovaná. Funkcia **binding** vytvára nové členy premenných s menom zhodujúcim sa s menami v triede OTcl `Agent/MyAgentOTcl` a vytvára obojsmerné väzby medzi premennými triedy OTcl a C++, ktorej adresa je uvedená ako druh premennej.

```
1: MyAgent::MyAgent() : Agent(PT_UDP) {
2:     bind("my_var1_otcl", &my_var1);
3:     bind("my_var2_otcl", &my_var2);
4: }
```

NS podporuje štyri rozdielne funkcie *binding* pre päť rôznych typov premenných takto:

- `bind`(): real alebo integer premenná,
- `bind_time`(): časová premenná,
- `bind_bw`(): premenná šírky pásma,
- `bind_bool`(): premenná boolean.

Týmto spôsobom je možné navrhnúť a spustiť simuláciu pomocou skriptu OTcl a taktiež meniť alebo sprístupniť konfigurovateľné parametre (a hodnoty premenných) na sieťové komponenty implementované v jazyku C++. Pri každom exporte premenných C++ je doporučené nastaviť predvolené hodnoty pre túto premennú v súbore `ns-2/tcl/lib/ns-lib.tcl`. V opačnom prípade sa objaví výstražné hlásenie pri vytvorení instancie na nový objekt.

Okrem exportu niektorých premenných z C++, je možné nastaviť riadenie (kontrolu) nad C++ objektmi pomocou OTcl. Toto je možné definovaním funkcie **command** objektu C++ `MyAgent`, ktorá pracuje ako príkazový prekladač jazyka OTcl. V skutočnosti, OTcl príkaz definovaný funkciou **command** vyzerá rovnako ako funkcia zodpovedajúceho objektu OTcl k užívateľovi.

```
1: int MyAgent::command(int argc, const char*const* argv) {
2:     if(argc == 2) {
3:         if(strcmp(argv[1], "call-my-priv-func") == 0) {
4:             MyPrivFunc();
5:             return(TCL_OK);
6:         }
7:     }
8:     return(Agent::command(argc, argv));
9: }
```

Ak instancia OTcl, ktorá odpovedá objektu `MyAgent`, je vytvorená v OTcl `set myagent [new Agent/MyAgentOTcl]` a užívateľ sa pokúša zavolať funkciu tohto objektu `$myagent call-my-priv-func` , OTcl vyhľadá dané meno funkcie v objekte OTcl. Ak daný názov funkcie nemožno nájsť, potom sa dovoláva na `MyAgent::command` prechádzaním apelovaného OTcl mena funkcie a argumentov v argc/argv formáte. Ak sa tu nachádza akcia, ktorá sa odvoláva na meno funkcie OTcl, vykoná to, čo sa aj žiada, a vráti výsledok. Ak nie, funkcia **command** je volaná pre predchodcu objektu, až pokým nie je nájdená. Ak meno nemôže byť nájdené v žiadnom z predchodcov, príde k chybovému hláseniu. Týmto spôsobom je možné v OTcl kontrolovať správanie objektov C++.

Po implementovaní nových sieťových objektov v C++, je taktiež možné vykonávať príkazy OTcl z objektov C++. Implementáciou `MyPrivFunc`, člena funkcie `MyAgent`, ktorý predstavuje OTcl interpreta pre hodnoty `my_var1` a `my_var2`. Pre spustenie príkazu OTcl z C++, je treba získať odkaz na `Tcl::instance`(), ktorý je deklarovaný ako statická premenná. Takto sa získajú funkcie, z ktorými je možné dostať OTcl príkazy do interpreta.

```
1: void MyAgent::MyPrivFunc(void) {
2:     Tcl& tcl = Tcl::instance();
3:     tcl.eval("puts\"Message From MyPrivFunc\"");
4:     tcl.evalf("puts\"      my_var1 = %d\"", my_var1);
5:     tcl.evalf("puts\"      my_var2 = %d\"", my_var2);
6: }
```

7.3 Kompilovanie, spustenie a testovanie nového agenta

- Všetky hore uvedené zdrojové kódy sa nachádzajú v súbore s názvom `priklad.cc`, ktorý je uložený v adresári `ns-2`. Do súboru `priklad.cc` je treba ešte pridať nasledujúce tri hlavičkové súbory:

  ```
  #include <stdio.h>
  #include <string.h>
  #include "agent.h"
  ```

- Do súboru `Makefile` sa pridá objektový súbor `priklad.o` až na koniec zoznamu `OBJ_CC`.

- Prevedie sa rekompilácia NS za pomoci príkazu **make**.

- Ďalej je potrebné vytvoriť testovací súbor, ktorý obsahuje testovacie OTcl príkazy. Nazvaný bude `priklad.tcl` a jeho spustenie sa uskutoční za pomoci príkazu **ns priklad.tcl**. Obsah súboru `priklad.tcl` bude nasledovný:

```
1:      set myagent [new Agent/MyagentOTcl]
2:
3:      $myagent set my_var1_otcl 2
4:      $myagent set my_var2_otcl 3.14
5:
6:      $myagent call-my-priv-func
```

7.4 Výsledok

Obr.7.2 zobrazuje výsledok testovaného skriptu. Keďže neboli nastavené predvolené hodnoty pre premenné `my_var1_otcl` a `my_var2_otcl` v adresári `ns-2/tcl/lib/ns-lib.tcl`, program vypísal výstražné hlásenie pri instancii na nový objekt.

```
warning: no class variable Agent/MyAgentOtcl::my_var1_otcl

        see tcl-object.tcl in tclcl for info about this warning

warning: no class variable Agent/MyAgentOtcl::my_var2_otcl

Message From MyPrivFunc
     my_var1 = 2
     my_var2 = 3.140000
```

Obrázek 7.2: Výsledok testovaného skriptu.

Kapitola 8

Implementácia protokolu MAODV do prostredia NS2

Táto kapitola je sústredená na implementáciu multicast smerovacieho protokolu MAODV do simulačného prostredia Network Simulator 2. Celá implementácia prebiehala pod OS **Ubuntu 9.10** a bola implementovaná do verzie simulátoru **NS-2.34**.

8.1 Inštalácia NS2

Nasledujúce riadky popisujú inštaláciu NS-2.34 v OS Ubuntu 9.10. Inštalácia bola prevedená cez terminál za pomoci nasledovných príkazov:

```
$ wget http://nchc.dl.sourceforge.net/sourceforge/nsnam/ns-allinone
  -2.34.tar.gz
$ tar -xzvf ns-allinone-2.34.tar.gz
$ sudo apt-get install g++-4.3
```

Nasledovne je potrebné otvoriť si zložku `ns-allinone/otcl-1.13` a v súboroch `Makefile` a `Makefile.in` zmeniť nasledovné:

z `CC = @CC@`
na `CC = gcc-4.3`

Ďalej sa pokračuje v terminály. Nasledujúcimi príkazmi sa prevedie presun z domovského adresára do zložky `ns-allinone-2.34`, kde bude spustená inštalácia:

```
$ cd ns-allinone-2.34
$ sudo apt-get install build-essential autoconf automake libxmu-dev
$ ./install
```

Po inštalácii sa zobrazí výzva pre definovanie ciest pre spustenie NS2 v domovskom adresári. Je nutné si uvedomiť, že `/your/path/` sa musí nahradiť cestou k domovskému adresáru kde bude simulácia spúšťaná, napr.: `/home/UserStudent/`. Do terminálu je treba zapísať nasledovný príkaz:

```
$ gedit ~/.bashrc
```

A na koniec súboru bude treba dopísať nasledovné riadky:

```
# LD_LIBRARY_PATH
  OTCL_LIB=/your/path/ns-allinone-2.34/otcl-1.13
  NS2_LIB=/your/path/ns-allinone-2.34/lib
  X11_LIB=/usr/X11R6/lib
  USR_LOCAL_LIB=/usr/local/lib
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:
  $NS2_LIB:$X11_LIB:$USR_LOCAL_LIB

# TCL_LIBRARY
  TCL_LIB=/your/path/ns-allinone-2.34/tcl8.4.18/library
  USR_LIB=/usr/lib
  export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
  XGRAPH=/your/path/ns-allinone-2.34/bin:/your/path/ns-allinone-2.34/
  tcl8.4.18/unix:/your/path/ns-allinone-2.34/tk8.4.18/unix
  NS=/your/path/ns-allinone-2.34/ns-2.34/
  NAM=/your/path/ns-allinone-2.34/nam-1.14/
  PATH=$PATH:$XGRAPH:$NS:$NAM
```

Posledným krokom je nasledujúci príkaz zapísaný do terminálu:

```
$ source ~/.bashrc
```

A ak sa postupovalo správne, po zadaní príkazu `ns` do okna terminálu sa objaví znak `%`:

```
$ ns
%
```

V prípade problémov sú návody pre inštaláciu NS2 popísané na nasledujúcich stránkach **http://www.isi.edu/nsnam/ns/** alebo **http://nsnam.isi.edu/nsnam/index.php**
Podnetom pre nasledujúce kapitoly bola literatúra [4].

