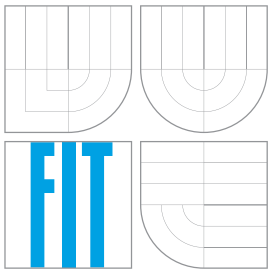
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODELOVÁNÍ SMĚROVACÍCH PROTOKOLŮ OSPF V SIMULÁTORU OMNET++

MODELLING OSPF ROUTING PROTOCOLS USING OMNET++ SIMULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN DANKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ONDŘEJ RYŠAVÝ, Ph.D.

BRNO 2009

Abstrakt

Open Shortest Path First (OSPF) je dynamický, hierarchický směrovací protokol navrhnutý pro podporu směrovaní v sítích TCP/IP. Je tu potřeba pro rozsáhlé simulace protokolu OSPF, aby bylo možné lépe porozumět jeho dynamice. Obsahem této bakalářské práce je modelovaní a simulace protokolu OSPF v diskrétním simulačním systéme OMNeT++. Hlavní zaměření patří architektuře již existujíce implementaci v module OSPFRouting. Modul OSPFRouting je taktéž ověřeny vůči OSPF implementaci na reálních směrovačích. Dále je na konkrétných problémech demonstrované použití OSPF simulace.

Abstract

Open Shortest Path First (OSPF) is a dynamic, hierarchical routing protocol designed to support routing in TCP/IP networks. There is a deliberate need for large-scale simulation of OSPF protocol in order to understand its dynamics. Content of this bachelor's thesis is modeling and simulation of OSPF protocol in discrete event simulation system OMNeT++. The main focus is to give an insight of the architecture of the existing implementation in module OSPFRouting. The module OSPFRouting is also validated against OSPF implementation on the real routers. Next, the usage of OSPF simulation is demonstrated on the specific experiments.

Klíčová slova

OSPF, OMNeT++, INET Framework, diskrétní simulace, směrovací protokol

Keywords

OSPF, OMNeT++, INET Framework, discrete simulation, routing protocol

Citace

Martin Danko: Modelování směrovacích protokolů OSPF v simulátoru OMNeT++, bakalářská práce, Brno, FIT VUT v Brně, 2009

Modelování směrovacích protokolů OSPF v simulátoru OMNeT++

Prohlášení

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

.....
Martin Danko
18. mája 2009

Poděkování

Děkuji vedoucímu mé bakalářske práce Ing. Ondřejovi Ryšavému Ph.D. za možnost odborných konzultací a za jeho cenné rady.

Obsah

Kapitola 1

Úvod

Cieľom tejto práce je zanalyzovať implementáciu smerovacieho protokolu OSPF v prostredí INET Framework a porovnať jeho správanie voči implementácii na reálnych zariadeniach spoločnosti Cisco, respektíve štandardu definovanom v RFC 2328 [5]. Korektnosť simulácie je potrebné demonštrovať na prípadovej štúdii, vytvorenej na základe návrhových vzorov. Ďalšou úlohou je ukázať možnosti využitia simulácie pre overenie vlastností typu bezpečnosť v danom modeli.

Práca je vytvorená v rámci výskumnej skupiny NES@FIT a je súčasťou projektu ANSA (Automated Network-Wide Security Analysis). Projekt sa zaoberá verifikáciou sieťových topológii na špecifické vlastnosti ako dostupnosť, bezpečnosť či schopnosť odolávať poruchám. Práca bude spolu s prácami ostaných študentov vytvárať modul reálneho smerovača v prostredí INET Framework. Pomocou tohto smerovača bude následne možné vytvárať a simulovať komplexné topológie rozsiahlych sieti.

Prvé dve kapitoly majú čitateľa uviesť do problematiky smerovania a simulácie. Konkrétne kapitola 2 sa zaoberá smerovaním v IP sieťach a obsahuje prehľad smerovacích protokolov so zameraním na OSPF. Kapitola 3 popisuje architektúru simulátora OMNeT++ a spôsob vytvárania modulov. Tiež popisuje moduly INET Framework, ktorý je nadstavbou OMNeT++ pre simuláciu IP sietí. V kapitole 4 je podrobnejšie preskúmaná implementácia OSPF v INET Framework – modul OSPFRouting. V rámci tejto kapitoly je ukázaná korektnosť simulácie voči reálnym zariadeniam a na module sú vykonané výkonnostné testy definované pre smerovací protokol OSPF v RFC 4061 [4]. V poslednej kapitole 5 sú ukázané možnosti využitia OSPF simulácie. Na začiatku sú prezentované zmeny v INET Framework a v module OSPFRouting, ktoré bolo treba implementovať, aby bolo možné simulovať zmeny stavu rozhraní smerovača. Následne je na troch konkrétnych simuláciách skúmaný protokol OSPF z rôznych aspektov a to rýchlosti konvergencie, odolnosti voči chybovosti liniek a odolnosť voči častým zmenám topológie.

Kapitola 2

Smerovanie a protokol OSPF

Táto kapitola je úvodom do problematiky smerovania a obsahuje prehľad smerovacích protokolov. Zameriava sa na smerovací protokol OSPF, na jeho architektúru definovanú v RFC 2328 [5]. Dôležitosť tejto kapitoly spočíva v definovaní a objasnení pojmov z oblasti smerovania, ktoré budú následne diskutované v ďalších kapitolách.

2.1 Smerovanie

Smerovanie je proces výberu cesty v počítačovej sieti, ktorou budú posielané dátové informácie (pakety). Smerovače môžu pracovať dvoma základnými spôsobmi: môžu využívať dopredu nakonfigurované statické cesty, alebo si môžu cesty vypočítavať dynamicky, prostredníctvom niektorého z mnohých dynamických smerovacích protokolov. Pomocou týchto dynamických smerovacích protokolov smerovače rozpoznávajú jednotlivé cesty.

Staticky konfigurované smerovače nemôžu rozpoznať žiadne cesty. Chýba v nich akýkoľvek prostriedok pre výmenu smerovacích informácií s ostatnými smerovačmi. Tieto statické smerovače dokážu odosielať pakety len podľa ciest definovaných administrátorom.

Okrem statickej konfigurácie ciest máme k dispozícii tri kategórie dynamických smerovacích protokolov:

- Protokoly s vektorom vzdialenosti
- Protokoly so stavom linky
- Hybridné protokoly

Zoznam smerovacích ciest, ktoré má smerovač k dispozícii, sa nazýva smerovacia tabuľka. Každý smerovač v sieti udržuje vlastnú smerovaciu tabuľku, takže obsah jednej smerovacej tabuľky sa líši od obsahu smerovacích tabuliek iných smerovačov. Smerovacia tabuľka môže obsahovať ako statické cesty, tak aj cesty získané rôznymi smerovacími protokolmi. V prípade, že viacej protokolov má informácie o ceste do toho istého cieľa, do smerovacej tabuľky vloží informáciu ten protokol, ktorý má najmenšiu administratívnu vzdialenosť. Administratívna vzdialenosť vyjadruje relatívnu dôveryhodnosť jednotlivých smerovacích protokolov. Okrem toho má každý smerovací protokol vlastné hodnotiace kritéria na vyhodnotenie najlepšej cesty nazývané metrika. Metrika môže zohľadňovať náklady, počet preskokov, šírku pásma či oneskorenie linky. V prípade, že existuje viac ciest s najlepšou metrikou, niektoré smerovacie protokoly umožňujú vložiť tieto cesty do smerovacej tabuľky a vykonávať nad nimi rozloženie záťaže, a tým zvýšiť priepustnosť siete.

2.1.1 Protokoly s vektorom vzdialenosti

V smerovacích algoritmoch postavených na vektore vzdialenosti, nazývaných niekedy algoritmy Bellman-Ford, smerovače pravidelne predávajú kópie svojej smerovacej tabuľky bezprostredným susedom v sieti. Každý príjemca pričíta k tabuľke svoj vektor vzdialenosti – teda hodnotu svojej vlastnej vzdialenosti a opäť ju predá svojim bezprostredným susedom. Postupnými krokmi sa smerovače dozvedia o ostaných smerovačoch a urobia si ucelenú predstavu o "vzdialenostiach" v sieti.

Výhodou protokolov s vektorom vzdialenosti je jednoduchá implementácia a nízke nároky na procesor a pamäť smerovača. Nevýhodou je enormná záťaž na prenosové pásmo linky a pomalá konvergencia. V priebehu konvergencie môže byť sieť zraniteľná voči možnému nekonzistentnému smerovaniu alebo dokonca aj nekonečným slučkám.

Zástupcovia tejto skupiny protokolov sú RIP a IGRP.

2.1.2 Protokoly so stavom linky

Smerovacie algoritmy so stavom linky, označované ako protokoly najkratších ciest (shortest path first, SPF), si udržiavajú zložitú databázu topológie siete. Na rozdiel od protokolov s vektorom vzdialenosti tieto protokoly zisťujú úplné informácie o smerovačoch v sieti a spôsob ich vzájomného prepojenia a ďalej si tieto informácie udržujú. V rámci potrebnej informovanosti si s ostanými smerovačmi vymieňajú takzvané oznámenia o stave linky (link-state advertisement, LSA).

Každý smerovač, zapojený do výmeny smerovacích informácii, si zo všetkých prijatých oznámení o stave liniek konštruuje databázu s topológiou siete. Potom pomocou algoritmu najkratších ciest vypočíta dosažiteľnosť jednotlivých cieľov v sieti a podľa zistených informácií aktualizuje smerovaciu tabuľku. Uvedený proces dokáže rozpoznať zmeny v topológii siete spôsobené haváriou určitej komponenty alebo naopak rastom siete.

Výmeny oznámenia o stave linky (LSA) sa spúšťajú len pri vzniku udalosti v sieti. Proces teda nebeží periodicky. Tým sa výrazne urýchľuje konvergencia smerovacieho protokolu, pretože pre zahájenie procesu konvergencie nemusí smerovač čakať, kým vyprší rad rôzne nastavených časovačov.

Výhodou protokolov so stavom linky je ich už spomínaná rýchla konvergencia a nízka réžia pri stabilnom stave siete. Sú vhodné pre rozsiahle siete a vyznačujú sa svojou veľmi dobrou škálovatelnosťou. Na druhej strane nevýhodami sú vysoké nároky na pamäť a procesor smerovača, či zložitejšia implementácia a konfigurácia. Taktiež počas procesu prvotného rozpoznávania môžu zaplaviť prenosové prostriedky siete.

Zástupcovia tejto skupiny protokolov sú OSPF a ISIS.

2.1.3 Hybridné protokoly

Poslednou formou smerovania je takzvané hybridné smerovanie. Vyvážené hybridné smerovacie protokoly používajú metriky vektoru vzdialenosti, ale kladú dôraz na uplatnenie presnejších metrík než v konvenčných protokoloch postavených na vektore vzdialenosti. Konvergujú rýchlejšie než protokoly s vektorom vzdialenosti, zároveň sa však vyhýbajú réžii spojenej s aktualizáciou stavu liniek. Vyvážené hybridné protokoly nepracujú v pravidelných aktualizáciách, ale sú riadené udalosťami, takže zachovávajú dostupnú šírku pásma.

I keď existujú aj určité "otvorené" hybridné protokoly, z obrovskej časti sú tieto technológie spojené s proprietárnym vývojom spoločnosti Cisco. Smerovací protokol EIGRP

v sebe spája tie najlepšie stránky smerovacích protokolov s vektorom vzdialenosti a so stavom linky.

2.2 Smerovací protokol OSPF

Smerovací protokol OSPF (Open Shortest Path First) bol vyvinutý organizáciou IETF (Inernet Engineering Task Force), ako náhrada problematického protokolu RIP. OSPF je protokol so stavom linky, a ako jeho meno napovedá, používa Dijkstrov algoritmus najkratšej cesty. Prvá verzia OSPF bola definovaná v RFC 1131. Od tej doby bol protokol vylepšený cez niekoľko RFC a v súčasnej dobe je aktuálne definovaný v RFC 2328.

2.2.1 Činnosť OSPF

Činnosť OSPF, veľmi zjednodušene, je možné popísať nasledovne:

1. Smerovače posielajú Hello pakety všetkými rozhraniami, na ktorých je povolený protokol OSPF. Keď dvom smerovačom, zdieľajúcim spoločnú linku, súhlasia parametre v patričných Hello paketoch, tak sa môžu stať susedmi (neighbors).

2. OSPF definuje niekoľko typov sietí a niekoľko typov smerovačov. Vytvorenie susedstva je určené typom smerovačov vymieňajúcich si Hello pakety a typom siete, po ktorej sú Hello pakety prenášané.

