VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

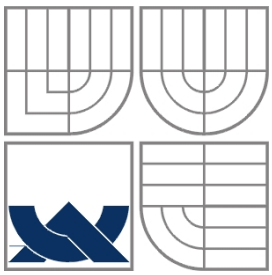
SMĚROVÁNÍ V BEZDRÁTOVÝCH SÍTÍCH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

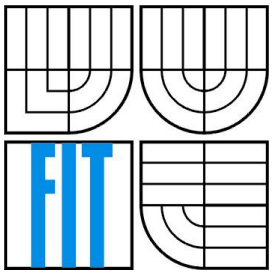
AUTOR PRÁCE
AUTHOR

BC. VÁCLAV JANSKÝ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SMĚROVÁNÍ V BEZDRÁTOVÝCH SÍTÍCH

ROUTING IN WIRELESS NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. VÁCLAV JANSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

MGR. ROMAN TRCHALÍK

BRNO 2008

Abstrakt

Práce se zabývá směrovacími protokoly pro ad hoc bezdrátové sítě. Nejdříve je nastíněna problematika ad hoc sítí. Poté je představeno rozdělení směrovacích protokolů pro ad hoc sítě podle různých kritérií. Největší pozornost je věnována rozdělení podle mechanismu získávání cest. U čtyř protokolů jsou popsány jejich algoritmy. Jedná se o proaktivní protokoly DSDV a CGSR, reaktivní protokol DSR a hybridní protokol ZRP. Dále je popsán protokol AODV detailně a jsou uvedeny jeho výhody a nevýhody. Jsou také zmíněny jeho dvě varianty AODVjr a AODVbis. Na základě nevýhod protokolu AODV je navržen nový protokol. Součástí práce je také popis implementace a integrace tohoto protokolu do simulátoru ns-2 a následně vyhodnocení simulací provedených v tomto simulátoru.

Klíčová slova

Ad-hoc sítě, bezdrátové sítě, směrování, směrovací protokol, reaktivní směrování, proaktivní směrování, hybridní směrování, DSDV, CGSR, DSR, ZRP, AODV, AODVjr, AODVbis, ns-2, implementace, C++, simulace směrování.

Abstract

This work deals with routing protocols for ad hoc wireless networks. First ad hoc networks are introduced. Routing protocols are then classified according to several criteria. Four routing protocols algorithms are described. They are proactive protocols DSDV and CGSR, reactive DSR and hybrid ZRP. Next AODV routing protocol is described in details. Advantages and disadvantages of AODV and two variants of AODV are also introduced. A new protocol is designed based on the disadvantages of the AODV protocol. This work also describes the implementation and integration of the new protocol in the ns-2 simulator. Results of the simulations are presented.

Keywords

Ad hoc networks, wireless networks, routing, routing protocols reactive routing, proactive routing, hybrid routing, DSDV, CGSR, DSR, ZRP, AODV, AODVjr, AODVbis, ns-2, implementation, C++, routing simulation.

Citace

Janský Václav: Směrování v bezdrátových sítích. Brno, 2008, diplomová práce, FIT VUT v Brně.

Směrování v bezdrátových sítích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Mgr. Romana Trchalíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Václav Janský
10. května 2008

Poděkování

Na tomto místě bych rád poděkoval Mgr. Romanu Trchalíkovi za odborné vedení, užitečné rady, připomínky a konzultace.

Obsah

1 Úvod

Žijeme v době, kdy si již oblast informačních technologií nedovedeme představit bez existence počítačových sítí. Až do počátku 70. let byly oblast počítačů a oblast komunikace dvěmi oddělenými a naprosto nezávislými oblastmi. Na přelomu 70. a 80. let došlo k jejich vzájemnému prolnutí, což vedlo k počátku vývoje počítačových sítí.

Pojem počítačová síť může být chápána jako soubor vzájemně propojených a nezávislých počítačů, které spolu mohou komunikovat. Z této možnosti komunikace počítačů plynou výhody počítačových sítí jako například:

- sdílení dostupných hardwarových a softwarových prostředků.
- programy a data na jednom počítači jsou dostupné uživatelům ostatních počítačů v síti.
- větší spolehlivost služeb (stejnou službu poskytuje více počítačů v síti, při výpadku jednoho ho druhý zastoupí).

Současné počítačové sítě jsou složeny ze dvou základních částí, kterými jsou síťová infrastruktura a distribuované aplikace. Infrastruktura umožňuje přenos dat mezi jednotlivými prvky sítě. V dnešních sítích se jako přenosová media používají optické kabely, měděné kabely a bezdrátový přenos pomocí radiových vln. Právě bezdrátový přenos zažívá mohutný rozvoj.

V posledních letech se objevil velký zájem o takzvané ad hoc bezdrátové sítě, které mají obrovský vojenský i komerční potenciál. Ad hoc bezdrátová síť je bezdrátová síť, která nemá pevnou infrastrukturu, neobsahuje tudíž žádná centrální zařízení jako například základnové stanice v buňkových sítích nebo access pointy v bezdrátových lokálních sítích. Ad hoc bezdrátová síť je složená s výpočetních zařízení, které pro komunikaci využívají bezdrátové vysílání. Tyto zařízení slouží zároveň jako směrovače. Tyto sítě mohou být rychle rozmístěny kdekoliv a kdykoliv právě díky tomu, že nevyžadují složité budování infrastruktury. Mohou být využity v několika různých oblastech. Například pro účely vojenské komunikace (skupina vojáků může přímo komunikovat na území nepřítele, kde není možné vytvořit nějakou komunikační infrastrukturu), záchranářské systémy (např. v oblastech postižených živelnými katastrofami), které potřebují rychle vytvořit síť, dále lze tyto sítě využít pro distribuované výpočty, bezdrátové mesh sítě a senzorové sítě.

Náplní této práce bude představení současných algoritmů používaných pro směrování v ad hoc bezdrátových sítích, budou rozebrány klady a zápory jednotlivých přístupů. Dále bude jeden směrovací protokol představen detailněji. Tyto první tři kapitoly byly vypracovány v rámci semestrálního projektu. Praktickou částí práce bude navržení vylepšení pro tento směrování protokol nebo navržení úplně nového protokolu a jeho následná implementace.

2 Směrovací protokoly v bezdrátových sítích

Pojmem směrování se označuje hledání cest v počítačových sítích. Jeho úkolem je dopravit datový paket cílovému uzlu, pokud možno co nejefektivnější cestou. Směrování využívá různých algoritmů. Tyto algoritmy ve spojení s přesnými pravidly pro komunikaci a s typy přenášených zpráv tvoří takzvané směrovací protokoly.

Směrovací protokoly jsou v bezdrátové síti zodpovědné za následující činnosti: hledání vhodné cesty k cílovému uzlu na základě různých kritérií (počet skoků, rychlost linky, spotřeba energie...), shromažďování informací o nefunkčních linkách a oprava nefunkčních linek s využitím minimálního množství energie a šířky pásma.

Vzhledem k bezdrátovému prostředí musí směrovací protokoly čelit problémům, jako jsou mobilita, omezení šířky pásma, skryté a exposed terminály, spotřeba a výdrž baterie. Vzhledem k těmto problémům je použití běžných směrovacích protokolu nevhodné. Ideální směrovací protokol [1] pro ad hoc bezdrátové sítě by měl mít následující vlastnosti:

1. Musí být plně distribuovaný. Distribuované směrování je více tolerantní k vzniklým chybám než centralizované směrování.
2. Musí se přizpůsobit častým změnám topologie souvisejících s mobilitou uzlů.
3. Na získávání a správě cest se musí podílet minimum uzlů.
4. Musí být lokalizovaný.
5. Musí udržovat pouze cesty bez smyček a cesty aktuální.
6. Používat minimální počty broadcastů, aby nedocházelo k příliš častým kolizím.
7. Jakmile se topologie ustálí, musí rychle zkonvergovat.
8. Musí optimálně využívat omezené zdroje jako jsou šířka pásma, výpočetní výkon, paměť a životnost baterie.
9. Musí spravovat jen lokální topologii, změny ve vzdálených částech sítě by neměly způsobovat aktualizace v daném uzlu.
10. Měl by umět upřednostňovat naléhavá data (QoS).

Všechny obrázky v této kapitole jsou převzaty z [1].

2.1 Rozdělení směrovacích protokolů

Směrovací protokoly pro ad hoc bezdrátové sítě lze podle různých kritérií rozdělit do několika skupin. Skupiny na obrázku Obr. 2.1.1 **– Rozdělení protokolů** se vzájemně nevylučují, některé protokoly spadají do několika různých skupin. [1]

Obr. 2.1.1 – Rozdělení protokolů

Směrovací protokoly pro ad hoc bezdrátové sítě lze všeobecně rozdělit do čtyř skupin podle

- Aktualizačního mechanismu směrovacích informací
- Použití temporálních informací pro směrování
- Směrovací topologie
- Spotřeby specifických zdrojů

2.1.1 Podle aktualizačního mechanismu směrovacích informací

Podle aktualizačního mechanismu můžeme směrovací protokoly ad hoc bezdrátových sítí rozdělit do tří kategorií:

1. **Proaktivní neboli tabulkou řízené směrovací protokoly**: V tabulkou řízených směrovacích protokolech si uzly navzájem pravidelně vyměňují směrovací informace, a udržují si tak aktuální informace o topologii sítě ve formě směrovací tabulky. Směrovací informace jsou zpravidla rozesílány do celé sítě. Když uzel potřebuje zjistit cestu k cílovému uzlu, spustí algoritmus pro nalezení cesty na informacích o topologii sítě. Tyto protokoly vychází z běžných směrovacích protokolů využívaných v drátových sítích. Mezi zástupce patří:
    - DSDV – Destination sequenced distance-vector
    - WRP – Wireless routing protocol
    - STAR – Source-tree adaptive routing
    - CGSR – Cluster-head gateway switch routing

2. **Reaktivní neboli on-demand směrovací protokoly:** Protokoly náležící do této kategorie si neudržují informace o topologii sítě. Obstarají si potřebnou cestu k cílovému uzlu až v okamžiku, kdy ji potřebují. Ustanoví spojení, pomocí kterého zjišťují cestu od ostatních uzlů. Z tohoto důvodu si tyto protokoly neposílají pravidelné směrovací informace. Patří sem například:
    - DSR – Dynamic source routing
    - AODV – Ad hoc on-demand distance vector
    - ABR – Associativity-based routing

3. **Hybridní směrovací protokoly**: Protokoly spadající do této kategorie kombinují nejlepší vlastnosti z prvních dvou kategorií. Uzly jsou rozděleny do zón. Uvnitř zóny se využívá tabulkou řízené směrování a mimo zónu se využívá on-demand směrování. Každý uzel spravuje směrovací informace jen od uzlů ve vzdálenosti *m* skoků od daného uzlu. Mezi zástupce patří:

- CEDAR – Core extraction distributed ad hoc routing
- ZPR – Zone routing protocol
- ZHLS – Zone-based hierarchical link state

2.1.2 Podle použití temporálních informací pro směrování

Protokoly v této skupině využívají ke směrování temporálních informací. Ad hoc sítě se vyznačují častými změnami topologie, a k výpadkům linek tak dochází mnohem častěji než u drátových sítí. To je důvodem, proč jsou temporální informace (doba funkčnosti linky, doba funkčnosti zvolené cesty...) v těchto protokolech významné. Směrovací protokoly z této skupiny se dále dělí do dvou kategorií:

1. **Směrovací protokoly využívající minulou temporální informaci**: Tyto směrovací protokoly používají informace o minulých nebo aktuálních stavech linky ke směrovacím rozhodnutím. Například směrovací metrika založená na dostupnosti bezdrátový linek (což je aktuální temporální informace) ve spojení s algoritmem pro nalezení nejkratší cesty poskytují cestu, která je vhodná a stabilní v čase hledání cesty. Změna topologie může způsobit nefunkčnost linky a tím pádem nutnost provádění procesu rekonfigurace s rozumným využitím zdrojů.

2. **Směrovací protokoly využívající budoucí temporální informaci:** Protokoly z této kategorie používají k získání přibližných směrovacích rozhodnutí informace o očekávaném budoucím stavu bezdrátové linky. Bez ohledu na to jak dlouho je již linka ve funkčním stavu, budoucí informace zahrnuje informaci o životnosti uzlu (která je dána zbývající energií baterie), predikci místa a predikci dostupnosti linky.

2.1.3 Podle směrovací topologie

V Internetu je použita hierarchická směrovací topologie a to z důvodu snížení množství informace obhospodařované jednotlivými směrovači. Počet uzlů v ad hoc sítích je však menší, proto lze využít hierarchickou i nehierarchickou (plochou) topologii.

1. **Směrovací protokoly s plochou topologií**: Tyto protokoly používají ploché adresové schéma podobné jako v IEEE 802.3 LAN sítích. Předpokládají existenci jednotného adresového mechanismu pro uzly v ad hoc sítích.

2. **Směrovací protokoly s hierarchickou topologií**: Protokoly z této kategorie využívají logickou hierarchii sítě a asociované adresové schéma. Hierarchie je založena buď na geografické informaci nebo na počtu skoků mezi uzly.

2.1.4 Podle spotřeby specifických zdrojů

1. **Směrování s ohledem na spotřebu energie**: Směrovací protokoly z této kategorie se snaží minimalizovat spotřebu energie. Směrovací rozhodnutí jsou prováděna na základě minimalizace spotřebované energie v síti, ať už globálně nebo lokálně.

2. **Směrování s ohledem na geografickou informaci**: Tyto protokoly zvyšují výkon směrování efektivním využitím dostupných geografických informací.

2.2 Destination-Sequenced Distance Vector Routing Protocol (DSDV)

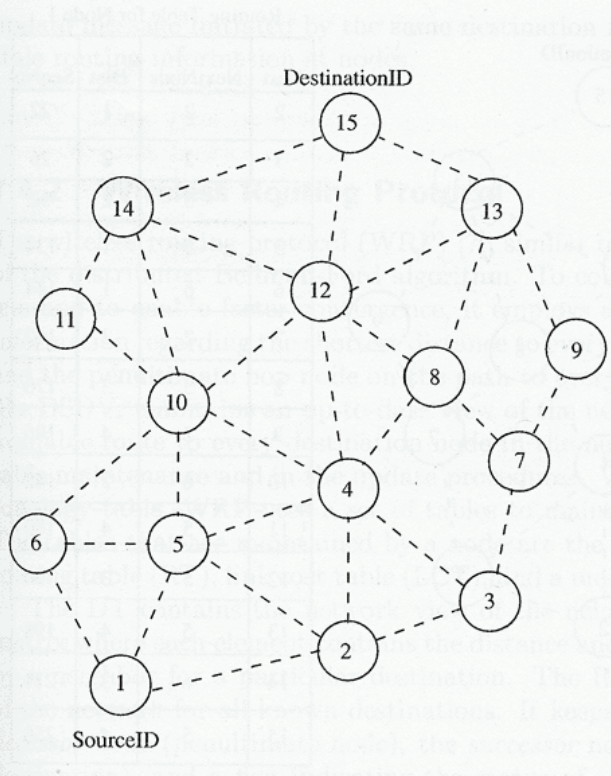
DSDV [9] je jedním z prvních protokolů navržených pro směrování v ad hoc bezdrátových sítích. Jedná se o rozšířenou verzi Bellman-Fordova algoritmu. Podklady pro DSDV jsem studoval z [1] [2].

2.2.1 Popis algoritmu

V DSDV si každý uzel vytváří a spravuje vlastní směrovací tabulku, která obsahuje záznamy, jak se dostat ke všem ostatním uzlům v síti. Každý záznam tabulky obsahuje následující atributy: další skok na cestě ke koncovému uzlu, metriku (např. počet skoků) a sekvenční číslo vydané koncovým uzlem. K udržování konzistence směrovacích tabulek využívá DSDV jak periodické směrovací aktualizace, tak směrovací aktualizace spouštěné na základě určité události (*triggered*). *Triggered* aktualizace jsou používány jako doplněk periodických aktualizací například v případě, kdy dojde ke změně topologie a je třeba rozšířit nové směrovací informace co nejrychleji. Aktualizační pakety obsahují adresy dostupných destinací a počet skoků potřebných k dosažení dané destinace společně se sekvenčním číslem asociovaným s danou cestou.

Když uzel obdrží aktualizační paket, porovná informace v něm obsažené s informacemi ve směrovací tabulce. Aktualizace se staršími sekvenčními čísly, než jsou sekvenční čísla ve směrovací tabulce, se nepoužijí a jsou zahozeny. Pokud se sekvenční čísla rovnají, dojde k nahrazení aktuální cesty ve směrovací tabulce za novou cestu z aktualizačního paketu jen v případě, že má nová cesta lepší metriku. K nahrazení samozřejmě dojde i v případě, kdy má aktualizační paket novější sekvenční číslo než cesta ve směrovací tabulce. Metrika je poté zvětšena o jeden skok, protože příchozí paket bude potřebovat o jeden skok více, aby se dostal k cílovému uzlu. Nově uložená cesta je pak okamžitě rozeslána sousedním uzlům.

Pokud přestane fungovat linka se sousedním uzlem, jsou všechny metriky cest, pro něž byl tento sousední uzel dalším skokem, nastaveny na nekonečno a pro dané cesty je vytvořeno nové sekvenční číslo. Toto je jediný případ, kdy sekvenční číslo nevytváří cílový uzel. Pokud uzel obdrží cestu s metrikou nekonečno a ve své směrovací tabulce má stejnou cestu se stejným nebo starším sekvenčním číslem a s konečnou metrikou, rozešle aktualizaci. Cesty s nekonečnou metrikou jsou pak rychle nahrazeny novými funkčními cestami, samozřejmě jen v případě, že k cílovému uzlu existuje nějaká jiná cesta. Obrázek Obr. 2.2.1 ukazuje síť používající DSDV a směrovací tabulku uzlu č.1.



