VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

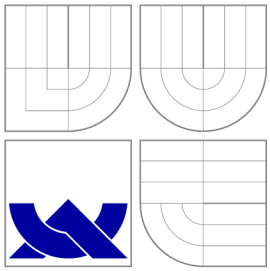
MODEL CISCO SMĚROVAČE V SIMULAČNÍM NÁSTROJI OMNET++

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

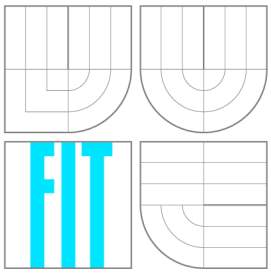
AUTOR PRÁCE
AUTHOR

VLADIMÍR SIVÁK

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODEL CISCO SMĚROVAČE V SIMULAČNÍM NÁSTROJI OMNET++

MODELLING CISCO ROUTER IN SIMULATION TOOL OMNET++

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VLADIMÍR SIVÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR MATOUŠEK, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

V této bakalářské práci se zabýváme simulováním počítačových sítí v simulačním nástroji OMNeT++ s využitím rozšíření INET Framework. Pro INET Framework vytvářímе nový modul, který bude v sobě obsahovat základní funkcionalitu směrovačů firmy Cisco. Je zde popsána implementace potřebných změn v již existujícím modulu, z kterého jsme vycházeli. Tato práce také popisuje základní vlastnosti vybraných směrovacích protokolů a ACL. Na případové studii je porovnáno chování námi vytvořeného modulu s chováním reálné sítě vytvořené v školní laboratoři.

Abstract

This bachelor's thesis describes simulation of computer networks using open source simulator OMNeT++ including the INET Framework extension. According to real behaviour of Cisco routers, we implemented a new modul for the INET Framework extension. Casy study compares behaviour of real network with simulation of the same network in OMNeT++ using the implemented modul.

Klíčová slova

simulace počítačové sítě, OMNeT++, jazyk NED, směrovací protokoly, Cisco

Keywords

computer network simulation, OMNeT++, NED language, routing protocols, Cisco

Citace

Vladimír Sivák: Model Cisco směrovače v simulačním nástroji OMNeT++, bakalářská práce, Brno, FIT VUT v Brně, 2009

Model Cisco směrovače v simulačním nástroji OMNeT++

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Matouška Ph.D.

.....
Vladimír Sivák
19. května 2009

Poděkování

Chcem poďakovať pánovi Ing. Petrovi Matouškovi, Ph.D., ako vedúcemu mojej bakalárskej práce, za jeho odborné vedenie, rady a všetku pomoc, ktorú mi poskytol pri vytváraní mojej práce. Rovnako si poďakovanie zaslúžia všetci spolupracovníci z projektu ANSA, ktorého súčasťou bola moja práca, za vytvorenie priateľského a inšpiratívneho prostredia.

Obsah

Kapitola 1

Úvod

Oblasť počítačových sietí sa v posledných niekoľkých rokoch prudko rozvíjala a aj v blízkej budúcnosti bude tempo vývoja a nasadzovania nových technológií pravdepodobne narastať. Zvyšujú sa hlavne nároky na šírku prenosového pásma, spoľahlivosť použitej technológie a prenosu samotného, bezpečnosť a možnosti administrácie. Konkurencia na trhu núti výrobcov zariadení a dodávateľov komplexných sieťových služieb, aby sa cena riešenia znižovala, alebo aby aspoň narastala primerane k poskytnutým parametrom a službám.

Komplikovanosť novovznikajúcich sietí v podstate neumožňuje zostavenie kompletnej topógie v laboratórnych podmienkach. Testovanie siete pre nasadením do prevádzky je buď úplne nemožné, alebo uskutočniteľné iba v obmedzenej miere. Rovnako pri rozširovaní už existujúcej siete je potrebné poznať dopredu reakciu celej siete na nové zariadenia alebo služby. Výpadok prevádzky musí byť čo najkratší, v ideálnom prípade žiadny.

Uvedené problémy pri testovaní je možné aspoň čiastočne riešiť využitím rôznych simulačných nástrojov a techník. Umožňujú zostavenie celej topológie a simulovanie chovania siete pri rôznych udalostiach. Najčastejšie sa simulujú výpadky liniek a zahltenie prenosového pásma. Okrem reakcie siete na rôzne neplánované udalosti, ako sú napríklad výpadky liniek alebo nadmerná záťaž, môže simulácia sledovať aj parametre, ktoré charakterizujú sieť pri bežnej činnosti. Výsledky sa ukladajú do textových logovacích súborov, aby mohli byť následne analyzované a výhodnotené. Niektoré simulačné nástroje už obsahujú aj utility na grafickú reprezentáciu nameraných dát napr. pomocou grafov alebo histogramov.

Výhody simulačných techník sú hlavne časové a finančné. Počas krátkej simulácie je možné sledovať naraz niekoľko parametrov. Rovnako je možné jednoducho zmeniť počiatočné podmienky a simuláciu opakovať podľa potreby. Náklady na simuláciu predstavujú väčšinou iba potrebu naučiť sa ovládať konkrétny nástroj.

Nevýhody sú spojené hlavne s požadovanou úrovňou abstrakcie, teda do akých detailov je potrebné zachádzať. Simulovanie špecifických vlastností smerovacích protokolov je preto veľmi náročné, alebo aj nemožné. V prípade bezdrátových sietí sa ťažko simuluje napríklad vplyv prostredia. Obmedzujúcim faktorom je pri rozľahlých sieťach celkový výkon počítača.

1.1 Cieľ práce

V bakalárskej práci popisujem simulačný nástroj OMNeT++ a jeho použitie pri simulácií počítačových sietí s využitím rozšírenia INET Framework. Zaoberám sa vytvorením nového modulu pre INET Framework, ktorý bude reprezentovať smerovač firmy Cisco.

1.2 Štruktúra práce

Prvá kapitola opisuje nároky siete v dnešnej dobe a zaoberá sa význam modelovacích a simulačných techník pri návrhu a testovaní sietí.

V druhej kapitole je opísaný simulačný nástroj OMNeT++, jazyk NED (Network Description), ktorý sa používa na popis modelovaných systémov, a rozšírenie nástroja OMNeT++ o moduly na simuláciu prevažne IP sietí - INET Framework.

Základné vlastnosti smerovačov sú opísané v tretej kapitole. Stručne sú chrakterizované jednotlivé smerovacie protokoly a kontrolné prístupové zoznamy.

Aktuálny stav implementácie modulu, ktorý sa stará o dynamické smerovanie, je popísany vo štvrtej kapitole. Bližšie sú charakterizované aj jednotlivé podmoduly a ich funkcie.

Návrhom nového modulu, ktorý bude podporovať funkcie smerovačov firmy Cisco sa zaoberá piata kapitola. Popísané sú aj novovytvorené podmoduly a úpravy vo funkcionalite existujúcich modulov.

Šiesta kapitola obsahuje prípadovú štúdiu zameranú na porovnanie správania sa vytvoreného modulu s reálnymi zariadeniami.

V záverečnej siedmej kapitole sú zhrnuté výsledky celej bakalárskej práce.

1.3 Použité vývojové prostredie

Pre vývoj nových modulov som použil vývojové prostredie, ktoré obsahuje zdrojové súbory OMNeT++ 4.0, prekladač MinGW a grafické užívateľské rozhranie vychádzajúce z programu Eclipse. Celá inštalácia je tak výraznej zjednodušila, lebo spočívala iba v rozbalení archívu a kompilácií zdrojových súborov. Rozšírenie INET Framework som nainštaloval samostatne, pričom inštalácia prebiehala rovnako ako pri OMNeT++ (rozbalenia archívu a kompilácia). Je však potrebné inštalovať INET Framework vo verzii pre OMNeT++ 4.0, lebo najnovšia verzia nie je spätne kompatibilná s predchádzajúcou verziou OMNeT++ 3.0. Vývojové prostredie je dostupné vo verzii pre operačné systémy Windows aj Linux. Ja som použil Windows verziu spolu s operačným systémom Windows XP.

Na porovnanie modelu s reálnymi nástrojmi som použil program Packet Tracer vo verzii 5.0. Je to program od firmy Cisco a slúži na výuku smerovania a prepínania paketov v IP sieťach. Vďaka jednoduchému užívateľskému rozhraniu nevyžaduje jeho použitie pokročilé znalosti a je vhodný aj na základnú simuláciu počítačových sietí.

Kapitola 2

Popis simulačného nástroja OMNeT++

OMNet++ je dikrétny, objektovo orientovaný simulačný nástroj. Jeho autorom je András Varga a verejnosti je prístupný od roku 1997. Bol vytvorený ako simulátor počítačových sietí, jeho architektúra však nie je špecializovaná iba na siete. OMNeT++ bol navrhnutý tak, aby bol čo najviac všeobecný. Je k dispozícii zdarma ako open source pre nekomerčné použitie. Pre komerčné využitie existuje modifikácia OMNEST. Jeho flexibilita spôsobila, že v súčasnosti je OMNeT++ používaný na simulovanie vo viacerých oblastiach.

2.1 Oblasti využitia

Najčastejšie sa využíva pre modelovanie a simuláciu:

- počítačových sietí,
- protokolov,
- multiprocesorových a distribuovaných systémov,
- „business“ procesov.

Platformovo je OMNeT++ nezávislý, dostupné sú verzie pre všetky bežne používané platformy (Linux, Mac OS, Windows). Reprezentuje tzv. rámcový (framework) prístup. Neposkytuje priamo komponenty potrebné pre jednotlivé simulácie, ale poskytuje mechanizmy a nástroje na vytvorenie požadovaných komponent. Každý si teda môže vytvoriť vlastné komponenty presne podľa svojich požiadaviek. Postupne vznikli viaceré sady už hotových modelov, ktoré sú zamerané vždy na určitú oblasť. Napríklad Mobility Framework (MF), určený na modelovanie bezdrátových a mobilných sietí, alebo INET Framework, ktorý je podrobnejšie opísaný v ďalšej sekcii.

2.2 Štruktúra modelov

Model v OMNeT++ ma hierarchickú štruktúru. Skladá sa z viacerých modulov, ktoré sa môžu vzájomne vnorovať. Hĺbka vnorenia pritom nie je obmedzená. Umožňuje to, aby sa

štruktúra modelu čo najviac podobala logickej štruktúre modelovaného systému. Užívateľovi to taktiež umožňuje modelovať systémy s požadovanou úrovňou abstrakcie. Z hľadiska štruktúry sa rozlišujú dva typy modulov: jednoduché (simple) a zložené (compound) moduly. Jednoduché moduly prestavujú najnižšiu logickú úroveň a neobsahujú už žiadne ďalšie vnorené moduly. Implementované sú v programovacom jazyku C++. Zložené moduly vznikajú zoskupením viacerých modulov, nie len jednoduchých, ale aj zložených.

Moduly medzi sebou komunikujú zasielaním správ. Pri simulovaní sietí správy môžu predstavovať rámce (frames) alebo pakety (packets). Každá správa obsahuje „timestamp“ (časovú značku), ktorá udáva v akom čase sa má správa poslať. Okrem toho môžu správy obsahovať ľubovolné dátove štruktúry. Jednoduché moduly väčšinou zasielajú správy cez brány (gates), ale možné poslať ich aj priamo do cieľa. Brány sú vstupné a výstupné rozhrania modulov, pričom každá brána má vopred určené, či bude vstupom pre správy, alebo výstupom. Vstupná brána jedného modulu môže byť spojené s výstupnou bránou druhého modulu, vytvorí sa tak spojenie (connection). Spojenia sa dajú vytvoriť iba na rovnakej hierarchickej úrovni. Podmoduly modulu môžu byť spojené vzájomne medzi sebou iba v rámci nadradeného modulu, spojenie medzi submodulmi dvoch rôznych modulov nie je dovolené. Takéto spojenie cez dve rôzne úrovne by bránilo znovupoužiteľnosti modulu v ďalšej simulácii.

2.3 Jazyk NED

2.3.1 Popis jazyka NED

Na popis štruktúry simulovaného modelu sa používa jazyk *NED* (Network Description). Umožňuje užívateľovi zadeklarovať jednoduché a zložené moduly. Rovnako je využívaný aj na popis celej simulovanej topógie.

2.3.2 Popis jednoduchého modulu

Jednoduché moduly tvoria základ každého modelu. Popis v jazyku NED neobsahuje priamo žiadnu informáciu o funkcionalite jednoduchého modulu. Funkcionalita je implementovaná v jazyku C++. Štandardne OMNeT++ hľadá pri kompilácií C++ triedu s rovnakým názvom ako konkrétny jednoduchý modul.