3. Každý smerovač posiela oznámenie o stave linky (LSA) všetkým smerovačom, s ktorými má vytvorené susedstvo. LSA obsahujú informácie o všetkých linkách a rozhraniach smerovača a o stave týchto liniek a rozhraní. Pretože typy informácií o stave linky sú rôzne, OSPF definuje viacero typov LSA.

4. Každý smerovač, ktorý príjme LSA od svojho suseda, zaznamená tieto informácie do databázy stavu linky (link state database). Následne pošle LSA svojim ostatným susedom.

5. Pomocou záplavy LSA (LSA flooding) cez celú oblasť (area), všetky smerovače vytvoria identické databázy stavu linky.

6. Keď sú databázy kompletné, všetky smerovače použijú SPF algoritmus k výpočtu acyklického grafu, ktorý obsahuje najkratšiu cestu do každého známeho cieľa. Tento graf sa nazýva SPF strom (SPF tree).

7. Každý smerovač vytvorí svoju vlastnú smerovaciu tabuľku na základe SPF stromu.

Keď všetky LSA dorazili všetkým smerovačom v oblasti, všetky databázy stavu linky boli zosynchronizované a smerovacie tabuľky boli vytvorené, OSPF sa stáva takzvaným "tichým" protokolom. Hello pakety sú vymieňané medzi susedmi kvôli udržaniu si kontaktu. LSA sú opätovne prenesené každých 30 minút. Pokiaľ je sieť stabilná, žiadna iná aktivita by nemala nastať.

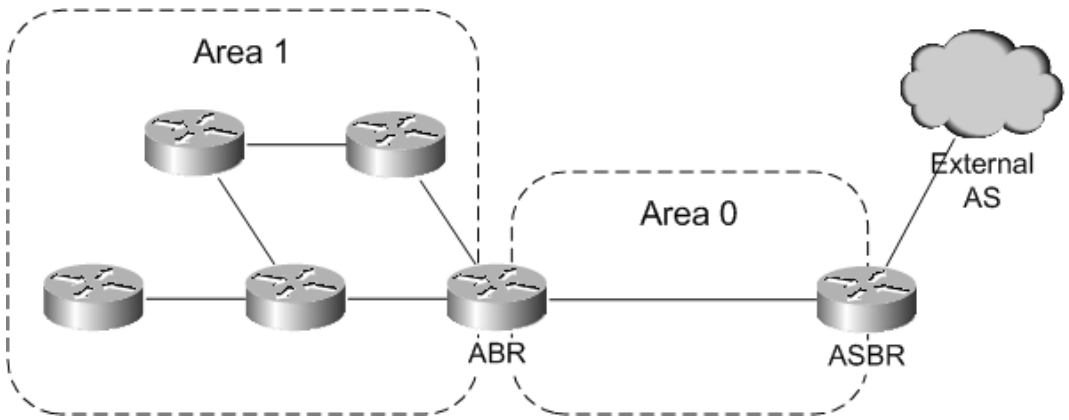
2.2.2 Oblasti a virtuálne linky

Sieť so smerovacím protokolom OSPF je možné rozdeliť do menších častí nazývaných oblasti. Oblasť je logický súbor OSPF sietí, smerovačov a liniek, ktoré majú rovnaký identifikátor oblasti. Pre každú oblasť, do ktorej smerovač patrí, si musí udržiavať databázu stavu linky. Smerovač nemá podrobné informácie o topológii siete mimo oblastí, do ktorých patrí. To umožňuje redukovať veľkosť databázy stavu linky.

Pre vytváranie oblastí platia nasledujúce pravidlá:

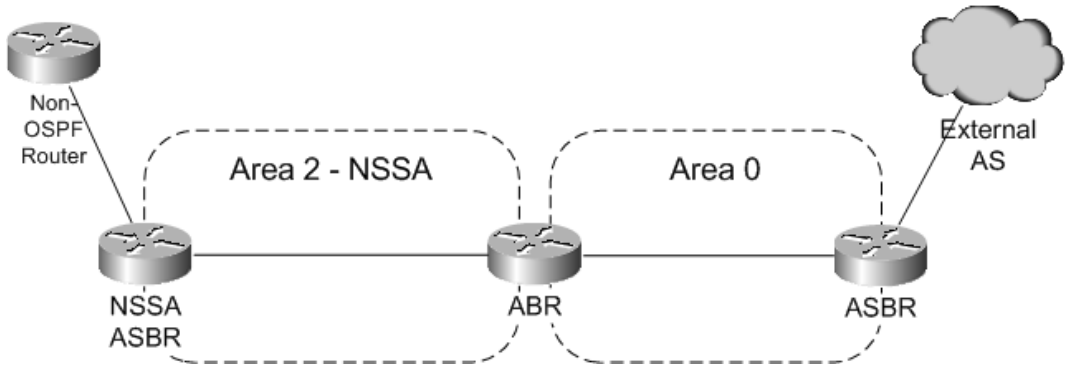
- V každej OSPF sieti musí existovať tzv. chrbticová oblasť (oblasť 0).
- Každá nechrbticová oblasť musí byť priamo pripojená do chrbticovej oblasti (fyzicky alebo pomocou virtuálnej linky).
- Chrbticová oblasť nesmie byť za žiadnych podmienok prerušená.

Smerovač s rozhraniami v dvoch alebo viacerých oblastiach sa nazýva hraničný smerovač oblasti (ABR). ABR tvorí hranicu medzi oblasťami. Hranicu medzi autonómnym systémom OSPF siete a iným autonómnym systémom tvorí hraničný smerovač autonómneho systému (ASBR).



Obrázok 2.1: Pozícia ABR a ASBR v sieti

Okrem štandardných oblastí OSPF definuje tzv. stub oblasti. Stub oblasť je oblasť, do ktorej sa nešíria externé cesty. Šíri sa len východisková cesta, pomocou ktorej je možné dosiahnuť externé siete. Ďalším typom oblasti je no-so-stuby oblasť (NSSA). Presne ako stub oblasť aj NSSA bráni šíreniu externých LSA do oblasti, ale umožňuje prenos externých ciest z ASBR do ABR.



Obrázok 2.2: Ukážka NSSA oblasti

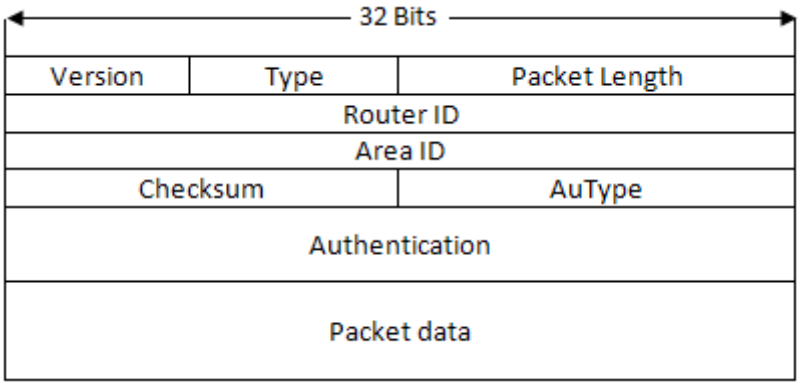
Všetky oblasti v autonómnom systéme OSPF musia byť spojené s chrbticovou oblasťou. V niektorých prípadoch však fyzické spojenie nie je možné, preto je možné použiť virtuálnu linku pre pripojenie do chrbticovej oblasti cez nechrbticovú oblasť. Virtuálna linka môže byť použitá aj na prepojenie dvoch rozdelených chrbticových oblastí cez nechrbticovú oblasť. Oblasť, cez ktorú je vytváraná virtuálna linka, musí mať kompletné smerovacie informácie. Stub oblasť nemôže byť súčasne aj tranzitnou oblasťou virtuálnej linky.

2.2.3 Dátové štruktúry OSPF

OSPF podporuje rôzne typy sietí a hierarchické členenie sietí do oblasti, preto je komunikácia medzi jednotlivými smerovačmi pomerne zložitá. OSPF pre komunikáciu definuje viacero typov správ, ktoré sú zapuzdrené do jednotnej štruktúry - OSPF paketu.

OSPF paket

OSPF pakety sú prenášané priamo na sieťovej vrstve (nevyužívajú služby žiadneho transportného protokolu). Všetky OSPF pakety začínajú 24 bytovou hlavičkou, ako je znázornená na obrázku 2.3.



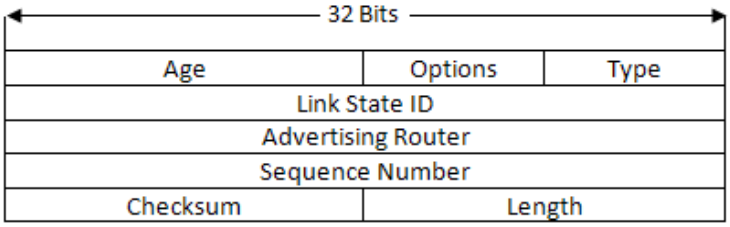
Obrázok 2.3: Hlavička OSPF paketu

Nasleduje popis jednotlivých častí hlavičky:

- **Version** – identifikuje verziu OSPF.
- **Type** – identifikuje typ OSPF paketu. Existuje 5 nasledujúcich typov:
  - **Hello** – ustanovenie a udržiavanie susedstva.
  - **Database description** – popisuje obsah databázy stavu linky. Tieto správy sú posielané po inicializácii susedstva.
  - **Link-state request** – požiadavka konkrétnych častí databázy stavu linky. Tieto správy sú posielané, keď smerovač zistí (vyhodnotením database description paketu), že niektoré časti jeho databázy stavu linky sú zastaralé.
  - **Link-state update** – odpoveď na link-state request paket. Tieto správy sa používajú aj pri pravidelnom rozosielaní LSA. Jeden link-state update paket môže obsahovať niekoľko LSA.
  - **Link-state acknowledgment** – potvrdenie príjmu link-state update paketu.
- **Packet length** – určuje dĺžku paketu aj s OSPF hlavičkou v bytoch.
- **Router ID** – identifikuje zdroj paketu.
- **Area ID** – identifikuje oblasť, do ktorej paket patrí. Každý OSPF paket patrí len do jednej oblasti.
- **Checksum** – kontrolný súčet paketu.
- **AuType** – obsahuje typ autentifikácie.
- **Authentication** – obsahuje autentifikačné informácie.
- **Packet data** – obsahuje informácie vyšších vrstiev.

LSA paket

LSA pakety obsahujú informácie o stave liniek. Je definovaných 11 rôznych typov LSA podľa typu siete, o ktorej sú prenášané informácie. Súbor všetkých LSA vytvára databázu stavu linky. Jednotlivé LSA pakety sú prenášane v rámci Link-state update (LSU) paketov. Všetky LSA pakety začínajú 20 bytovou hlavičkou, ako je znázornená na obrázku 2.4.



Obrázok 2.4: Hlavička LSA paketu

Nasleduje popis jednotlivých častí hlavičky:

- **Age** – udáva počet sekúnd uplynutých od okamžiku vytvorenia LSA.
- **Options** – obsahuje niekoľko príznakov s identifikáciou rôznych voliteľných služieb.
- **Type** – definuje typ LSA. Existuje 11 nasledujúcich typov:
  - **Typ 1** – Router LSA posielané zo smerovača ostatným smerovačom v tej istej oblasti. Obsahuje informácie o rozhraniach smerovača patriacich do danej oblasti.
  - **Typ 2** – Network LSA je generované DR smerovačom na segmentoch s mnohonásobným prístupom. Obsahuje podobné informácie ako typ 1.
  - **Typ 3** – Network Summary LSA je generované ABR smerovačom a obsahuje podsiete a ich ceny, ale neobsahuje informácie o stavoch liniek.
  - **Typ 4** – ASBR summary LSA je svojou štruktúrou podobné typu 3 a posiela sa pri prekročení hraníc autonómneho systému.
  - **Typ 5** – AS external LSA je vytvorené v ASBR a popisuje externé siete.
  - **Typ 6** – Group Membership LSA je definované ako multicastové rozšírenie OSPF.
  - **Typ 7** – NSSA External LSA je generované ASBR v NSSA oblasti.
  - **Typ 8** – link-local only LSA je používané pre IPv6 verziu OSPF.
  - **Typ 9 až 11** – definované ako Opaque LSA sú rezervované pre budúce použitie.
- **Link State ID** – vymedzuje konkrétne časti sieťového prostredia, ktoré toto LSA popisuje.
- **Advertising Router** – identifikuje pôvodcu LSA.
- **Sequence number** – určuje poradové číslo LSA a slúži k detekcii novších LSA.
- **Checksum** – kontrolný súčet slúžiaci k detekcii poškodenia LSA.
- **Length** – oznamuje veľkosť celého LSA.