Dest	NextNode	Dist	SeqNo
2	2	1	22
3	2	2	26
4	5	2	32
5	5	1	134
6	6	1	144
7	2	3	162
8	5	3	170
9	2	4	186
10	6	2	142
11	6	3	176
12	5	3	190
13	5	4	198
14	6	3	214
15	5	4	256

Obr. 2.2.1 – Směrovací tabulka pro uzel č.1

2.2.2 Výhody a nevýhody

Jednou z hlavních výhod DSDV je, že vždy poskytuje cesty beze smyček. Díky aktualizačnímu mechanismu s inkrementovanými sekvenčními čísly mohou být síťové protokoly používané v běžných drátových sítích použity i v ad hoc bezdrátových sítích. Aktualizace posílané po celé síti sice udržují v každém uzlu aktuální směrovací informace, jenomže v případě sítě s uzly s velkou mobilitou, dochází k častým poruchám linek a tím pádem k vysoké řídicí režii. Dokonce malá síť s velkou mobilitou nebo velká síť s malou mobilitou může úplně vyplýtvat celou šířku pásma a učinit tak síť nefunkční. Kvůli vysoké řídicí režii není DSDV vhodný pro ad hoc bezdrátové sítě, které mají malou šířku pásma a vyznačují se častými změnami topologie. Další nevýhodou DSDV je, že když chce uzel získat informace o konkrétním cílovém uzlu, musí čekat na aktualizaci, jejímž původcem je právě daný cílový uzel. Toto zpoždění může vést k neaktuálním informacím v jednotlivých uzlech. A poslední nevýhodou je nemožnost mít k jednomu cíli více cest.

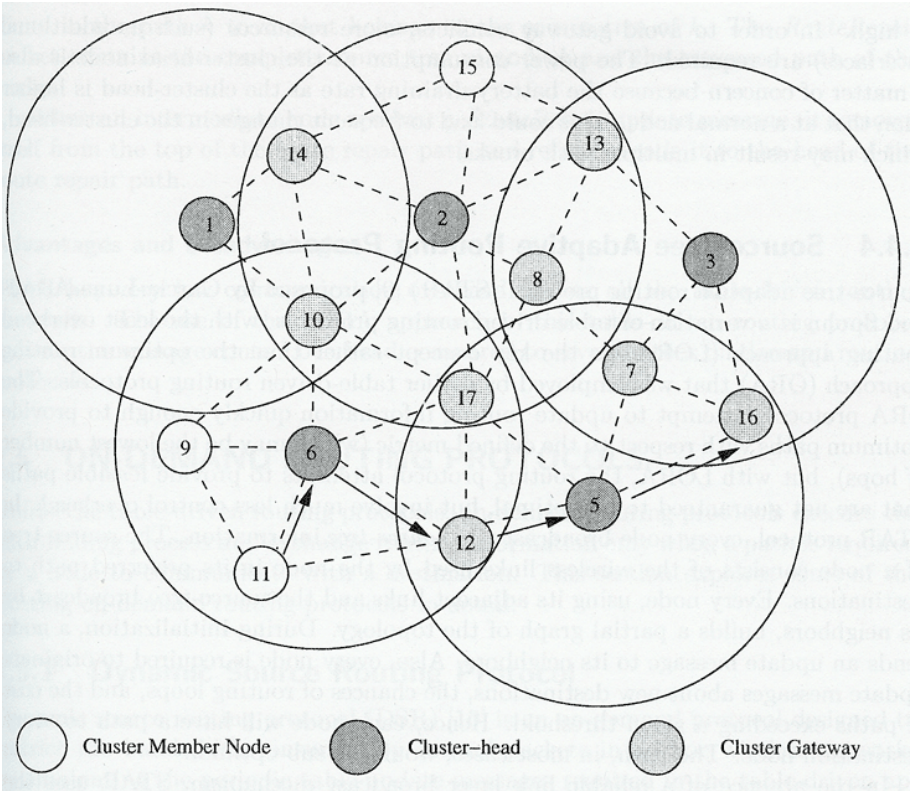
2.3 Cluster-Head Gateway Switch Routing Protocol (CGSR)

CGSR [10] patří mezi protokoly využívající hierarchickou topologii narozdíl od ostatních proaktivních protokolů, které využívají topologii plochou. Principy CGSR jsem čerpal z [1] [2] [8].

2.3.1 Popis algoritmu

CGSR organizuje uzly do menších skupin zvaných shluky (*clusters*). Koordinaci členů v rámci jednoho shluku zajišťuje speciální uzel označovaný jako *cluster-head*. Všechny uzly ve vysílacím rozsahu speciálního uzlu náleží do jeho shluku. *Cluster-head* je volen dynamicky pomocí LCC algoritmu (least cluster change). Podle tohoto algoritmu může dojít ke změně *cluster-headu* v případě, kdy se jeden *cluster-head* přesune do shluku pokrytého signálem jiného *cluster-headu*, k rozhodnutí o novém *cluster-headu* se pak použije nejnižší ID nebo algoritmus nejvyšší konektivity. Druhým případem změny je situace, kdy se nějaký obyčejný uzel přesune mimo dosah všech *cluster-headů*. Díky shlukování lze alokovat šířku pásma nad všemi shluky, čímž se nabízí možnost opakovaného využití. Například, dva různé *cluster-heady* mohou fungovat za použití dvou různých rozprostřených kódů v CDMA systému. *Cluster-head* může uvnitř svého shluku řídit přístup ke kanálu vyzýváním pomocí tokenů. CGSR předpokládá, že veškerá komunikace prochází přes *cluster-head*. Komunikace mezi dvěma shluky probíhá přes běžné uzly, kteří jsou členy obou shluků. Tyto uzly, které jsou členy dvou a více shluků, se nazývají brány (*gateways*). Jelikož je brána členem alespoň dvou shluků, musí dokázat naslouchat dvěma různým rozprostřeným kódům CDMA. Ke konfliktu může dojít, když brána obdrží token v jiném kódu, než na který je zrovna naladěna. Brány které dokáží současně komunikovat přes více rozhraní se dokáží konfliktům vyhnout.

Samotný směrovací algoritmus je založen na DSDV algoritmu. Každý uzel si udržuje dvě tabulky. První z nich je tabulka členů shluku (*cluster member table*), která obsahuje pro každý uzel v síti jeho odpovídající *cluster-head*. Druhou tabulkou je směrovací tabulka, která obsahuje další skok pro daný cílový uzel. *Cluster member* tabulka je broadcastována v pravidelných intervalech. Když uzel dostane od svého souseda aktualizaci *cluster member* tabulky, rozhodne se podle sekvenčního čísla, jestli touto novou tabulkou nahradí svoji současnou tabulku (stejný princip jako u DSDV).



Obr. 2.3.1 – Typy uzlů v CGSR

Při směrování paketu k cílovému uzlu nejdříve zdrojový uzel zjistí z *cluster member* tabulky *cluster-head*, který odpovídá cílovému uzlu (s nejmenším počtem skoků, brány můžou mít více nadřazených *cluster-headů*), a pak podle směrovací tabulky zjistí další uzel na cestě k danému *cluster-headu*. V praxi to pak vypadá následovně. Zdrojový uzel pošle paket svému *cluster-headu*. Ten potom paket pošle bráně, která propojuje tento a sousední shluk na cestě k cílovému uzlu. Brána poté pošle paket sousednímu *cluster-headu* a tak dále, dokud paket nedorazí do cílového shluku, kde jej *cluster-head* pošle cílovému uzlu. Obecně tedy cesta ze zdrojového uzlu *a* do cílového uzlu *b* vypadá takto: *a* – C1 – G1 – C2 – G2 – ... Cn – *b*, kde C je cluster head a G je brána. Na obrázku Obr. 2.3.1 jsou ukázány *cluster-heady*, brány a normální uzly v ad hoc bezdrátové síti. Dále je v něm znázorněna cesta od uzlu č. 11 k uzlu č. 16.

2.3.2 Výhody nevýhody

Velkou výhodou CGSR je, že uzly si díky hierarchickému směrování udržují pouze cesty ke *cluster-headům*. Z toho plyne lepší využití šířky pásma. Nicméně pořád je potřeba režie pro správu clusterů. Další výhodou je možnost jednoduché implementace prioritních plánovacích schémat pomocí plánování tokenů a plánování kódu na branách. *Cluster-heady* mohou díky tomuto dynamickému plánovacímu mechanismu dosáhnout dobré výkonnosti pro real-time přenosy.

Mezi nevýhody patří někdy neoptimální délka cesty (všechno se posílá přes *cluster-heady* a brány). Další nevýhodou je nestabilita při velké mobilitě uzlů, kdy dochází k častým změnám *cluster-headů*. Nezanedbatelným problémem je také otázka spotřeby energie. *Cluster-heady* mají větší spotřebu energie a to může vést k častým změnám v topologii.

2.4 Dynamic Source Routing (DSR)

DSR [11] patří mezi reaktivní směrovací protokoly. Využívá takzvaný *source routing*, což je směrování, kdy zdrojový uzel uvádí kompletní cestu k cílovému uzlu v hlavičce paketu. Každý uzel na této cestě pak jen paket pošle dalšímu uzlu v pořadí. Podklady pro DSR jsem studoval z [1] [2] [8].

2.4.1 Popis algoritmu

V tomto protokolu si uzly nemusí pravidelně posílat informace ze směrovací tabulky, což vede ke snížení režie a lepšímu využití šířky pásma v síti. Každý mobilní uzel v síti si udržuje směrovací cache obsahující cesty, které se uzel doposud naučil. Kdykoliv uzel zjistí novou cestu, přidá ji do své směrovací cache. Každý uzel taky disponuje sekvenčním čítačem (*request id*), pomocí něhož jednoznačně identifikuje generovaný požadavek. Dvojice <adresa zdrojového uzlu, *request id*> vždy určuje přesně jeden konkrétní požadavek v síti. Dva hlavní procesy v DSR jsou proces objevování cest (*Route Discovery)* a proces správy cest (*Route Maintenance).*

2.4.1.1 Route Discovery

Každý uzel může dynamicky zjišťovat cesty k cílovým uzlům v ad hoc síti. Pokud uzel chce poslat paket cílovému uzlu, prohledá nejdříve směrovací cache. Pokud najde cestu, použije ji, jinak spustí *Route discovery* proces. Zdroj rozešle všem svým sousedům *route request* pakety. *Route request* paket obsahuje adresu zdroje, *request id* a seznam všech uzlů, přes které paket doposud prošel.

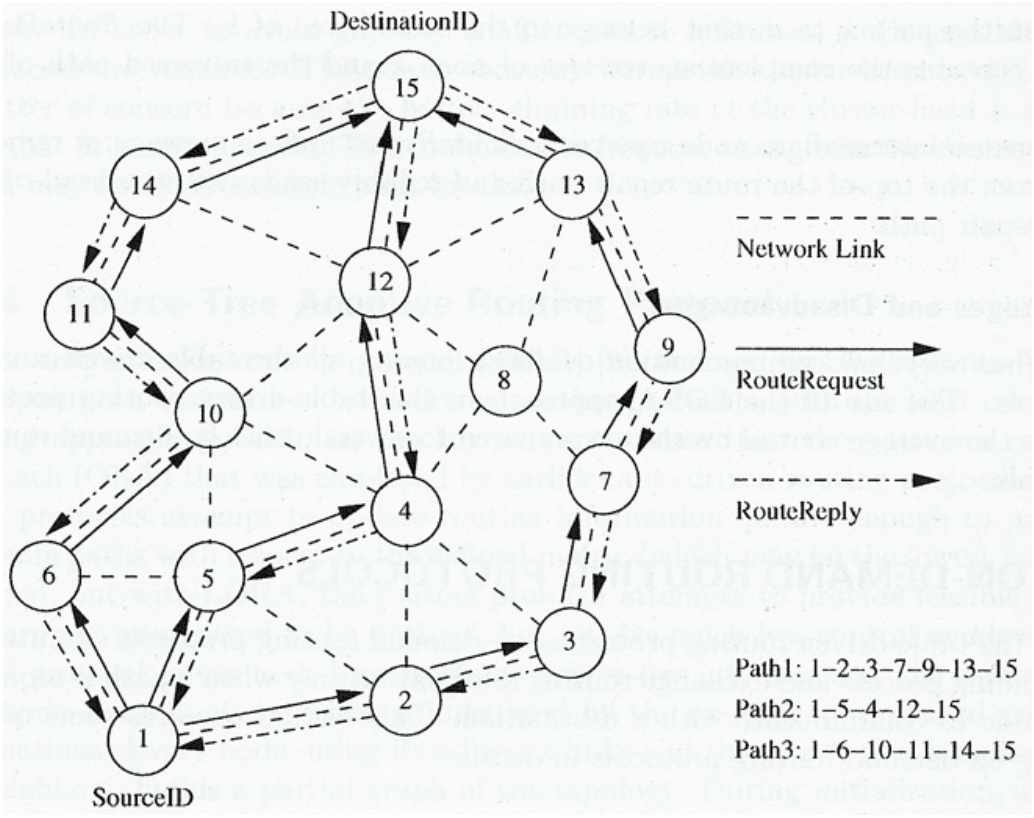
Po obdržení *route request* paketu provede uzel následují operace:

- podle dvojice < adresa zdrojového uzlu, *request id* > ověří, zda-li tento paket už neobdržel dříve, pokud ano zahodí ho.
- pokud se jeho adresa shoduje s adresou cílového uzlu, odešle zdrojovému uzlu *Route reply* paket s cestou, po které dorazil *route request* paket.
- jinak přidá svoji adresu do paketu a pošle ho dále svým sousedům.

Route request pakety putují sítí dokud nedorazí k cíli nebo dokud nejsou všechny zahozeny. *Route reply* pakety jsou ke zdroji směrovány buď podle směrovací cache jednotlivých uzlů nebo podle seznamu uzlů v paketu. Na obrázku Obr. 2.4.1 je znázorněn proces *Route discovery*.

2.4.1.2 Route Maintenance

Tento proces monitoruje správnou funkčnost cest. Výpadek linky je zjišťován pomocí protokolu linkové vrstvy. Když uzel zjistí, že došlo k výpadku linky z jeho sousedem, zjistí dále, jestli byl soused dalším skokem v nějaké cestě. Pokud ano, odešle zdroji cesty paket *Route error*. Všechny uzly po cestě ke zdrojovému uzlu zruší svoje cesty obsahující adresu souseda.



Obr. 2.4.1 – Proces *Route discovery* v DSR

2.4.2 Výhody a nevýhody

Mezi výhody DSR patří využití reaktivního přístupu, čili není potřeba zatěžovat síť pravidelnými aktualizacemi tabulek jako je tomu u tabulkových protokolů. Nemusí se hledat cesty ke všem uzlům

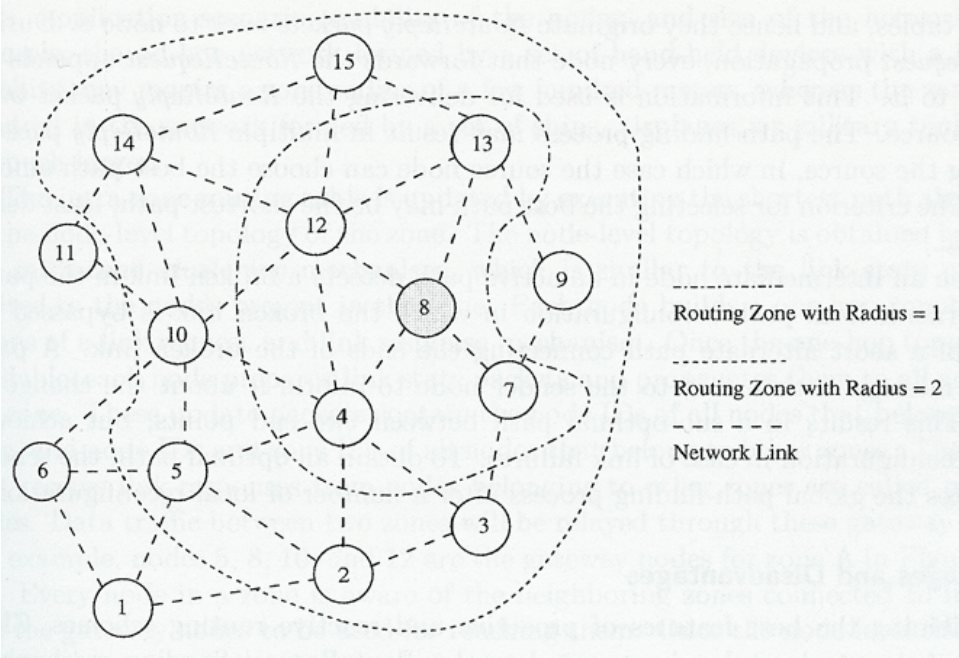
v síti. Další výhodou je, že zdrojový uzel může obdržet více *Route reply* paketů a při případném výpadku linky v používané cestě použije jinou cestu, aniž by musel spouštět *Route discovery* proces. Ve statickém a málo mobilním prostředí dosahuje DSR dobré výkonnosti. Nevýhodou může být delší doba potřebná k odeslání dat v případě dosud neznámé cesty (potřeba spustit *Route discovery*). Dalším problémem je omezení škálovatelnosti. Kvůli použití *source* směrování narůstá velikost paketů, protože se s paketem přenáší adresy všech uzlů pro danou cestu.