Príklad jednoduchého modulu:

```
simple TestModul
{
    parameters:
        @display(,,i=block/queue‘‘);
    gates:
        input in;
        output out;
}
```

Kľúčové slovo `simple` definuje meno modulu. V uvedenom príklade ide teda o modul s názvom `TestModul`. Funkcionalita musí byť implementovaná v C++ triede s názvom `TestModul`. Telo popisu obsahuje ďalšie dve sekcie: `parameters:` a `gates:`. Obidve sekcie sú

voliteľné, teda na základnú deklaráciu modulu stačí uviesť jeho názov. Sekcia `parameters:` popisuje vlastnosti modulu. V uvedenom príklade obsahuje iba záznam, ktorý špecifikuje ako sa bude modul zobrazovať v grafickom prostredí GNED (parameter príkazu `@display` nastavuje, aká ikona sa má pre daný modul načítať). Rozhrania modulu, pomocou ktorých je schopný komunikovať s ostatnými modulmi, popisuje sekcia `gates:`. Najskôr je zadaný smer rozhrania a následne jeho názov. Smer je určený príkazmi `input` (vstupné rozhranie), `output` (výstupné rozhranie) alebo `inout` (obojsmerné rozhranie). Uvedený príklad teda obsahuje vstupné rozhranie `in` a výstupné rozhranie `out`.

2.3.3 Popis zloženého modulu

Zložený modul v podstate zapuzdruje viacej modulov do jedného väčšieho celku. Môže byť vytvorený buď priamo z jednoduchých modulov, alebo aj z viacerých zložených modulov. Takto je možné vytvoriť hierarchiu modulov, kde základné funkcie sú implementované jednoduchými modulmi a postupným zapuzdrovaním sa dá vytvoriť komplexný modul s bohatou funkčnosťou.

Príklad zloženého modulu:

```
module Router
{
    parameter:
        @display(,,i=block/router‘‘);
    gates:
        inout SerialInterface[];
        inout EthInterface[];
    submodules:
        tcp: TCP;
        ip: IP;
        layer1: physicalLayer;
    connections:
        tcp.ipVstup <-- ip.tcpOut;
        tcp.ipVystup --> ip.tcpIn;
        layer1.ipIn <-- ip.l1Vystup;
        layer1.ipOut --> ip.l1Vstup;

}
```

Za kľúčovým slovom `module` nasleduje názov celého zloženého modulu, ktorý ale už nepredstavuje meno C++ triedy, ktorá by mala implementovať jeho funkcionalitu. Tá je teraz odvodená priamo z funkcionality jednotlivých podmodulov.

Rovnako ako v prípade jednoduchého modulu, sa v tele deklarácie nachádzajú viaceré sekcie. Sekcie `parameters:` a `gates:` sú rovnako ako pri jednoduchých moduloch. Moduly, z ktorých sa zložený modul skladá, sú vymenované v sekcii `submodules:`. Horeuvedený príklad sa teda skladá z troch podmodulov `tcp`, `ip` a `layer1`. Z uvedeného popisu nie je jasné, ktorý z uvedených podmodulov je jednoduchý, prípadne zložený. Za názvom podmodulu je vždy uvedený aj názov NED súboru, ktorý špecifikuje jeho vlastnosti, rozhrania a pod. Podľa neho OMNeT++ pri preklade zistí aj mená príslušných C++ tried, ktoré popisujú funkcionalitu konkrétneho podmodulu.

V poslednej sekcii `connections:` sú popísané prepojenia jednotlivých podmodulov pomocou ich vlastných rozhraní. Ak je rozhranie typu `input`, spojenie je definované označením `<--`. Naopak pri `output` rozhraní sa používa `-->`. Obojsmerné prepojenie na rozhraní typu `inout` sa definuje pomocou `<-->`. V uvedenom popise prepojenia modulov je zrejmé, že modul `ip` komunikuje pomocou rozhraní `tcpOut`, `tcpIn` s modulom `tcp` a prostredníctvom zvyšných dvoch rozhraní (`l1Vystup`, `l1Vstup`) s modulom `layer1`. Moduly `tcp` a `layer1` môžu spolu komunikovať iba cez modul `ip`, keďže medzi nimi neexistuje žiadne priame prepojenie.

2.3.4 Popis simulovanej topológie

Na najvyššej úrovni abstrakcie sa jazyk NED používa aj na popis celej simulovanej topológie (siete). Sieť je definovaná kľúčovým slovom `network`, za ktorým nasleduje jej názov. V sekcii `submodules:` sú vymenované jednotlivé podmoduly, z ktorých sa simulovaná sieť skladá. Za menom podmodulu nasleduje definícia jeho typu, teda z akého NED súboru sa má načítať jeho popis a v prípade jednoduchého modulu aj názov triedy C++, ktorá popisuje jeho funkcionalitu. Časť `connections` tiež popisuje vzájomné prepojenie jednotlivých komponent, z ktorých sa celá sieť skladá.

```
network SimpleCircle
{
    types:
        channel EthLink extends ned.DatarateChannel {
            datarate = 100Mbps;
        }
    submodules:
        router1: Router;
        router2: Router;
        router3: Router;
        router4: Router;
    connections:
        router1.interface++ <-->  EthLink <--> router2.interface++;
        router2.interface++ <-->  EthLink <--> router4.interface++;
        router3.interface++ <-->  EthLink <--> router1.interface++;
        router3.interface++ <-->  EthLink <--> router4.interface++;
}
```

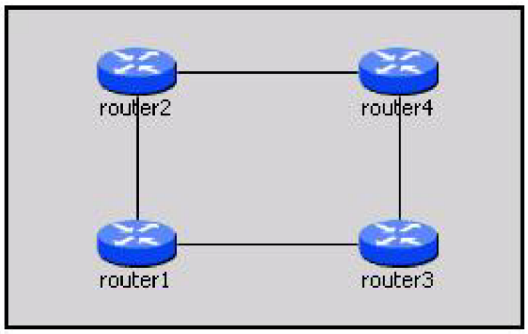
Uvedený príklad popisuje topológiu, ktorá sa skladá zo štyroch smerovačov, ktoré sú vzájomne prepojené tak, aby vytvárali kruhovú topógiu.

Pre zjednodušenie zápisu je v časti `types` vytvorený typ linky, ktorou sú smerovače prepojené. Parameter `datarate` definuje šírku pásma v megabitoch za sekundu. Okrem toho je možné nastaviť aj prípadnú stratovosť a oneskorenie linky. Stratovosť sa špecifikuje kľúčovým slovom `ber` (Bit Error Rate) alebo `per` (Packet Error Rate). Za nimi nasleduje percentuálne vyjadrenie pravdepodobnosti chyby alebo straty paketu. Oneskorenie linky sa definuje pomocou parametru `delay` a časom, o ktorý majú byť pakety oneskorené na strane príjemcu oproti času vyslania.

Pri popise prepojenia modulov je v tomto prípade použitá notácia `interface++`, ktorá pridá ďalšie rozhranie do vektoru rozhraní. V NED súbore popisujúcom modul `Router` je

vektor definovaný ako `interface[]`. Vektor sa používa hlavne pri všeobecných moduloch, kde počet rozhraní nie je dopredu známy.

Na nasledujúcom obrázku je grafické znázornenie simulovanej topológie. Okrem textovej editácie je možné NED súbory upravovať aj pomocou grafického rozhrania. Ľahko sa takto dá upraviť hlavne vizuálne rozloženie topólogie, pri ktorom je úprava pozície textovou editáciou veľmi nepraktická.



Obrázek 2.1: Grafické znázornenie simulovanej topógie

Podrobné informácie o jazyku NED, vrátane formálneho popisu syntaxe ([10]) a ukážok použitia ďalších príkazov, je možné nájst v užívateľskej príručke k nástroju OMNeT++ ([11]).

2.4 INET Framework

INET Framework je nadstavba (rozšírenie) simulačného nástroja OMNeT++. Obsahuje už vytvorené komponenty a moduly určené na simuláciu IP sietí. Podporované sú drátové aj bezdrátové siete. Obsahuje implementáciu protokolov IPv4, IPv6, TCP, UDP, z protokolov linkovej vrstvy sú implementované moduly 802.11, Ethernet a PPP. Zo smerovacích protokolov je podporovaný OSPF a statické smerovanie. Ak pre simuláciu užívateľ potrebuje iba funkcie implementované v INETe, stačí keď pomocou pripravených modulov vytvorí požadovanú topológiu.

Kapitola 3

Základné vlastnosti Cisco smerovačov

Základná úloha smerovačov v počítačových sieťach je smerovanie paketov a výhľadávanie optimálnej cesty. Podporujú rôzne smerovacie protokoly, ktoré na základe rôznych kritérii ohodnotia cesty k požadovaným cieľovým sieťam. Ak ku cieľu existuje viac možných ciest, majú na starosti aj výber tej najvhodnejšej. Pre smerovanie v lokálnych sieťach podporujú Cisco smerovače protokoly RIP, EIGRP a OSFP.

Ďalšie funkcie, ktoré smerovače plnia, sú prepínanie paketov a segmentácia broadcastových domén (nepreposielanie správ typu broadcast). Po správnom smerovaní paketu, teda po výbere jeho optimálnej cesty k cieľu, je potrebné prepnúť ho na správne výstupné rozhranie a vytvoriť korektný rámec. Paket sa musí zapuzdriť s ohľadom na použitý protokol na rozhraní (napr. PPP, FrameRelay) a prepíšu sa adresy MAC.

Okrem toho smerovače poskytujú aj základné zabezpečenie. Na filtrovanie dátového toku používajú tzv. ACL (Access Control List), teda zoznamy na riadenie prístupu.

Pred samotnou implementáciou modulov reprezentujúcich jednotlivé smerovacie protokoly a ACL, sme sa museli s týmito technológiami oboznámiť a pochopiť ich princípy. Táto kapitola sa venuje základným vlastnostiam uvedených protokolov a prístupových kontrolných zoznamov. Pri implementácii boli znalosti podstatných vlastností protokolov a ACL potrebné pre vytvorenie modulov, aby sa čo najvernejšie podobali chovaniu na reálnych zariadeniach.

3.1 Protokol RIP

Smerovací protokol RIP (Routing Information Protocol) bol definovaný v RFC 1058 [1] v roku 1988 a patrí medzi dynamické smerovacie protokoly, konkrétne do skupiny Distance-Vector. Prvotná implementácia bola niekoľkokrát rozšírená, až v roku 1998 vznikla nová implementácia: protokol RIPv2 - RFC 2453 [2]. V minulosti bol protokol RIP používaný v lokálnych (LAN) aj v rozľahlých sieťach (WAN). Momentálne však už nevyhovuje aktuálnym nárokom na smerovacie protokoly a je postupne nahrádzaný protokolmi OSPF a EIGRP.

3.1.1 Technické informácie

Protokol RIP ako smerovaciu metriku používa počet skokov (hop count) na dosiahnutie cieľovej siete. Na výpočet najkratšej vzdialenosti používa Bellman-Fordov algoritmus. Ma-

ximálny počet skokov je 15, čo predstavuje výrazné obmedzenie. Siete, ktoré sú od seba vzájomne vzdialené viac ako 15 skokov (majú medzi sebou 16 alebo viac smerovačov), sú považované za nedosiahnuteľné.

Na výmenu smerovacích informácii sa využívajú správy typu broadcast, ktoré router rozosiela každých 30 sekúnd. Táto vlastnosť však spôsobuje problémy v sieťach, ktoré sa časom rozširujú. Každých 30 sekúnd totiž dochádza k výmenám smerovacích tabuliek medzi smerovačmi v sieti. To spôsobuje zbytočnú záťaž, lebo k výmenách dochádza pravidelne, bez ohľadu na zmeny v topológii, teda sa väčšinou vymieňajú stále rovnaké informácie.

Šíreniu nesprávnych smerovacích informácii zabraňujú mechanizmy split horizont a route poisoning. Split horizont je spôsob šírenia smerovacích informácii, pri ktorých sa zabraňuje šíreniu tej istej informácie naspäť k jej zdroju. V prípade smerovačov sa update nešíri ďalej po rozhraní, z ktorého prišiel. Z dôvodu pomalej konvergencie protokolu RIP pri výpadkoch liniek sa tak zabraňuje vytvorenie cyklu v sieti. Technika route poisoning sa tiež používa na zamedzenie cyklov v sieti. Pri výpadku určitej cesty sa táto cesta označí ako nedostupná, čo zabráni jej ďalšiemu rozširovaniu. Kombináciu oboch predchádzajúcich spôsob vznikol tzv. Split horizont with poison reverse. Keď sa smerovač cez určité rozhranie dozvie informáciu o nedostupnosti určitej cesty, posiela danú informáciu spať cez to isté rozhranie.

Jedným z dôvodov, prečo bol protokol RIP inovovaný a vznikol protokol RIPv2, bol ten, že RIPv1 je triedny (classful) protokol a nezasiela informáciu o maske podsiete. Ak cesta a rozhranie, na ktorom bola prijatá, patria do tej istej siete, uplatní sa maska podsiete aktuálne nastavená na danom rozhraní. V ostatných prípadoch je prijatá cesta automaticky sumarizovaná podľa jednotlivých tried IP adries. Protokol RIPv2 už patrí medzi beztriedne (classless). Okrem toho bola pridaná možnosť autentizácie a na výmenu smerovacích tabuliek sa využívajú správy typu multicast.