8.2 Vytvorenie paketov

V kapitole *"Vlastná špecifikácia protokolu MAODV"* boli ukázané štruktúry paketov protokolu MAODV. Pre implementovanie vlastného typu paketov je potrebné vytvoriť súbor nazvaný `maodv_packet.h`, ktorý bude obsahovať dátové štruktúry, konštanty a makrá pre nové pakety. Nasledujúci kód popisuje vytvorenie paketu RREQ:

`maodv/maodv_packet.h`

```
 1: #ifndef MAODV_PACKET_H
 2: #define MAODV_PACKET_H
 3:
 4: #include <config.h>
 5: #include <common/packet.h>
```

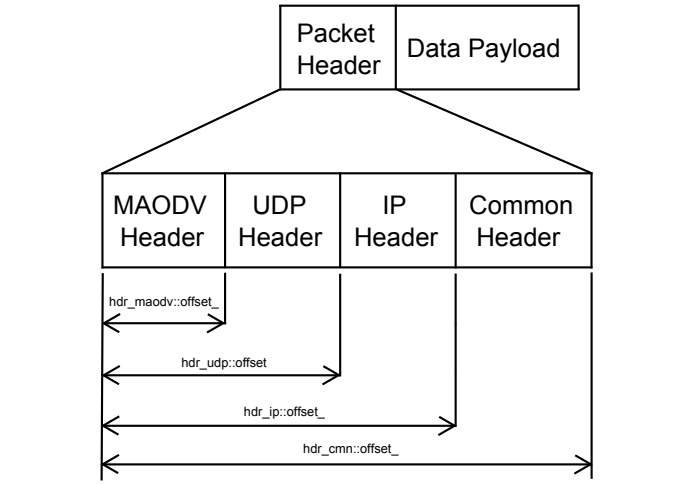
```
 6:
 7: #define HDR_MAODV(p)      ((struct hdr_maodv*)hdr_maodv::access(p))
 8: #define HDR_MAODV_RREQ(p) ((struct hdr_maodv_rreq*)hdr_maodv::access(p))
 9: #define MAODV_TYPE_RREQ   0
10:
11: struct hdr_maodv
12: {
13:    u_int8_t maodv_type;       /* Type */
14:
15:    static int offset_;
16:    inline static int &offset() { return offset_; }
17:    inline static hdr_maodv *access(const Packet *p) {
18:           return (hdr_maodv*) p->access(offset_);
19:    }
20: };
21:
22: #define RREQ_ 0
23: #define RREQ_J 1
24:
25: struct hdr_maodv_rreq
26: {
27:    u_int8_t rq_type;          /* Type */
28:    u_int8_t rq_flag;          /* Flag: -|J */
29:    u_int8_t reserved;         /* Reserved */
30:    u_int8_t rq_hop_count;     /* Hop Count */
31:    u_int32_t rq_rreq_id;      /* RREQ_ID */
32:    nsaddr_t rq_dst_addr;      /* Destination IP Address */
33:    nsaddr_t rq_src_addr;      /* Source IP Address */
34:    u_int32_t rq_src_seq_n;    /* Source Sequence Number */
35:
36:    inline int size()
37:    {
38:    int sz = 0;
39:    sz = 5 * sizeof(u_int32_t);
40:    assert (sz >= 0);
41:    return sz;
42:    }
43: };
44:
45: #endif
```

Na riadkoch $11-20$ je deklarovaná všeobecná štruktúra `hdr_maodv` pre všetky MAODV pakety. Riadky $25-43$ deklarujú štruktúru `hdr_maodv_rreq`, ktorá reprezentuje konkrétny typ správy a to RREQ. Na riadkoch $27-34$ je možné vidieť tri základné typy atribútov, ktoré paket má. Sú to:

- `nsaddr_t` – tento typ sa používa pre deklarovanie IP adries v NS2,
- `u_int8_t` – 8 bitový unsigned integer,

- `u_int32_t` – 32 bitový unsigned integer.

Všetky tieto typy i mnoho ďalších je definovaných v hlavičkovom súbore `config.h`. Riadok 5 inklúduje hlavičkový súbor `common/packet.h`, ktorý definuje triedu `Packet`. V triede `Packet` sú uložené všetky hlavičky paketov. Pre prístup ku konkrétnej hlavičke paketu je nevyhnutné definovať `offset`. `Offset` ukazuje pozíciu hlavičky každého protokolu, čo zobrazuje obr.8.1. Definovanie `offset`-u pre hlavičky paketu MAODV je na riadkoch 15−18.



Obrázek 8.1: Architektúra hlavičky paketu.

Pakety sú používané na výmenu informácii medzi objektmi v simulácii a cieľom implementácie je doplniť novú štruktúru paketov `hdr_maodv` k nim. Týmto bude možné novo vytvorené pakety odosielať a prijímať uzlami v simulácii. Ostatné typy MAODV paketov ako RREP, MACT a GRPH sú vytvorené podobne ako paket RREQ.

Na riadku 7 je definované makro pre určenie statického `offset`-u (pre všetky `hdr_maodv` štruktúry), členská funkcia pre prístup do hlavičky novo vytvoreného paketu a funkcia, ktorá vracia `hdr_maodv` v triede `Packet`.

8.3 Vytvorenie smerovacích tabuliek

8.3.1 RREQ Buffer Table

V tejto podkapitole bude ukázané vytvorenie jednoduchej smerovacej tabuľky *RREQ Buffer Table*. Každý záznam tabuľky bude uchovávať informácie pre dvojicu: *"RREQ_ID"* a *"zdrojová adresa"*. Tabuľka *RREQ Buffer Table* slúži pre kontrolu prijatých správ RREQ. Ak sa v tabuľke nachádza záznam o prijatej správe s rovnakou dvojicou *"RREQ_ID"* a *"zdrojová adresa"*, správa sa zahodí.

Implementácia funkcii je pomerne jednoduchá. Konštruktor bude obsahovať iba dve premenné a to *"RREQ_ID"* a *"zdrojová adresa"*. Jeho volanie v súbore `maodv_rreq_buffer.cc` bude nasledovné:

`maodv/maodv_rreq_buffer.cc`

```
 1: rreq_buffer_entry::rreq_buffer_entry(u_int8_t id, nsaddr_t src)
 2: {
 3:       this->buff_rreq_id = id;       /* RREQ_ID */
 4:       this->buff_src = src;          /* source IP address */
 5: };
```

Pri prijatí paketu RREQ sa zavolá funkcia `buff_add()`, ktorá najskôr za pomoci funkcie `buff_lookup()` kontroluje či sa daný paket v tabuľke už nenachádza. Telo funkcie `buff_lookup()` je nasledovné:

`maodv/maodv_rreq_buffer.cc`

```
 1: u_int8_t rreq_buffer_table::buff_lookup(u_int8_t id, nsaddr_t src)
 2: {
 3:     rreq_buffer_entry *buff = buffhead.lh_first;
 4:
 5:     for(; buff; buff = buff->buff_link.le_next)
 6:     {
 7:          if (buff->getBuffRreqId() == id && buff->getBuffSrc() == src)
 8:          {
 9:               printf("Record id=%d & src=%d already exists!\n",
10:               buff->getBuffRreqId(), buff->getBuffSrc());
11:               return BUFF_ENTRY_FOUND;
12:          }
13:     }
14:     return BUFF_ENTRY_NOT_FOUND;
15: }
```