Kapitola 3

OMNeT++ a INET Framework

Kapitola popisuje architektúru simulačného nástroja OMNeT++ a spôsob vytvárania modulov. Ďalej sa venuje nadstavbe INET Framework, ktorá obsahuje sadu simulačných modulov pre rodinu protokolov TCP/IP. Popisuje jednotlivé moduly a spôsob vytvárania simulačných modelov.

3.1 Simulátor OMNeT++

OMNeT++ je objektovo orientovaný, modulárny, diskrétny simulátor s otvorenou architektúrou. Je využívaný najmä v simulácii telekomunikačných a počítačových sietí a k modelovaniu rôznych sieťových protokolov. Vďaka svojej flexibilnej architektúre však umožňuje simuláciu radu iných systémov, ako napr. validácia hardwarových architektúr, vyhodnocovanie výkonnosti komplexných softwarových systémov, či dokonca modelovanie rôznych obchodných procesov. Taktiež má implicitnú podporu pre paralelnú distribuovanú simuláciu (napr. MPI).

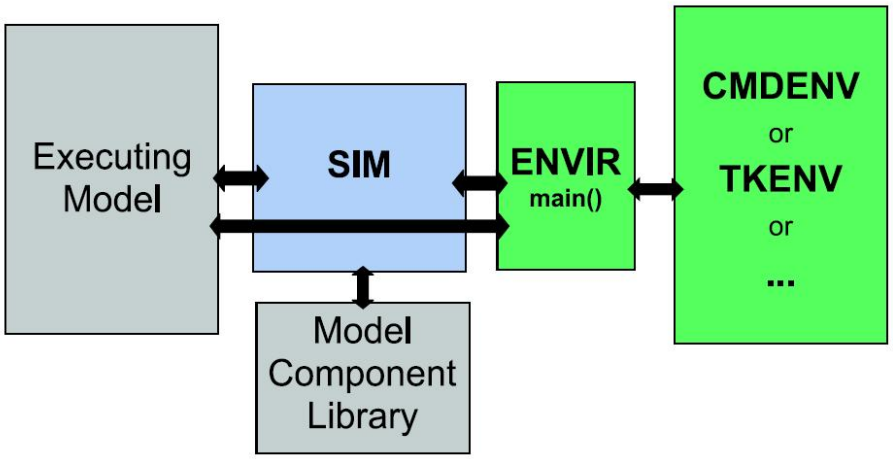
OMNeT++ poskytuje silnú podporu pre grafické rozhranie, čo mu zaručuje obľubu u užívateľov. Svedčí o tom aj viac ako 40 simulačných modelov, tzv. frameworkov, ktoré rozširujú jeho možnosti do rôznych oblastí, napr. simulácia lokálnych a internetových sietí – INET Framework, simulácia mobilných a bezdrôtových sietí – Mobility Framework či simulácia senzorových sietí – NesCT.

História OMNeT++ začala v roku 1992 na Technickej Univerzite v Budapešti. Hlavnú zásluhu na jeho vývoji má András Varga. V roku 2001 sa projekt dostal pod záštitu Univerzity v Karlsruhu. V dobe písania tejto práce je aktuálna verzia 4 a neustále je vyvíjaná.

Ako už bolo uvedené, OMNeT++ je diskrétny simulátor. Znamená to, že zmeny v simulačnom systéme sa odohrávajú v určitom okamihu spojitého alebo diskrétneho simulačného času. Každá zmena v simulačnom systéme (udalosť) je naplánovaná v kalendári udalostí a je určená časovou známkou (timestamp) a prioritou. V priebehu simulácie sa z kalendára udalostí vyberajú udalosti s najnižšou časovou známkou. Pokiaľ je časová známka rovnaká u viacerých udalostí, vyberie sa udalosť s najvyššou prioritou a tá sa vykoná. V prípade, že aj priority sa zhodujú, OMNeT++ vyberie udalosť na základe poradia v akom boli vkladané do kalendára udalostí. Doba spracovania udalosti je vzhľadom k simulačnému času nulová. Časove úseky simulačného času, v ktorých nenastáva žiadna zmena systému, sú jednoducho preskočené.

3.1.1 Architektúra OMNeT++

OMNeT++ má modulárnu architektúru. Obrázok 3.1 zobrazuje jej "high-level" pohľad. Sú v ňom naznačené jednotlivé stavebné moduly simulátora a ich vzájomná interakcia. Moduly budú podrobne popísané v nasledovnom texte.



Obrázok 3.1: Architektúra OMNeT++

Sim

Modul obsahuje simulačné jadro a knižnice tried, ktoré sú využívané počas simulácie. Knižnice sa prilinkujú k samotnej simulácii staticky alebo dynamicky.

Envir

Je ďalší modul, ktorý obsahuje knižnice zdieľané všetkými užívateľskými rozhraniami. Jeho súčasťou je funkcia *main()* určujúca vstupný bod programu a obsahuje cyklus pre výber udalostí z kalendára udalostí. Envir poskytuje funkcie pre spracovanie konfiguračných súborov ".ini" jednotlivých implementácii užívateľského rozhrania. Okrem toho sa stará o zachytenie výnimiek a riešenie chýb, ktoré by mohli nastať počas simulácie v jadre. Taktiež zaisťuje načítanie potrebných modulov pri inicializácii, zápis do výstupných súborov a výpis správ pri ladení.

Cmdenv a Tkenv

Sú to moduly špecifikujúce užívateľské rozhranie. Cmdenv je užívateľské rozhranie pre spúšťanie simulácie z príkazového riadku. Vyznačuje sa svojou rýchlosťou a možnosťou spúšťania dávok simulácií pomocou skriptov. Naproti tomu Tkenv je grafické užívateľské rozhranie implementované v multiplatformovom prostredí Tcl/Tk. Jeho výhodou je interaktívne spúšťanie simulácie s možnosťou krokovania a následným vizuálnym znázornením jej výsledkov.

Model Component Library

Obsahuje definície všetkých jednoduchých NED modulov simulácie a ich implentáciu v programovacom jazyku C++. Ďalej obsahuje zložené moduly, kanály, siete a správy. Zjedno-

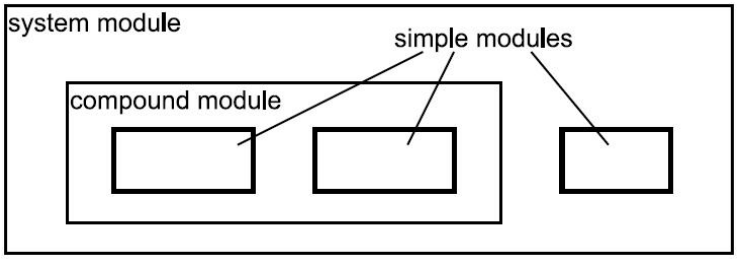
dušene povedané, obsahuje všetko, čo patrí k samotnému simulovanému modelu a všetko čo bolo prilinkované k simulačnému programu.

Executing model

Reprezentuje model vytvorený pre danú simuláciu. Obsahuje objekty, ktoré sú inštanciami komponent definovaných v Model Component Library. Prístup k týmto objektom je realizovaný cez globálny objekt simulation, ten tvorí koreň stromu všetkých objektov.

3.1.2 Modelovanie v OMNeT++

Každý model simulovaný v OMNeT++ je zložený z hierarchicky usporiadaných modulov, ktoré spolu komunikujú pomocou zasielania správ. Na najvyššej úrovni je systémový modul. Systémový modul obsahuje podmoduly, ktoré tiež môžu obsahovať vlastné podmoduly. Hĺbka zanorenia modulov nie je obmedzená. Moduly obsahujúce podmoduly sa nazývajú zložené moduly (Compound Modules). Naopak moduly na najnižšej úrovni hierarchie, ktoré už neobsahujú žiadne podmoduly sa nazývajú jednoduché moduly (Simple Modules). Správanie jednoduchých modulov sa implementuje jazykom C++. Štruktúra modelu v OMNeT++ sa popisuje pomocou interného jazyka NED. Príklad takejto štruktúry je znázornený obrázku 3.2.



Obrázok 3.2: Hierarchia vytváraných modulov v OMNeT++

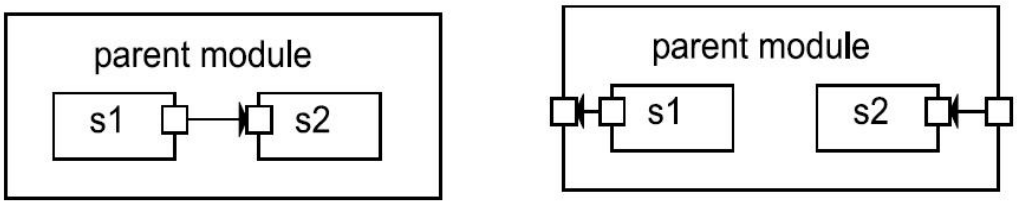
Správy, brány, linky

Moduly komunikujú výmenou správ. Správy môžu obsahovať ľubovoľné množstvo prenášaných dát. Jednoduché moduly umožňujú zasielanie správ priamo cieľovému modulu, alebo cez preddefinované cesty zložené z brán (Gates) a spojení (Connections). Brány sú vstupné a výstupné rozhrania. Správy sú posielané cez výstupné a prijímané cez vstupné brány. Spojenia, tiež nazývané linky, spájajú jednotlivé brány medzi sebou, a to vždy na úrovni jedného modulu. To znamená, že v rámci zloženého modulu je možné spojiť brány medzi jednotlivými podmodulmi alebo medzi podmodulom a rodičovským modulom (obrázok 3.3).

Prenos správ (paketov)

Každému spojeniu môžu byť definované tri parametre, ktoré umožňujú modelovanie komunikácie po sieti, ale môžu byť užitočné aj pre iné modely. Sú to nasledujúce parametre:

- Oneskorenie (propagation delay) – časový rozdiel medzi odoslaním a prijatím správy.



Obrázok 3.3: Spojenia medzi bránami modulov

- Chybovosť (bit error) – pravdepodobnosť, že správa nie je správne doručená.
- Rýchlosť (data rate) – počet bitov prenesených za jednu sekundu.

Konfigurácia modelu

Jednotlivým modulom je možné priraďovať parametre, ktoré umožňujú ovplyvňovať ich správanie. Parametre môžu byť zakomponované priamo v definícii modulu (súbore NED), alebo načítavané pri štarte simulácie z konfiguračného súboru (*omnetpp.ini*).

Parametre môžu byť rôzneho dátového typu: reťazce, celé čísla , logické hodnoty, ale aj XML súbory. V rámci číselných parametrov je možné zavolať funkcie jazyka C pre generovanie pseudonáhodných čísel s rôznym rozložením, alebo interaktívne získať vstup od užívateľa.

3.1.3 Inštalácia OMNeT++

OMNeT++ je voľne šíriteľný systém a môže byť používaný nielen na akademický výskum, ale aj na komerčné účely. Podlieha podobným licenčným podmienkam ako GNU GPL produkty. Teda umožňuje spúšťať program za akýmkoľvek účelom, študovať ako program funguje a meniť ho, šíriť ďalej kópie, vylepšovať program a zverejňovať vylepšenia.

OMNeT++ nie je viazaný na jednu platformu. Existuje verzia ako na Windows, tak aj na Unixové operačné systémy. V dobe písania tejto práce bol aktuálny prechod z verzie 3.3 na verziu 4.0. Z dôvodu, aby boli vytvorené moduly kompatibilné aj s budúcimi verziami OMNeT++, sme sa v rámci projektu ANSA rozhodli používať verziu 4.0 (konkrétne 4.0rc1). Osobne som používal platformu Windows XP, pre ktorú je k dispozícii archív so všetkými zdrojovými súbormi, knižnicami potrebnými pre preklad a samotným prekladačom jazyka C++. Po stiahnutí a rozbalení archívu je postup inštalácie priamočiary a nemali by nastať žiadne komplikácie:

1. Spustiť príkazový riadok pomocou *mingwenv.cmd* z hlavného adresára zdrojových súborov.
2. Z príkazového riadku zadať príkaz *./configure*, ktorý skontroluje dostupnosť závislých knižníc a vytvorí konfiguračný súbor pre preklad.
3. Spustiť preklad zdrojových súborov pomocou príkazu *make*.
4. Do systémovej premennej *PATH* pridať cestu k vzniknutým binárnym súborom OMNET++.

Korektnosť inštalácie je možné otestovať spustením ukážkových simulácii z adresára *samples/dyna*.

Všetky potrebné súbory k inštalácii je možné stiahnuť na webovej stránke `http://www.omnetpp.org` .