Obe verzie však neobsahujú žiadnu podporu pre hierarchický návrh sietí. Rozľahlé siete teda nie je možné členiť na viacero menších častí. Všetky smerovače tak obsahujú v smerovacej tabuľke záznamy o cestách do všetkých podsietí (ide o tzv. flat topológiu), čo môže spôsobovať príliš dlhé vyhľadanie najvhodnejšej cesty. Prípadné sumarizácie musí nastavovať administrátor ručne.

3.1.2 Časovače RIP protokolu

Činnosť protokolu RIP ovplyvňujú nastavenia viacerých časovačov. Smerovače Cisco majú implementované štyri rôzne časovače:

- **Update Timer** - Špecifikuje frekvenciu zasielania periodických updatov. Východzie nastavenie je 30 sekúnd. Rovnako aj každá cesta v smerovacej tabuľke má priradený vlastný update timer, ktorý sa resetuje vždy po obdržaní novej informácie o danej ceste.

- **Invalid Timer** - Keď počas doby jedného časovača Invalid nedostane smerovač novú informáciu o určitej ceste, označí ju ako nedostupnú (nastaví metriku na 16), ale naďalej zostane v smerovacej tabuľke. Prednastavená hodnota je 180 sekúnd.

- **Flush Timer** - Každá cesta, ktorá je označená ako neplatná (Invalid - po vypršaní Invalid časovača), zostáva ešte určitú dobu v smerovacej tabuľke, aby informácia o nedostupnosti mohla byť rozšírená ďalej. Predvolená hodnota je 240 sekúnd, teda cesta po označení za neplatnú je zo smerovacej tabuľky odstránená po 60 sekundách.

- **Holddown Timer** - Tento časovač je implementovaný iba na Cisco smerovačoch. Slúži k stabilizácii celého protokolu v sieťach, kde môže dojsť ku krátkodobým výpadkom. Definuje dobu od zmeny určitej cesty, počas ktorej smerovač ignoruje ostatné zmeny týkajúce sa danej cesty. Štandardne je to 180 sekúnd.

3.1.3 Konfigurácia protokolu RIP na smerovačoch Cisco

V nasledujúcej tabuľke sú uvedené základné príkazy potrebné na konfiguráciu protokolu RIP na smerovačoch Cisco.

`Router(config)# router rip`	Aktivuje RIP protokol na smerovači.
`Router(config-router)# network` *address*	Aktivuje protokol RIP na rozhraniach, ktoré spadajú do siete zadanej parametrom *address* a zahrnie túto sieť do updatov.
`Router(config-router)# version 2`	Aktivuje RIPv2 protokol na smerovači.
`Router# show ip protocols`	Zobrazí informácie o aktívnych smerovacích protokoloch.
`Router# debug ip rip`	Zapne podrobný výpis informácii o prebiehajúcej činnosti RIP protokolu.

Tabulka 3.1: Základná konfigurácia RIP protokolu

3.2 Protokol OSPF

Protokol Open Shortest Path First (OSPF) patrí medzi tzv. Link-State dynamické smerovacie protokoly. Jeho prvá implementácia bola publikovaná v roku 1989 a je popísaná v dokumente RFC1131. Postupom času bol modernizovaný a v súčasnosti sa používa už tretia verzia, ktorá je z roku 1999 (RFC2740) [3]. Je určený na smerovanie vo vnútri autonómnych systémov a momentálne je to doporučovaný protokol pre použitie v IP sieťach.

3.2.1 Technické informácie

Princíp činnosti Link-State je založený na tom, že každý smerovač má kompletnú predstavu o celej sieti. Na základe obdržaných smerovacích informácii si zostaví mapu celej topológie. Ide vlastne o graf, kde vrcholy predstavujú jednotlivé uzly a hrany sú ich prepojenia. Pomocou vhodného grafového algoritmu smerovač vyberie najlepšie cesty do všetkých dosiahnuteľných cieľov, z ktorých následne zostaví smerovaciu tabuľku. Protokol OSPF používa Dijkstrov algoritmus. Na rozdiel od Distance-Vector protokolov nedochádza medzi susedmi k výmene kompletných smerovacích tabuliek.

Medzi smerovačmi musí pred samotným nadviazaním susedstva prebehnúť počiatočná fáza, pri ktorej vzájomne porovnajú a potvrdia parametre komunikácie. Toto sa deje pomocou tzv. správ Hello, ktoré smerovač pravidelne rozposiela na rozhraniach, na ktorých je protokol OSPF spustený. Správy sú posielané na špecifickú multicastovú adresu 224.0.0.5 (AllOSPFRouters). Správy Hello slúžia okrem ustanovenia susedstva aj na jeho udržovanie. Ich pravidelnou výmenou si smerovače overujú dostupnosť jednotlivých susedov.

Po nadviazaní susedstva nasleduje výmena smerovacích informácii medzi susedmi. Na to sa využívajú tzv. správy LSA (Link-state advertisements), ktorých je viacero typov, ale na výmenu informácii o pripojených sieťach sa používajú iba dva typy (router LSA

a network LSA). Každý smerovač uchováva všetky prijaté LSA správy v tzv. Link-state database (LSB), z ktorej pomocou Dijstrovho algoritmu zostaví stromSFP (Shortest Path First) a následne smerovaciu tabuľku. V prípade viacerých možných ciest do cieľa sa o najlepšej rozhoduje na základe celkovej ceny cesty. Každá linka je na základe maximálnej možnej šírky pásma ohodnotená metrikou (napríklad FastEthernet má predvolenú metriku 1). Celková cena je potom súčet metrík jednotlivých liniek.

Pri sieťach, v ktorých na jednom sieťovom segmente je viacero smerovačov (napr. siete Ethernet alebo Frame Relay), nie je výhodný koncept, že každý smerovač nadväzuje susedstvo so všetkými ostatnými. Výmena smerovacích informácii a udržovanie veľké množstva susedstiev by zbytočne zaťažovalo celú sieť a znižovalo priepustnosť. Preto je vždy zvolený tzv. Designated Router (DR) a Backup Designated Router (BDR). Voľba prebieha na základe priority rozhrania alebo tzv. RouterID. Tieto parametre obsahujú priamo Hello správy a voľba prebieha pri počiatočnej konvergencii.

Podobne ako protokol RIPv2 je OSPF beztriedny a podporuje autentizáciu. Jedna z jeho výhod je, že podporuje hierarchický návrh sietí. V rámci jedného autonómneho systému je možné použiť viacej rôznych oblastí, ktoré môžu byť rozdielnych typov. Je však nutné, aby sa všetky oblasti pripájali na na oblasť 0 (area 0) autonómneho systému, ktorá býva zvyčajne vyhradená pre nosnú (backbone) časť siete. V praxi to potom vyzerá tak, že najvýkonnejšie smerovače sú súčasťou oblasti 0 a vždy jedným rozhraním sú členom aj ďalšej oblasti. Výhoda takéhoto návrhu je v tom, že smerovače, ktoré sú členmi iba jednej oblasti, udržujú v pamäti len jednu topologickú mapu. Výnimkou sú smerovače, ktoré sú na rozhraní dvoch (prípadne viacerých) oblastí. Tie musia udržovať oddelene jednu mapu pre každú oblasť. Podpora viacerých typov oblastí uľahčuje administrátorom sumarizáciu ciest v rámci autonómneho systému. Podľa typu oblasti smerovač, ktorý je na hranici s inou oblasťou, šíri do vnútra danej oblasti len zosumarizované, prípadne predvolené (default) cesty.

3.2.2 Konfigurácia OSPF protokolu na smerovačoch Cisco

V nasledujúcej tabuľke sú uvedené základné príkazy potrebné na konfiguráciu protokolu OSPF na smerovačoch Cisco.

`Router(config)# router ospf` *proces-id*	Zapne protokol OSPF, parameter *proces-id* slúži na identifikáciu procesu v prípade viacnásobných behov.
`Router(config-router)# network` *address wildcard-mask* `area` *area-id*	Pomocou adresy a wildcard masky definuje rozhrania, na ktorých bude protokol OSPF zapnutý. Parameter *area-id* identifikuje oblasť, do ktorej budú dané rozhrania patriť.
`Router# show ip ospf` *proces-id*	Zobrazí základné informácie o protokole OSPF.

Tabulka 3.2: Základná konfigurácia OSPF protokolu a jej overenie

Uvedené sú len tie najzákladnejšie príkazy potrebné na spustenie protokolu OSPF. Okrem nich existuje ešte veľké množstvo ďalších príkazov, ktoré umožňujú prispôsobiť správanie celého protokolu konkrétnym potrebám. Pri konfigurovaní je však potrebná detailná znalosť celého protokolu a aj nižších sieťových vrstiev. Popis jednotlivých príkazov, ktoré sa pri podrobnej konfigurácii na Cisco smerovačoch používajú, vrátane príkladov, je možné nájsť v dokumentácii ku konkrétnej verzii IOS operačného systému [12].

3.3 Protokol EIGRP

Protokol EIGRP (Enhanced Interior Gateway Routing Protocol) na rozdiel od protokol RIP a OSPF nepatrí medzi otvorené protokoly, ktorých špecifikácie sú voľne prístupné, ale je to uzatvorený protokol vyvinutý firmou Cisco. Prvý Cisco proprietárny protokol však bol IGRP (Interior Gateway Routing Protocol) a EIGRP odstraňuje jeho hlavné nedostatky.

3.3.1 Technické informácie

Z hľadiska použitia patrí EIGRP medzi vnútorné (interior) protokoly, používané na smerovanie vo vnútri autonómnych systémov. Na rozdiel od svojho predchodcu, protokolu IGRP, patrí medzi beztriedne a podporuje aj autentizáciu. Patrí medzi Distance-vector protokoly, hoci kombinuje princípy Distance-vector aj Link-state protokolov.

Pre správnu činnosť protokolu sú potrebné tri tabuľky:

- **Neighbor Table** - Tabuľka susedstva obsahuje informácie o susedných smerovačoch, pri ktorých sa uchováva ich adresa a rozhranie lokálneho smerovača, cez ktoré je sused dosiahnuteľný.
- **Topology Table** - Topologická tabuľka uchováva informácie obdržané od susedných smerovačov, ktoré sú potom použité na výpočet najkratšej cesty. Pri náhlej zmene topológie tak smerovač má všetky potrebné informácie k dispozícii a môže ihneď cesty opäť prepočítať.
- **Routing Table** - Smerovacia tabuľka obsahuje len najlepšie cesty do cieľových sietí. Smerovač na základe cieľovej adresy paketu vyhľadá najdlhšiu zhodu so záznamom v smerovacej tabuľke a určí adresu ďalšieho skoku. Okrem EIGRP do nej môžu zapisovať aj iné smerovacie protokoly.

Pred samotnou výmenou smerovacích informácii, prebieha najskôr fáza objavovania susedov a následné nadviazanie spojenia. Smerovač so spusteným protokolom EIGRP vysiela pravidelne na multicastovú adresu 224.0.0.10 správy Hello, ktoré obsahujú základné informácie potrebné pre ustanovenie spojenia. Susedný smerovač, ktorý ma tiež aktivovaný protokol EIGRP, po prijatí správy skontroluje informácie potrebné pre nadviazanie susedstva (napr. číslo autonómneho systému, hodnoty konštánt potrebných na výpočet metriky). V prípade zhody sa spojenie nadviaže. Spojenie je podobne ako pri protokole OSPF udržované periodickým posielaním správ Hello.

Po nadviazaní spojenia dochádza ku kompletnej výmene smerovacích informácii, ktorá je potrebná iba v počiatočnej fáze činnosti protokolu. V neskorších fázach činnosti, keď nastane zmena v topológii, o ktorej smerovač chce informovať svojich susedov, dochádza už len k čiastočným výmenám. Podľa potreby sú použité buď správy typu multicast, alebo správy typu unicast. Všetky obdržané dáta sú uložené do tabuľky topológie.

Medzi susedmi dochádza k výmene správ, ktoré obsahujú informácie o šírke pásma (bandwidth), oneskorení (delay), záťaži (load), spoľahlivosti (reliability) a MTU linky na ceste k určitej destinácii. Na ich základe sa vypočíta metrika cesty. K výpočtu sú okrem informácií o linke potrebné aj hodnoty piatich konštánt, ktoré musia byť nastavené na rovnaké hodnoty medzi susednými smerovačmi (tzv K-values). Štandardne sa však na výpočet metriky použije iba šírka pásma a oneskorenie, lebo ostatné informácie vhľadom na predvolené nastavenia konštánt nemajú na výsledok vplyv.