Riadok 7 obsahuje podmienku pre vyhľadanie záznamu v tabuľke. Ak podmienka prejde a funkcia `buff_lookup()` vráti BUFF_ENTRY_FOUND, príslušná správa RREQ sa zahodí. V opačnom prípade, ak funkcia vráti BUFF_ENTRY_NOT_FOUND, znamená to, že daný záznam sa nenašiel a správa môže byť spracovaná ďalej. Vo funkcii `buff_add()` bude splnená podmienka nenájdenia záznamu v tabuľke, a tak sa vytvorí záznam o prijatej správe RREQ. Funkcia `buff_add()` vyzerá nasledovne:

`maodv/maodv_rreq_buffer.cc`

```
 1: u_int8_t rreq_buffer_table::buff_add(u_int8_t id, nsaddr_t src)
 2: {
 3:      if (this->buff_lookup(id, src) == BUFF_ENTRY_NOT_FOUND)
 4:      {
 5:           rreq_buffer_entry *buff = new rreq_buffer_entry(id, src);
 6:           assert(buff);
 7:           LIST_INSERT_HEAD(&buffhead, buff, buff_link);
 8:           return TRUE;
 9:      }
10:      else
11:      {
12:           return FALSE;
```

```
13:      }
14: }
```

Na riadku 5 sa volá konštruktor pre vytvorenie záznamu do tabuľky *RREQ Buffer Table*. Pre kontrolu funkcia vracia TRUE/FALSE o tom či bolo možné záznam vytvoriť.

8.3.2 Routing Table

Smerovacia tabuľka *Routing Table* v sebe nesie záznamy o susedných uzloch. Pre pridanie a aktualizovanie si záznamu o uzle, od ktorého správa prišla, slúži funkcia `rt_add()`. Tá si pred pridaním záznamu overuje jeho prítomnosť v smerovacej tabuľke za pomoci funkcie `rt_lookup_src()`.

`maodv/maodv_rtable.cc`

```
 1: u_int8_t maodv_rtable::rt_lookup_src(nsaddr_t src, u_int32_t seq_n,
 2:                                      u_int8_t hop_count)
 3: {
 4:     maodv_rt_entry *rt = rthead.lh_first;
 5:     for (; rt; rt = rt->rt_link.le_next)
 6:     {
 7:          if (rt->getRtDstAddr() != src)
 8:          {
 9:              continue;
10:          }
11:          if (rt->getRtDstAddr() == src)
12:          {
13:              if (rt->getRtDstSeqN() < seq_n &&
14:                  rt->getRtHopCount() > hop_count)
15:              {
16:                  rt->setRtDstSeqN(seq_n);       // upgrade seq_n
17:                  rt->setRtHopCount(hop_count);  // upgrade hop_count
18:                  return RT_ENTRY_UPGRADE;
19:              }
20:              else if(rt->getRtDstSeqN() < seq_n)
21:              {
22:                  rt->setRtDstSeqN(seq_n);       // upgrade seq_n
23:                  return RT_ENTRY_UPGRADE;
24:              }
25:              else if(rt->getRtHopCount() > hop_count)
26:              {
28:                  rt->setRtHopCount(hop_count);  // upgrade hop_count
29:                  return RT_ENTRY_UPGRADE;
30:              }
31:              return RT_ENTRY_FOUND;
32:          }
33:     }
34:     return RT_ENTRY_NOT_FOUND;
35: }
```

Hľadaný záznam sa v tabuľke vyhľadáva za pomoci IP adresy hľadaného uzla. Ak sa záznam v tabuľke nachádza (riadok 11), môžu nastať dva prípady spracovania nových informácii o uzle. V prvom prípade, keď informácia o sekvenčnom čísle v prijatej správe je vyššia, bude splnená podmienka na riadkoch $20 - 24$. Alebo ak informácia o počte skokov bude nižšia, bude splnená podmienka na riadkoch $25 - 30$. Ak správa obsahuje lepšie informácie o sekvenčnom čísle i o počte skokov, bude splnená podmienka na riadkoch $13 - 19$. Funkcia v prvom prípade vráti RT_ENTRY_UPGREAT. V druhom prípade sa neaktua-lizujú záznamy, pretože sú vyhodnotené ako rovnaké alebo horšie. Funkcia v druhom prípade vráti RT_ENTRY_FOUND. V prípade ak sa záznam v tabuľke nenachádza, funkcia vráti RT_ENTRY_NOT_FOUND a bude splnená podmienka pre pridanie záznamu do tabuľky za pomoci funkcie `rt_add()`.

`maodv/maodv_rtable.cc`

```
 1: u_int8_t maodv_rtable::rt_add(nsaddr_t src, u_int32_t seq_n,
 2:                               u_int8_t hop_count)
 3: {
 4:    if(this->rt_lookup_src(src, seq_n, hop_count) == RT_ENTRY_NOT_FOUND)
 5:    {
 6:         maodv_rt_entry *rt = new maodv_rt_entry(src, seq_n, hop_count);
 7:         assert(rt);
 8:         LIST_INSERT_HEAD(&rthead, rt, rt_link);
 9:         return TRUE;
10:    }
11:    else
12:    {
13:         return FALSE;
14:    }
15: }
```

Uzol môže preposlať správu ďalej iba v prípade, ak má uložený záznam o cieľovom uzle. Ak záznam nemá, odošle správu RREQ. Pri preposielaní správy uzol prehľadáva svoju smerovaciu tabuľku . Pre tento účel slúži funkcia `rt_lookup_t()`.

`maodv/maodv_rtable.cc`

```
 1: maodv_rt_entry *maodv_rtable::rt_lookup_t(nsaddr_t dst)
 2: {
 3:    maodv_rt_entry *rt = rthead.lh_first;
 4:    for (; rt; rt = rt->rt_link.le_next)
 5:    {
 6:        if (rt->getRtFlag() == TRUE && rt->getRtDstAddr() == dst)
 7:        {
 8:             printf("I've found node(%d).\n", rt->getRtDstAddr());
 9:             return RT_ENTRY_FOUND;
10:        }
11:    }
12:    printf("Entry not found!\n");
13:    return RT_ENTRY_NOT_FOUND;
```

```
14: }
```

Vyhľadávať sa budú iba záznamy, ktoré majú nastavený príznak TRUE. Záznamy s príznakom FALSE nie sú aktuálne, pretože im exspiroval časovač. Ak sa hľadaný záznam v tabuľke nachádza, funkcia vráti RT_ENTRY_FOUND a správa môže byť preposlaná ďalej. V opačnom prípade ak sa záznam v smerovacej tabuľke nenachádza, vráti funkcia RT_ENTRY_NOT_FOUND. Riadky 8 a 11 slúžia pre účely testovania. Testovanie súborov bude popísané v kapitole *"Testovanie protokolu MAODV"*.

Aby smerovacia tabuľka neniesla v sebe neaktualizované informácie, je v nej nadefinovaná funkcia `rt_delete()`, ktorá slúži pre vymazanie takýchto záznamov. Tieto záznamy majú nastavený príznak FALSE. Neaktualizované záznamy vznikajú exspiráciou časovačov. Časovač pre daný záznam sa obnoví v prípade ak uzol prijme správu od uzla, o ktorom je záznam vedený. Pre vyhľadanie exspirovaných záznamov slúži funkcia `rtdelete_lookup()`. Obidve funkcie sú popísané nasledovne:

`maodv/maodv_rtable.cc`

```
 1: maodv_rt_entry* maodv_rtable::rtdelete_lookup(nsaddr_t dst)
 2: {
 3:    maodv_rt_entry *rt = rthead.lh_first;
 4:    for (; rt; rt = rt->rt_link.le_next)
 5:    {
 6:       if (rt->getRtDstAddr() == dst) { break; }
 7:    }
 8:    return rt;
 9: }
10:
11: void maodv_rtable::rt_delete(nsaddr_t dst)
12: {
13:    maodv_rt_entry *rt = rtdelete_lookup(dst);
14:    if (rt->getRtFlag() == FALSE)
15:    {
16:       if (rt)
17:       {
18:          LIST_REMOVE(rt, rt_link);
19:          delete rt;
20:       }
21:    }
22: }
```

8.3.3 Multicast Membership Table

V tabuľke *Multicast Membership Table* sa nachádzajú záznamy o tom, do ktorých multicast skupín príslušný uzol patrí. Uzol môže odpovedať na správu RREQ-J iba v prípade, ak je členom hľadanej multicast skupiny. Uzol teda na základe tejto tabuľky zisťuje či môže na správu RREQ-J odpovedať a taktiež prijímať správy pre príslušnú multicast skupinu.