2.5 Zone Routing Protocol (ZRP)

ZRP [12] je hybridní směrovací protokol, který efektivně kombinuje vlastnosti proaktivních a reaktivních směrovacích protokolů. Principy ZRP jsem nastudoval z [1] [2].

2.5.1 Popis algoritmu

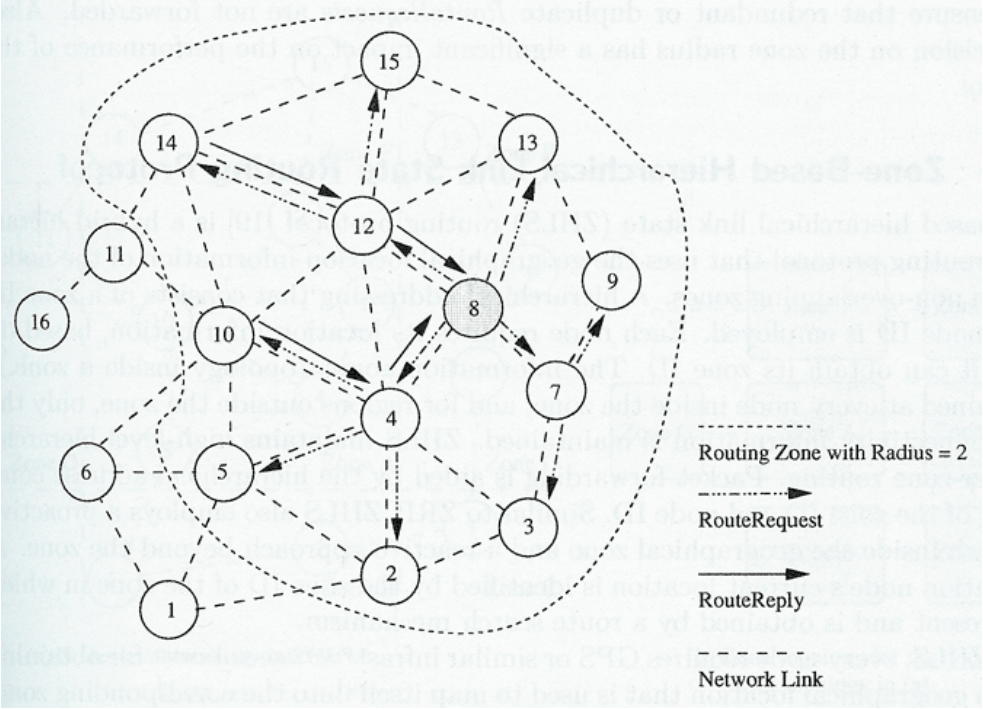
Klíčovým konceptem tohoto protokolu je použití omezených zón pro každý uzel. Zóna pro daný uzel zahrnuje všechny uzly, které jsou dostupné pomocí méně než nebo rovno *r* skoků (*zone radius*). Směrovací zóna je podmnožinou celé sítě. Uvnitř zóny se používá takzvaný *intra-zone routing protocol* (IARP). IARP využívá proaktivního přístupu. Mimo zónu se využívá reaktivního *inter-zone routing protocol* (IERP). Jako IARP je možné použít proaktivní směrovací protokol DSDV a jako IERP reaktivní protokol DSR. Na obrázku Obr. 2.5.1 jsou zobrazeny zóny s $r = 1$ a $r = 2$ pro uzel č. 8. Každý uzel si udržuje informace o cestách ke všem uzlům v rámci své zóny pravidelným rozesíláním aktualizačních paketů (součást IARP). Pro velká *r* se ZRP chová více proaktivně a pro malá *r* naopak více reaktivně.



Obr. 2.5.1 – Zóny pro uzel č. 8

IERP zodpovídá za hledání cest k uzlům, které jsou mimo zónu. Pokud chce zdrojový uzel *s* odeslat paket cílovému uzlu *d*, zjistí nejdříve, zda-li je *d* ve stejné zóně. Pokud se cílový uzel *d* nachází v jeho zóně, odešle mu paket přímo. Pokud je *d* mimo zónu rozešle zdrojový uzel *s* paket

všem okrajovým uzlům zóny. Pokud některý z okrajových uzlů zjistí, že *d* se nachází v jeho zóně, pošle zdrojovému uzlu *s Route reply* paket, jinak pošle svým okrajovým uzlům *Route request* paket. Tento proces je znázorněn na obrázku Obr. 2.5.2. Během rozesílání *Route request* paketů přidává každý uzel do paketu svoji adresu. Tato informace je poté použita pro doručení *Route reply* paketu zpět ke zdrojovému uzlu *s*.



Obr. 2.5.2 – Zjišťování cesty k uzlu č. 16

2.5.2 Výhody a nevýhody

ZRP snižuje řídicí režii při rozesílání *Route request* paketů v porovnání se stejnou činností v čistě reaktivních protokolech. *Route request* pakety jsou posílány jen hraničním uzlům. Také je snížena komunikační režie v porovnání s proaktivními protokoly.

3 Ad hoc On-Demand Distance Vector (AODV)

AODV [3] je směrovací protokol využívaný v mobilních ad hoc bezdrátových sítích. Tento protokol je založen na reaktivním přístupu získávání cest k cílovým uzlům. V porovnání s ostatními protokoly pro mobilní ad hoc sítě se jedná o poměrně jednoduchý protokol. AODV je však jedním z nejvíce používaných protokolů v této oblasti. Podobný reaktivní přístup je například použit ve směrovacím protokolu v sítích ZigBee.

V této kapitole budou nejdříve popsány principy základního AODV protokolu a v dalších částech pak budou nastíněny jeho různé modifikace.

3.1 Principy AODV

Směrovací protokol AODV je specifikován v dokumentu RFC 3561 [3]. AODV je reaktivním rozšířením proaktivního protokolu DSDV. AODV je navrženo tak, aby dosahovalo lepších výkonnostních charakteristik v oblasti vytváření a správy cest, než je tomu právě u DSDV. Hlavními cíli AODV je:

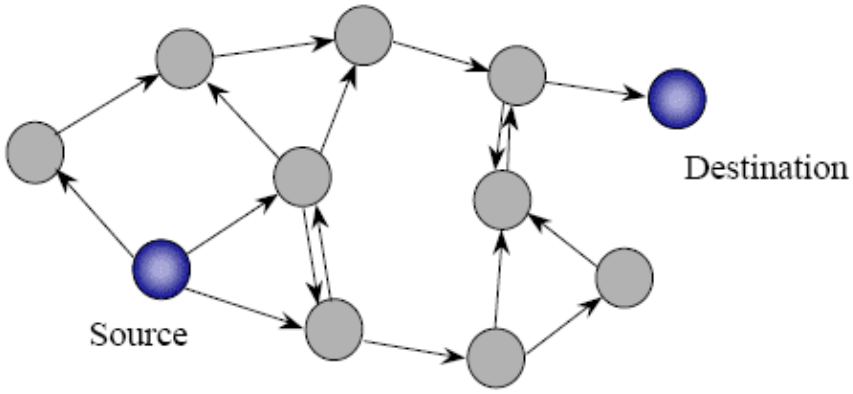
1. Rozesílat broadcast pakety, jen když je to potřeba (uzel potřebuje zjistit cestu).
2. Rozlišit správu lokální konektivity a správu celé topologie.

Díky čistě reaktivnímu přístupu AODV snižuje kontrolní režii a minimalizuje množství rozesílaných broadcastů. AODV předpokládá obousměrné linky mezi uzly, existují však mechanismy, které se pokouší vyrovnat i s jednosměrnými linkami (např. blacklist množiny). Každý uzel v AODV si udržuje dva čítače: 1. čítač sekvenčních čísel, což je jednoduchý čítač používaný k udržování aktuálnosti reverzní cesty ke zdrojovému uzlu a 2. RREQ-ID čítač, který je inkrementován vždy, když uzel vytvoří novou RREQ zprávu. Uzel si také udržuje informace o dostupných sousedech. Obrázky Obr. 3.1.1, Obr. 3.1.2 a Obr. 3.1.3 byly převzaty z [6]. Kromě [3] jsem čerpal také z [8].

3.1.1 Získávání cest (Route discovery)

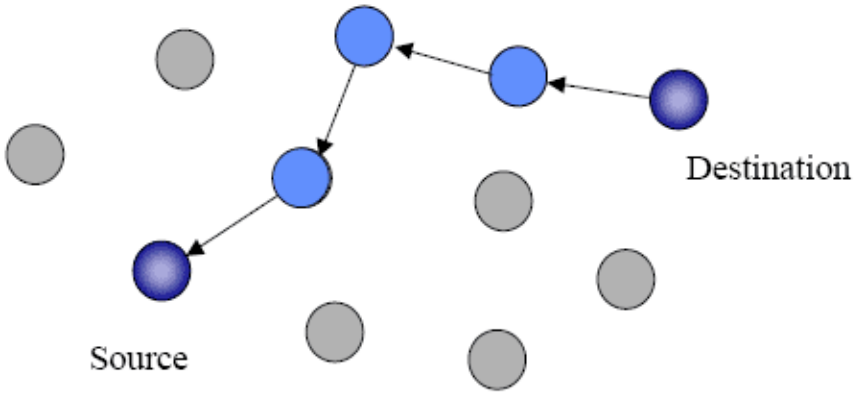
AODV získává nové cesty pomocí *route discovery* mechanismu, který je založen na broadcastovém rozesílání. Když uzel potřebuje komunikovat s jiným uzlem, ke kterému nezná cestu, spustí proces *route discovery* rozesláním RREQ (*Route Request*) zpráv (viz. obrázek Obr. 3.1.1). RREQ zpráva obsahuje zdrojovou adresu, zdrojové sekvenční číslo, RREQ-ID, cílovou adresu, cílové sekvenční číslo a počet skoků. Počet skoků je nastaven na nulu a na každém dalším uzlu po cestě k cíli je inkrementován jedničkou. Po obdržení RREQ zprávy uzel nejdříve ověří, jestli není cílovým uzlem, pokud je, odešle RREP (*Route Reply*) zprávu. Pokud cílovým uzlem není, tak ověří, jestli už neobdržel RREQ se stejnou zdrojovou adresou a stejným RREQ-ID, pokud ano, zpráva je zahozena. Pokud danou RREQ zprávu obdržel prvně, zjistí, jestli nemá ve své cache záznam s cestou k cílovému uzlu. Pokud takový záznam má pošle zdrojovému uzlu RREP zprávu, ale jen v případě že cílové sekvenční číslo v jeho tabulce je větší než cílové sekvenční číslo v RREQ zprávě, jinak rozešle

RREQ zprávu dál s inkrementovaným počtem skoků. Zpáteční cesta je ustanovena podle předchozího průchodu RREQ zprávy. V případě, kdy uzel najde cestu k cíli ve své cache, může poslat cílové stanici takzvaný *gratuitous* RREP. Účelem je zaručit, že cílový uzel se dozví cestu k původnímu zdrojovému uzlu. Výsledek je stejný je stejný, jako by si cílový uzel zažádal zprávou RREQ o cestu k uzlu, který posílal *gratuitous* RREP. O tom zda-li se pošle *gratuitous* RREP, rozhoduje příznak G v původní zprávě RREQ.



Obr. 3.1.1 – Šíření zpráv RREQ

Cílový uzel nebo uzel s cestou v cache tedy odpoví na RREQ zprávu RREP zprávou, která obsahuje vesměs stejné atributy jako RREQ zpráva. RREP zpráva cestuje zpět k cílovému uzlu reverzní cestou (viz obrázek Obr. 3.1.2). Všechny uzly po cestě si vytvoří ve směrovací tabulce nový záznam, nastaví si další skok na uzel, od kterého přijali RREP zprávu, a uloží si počet skoků a cílové sekvenční číslo. Dále si rozšíří seznam sousedů o uzel, který jim poslal RREP zprávu a konečně restartují časovače u záznamu pro danou cestu. Nepoužívaná cesta je po uplynutí dané časové hodnoty smazána. Když RREP zpráva dorazí k zdrojovému uzlu, uloží si tento uzel informace do své směrovací tabulky, ale jen v případě, že cílové sekvenční číslo nové cesty je vetší nebo rovno sekvenčnímu číslo současné cesty (pokud nějaká existuje). V případě rovnosti sekvenčních čísel je použita cesta s nižším počtem skoků. Cesta k cíli je v síti uzlů vytvořena průchodem RREP od cílového ke zdrojovému uzlu.



Obr. 3.1.2 – Šíření zprávy RREP

3.1.2 Správa směrovací tabulky

V AODV si každý uzel udržuje směrovací tabulku se záznamy obsahujícími následující informace: 1. cílová adresa, 2. další skok, 3. počet skoků (metrika), 4. cílové sekvenční číslo, 5. seznam aktivních uzlů pro tuto cestu, pro které je daný uzel dalším skokem (*precursor list*), 6. časový údaj o vypršení platnosti cesty.

Pro každou cestu, která je ve formě záznamu ve směrovací tabulce, si uzel udržuje seznam uzlů (*precursor list*), kteří přes něj posílají pakety. Tyto uzly obdrží upozornění v případě, že daný uzel zjistí nefunkčnost linky s uzlem, který je další skok na cestě k cíli. Tento seznam obsahuje ty uzly, kterým daný uzel poslal nebo přeposlal RREP zprávu.

Soused je považován za aktivního pro danou cestu, pokud je od něj přijat alespoň jeden paket pro cílový uzel během časového intervalu ACTIVE_ROUTE_TIMEOUT. Díky tomu se všechny aktivní zdrojové uzly dozví o chybě linky na cestě k cíli.

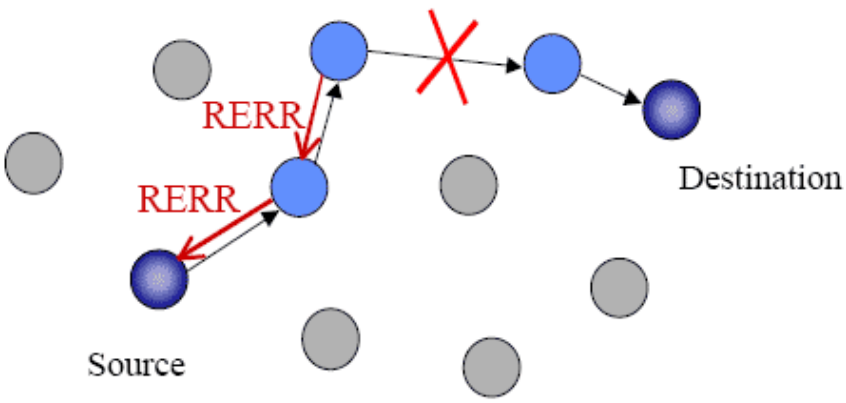
3.1.3 Správa cest

Při výpadku linky v aktivní cestě se uzel může pokusit linku lokálně opravit, a když se to nepovede, je potřeba učinit následující opatření:

- existující cesty označit za neplatné
- zjistit sousední uzly, kteří mohou být výpadkem ovlivněni
- odeslat těmto uzlům RERR (*Route Error*) zprávu.

RERR zpráva může být buď rozeslána broadcastem (pokud je ovlivněno více sousedů) nebo unicastem (pokud je to pouze jeden soused) nebo opakovaným unicastem (pokud je broadcast nevhodný). Zde je však omezení maximálního počtu RERR zpráv za sekundu. Uzel iniciuje zpracování RERR zprávy v těchto situacích:

1. detekuje výpadek linky s *next hop* uzlem aktivní cesty, když chce posílat data (a oprava cesty byla neúspěšná)
2. pokud obdrží paket určený uzlu, pro který daný uzel nemá aktivní cestu
3. když obdrží RERR zprávu od souseda pro jednu nebo více aktivních cest



Obr. 3.1.3 – Šíření zprávy RERR

Pokud uzly ještě potřebují komunikovat s cílem, se kterým jim byla cesta označena za neplatnou, musí znovu iniciovat *Route Request* proces. Obrázek Obr. 3.1.3 ukazuje poslání RERR zprávy při výpadku linky.

3.1.4 Správa lokální konektivity

Prvním způsobem, jak může uzel poskytovat informaci o konektivitě, je rozesílání lokálních *Hello* zpráv. Uzel kontroluje v pravidelných intervalech (*HELLO_INTERVAL* v ms), jestli odeslal nějaký broadcast (např. RREQ nebo odpovídající zprávu linkové vrstvy), pokud neodeslal žádný broadcast, tak rozešle *Hello* zprávu, což je RREP zpráva s TTL nastaveným na 1. Uzel může zjišťovat konektivitu naslouchaním paketů od množiny sousedů. Pokud během dané doby (DELETE_PERIOD) obdržel od souseda *Hello* zprávu a pak už neobdržel žádný další paket (*Hello* zprávu nebo jiný) po dobu delší než ALLOWED_HELLO_LOSS * HELLO_INTERVAL ms, měl by uzel považovat linku s tímto sousedem za ztracenou.

Druhou možnosti detekce konektivity je využití mechanismů linkové vrstvy. Například, neúspěšné vysílání linkové vrstvy, absence ACK linkové vrstvy nebo neobdržení signálu CTS po odeslání maximálního počtu signálů RTS mohou být použity k indikaci nefunkční linky v AODV. Tato metoda je nazývána LLN (*Link Layer Notification*). Není ji však možné vždy použít, protože ne každý hardware umožňuje podporu detekce linkové vrstvy.