3.2 INET Framework

INET Framework je balík simulačných modelov pre rodinu protokolov TCP/IP. Konkrétne obsahuje implementáciu protokolov IPv4, IPv6, TCP, UDP a niekoľko ďalších aplikačných protokolov. Framework taktiež obsahuje model MPLS a modely linkovej vrstvy PPP, Ethernet a 802.11. Statické smerovanie je možné nastaviť pomocou modulu autokonfigurácie, alebo je možné použiť implementáciu jedného zo smerovacích protokolov. INET Framework má podporu pre bezdrôtové a mobilné simulácie. Táto podpora bola prevzatá z Mobility Frameworku.

Predchodca INET Framework bol vyvíjaný na univerzite v Karlsruhe v rokoch 2000 a 2001 pod názvom IPSuite. V roku 2003 prevzal vývoj Andras Varga, ktorý jednotlivé moduly zreorganizoval, zdokumentoval, a mnohé od základu prepísal. V roku 2004 boli pridané rozšírenia TCP implementácie a pôvodný názov sa zmenil na INET Framework. V tomto roku bol taktiež vytvorený model OSPFv2, ktorý vytvoril v rámci svojej diplomovej práce Andras Babos. Vývoj prebieha aj v súčasnosti a hlavné zameranie patrí protokolom 802.11 a IPv6.

3.2.1 Architektúra INET Framework

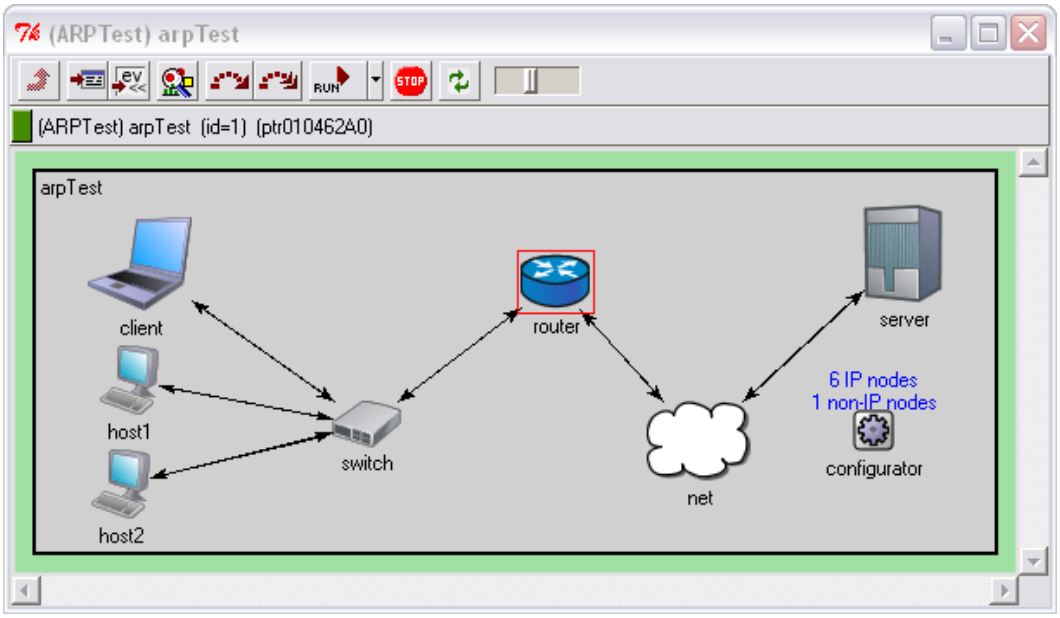
INET Framework je postavený nad OMNET++ a používa rovnaký koncept – moduly komunikujúce zasielaním správ. Jednotlivé protokoly sú reprezentované jednoduchými modulmi. Rozhrania jednoduchých modulov sú popísané v NED súboroch a implementácia v súboroch s C++ kódom. Tieto jednoduché moduly môžu byť voľne kombinované a spájané do zložitých modulov pomocou jazyka NED. Veľa takýchto modulov je už vytvorených, napríklad StandardHost, Router, OSPFRouter, WirelessAP. Samozrejme každý si môže vytvoriť vlastný modul šitý na mieru pre svoju simuláciu. Hlavičky protokolov a formát paketov je popísaný v MSG súboroch (message definition files), ktoré sú preložené do C++ tried pomocou OMNeT++ *opp_msgc* nástroja.

Kompiláciou INET Framework vznikne binárny súbor (*bin/INET.exe* pre platformu Windows a *bin/INET* pre unixové systémy), na ktorom je založený princíp používania frameworku. Program vznikne kompiláciou všetkých jednoduchých modulov a ich prilinkovaním k simulátoru OMNeT++ (ten je pre preklad nutný). Pri spúšťaní programu sa z aktuálneho adresára načíta obsah konfiguračného súboru *omnetpp.ini*. Jedným z parametrov uložených v súbore je odkaz na hlavný NED súbor, ten následne obsahuje odkazy na ďalšie NED definície modulov, z ktorých je modul zložený.

3.2.2 Simulovanie v INET Framework

Samotné vytváranie konkrétnej simulácie má základnú ideu stále rovnakú. Už implementované moduly je potrebné ako stavebnicu poskladať do jedného celku, ktorý sa bude navonok tváriť ako modelovaný systém. Detaily sú však skoro stále odlišne. Modul od modulu má iné parametre a na ich nastavení stojí správanie celej simulácie.

V tejto časti bude na jednoduchej ukážke ARP testu demonštrovaný základný princíp vytvárania a sledovania simulácie. Uvedený príklad je súčasťou demonštračných simulácii INET Framework v zložke *Examples/Ethernet/ARPTest*.



Obrázok 3.4: Topológia simulácie ARPTest

Vytváranie simulácie

Základom simulácie je súbor NED, ktorý definuje simulačný model. Určuje, ako sú jednotlivé moduly poprepájané, a môže obsahovať definíciu parametrov.
Komentovaná ukážka súboru *ARPTest.ned*:

```
//
// importovanie použitých modulov
//
import inet.networklayer.autorouting.FlatNetworkConfigurator;
import inet.nodes.ethernet.EtherSwitch;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import ned.DatarateChannel;
//
// vytvorenie modulu simulácie
//
network ARPTest
{
//
//definovanie vlastnosti linky
//
    types:
        channel fiberline extends DatarateChannel {
            delay = 1us;
            datarate = 512Mbps; }
//
//
```

```
// deklarácia podmodulov a ich parametrov
// sú vytvorené 4 moduly PC (StandardHost), 1 switch (EtherSwitch), 1 router (Router)
//  modul FlatNetworkConfigurator slúžiaci na konfiguráciu IP adries
//
    submodules:
        client: StandardHost {
            @display("p=71,64;i=device/laptop_l");}
        host1: StandardHost {
            @display("p=65,131;i=device/pc"); }
        host2: StandardHost {
            @display("p=60,191;i=device/pc"); }
        switch: EtherSwitch {
            @display("p=202,156"); }
        net: Router {
            @display("p=394,166"); }
        router: Router {
            @display("p=311,74"); }
        server: StandardHost {
            @display("p=512,58;i=device/server_l"); }
        configurator: FlatNetworkConfigurator {
            @display("p=495,160"); }

    connections:
//
// prepojenie jednotlivých modulov
//
        client.ethg++ <--> switch.ethg++;
        switch.ethg++ <--> host1.ethg++;
        switch.ethg++ <--> host2.ethg++;
        router.ethg++ <--> switch.ethg++;
        router.pppg++ <--> fiberline <--> net.pppg++;
        server.pppg++ <--> fiberline <--> net.pppg++;
}
```

Ďalším súborom, potrebným pre spustenie simulácie, je konfiguračný súbor. V súbore je možné priradiť parametre jednotlivým modulom (jednoduchým aj zloženým). Taktiež umožňuje nastavovať správanie prostredia simulácie. Implicitne sa súbor volá *omnetpp.ini*, ale je možné vytvoriť aj súbor s iným názvom, a potom spúšťať simuláciu s parametrom *–f <názov súboru>*.
Skrátená komentovaná ukážka súboru *omnetpp.ini*:

```
#
# obecné nastavenie pre všetky rozhrania a behy
#
[General]
  network = arpTest   # meno požitého systémového modulu
  preload-ned-files = *.ned @../../../nedfiles.lst  # dynamicky načítané moduly

  warnings = yes   # povolenie výstrah
  sim-time-limit = 500s   # maximálný simlačný čas
  cpu-time-limit= 600s   # maximálny strojový čas

#
# nastavenie grafického rozhrania Tkenv
#
[Tkenv]
  plugin-path=../../../Etc/plugins  # cesta k zásuvným modulom
  default-run=1  # implicitný beh
```

```
#
# obsahuje nastavenie parametrov pre prvý beh
# je možné definovať viacej behov, výber behu je realizovaný pomocou dialógu v Tkenv
#
[Run 1]
# parametre modulu udpApp
# ** zodpovedá žiadnemu alebo niekoľkým znakom (spôsob pridelenia parametrov viacerým
#  modulom)
  **.numUdpApps=0
  **.udpAppType="UDPBasicApp"

# parametre modulu tcp
# pre výklad jednotlivých parametrov je najlepšie nahliadnuť do dokumentácie INET
  **.tcp.mss = 1024
  **.tcp.advertisedWindow = 14336  # 14*mss
  **.tcp.sendQueueClass="TCPVirtualDataSendQueue"
  **.tcp.receiveQueueClass="TCPVirtualDataRcvQueue"
  **.tcp.tcpAlgorithmClass="TCPReno"
  **.tcp.recordStats=true
```

Pre spúšťanie simulácie je dobré si vytvoriť dávkový súbor BAT, ktorý zariadi korektné načítanie modulov a parametrov. Tento súbor nie je povinný.
Ukážka súboru *ARPTest.bat*:

```
..\..\..\bin\INET %*
```

Priebeh simulácie

Simuláciu je možné v grafickom prostredí spustiť v rôznych režimoch, ktoré sa líšia rýchlosťou spracovania udalostí. Na výber sú režimy "Step", "Run", "Fast", "Expres" a "Run until".

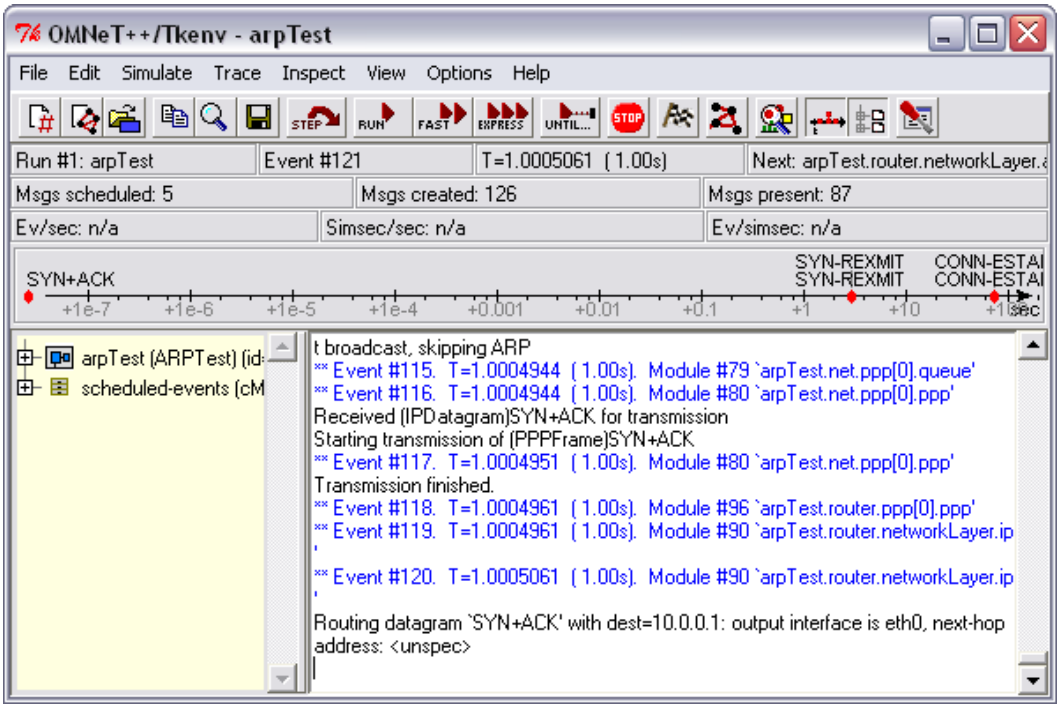
V priebehu simulácie sú do hlavného okna zapisované informácie o vykonávaných udalostiach a ich výstupy (pokiaľ sa nejaké generujú). Je možné si zobraziť kalendár udalostí a prezrieť si, ako sú jednotlivé udalosti naplánované. Pre každý modul, či už jednoduchý alebo zložený, je možné zobraziť tzv. aktívny inšpektor, ktorý pre zložený modul graficky zobrazuje jeho kompozíciu a pre jednoduchý modul zobrazuje hodnoty jednotlivých premenných alebo výstup modulu.