Výber optimálnej cesty bez cyklov v topológií má na starosti tzv. DUAL (Diffusing Update Algorithm). Je to konečný automat, ktorý na základe porovnávania metrík ciest, od jednotlivých susedov k tej istej cieľovej sieti vyberie tú najlepšiu. Pri rozhodovaní berie do úvahy dve hodnoty metrík pre každú cieľovú sieť: Advertised Distance (AD) a Feasible Distance (FD). Advertised Distance je metrika zaslaná susedným smerovačom, Feasible Distance je celková metrika, ktorá v sebe obsahuje danú AD spolu s pripočítanou hodnotou metriky cesty k susedovi. Cesta s najnižšou FD k cieľu sa vyberie a zapíše do smerovacej tabuľky. Susedný smerovač, cez ktorý táto cesta vedie, sa pre danú sieť označí ako následník (Successor). V prípade existencie viacerých ciest s rovnakou FD sú do smerovacej tabuľky inštalované všetky cesty a dátový tok je rovnomerne rozdelovaný.

Okrem primárnej cesty môže protokol EIGRP za určitých podmienok vybrať aj cestu záložnú, pričom je stále garantovaná cesta bez cyklov. Ak susedný smerovač zasiela AD k určitej sieti, ktorá je nižšia ako FD aktuálne zvolenej najlepšej cesty (ale celková FD je vyššia), je tento smerovač označený ako **možný následník (Feasible Successor)**. V prípade výpadku primárnej cesty sa potom záložná cesta môže použiť okamžite, čo výrazne znižuje dobu výpadku a urýchľuje konvergenciu.

Protokol EIGRP podporuje viacero rôznych protokolov sieťovej vrstvy. Pre každý jeden protokol existuje modul, ktorý má na starosti zasielanie správ EIGRP v správnom formáte pre daný protokol. V praxi to znamená, že EIGRP je schopný pracovať okrem IP sietí aj so sieťami IPX, AppleTalk a inými. Rovnako aj podpora IPv6 je implementovaná pomocou špeciálneho modulu.

Na zredukovanie veľkosti smerovcích tabuliek dochádza k automatickej sumarizácii. Táto vlastnosť však môže pri použití beztriednych adresných rozsahov spôsobovať problémy, preto je možné automatickú sumarizáciu vypnúť.

3.3.2 Konfigurácia EIGRP protokolu na smerovačoch Cisco

V nasledujúcej tabuľke sú uvedené základné príkazy potrebné na konfiguráciu protokolu EIGRP na smerovačoch Cisco. Podrobný popis všetkých príkazov ([9]) a ukážky konfigurácie ([7]) sú prístupné na stránkach firmy Cisco.

`Router(config)# router eigrp` *as-number*	Aktivuje protokol EIGRP, parameter *as-number* špecifikuje číslo autonómneho systému, ktoré musí byť rovnaké pre všetky smerovače.
`Router(config-router)# network` *address [wildcard-mask]*	Aktivuje zasielanie a príjem správ EIGRP na rozhraniach, ktoré spadajú do zadanej adresy siete. Nepovinný parameter *wildcard-mask* slúži na detailnejšiu špecifikáciu sieťovej adresy. Ak nieje zadaný, použije sa triedna adresa.
`Router(config-router)# no auto-summary`	Vypne automatickú sumarizáciu adries.
`Router# show ip eigrp neighbors`	Zobrazí informácie o objavených susedoch.
`Router# show ip eigrp topology`	Zobrazí topologickú tabuľku, ktorú smerovač používa na určenie najlepšej cesty.

Tabulka 3.3: Základná konfigurácia EIGRP protokolu a jej overenie

EIGRP predstavuje moderný smerovací protokol, ktorého hlavné prednosti sú rýchla doba konvergencie a nízke zaťaženie prenosovej kapacity siete. Jeho nevýhoda však spočíva

v uzavretosti protokolu, čo vylučuje jeho použitie v heterogénnom prostredí.

3.4 ACL

Access Control Lists (ACL) predstavujú spôsob na kontrolu toku dát v sieti, čím vlastne implementujú základné bezpečnostné funkcie. Sú to textové zoznamy pravidiel, ktoré povoľujú alebo zakazujú určitý typ dátového toku, na základe vopred zadaných kritérií. Viažu sa na jednotlivé rozhrania pričom je potrebné špecifikovať aj smer filtrácie na vstupný (inbound) alebo výstupný (outbound). Pri filtrácií na vstupe sa pakety porovnávajú s pravidlami v ACL ešte pred ich spracovaním smerovacím protokolom. Naopak pri výstupnej filtrácií môže dojsť ku kontrole s pravidlami ACL až po spracovaní smerovacím protokolom a určení výstupného rozhrania.

Každý zoznam je identifikovaný unikátnym číslom, v prípade pomenovaných zoznamov menom. Následne obsahuje pravidlá, s ktorými sú dáta porovnávané. Každé pravidlo ešte špecifikuje, čo sa má s paketmi vykonať, ak budú vyhovovať danému pravidlu. Možné akcie sú povolenie (permit) a zakázanie (deny). Okrem toho na konci každého ACL je implicitné pravidlo, ktoré zakazuje všetky pakety (tzv. pravidlo deny any). Ak teda v ACL nieje aspoň jedno pravidlo typu permit, ACL odfiltruje a zakáže celý dátový tok.

Spracovávanie pravidiel je sekvenčné a začína prvým pravidlom v poradí. Skontrolované sú zadané kritéria (IP adresy, čísla portov atď.) a v prípade, že dáta kritériam vyhovujú, je vykonaná akcia špecifikovaná daným pravidlom (povolenie alebo zákaz) a prehľadávanie ACL sa ukončí. Znamená to, že vždy sa kontrola dostane iba po prvú zhodu s pravidlom. Ak sa počas spracovávania paketu nevyskytne žiadna zhoda, uplatňuje sa implictiné pravidlo **deny any**.

3.4.1 Štandardné ACL

Štandardné ACL predstavujú základnú možnosť kontroly a filtrovania dátového toku. Na Cisco smerovačoch sú identifikované číslami z rozsahu 1 - 99, 1300 až 1999. Umožňujú porovnávať dáta iba na základe zdrojovej adresy, preto sa odporúča aplikovať ich čo najbližšie k cieľu, aby sa zamedzilo filtrovaniu dát určených pre iné cieľové siete.

Všeobecná syntax štandardných ACL, konfigurácia prebieha v globálnom konfiguračnom móde:

`access-list` *acl-number* `{permit|deny}` *source source-wildcard*

Za kľúčovým slovom `access-list` nasleduje číslo, ktorým bude dané ACL identifikované, potom špecifikácia akcie, ktorá sa má vykonať a nakoniec je potrebné zadať zdrojovú adresu a masku vo wildcard formáte. Aby filtrovanie začalo fungovať, musí sa ACL aktivovať na rozhraní na vstupnom, prípadne výstupnom smere.

Príklad konfigurácie:

```
Router(config)# access-list 10 permit 192.168.10.0 0.0.0.255
Router(config)# interface FastEthernet0/1
Router(config-if)# ip access-group 10 in
```

V uvedenom príklade je najskôr vytvorený kontrolný zoznam, identifikovaný číslom 10, ktorý pozostáva iba z jedného pravidla typu permit (okrem implicitného zákazu). Nasleduje aktivácia ACL na rozhraní FastEthernet0/1 na vstupnom smere. Znamená to, že na vstupe bude povolený iba dátový tok z podsiete 192.168.10.0/24.

3.4.2 Rozšírené ACL

Rozšírene ACL prestavujú už pokročilý nástroj na filtrovanie dátového toku. Na Cisco smerovačoch sú identifikované číslami z rozsahov 100 - 199, 2000 - 2699. Umožňujú filtrovanie na základe zdrojovej a cieľovej IP adresy, zdrojových a cieľových čísel TCP a UDP portov a podľa protokolu. Protokol je možné zadať kľúčovým slovom alebo jeho číslom.

Základná syntax príkazu, ktorým sa konfigurujú rozšírené ACL:

`access-list` *acl-number* `{permit|deny}` *protocol source source-wildcard destination destination-wildcard*

Oproti konfigurácií štandardného ACL pribudla možnosť špecifikácia protokolu a cieľovej adresy. Uvedená syntax ešte neumožnuje špecifikovať aj TCP alebo UDP port. To je možné až vtedy, keď sa ako protokol uvedie TCP alebo UDP. Potom sa za parameter *source-wildcard* uvedie ešte dvojica parametrov *operator port-number*.

Znalosť viacerých kritérií umožňuje umiestniť rozšírené ACL čo najbližšie ku zdroju dát a ich prípadné odfiltrovanie hneď na začiatku ich cesty k cieľu. Dochádza tak v šetreniu šírky pásma, lebo dátový tok nemusí prejsť celú cestu k cieľu a byť odfiltrovaný až v cieli.

Príklad konfigurácie:

```
Router(config)# access-list 100 permit icmp 192.168.10.0 0.0.0.255
172.16.0.0 0.0.255.255
Router(config)# access-list 110 permit icmp 172.16.0.0 0.0.255.255
192.168.10.0 0.0.0.255
Router(config)# interface FastEthernet0/1
Router(config-if)# ip access-group 100 in
Router(config-if)# ip access-group 110 out
```

V uvedenom príklade sú vytvorené dva rozšírené ACL, pričom povolujú iba icmp komunikáciu medzi sieťami 192.168.10.0/24 a 172.16.0.0/16. Následne sú aktivované na rozhraní FastEthernet0/1. Daný príklad by spôsobil, že medzi danými sieťami by cez rozhranie FastEthernet0/1 prešla iba icmp komunikácia. Všetko ostatné by bolo zakázane implicitným pravidlom `deny ip any any`.

3.4.3 Pomenované ACL

Pomenované ACL netvoria samostatný typ, ide skôr o iný spôsob konfigurácie. Pomenované ACL sú identifikované menom (nie unikátnym číslom), ale stále sú to buď štandardné alebo rozšírené ACL. Majú však výhody v tom, že sú ľahko editovateľné. Pri číslovaných ACL je každé nové pravidlo pridané vždy na koniec celého zoznamu. Ak teda administrátor potrebuje pridať pravidlo na určité miesto, musí zoznam nanovo celý vytvoriť. Pri pomenovaných zoznamoch je možné priradiť každému pravidlo sekvenčné číslo, podľa ktorého sú potom

pravidlá zoradené. Vhodne zvoleným sekvenčným číslom sa dá vložiť pravidlo na konkrétne miesto v zozname.

3.4.4 Ostatné typy ACL

Exitujú ešte ďalšie typy ACL, ktoré sa však používajú menej často.

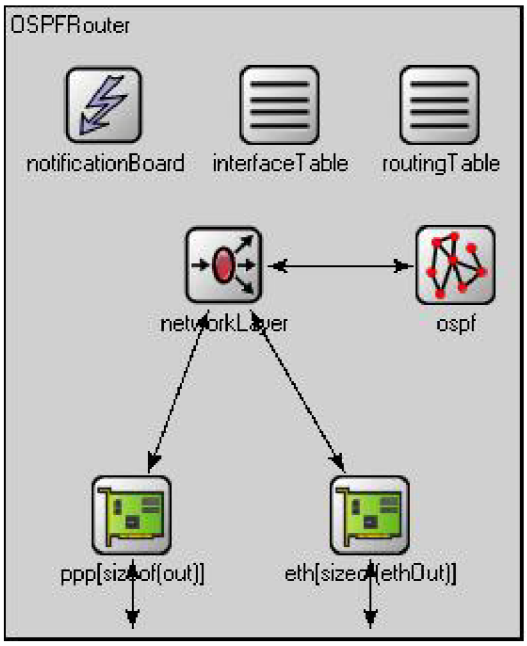
- **Dynamické ACL** - Predtým, ako je umožnené užívateľovi prejsť cez určitý smerovač, je potrebné, aby sa prihlásil na daný smerovač cez Telnet a autentizoval. Po úspešnej autentizácií je pridané na určitú dobu dynamické pravidlo, ktoré umožní dátam prechod cez smerovač.
- **Reflexívne ACL** - Povolujú odchádzajúce dáta a filtrujú prichádzajúce. Ak prichádzajúce dáta sú odpoveďou na reláciu, ktorá vznikla vo vnútri siete, sú povolené, inak sú zakázané.
- **Časové ACL** - Časové ACL sú v podstate rozšírené ACL, ktoré sú aktívne iba v určený čas. Pre správnu činnosť je potrebné nastaviť na smerovači konkrétny časový interval a ten následne priradiť k určitému rozšírenému ACL.

Podrobné informácie o jednotlivých typoch ACL, spolu s ukážkami konfigurácie, je možné nájsť v oficiálnej dokumentácii [8].

Kapitola 4

Pôvodný stav

Táto práca sa zaoberá implementáciou modulu, ktorý by sa mal funkcionalitou čo najviac podobať smerovačom Cisco. V aktuálnej verzii INET Frameworku má dynamické smerovanie v IP sieťach na starosti modul OSPFRouter. Štruktúra modulu je znázornená na obrázku 4.1. Okrem dynamického smerovania sú podporované aj statické cesty. V tejto kapitole sú opísané jednotlivé moduly, z ktorých sa OSPFRouter skladá. Keďže náš modul bude z modulu OSPFRouter vychádzať, je potrebné oboznámiť sa s jeho časťami a funkcionalitou. Na základe získaných znalostí sme mohli následne odhaliť nedostatky a implementovať potrebné zmeny.