3.2 Výhody a nevýhody AODV

Hlavní výhodou tohoto protokolu je, že jsou cesty sestavovány na požádání. Uzly tak udržují pouze cesty, o které mají opravdu zájem. Další výhodou je použití sekvenčních čísel k nalezení aktuálních cest k cílovému uzlu.

Nevýhodou je počáteční zpoždění při odeslání paketu uzlu, pro který zdrojový uzel nezná cestu. Další nevýhodou je, že uzly mezi zdrojovým a cílovým uzlem mohou poskytnout neaktuální cestu. A to v případě kdy tyto uzly mají novější sekvenční číslo než zdrojový uzel, ale není to nejaktuálnější sekvenční číslo. Jinou nevýhodou je větší zatížení sítě při rozesílání více RREP na jeden RREQ. Poslední, zde zmíněnou, nevýhodou je rozesílání pravidelných zpráv, které může vést nadbytečné spotřebě šířky pásma a energie uzlu..

3.3 Varianty AODV

Různé varianty AODV protokolu mohou být využity pro směrování v bezdrátových sítích s nízkými zdroji energie v takzvaných LoWPANs (low-power wireless personal area networks). Gomez a spol. ve článku [4] poskytují přehled několika variant AODV použitelných pro LoWPANs. Jsou to následující:

- AODVjr
- AODVbis
- LoWPAN-AODV

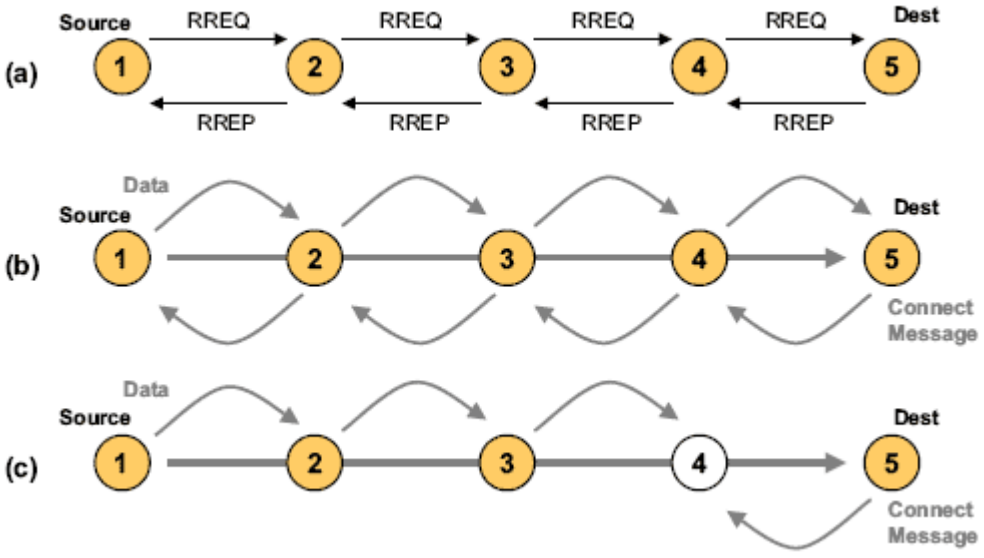
- LOAD
- TinyAODV

3.3.1 AODVjr

AODVjr [5] je jedna z prvních zjednodušených verzí protokolu AODV. Autoři článku [5] považují některé části AODV specifikace za náchylné k chybnému naprogramování. Proto některé části původní specifikace odebrali a ponechali jen stěžejní prvky. Následující prvky byly odebrány:

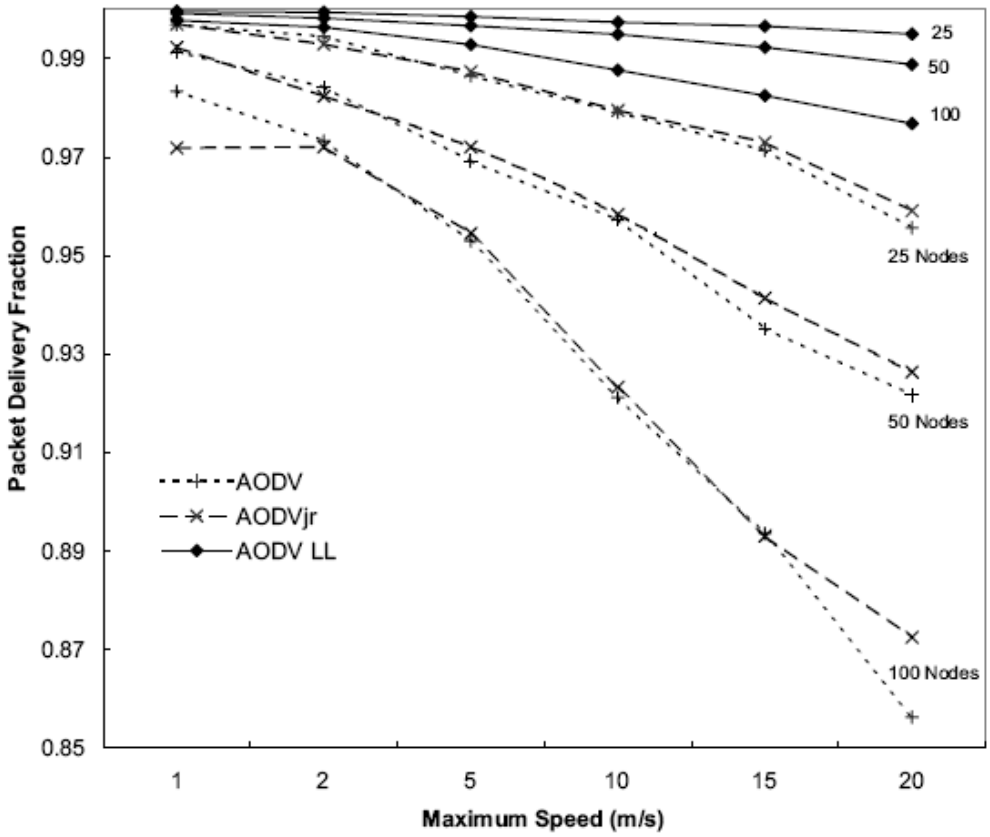
- Sekvenční čísla
- *Gratuitous* RREP
- Metrika počet skoků
- *Hello* zprávy
- RERR zprávy
- *precursor list*

Ke splnění tohoto požadavku musí AODVjr použít mírně jiné operace než standardní AODV. Na RREQ zprávy odpovídá RREP zprávou pouze cílový uzel (viz obrázek Obr. 3.3.1). Uzly po cestě mezi zdrojovým a cílovým uzlem nesmí zprávu RREP posílat. Takhle se také eliminuje potřeba *Gratuitous* RREP zpráv. Cílový uzel odpovídá pouze na první obdrženou RREQ zprávu, a proto je vždy zvolena „nejlepší“ (nejrychlejší) cesta bez ohledu na počet skoků. Při vykonávání správy cest je doba platnosti cesty aktualizována pouze při přijetí paketů a ne při odesílání paketů. Proto musí cílový uzel občas zaslat paket zdrojovému uzlu. Pokud probíhá přenos dat pouze v jednom směru, tak se musí zasílat speciální periodické zprávy *connect* (viz obrázek Obr. 3.3.1). Pokud probíhá přenos dat v obou směrech, není třeba žádná přídavná režie v podobě takových zpráv. Použitím této strategie odpadá potřeba *hello* zpráv, RERR zpráv a *precursor* listů. Pokud dojde k přerušení cesty, zdrojový uzel přestane dostávat zprávy od cílového uzlu. Na obrázku Obr. 3.3.1 opustil cestu uzel č. 4. Po nějakém čase uzel detekuje výpadek linky, protože přestane dostávat zprávy od cílového uzlu. Pokud cestu k cílovému uzlu nadále potřebuje, musí spustit nový *Route discovery* proces.



Obr. 3.3.1 – Zasílání zpráv v AODVjr

Na základě provedených simulací autoři článku uvádějí, že AODVjr dosahuje srovnatelné výkonnosti jako AODV. To je patrné z grafu na obrázku Obr. 3.3.2. Nepopiratelnou výhodou AODVjr je jednodušší implementace oproti AODV. Jednotlivé odebrané prvky lze pak doimplementovat v případě potřeby.



Obr. 3.3.2 – Srovnání AODVjr s AODV – úspěšnost doručení paketů

3.3.2 AODVbis

AODVbis [6] je revizí originální AODV specifikace. Tato revize se zaměřuje na objasnění některých funkčních aspektů a označení některých vlastností jako volitelné. RERR zprávy nejsou povinné. Pokud jsou takové zprávy použity, jsou lokálně rozesílány broadcastem, díky čemuž není potřeba udržovat *precursor list* pro každý záznam o cestě. Každý uzel, který obdrží RERR zprávu, by měl v důsledku jejího zpracování přidat nedostupné cílové uzly do nově generované RERR zprávy. Počet skoků už není povinnou směrovací metrikou. Z čehož plyne možnost stejného přístupu jako v AODVjr. V AODVbis není zahrnuta lokální oprava cest. Volitelnou vlastností je akumulace cesty, která umožňuje uzlu získat směrovací informace ze seznamu absolvované cesty v RREQ zprávě nebo v RREP zprávě.

4 Návrh

Tato kapitola popisuje návrh směrovacího protokolu, který by měl upravit stávající AODV protokol tak, aby zmírnil některé z jeho záporných vlastností. Nejdříve bude popsán samotný protokol a zmíněny jeho odlišnosti od AODV. V dalších podkapitolách budou popsány jednotlivé typy zpráv, které protokol bude používat k získávání a údržbě cest. Popsány budou také tabulky (směrovací tabulka, broadcast-id tabulka) potřebné ke správné činnosti protokolu. V závěru kapitoly bude popsán návrh tříd, ze kterého se bude vycházet při implementaci.

4.1 Návrh protokolu

Protokol AODV je jedním z nejvíce studovaných protokolů pro ad hoc bezdrátové sítě. Jeho zápornou vlastností je, že první paket určený k odeslání musí čekat, až protokol zjistí s pomocí dalších uzlů cestu k cíli. Tato nevýhoda se nedá nijak eliminovat, vychází prostě z reaktivní povahy protokolu. Zpoždění se však týká jen prvního paketu, tudíž to nemusí být pro celkový datový přenos kritické. Mezi další záporné vlastnosti protokolu AODV můžeme zařadit velkou kontrolní režii, posílá se množství RREQ zpráv při procesu *route discovery* a v návaznosti na ně zprávy RREP a dále pak velké množství *hello* zpráv pro správu cest a RERR zpráv pro informování a špatných linkách. Právě snížení množství kontrolních paketů je hlavním cílem upraveného protokolu.

4.1.1 Algoritmus

Reaktivní přístup zůstává zachován stejně jako u běžného AODV. Čili pokud chce zdrojový uzel odeslat paket cílovému uzlu, musí nejdříve provést *route discovery* proces, a pak pokud je proces úspěšný a cesta je nalezena, může teprve paket odeslat. Od AODV se liší použitou metrikou. V novém protokolu to není počet skoků, ale nejrychlejší RREQ. Údržba cest je prováděna odlišným způsobem než u AODV. Využívá se při ní speciálních zpráv *ALIVE*.

4.1.1.1 Proces získávání cest (Route discovery)

V situaci, kdy zdrojový uzel má k odeslání paket, ale nezná cestu, musí spustit *route discovery* proces. Zdrojový uzel broadcastem pošle zprávu *route request*, každý další uzel, který tuto zprávu obdrží, ji musí opět broadcastovat dál. Vždy se reaguje jen na první přijatou zprávu *route request*. Tudíž je nutné, si nějakým způsobem evidovat přijaté zprávy *route request*. A zprávy, které byly již dříve obdrženy, zahazovat. Navíc si každý z těchto uzlů přidá do směrovací tabulky cestu ke zdroji (tzv. reverzní cesta).

Jakmile zprávu *route request* obdrží cílový uzel, odpoví na ni zprávou *route reply*. Na *route request* odpovídá vždy jen cílový uzel, nedochází tedy k posílání *route reply* jinými uzly, které znají cestu k cíli. Což je změna oproti běžnému AODV. Další změnou je, že cílový uzel odpoví vždy jen na první přijatý *route request*. Tento navrhovaný upravený protokol tedy nepoužívá metriku počet skoků, ale jednoduše je zvolena cesta, po které přišel první *route request*.

Zpráva *route reply* je směrována po reverzní cestě vytvořené při šíření zprávy *route request*. Při šíření zprávy *route reply* si naopak uzly přidávají do směrovací tabulky cestu k cílovému uzlu. Tato cesta je pak využita k poslání datového paketu.

4.1.1.2 Správa cest (Route maintenance)

Správa cest obsahuje oproti původnímu AODV největší změny. Upravený protokol totiž nepoužívá zprávy *route error* ani *hello* zprávy a nepoužívá ani lokální opravu cesty. Jsou zde použity jiné principy. Každý cílový uzel musí zdrojovému uzlu posílat speciální zprávu *ALIVE* (stejný princip jako zpráva *connect* v AODVjr). Když zdrojový uzel tyto zprávy dostává, tak si může být jistý, že cesta je stále funkční. Cílový uzel odesílá *ALIVE* zprávy v pravidelných intervalech a zdrojový uzel očekává jejich přijetí. Po uplynutí dané doby bez přijetí *ALIVE* zprávy, je cesta označena za neplatnou. Tímto způsobem protokol funguje pro jednosměrnou datovou komunikaci. Pokud je datový provoz obousměrný, není zpráv *ALIVE* potřeba.

Životnost cesty je prodloužena vždy při přijatém paketu nebo zprávě. Což je rozdíl oproti původnímu AODV, kde je životnost prodloužena při odeslání, a o nefunkčnosti linky je pak uzel informován *route error* zprávou.

4.1.2 Zprávy směrovacího protokolu

Nový protokol pro svou činnost používá pouze tři typy zpráv:

- *Route request*
- *Route reply*
- *ALIVE*

4.1.2.1 Zpráva Route Request

Zpráva je posílána zdrojovým uzlem cílovému uzlu. Je to požadavek na získání cesty k cíli. Tyto zprávy jsou rozesílány brodcastem. Zprávy jednotlivých *route discovery* procesů musí být jednoznačně identifikovatelné, aby bylo šíření broadcastu kontrolované. Uzel reaguje vždy jen na první zprávu *route request* z daného procesu *route discovery*, ostatní zprávy ze stejného procesu jsou zahozeny.

Zpráva *route request* by měla obsahovat následující políčka:

- TYP ZPRÁVY
- ADRESA ZDROJOVÉHO UZLU
- ADRESA CÍLOVÉHO UZLU
- POČET SKOKŮ
- BROADCAST ID

Význam políček je následující. TYP ZPRÁVY specifikuje, o kterou ze tří zpráv se jedná. ADRESA ZDROJOVÉHO UZLU je adresou uzlu, který rozeslal zprávu *route request*. ADRESA CÍLOVÉHO UZLU je adresou uzlu, který je cílem datových paketů. POČET SKOKŮ udává počet uzlů, přes které zpráva doposud prošla. Upravený protokol sice nepoužívá počet skoků jako metriku pro výběr cesty, ale tento údaj je využit při příštím *route request* procesu (*expanding ring search*). Broadcast se pak posílá se sníženým TTL. BROADCAST ID je číslo identifikující *route discovery* proces. Dvojice políček ADRESA ZDROJOVÉHO UZLU a BROADCAST ID jednoznačně identifikuje konkrétní *route discovery* proces.

4.1.2.2 Zpráva Route Reply

Tuto zprávu odesílá cílový uzel zdrojovému uzlu jako reakci na obdržení první zprávy *route request*. Zpráva je posílána unicastem.

Zpráva *route reply* by měla obsahovat následující políčka:

- TYP ZPRÁVY
- ADRESA ZDROJOVÉHO UZLU
- ADRESA CÍLOVÉHO UZLU
- POČET SKOKŮ

Význam políček je následující. TYP ZPRÁVY specifikuje, že se jedná o *route reply* zprávu. ADRESA ZDROJOVÉHO UZLU je adresou uzlu, který rozeslal zprávu *route request*. ADRESA CÍLOVÉHO UZLU je adresou uzlu, který je cílem datových paketů. POČET SKOKŮ udává počet uzlů, přes které zpráva prošla.

4.1.2.3 Zpráva ALIVE

ALIVE zpráva je pravidelně zasílána cílovým uzlem zdrojovému uzlu, čímž je kontrolována funkčnost cesty.

Zpráva *ALIVE* obsahuje jediné pole:

- TYP ZPRÁVY

TYP ZPRÁVY specifikuje, že se jedná o *ALIVE* zprávu. Žádné další pole nejsou vzhledem k účelu této zprávy zapotřebí.

4.1.3 Směrovací tabulka

Směrovací tabulka je základní datovou strukturou pro směrování. Obsahuje záznamy podle, kterých se určuje cesta k cílovému uzlu. Každý uzel navrženého protokolu si udržuje vlastní směrovací tabulku. Záznam by měl obsahovat následující položky:

- ADRESA CÍLOVÉHO UZLU
- DALŠÍ SKOK
- STAV ZÁZNAMU
- POČET SKOKŮ
- PLATNOST

Význam položek je následující. ADRESA CÍLOVÉHO UZLU je adresa uzlu jemuž jsou zasílány datové pakety, tato položka je zároveň klíčem pro vyhledávání ve směrovací tabulce. DALŠÍ SKOK je adresou dalšího uzlu na cestě k cílovému uzlu. STAV ZÁZNAMU signalizuje, zda-li je záznam o cestě platný nebo neplatný, případně zda-li probíhá *route discovery* proces. POČET

SKOKŮ udává počet uzlů přes něž je paket směrován k cíli (využití při *expanding ring search*.) PLATNOST udává čas, kdy vyprší platnost cesty.