Pre pridanie záznamu do tabuľky *Multicast Membership Table* slúži funkcia `my_add()`. Táto funkcia volá funkciu `my_lookup()`, ktorá prehľadáva obsah tabuľky. Ak funkcia vráti

MT_ENTRY_NOT_FOUND, uzol si vytvorí záznam do *Multicast Membership Table*. Telo funkcie `my_add()` je popísané nasledovne:

`maodv/maodv_mtable.cc`

```
 1: u_int8_t myMt_table::my_add(nsaddr_t grp_addr, u_int32_t grp_seq_n,
 2:                             nsaddr_t leader_addr, nsaddr_t src_addr)
 3: {
 4:    if (this->my_lookup(grp_addr) == MT_ENTRY_NOT_FOUND)
 5:    {
 6:       myMt_entry *my = new myMt_entry(grp_addr, grp_seq_n,
 7:                                       leader_addr, src_addr);
 8:       assert(my);
 9:       LIST_INSERT_HEAD(&myhead, my, myMt_link);
10:       return TRUE;
11:    }
12:    else
13:    {
14:       return FALSE;
15:    }
16: }
```

Opačný význam funkcie `my_lookup()` nastáva v prípade, ak uzol zisťuje či môže odpovedať na správu RREQ-J a funkcia vracia MT_ENTRY_FOUND. V tomto prípade ak sa záznam v tabuľke nachádza, odpovie uzol tázanému uzlu správou RREP-M. Obsah funkcie `my_lookup()` zobrazuje nasledovný kód:

`maodv/maodv_mtable.cc`

```
 1: u_int8_t myMt_table::my_lookup(nsaddr_t grp_addr)
 2: {
 3:    myMt_entry *my = myhead.lh_first;
 4:    for (; my; my = my->myMt_link.le_next)
 5:    {
 6:       if (my->getMyMtGrpAddr() == grp_addr)
 7:       {
 8:           printf("You can send RREP-H.");
 9:           return MT_ENTRY_FOUND;
10:       }
11:    }
12:    printf("You can not answer for a~RREQ-J!");
13:    return MT_ENTRY_NOT_FOUND;
14: }
```

Pre vyhľadanie a získanie záznamu z tabuľky *Multicast Membership Table* slúži funkcia `mynew_lookup()`. Používa sa napríklad v prípade prijatia niekoľkých správ RREP-M, kedy uzol vyberá tú správu, ktorá má najväčšie sekvenčné číslo. Alebo v prípade prijatia správy RREP-J, kedy sa táto funkcia volá pre potreby vytvorenia záznamu o novom členovi multicast skupiny do tabuľky *Multicast Routing Table*. Funkcia `mynew_lookup()` vyzerá nasledovne:

maodv/maodv_mtable.cc

```
 1: myMt_entry* myMt_table::mynew_lookup(nsaddr_t grp_addr)
 2: {
 3:     myMt_entry *my = myhead.lh_first;
 4:     for (; my; my = my->myMt_link.le_next)
 5:     {
 6:         if (my->getMyMtGrpAddr() == grp_addr)
 7:         {
 8:             break;
 9:         }
10:     }
11:     return my;
12: }
```

Ak je uzol členom multicast skupiny a prijme správu GRPH, svoj záznam v tabuľke *Multicast Membership Table* si aktualizuje za pomoci funkcie `my_lookup_grph()`.

maodv/maodv_mtable.cc

```
 1: u_int8_t myMt_table::my_lookup_grph(nsaddr_t grp_addr,
 2:                                     u_int32_t grp_seq_n)
 3: {
 4:     myMt_entry *my = myhead.lh_first;
 5:     for (; my; my = my->myMt_link.le_next)
 6:     {
 7:         if (my->getMyMtGrpAddr() != grp_addr)
 8:         {
 9:             continue;
10:         }
11:         if (my->getMyMtGrpAddr() == grp_addr)
12:         {
13:             if (my->getMyMtSeqN() < grp_seq_n)
14:             {
15:                 my->setMyMtSeqN(grp_seq_n);
16:                 return MT_ENTRY_UPGRADE;
17:             }
18:             return MT_ENTRY_FOUND;
19:         }
20:     }
21:     return MT_ENTRY_NOT_FOUND;
22: }
```

V prípade, kedy chce uzol vystúpiť z multicast skupiny, je potrebné vymazať príslušný záznam v tabuľke *Multicast Membership Table*. Pre tento účel slúži funkcia `my_delete()`, ktorá využíva pre vyhľadenie záznamu funkciu `mydel_lookup()`.

maodv/maodv_mtable.cc

```
 1: myMt_entry* myMt_table::mydel_lookup(nsaddr_t grp_addr)
 2: {
 3:   myMt_entry *my = myhead.lh_first;
 4:   for (; my; my = my->myMt_link.le_next)
 5:   {
 6:       if (my->getMyMtGrpAddr() == grp_addr)
 7:       {
 8:           break;
 9:       }
10:   }
11:   return my;
12: }
13:
14: void myMt_table::my_delete(nsaddr_t grp_addr)
15: {
16:    myMt_entry *my = mydel_lookup(grp_addr);
17:    if (my)
18:    {
19:       LIST_REMOVE(my, myMt_link);
20:       delete my;
21:    }
22: }
```

8.3.4 Multicast Routing Table

Tabuľka *Multicast Routing Table* slúži pre uchovávanie si záznamov o členoch multicast skupín. Uzol, člen multicast skupiny, ktorý môže odpovedať na správu RREQ-J, si vytvorí záznam o budúcom členovi multicast skupiny za pomoci funkcie `mt_add_rreq()`. Táto funkcia volá funkciu `my_lookup()` z triedy `maodv_mtable` pre spomínané overenie členstva príslušného uzlu v hľadanej multicast skupine. Ak sa uzol v hľadanej multicast skupine nachádza, vytvorí si záznam o potenciálnom členovi do tabuľky *Multicast Routing Table* a odošle správu RREP-M. Potenciálnom preto, lebo záznam bude uložený s príznakom FALSE do času, pokým uzol neprijme správu MACT-J. Funkcia `mt_add_rreq()` vyzerá nasledovne:

`maodv/maodv_mtable.cc`

```
 1: u_int8_t maodv_mtable::mt_add_rreq(nsaddr_t grp_addr, u_int32_t grp_seq_n,
 2:          nsaddr_t member, u_int8_t hop_count, nsaddr_t leader_addr)
 3: {
 4:    if (parent->getMyMtTable()->my_lookup(grp_addr) == MT_ENTRY_FOUND)
 5:    {
 6:       maodv_mt_entry *mt = new maodv_mt_entry(grp_addr, grp_seq_n, member,
 7:                                               hop_count, leader_addr);
 8:       assert(mt);
 9:       LIST_INSERT_HEAD(&mthead, mt, mt_link);
10:       return TRUE;
11:    }
12:    else
```

```
13:    {
14:       return FALSE;
15:    }
16: }
```