Simulácia končí, keď sú spracované všetky udalosti, alebo je dosiahnutý maximálny časový limit simulácie (pokiaľ bol nastavený v súbore *omnetpp.ini*). Pre reštart simulácie slúži voľba "Rebuilt network" z menu "Simulation".

Výsledky simulácie je možné zaznamenávať do súborov SCA a VEC, z ktorých sa neskôr dajú vytvárať grafy.

Napriek tomu, že OMNeT++ je silným simulačným nástrojom, nemá zabudovanú implicitnú podporu pre dynamické skriptovanie priebehu simulácie (zmenu parametrov simulačného systému v naplánovanom čase). Ako riešenie je možné použiť ScenarioManager - modul implementovaný v INET Framework. Bohužiaľ, v aktuálnej verzii ScenarioManager spolupracuje len s modulmi pre simuláciu bezdrôtových sietí, a pre simuláciu smerovacích protokolov je potrebné danú funkcionalitu doplniť.

ScenarioManager, z hľadiska svojej funkcie načíta príkazy, ktoré má vykonať zo vstupného XML súboru. Každý príkaz má definovanú časovú známku, v ktorom okamihu simulačného času sa má daný príkaz vykonať. Príkazy môžu meniť hodnoty parametrov, meniť atribúty spojenia (napr. chybovosť), pridávať alebo rušiť spojenia, či dokonca posielať príkazy modulom. Samozrejme, patričný modul musí daný príkaz podporovať.



Obrázok 3.5: Hlavné okno simulácie

Aby bol model skiptovateľný, vyžaduje to nasledovné kroky:

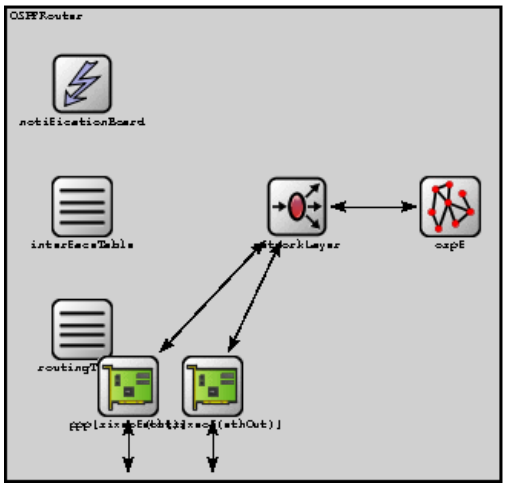
- Pridanie modulu ScenerioManger do sieťoveho modelu (na najvyššiu uroveň).
- Vytvorenie súboru *scenario.xml* s definíciou príkazov a ich časovou známkou.
- Rozšírenie modulov, ktoré nepodporujú ScenerioManger, aby vedeli reagovať na zmeny.

Pre rozšírenie modulov je potrebné v C++ triede modulu implementovať rozhranie IScriptable a definovať jeho metódu *processCommand()*. Aby modul vedel reagovať na zmenu parametrov, je potrebné predefinovať metódu *handleParameterChange()* triedy *cSimpleModule*.

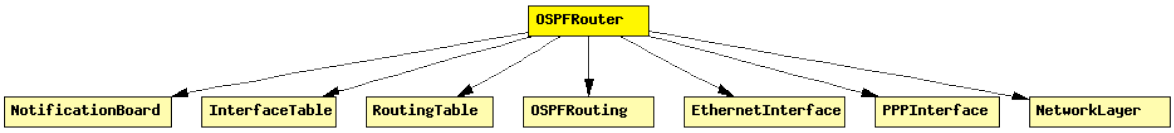
3.2.3 Modul OSPFRouter

Táto práca je zameraná na modelovanie a simuláciu OSPF protokolu, preto ako najdôležitejší modul pre potreby tejto práce je OSPFRouter. Ide o modul, ktorý reprezentuje reálny smerovač s implementovaným OSPF smerovaním. Jeho definícia sa nachádza v súbore *Nodes/INET/OSPFRouter.ned*. Samotný modul neimplementuje žiadne správanie kódom v C++, ale iba zastrešuje ostatné moduly nižšej úrovne a vytvára funkčný celok. Vnútorná štruktúra je zobrazená na obrázku 3.6.

Obrázok 3.7 znázorňuje diagram vzťahov medzi modulmi tvoriacimi OSPFRouter. Následne budú jednotlivé moduly popísané.



Obrázok 3.6: Vnútorná štruktúra modulu OSPFRouter



Obrázok 3.7: Vzťahy medzi modulmi tvoriacimi OSPFRouter

NetworkLayer

Je to opäť zložený modul, ktorý reprezentuje sieťovú vrstvu. Jeho vnútorná štruktúra je zobrazená na obrázku 3.8.

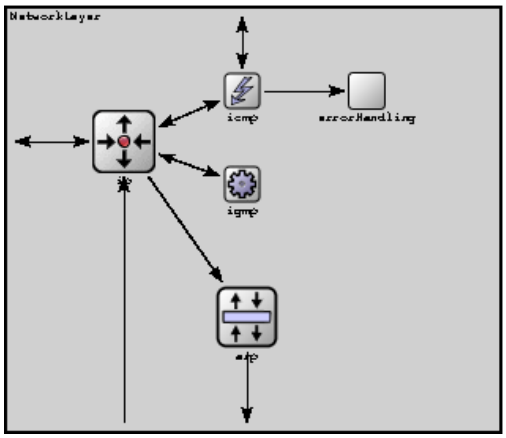
Modul sa skladá z nasledujúcich jednoduchých modulov:

Modul IP – implementuje IP protokol. Hlavička protokolu je definovaná v IPDatagram.msg. Umožňuje komunikáciu s viacerými protokolmi vyššej vrstvy. Protokol, ktorému bude paket predaný, určuje položka "Protocol" v hlavičke paketu. Mapovanie čísla protokolu na výstupnú bránu je definované v reťazci protocolMapping. Výber rozhrania linkovej vrstvy, na ktoré má byť paket predný, je realizovaný pomocou modulu RoutingTable. Keď IP modul potrebuje zistiť rozhranie, jednoducho zavolá funkciu modulu RoutingTable *findBestMatchingRoute(destAddress)*, ktorá mu na základe cieľovej adresy vráti identifikátor rozhrania.

ARP – implementuje ARP (Address Resolution Protocol) pre IPv4 a IEEE 802 6-bajtové MAC adresy. Používa sa pre zisťovanie MAC adresy z IP adresy na broadcastových rozhraniach, napr. Ethernetových. Či je rozhranie broadcastové, je uvedené v zázname daného rozhrania v module InterfaceTable (je možné testovať funkciou *isBroadcast()*). Pre PPP rozhrania nemá ARP zmysel.

ICMP – implementuje protokol ICMP (Internet Control Message Protocol). Prijíma, spracuváva a odosiela odpovede správ ICMP echo a ICMP timestamp. Ignoruje správy ICMP destination unreachable, ICMP time exceeded, ICMP parameter problem a ICMP redirect.

IGMP – je modul vyhradený pre protokol IGMP, ktorý slúži na prihlasovanie do multicastových skupín. Zatiaľ však nie je implementovaný.



Obrázok 3.8: Vnútorná štruktúra modulu NetworkLayer

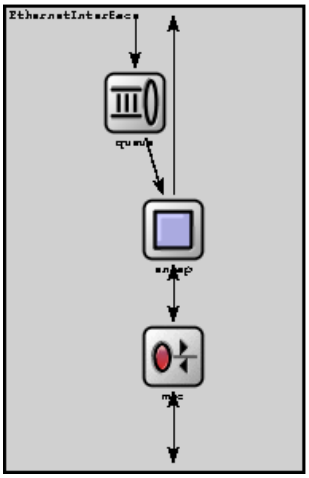
ErrorHandling – spracuváva správy o chybách, ktoré prídu od ostatných protokolov. Terajšia implementácie iba vypíše oznámenie na výstup a správu zruší.

OSPFRouting

Implementuje smerovací protokol OSPF. Tento modul bude podrobne zanalyzovaný v kapitole 4.

EthernetInterface

Zložený modul, ktorý predstavuje ethernetové sieťové rozhranie. Jeho vnútorná štuktúra je zobrazená na obrázku 3.9.



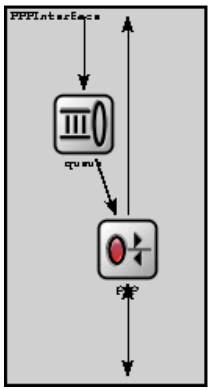
Obrázok 3.9: Vnútorná štruktúra modulu EthernetInterface

Modul sa skladá z nasledujúcich jednoduchých modulov:
EtherMAC – implementuje vrstvu MAC, ktorá realizuje vysielanie a príjem rámcov. Modul nerealizuje zapuzdrovanie, o túto funkciu sa starajú moduly EtherLLC alebo EtherEn-

cap. Podporované sú rôzne verzie 10Mb Ethernet, 100Mb Ethernet half/full duplex a 1Gb Ethernet. Počas spracuvávania rámcov prijatých od vyšších vrstiev sú doplnené zdrojové adresy, pokiaľ chýbajú, následne je rámec zaradený do fronty, kde ostáva dovtedy, kým nie je prenesený. Prenos je modelovaný podľa protokolu CSMA/CD. Pokiaľ požiada vyššia vrstva, tak modul môže odoslať rámec PAUSE (používanú v prepínačoch). Príjem rámcov zo siete taktiež prebieha protokolom CSMA/CD. Modul skontroluje CRC. V prípade, že nastane chyba, tak je rámec zahodený. V promiskuitnom móde spracuváva všetky prijaté rámce, inak iba rámce so zodpovedajúcou MAC adresou alebo broadcastové rámce. Na začiatku každej simulácie dochádza medzi jednotlivými modulmi k autokonfigurácii, počas ktorej dochádza k výberu prenosovej rýchlosti a duplexu. V súčasnej implementácii nie je možné počas simulácie dynamicky pripojiť alebo odpojiť MAC rozhranie.
EtherEncap – modul zapuzdruje pakety prichádzajúce od vyšších vrstiev do Ethernet II rámcov a posiela ich ďalej modulu MAC. Pre rámce prichádzajúce od MAC modulu je operácia opačná.
OutputQueue – implementuje výstupnú frontu sieťového rozhrania. Konkrétne môže byť použitá fronta typu drop-tail, RED (Random Early Detection) alebo policy.

PPPInterface

Zložený modul, funkcionálne veľmi podobný modulu EthernetInterface. V tomto prípade však ide o modelovanie dvojbodového spoja, nie zdieľaného segmentu, ako je to u EthernetInterface. Obrázok 3.10 zobrazuje vnútornú štruktúru PPPInterface.



Obrázok 3.10: Vnútorná štruktúra modulu PPPInterface

Modul sa skladá z nasledujúcich jednoduchých modulov:
PPP – implementuje protokol PPP. PPP je komplexný protokol so silnou podporou konfigurácie linky a jej údržby. Tento modul všetky tieto detaily ignoruje a jedinou jeho funkciou je zapuzdrovať rámce. Rámce po zapuzdrení posiela na výstupnú frontu. Prijímané rámce nie sú zaraďované do žiadnej fronty, ale sú okamžite spracované.
OutputQueue – je modul totožný s modulom používaným v EthernetInterface.

InterfaceTable

Modul udržuje záznamy o jednotlivých sieťových rozhraniach. Rozhrania sú dynamicky registrované príslušnými L2 modulmi (napr. PPPInterface). Okrem toho sa automaticky

vytvorí rozhranie loopback. Tabuľka rozhraní obsahuje iba na protokole nezávislé informácie. Všetky IPv4 alebo IPv6 špecifické informácie sú udržiavané v moduloch RoutingTable, respektíve RoutingTable6.

Tento modul nemá žiadne vstupné ani výstupné brány. Celá funkcionalita je prístupná cez členské funkcie C++ triedy.

RoutingTable

Modul udržiava smerovaciu tabuľku. Pre smerovače musí byť nastavený parameter routerId. Vo väčšine prípadov je ako routerId nastavená IP adresa rozhrania Loopback. Je možné zvoliť aj automatický výber najvyššej IP adresy aktívneho rozhrania.

Smerovacia tabuľka je načítavaná zo súboru (parameter *routingFile*). Následne môže byť menená dynamickými smerovacími protokolmi. Formát súboru zodpovedá unixovým výpisom rozhraní *ifconfig* a smerovacej tabuľky *netstat -rn*. Podrobnejší popis je možné nájsť v dokumentácii INET Framework.

Tento modul nemá žiadne vstupné ani výstupné brány. Celá funkcionalita je prístupná cez členské funkcie C++ triedy.