Pro správné broadcastové rozesílání zpráv *route request* potřebujeme uchovávat ještě některé další položky související s danou cestou. Tyto položky lze udržovat ve zvláštní tabulce nebo jako součást směrovací tabulky (doporučeno). Jedná se o položky:

- RREQ TIMEOUT
- ČAS POSLEDNÍHO ODESLANÉHO PAKETU
- POČET RREQ POKUSŮ
- POSLEDNÍ TTL

Význam je následující. RREQ TIMEOUT udává čas, do kdy se čeká na skončení aktuálního *route discovery* procesu. ČAS POSLEDNÍHO ODESLANÉHO PAKETU udržuje informaci o čase, kdy byl odeslán poslední paket adresovaný cíli. POSLEDNÍ TTL udává poslední známý počet skoků k danému cíli.

4.1.4 Broadcast-id tabulka

Aby bylo broadcastové šíření *route request* zpráv kontrolované, je třeba evidovat již přijaté zprávy a při dalším přijetí stejné zprávy, již na tuto nereagovat. K tomu slouží Broadcast-id tabulka. Tato tabulka by měla obsahovat následující položky:

- ADRESA ZDROJOVÉHO UZLU
- BROADCAST-ID
- PLATNOST

Význam položek je následující. ADRESA ZDROJOVÉHO UZLU je adresou uzlu, který původně rozeslal zprávu *route request*. BROADCAST-ID je číslo identifikující *route discovery* proces. PLATNOST udává čas kdy vyprší platnost záznamu.

Broadcast-id tabulka by měla být pravidelně promazávána od záznamů s prošlou platností, aby se šetřila paměť uzlu.

4.1.5 Paketová fronta

Pokud chce uzel odeslat paket, ale zatím nezná cestu pro cílový uzel, musí si paket někde uchovat, dokud cestu nezjistí. K tomu slouží paketová fronta. Nejedná se o frontu v pravém slova smyslu, protože upravený protokol může vyžadovat odebrání i z jiných míst fronty než jen z jejího začátku.

Pakety by měli být ve frontě drženy jen po určitou dobu, aby nedocházelo k přílišnému čerpání paměti. Pakety, které jsou ve frontě déle něž danou maximální dobu by měli být zahazovány. Velikost fronty je omezená. Pokud je fronta plná a je potřeba přidat další paket, tak je zahozen nejstarší paket a nový paket je přidán.

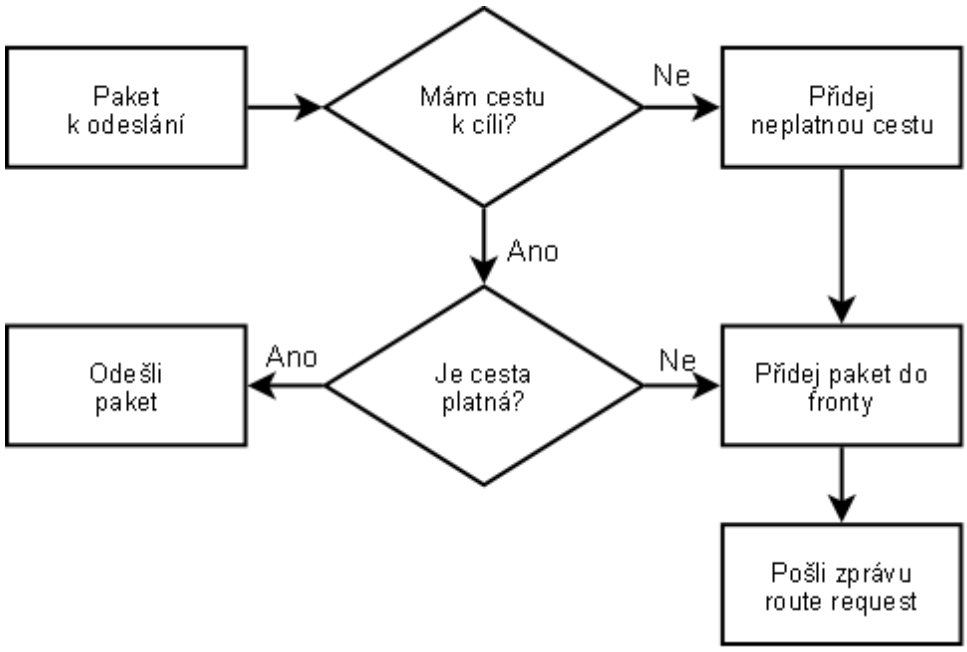
4.2 Vývojové diagramy

Vývojový diagram je grafické znázornění konkrétního algoritmu nebo procesu. Vývojový diagram používá pro znázornění jednotlivých dílčích operací značky, které jsou navzájem propojeny pomocí orientovaných šipek tak, aby byl zřejmý směr průchodu algoritmu. Vývojový diagram má nečastější použití v informatice a programování. Méně časté, ale taky možné, je využití v managementu.

Vývojové diagramy bývají často součástí dokumentací projektů. A nejinak tomu je i v mém případě. Vývojovými diagramy bude znázorněna konkrétní činnost protokolu při odeslání paketu, při přijetí *route request* zprávy a při přijetí *route reply* zprávy.

4.2.1 Odeslání paketu

Pokud uzel obdrží od vyšší vrstvy paket, který má poslat cílovému uzlu, tak nejdříve vyhledá ve své směrovací tabulce záznam pro cílový uzel. Pokud takový záznam ve směrovací tabulce nemá, tak vytvoří nový záznam, ale cestu zatím označí za neplatnou. Paket vloží do fronty, a pak rozesláním *route request* zprávy spustí *route discovery* proces. Pokud uzel ve směrovací tabulce najde záznam pro daný cíl, tak ověří, jestli je cesta platná. Když je cesta platná, paket se odešle. Pokud cesta platná není, vloží se paket do fronty a rozešle se *route request* zpráva. Celý proces je znázorněn na obrázku Obr. 4.2.1.



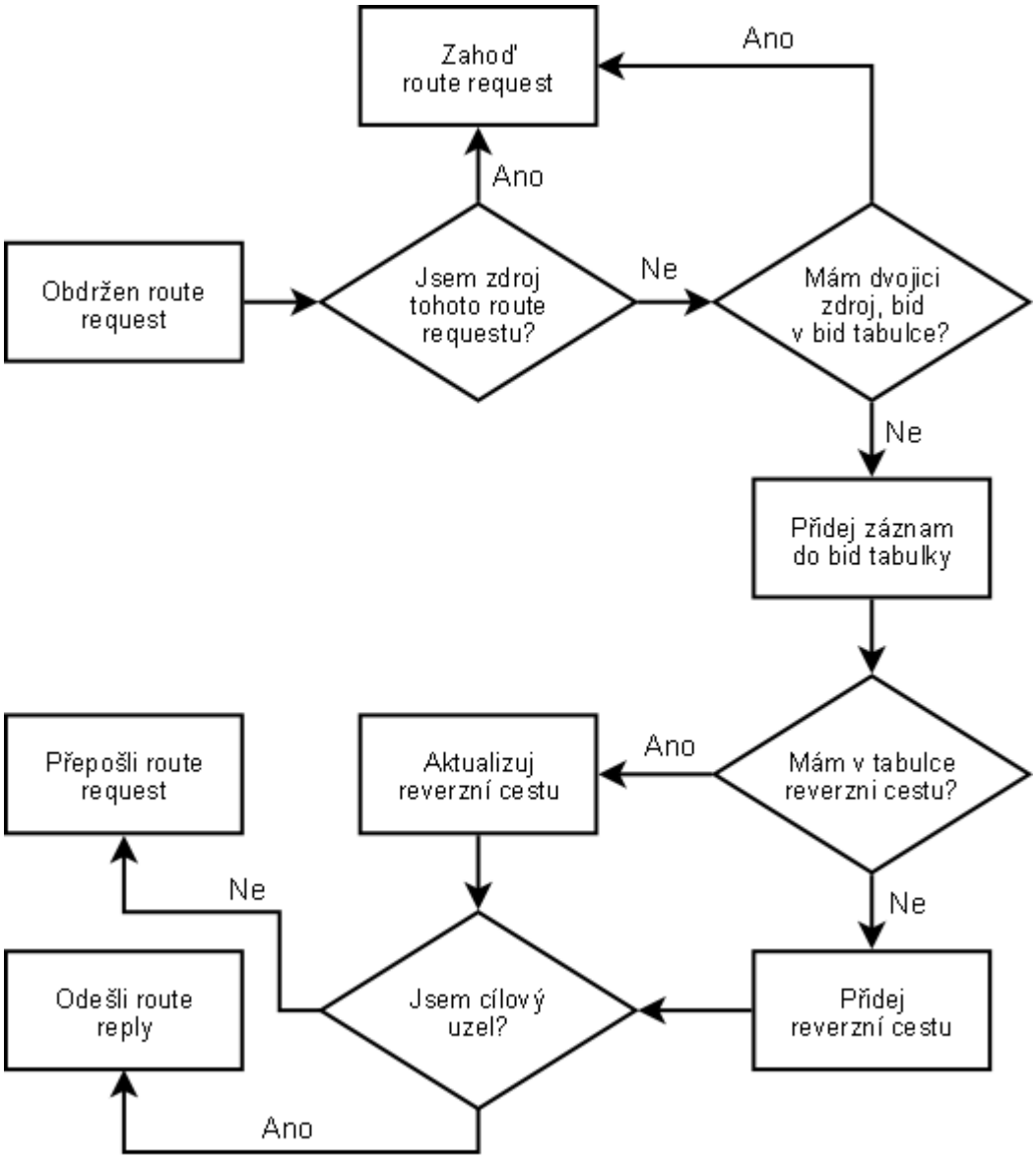
Obr. 4.2.1 – Vývojový diagram – odeslání paketu

4.2.2 Přijetí *route request* zprávy

Zpracování obdržené *route request* zprávy je následující. Nejdříve uzel ověří, zda-li neobdržel svůj vlastní požadavek. Pokud ano, zahodí jej. Dále uzel ověří, zda-li již požadavek se stejnou zdrojovou adresou a stejným broadcast-id nemá v broadcast-id tabulce. Pokud má, zahodí tento požadavek.

Pokud nemá, přidá do broadcast-id tabulky záznam se zdrojovou adresou a s broadcast-id z *route request* zprávy. Zpracování pokračuje ověřením existence reverzní cesty ke zdroji. Pokud tato cesta neexistuje, je vytvořena, pokud existuje je aktualizována. Reverzní cesta je cesta, kterou využije *route reply* zpráva při cestě ke zdroji.

Další zpracování záleží na tom, zda-li je daný uzel cílovým uzlem pro *route request* zprávu. Pokud je uzel cílovým uzlem, tak odešle zdroji *route reply* zprávu. Případě že se nejedná o cílový uzel, rozešle se *route request* zpráva dál. Celý proces zpracování *route request* zprávy je znázorněn na obrázku Obr. 4.2.2.

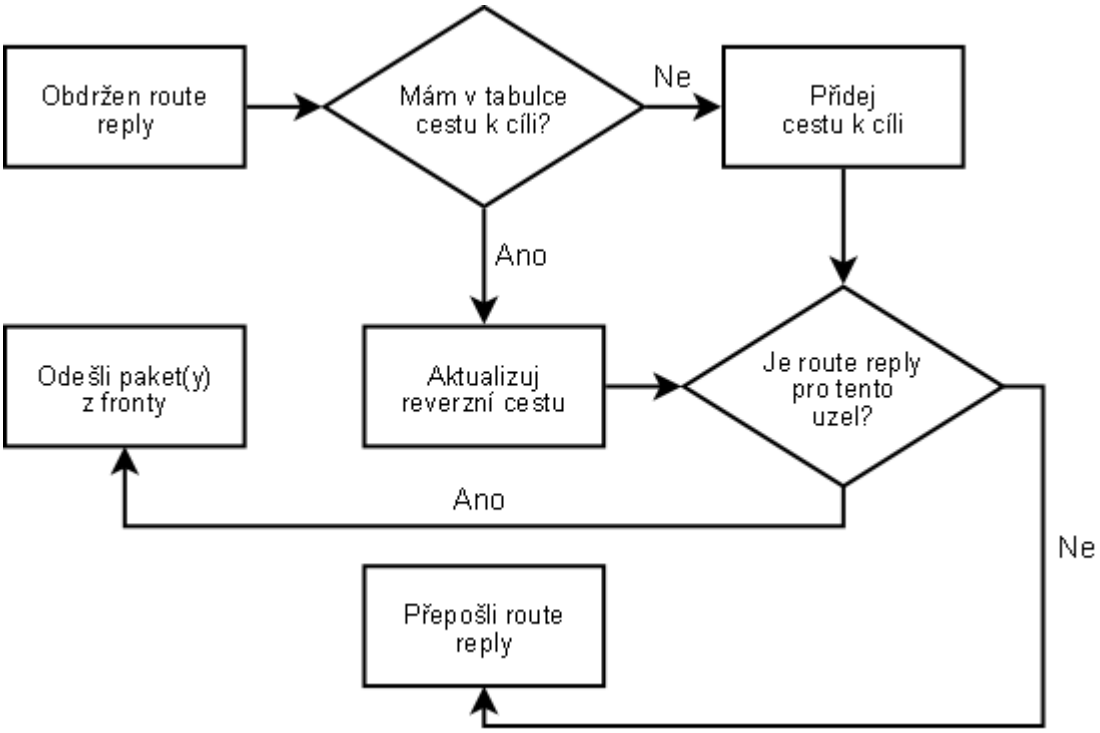


Obr. 4.2.2 – Vývojový diagram – přijetí *route request*

4.2.3 Přijetí *route reply* zprávy

Pokud uzel přijme *route reply* zprávu, postará se nejdříve o cestu k cíli. Pokud už tuto cestu ve směrovací tabulce má, tak ji aktualizuje, pokud nemá, tak ji vytvoří. Když je *route reply* zpráva

určena pro tento uzel, znamená to, že *route discovery* proces byl úspěšně dokončen a uzel tedy může odeslat pakety čekající ve frontě. Pokud není *route reply* zpráva určena pro tento uzel, je přeposlána dál podle reverzní cesty získané šířením *route request* zprávy. Zpracování *route reply* zprávy je znázorněno na obrázku Obr. 4.2.3.



Obr. 4.2.3 – Vývojový diagram – přijetí *route reply*

4.3 Analýza

Pro navržený protokol bude potřeba zvolit implementační prostředí, a v závislosti na něm pak vybrat implementační jazyk.

4.3.1 Implementační prostředí

Při volbě implementačního prostředí jsem se rozhodoval mezi dvěma možnostmi. První možností byla implementace pro softwarový simulátor, druhou pak implementace na hardwarových modulech ZigBee. Nakonec jsem se rozhodl pro softwarové řešení. A to z následujících důvodů:

- Simulační možnosti – u simulátoru je možno simulovat protokol pro různé scénáře s různým počtem (větším) uzlů na různé ploše.
- Zhodnocení výsledků – simulátor umožní jednodušší sběr výsledných dat a jejich následné vyhodnocení.
- Srovnání s jinými protokoly – simulátor disponuje jinými už implementovanými protokoly, se kterými je možné provést srovnání.

- Grafický výstup simulace – simulátor může poskytnout průběh simulace v podobě animace přenosu dat a pohybu jednotlivých uzlů.

Se softwarových simulátorů jsem zvolil simulátor ns-2 [14], který disponuje všemi výše popsanými možnostmi.

4.3.2 Jazyk implementace

Volbu implementačního jazyka mi zjednodušilo předchozí rozhodnutí o použití simulátoru ns-2. Protokoly pro tento simulátor se programují z větší části v C++, jen malou část tvoří implementace v Tcl. V Tcl je vytvořeno rozhraní pro simulaci.

4.3.2.1 Jazyk C++

C++ [13] je objektově orientovaný programovací jazyk, který vyvinul Bjarne Stroustrup a další v Bellových laboratořích AT&T rozšířením jazyka C. C++ podporuje několik programovacích stylů (paradigmat) jako je procedurální programování, objektově orientované programování a generické programování, není tedy jazykem čistě objektovým. V současné době patří C++ mezi nejrozšířenější programovací jazyky.

Mezi základní vlastnosti jazyka C++ patří:

- Objekty – jsou pojaty jako přirozené rozšíření datových struktur jazyka C o možnost vkládání členských funkcí. C++ umožňuje řídit viditelnost složek objektů pro ostatní části programu. Pro objekty je možná vícenásobná dědičnost.
- Dědičnost – hlavní myšlenka dědičnosti je znovupoužitelnost, to znamená, že můžeme vytvářet nové třídy založené na třídě, která již byla definována, místo toho abychom museli znovu psát již jednou napsaný kód jen s jinými typy proměnných. Díky dědičnosti je možné napsat kód jednou pro obecnější typ a poté ho používat pro všechny jeho potomky.
- Šablony – šablony dále rozšiřují znovupoužitelnost kódu, neboť umožňují napsat kód se zcela obecným (neurčeným) datovým typem. Jsou užitečné především pro základní typy, které v C++ nejsou objekty: mnohé jiné jazyky mohou dosáhnout stejné funkcionality použitím kořene objektové hierarchie.
- Přetěžování (Polymorfizmus) funkcí a operátorů – jazyk C++ umožňuje deklarovat více funkcí se stejným názvem. Kompilátor určí správné použití podle počtu a typu parametrů. Tato technika se nazývá přetěžování funkcí. Velmi silnou vlastností jazyka je i možnost přetěžovat standardní operátory (například '+' nebo '='), a tak přirozeně využívat tyto operátory pro nově vytvářené třídy a tvorbu abstraktních datových typů.
- Standardní knihovna – standard jazyka C++ z roku 1998 se skládá ze dvou částí: popis jazyka a standardní knihovny. Standardní knihovna jazyka C++ obsahuje mírně modifikovanou verzi standardní knihovny jazyka C a Standard Template Library (STL).
- Standard Template Library – obsahuje velké množství užitečných datových struktur a algoritmů, jako například vektory (vylepšené pole), spojové seznamy, iterátory, zobecněné ukazatele, (multi)mapy, (multi)sety. Všechny tyto struktury mají

konzistentní rozhraní. S použitím šablon je pak možné programovat generické algoritmy schopné pracovat s kterýmkoliv kontejnerem nebo sekvencí definovanou iterátory.