Medziľahlý uzol prijímajúci správu RREP-M si vytvorí záznam o potenciálnom členovi multicast skupiny za pomoci funkcie `mt_add_rrep()`, ktorá volá funkciu `mt_lookup_rrep()` pre kontrolu existencie a aktualizácie záznamu v tabuľke. Ak medziľahlý uzol prijme rovnakú správu RREP-M od viacerých členov multicast skupiny, bude záznam o potenciálnom členovi aktualizovaný správou s tými najlepšími parametrami. Nasledujúce riadky opisujú telo funkcie `mt_lookup_rrep()`:

maodv/maodv_mtable.cc

```
 1: u_int8_t maodv_mtable::mt_lookup_rrep(nsaddr_t grp_addr,
 2: u_int32_t grp_seq_n, nsaddr_t member, u_int8_t hop_count)
 3: {
 4:    maodv_mt_entry *mt = mthead.lh_first;
 5:    for (; mt; mt = mt->mt_link.le_next)
 6:    {
 7:       if (mt->getMtGrpAddr() != grp_addr && mt->getMtMember() != member)
 8:       {
 9:           continue;
10:       }
11:       if (mt->getMtGrpAddr() == grp_addr && mt->getMtMember() == member)
12:       {
14:           if (mt->getMtGrpSeqN() < grp_seq_n &&
15:           mt->getMtHopCount() > hop_count)
16:           {
17:               mt->setMtGrpSeqN(grp_seq_n);    // upgrade grp_seq_n
18:               mt->setMtHopCount(hop_count);   // upgrade hop_count
19:               return MT_ENTRY_UPGRADE;
20:           }
21:           if (mt->getMtGrpSeqN() < grp_seq_n)
22:           {
23:               mt->setMtGrpSeqN(grp_seq_n);    // upgrade grp_seq_n
24:               return MT_ENTRY_UPGRADE;
25:           }
26:           if (mt->getMtHopCount() > hop_count)
27:           {
28:               mt->setMtHopCount(hop_count);   // upgrade hop_count
29:               return MT_ENTRY_UPGRADE;
30:           }
31:           return MT_ENTRY_FOUND;
32:       }
33:    }
34:    return MT_ENTRY_NOT_FOUND;
35: }
```

Na riadku 11 sa vyhľadaný záznam aktualizuje splnením jednej z podmienok popísaných na riadkoch 14 – 30. Ak ku aktualizácii nedôjde, záznam zostane nezmenený a funkcia `mt_lookup_rrep()` vráti MT_ENTRY_FOUND. Ak záznam neexistuje (riadok 7), vráti funkcia `mt_lookup_rrep()` MT_ENTRY_NOT_FOUND a za pomoci funkcie `mt_add_rrep()` sa vytvorí záznam do tabuľky *Multicast Routing Table*, ktorý bude mať nastavený príznak FALSE až pokým uzol neprijme správu MACT-J od uzla, ktorému je správa RREP-M určená. Funkcia `mt_add_rrep()` pre pridanie záznamu vyzerá nasledujúco:

`maodv/maodv_mtable.cc`

```
 1: u_int8_t maodv_mtable::mt_add_rrep(nsaddr_t grp_addr, u_int32_t grp_seq_n,
 2: nsaddr_t member, u_int8_t hop_count, nsaddr_t leader_addr)
 3: {
 4:    if (this->mt_lookup_rrep(grp_addr, grp_seq_n,
 5:    member, hop_count) == MT_ENTRY_NOT_FOUND)
 6:    {
 7:        maodv_mt_entry *mt = new maodv_mt_entry(grp_addr, grp_seq_n,
 8:                                                member, leader_addr);
 9:        assert(mt);
10:        LIST_INSERT_HEAD(&mthead, mt, mt_link);
11:        return TRUE;
12:    }
13:    else
14:    {
15:        return FALSE;
16:    }
17: }
```

Uzol, pre ktorého je správa RREP-M určená si vyberie tú správu, ktorá má najväčšie sekvenčné číslo a najmenší počet skokov. Pre tento účel slúži funkcia `mt_add_target()`, ktorá volá funkciu `mt_lookup_target()` aktualizujúcu záznam o multicast skupine. Ak funkcia `mt_lookup_target()` vráti MT_ENTRY_NOT_FOUND, uzol si vytvorí záznam o členovi multicast skupiny, od ktorého správu prijal do svojej *Multicast Routing Table*. Funkcia `mt_lookup_target()` sa odlišuje od funkcie `mt_lookup_rrep()` iba v nastavení príznaku pre záznam na TRUE. Po vytvorení a aktualizovaní si záznamu o multicast skupine vytvorí uzol správu MACT-J, čím potvrdí členstvo k multicast skupine a aktivuje cestu v sieti. Nasledujúce riadky opisujú funkciu `mt_add_target`):

`maodv/maodv_mtable.cc`

```
 1: u_int8_t maodv_mtable::mt_add_target(nsaddr_t grp_addr,
 2: u_int32_t grp_seq_n,  nsaddr_t member, u_int8_t hop_count,
 3: nsaddr_t leader_addr)
 4: {
 5:     if (this->mt_lookup_rrep(grp_addr, grp_seq_n, member,
 6:     hop_count) == MT_ENTRY_NOT_FOUND)
 7:     {
 8:         maodv_mt_entry *mt = new maodv_mt_entry(grp_addr, grp_seq_n,
 9:         member, hop_count, leader_addr);
```

```
10:         assert(mt);
11:         LIST_INSERT_HEAD(&mthead, mt, mt_link);
12:         return TRUE;
13:     }
14:     else
15:     {
16:         return FALSE;
17:     }
18: }
```

Medziľahlý i cieľový uzol prijímajúci správu MACT-J si za pomoci funkcie `mt_lookup_mact()` vyhľadajú záznam o uzle, ktorý chce touto správou aktivovať trasu (vetvu stromu) ku multicast skupine. Ak funkcia vráti MT_ENTRY_NOT_FOUND, záznam sa teda nenašiel, správa sa ignoruje a zahodí. V opačnom prípade si uzol aktualizuje záznam v tabuľke *Multicast Routing Table* pre daný uzol a nastaví tomuto záznamu príznak TRUE a aktuálny počet skokov. Od tejto chvíle sa uzol stáva právoplatným členom multicast skupiny. Funkcia `mt_lookup_mact()` je popísaná nasledovne:

`maodv/maodv_mtable.cc`

```
 1: u_int8_t maodv_mtable::mt_lookup_mact(nsaddr_t grp_addr, nsaddr_t member,
 2: u_int8_t hop_count)
 3: {
 4:     maodv_mt_entry *mt = mthead.lh_first;
 5:     for (; mt; mt = mt->mt_link.le_next)
 6:     {
 7:         if (mt->getMtFlag() == FALSE && mt->getMtGrpAddr() == grp_addr &&
 8:         mt->getMtMember() == member)
 9:         {
10:             mt->setMtHopCount(hop_count);   // upgade hop_count
11:             mt->setMtFlag(TRUE);            // upgrade flag
12:             return MT_ENTRY_UPGRADE;
13:         }
14:         else
15:         {
16:             return MT_ENTRY_NOT_FOUND;
17:         }
18:     }
19: }
```

Ak sa člen multicast skupiny od príslušnej skupiny odpojí (odošle správu MACT-P), nie je potrebné o ňom naďalej viesť záznam v tabuľke *Multicast Routing Table*. Pre vymazanie záznamu z tabuľky slúži funkcia `mt_delete()`, ktorá za pomoci funkcie `mtdel_lookup()` vyhľadá vymazávaný záznam. Obe funkcie sú popísané na nasledujúcich riadkoch:

`maodv/maodv_mtable.cc`

```
 1: maodv_mt_entry* maodv_mtable::mtdel_lookup(nsaddr_t grp_addr,
 2:                                            nsaddr_t member) {
```

```
 3:     maodv_mt_entry *mt = mthead.lh_first;
 4:     for (; mt; mt = mt->mt_link.le_next)
 5:     {
 6:          if (mt->getMtGrpAddr() == grp_addr &&
 7:              mt->getMtMember() == member) {
 8:              break;
 9:          }
10:     }
11:     return mt;
12: }
13:
14: void maodv_mtable::mt_delete(nsaddr_t grp_addr, nsaddr_t member)
15: {
16:    maodv_mt_entry *mt = mtdel_lookup(grp_addr, member);
19:    if (mt)
20:    {
21:        LIST_REMOVE(mt, mt_link);
22:        delete mt;
23:    }
25: }
```

Prijatím správy GRPH si uzol aktualizuje záznamy pre príslušnú multicast skupinu v tabuľke *Multicast Routing Table* za pomoci funkcie `mt_lookup_grph()`.