NotificationBoard

Modul NotificationBoard umožňuje informovať ostané moduly o udalostiach (zmenách), ktoré nastali v simulačnom modeli. Udalosti sa môžu týkať zmien v smerovacej tabuľke, zmien konfigurácie rozhraní, zmien kanálu bezdrôtovej siete či pozície uzlu.

Modul je prístupný priamo cez volanie C++ metód (neprebieha výmena správ). Jednotlivé moduly môžu reagovať na zvolený typ zmeny. Keď daná zmena nastane, patričný modul informuje NotificationBoard, a ten rozdistribuuje informáciu všetkým ostatným zainteresovaným modulom.

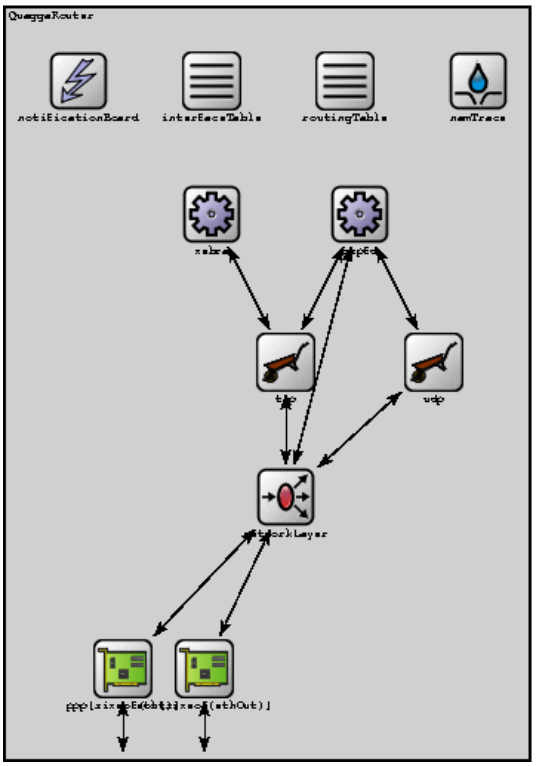
3.2.4 Modul QuaggaRouter

QuaggaRouter je ďalším zloženým modulom, ktorý umožňuje simulovať smerovací protokol OSPF. Jeho vnútorná štruktúra zobrazená na obrázku 3.11 je náramne podobná modulu OSPFRouter.

Presne ako OSPFRouter, aj QuaggaRouter obsahuje moduly NetworkLayer, EthernetInterface, PPPInterface, InterfaceTable, RoutingTable a NotificationBoard. Rozdiel je však v samotnom module OSPF procesu. Namiesto modulu OSPFRouting, QuaggaRouter využíva emulovaný prístup, kde simulovanie OSPF je nahradené použitím systémového procesu unixovej implementácie smerovacích protokolov Quagga.

Quagga je softwarový balík smerovacích protokolov. Vo forme démonov implementuje OSPFv2 (ospfd), RIP (ripd), OSPFv3 (ospf6d), RIPng (ripngd) a BGPv4+ (bgpd). Architektúra je založená na hlavnom démonovi Zebra. Zebra vytvára abstrakčnú vrstvu medzi sieťovým rozhraním a jednotlivými démonmi smerovacích protokolov, ktoré v rámci komunikácie vystupujú ako klienti.

QuaggaRouter, pre potreby simulovania smerovacích protokolov, využíva upravenú verziu balíčka Quagga. Pomocou modulu Daemon dokážu komponenty INET Framework priamo komunikovať s Quagga procesmi. Daemon vytvára abstraktnú vrstvu, ktorá zachytáva a spracuváva systémové volania procesov Quagga. Zachytávanie všetkých systémových



Obrázok 3.11: Vnútorná štruktúra modulu QuaggaRouter

volaní je jednoducho realizované modifikáciu zdrojových súborov Quagga, kde volania systémových rutín sú nahradené volaniami rutín implementovaných v module Daemon (napr. volanie *socket()* je nahradené *oppsim_socket()*).

Tento emulovaný prístup má výhodu v tom, že ide o funkčnú implementáciu, ktorá dokáže komunikovať s reálnymi zariadeniami, a má podporu radu pokročilých funkcií. Nevýhodou sú zvýšené hardwarové nároky, najmä na pamäť a procesor počítača. Keďže nejde o klasický simulovaný prístup, tak vo zvyšku práce modulu QuaggaRouter už nebude venovaná pozornosť.

3.2.5 Inštalácia INET Framework

S príchodom OMNeT++ verzie 4.0 sa aj inštalácia INET Framework podstatne zjednodušila. Keďže prekladač zdrojových kódov C++ je pre platformu Windows už súčasťou distribúcie, nie potrebné inštalovať externý prekladač. Podstatne sa tým redukujú problémy, ktoré môžu vznikať pri nekompatibilite prekladačov. Postup inštalácie pre INET Framework verzie INETMANET-20080920 je nasledovný:

1. Prerekvizitou je mať správne nainštalovaný OMNeT++ (viď sekcia 3.1.3). Systémová premenná *PATH* musí obsahovať cestu k binárnym súborom OMNeT++ (overiť je to možné zadaním príkazu *nedtool*).

2. Spustiť príkazový riadok pomocou *mingwenv.cmd* z hlavného adresára inštalácie OMNeT++.

3. Za pomoci príkazu *cd* sa v príkazovom riadku prepnúť do zložky s rozbalenými zdrojovými súbormi INET Framework.

4. Spustiť preklad zdrojových súborov pomocou príkazu *make*.

Po preložení je možné inštaláciu otestovať spustením demonštračnej simulácie *rundemo.bat* zo zložky *examples*.

Pri preklade sa automaticky prekladajú všetky zdrojové súbory zo zložky *src*. V rámci projektu ANSA sme v tejto zložke vytvorili vlastnú podzložku *ansa*, do ktorej sú zhromažďované všetky nové alebo upravené moduly. Ich preklad je vykonaný automaticky, bez nutnosti meniť parametre prekladu v súbore *makefile*.

Kapitola 4

INET Framework a OSPF

Kapitola podrobne analyzuje modul OSPFRouting. V prvej časti sú predstavené zdrojové súbory, triedy a funkcie modulu. Následne je vyhodnotená správnosť výberu cesty, ktorá potvrdzuje korektnosť implementácie SPF algoritmu. V ďalšej časti sú na module vykonané výkonnostné testy. Tie majú zhodnotiť implementáciu z hľadiska korektnosti času simulácie k reálnemu prostrediu. A nakoniec sú zosumarizované implementované vlastnosti protokolu OSPF a funkcionalita, ktorá chýba.

4.1 Implementácia OSPFRouting

OSPFRouting je modul v INET Framework, ktorý umožňuje simuláciu dynamického smerovacieho protokolu OSPF. Všetky zdrojové súbory sa nachádzajú v zložke src/networklayer/-ospfv2. Samotná implementácia je pomerne komplikovaná a rozsiahla (cca. 10000 riadkov kódu), preto sú zdrojové súbory organizované do podzložiek podľa funkcie, ktorú implementujú. V nasledujúcom texte bude predstavený obsah a funkcie jednotlivých súborov.

4.1.1 Hlavná zložka

Zložka obsahuje súbory *OSPFRouting.ned*, *OSPFRouting.cc*, *OSPFRouting.h*, *OSPFPacket.msg*, *OSPFTimer.msg*. Súbory v tejto zložke implementujú základné rozhranie modulu, načítavanie konfigurácie, definíciu správ a časovačov.

OSPFRouting.ned

Obsahuje definíciu jednoduchého modulu OSPFRouting. Modul má jednu vstupnú a jednu výstupnú bránu. Tieto sa pripájajú na modul NetworkLayer a slúžia zasielaniu a prijímaniu OSPF paketov. Modul má jeden parameter typu reťazec nazvaný *ospfConfigFile*, pomocou ktorého je modulu predávaný názov konfiguračného súboru.

OSPFRouting.cc, OSPFRouting.h

Obsahom súborov je C++ definícia jednoduchého modulu ako triedy *OSPFRouting*, odvodenej od triedy *cSimpleModule*. Trieda *OSPFRouting* obsahuje objekt *OSPF::Router*, ktorý je koreňovým objektom celej dátovej štruktúry OSPF procesu. Jeho implementácia bude predstavená v časti 4.1.2. Ďalej si trieda uchováva ukazovateľ na tabuľku rozhraní (objekt *IInterfaceTable*) a smerovaciu tabuľku (objekt *IRoutingTable*) smerovača. Z obsiahnutých

pomocných funkcií je najdôležitejšou funkciu *LoadConfigFromXML()*, ktorá slúži k načítaniu konfigurácie zo XML súboru. Funkcia na začiatku otvorí súbor s XML konfiguráciou, podľa parametra *ospfConfigFile*. V prípade úspechu, podľa identifikácie RouterID, získa XML element s konfiguráciu určenou pre daný smerovač. Následne sú načítané informácie o oblastiach, do ktorých smerovač patrí (funkciou *LoadAreaFromXML()*) a informácie o jednotlivých rozhraniach smerovača (funkciou *LoadInterfaceParameters ()*). Všetky informácie sú vkladané do objektu *OSPF::Router*. Po úspešnom načítaní je spustený OSPF proces (funkciou *ProcessEvent(OSPF::Interface::InterfaceUp)* na každom OSPF rozhraní).

OSPFPacket.msg

Súbor obsahuje definíciu jednotlivých typov OSPF paketov. Je písaný v internom jazyku OMNeT++ pre definíciu zasielaných správ. Tento jazyk je automaticky prekladaný do C++ reprezentácie.

Definícia zodpovedá štandardu popísanému v RFC 2328. Je definovaných 5 typov OSPF paketov:

- OSPFHelloPacket
- OSPFDatabaseDescriptionPacket
- OSPFLinkStateRequestPacket
- OSPFLinkStateUpdatePacket
- OSPFLinkStateAcknowledgementPacket

V rámci OSPFLinkStateUpdatePacket je definovaných 5 typov LSA, čo tiež zodpovedá štandardu RFC 2328:

- OSPFRouterLSA
- OSPFNetworkLSA
- OSPFSummaryLSA (3. a 4. typ rozlišuje sa na základe obsahu)
- OSPFASExternalLSA

Z hľadiska rozsahu nie je možné uviesť definíciu všetkých typov paketov. Na ukážku bude uvedená definícia hlavičky OSPF paketu, spoločná pre všetky OSPF pakety:

```
packet OSPFPacket
{
    char version = 2;
    char type enum(OSPFPacketType) = HelloPacket;
    short packetLength = 0;

    IPAddress routerID;
    IPAddress areaID;

    short checksum = 0;
    short authenticationType = 0;
    char authentication[8];
}
```

OSPFTimer.msg

Súbor obsahuje definíciu časovačov potrebných k správnej komunikácii v rámci OSPF procesu. Časovače sú vytvárané vo forme časovaných správ zasielaných samému sebe. Keďže ide o správy, tak sú popísané rovnakým jazykom ako OSPF pakety. OSPF pre svoju funkciu potrebuje nasledovné časovače:

- InterfaceHelloTimer – pravidelné zasielanie Hello paketov v danom intervale.
- InterfaceWaitTimer – čas, počas ktorého sa rozhranie nachádza v stave Wait (viď 4.1.3).
- InterfaceAcknowledgementTimer – potvrdenie prijatia OSPF paketu je zasielané s určitým oneskorením, ktoré sa odpočítava práve týmto časovačom.
- NeighborInactivityTimer – čas za ktorý sa rozviaže susedstvo v prípade, že od suseda nebol v danom intervale prijatý žiadny OSPF paket.
- NeighborDDRetransmissionTimer – časový limit, počas ktorého musí byť prijaté potvrdenie na odoslaný OSPFDatabaseDescriptionPacket.
- NeighborUpdateRetransmissionTimer – časový limit, počas ktorého musí byť prijaté potvrdenie na odoslaný OSPFDatabaseDescriptionPacket.
- NeighborRequestRetransmissionTimer – časový limit, počas ktorého musí byť prijaté potvrdenie na odoslaný OSPFLinkStateRequestPacket.
- DatabaseAgeTimer – časovač, ktorý každú sekundu zvyšuje vek (age) jednotlivým LSA.

4.1.2 Zložka router

Zložka obsahuje súbory *OSPFRouter.cc*, *OSPFRouter.h*, *OSPFArea.cc*, *OSPFArea.h*, *LSA.h*, *OSPFcommon.h*, *OSPFRoutingTableEntry.h*, *RouterLSA.cc*, *NetworkLSA.cc*, *SummaryLSA.cc*, *ASExternalLSA.cc*. Súbory v tejto zložke implementujú podstatnú funkcionalitu OSPF procesu, ako napríklad inštaláciu LSA do databázy, zostavovanie SPF(short path first) stromu, či inštaláciu ciest do smerovacej tabuľky.