4.3.3 Objektový návrh

Při analýze jsem se rozhodl pro implementaci v programovacím jazyce C++. Jazyk C++ je objektově orientovaný, a proto jsem hledal prostředek, který by mi umožnil přehledně a efektivně navrhnout objektový model. Jako nejvhodnější se nakonec ukázal jazyk UML.

4.3.3.1 UML

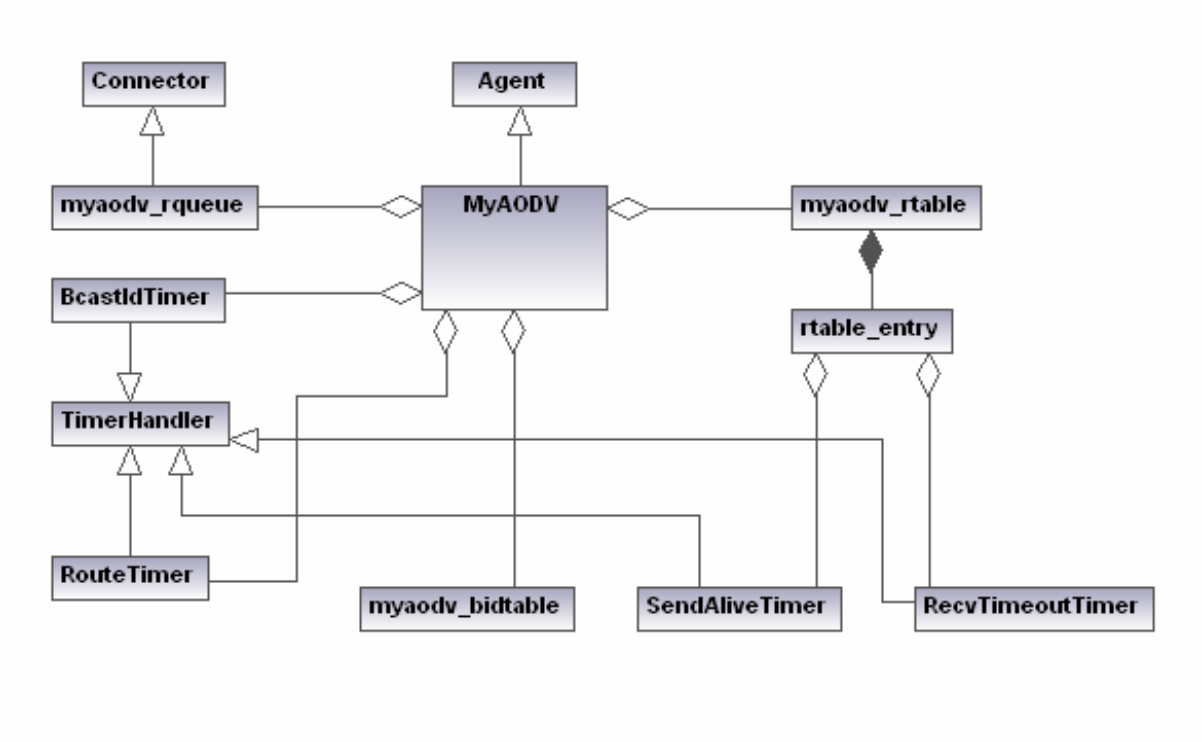
UML (Unified Modeling Language) je jazyk, který vznikl v souvislosti s rozvojem objektově orientovaného návrhu. Jazyk UML zjednodušuje dřívější metody pro objektově orientovaný návrh (Booch, OMT, OOSE). V současné době je UML uznávaným standardem objektově orientovaného návrhu. UML se nevyužívá pouze v softwarovém inženýrství, ale také při návrhu hardwaru, a běžně také při modelování obchodních procesu.

UML poskytuje několik diagramů pro celkový popis systému. Každý diagram nahlíží na daný systém z jiného uhlu.

4.3.3.2 Diagram tříd pro implementaci navrženého protokolu

Pro model implementace protokolu jsem použil pouze jeden z diagramů jazyka UML, a sice diagram tříd. Jedná se o statický typ diagramu, který popisuje struktury systému zobrazením jednotlivých tříd a vztahů mezi nimi.

Na obrázku Obr. 4.3.1 je výsledný diagram implementace protokolu. Diagram je kvůli přehlednosti zjednodušen. Neposkytuje žádné informace o atributech a funkcích jednotlivých tříd. Tyto podrobnosti budou uvedeny až v implementační části práce. Z diagramu je patrné, že třída `MyAODV` dědí od třídy `Agent` a je stěžejní třídou celého směrovacího agenta. Třída `MyAODV` je ve vlastnickém vztahu s většinou ostatních tříd, tedy objekt třídy `MyAODV` má k dispozici instance těchto tříd a může využívat jejich funkce. Diagram obsahuje také čtyři třídy, které dědí od třídy `TimerHandler`, čili se jedná o časovače. Jsou to třídy `BcastIdTimer`, `RouteTimer`, `SendAliveTimer` a `RecvTimeoutTimer`. Podrobnější popis všech tříd včetně jednotlivých atributů a funkcí a jejich význam pro funkci směrovacího agenta je součástí podkapitoly 5.1.



Obr. 4.3.1 – Diagram tříd směrovacího agenta

5 Implementace

Protokol je navržen a nyní zbývá ho implementovat a integrovat do simulátoru ns-2. Při analýze byl zvolen jazykem implementace jazyk C++. V podkapitole 5.1 jsou popsány třídy, jejichž objekty tvoří jádro implementovaného směrovacího agenta.

5.1 Popis implementovaných tříd

Struktura celého systému tříd je znázorněna na obrázku Obr. 4.3.1. V následujícím textu budou podrobněji popsány některé důležité třídy a jejich význam pro funkci směrovacího agenta.

5.1.1 Třída MyAODV

Jedná se o hlavní třídu celé implementace. Objekt této třídy je vlastně samotným směrovacím agentem. Tento objekt disponuje objekty ostatních tříd a využívá je ke správné funkčnosti směrovacího protokolu.

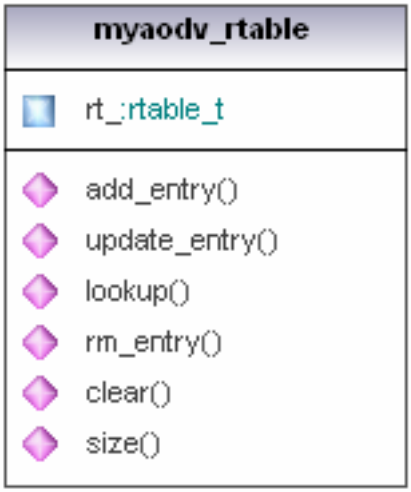
Obr. 5.1.1 – Třída MyAODV

Na obrázku Obr. 5.1.1 je digram třídy `MyAODV`, z nějž jsou patrné jednotlivé atributy a funkce. Za zmínku stojí atribut `ra_addr_`, což je adresa uzlu, na kterém se směrovací agent nachází. Další atributy jsou `rtable_`, což je objekt třídy `myaodv_rtable`, tedy je to směrovací tabulka (více v sekci 5.1.2), `rqueue_` je fronta na pakety (samotná třída bude popsána v sekci 5.1.3), `bidtable_` je *broadcast-id* tabulka (více v 5.1.4). `Bid_` je aktuální *broadcast-id* daného agenta. Poslední dva atributy jsou časovače pro kontrolu platnosti cest (`route_timer_`) a pro promazávání *broadcast-id* tabulky (`bcast_id_timer_`). Časovačům je věnována sekce 5.1.5.

Objekt třídy `MyAODV` obsahuje také množství funkcí. Většina souvisí s odesílání a přijímáním zpráv a dat. Všechny přijaté pakety jsou nejdříve obslouženy funkcí `recv()`, která volá další funkce podle typu dat v paketu. Zprávy směrovacího protokolu jsou obslouženy nejdříve funkcí `recv_myaodv_pkt()`, a pak konkrétně podle typu zprávy funkcemi `recv_reply()`, `recv_request()` a `recv_alive()`. Zpracování ostatních paketů provádí funkce `resolve()`. Přeposlání dat provádí funkce `forward_data()` a `forward_data_bcast()`. Odesílání jednotlivých zpráv směrovacího protokolu provádí funkce `send_request()`, `send_reply()` a `send_alive()`.

5.1.2 Třída myaodv_rtable

Objektem této třídy je směrovací tabulka, kterou potřebuje ke své činnosti směrovací agent. Záznam ve směrovací tabulce obsahuje všechny položky, které byly navrženy v podkapitole 4.1.3 a navíc obsahuje reference na časovače `SendAliveTimer` a `RecvTimeoutTimer` (více v sekci 5.1.5) Diagram třídy `myaodv_rtable` je znázorněn na obrázku Obr. 5.1.2.



Obr. 5.1.2 – Třída myaodv_rtable

Jediným atributem této třídy je `rt_`, což je samotná datová struktura uchovávající záznamy. Použil jsem datový kontejner `std::map`, v němž se vyhledává podle cílové adresy. Významy funkcí třídy myaodv_rtable, jsou zřejmé z názvů. `Add_entry()` přidá nový záznam do tabulky, `update_entry()` aktualizuje stávající záznam, `lookup()` vyhledá v tabulce záznam pro konkrétní cílovou adresu a vrátí ukazatel na tento záznam, `rm_entry()` odstraní záznam z tabulky, `clear()` smaže celou tabulku a `size()` vrací počet záznamů v tabulce.

5.1.3 Třída myaodv_rqueue

Objekt této třídy zastává funkci odkládací fronty na čekající pakety. Jednotlivé atributy a funkce této třídy jsou znázorněny na obrázku Obr. 5.1.3.

Obr. 5.1.3 – Třída myaodv_rqueue

Atributy `head_` resp. `tail_` jsou ukazateli na první resp. poslední paket ve frontě, atribut `len_` udává aktuální počet paketů ve frontě, `limit_` je maximální velikost fronty a `timeout_` je maximální doba, po kterou může být paket ve frontě. Atributy `limit_` a `timeout_` jsou iniciovány v konstruktoru konstantami.

Funkce `push()` vloží paket na konec fronty, `pop()` odebere první paket ve frontě, nebo první nalezený paket s danou cílovou adresou, `find()` vrací `true` pokud je ve frontě paket s danou cílovou adresou, jinak vrací `false`. Funkce `findPacketWithDst()` je privátní funkce, která najde paket s cílovou adresou nastaví na něj ukazatel. Funkce `findAgedPacket()` hledá pakety, kterým již vypršela maximální doba pobytu ve frontě, a nastaví ukazatel na první takový nalezený paket. `Remove_head()` pouze odstraní první paket ve frontě. `Purge()` provádí kontrolu celé fronty a odstraňuje všechny prošlé pakety.

5.1.4 Třída myaodv_bidtable

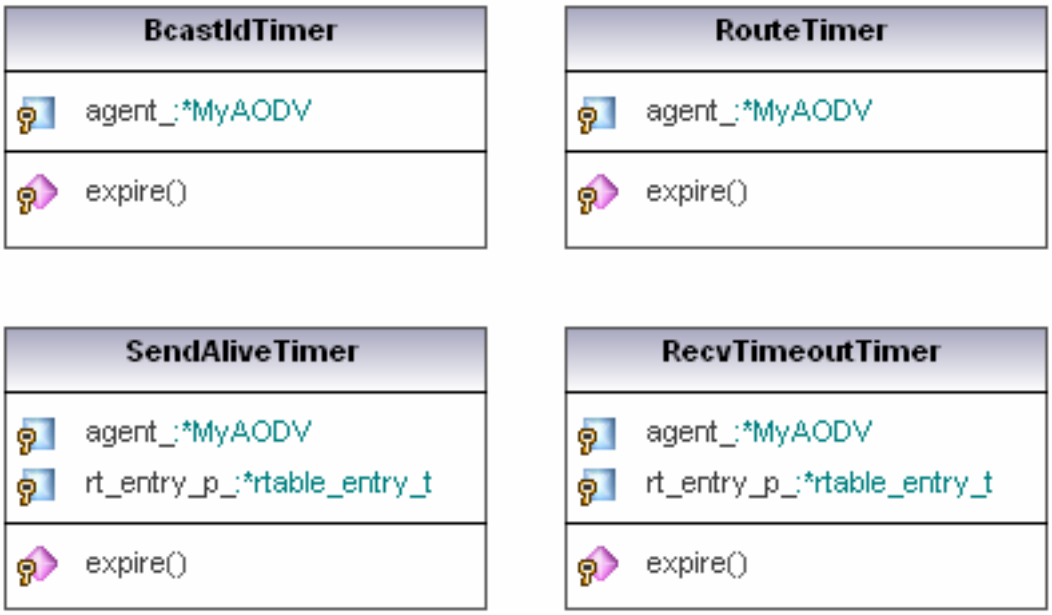
Objekt této třídy udržuje záznamy o *broadcast-id* přijatých v *route request* zprávách. Položky záznamu jsou stejné, jako byly navrženy v sekci 4.1.4. Na obrázku Obr. 5.1.4 je znázorněn diagram třídy `myaodv_bidtable`.

Obr. 5.1.4 – Třída myaodv_bidtable

Jediným atributem je `bidt_`, což je samotná datová struktura pro ukládání záznamů. Struktura je implementována objektem třídy `std::list`.. Význam většiny funkcí odpovídá běžným funkcím pro práci se seznamem. Funkce `add_bid()` přidá záznam, `lookup_bid()` vrací `true` pokud byl daný záznam nalezen, jinak vrací `false`. Funkce `id_purge()` je volána periodicky a promazává staré záznamy, funkce `rm_bid()` smaže záznam a funkce `size()` vrací počet záznamů tabulky.

5.1.5 Třídy s časovači

Směrovací agent využívá čtyři různé časovače. Jedná se o objekty tříd `BcastIdTimer`, `RouteTimer`, `SendAliveTimer` a `RecvTimeoutTimer`. Na obrázku Obr. 5.1.5 je jejich diagram tříd.



Obr. 5.1.5 – Třídy časovačů

Všechny objekty těchto tříd mají atribut `agent_`, což je ukazatel na agenta, který těmito objekty disponuje. Je to z důvodu, aby časovače mohly volat některé agentovy metody. Společná je také funkce `expire()`, kterou musí implementovat každý časovač a která je provedena při vypršení daného časovače.

Časovač `BcastIdTimer` v pravidelném intervalu promazává staré záznamy z *broadcast-id* tabulky. Původní nastavení je 6 sekund, tuto hodnotu lze změnit. Objekt třídy `RouteTimer` v pravidelných intervalech kontroluje platnost cest. Kontrola probíhá vždy po půl sekundě.

Časovače `SendAliveTimer` a `RecvTimeoutTimer` obsahují navíc atribut `rt_entry_p_`, který je ukazatelem na záznam v směrovací tabulce. Toto je drobné rozšíření oproti návrhu, kde nebyly tyto časovače původně navrženy. Časovač třídy `SendAliveTimer` při vypršení posílá *ALIVE* zprávu zdroji dat. Tento časovač je tedy vždy součástí záznamu pro cestu směrem ke zdroji dat. Interval je 1 sekunda, odeslaná data ke zdroji tento časovač restartují. Protikladem je časovač třídy `RecvTimeoutTimer`, který kontroluje přijaté pakety od cíle, pokud během daného intervalu (4 s) neobdrží od cíle *ALIVE* zprávu nebo datový paket, je cesta k cíli označena za neplatnou. Tento timer je tedy součástí záznamu s cestou k cíli dat.

5.2 Implementace zpráv protokolu

Byly implementovány tři zprávy, které byly navrženy již dříve v sekci 4.1.2. Jedná se o zprávy *route request*, *route reply* a *ALIVE*. Hlavičky jednotlivých zpráv byly implementovány jako obyčejné struktury (více v 1. příloze) s položkami stejnými, jako byly navrženy v sekcích 4.1.2.1, 4.1.2.2 a 4.1.2.3.

V rámci simulace jsou tyto zprávy zasílány zapouzdřené do běžných paketů, které ns-2 simulátor používá pro všechny přenosy mezi uzly.

5.3 Integrace protokolu do simulátoru ns-2

Abychom mohli implementovaného směrovacího agenta použít k simulacím v simulátoru ns-2, je potřeba provést několik nezbytných změn přímo ve zdrojových kódech simulátoru, a poté jej znovu přeložit.

5.3.1 Deklarace typu paketu

Aby mohl simulátor pracovat s pakety nového směrovacího protokolu, je třeba do souboru *common/packet.h* přidat konstantu `PT_MYAODV`. V tomtéž souboru je také nutné zadat textové pojmenování našeho paketu. Toto jméno bývá pak zobrazeno v souborech s výstupy simulace.