`maodv/maodv_mtable.cc`

```
 1: u_int8_t maodv_mtable::mt_lookup_grph(nsaddr_t grp_addr,
 2: u_int32_t grp_seq_n)
 3: {
 4:     maodv_mt_entry *mt = mthead.lh_first;
 5:     for (; mt; mt = mt->mt_link.le_next)
 6:     {
 7:         if (mt->getMtGrpAddr() != grp_addr)
 8:         {
 9:             continue;
10:         }
11:         if (mt->getMtGrpAddr() == grp_addr)
12:         {
13:             if (mt->getMtGrpSeqN() < grp_seq_n)
14:             {
15:                 mt->setMtGrpSeqN(grp_seq_n);
16:                 return MT_ENTRY_UPGRADE;
17:             }
18:             return MT_ENTRY_FOUND;
19:         }
20:     }
21:     return MT_ENTRY_NOT_FOUND;
22: }
```

Kapitola 9

Testovanie protokolu MAODV

Pre overenie funkčnosti naprogramovaného kódu protokolu bol vytvorený súbor `test.cc`. V tejto kapitole bude popísaný jeho obsah zvlášť pre testovanie jednotlivých súborov. Pre skompilovanie a preloženie zdrojových súborov do binárnej podoby spustiteľného programu bol vytvorený súbor `Makefile`, jeho obsah je nasledovný:

`test/test.cc`

```
 1: CXX=g++
 2: CXXFLAGS=-std=c++98 -Wall
 3: BIN=test
 4:
 5: OBJ= \
 6:      ../ns-allinone-2.34/ns-2.34/maodv/node.o \
 7:      ../ns-allinone-2.34/ns-2.34/maodv/maodv_rreq_buffer.o \
 8:      ../ns-allinone-2.34/ns-2.34/maodv/maodv_mtable.o \
 9:      ../ns-allinone-2.34/ns-2.34/maodv/maodv_rtable.o \
10:      ../ns-allinone-2.34/ns-2.34/maodv/maodv_packet.o
11:
12: install: all
13: all:     $(BIN)
14: test:   test.o
15:         $(CXX) test.o $(OBJ) -o $(BIN)
16: test.o: test.cc
17:         $(CXX) $(CXXFLAGS) -o test.o -c test.cc
18: clean:
19:         rm -f test.o $(BIN)
```

Pre zjednodušenie bude ďalej popísaný testovaný kód jednotlivých súborov už bez nasledujúcich riadkov, ktoré bude potrebné vložiť na začiatok každého súboru `test.cc`:

```
1: #include <stdio.h>
2: #include "../ns-allinone-2.34/ns-2.34/maodv/maodv_rreq_buffer.h"
3: #include "../ns-allinone-2.34/ns-2.34/maodv/node.h"
4: #include "../ns-allinone-2.34/ns-2.34/maodv/maodv_mtable.h"
5: #include "../ns-allinone-2.34/ns-2.34/maodv/maodv_rtable.h"
6: #include "../ns-allinone-2.34/ns-2.34/maodv/maodv_packet.h"
```

9.1 Testovanie RREQ Buffer Table

Ako bolo spomenuté už v prechádzajúcich kapitolách, tabuľka *RREQ Buffer Table* slúži pre zamietnutie prijatia správ RREQ, ktoré už uzol prijal. Týmto sa ušetrí výpočtový výkon uzlu, pretože sa nebude zaoberať spracovaním už prijatého paketu a zabezpečí sa priepustnosť siete, pretože správa RREQ nebude preposielaná ďalej medzi uzlami.

Nasledujúci kód testuje funkčnosť tabuľky *RREQ Buffer Table*:

`test/test.cc`

```
 7: int main(int argc, char **argv)
 8: {
 9:      node* myNode = new node(1, 0, 0);
10:      rreq_buffer_table *buffer = new rreq_buffer_table(myNode);
11:
12:      buffer->buff_add(2, 2);
13:      buffer->buff_add(3, 4);
14:        buffer->show_table();
15:      buffer->buff_add(2, 2);
16:        buffer->show_table();
17: }
```

Na riadku 9 sa najskôr vytvorí nový objekt `myNode` triedy `node`, ktorý bude mať za pomoci konštruktoru `node(1, 0, 0)` nastavenú *IP adresu* na hodnotu "1"a *sekvenčné číslo* spolu s poradovým číslom *RREQ_ID* na hodnotu "0". Riadok 10 zobrazuje vytvorenie objektu `buffer` triedy `rreq_buffer_table`, ktorá umožňuje prístup do tabuľky uzla `myNode`. Na riadkoch 12,13 a 15 sa volá funkcia `buff_add()`, ktorá prijíma dva celočíselné parametre. Prvým je poradové číslo RREQ_ID a druhým IP adresa zdroja správy. Ako bolo popísané v kapitole *"Implementácia protokolu MAODV do prostredia NS2"*, funkcia `buff_add()` volá funkciu `buff_lookup()`, ktorá overuje či sa daný záznam v tabuľke už nenachádza. Ak funkcia `buff_lookup()` vráti BUFF_ENTRY_NOT_FOUND, správa bude spracovaná ďalej a záznam o jej prijatí sa uloží do tabuľky. Riadok 15 predstavuje znovu prijatie správy s poradovým číslom RREQ_ID "2"a IP adresou "2". Keďže uzol už túto správu prijal, záznam o prijatej správe sa do tabuľky nepridá a uzol správu zahodí. Pre zobrazenie záznamov v tabuľke sa volá na riadkoch 14 a 16 funkcia `show_table()`. Výstup testu je možné vidieť na obr.9.1.

```
1:  +--------RREQ-Buffer-Table--------+ 9:  Entry with id=2 & src=2 already exists!
2:  +--------+------------------------+ 10:
3:  |   ID   |          src           | 11: +--------RREQ-Buffer-Table--------+
4:  +--------+------------------------+ 12: +--------+------------------------+
5:  |   3    |          4             | 13: |   ID   |          src           |
6:  +--------+------------------------+ 14: +--------+------------------------+
7:  |   2    |          2             | 15: |   3    |          2             |
8:  +--------+------------------------+ 16: +--------+------------------------+
                                        17: |   2    |          1             |
                                        18: +--------+------------------------+
```

Obrázek 9.1: Výstup testu pre tabuľku RREQ Buffer Table.

9.2 Testovanie Routing Table

`test/test.cc`

```
 7: int main(int argc, char **argv)
 8: {
 9:      node* myNode = new node(1, 0, 0);
10:      maodv_rtable* rtable = new maodv_rtable(myNode);
11:
12:      rtable->rt_add(2, 3, 2);
13:      rtable->rt_add(3, 1, 3);
14:        rtbale->show_rtable();
15:      rtable->rt_add(2, 4, 2);
16:      rtable->rt_add(3, 1, 3);
17:        rtable->show_rtable();
18:      rtable->rt_lookup_t(3);
19:      rtable->rt_lookup_t(5);
20: }
```

Podobne ako u tabuľky *RREQ Buffer Table* sa na riadku 9 vytvorí nový objekt `myNode` triedy `node`, ktorý bude mať za pomoci konštruktoru `node(1, 0, 0)` nastavenú *IP adresu* na hodnotu "1"a *sekvenčné číslo* spolu s poradovým číslom *RREQ_ID* na hodnotu "0". Riadok 10 zobrazuje vytvorenie objektu `rtable` triedy `maodv_rtable`, ktorá má umožňuje prístup do tabuľky uzla `myNode`. Na riadkoch 12 a 13 bude testovaná funkcia `rt_add()`, ktorá slúži pre pridanie a aktualizovanie záznamov o uzloch. Prvé číslo v tele funkcie `rt_add()` predstavuje *IP adresu* hľadaného uzla, potom nasleduje *sekvenčné číslo* a za ním *počet skokov*. Keďže tabuľka je na začiatku testovania prázdna, oba záznamy sa pridajú. Funkcia `show_rtable()` slúži pre zobrazenie obsahu smerovacej tabuľky. Riadok 15 predstavuje prijatie správy od uzla s *IP adresou* "2", so *sekvenčným číslom* "4"a s rovnakým *počtom skokov* "2". Keďže záznam zo správy nesie vyššie sekvenčné číslo ako záznam uložený v smerovacej tabuľke, bude príslušný záznam aktualizovaný. Riadok 16 predstavuje prijatie správy od uzlu "3"s tými istými parametrami ako predtým. Nenastane teda žiadna zmena a správa sa zahodí. Na riadkoch 18 a 19 sa bude testovať funkcia `rt_lookup_t()`, ktorá slúži pre vyhľadanie záznamu o uzle, ktorému má byť prijatá správa preposielaná. Ak uzol má uložený záznam o cieľovom uzle, môže preposlať správu ďalej. Ak záznam nemá, odošle správu RREQ pre vyhľadanie uzla v sieti. Výsledok testovania zobrazuje obr.9.2.

```
 1:  +---------------Routing-Table---------------+ 10: +--------------Routing-Table---------------+
 2:  +----------+-------+---------+------+------+ 11: +----------+-------+---------+------+------+
 3:  | dst_addr | seq_n |hop_count| cost | flag | 12: | dst_addr | seq_n |hop_count| cost | flag |
 4:  +----------+-------+---------+------+------+ 13: +----------+-------+---------+------+------+
 5:  |    3     |   1   |    3    |  255 |  1   | 14: |    3     |   1   |    3    |  255 |  1   |
 6:  +----------+-------+---------+------+------+ 15: +----------+-------+---------+------+------+
 7:  |    2     |   3   |    2    |  255 |  1   | 16: |    2     |   4   |    2    |  255 |  1   |
 9:  +----------+-------+---------+------+------+ 17: +----------+-------+---------+------+------+
                                                  18:
                                                  19: I've found node(3).
                                                  20: Entry not found!
```