OSPFRouter.cc, OSPFRouter.h

Obsahom súborov je implementácia triedy *OSPF::Router*, ktorá v sebe udržiava kompletné informácie potrebné pre funkciu OSPF procesu na danom smerovači. Trieda obsahuje zoznam objektov reprezentujúcich oblasti (*OSPF::Area*), ktorých je smerovač súčasťou. Ďalej obsahuje zoznam všetkých externých LSA a záznamy OSPF smerovacej tabuľky (používa ju len OSPF proces pre vlastné výpočty, smerovač ma vlastnú globálnu smerovaciu tabuľku).

V princípe trieda implementuje funkcie, ktoré buď pracujú s jednotlivými oblasťami (volajú funkcie triedy *OSPF::Area* pre danú oblasť), alebo implementujú nejakú globálnu funkciu, ktorá má vplyv na celý OSPF proces. Funkcie triedy *OSPF::Area* budú predstavené v nasledujúcej časti, preto budú teraz uvedené len funkcie globálneho charakteru:

- *InstallASExternalLSA()* – vloží ASExternal LSA do databázy.

- *AgeDatabase()* – u ASExternal LSA zvýši vek (age) LSA. V prípade, že je dosiahnutá maximálna hodnota, musí byť vykonaná rutina, ktorá buď dané LSA zahodí, alebo vytvorí nové LSA a rozpošle ho (v prípade, že daný smerovač je pôvodcom LSA).
- *OriginateASExternalLSA()* – vytvorí OSPF paket obsahujúci ASExternal LSA.
- *RebuildRoutingTable()* – vytvorí novú OSPF internú smerovaciu tabuľku na základe SPF stromu.
- *CalculateASExternalRoutes()* – na základe ASExtrenal LSA z databázy vypočíta cesty k externým sieťam.
- *NotifyAboutRoutingTableChanges()* – v prípade zmeny v smerovacej tabuľke, vygeneruje nové alebo odstráni neaktuálne Summary LSA z databázy.

OSPFArea.cc, OSPFArea.h

V súboroch sa nachádza implementácia triedy OSPF::Area. Táto trieda reprezentuje oblasť (area) a obsahuje zoznamy rozhraní, Router LSA, Network LSA a Summary LSA, patriacich danej oblasti. Nad týmito informáciami implementuje potrebné funkcie:

- *AddInterface()* – pridá OSPF::Interface do danej oblasti.
- *InstallRouterLSA()* – vloží Router LSA do databázy.
- *InstallNetworkLSA()* – vloží Network LSA do databázy.
- *InstallSummaryLSA()* – vloží Summary LSA do databázy.
- *AgeDatabase()* - u Router, Network a Summary LSA zvýši vek (age) LSA V prípade, že je dosiahnutá maximálna hodnota, musí byť vykonaná rutina, ktorá buď dané LSA zahodí, alebo vytvorí nové LSA a rozpošle ho (v prípade, že daný smerovač je pôvodcom LSA).
- *OriginateRouterLSA()* – vytvorí OSPF paket obsahujúci Router LSA.
- *OriginateNetworkLSA()* – vytvorí OSPF paket obsahujúci Network LSA.
- *OriginateSummaryLSA()* – vytvorí OSPF paket obsahujúci Summary LSA.
- *CalculateShortestPathTree()* – na základe prijatých LSA vytvorí SPF strom.
- *CreateRoutingTableEntryFromSummaryLSA()* – vytvorí záznam do smerovacej tabuľky podľa Summary LSA.

OSPFcommon.h, OSPFRoutingTableEntry.h, LSA.h

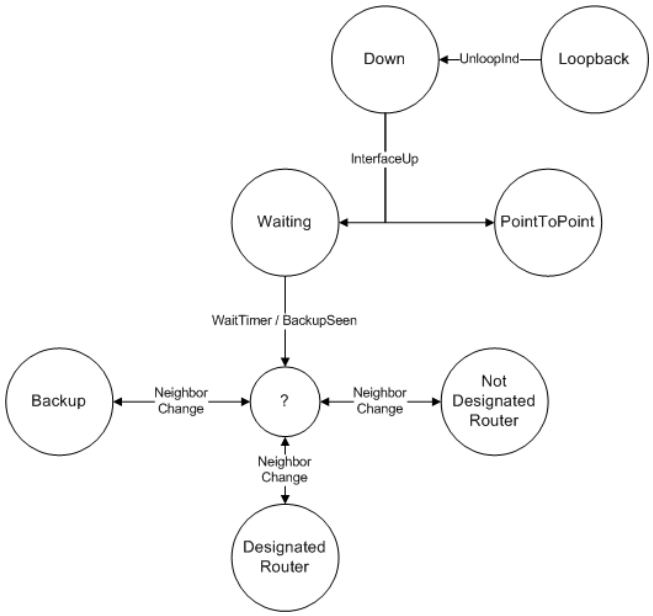
Definujú pomocné štruktúry pre reprezentáciu IP adries, rozsahov IP adries, identifikátorov oblastí, LSA a záznamov smerovacej tabuľke. Tiež implementujú operácie nad týmito štruktúrami, ako sú porovnávanie, maskovanie či prevod do inej reprezentácie.

RouterLSA.cc, NetworkLSA.cc, SummaryLSA.cc, ASExternalLSA.cc

Všetky 4 súbory implementujú rovnaké 2 funkcie. Rozdiel je len v tom, s akým typom LSA daná funkcia pracuje. Prvou z funkcií je DiffersFrom(), ktorá porovná prijaté LSA s LSA uloženým v databáze. Túto funkciu následne využíva druhá funkcia Update(). Jej úlohou je zistiť, či je po prijatí nového LSA potrebné aktualizovať databázu.

4.1.3 Zložka interface

Zložka obsahuje súbory *OSPFInterface.cc*, *OSPFInterface.h*, *OSPFInterfaceState.cc*, *OSPFInterfaceState.h* + 12 ďalších súborov s názvami v tvare *OSPFInterface<názov stavu>.cc* a *OSPFInterface<názov stavu>.h*. Hlavnou úlohou zdrojových súborov v tejto zložke je reprezentovať informácie o OSPF rozhraní a jeho stave (trieda *OSPF:: Interface*). Ďalšou nemenej dôležitou úlohou je implementovať prechod medzi jednotlivými stavmi rozhrania. Prechod medzi stavmi sa riadi podľa stavového automatu zobrazeného na obrazku 4.1. Základnou funkciou implementujúcou tento stavový automat je funkcia *ProcessEvent()*, ktorá ako parameter prijíma udalosť na rozhraní. Na základe aktuálneho stavu a tejto udalosti, rozhodne o nasledujúcom stave a krokoch, ktoré treba pre prechod do nového stavu vykonať. Za udalosť na rozhraní sú považované, prijatie OSPF paketu, vypršanie časovača a signál od fyzického rozhrania o zmene stavu (Up / Down).

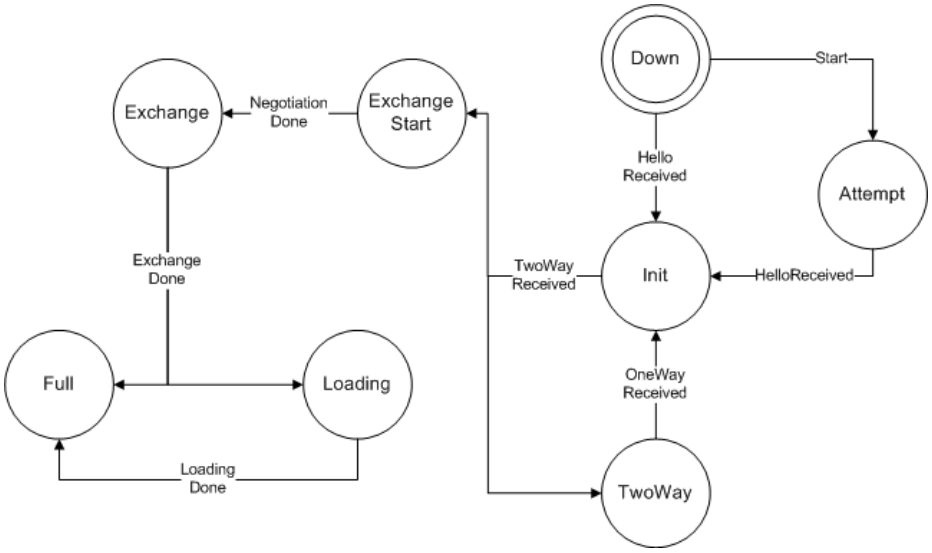


Obrázok 4.1: Diagram stavového automatu rozhrania

4.1.4 Zložka neighbor

Obsahom zložky sú súbory *OSPFNeighbor.cc*, *OSPFNeighbor.h*, *OSPFNeighborState.cc*, *OSPFNeighborState.h* + 16 ďalších súborov s názvami v tvare *OSPFNeighbor<názov stavu>.cc* a *OSPFNeighbor<názov stavu>.h*. Štruktúra, ako aj implementovaná funkcionalita, je veľmi podobná zložke interface. Na rozdiel od zložky interface, reprezentuje informácie a funkcie pre vytváranie susedstiev (trieda *OSPF::Neighbor*). Vytváranie susedstiev sa riadi

stavovým automatom podľa obrázku 4.2. Opäť je implementovaná funkcia *ProcessEvent()*, ktorá riadi prechod medzi jednotlivými stavmi.



Obrázok 4.2: Diagram stavového automatu susedstva

4.1.5 Zložka messagehandler

Súbory v zložke implementujú funkcionalitu spojenú s odosielaním a prijímaním OSPF paketov a následnú reakciu na prijatý OSPF paket. Každý zo súborov má na starosti spracovanie určitého typu OSPF paketu. Napríklad v súboroch *HelloHandler.cc* a *HelloHandler.h* je implementované spracovanie Hello paketu, a to vo funkcii *ProcessPacket()*. Na začiatku musí funkcia overiť, či informácie z Hello paketu, ako HelloInterval, DeadInterval a AreaId, sú zhodné s hodnotami nakonfigurovanými na rozhraní, cez ktoré bol Hello paket prijatý. V prípade, že nejaké hodnoty sú odlišné, je paket zahodený. V opačnom prípade sa zisťuje, v akom stave je susedstvo so smerovačom, od ktorého bol Hello paket prijatý. V prípade, že susedstvo vôbec neexistuje, musí sa vytvoriť nový objekt triedy *OSPF::Neighbor* a naštartovať procedúra vytvárania susedstva. Pokiaľ susedstvo je už vo fáze vytvárania, tak sa vykoná ďalší krok k jeho úspešnému vytvoreniu. Respektíve, keď je susedstvo už vytvorené, zresetuje sa len časovač NeighborInactivityTimer.

Podobne je implementované aj spracovanie ostatných OSPF paketov, samozrejme, s funkcionalitou závislou na jeho type.

4.2 Overenie správnosti implementácie modulu OSPFRouting

Úlohou smerovacích protokolov, teda aj protokolu OSPF, je vybrať najlepšie cesty pre konkrétne cieľové siete. Implementáciu modulu OSPFRouting môžeme prehlásiť za korektnú vtedy, keď pre danú cieľovú sieť s ohodnotenými cestami vyberie tú najlepšiu. Samozrejme, cesta musí byť nájdená v rozumnom čase. Skúmaniu časových a výkonnostných

veličín sa bude venovať podkapitola 4.3 . V tejto podkapitole bude overená správnosť výberu cesty. Overovanie prebehne porovnávaním obsahu smerovacej tabuľky na jednotlivých smerovačoch v simulácii so smerovacími tabuľkami reálnych smerovačov (v našom prípade smerovačov Cisco rady 2800). Implementáciu OSPF na smerovačoch Cisco je možné považovať za RFC 2328 kompatibilnú.

Pred samotným porovnávaním je potrebné poznamenať pár odlišností medzi správaním simulácie a hardwarovými smerovačmi:

- V prípade, že do cieľovej siete vedú dve alebo viaceré ciesty s rovnakou metrikou, v simulácii nie možné robiť tzv. vyvažovanie záťaže. Preto je pri konfigurácii hardwarových smerovačov použitý príkaz *maximum-paths 1*, ktorý zakáže vyvažovanie záťaže a spôsobí to, že smerovač vyberie iba jednu cestu pre každú cieľovú sieť.

- RFC 2328 nedefinuje, akým spôsobom vybrať jednu cestu v prípade, že viac ciest má rovnakú metriku. Záleží teda na konkrétnej implementácii, ako uskutoční tento výber. Preto sa môžu líšiť cesty, ktoré vyberie modul OSPFRouting od tých, ktoré vyberie reálny smerovač. No metriku musia mať vždy rovnakú.