5.3.2 Podpora pro textový výstup simulace

Výsledkem simulace je soubor (trace) s informacemi o událostech, kdy a kým byl paket poslán, přijat nebo zahozen. Aby mohl simulátor tyto informace zapisovat i o paketech nového směrovacího protokolu, je potřeba do souborů *trace/cmu-trace.h* a *trace/cmu-trace.cc* dopsat funkci *format_myaodv()* pro zpracování daných paketů.

5.3.3 Tcl knihovna

Do souboru *tcl/lib/ns-packet.tcl* je nutné přidat název protokolu, aby simulátor věděl, že pro něj existuje speciální typ paketů. A do souboru *tcl/lib/ns-lib.tcl* je nutné přidat kód pro vytvoření nového bezdrátového uzlu, který bude moci používat navržený směrovací protokol.

5.3.4 Prioritní fronta

Pro funkci směrovacího protokolu je žádoucí, aby jeho pakety byly upřednostňovány před ostatními pakety. To se zajistí přidáním typu paketu `PT_MYAODV` do souboru *queue/priqueue.cc*.

5.3.5 Makefile

Do souboru *Makefile* je potřeba přidat názvy souborů se zdrojovými kódy protokolu a je nutné spustit znovu kompilaci, aby došlo k integraci nového protokolu do simulátoru.

6 Simulace

Funkčnost navrženého a implementovaného protokolu byla testována simulacemi v softwarovém simulátoru. Bylo provedeno několik simulací s různými simulačními scénáři. Jak již bylo uvedeno v kapitole o návrhu, použitým simulátorem byl ns-2.

6.1 Ns-2 simulátor

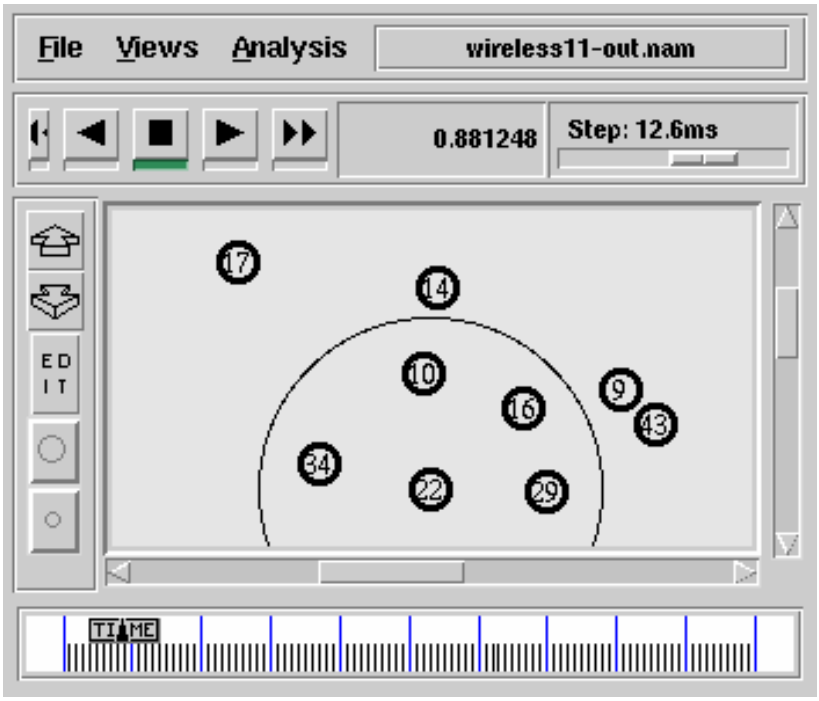
Ns-2 (Network simulator version 2) [14] je bezplatný a volně šiřitelný simulátor zaměřený na výzkum různých počítačových sítí. V současné době je jedním z nejpoužívanějších simulátorů. Své široké uplatnění nachází hlavně v akademické oblasti a ve výzkumu při návrhu, testování a vyhodnocování nových i stávajících protokolů a architektur. Je také velmi užitečným nástrojem pro účely výuky. V době vzniku této práce byla aktuální verze ns-2 verze 2.32. Do budoucna se chystá také vydání simulátoru Ns-3, ale ten je zatím pouze ve stádiu vývoje.

Ns-2 je diskrétní simulátor. Jeho možnosti v oblasti síťové simulace jsou rozsáhlé, například dokáže simulovat TCP a UDP datové přenosy a směrovací a multicastové protokoly. Díky tomu, že jsou jeho zdrojové kódy volně dostupné, je možné doimplementovat různé funkce a nové protokoly. Simulace je možné provádět jak na běžných drátových sítích, tak na sítích bezdrátových (lokálních i satelitních). Časté je použití pro simulace ad hoc bezdrátových síti.

Základními stavebními kameny simulátoru jsou uzly a linky (v případě drátových sítí), které mezi nimi vytvářejí konkrétní topologie. Každý uzel má jednu unikátní adresu. Pro použití více adres by bylo potřeba doimplementovat podporu pro více rozhraní. Na uzly jsou navázáni různí agenti. Agenti reprezentují protokoly (různých vrstev síťového modelu), a vytvářejí a zpracovávají pakety (např. TCP agenti). Agenti jsou také použiti pro implementaci směrovacích protokolů, což je právě případ této práce.

6.2 Nam – network animator

Součástí balíku s ns-2 simulátorem je také grafický animátor Nam. Tento animátor umožňuje grafické znázornění simulované topologie. Dokáže také animovat přenášená data a pohyb uzlů. Lze nastavit rychlost animace, animaci zastavit a libovolně posouvat po časové ose. Znázornění simulace je provedeno na základě speciálního textového výstupu simulace. Ukázka grafického uživatelského rozhraní animátoru je na obrázku Obr. 6.2.1.



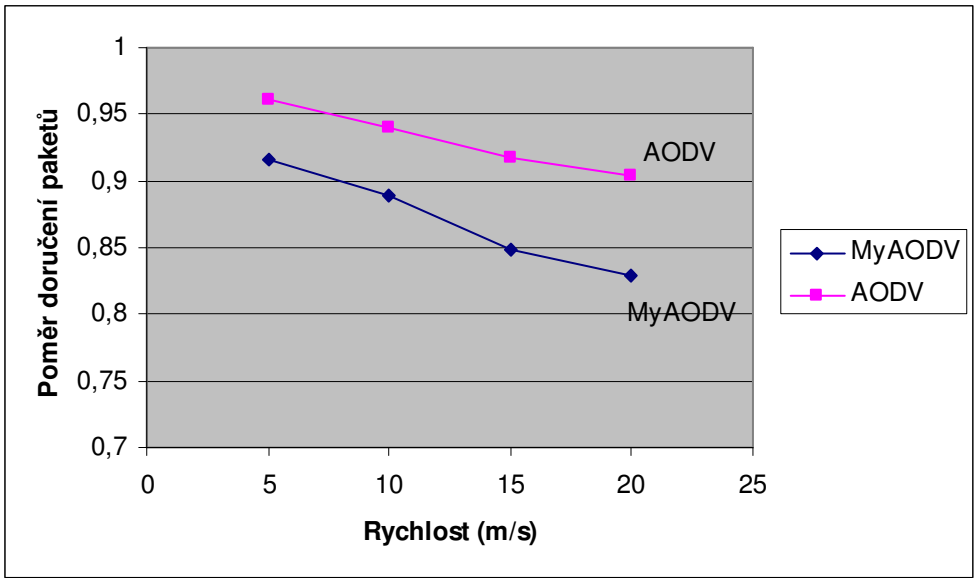
Obr. 6.2.1 – Nam – grafické uživatelské rozhraní

6.3 Simulace implementovaného protokolu

Pro zjištění výkonnostních charakteristik implementovaného protokolu bylo provedeno množství simulací s různými scénáři. Zkoumán byl počet kontrolních paketů směrovacího protokolu, úspěšnost doručení paketů, normalizovaná zátěž směrovacího protokolu a reálný čas simulace. Výsledky byly porovnány s protokolem AODV (byla použita implementace AODV kompatibilní s RFC 3561, [15]). Každá simulace je provedena podle simulačního skriptu, který určuje jednotlivé parametry simulace, jako jsou počet uzlů, rozmístění uzlů, pohyb uzlů, určení uzlů, které budou zdroji dat, určení cílů komunikace a další. Skripty jsou psány v jazyce Tcl, což je interpretovaný jazyk často používaný jako nástroj řízení různých aplikaci, to je přesně případ ns-2.

6.3.1 Úspěšnost doručení paketů

Jedná se o poměr paketů odeslaných zdrojovými uzly a přijatých cílovými uzly. Výsledky pro tuto výkonnostní charakteristiku byly získány simulací následujících scénářů. Na ploše 750 m x 750 m bylo rozmístěno 50 uzlů. Dosah uzlu je 240 m. Zdrojem dat bylo 10 uzlů. Tyto uzly posílají každou sekundu 4 pakety o velikosti 512 bajtů. Šířka pásma byla 1 Mbit. To však nebylo pro simulaci důležité. Cílem bylo zjistit, jak jsou pakety doručovány v podmínkách mobilního prostředí a nikoliv zatížení přenosové kapacity. Simulace byly provedeny s maximálními rychlostmi pohybu uzlů 5, 10, 15 a 20 m/s. Pro každou rychlost bylo provedeno 5 simulací o délce 500 sekund s různými topologiemi a různým náhodným pohybem. Zdroje dat byly pro všechny simulace stejné. Hodnoty pěti simulací pro každou rychlost byly nakonec zprůměrovány. Výsledný graf je na obrázku Obr. 6.3.1.

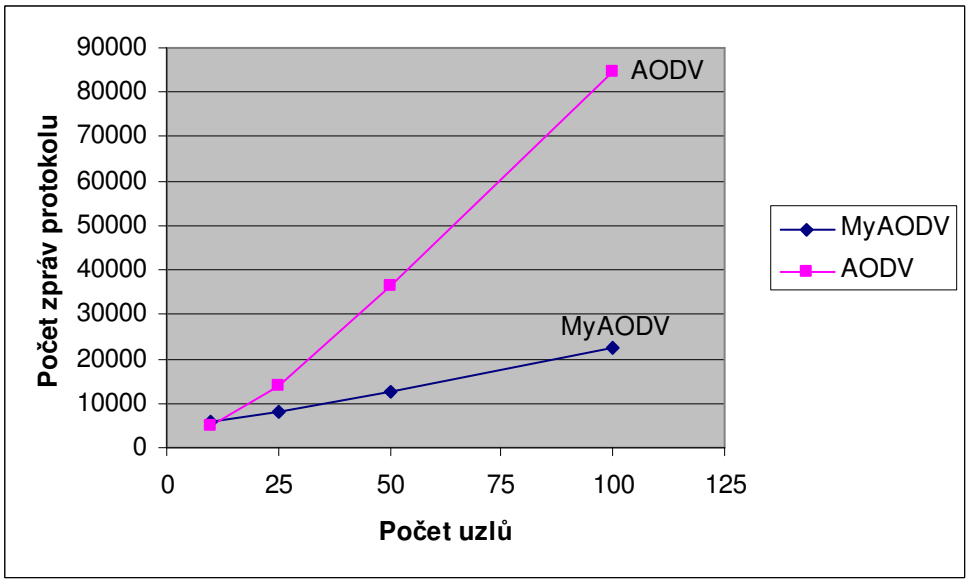


Obr. 6.3.1 – Úspěšnost doručení paketů

Z grafu na obrázku Obr. 6.3.1 je patrné, že s rostoucí rychlostí pohybu uzlů klesá procento doručených paketů. To je pochopitelné, protože dochází k častým změnám topologie a tím pádem i k výpadkům linek. V porovnání se standardním AODV nově navržený protokol mírně zaostává. Rozdíl však nepovažuji za zásadní.

6.3.2 Počet zpráv směrovacího protokolu

Druhou zkoumanou charakteristikou bylo množství zpráv směrovacího protokolu. Počítá se každé poslání i přeposlání zprávy. Postupně byly provedeny simulace s 10 uzly na ploše 350 m x 350 m, 25 uzly na ploše 500 m x 500 m, 50 uzly na ploše 750 m x 750 m a 100 uzly na ploše 1000 m x 1000 m.Vždy bylo použito 10 datových zdrojů a maximální rychlost pohybu byla 5 m/s.



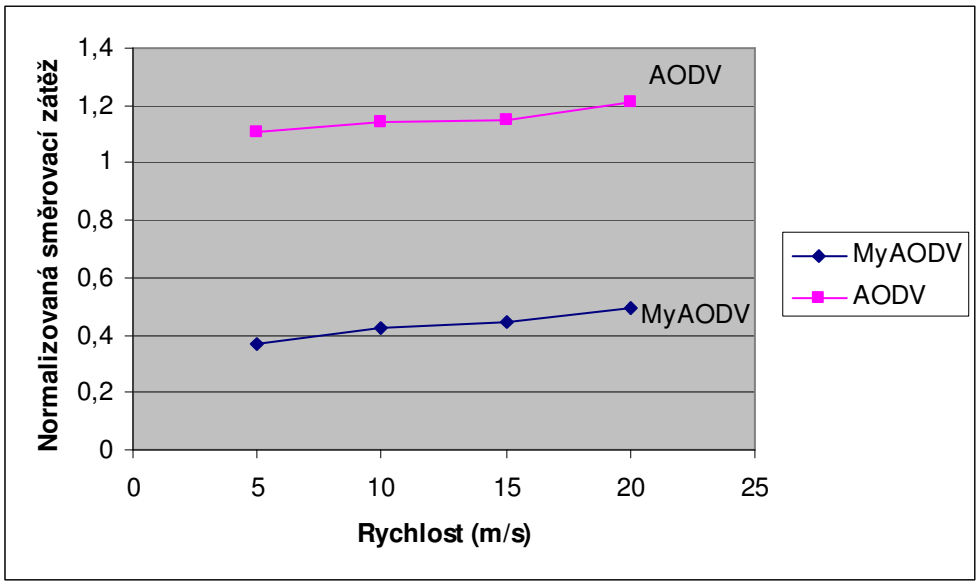
Obr. 6.3.2 – Počet zpráv směrovacího protokolu

Parametry uzlu jako množství poslaných paketů za sekundu, šířka pásma a dosah uzlu byly stejné jako u simulací v sekci 6.3.1. Také délka simulace byla 500 sekund. Pro jednotlivé počty uzlů bylo provedeno po pěti simulacích a výsledky byly opět zprůměrovány.

Na obrázku Obr. 6.3.2 je znázorněn graf s počty zpráv směrovacích protokolů. Je z něj patrné, že s rostoucím počtem uzlů roste také počet zpráv směrovacího protokolu. Nově navržený protokol však oproti standardnímu AODV dosahuje mnohem lepších výsledků. Je to způsobeno tím, že ve standardním AODV posílá každý uzel *HELLO* zprávy a navíc jsou zasílány také zprávy oznamující chybu linky. Kdežto u nově navrženého protokolu jsou posílány pouze *ALIVE* zprávy a to pouze na aktivních cestách. Navíc pokud je komunikace obousměrná nejsou zprávy *ALIVE* zasílány vůbec. U obou protokolů jsou samozřejmě posílány zprávy *route request* a *route reply*. Menší počet zasílaných zpráv by se měl projevit nejen menším zatížením sítě, ale také menší spotřebou energie jednotlivých uzlů.

6.3.3 Normalizovaná zátěž směrovacího protokolu

Tato charakteristika je poměrem počtu zpráv směrovacího protokolu a počtu datových paketů. Počítá se odeslaní i přeposlání zprávy nebo paketu. Jedná se tedy vlastně o množství zpráv směrovacího protokolu potřebného pro doručení jednoho datového paketu. Simulace byla provedena s 50 uzly na ploše 750 m x 750 m. Maximální rychlost pohybu uzlů byla postupně 5, 10, 15 a 20 m/s. Parametry uzlu jako množství poslaných paketů za sekundu, šířka pásma a dosah uzlu byly opět stejné jako u simulací v sekci 6.3.1. Také délka simulace byla 500 sekund. Pro jednotlivé rychlosti pohybu uzlů bylo provedeno po pěti simulacích a výsledky byly opět zprůměrovány. Výsledný graf simulace je na obrázku Obr. 6.3.3.



Obr. 6.3.3 – Normalizovaná směrovací zátěž

Z tohoto grafu je vidět, že s rostoucí rychlostí pohybu uzlu, stoupá průměrné množství zpráv směrovacího protokolu potřebné na doručení datového paketu. U protokolu MyAODV je toto množství výrazně nižší než u protokolu AODV. Opět to souvisí s velkým počtem zpráv produkovaných směrovacím protokolem AODV.

6.3.4 Reálný čas simulace

Za zmínku také stojí porovnání reálných časů simulace při použití protokolů MyAODV a AODV. Simulace byly prováděny na Celeronu 1.2 GHz s 640 MB RAM paměti. Všechny následující údaje se týkají simulací se simulačním časem 500 s a jsou vždy průměrem dob 5 simulací se stejným počtem uzlů.

Simulace s 10 uzly trvala 1 minutu (MyAODV i AODV), s 25 uzly trvala 2 minuty (MyAODV) a 4 minuty (AODV), s 50 uzly trvala 3 minuty (MyAODV) a 13 minut (AODV) a se 100 uzly trvala 10 minut (MyAODV) a 40 minut (AODV).

S rostoucím počtem uzlů tedy reálná doba simulace roste. U protokolu AODV však roste mnohem strměji než u MyAODV. To je způsobeno tím, že při použití AODV musí simulátor zpracovávat více událostí souvisejících s velkým počtem zpráv směrovacího protokolu.