Obrázek 4.1: Štruktúra modulu OSPF Router

4.1 NotificationBoard

Slúži na výmenu informácii medzi modulmi, ktoré sa môžu navzájom informovať o určitých zmenách, ako sú napríklad zmeny v smerovacej tabuľke, zmeny stavu rozhraní, zmeny

konfigurácie atď. Modul NotificationBoard teda slúži ako sprostredkovateľ informácie medzi modulom, kde nastala zmena a modulom, ktorý sa má o danej zmene dozvedieť a prípadne na ňu reagovať. Komunikácia medzi modulmi a NotificationBoard prebieha iba prostredníctvom volania implementovaných metód. Na komunikáciu prostredníctvom zasielania správ by boli potrebné rozhrania, ktoré ale modul NotificationBoard nemá. Musel by byť totiž spojený s každým ďalším modulom osobitne, čo by pri väčšom počte modulov mohlo spôsobovať problémy.

Jednotlivé udalosti sú rozdelené do viacerých kategorii a každý klientsky modul sa pomocou metódy `subscribe()` z modulu NotificationBoard zaregistruje do kategórie, z ktorej chce oznámenia prijímať.

Na rozšírenie informácie sa používa metóda `fireChangeNotification()`. Po jej zavolaní všetky zaregistrované moduly okamžite obdržia oznámenie o danej udalosti a môžu ihneď reagovať.

4.2 InterfaceTable

Obsahuje zoznam všetkých rozhraní smerovača a každý smerovač obsahuje práve jeden modul InterfaceTable. Sú v nej uchovávané iba základné informácie o rozhraniach ako IP adresa, MAC adresa, MTU a členstvo v skupinách multicast. Štandardne sa registruje rozhranie Loopback s IP adresou 127.0.0.1.

Rozhrania sú registrované dynamicky príslušnými modulmi ihneď po spustení simulácie. Jednotlivé rozhrania sú reprezentované objektami typu `InterfaceEntry` a do modulu InterfaceTable sú pridávané volaním metódy `addInterface()`. Okrem pridania a odobrania rozhraní, umožňuje InterfaceTable aj vyhľádvať rozhrania na základe rôznych parametrov (meno, Id, poradia atď.).

4.3 NetworkLayer

Zložený modul NetworkLayer implementuje v smerovači sieťovú vrstvu. Skladá sa z viacerých podmodulov:

- **ip** - Modul ip implementuje IP protokol. Okrem komunikácie s protokolmi vyšších vrstiev má na starosti aj samotné smerovanie paketov. Keď potrebuje smerovať určitý datagram, modul IP využíva metódy modulu RoutingTable. Volaním metódy `findBestMatchingRoute(destAddress)` získa z RoutingTable informáciu o výstupnom rozhraní a adrese ďalšieho uzlu (next hop address). Komunikácia prebieha iba volaním príslušných metód, nie sú posielané žiadne správy.
- **icmp** - Spracováva icmp správy.
- **igmp** - V aktuálnej implementácii nie je tento modul využitý, je iba pripravený pre budúce rozšírenie.
- **arp** - Implementuje Address Resolution Protocol (ARP).
- **errorHandling** - Spracováva oznámenia o chybách, ktoré prichádzajú z iných modulov.

4.4 RoutingTable

Jednoduchý modul RoutingTable uchováva smerovaciu tabuľku. Má tri parametre, ktoré upresňujú jeho funkcionalitu:

- **routerId** - Reťazec, predvolená hodnota pre smerovače je „auto“, čo znamená, že routerId bude určené na základe najvyššej IP adresy rozhrania.
- **IPForward** - Prepínač, ktorý zapne/vypne smerovanie. Východzie nastavenie je „true“.
- **routingFile** - Meno súboru, z ktorého sa má smerovacia tabuľka načítať. Ak nie je meno zadané, odvodí sa podľa mena smerovača.

Modul nemá žiadne rozhrania, preto nekomunikuje s ostatnými modulmi zasielaním správ. Funkcionalita je prístupná priamym volaním implementovaných metód. Popis najdôležitejších metód:

Smerovacia tabuľka je spolu s nastaveniami jednotlivých rozhraní načítavaná z externého súboru. V adresári simulácie sú to súbory `*.irt` alebo `*.mrt`. Tieto externé súbory obsahujú dve sekcie. Jedna obsahuje nastavenia rozhraní, druhá statické cesty, ktoré sa nainštalujú do smerovacej tabuľky.

Príklad súboru, ktorý obsahuje nastavenia rozhrania a jednej statickej cesty:

```
ifconfig:

# ethernet card 0 of router R1 - connected to R2
name: eth0
  inet_addr: 192.168.3.1
  Mask: 255.255.255.0
  Groups: 224.0.0.5:224.0.0.6
  MTU: 1500
  Metric: 1
  BROADCAST MULTICAST

ifconfigend.

route:

224.0.0.0 * 240.0.0.0   H 0 eth0

routeend.
```

V časti medzi príkazmi `ifconfig:` a `ifconfigend.` sú postupne uvedené nastavenia rozhraní. Formát je odvodený od výpisu na systémoch Unix. Statické cesty sú zapísané v sekcii medzi príkazmi `route:` a `routeend.`. Jednotlivé rozhrania sa pri spustení simulácie automaticky zaregistrujú s prednastavenými hodnotami, ktoré je možné prepísať práve nastaveniami zo smerovacieho súboru.

Statické cesty sú zapisované vo formáte:
`Destination Gateway Netmask Flags Metrich Interface`

Zápis pomocou `*` ako výstupnej brány znamená `0.0.0.0`. Cieľovú sieť je možné zadať príkazom `default`, čím sa vytvorí cesta, na ktorú sa budú posielať všetky dáta, pre ktoré sa iná cesta nenájde. Parameter `Flag` definuje typ cesty. Prípustné sú dve hodnoty: `H` (predstavuje priamo pripojenú cestu) a `G` (vzdialená cesta, dosiahnuteľná cez ďalší smerovač).

Modul RoutingTable má implementované aj metódy na prácu so smerovacou tabuľkou, ktoré sú potom využívané hlavne pri smerovaní dát:

- **IPRoute* findBestMatchingRoute(const IPAddress &dest)** - Vstupom je cieľová IP adresa a návratová hodnota je záznam v smerovacej tabuľke, ktorý sa s danou adresou najviac zhoduje. Optimálna cesta sa vyberie na základe dĺžky zhody. Ak existuje aj východzia cesta (default route), tá má nulovú dĺžku prefixu, preto bude vybraná až ako posledná.
- **InterfaceEntry* getInterfaceForDestAddr (const IPAddress &dest)** - Metóda, ktorá vráti informácie o rozhraní, cez ktoré by sa mali dáta s danou cieľovou adresou smerovať. Ak cieľová adresa nie je v smerovacej tabuľke, návratová hodnota je NULL.

4.5 EthernetInterface a PPPInterface

Sú to zložené moduly, ktoré implementujú funkcionalitu druhej vrstvy sieťového modelu ISO/OSI.

Modul EthernetInterface implementuje rozhrania typu Ethernet, pričom podporuje rýchlosti 10Mbps, 100Mbps a 1Gbps. Skladá sa z troch podmodulov:

- **OutputQueue** - Implementácia jednoduchej fronty, do ktorej sa radia pakety pred odoslaním. Volaním metódy requestPacket() odošle jeden paket na spracovanie príslušnému L2 modulu.
- **EtherEncap** - Modul, ktorý má na starosti zapuzdrenie, ak paket prichádza z vyššej vrstvy, alebo odstránenie zapuzdrenia, ak prichádza rámec z MAC vrstvy.
- **EtherMac** - Implementuje MAC vrstvu Ethernet protokolu. Má na starosti príjem a odosielanie rámcov.

Modul PPPInterface sa používa v OMNeT++ pre pomalé linky a sa skladá iba z dvoch podmodulov:

- **OutputQueue** - Rovnaká funkcionalita ako pri module EthernetInterface.
- **PPP** - Modul, ktorý implementuje protokol Point-to-Point, pričom podporovaná je iba základná funkcionalita (zapuzdrenie a jednoduchá fronta).

4.6 Zhrnutie vlastností modulu OSPFRouter

V aktuálnej implementácii ponúka modul OSPFRouter základnú funkcionalitu, ktorá je potrebná pre smerovače. Ide hlavne o podporu IP adresácie, prácu z rozhraniami, podpora dynamického smerovania a statických ciest. Chýba však podpora pre viacej dynamických smerovacích protokolov. Smerovacia tabuľka totiž nerozlišuje z akého zdroja bola cesta nainštalovaná a umožňuje inštalovať viacej rovnakých ciest pre rovnakú cieľovú adresu.

Rovnako spôsob, akým sú načítavané nastavenia rozhraní a statické cesty nie je vyhovujúci. V nasledujúcej kapitole sme popísali zmeny, ktoré bolo potrebné implementovať, aby sa uvedené hlavné nedostatky odstránili.

Kapitola 5

Popis implementácie

Cieľom tejto práce je vytvoriť modul, ktorý sa svojou funkcionalitou bude podľa možnosti čo najviac približovať reálnym smerovačom Cisco.

Ako základ pre nový modul sme si zvolili už implementovaný modul OSPFRouter, keďže sú v ňom už implementované niektoré potrebné súčasti ako IP adresácia, smerovacia tabuľka a dynamický smerovací protokol OSPF. Modul OSPFRouter je potrebné rozšíriť o ďalšie smerovacie protokoly a kontrolu dátového toku s použitím Access Control List (ACL). Moduly implementujúce protokol RIP ([4]) a ACL ([6]) boli vyvíjané súbežne s touto prácou. Detailné informácie ohľadom ich vývoja a implementovaných vlastností preto nie sú v tejto kapitole zahrnuté.

Určité vlastnosti modulu OSPFRouter bolo potrebné znovu naimplementovať. Išlo hlavne o nevyhovujúce načítavanie nastavení rozhraní z textových súborov, spôsob zobrazovania informácii o smerovacej tabuľke a vkladanie duplicitných ciest do smerovacej tabuľky.

5.1 Načítavanie konfigurácie z XML súboru

V pôvodnej verzii sa nastavenia rozhraní načítavali z textových súborov. Každé rozhranie malo vlastný súbor, ktorý okrem toho mohol obsahovať aj statické cesty.

Súbežne s touto prácou bol vytvorený aj nástroj na preklad textových konfigurácii z reálnych zariadení do XML súboru [5]. V ňom sú nastavenie rozhraní a informácie o statických cestách uložené v nasledujúcich štruktúrach.

Vetva s nastaveniami rozhrania:

```
<Interfaces>
     <Interface name=,,name‘‘>
          <IPAddress></IPAddress>
          <Mask></Mask>
          <Duplex></Duplex>
          <Speed></Speed>
          <Bandwidth></Bandwidth>
    </Interface>
</Interface>
```

Vetva s informáciami o statických cestách:

```
<Routing>
      <Static>
```

```
        <Route>
          <NetworkAddress></NetworkAddress>
          <NetworkMask></NetworkMask>
          <NextHopAddress></NextHopAddress>
        </Route>
      </Static>
</Routing>
```

Vytvorili sme nový modul `RoutingTableXmlParser`, ktorý využíva už implementované metódy na prácu s XML súbormi a stará sa o samotné načítanie potrebných dát.

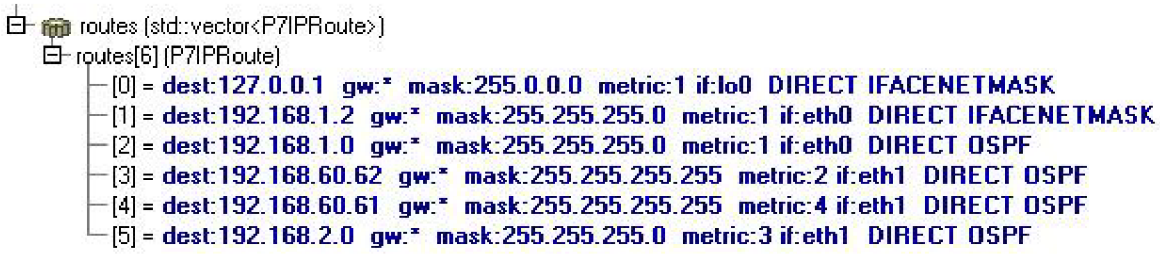
Obsahuje nasledovné metódy:

- **virtual bool readRoutingTableFromXml (const char *filename, const char *RouterId)** - Podľa názvu konfiguračného súboru (parameter `filename`) a identifikácie smerovača (parameter `RouterId`) zistí, či sa v uvedenom konfiguračnom súbore nachádzajú dáta o danom smerovači. Následne vyhľadáva prítomnosť tagov <Interfaces>, <Routing> (spolu s podvetvou <Static>) a volá ďalšie metódy.
- **void readInterfaceFromXml(cXMLElement* Node)** - Obdrží ako parameter odkaz na uzol v XML súbore, od ktorého ma začať vyhľadávať. Pomocou funkcie getChildren získa potomka tagu `Node` iterátorom prehľadá všetky jeho atribúty. Ak je správne dodržaná štruktúra XML súboru, načíta nastavenia rozhrania a uloží ich do InterfaceTable.
- **void readStaticRouteFromXml(cXMLElement* Node)** - Princíp činnosti je rovnaký ako pri predchádzajúcej metóde. Rozdielna je iba predvolená štruktúra, podľa ktorej sú vyhľadávané jednotlivé tagy. Táto metóda ukladá informácie o statickej ceste do štruktúry typu `IPRoute`, ktorú potom volaním metódy addRoute() uloží do smerovacej tabuľky.