Obrázek 9.2: Výstup testu pre tabuľku Routing Table.

9.3 Testovanie Multicast Membership Table

test/test.cc

```
 7: int main(int argc, char **argv)
 8: {
 9:       node* myNode = new node(1, 0, 0);
10:       myMt_table *my = new myMt_table(myNode);
11:
12:       my->my_add(5, 1, 9);
13:        my->show_mytable();
14:       my->my_add(5, 1, 9);
15:        my->show_mytable();
16:       my->my_lookup(5);
17:       my->my_lookup(2);
18:       my->my_delete(5);
19:        my->show_mytable();
20: }
```

Tabuľka *Multicast Membership Table* bude testovaná nasledovne. Najskôr sa vytvorený uzol `myNode` pridá do multicast skupiny za pomoci funkcie `my_add()` (riadok 12), kde prvá číslica definuje IP adresu multicast skupiny "5", potom nasleduje sekvenčné číslo multicast skupiny "1"a nakoniec IP adresa lídra skupiny multicast "9". Zavolaním funkcie `show_mytable()` na riadku 13 sa overí pridanie záznamu do tabuľky. Riadkom 14 sa testuje správnosť funkčnosti funkcie `my_add()`, ktorá pri nájdení už existujúceho záznamu prijatú správu zahodí. Ako vidieť z výstupu uvedeného nižšie, záznam v tabuľke sa nezmení. Na riadkoch 16 a 17 sa testuje, či môže uzol odpovedať na prijatú správu RREQ-J. Uzol môže odpovedať na správu RREQ-J iba v prípade ak je členom hľadanej multicast skupiny. V prípade, že sa uzol chce odpojiť od multicast skupiny, je potrebné vymazať záznam pre príslušnú multicast skupinu. Testovanie funkcie `my_delete()` prebieha na riadku 18. Zavolaním funkcie `show_mytable()` na riadku 19 sa overí zmazanie príslušného záznamu. Výsledok testovania zobrazuje obr.9.3.

```
 1: +-------Multicast-Membership-Table------+ 13: You can send RREP-H.
 2: +----------+---------+------------------+ 14: You can not answer for a RREQ-J!
 3: | grp_addr |  seq_n  |    leader_addr   | 15:
 4: +----------+---------+------------------+ 16: +-------Multicast-Membership-Table------+
 5: |    5     |    1    |       9          | 17: +----------+---------+------------------+
 6: +----------+---------+------------------+ 18: | grp_addr |  seq_n  |    leader_addr   |
 7: +-------Multicast-Membership-Table------+ 19: +----------+---------+------------------+
 8: +----------+---------+------------------+
 9: | grp_addr |  seq_n  |    leader_addr   |
10: +----------+---------+------------------+
11: |    5     |    1    |       9          |
12: +----------+---------+------------------+
```

Obrázek 9.3: Výstup testu pre tabuľku Multicast Membership Table.

9.4 Testovanie Multicast Routing Table

test/test.cc

```
 7: int main(int argc, char **argv)
 8: {
 9:      node* myNode = new node(1, 0, 0);
10:      maodv_mtable *mtable = new maodv_mtable(myNode);
11:
12:      myNode->getMyMtTable()->my_add(2, 2, 5);
13:       myNode->getMyMtTable()->show_mytable();
14:
15:      mtable->mt_add_rreq(3, 3, 7, 3, 8);
16:      mtable->mt_add_rreq(2, 3, 4, 2, 6);
17:                  mtable->show_mtable();
18:
19:      mtable->mt_add_rrep(3, 3, 7, 3, 8);
20:                  mtable->show_mtable();
21:
22:      mtable->mt_lookup_mact(3, 7, 3);
23:                mtable->show_mtable();
24:
25:      mtable->mt_delete(3, 7);
26:        mtable->show_mtable();
```

Kód zobrazený vyššie testuje funkčnosť funkcii tabuľky *Multicast Routing Table* pre medziľahlý uzol. Ako prvé sa vytvorí objekt `myNode` triedy `node` (riadok 9), ktorý bude mať nastavenú IP adresu na "1"a sekvenčné číslo spolu s poradovým číslom RREQ_ID na "0". Pre prístup do tabuľky slúži objekt `mtable` (riadok 10). Následne na riadku 12 bude uzol pridaný do multicast skupiny s IP adresou "2". Záznam bude uložený v tabuľke *Multicast Membership Table*. Uzol na riadkoch 15 a 16 prijíma správy RREQ-J, pričom odpovedať môže iba v prípade, ak je členom hľadanej multicast skupiny. V tomto prípade môže odpovedať na správu RREQ-J prijatú na riadku 16, pre ktorú má príslušný záznam vo svojej *Multicast Membership Table*. Správu prijatú na riadku 15 odpovedať nemôže, a tak ju prepošle ďalej. Riadok 19 simuluje prijatie správy RREP-M. Uzol si doplní či aktualizuje záznam do svojej *Multicast Routing Table* tabuľky. Záznam bude mať nastavený príznak FALSE a počet skokov na hodnotu "255". To znamená, že cesta ku multicast stromu nie je zatiaľ aktivovaná a čaká sa na prijatie správy MACT-J pre jej aktiváciu. Riadok 22 simuluje prijatie správy MACT-J. Uzol vyhľadá záznam pre multicast skupinu s IP adresou "3"pre člena s IP adresou "7"a nastaví príslušnému záznamu príznak TRUE a počet skokov na hodnotu uvedenú v správe MACT-J. Od tejto chvíle je cesta ku multicast stromu aktívna. V prípade, kedy sa uzol odpája od multicast skupiny je potrebné vymazať záznam o ňom zo smerovacej tabuľky *Multicast Routing Table*. Zavolaním funkcie `mt_delete()` na riadku 25 sa vymaže záznam o členovi multicast skupiny. Pre kontrolu obsahu tabuľky *Multicast Routing Table* slúži funkcia `show_mtable()`.