6.4 Zpracování výsledků simulace

Ačkoliv je Ns-2 simulátor velmi silným nástrojem, bohužel neposkytuje žádné nástroje pro zpracování výsledků. Pouze simuluje síť a všechny události, které nastali v rámci simulace, zaznamenává do trace souboru. Formát trace souboru je popsán ve 2. příloze této práce. Pro vyhodnocení simulace je tedy potřeba napsat si jednoduché skripty, které trace soubor zpracují. Já jsem si tyto jednoduché skripty napsal v jazyce perl. Skripty jsou obsahem 3. přílohy.

Závěr

Tato práce se zabývala směrovacími protokoly pro ad hoc bezdrátové sítě. Nejdříve byla nastíněna problematika ad hoc sítí. Poté bylo představeno rozdělení směrovacích protokolů pro ad hoc sítě podle různých kritérií. Největší pozornost byla věnována rozdělení podle mechanismu získávání cest. U čtyř protokolů byly popsány jejich algoritmy. Byly to proaktivní protokoly DSDV a CGSR, reaktivní protokol DSR a hybridní protokol ZRP. Dále byl popsán protokol AODV detailně. Tento protokol jsem zvolil, protože se jedná o hodně studovaný a používaný protokol v prostředí ad hoc bezdrátových sítí. Dále byly v práci popsány výhody a nevýhody protokolu AODV. Byly také zmíněny jeho dvě varianty AODVjr a AODVbis.

V další části práce byl navržen upravený směrovací protokol, který se snaží o odstranění některých nevýhod protokolu AODV. Zejména se snaží snížit počet zpráv směrovacího protokolu zasílaných jednotlivými uzly. Využívá v podstatě stejný přístup jako varianta AODVjr. Navržený protokol nepoužívá sekvenční čísla a oproti AODV nemá metriku počet skoků. Na požadavek na získání cesty odpovídá jedině cílový uzel. A odpovídá jen na první obdržený požadavek, což vede k zisku nejrychlejší cesty. Správa cest je prováděna zasíláním speciálních *ALIVE* zpráv od cíle ke zdroji, pokud je přenos dat obousměrný, není těchto zpráv potřeba.

Práce dále popisuje implementaci navrženého protokolu pro simulátor ns-2. Jsou uvedeny výsledky simulací s implementovaným protokolem a jejich porovnání se standardním protokolem AODV. Výsledky ukazují, že se podařilo splnit hlavní cíl, a to snížení počtu zasílaných zpráv směrovacího protokolu, i když za cenu mírného snížení úspěšnosti doručení paketů. To se týká hlavně scénářů s vyšší rychlostí pohybu uzlu. Toto snížení úspěšnosti však není příliš výrazné. Vlivem sníženého množství zasílaných zpráv, se dá předpokládat také snížení spotřeby energie jednotlivých uzlů. Upravený protokol by tak mohl být vhodný pro zařízení s omezenými zdroji energie.

V rámci dalšího vývoje by bylo vhodné pokusit se implementovat upravený protokol na reálném hardwaru a provést další testování a simulace, například právě testy na spotřebu energie jednotlivých uzlů. Dále by mohl být implementovaný protokol rozšířen o nějakou formu zabezpečení.

Přínos práce shledávám hlavně ve vytvoření implementace směrovacího agenta pro simulátor ns-2, který může být využit v dalších simulacích. Užitečná může být taky část práce popisující integraci nového protokolu do simulátoru ns-2. K tomuto simulátoru je sice rozsáhlá dokumentace, ale pro začátečníka je někdy moc složitá a nesrozumitelná. Ze svého pohledu vidím přínos právě v seznámení se s tímto simulátorem a ve zdokonalení svých dovedností v programování v C++.

Literatura

[1] Murthy, C., Manoj, B.: *Ad Hoc Wireless Network:Architectures and Protocols*. New Jersey, Prentice Hall 2004.

[2] Mukherjee, A., Bandyopadhyay, S., Saha, D.: *Location Management and Routing in Mobile Wireless Networks*. Boston, Artech House, 2003.

[3] Perkins C., Belding-Royer E., Das S.: *Ad hoc on demand distance vector (AODV) routing*. RFC 3561, IETF, 2003.

[4] Gomez C., aj.: *Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment*. IEEE, 2006.

[5] Chakeres I., Klein-Berndt L.: *AODVjr, AODV Simplified.*. Mobile Computing and Communications Review, Vol. 6, No. 3, pp. 100-101, July 2002.

[6] Perkins C., Belding-Royer E., Chakeres I.: *Ad hoc On-Demand Distance Vector (AODV) Routing*. draft-perkins-manet-adovbis-01, IETF Internet Draft, 2004.

[7] Perkins C.: *Ad Hoc Networking with AODV*. http://people.nokia.net/charliep/txt/Daedeok2002/AODV-Daedeok.pdf [Online], [rev. 2008-5-10], [cit. 2007-12-29].

[8] Chowdavarapu P., Bhagwat P.: *An Overview of Routing Protocols in Adhoc Networks*. http://voip.netlab.uky.edu/~venu/cs690/adhocprots.doc [Offline], [rev. 2008-5-10], [cit. 2008-1-3].

[9] Perkins C., Puram V.: *Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers*. ACM SIGCOMM, 1994.

[10] Chiang C., Wu H., Liu W., Gerla M.: *Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel*. IEEE, 1997.

[11] Johnson D., Hu Y., Maltz D.: *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*. RFC 4728, IETF, 2007.

[12] Haas Z., Pearlman M., Samar P.: *The Zone Routing Protocol (ZRP) for Ad Hoc Networks*. draft-ietf-manet-zone-zrp-04, IETF Internet Draft, 2002.

[13] Wikipedie.: *C++*. http://cs.wikipedia.org/wiki/C++ [Online], [rev. 2008-5-10], [cit. 2008-5-3].

[14] Fall K., Varadhan K.: *The ns Manual*. http://www.isi.edu/nsnam/ns/doc/index.html [Online], [rev. 2008-5-10], [cit. 2008-5-3].

[15] AODV-UU – CoReSoftware. http://core.it.uu.se/core/index.php/AODV-UU [Online], [rev. 2008-5-10], [cit. 2008-5-3].

Seznam příloh

Příloha 1: Struktury hlaviček zpráv implementovaného protokolu

```
// request zpráva
struct hdr_myaodv_request {

    u_int8_t  type_;        // typ zprávy - MYAODV_RREQ
    nsaddr_t  src_;         // adresa zdrojového uzlu
    nsaddr_t  dst_;         // adresa cílového uzlu
    u_int16_t bid_;         // broadcast ID
    u_int8_t  hop_count_;   // počet skoků
    double    timestamp_;   // čas odeslání RREQ
};


//reply zpráva
struct hdr_myaodv_reply {

    u_int8_t  type_;         // typ zprávy - MYAODV_RREP
    nsaddr_t  src_;          // adresa zdrojového uzlu
    nsaddr_t  dst_;          // adresa cílového uzlu
    u_int8_t  hop_count_;    // počet skoků
    double    timestamp_;    // čas odeslání RREQ
};


//alive zpráva
struct hdr_myaodv_alive {

    u_int8_t  type_;         // Typ zprávy - MYAODV_ALIVE
};
```

Příloha 2: Příklad formátu trace souboru

`r 129.074310030 _31_ RTR --- 1683 cbr 532 [13a 1f 8 800] ------- [8:1 4:1 29 31]`

`r`: přijal (received) – další možnosti: **`s`**: přijal (sent), **`f`**: přeposlal (forwarded), **`D`**: zahodil (dropped)

`129.074310030`: čas události

`_31_`: id (adresa) uzlu

`RTR`: směrování (router) – některé další možnosti jsou **`AGT`**: aplikace (agent), **`MAC`**: mac

`1683`: číslo paketu

`cbr`: typ paketu UDP, další možnosti např. **`DSDV`**: paket protokolu DSDV, **`MyAODV`**: paket implementovaného protokolu pro tuto práci

`532`: velikost paketu v bajtech

`[13a 1f 8 800]`: MAC informace: **`13a`**: očekávané trvání odesílání paketu

`1f`: adresa přijímajícího uzlu (hexa) 31

`8`: adresa odesílajícího uzlu (hexa) 8

`800`: typ přenášených dat IP

`[8:1 4:1 29 31]` IP informace: **8:1**: adresa:port zdrojového uzlu (255 port směr. agenta)

4:1: adresa:port cílového uzlu

29: TTL

31: další skok

Implementovaný protokol přidává za výše uvedený formát další informace např.:

Pro zprávu *route request*

`[1 7 9 1 1]` MyAODV informace: **`1`**: typ zprávy (**request**), **`7`**: adr. zdroje dat, **`9`**: adr. cíle dat

`1`: Broadcast ID, **`1`**: počet skoků

Pro zprávu *route reply*

`[2 5 1 1]` MyAODV informace: **`2`**: typ zprávy (**reply**), **`5`**: adr. zdroje dat, **`1`**: adr. cíle dat

`1`: počet skoků

Pro zprávu *ALIVE*

`[4]` MyAODV informace: **`4`**: typ zprávy (**alive**)

Příloha 3: Skripty na zpracování výsledků simulace

Počet zpráv směrovacího protokolu a normalizovaná zátěž

```
#!/usr/bin/perl
use strict;

if($#ARGV<0){
  printf("Pouziti: <trace-soubor>\n");
  exit 1;
}

open(Trace, $ARGV[0]) or die "Nepodarilo se otevrit dany soubor";

my $rc = 0;  # počet zpráv
my $dc = 0;  # počet zpráv

while(<Trace>){ # čte soubor po řádcich
  my @line = split;  # rozdělení řádku podle mezer

  if($line[3] eq "RTR" && $line[6] eq "MyAODV"){ # MyAODV zprávy
    if($line[0] eq "s" || $line[0] eq "f" ){ # poslané i přeposlané
      $rc++;
    }
  }

  if($line[3] eq "RTR" && $line[6] eq "cbr"){ # cbr pakety
    if($line[0] eq "s" || $line[0] eq "f" ){ # poslané i přeposlané
      $dc++;
    }
  }


}
close(Trace); #zavři soubor

if($rc > 0){
  printf("Normalizovany pocet zprav: %f  Pocet MyAODV: %d \n",
                                                $rc/$dc, $rc);
}
```

Úspěšnost doručených paketů

```
#!/usr/bin/perl
use strict;

if($#ARGV<0){
  printf("Pouziti: <trace-soubor>\n");
  exit 1;
}

open(Trace, $ARGV[0]) or die "Nepodarilo se otevrit dany soubor";

my $sc = 0;  # počet odeslaných
my $rc = 0;  # počet přijatých

while(<Trace>){ # čte soubor po řádcích

  my @line = split; # rozdělení řádku podle mezer

  if($line[3] eq "AGT" && $line[6] eq "cbr"){ # cbr zprávy
    if($line[0] eq "s"){ # odeslané
      $sc++;
    }
    if($line[0] eq "r"){ # přijaté
      $rc++;
    }
  }
}

close(Trace); #zavři soubor

if($rc > 0){
  printf("Pomer: %f\n", $rc/$sc);
}
```

Příloha 4: Tutoriál Ns-2 a nam

Tento tutoriál popisuje postupně instalaci simulátoru ns-2, poté integraci nového směrovacího protokolu do tohoto simulátoru, spuštění simulace, zobrazení průběhu simulace v animátoru nam, vyhodnocení simulace pomocí skriptu.

CD-ROM – představuje kořenový adresář přiloženého cd.

1 Instalace Ns-2

Ns-2 je linuxový simulátor. Ve Windows ho lze provozovat jedině přes Cygwin (více o instalaci přes cygwin k nalezení zde http://www.isi.edu/nsnam/ns/ns-cygwin.html). V době vytváření této práce byla aktuální verze ns-2 v2.32, té se také věnuje tento tutoriál. S jinými verzemi nelze zaručit funkčnost – bylo by potřeba jiného postupu.

V adresáři `CD-ROM/ns2/` se nachází instalační balík `ns-allinone-2.32.tar.gz`. Případně ke stažení na http://sourceforge.net/projects/nsnam/ .

Rozbalení

```
tar xvfz ns-allinone-2.32.tar.gz
```

Instalátor vyžaduje v systému přítomnost následujících balíčků: `gcc`, `gcc-g++`, `gawk`, `tar`, `gzip`, `make`, `patch`, `perl` a `w32api`. A grafické prostředí X11 (balíčky `XFree86-base`, `XFree86-bin`, `XFree86-prog`, `XFree86-lib` a `XFree86-etc` nebo balíčky `xorg-x11-bin`, `xorg-x11-bin-dlls`, `xorg-x11-devel`, `xorg-x11-libs-data`, a `xorg-x11-etc`). Na chybějící balíčky instalátor upozorní.

Spuštění instalace z adresáře `ns-allinone-2.32`

```
./install
```

Na konci instalace je potřeba aktualizovat systémové proměnné prostředí PATH, LD_LIBRARY_PATH a TCL_LIBRARY podle instrukcí zobrazených na konci instalace.

2 Integrace nového protokolu

- Nakopírovat celý adresář `CD-ROM/zdrojove_kody/myaodv` do adresáře `ns-allinone-2.32/ns-2.32/`

- Zkopírovat obsah adresáře `CD-ROM/ns2/integrace/ns-2.32/` do adresáře `ns-allinone-2.32/ns-2.32/` a přepsat všechny soubory.

Upozornění: Soubory v adresáři `CD-ROM/ns2/integrace/ns-2.32/` jsou upravené soubory z verze 2.32 doplněné o potřebné změny protokolu MyAODV. Pokud by se integroval

protokol do jiné verze než 2.32, bylo by potřeba manuálně přidat pouze části kódu týkající se protokolu MyAODV. Jednotlivé změny jsou vysvětleny v technické zprávě v podkapitole 5.3.

- **Důležité**: u některých přepsaných souborů se musí aktualizovat čas poslední změny (např. pomocí příkazu `touch`), jinak by nedošlo k jejich rekompilaci.

```
touch ns-allinone-2.32/ns-2.32/common/packet.cc
touch ns-allinone-2.32/ns-2.32/queue/priqueue.cc
touch ns-allinone-2.32/ns-2.32/tcl/lib/ns-lib.tcl
touch ns-allinone-2.32/ns-2.32/tcl/lib/ns-packet.tcl
touch ns-allinone-2.32/ns-2.32/trace/cmu-trace.cc
```

- Upravit `Makefile` v adresáři `ns-allinone-2.32/ns-2.32/` do `OBJ_CC` přidat následující řádky:

```
myaodv/myaodv.o myaodv/myaodv_rtable.o \
myaodv/myaodv_rqueue.o \
```

před:

```
...
OBJ_CC = \
tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
common/scheduler.o common/object.o common/packet.o \
...
```

po:

```
...
OBJ_CC = \
myaodv/myaodv.o myaodv/myaodv_rtable.o \
myaodv/myaodv_rqueue.o \
tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
common/scheduler.o common/object.o common/packet.o \
...
```

- Teď už zbývá jen překompilovat. V adresáři `ns-allinone-2.32/ns-2.32/` spustit příkaz `make`

3 Simulace

V adresáři `CD-ROM/simulace/skripty/` je uložen simulační skript `simple-wireless.tcl`. Spuštění simulace se provede jednoduše příkazem:

```
ns simple-wireless.tcl
```

Tento skript provádí pouze jednoduchou simulaci 2 uzlů, které jsou nejdříve na stejné pozici, pak se oddálí, přiblíží a zase oddálí mimo svůj dosah. Skript je srozumitelně okomentován.

Výstupem simulace jsou 2 soubory:

`simple.tr` – trace soubor s událostmi simulace (formát popsán v příloze technické zprávy)

`simple-out.nam` – trace soubor pro animátor nam

Zobrazení animace se provede příkazem:

```
nam simple-out.nam
```

V adresáři `CD-ROM/simulace/skripty/` jsou také 2 simulační skripty se složitějším scénářem:

`wireless50.tcl` – simulace s 50 uzly s rychlostí 20 m/s, délka simulace 500s

`wireless100.tcl` – simulace se100 uzly s rychlostí 5 m/s, délka simulace 500s

Ve stejném adresáři jsou také externí soubory s definovanou topologií a zdroji dat:
`scen-1000x1000-100-5-500`, `scen-750x750-50-20-500` a `udp-10-test`

Spuštění simulace opět příkazem:

```
ns nazev_skriptu.tcl
```

Spuštění animace:

```
nam trace_soubor.nam
```

Pro podrobnější informace pro vytváření skriptů s různými scénáři doporučuji výborně napsaný tutoriál od Marca Greise (http://www.isi.edu/nsnam/ns/tutorial/).

Kompletní dokumentaci k Ns-2 lze nalézt na http://www.isi.edu/nsnam/ns/ns-documentation.html

4 Vyhodnocení trace souboru

V adresáři CD-ROM/simulace/perl/ jsou připraveny 2 skripty pro vyhodnocení trace souborů vniklých při simulaci (přípona .tr). Skript `myaodvnormcnt.pl` vyhodnocuje pouze simulace s protokolem MyAODV, pro vyhodnocení jiných protokolů, by bylo potřeba upravit jméno protokolu uvnitř skriptu.

Jsou to skripty:

`myaodvperc.pl` – počítá úspěšnost doručení datových paketů.
`myaodvnormcnt.pl` – počítá normalizovanou směr. zátěž a počet zpráv směr. protokolu.

Přidání práv na spuštění:

```
chmod a+x myaodvperc.pl myaodvnormcnt.pl
```

Použití např:

```
./myaodvperc.pl wireless50-out.tr
```

```
./myaodvnormcnt.pl wireless50-out.tr
```