Aby bolo celý modul funkčný, bolo ešte potrebné nastaviť v module RoutingTable, aby používal pri svojej činnosti novovytvorený parser. Stačilo však iba prepísať meno pôvodného parsera (RoutingTableParser) a meno nového (readRoutingTableFromXML). Funkčnosť načítavania dát z XML súboru sme overili pri prípadovej štúdii, ktorá je popísaná v nasledujúcej kapitole.

5.2 Zmeny v RoutingTable

V module RoutingTable sme okrem nového parsera rozhraní a statických ciest implementovali aj ďalšie zmeny. Jedno z nich bola úprava formátu, v ako sa cesty vypisujú pri simulácii. Pôvodný formát je zobrazený na nasledujúcom obrázku.



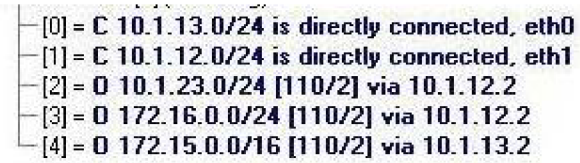
Obrázek 5.1: Pôvodný formát výpisu ciest smerovacej tabuľky

Je zrejmé, že hoci uvedený formát poskytuje takmer rovnaké informácie ako formát výpisu reálneho zariadenia, ale nie je príliš prehľadný. Aj smerovače Cisco používajú dva rozdielne formáty výpisu ciest zo smerovacej tabuľky. Jeden je pre cesty pochádzajúce z priamo pripojených sietí a druhý je pre cesty, ktoré vložili smerovacie protokoly.

Celkové upravenie výstupu do požadovaného formátu má na starosti metóda `void generateShowIPRoute()`. V cykle prejde celú smerovaciu tabuľku a pomocou volania funkcie `getSource()` zisťuje zdroj každej cesty.

Ak je zdroj cesty smerovací protokol (`getSource()` vráti jednu z hodnôt RIP, OSPF, BGP) zavolá ďalšiu metódu `std::string otherIPRouteFormat(const IPRoute* entry)`. V prípade iného zdroju cesty (`getSource()` vráti MANUAL alebo IFACENETMASK) zavolá metódu `std::string directIPRouteFormat(const IPRoute* entry)`. V tomto prípade slúži metóda `getSource()` na zistenie zdroja cesty. Je implementovaná v triede IPRoute a aktuálne môže vrátiť jednu z hodnôt MANUAL, IFACENETMASK, RIP, OSPF alebo BGP. V prípade rozšírenia o ďalší smerovací protokol, stačí zo zoznamu návratových hodnôt pridať požadovanú položku a metóda bude pracovať korektne aj s novým protokolom.

Tieto metódy dostanú na vstupe záznam typu IPRoute a vrátia reťazec v požadovanom formáte. Všetky reťazce sa uložia do vektoru reťazcov a ten je následne vypísaný. Nedochádza tak priamo k zmene formátu, v akom sú záznamy uložené, ale iba ku zmene formátu výstupu. Výsledný formát je zobrazený na nasledujúcom obrázku.



Obrázek 5.2: Upravený formát výpisu ciest smerovacej tabuľky

Povôdný modul pred pridaním cesty nekontroloval, či pridávaná cesta už nie je prítomná od iného zdroja. Inštalovaných teda mohlo byť viac rovnakých ciest. Napríklad cesta k sieti, ktorá bola priamo k smerovaču pripojená, sa mohla vyskytnúť aj ako cesta od smerovacieho protokolu.

Implementovali sme preto metódu `checkRoute()`, ktorá skontroluje prítomnosť zadanej cesty v smerovacej tabuľke. Aby sme v prípade zhody mohli vybrať lepšiu cestu, museli sme implementovať aj statickú mapu, ktorá mapuje návratové hodnoty metódy `getSource()` na hodnoty príslušných administratívnych vzdialeností.

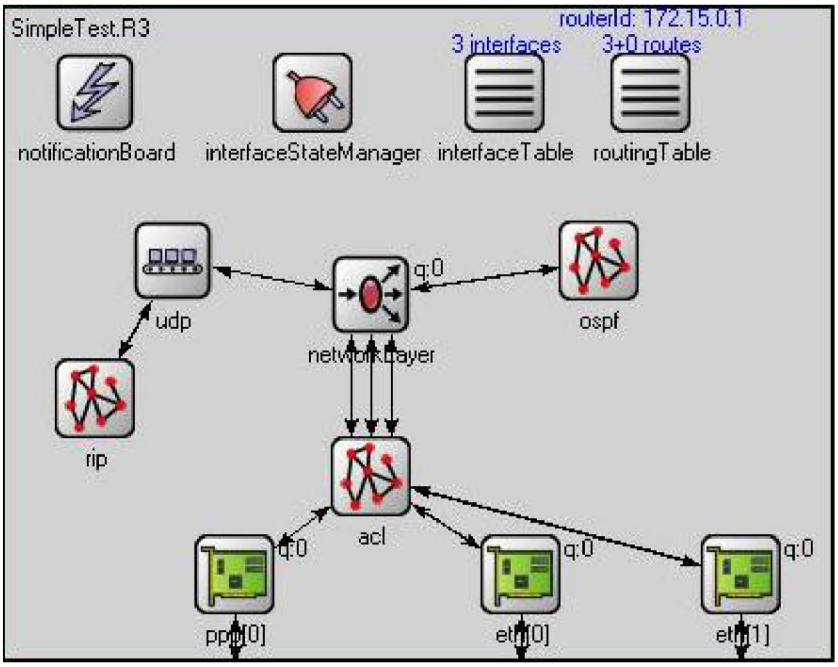
Metóda `checkRoute()` najskôr skontroluje, či sa v smerovacej tabuľke nachádza rovnaká cesta ako cesta zadaná na vstupe. Porovnáva na základe zhody sieťovej adresy a sieťovej masky. Ak rovnakú cestu nájde, musí následne rozhodnút, ktorá z oboch ciest je lepšia. Pomocou metódy `getSource()` zistí zdroje daných ciest, následne využije mapovanie zdroja cesty na hodnotu AD a rozhodne, ktorá z ciest je dôveryhodnejšia. Definícia tejto metódy:
`bool checkRoute(const IPRoute* entry);`
Návratová hodnota je `true` v prípade, že cestu zadanú parametrom na vstupe je možné do smerovacej tabuľky priamo nainštalovať. Hodnotu `false` vráti ak už exituje lepšia cesta, ako cesta zadaná na vstupe.

5.3 Zmeny v InterfaceTable

Modul InterfaceTable pri inicializácii automaticky registroval rozhranie typu Loopback a priradil mu adresu 127.0.0.1. Toto rozhranie však nie je pre činnosť smerovača potrebné. Preto sme inicializáciu modulu upravili a registráciu Loopback rozhrania odstránili.

5.4 Výsledný stav

Nami vytvorený modul obsahuje dynamické smerovacie protokoly RIP a OSPF, podporu filtrovania paketov pomocou ACL a podporu statického smerovania. Okrem toho boli upravené niektoré moduly, aby sa funkcionalita výsledného modulu smerovača čo najviac približovala reálnym zariadeniam. Na nasledujúcom obrázku je grafické znázornenie celého modulu ANSARouter, popis v jazyku NED je uvedený v prílohách (B)



Obrázek 5.3: Grafické znázornenie modulu ANSARouter

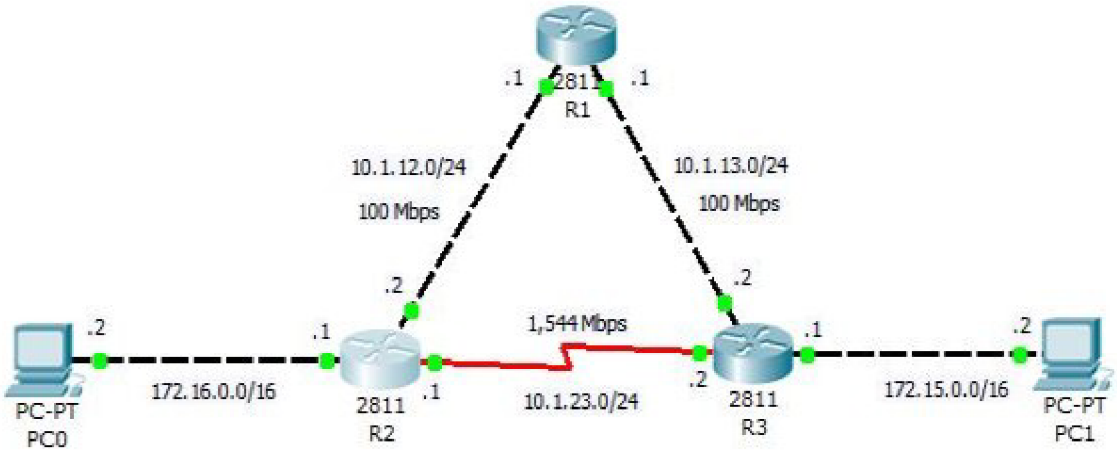
Kapitola 6

Prípadová štúdia

V tejto kapitole je uvedená základná prípadová štúdia, ktorá slúži na porovnanie funkcionality implementovaného modulu AnsaRouter s reálnymi zariadeniami. Ako referenčné zariadenia sme použili smerovače Cisco a testovaciu sieť sme vytvorili v školskom laboratóriu.

6.1 Popis simulovanej topológie

Cieľom simulácie bolo overenie súbežnej činnosti protokolov RIP a OSPF na smerovačoch. Testovacia topológia je znázornená na nasledujúcom obrázku:



Obrázek 6.1: Simulovaná topológia

Sledovali sme hlavne smerovacie tabuľky na jednotlivých smerovačoch a ich dynamické zmeny v závislosti od spustených smerovacích protokolov. Na základe smerovacej tabuľky sa následne menili cesty v topológii, ktorými boli dáta smerované.

Topológia bola zvolená s ohľadom na špecifické vlastnosti protokolov RIP a OSPF. Keďže RIP smeruje iba podľa počtu skokov k cieľu, v uvedenej topológii by mal dáta medzi

počítačmi smerovať cez serial linku medzi smerovačmi R2 a R3. Uvedená cesta však nie je optimálna z hľadiska šírky pásma. Protokol OSPF berie do úvahy aj šírku pásma linky, preto by nemal použiť pomalú serial linku. Dáta medzi počítačmi by mal smerovať síce po dlhšej (cez všetky tri smerovače), ale optimálnej trase.

Pri paralelnom behu viacerých smerovacích protokolov musí smerovač rozhodnúť, ktoré cesty nainštaluje do smerovacej tabuľky. Na rozhodovanie používa parameter administratívnu vzdialenosť (AD - Administrative Distance), ktorá vyjadruje mieru dôveryhodnosti a spoľahlivosti protokolu. Určité smerovacie protokoly sú prioritizované podľa spôsobu, akým vyberajú najlepšie cesty.

V nasledujúcej tabuľke sú uvedené predvolené hodnoty najčastejšie používaných protokolov. Nižšia AD znamená dôveryhodnejší protokol a jeho cesty sú uprednostnené a umiestnené do smerovacej tabuľky. Podrobné informácie o AD je možné nájsť v online dokumentácii ([13])

priamo pripojené cesty	0
statické cesty	1
EIGRP sumárne cesty	5
vonkajšie BGP cesty	20
vnútorné EIGRP cesty	90
OSPF	110
RIP	120
vonkajšie EIGRP cesty	170
vnútorné BGP cesty	200

Tabulka 6.1: Hodnoty AD jednotlivých smerovacích protokolov

6.2 Konfigurácia na reálnych zariadeniach

Po fyzickom zostavení topológie sme zariadenia nakonfigurovali cez rozhranie príkazovej riadky. Bolo potrebné správne nastaviť IP adresy rozhraní a spustiť jednotlivé smerovacie protokoly.

Najskôr sme sputili iba protokol RIP a skontrolovali sme celkovú konfiguráciu aj obsah smerovacej tabuľky.

Na overenie celkovej konfigurácie je možné použiť príkaz `show running-config`, ktorý zobrazí aktuálne platné nastavenia. Smerovacia tabuľka sa zobrazí zadaním príkazu `show ip route`.