Obr.9.4 znázorňuje výstup testu. Na riadkoch $1-6$ je vidieť obsah tabuľky *Multicast Membership Table*. Záznam v tabuľke hovorí o tom, že bol uzol pridaný do multicast skupiny s IP adresou "2", sekvenčným číslom "2"a lídrom skupiny "5". Na riadkoch 7 a 8 pre-

```
1:  +-------Multicast-Membership-Table------+
2:  +----------+---------+------------------+
3:  | grp_addr |  seq_n  |    leader_addr   |
4:  +----------+---------+------------------+
5:  |    2     |    2    |       5          |
6:  +----------+---------+------------------+
7:  You can not answer for a RREQ-J!
8:  You can send RREP-H!
9:  +-------------------Multicast-Routing-Table--------------------+
10: +----------+---------+------- +-----------+-------------+------+
11: | grp_addr |  seq_n  | member | hop_count | leader_addr | flag |
12: +----------+---------+--------+-----------+-------------+------+
13: |    2     |    3    |    4   |     255   |      6      |   0  |
14: +----------+---------+--------+-----------+-------------+------+
15: +-------------------Multicast-Routing-Table--------------------+
16: +----------+---------+------- +-----------+-------------+------+
17: | grp_addr |  seq_n  | member | hop_count | leader_addr | flag |
18: +----------+---------+--------+-----------+-------------+------+
19: |    3     |    3    |    7   |     255   |      8      |   0  |
20: +----------+---------+--------+-----------+-------------+------+
21: |    2     |    3    |    4   |     255   |      6      |   0  |
22: +----------+---------+--------+-----------+-------------+------+
23: +-------------------Multicast-Routing-Table--------------------+
24: +----------+---------+------- +-----------+-------------+------+
25: | grp_addr |  seq_n  | member | hop_count | leader_addr | flag |
26: +----------+---------+--------+-----------+-------------+------+
27: |    3     |    3    |    7   |     3     |      8      |   1  |
28: +----------+---------+--------+-----------+-------------+------+
29: |    2     |    3    |    4   |     255   |      6      |   0  |
30: +----------+---------+--------+-----------+-------------+------+
31: +-------------------Multicast-Routing-Table--------------------+
32: +----------+---------+------- +-----------+-------------+------+
33: | grp_addr |  seq_n  | member | hop_count | leader_addr | flag |
34: +----------+---------+--------+-----------+-------------+------+
35: |    2     |    3    |    4   |     255   |      6      |   0  |
36: +----------+---------+--------+-----------+-------------+------+
```

Obrázek 9.4: Výstup testu pre tabuľku Multicast Routing Table.

behol úspešný test prijatia správy RREQ-J. Keďže v prvom prípade uzol prijal správu RREQ-J s hľadanou multicast skupinou, ktorej nie je členom, nemôže na túto správu odpovedať (riadok 7). Druhá prijatá správa RREQ-J bola úspešne spracovaná, pretože uzol je členom hľadanej multicast skupiny. Uzol si do tabuľky *Multicast Routing Table* vložil záznam o novom členovi multicast skupiny. Výstup tabuľky je možné vidieť na riadkoch $9 - 14$. Ďalej bolo testované prijatie správy RREP-M. Uzol si uložil záznam do tabuľky s nastaveným príznakom FALSE (riadok 19). Po prijatí správy MACT-J sa záznam aktualizoval a za pomoci príznaku TRUE nastavil ako platný (riadok 27). Aktualizoval sa i počet skokov od uzla. Posledným testom bolo prijatie správy MACT-P, ktorá inicializuje vymazanie záznamu o členovi s IP adresou "7"z multicast skupiny s IP adresou "3". Riadky $31 - 36$ zobrazujú obsah tabuľky *Multicast Routing Table*. Vymazávaný záznam sa v tabuľke nenachádza, test prebehol úspešne.

Kapitola 10

Záver

Bezdrôtové siete ponúkajú komfort komunikácie spojenej s mobilitou zariadení. Pretože senzor je zariadenie s obmedzeným zdrojom napájacej energie, je voľba správneho protokolu veľmi dôležitá. Protokol by mal pre svoje fungovanie senzor čo najmenej zaťažovať. Nevhodnou voľbou protokolu bude zaťažený najmä procesor senzoru, ktorý bude spracovávať veľké množstvo dát, a tým bude dochádzať k úbytku obmedzenej napájacej energie senzoru.

MAODV protokol umožňuje dynamické, samočinné smerovanie medzi mobilnými uzlami. Jedná sa o multicast smerovací protokol založený na báze protokolu AODV. Typickým znakom MAODV je používanie sekvenčných čísiel, ktoré zabezpečujú aktuálnosť záznamov medzi uzlami. Princípy fungovania protokolu boli čerpané z dokumentov *RFC-3561* a *ietf-manet-maodv-00.txt*, no táto diplomová práca popisuje vlastné zmeny princípu fungovania protokolu MAODV. Jednou z najdôležitejších zmien bolo definovanie vlastných správ a ich štruktúr. Zjednodušila sa tak komunikácia medzi mobilnými uzlami. Aby nedochádzalo k nadbytočnému zaťaženiu siete a tým pádom aj k zaťaženiu samotných uzlov, spracovávajú uzly len tie správy, ktoré ešte neprijali. To je zabezpečené kontrolou IP adresy a sekvenčného čísla v správe od uzla, od ktorého správa prišla. Ďalším faktorom pre zaťaženie uzlov je správne nastavenie odosielania periodicky sa opakujúcich správ RREP-H a GRPH. Nesprávna voľba časovačov by mohla mať kritický dopad na fungovanie siete.

Network Simulator 2 je nástroj, ktorý umožňuje simulovať rôzne sieťové protokoly a je hojne využívaný pre výskum ad-hoc sietí. Rozšírenie NS2 o nový protokol zahrňuje vytvorenie väzby medzi jazykmi OTcl a C++. Implementácia protokolu do simulačného prostredia nesie v sebe veľa zmien a nastavení v samotných zdrojových kódoch simulátoru. Návrh protokolu pozostával z niekoľkých modulov. Pre jednotlivé moduly som vytvoril príslušné testy. Každý test mal za úlohu overiť správnosť daného modulu. Avšak testovanie celého behu programu bolo obmedzené zložitosťou samotného simulátoru a mnou vytvorené testovacie aplikácie pre overenie funkčnosti kódu už nepostačovali. Následné testovanie by muselo prebiehať logovými výpismi do súboru, čím by sa dala analyzovať funkčnosť kódu. Výsledný kód protokolu som podrobil bezdrôtovej simulácii napísanej v jazyku Tcl. Uzly sa behom simulácie inicializujú, avšak žiadne dáta medzi sebou nepreposielajú.

Literatura

[1] Elizabeth M. Belding-Royer, Charles E. Perkins, Samir R. Das: *Ad hoc On-Demand Distance Vector (AODV) Routing, draft-ietf-manet-aodv-13.txt*. 2003.

[2] Elizabeth M. Royer, Charles E Perkins: Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing Protocol.
URL http://moment.cs.ucsb.edu/AODV/aodv.html

[3] Elizabeth M. Royer, Charles E. Perkins: *Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing, draft-ietf-manet-maodv-00.txt*. 2000.

[4] Francisco J. Ros, Pedro M. Ruiz: *Implementing a New Manet Unicast Routing Protocol in NS2*. 2004.

[5] Jae Chung, Mark Claypool: NS by Example.
URL http://nile.wpi.edu/NS/

[6] Ondřej Hynčica: Bezdrátové sítě typu mesh.
URL http://www.odbornecasopisy.cz/index.php?id_document=30826

[7] Petr Buryan: SensiNet a bezdrátové sítě typu mesh.
URL http://www.odbornecasopisy.cz/index.php?id_document=33945

[8] Rashmi: Introduction - What is MANET.
URL http://www.saching.com/Article/MANET---Mobile-Adhoc-NETwork--/334

[9] Sam Leffler: Wireless networking basics.
URL http://www.freebsdmall.com/~loader/en_US.ISO8859-1/articles/wireless/article.html

[10] Toby J. Velte, Anthony T. Velte: *Cisco A Beginner's Guide (Fourth Edition)*. McGraw-Hill, 2007, iSBN 13: 978-0-07-226383-1.

[11] Zdeněk Bradáč, Petr Fiedler, Milan Kačmář: Bezdrátové komunikace v automatizační praxi I: historie a současnost.
URL http://www.odbornecasopisy.cz/index.php?id_document=28818

Abecední přehled použitých zkratek

DSSS – *Direct-Sequence Spread Spectrum*

FHSS – *Frequency Hopping Spread Spectrum*

FTP – *File Transfer Protocol*

GPRS – *General Packet Radio Service*

GSM – *Global System for Mobile Communications*

HSCSD – *High Speed Circuit Switch Data*

IEEE – *Institute of Electrical and Electronics Engineers*

IP – *Internet Protocol*

ISM – *Industrial, Scientific and Medical radio bands*

LAN – *Local Area Network*

RFC – *Request for Comments*

TTL – *Time To Live*

UDP – *User Datagram Protocol*

UMTS – *Universal Mobile Telecommunications System*

Wi-Fi – *Wireless LAN, WLAN*

WPAN – *Wireless Personal Area Network*