Nasledujúce výpisy sú zo smerovača R2. Najskôr je zobrazená celková konfigurácia pričom sú zobrazené iba informácie podstatné pre cieľ našej prípadovej štúdie.

```
R2#show running-config
Building configuration...
!
hostname R2
!
interface FastEthernet0/0
 ip address 10.1.12.2 255.255.255.0
 duplex auto
```

```
 speed auto
!
interface FastEthernet0/1
 ip address 172.16.0.1 255.255.255.0
 duplex auto
 speed auto
!
interface Serial0/0/0
 ip address 10.1.23.1 255.255.255.0
!
router rip
 network 10.0.0.0
 network 172.16.0.0
!
end
```

V prvej časti sú zobrazené nastavenia jednotlivých rozhraní a nasledujú informácie o protokole RIP. Z uvedeného výpisu je možné zistiť, že protokol RIP je spustený na všetkých rozhraniach, lebo spadajú do zadaného adresného rozsahu. Nasledujúci výpis zobrazuje obsah smerovacej tabuľky.

```
R2#sh ip route
Gateway of last resort is not set

     172.16.0.0/24 is subnetted, 1 subnets
C       172.16.0.0 is directly connected, FastEthernet0/1
     10.0.0.0/24 is subnetted, 3 subnets
R       10.1.13.0 [120/1] via 10.1.23.2, 00:00:17, Serial0/0/0
                  [120/1] via 10.1.12.1, 00:00:13, FastEthernet0/0
C       10.1.12.0 is directly connected, FastEthernet0/0
R       172.15.0.0/16 [120/1] via 10.1.23.3, 00:00:08, Serial0/0/0
C       10.1.23.0 is directly connected, Serial0/0/0
```

V smerovacej tabuľke je celkovo päť ciest. Priamo pripojené sú tri siete (C - connected), čo zodpovedá počtu aktívnych rozhraní. Cesty do sietí 172.15.0.0/16 a 10.1.13.0/24 nainštaloval protokol RIP (R - RIP). Testovacia topológia sa skladá z piatich rôznych podsietí, teda sieť skonvergovala správne a je v stave plnej konektivity. Konfigurácie a obsahy smerovacích tabuliek ostatných smerovačov by boli podobné, líšili by sa iba v detailoch spôsobených inými IP adresami.

Na overenie cesty dát v sieti sme použili program tracert spustený na jednom počítači, pričom sme sledovali cestu k druhému.

```
C:\Documents and Settings\root>tracert 172.15.0.2

Tracing route to 172.15.0.2 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  172.16.0.1
  2     1 ms    <1 ms    <1 ms  10.1.23.2
  3     1 ms    <1 ms    <1 ms  172.15.0.2
```

```
Trace complete.
```

Dáta teda putovali z PC0, na smerovač R2, potom na smerovač R3 a následne k cieľovému počítaču PC1.

Následne sme sieť nakonfigurovali tak, že bol spustený iba protokol OSPF. V jednotlivých konfigárcia nastala iba tá zmena, že namiesto informácii o protokole RIP boli na rovnakom mieste zobrazené nastavenia OSPF.

```
router ospf 1
 log-adjacency-changes
 network 10.0.0.0 0.255.255.255 area 0
 network 172.0.0.0 0.255.255.255 area 0
```

V smerovacej tabuľke nastala zmena v tom, že cesty o sieťach, ktoré nemá smerovač R2 priamo pripojené umiestnil protokol OSPF (O - OSPF). Inak je celá sieť tiež v stave plnej konektivity.

```
R2#sh ip route
Gateway of last resort is not set

     10.0.0.0/24 is subnetted, 3 subnets
C       10.1.12.0 is directly connected, FastEthernet0/0
O       10.1.13.0 [110/2] via 10.1.12.1, 00:00:39, FastEthernet0/0
C       10.1.23.0 is directly connected, Serial0/0/0
O    172.15.0.0/16 [110/65] via 10.1.23.3, 00:00:08, Serial0/0/0
C    172.16.0.0/16 is directly connected, FastEthernet0/1
```

Pomocou programu tracert sme tiež skontrolovali, ako sa zmenila cesta dát medzi počítačmi v sieti.

```
C:\Documents and Settings\root>tracert 172.15.0.2

Tracing route to 172.15.0.2 over a maximum of 30 hops

  1    <1 ms    <1 ms    <1 ms  172.16.0.1
  2     1 ms    <1 ms    <1 ms  10.1.12.1
  3     1 ms    <1 ms    <1 ms  10.1.13.2
  4     1 ms    <1 ms    <1 ms  172.15.0.2

Trace complete.
```

Z uvedeného výpisu je jasné, že pri použití protokolu OSPF sa cesta dát zmenila a smerovala z počítača PC0 cez všetky tri smerovače v poradí R2, R1, R3 a následne do cieľa. Vybraná cesta je v porovnaní s protokolom RIP síce dlhšia, ale ponúka vyššiu celkovú priepustnosť. Pomalá serial linka teda nebola použitá.

Nakoniec sme spustili na smerovačoch protokoly RIP aj OSPF súčasne. V konfiguráciach boli informácie o oboch protokoloch, ale inak nenastala žiadna iná zmena.

```
router ospf 1
 log-adjacency-changes
 network 10.0.0.0 0.255.255.255 area 0
 network 172.0.0.0 0.255.255.255 area 0
!
router rip
 network 10.0.0.0
 network 172.16.0.0
```

Obsah smerovacej tabuľky však zostal rovnaký, ako v prípade použitia iba samotného protokolu OSPF. Protokol RIP nemohol nainštalovať žiadne cesty, lebo jeho administratívna vzdialenosť je vyššia ako u protokolu OSPF, ktorý dostal pred protokolom RIP prednosť. Testovať cestu dát v sieti by bolo vhľadom k rovnakej smerovacej tabuľke zbytočné.

6.3 Simulácia v programe OMNeT++

Vstupom pre simuláciu v programe OMNeT++ boli kompletné konfigurácie reálnych smerovačov použitých pri testovaní v laboratóriu. Po preložení textových konfigurácii vznikol jeden XML, ktorý obsahoval všetky potrebné nastavenia jednotlivých smerovačov. V nasledujúcej ukážke sú uvedené iba nastavenia z jedného smerovača, lebo konfigurácie ostatných dvoch boli až na použité IP adresy identické.

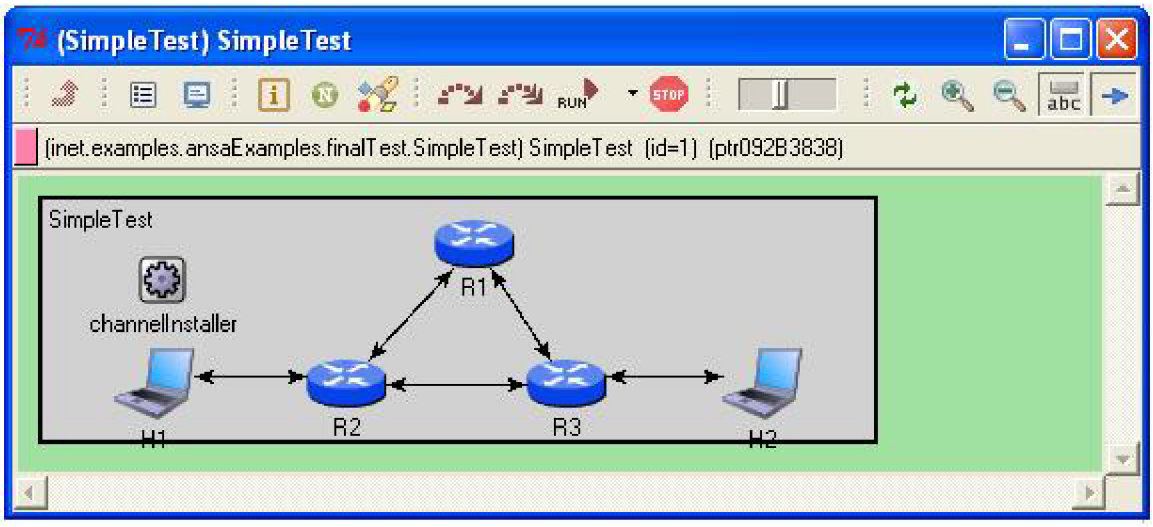
```
<Routers>
    <Router id=,,10.1.13.1‘‘>
        <Hostname>R1</Hostname>
        <Interfaces>
            <Interface name=,,eth1‘‘>
                <IPAddress>10.1.12.1</IPAddress>
                <Mask>255.255.255.0</Mask>
                <Duplex>auto</Duplex>
                <Speed>auto</Speed>
                <OspfNetworkType>point-to-point</OspfNetworkType>
            </Interface>
            <Interface name=,,eth0‘‘>
                <IPAddress>10.1.13.1</IPAddress>
                <Mask>255.255.255.0</Mask>
                <Duplex>auto</Duplex>
                <Speed>auto</Speed>
                <OspfNetworkType>point-to-point</OspfNetworkType>
            </Interface>
        </Interfaces>
        <Routing>
            <Ospf>
                <RFC1583Compatible />
                <Areas>
                    <Area id=,,0.0.0.0‘‘>
                        <Networks>
                            <Network>
```

```
                    <IPAddress>10.0.0.0</IPAddress>
                    <Wildcard>0.255.255.255</Wildcard>
                </Network>
            </Networks>
        </Area>
    </Areas>
</Ospf>
<Rip>
    <Network>10.0.0.0</Network>
</Rip>
</Routing>
</Router>
```

Pre spustenie simulácie bolo potrebné vytvoriť ešte NED súbor s popisom topológie a súbor omnetpp.ini, ktorom sa nastavuje meno XML súboru s nastaveniami smerovačov. Nastavenia klientskych počítačov museli byť uvedené v samostatných textových súboroch H1.irt a H2.irt, lebo analýzou konfigurácii smerovačov nie je možné ich nastavenia zistiť.

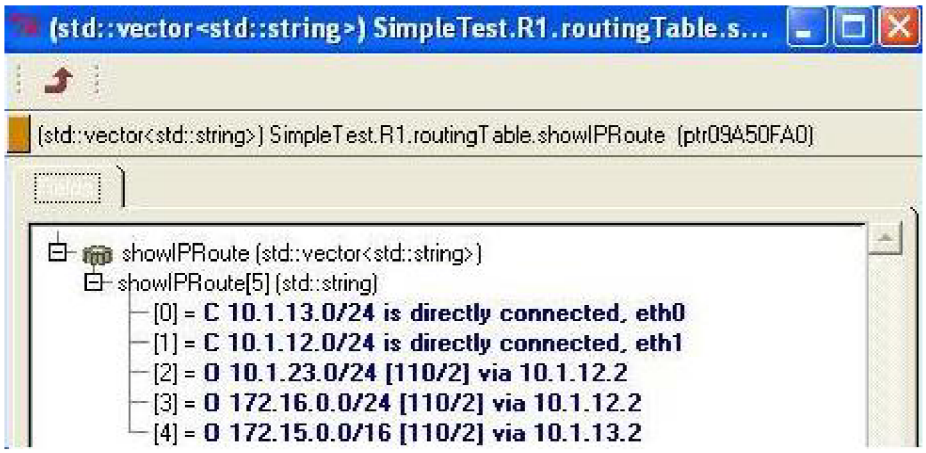
OMNeT++ pri spustení nekontroluje správnosť prepojenia jednotlivých modulov. Omylom je tak možné vytvoriť prepojenie sériového a Ethernet rozhrania. Preto sme museli pri vytváraní NED súboru postupovať presne podľa konfigurácie v XML a dôkladne kontrolovať mená jednotlivých rozhraní. Nezodpovedajú totiž názvom aké sa používajú v smerovačoch, ale musia byť pomenované trojznakovým reťazcom (eth pre Ethernet linky, ppp pre sériové linky) a číslom od 0 do 16.

Na nasledujúcom obrázku je zobrazená vizualizácia celej siete po spustení simulácie. Rozmiestnenie jednotlivých prvok je možné upravit zmenou príslušných súradníc v popisnom NED súbore.

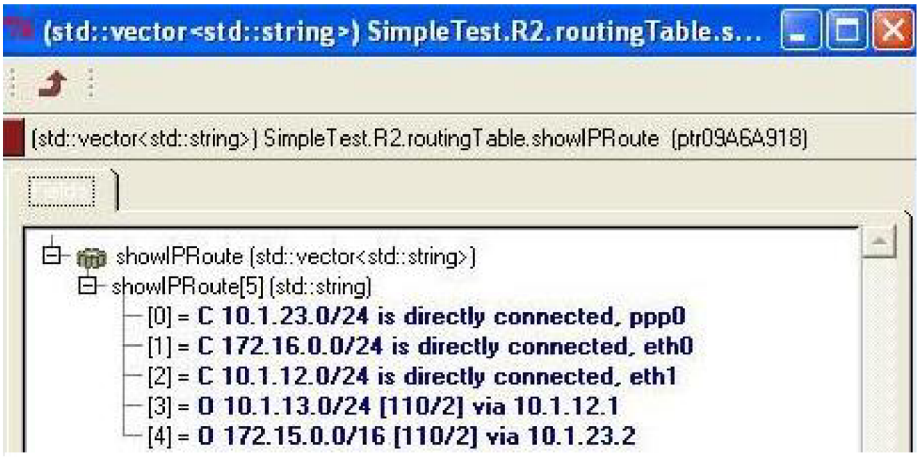


Obrázek 6.2: Vizualizácia simulovanej siete v OMNeT++

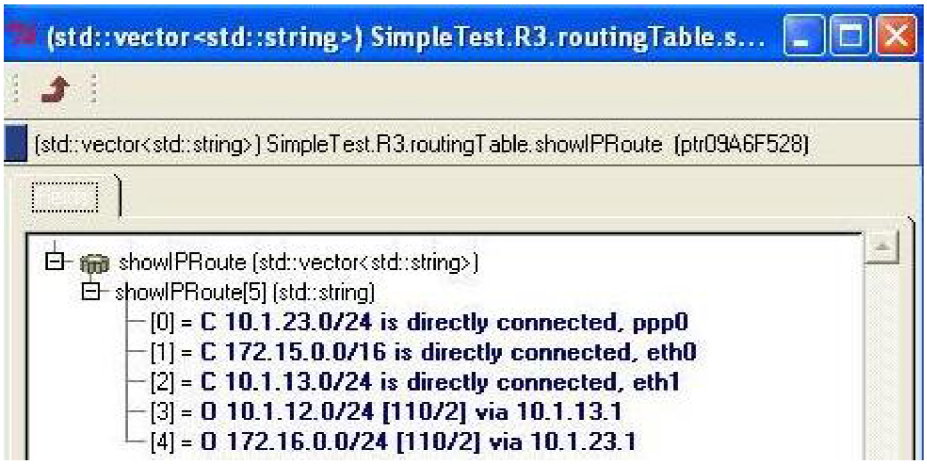
Na začiatku obsahujú smerovacie tabuľky iba cesty od priamo pripojených rozhraní. Po spustení simulácie na určitú dobu, potrebnú na skonvergovanie siete, sme si zobrazili obsahy smerovacích tabuliek všetkých smerovačov. Sú zobrazené na nasledujúcich obrázkoch.



Obrázek 6.3: Smerovacia tabuľka smerovača R1



Obrázek 6.4: Smerovacia tabuľka smerovača R2



Obrázek 6.5: Smerovacia tabuľka smerovača R3

Všetky smerovacie tabuľky obsahujú päť ciest do všetkých podsietí v topológii. Sieť teda skonvergovala správne bola zabezpečená konektivita medzi klientskymi počítačmi. OMNeT++ ale nemá implementovaný žiaden nástroj s podobnou funkcionalitou ako program tracert. Bola tak možná iba vizuálna kontrola toku dát.

6.4 Porovnanie dosiahnutých výsledkov

Porovnaním smerovacích tabuliek získaných z reálnych zariadení s tabuľkami modulov simulácie sme zistili, že výsledky sa v nami sledovaných parametroch zhodujú. Zároveň sme overili základnú funkčnosť vytvoreného modulu ANSARouter. Výsledky simulácie sa však nemusia zhodovať vždy so správaním reálnych zariadení. Za určitých podmienok, pri ktorých záleží na detailoch implementácie jednotlivých protokolov, sa výsledky môžu výrazne líšit. Použiteľnosť modulu ANSARouter na simulovanie základných vlastností protokolov RIP a OSPF sme však potvrdili.

Kapitola 7

Záver

Cieľom tejto práce bolo zoznámiť sa so simulčným nástrojom OMNeT++, navrhnúť a implementovať nový modul, ktorý by obsahoval základnú funkcionalitu smerovačov Cisco.

V teoretickej časti práce je opísaný jazyk NED, ktorý sa používa na popis simulovaných sietí a jednotlivých modulov, z ktorých sa simulácia skladá. Následne som sa v práci zaoberal funkcionalitou smerovačov Cisco. Čitateľ získa prehľad o základných vlastnostiach vybraných smerovacích protokolov a ACL.

V praktickej časti som sa zaoberal návrhom modulu ANSARouter a popisu zmien, ktoré bolo potrebné implementovať do už existujúceho modulu OSPFRouter, ktorý mi slúžil ako základ pre nový modul.

Práca by mohla pokračovať rozšírením modulu ANSARouter o ďalšie smerovacie protokoly, ako napríklad EIGRP alebo BGP. Zaujímavá je aj možnosť implementácie pokročilých vlastností už implementovaných protokolov, vrátane podpory dynamickej zmeny určitých nastavení (napríklad časovačov).

Práca mi priniesla nové znalosti z oblasti simulovania počítačových sietí, o fungovaní simulačného nástroja OMNeT++ a spôsobe rozšírenia INET Frameworku o nový modul. S rastúcimi nárokmi na priepustnosť a dostupnosť počítačových sietí, bude v budúcnosti pravdepodobne rásť aj význam ich simulácie, ako spôsobu na overenie a testovanie kľúčových parametrov jednotlivých topológii.

Literatura

[1] C. Hedrick: Routing Information Protocol. http://tools.ietf.org/html/rfc1058, Jún 1988.

[2] Gary Scott Malkin: RIP Version 2. http://tools.ietf.org/html/rfc2453, November 1998.

[3] Rob Coltun, Dennis Ferguson, John Moy : OSPF for IPv6. http://tools.ietf.org/html/rfc2740, December 1999.

[4] Rybová, V.: *Modelování a simulace návrhových vzorů směrování v počítačových sítích.* Bakalářská práce, FIT VUT v Brně, 2009.

[5] Scherfel, P.: *Simulace chování sítě na základě analýzy konfiguračních souborů aktivních síťových zařízení.* Bakalářská práce, FIT VUT v Brně, 2009.

[6] Suchomel, T.: *Rozšíření simulátoru OMNeT++ o filtrovací pravidla ACL.* Bakalářská práce, FIT VUT v Brně, 2009.

[7] WWW stránky: Configuring EIGRP. http://www.cisco.com/en/US/docs/ios/12_2/ip/configuration/guide/1cfeigrp.html, [cit. 17. 05. 2009].

[8] WWW stránky: Configuring IP Access Lists. http://www.cisco.com/en/US/products/sw/secursw/ps1018/products_tech_note09186a00800a5b9a.shtml, [cit. 17. 05. 2009].

[9] WWW stránky: EIGRP Commands. http://www.cisco.com/en/US/docs/ios/12_2/iproute/command/reference/1rfeigrp.html, [cit. 17. 05. 2009].

[10] WWW stránky: NED Reference. http://omnetpp.org/doc/omnetpp40/manual/usman.html#sec426, [cit. 17. 05. 2009].

[11] WWW stránky: OMNeT++ User Manual. http://omnetpp.org/doc/omnetpp40/manual/usman.html, [cit. 17. 05. 2009].

[12] WWW stránky: OSPF Commands. http://www.cisco.com/en/US/docs/ios/12_2t/ip_route/command/reference/p2ftospf.html, [cit. 17. 05. 2009].

[13] WWW stránky: What Is Administrative Distance? http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094195.shtml, [cit. 17. 05. 2009].

Dodatek A

Obsah CD

Obsah priloženého CD:

- **zdrojové súbory textu práce** - v adresári *sprava*
- **zdrojové súbory OMNeT++ a INET Framework** - v adresári *install*
- **zdrojové súbory modulu ANSARouter** - v adresári *sources*
- **ukážky simulácii** - v adresári *examples*
- **manual.pdf** - návod na inštaláciu

Dodatek B

Popis modulu ANSARouter v jazyku NED

```
package inet.ansa;

import inet.base.NotificationBoard;
import inet.ansa.InterfaceStateManager.InterfaceStateManager;
import inet.ansa.ethernet.EthernetInterface;
import inet.ansa.ppp.PPPInterface;
import inet.nodes.inet.NetworkLayer;
import inet.ansa.routingTable.AnsaInterfaceTable;
import inet.ansa.routingTable.AnsaRoutingTable;
import inet.ansa.ospfv2.AnsaOSPFRouting;
import inet.ansa.rip.RIPRouting;
import inet.transport.udp.UDP;
import inet.ansa.acl.acl;

module ANSARouter
{
    parameters:
        @node();
        string routingFile = default(,,‘‘);
        string hostname = default(,,‘‘);
        string routerId = default(,,‘‘);
        @display(,,i=srouter‘‘);
    gates:
        inout pppg[];
        inout ethg[];
    submodules:
        notificationBoard: NotificationBoard {
            parameters:
                @display(,,p=45,42‘‘);}
        interfaceStateManager: InterfaceStateManager {
            parameters:
                @display(,,p=154,42‘‘);}
```

```
    interfaceTable: AnsaInterfaceTable {
        parameters:
            @display(,,p=250,42‘‘);}
    routingTable: AnsaRoutingTable {
        parameters:
            IPForward = true;
            routerId = routerId;
            @display(,,p=323,42‘‘);}
    ospf: AnsaOSPFRouting {
        parameters:
            @display(,,p=297,126,row‘‘);}
    rip: RIPRouting {
        parameters:
            hostname = hostname;
            @display(,,p=45,195,row;i=block/network2‘‘);}
    udp: UDP {
        parameters:
            @display(,,p=84,126;i=block/transport‘‘);
        gates:
            appIn[1];
            appOut[1];}
    acl: acl {
        parameters:
            @display(,,p=183,234,row;i=block/network2‘‘);
        gates:
            toNetworkLayerIn[sizeof(pppg)+sizeof(ethg)];
            toNetworkLayerOut[sizeof(pppg)+sizeof(ethg)];
            ifIn[sizeof(pppg)+sizeof(ethg)];
            ifOut[sizeof(pppg)+sizeof(ethg)];}
    networkLayer: NetworkLayer {
        parameters:
            @display(,,p=183,143;q=queue‘‘);
        gates:
            ifIn[sizeof(pppg)+sizeof(ethg)];
            ifOut[sizeof(pppg)+sizeof(ethg)];}
    ppp[sizeof(pppg)]: PPPInterface {
        parameters:
            @display(,,p=115,283,row,110;q=l2queue‘‘);}
    eth[sizeof(ethg)]: EthernetInterface {
        parameters:
            @display(,,p=258,283,row,110;q=l2queue‘‘);}
connections allowunconnected:
    ospf.ipOut --> networkLayer.ospfIn;
    ospf.ipIn <-- networkLayer.ospfOut;
    rip.udpOut --> udp.appIn[0];
    rip.udpIn <-- udp.appOut[0];
    udp.ipOut --> networkLayer.udpIn;
    udp.ipIn <-- networkLayer.udpOut;
```

```
    for i=0..sizeof(pppg)-1 {
        pppg[i] <--> ppp[i].phys;
        ppp[i].netwOut --> acl.ifIn[i];
        acl.toNetworkLayerOut[i] --> networkLayer.ifIn[i];
        ppp[i].netwIn <-- acl.ifOut[i];
        acl.toNetworkLayerIn[i] <-- networkLayer.ifOut[i];}

    for i=0..sizeof(ethg)-1 {
        ethg[i] <--> eth[i].phys;
        eth[i].netwOut --> acl.ifIn[sizeof(pppg)+i];
        acl.toNetworkLayerOut[sizeof(pppg)+i]
--> networkLayer.ifIn[sizeof(pppg)+i];
        eth[i].netwIn <-- acl.ifOut[sizeof(pppg)+i];
        acl.toNetworkLayerIn[sizeof(pppg)+i]
<-- networkLayer.ifOut[sizeof(pppg)+i];}

}
```

Dodatek C

Konfiguračný NED súbor

```
package inet.examples.ansaExamples.finalTest;
import inet.ansa.ANSARouter;
import inet.world.ChannelInstaller;
import ned.DatarateChannel;
import inet.nodes.inet.StandardHost;

network SimpleTest
{
    parameters:
        @display(,,p=10,10;b=982,152‘‘);
    types:
        channel C extends DatarateChannel
        { delay = 0.1us; }
    submodules:
        channelInstaller: ChannelInstaller {
            parameters:
                channelClass = ,,ThruputMeteringChannel‘‘;
                channelAttrs = ,,format=#N‘‘;
                @display(,,p=60,40‘‘);}
      R1: ANSARouter {
            parameters:
                @display(,,p=216,22‘‘);
            gates:
                ethg[2];}
        R2: ANSARouter {
            parameters:
                @display(,,p=152,92‘‘);
            gates:
                ethg[2];
                pppg[1];}
        R3: ANSARouter {
            parameters:
                @display(,,p=262,92‘‘);
            gates:
                ethg[2];
```

```
                pppg[1];}
        H1: StandardHost {
            parameters:
                @display(,,p=56,92;i=device/laptop‘‘);
            gates:
                ethg[1];}
        H2: StandardHost {
            parameters:
                @display(,,p=360,92;i=device/laptop‘‘);
            gates:
                ethg[1];}
    connections:
        R1.ethg[1] <--> C <--> R2.ethg[1];
        R1.ethg[0] <--> C <--> R3.ethg[1];
        R2.pppg[0] <--> C <--> R3.pppg[0];
        R2.ethg[0] <--> C <--> H1.ethg[0];
        R3.ethg[0] <--> C <--> H2.ethg[0];
}
```