- Súčasná implementácia smerovacej tabuľky v INET Framework nerozlišuje administratívnu vzdialenosť pri jednotlivých cestách. Preto cesty do sietí, ktoré sú priamo pripojené k danému smerovaču, sú v smerovacej tabuľke obsiahnuté dvakrát. Raz ako statická cesta rozhrania a druhýkrát ako dynamická cesta, ktorú do nej vložil modul OSPFRouting. Smerovacia tabuľka je predmetom rekonštrukcie inej študentskej práce.

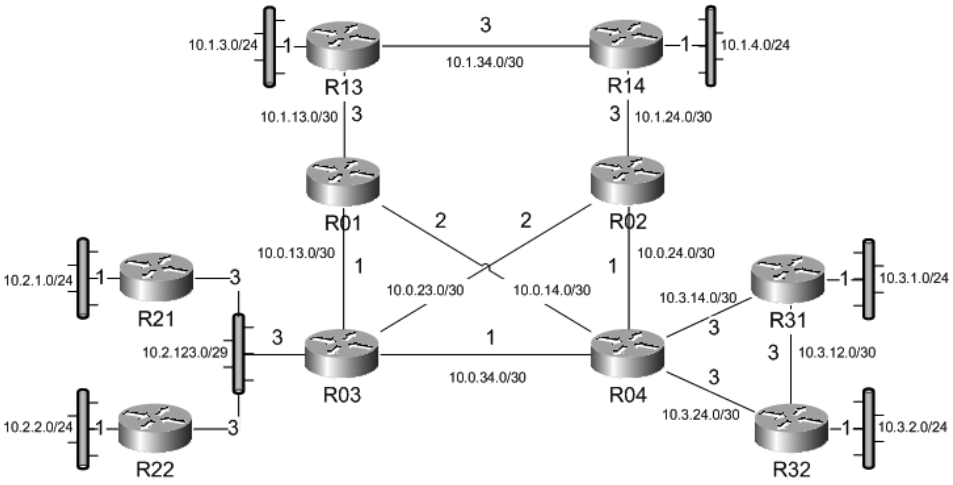
- RFC 2328 špecifikuje prenos masky siete point-to-point rozhrania dvoma spôsobmi. Buď je prenášaná skutočná maska nastavená na rozhraní, alebo sa implicitne nastaví maska 255.255.255.255. Cisco implementovalo prvú variantu. V module OSPFRouting bola zvolená druhá varianta. Pre vzájomnú zhodnosť obsahu oboch smerovacích tabuliek, bolo treba uskutočniť zmeny v module OSPFRouting. Aktuálne obe implementácie zhodne prenášajú masku nastavenú na rozhraní.

- Simulácia pre point-to-point spoje nevyužíva modul PPPInterface, ale emuluje point-to-point spoj na ethernetovom rozhraní. Dôvodom je jednoduchšie nastavovanie priepustnosti linky. Preto je v smerovacej tabuľke, ako odchádzajúce rozhranie uvedené ethernetové rozhranie (napr. eth0), i keď ide o point-to-point spojenie.

Topológia použitá pre samotné overenie obsahuje celkovo 10 smerovačov, 14 liniek a 6 stub sietí. Najvyšší stupeň smerovača je 5, najnižší 2. Všetky linky majú rovnakú prenosovú rýchlosť a rovnakú latenciu. Topológia s adresovou schémou a hodnotami OSPF Cost je znázornená na obrázku 4.3.

Na obrázkoch 4.4, 4.5 a 4.6 sú uvedené ukážky smerovacích tabuliek jednotlivých smerovačov. Z dôvodu udržania stanoveného rozsahu práce sú uvedené len smerovacie tabuľky smerovačov R02, R21 a R31. Pre korektné vyhodnotenie to však bude postačovať, lebo všetky smerovače v oblasti majú rovnakú databázu stavu linky.

Z ukážok smerovacích tabuliek je možné odpozorovať, že ako smerovacie tabuľky na reálnych smerovačoch, tak aj smerovacie tabuľky v simulátore obsahujú záznamy o rovnakých cieľových sieťach. Smerovacie tabuľky v simulátore obsahujú viacej záznamov, keďže pre niektoré cieľové siete majú v tabuľke dva záznamy. Je to spôsobené hore uvedenými odlišnosťami, resp. nedostatkami implementácie. Ďalším dôležitým porovnávacím parametrom



Obrázok 4.3: Topológia použitá pre overenie smerovacích tabuliek

Cisco Router

```
Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 18 subnets, 3 masks
O       10.0.14.0/30 [110/3] via 10.0.24.2, 01:11:21, Serial0/0/1
O       10.3.12.0/30 [110/7] via 10.0.24.2, 01:09:03, Serial0/0/1
O       10.1.13.0/30 [110/9] via 10.1.24.2, 01:44:30, Serial0/1/0
O       10.3.14.0/30 [110/4] via 10.0.24.2, 01:11:21, Serial0/0/1
O       10.0.13.0/30 [110/3] via 10.0.23.2, 01:11:21, Serial0/0/0
O       10.3.1.0/24 [110/5] via 10.0.24.2, 01:10:27, Serial0/0/1
O       10.1.3.0/24 [110/7] via 10.1.24.2, 01:44:31, Serial0/1/0
O       10.2.1.0/24 [110/6] via 10.0.23.2, 01:11:23, Serial0/0/0
O       10.2.2.0/24 [110/6] via 10.0.23.2, 01:11:23, Serial0/0/0
O       10.3.2.0/24 [110/5] via 10.0.24.2, 01:08:20, Serial0/0/1
O       10.1.4.0/24 [110/4] via 10.1.24.2, 01:44:31, Serial0/1/0
O       10.3.24.0/30 [110/4] via 10.0.24.2, 01:09:05, Serial0/0/1
C       10.0.24.0/30 is directly connected, Serial0/0/1
C       10.1.24.0/30 is directly connected, Serial0/1/0
C       10.0.23.0/30 is directly connected, Serial0/0/0
O       10.0.34.0/30 [110/2] via 10.0.24.2, 01:11:23, Serial0/0/1
O       10.1.34.0/30 [110/6] via 10.1.24.2, 01:44:31, Serial0/1/0
O       10.2.123.0/29 [110/5] via 10.0.23.2, 01:11:23, Serial0/0/0
```

INET OSPFRouting

```
[0] = dest:127.0.0.1  gw:*  mask:255.0.0.0  metric:1 if:lo0  DIRECT IFACENETMASK
[1] = dest:10.0.23.1  gw:*  mask:255.255.255.252  metric:2 if:eth0  DIRECT IFACENETMASK
[2] = dest:10.0.24.1  gw:*  mask:255.255.255.252  metric:1 if:eth1  DIRECT IFACENETMASK
[3] = dest:10.1.24.1  gw:*  mask:255.255.255.252  metric:3 if:eth2  DIRECT IFACENETMASK
[4] = dest:10.2.123.0  gw:*  mask:255.255.255.248  metric:5 if:eth0  DIRECT OSPF
[5] = dest:10.0.23.0  gw:*  mask:255.255.255.252  metric:2 if:eth0  DIRECT OSPF
[6] = dest:10.0.24.0  gw:*  mask:255.255.255.252  metric:1 if:eth1  DIRECT OSPF
[7] = dest:10.1.24.0  gw:*  mask:255.255.255.252  metric:3 if:eth2  DIRECT OSPF
[8] = dest:10.0.14.0  gw:*  mask:255.255.255.252  metric:3 if:eth1  DIRECT OSPF
[9] = dest:10.0.34.0  gw:*  mask:255.255.255.252  metric:2 if:eth1  DIRECT OSPF
[10] = dest:10.3.14.0  gw:*  mask:255.255.255.252  metric:4 if:eth1  DIRECT OSPF
[11] = dest:10.3.24.0  gw:*  mask:255.255.255.252  metric:4 if:eth1  DIRECT OSPF
[12] = dest:10.0.13.0  gw:*  mask:255.255.255.252  metric:3 if:eth0  DIRECT OSPF
[13] = dest:10.1.34.0  gw:*  mask:255.255.255.252  metric:6 if:eth2  DIRECT OSPF
[14] = dest:10.1.4.0  gw:*  mask:255.255.255.0  metric:4 if:eth2  DIRECT OSPF
[15] = dest:10.1.13.0  gw:*  mask:255.255.255.252  metric:6 if:eth1  DIRECT OSPF
[16] = dest:10.3.12.0  gw:*  mask:255.255.255.252  metric:7 if:eth1  DIRECT OSPF
[17] = dest:10.3.1.0  gw:*  mask:255.255.255.0  metric:5 if:eth1  DIRECT OSPF
[18] = dest:10.3.2.0  gw:*  mask:255.255.255.0  metric:5 if:eth1  DIRECT OSPF
[19] = dest:10.2.2.0  gw:*  mask:255.255.255.0  metric:6 if:eth0  DIRECT OSPF
[20] = dest:10.2.1.0  gw:*  mask:255.255.255.0  metric:6 if:eth0  DIRECT OSPF
[21] = dest:10.1.3.0  gw:*  mask:255.255.255.0  metric:7 if:eth2  DIRECT OSPF
```

Obrázok 4.4: Smerovacie tabuľky smerovača R02

Cisco Router

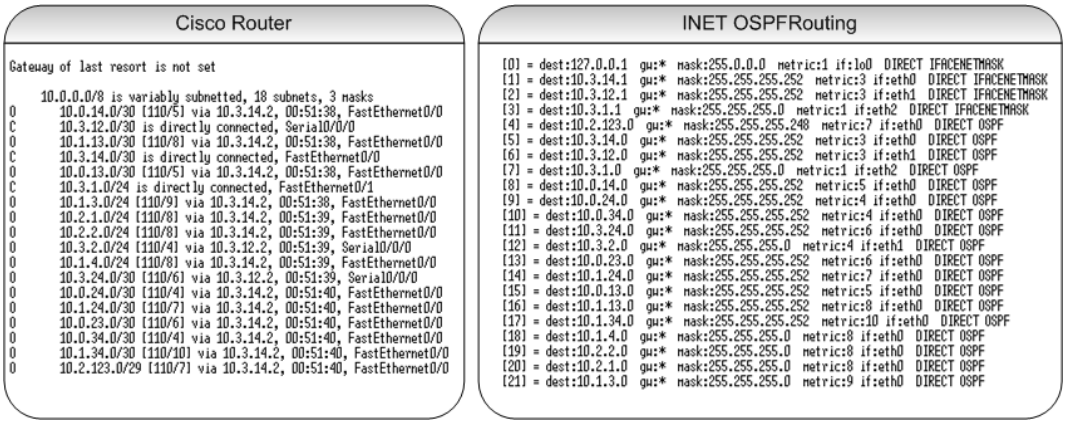
```
Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 18 subnets, 3 masks
O       10.0.14.0/30 [110/6] via 10.2.123.3, 01:33:37, FastEthernet0/0
O       10.3.12.0/30 [110/10] via 10.2.123.3, 01:20:09, FastEthernet0/0
O       10.1.13.0/30 [110/7] via 10.2.123.3, 01:33:37, FastEthernet0/0
O       10.3.14.0/30 [110/7] via 10.2.123.3, 01:22:27, FastEthernet0/0
O       10.0.13.0/30 [110/4] via 10.2.123.3, 01:33:37, FastEthernet0/0
O       10.3.1.0/24 [110/8] via 10.2.123.3, 01:21:32, FastEthernet0/0
O       10.1.3.0/24 [110/8] via 10.2.123.3, 01:33:38, FastEthernet0/0
C       10.2.1.0/24 is directly connected, FastEthernet0/1
O       10.2.2.0/24 [110/4] via 10.2.123.2, 01:33:38, FastEthernet0/0
O       10.3.2.0/24 [110/8] via 10.2.123.3, 01:19:25, FastEthernet0/0
O       10.1.4.0/24 [110/9] via 10.2.123.3, 01:33:38, FastEthernet0/0
O       10.3.24.0/30 [110/7] via 10.2.123.3, 01:20:10, FastEthernet0/0
O       10.0.24.0/30 [110/5] via 10.2.123.3, 01:33:39, FastEthernet0/0
O       10.1.24.0/30 [110/8] via 10.2.123.3, 01:33:39, FastEthernet0/0
O       10.0.23.0/30 [110/5] via 10.2.123.3, 01:33:39, FastEthernet0/0
O       10.0.34.0/30 [110/4] via 10.2.123.3, 01:33:39, FastEthernet0/0
O       10.1.34.0/30 [110/10] via 10.2.123.3, 01:33:39, FastEthernet0/0
C       10.2.123.0/29 is directly connected, FastEthernet0/0
```

INET OSPFRouting

```
[0] = dest:127.0.0.1  gw:*  mask:255.0.0.0  metric:1 if:lo0  DIRECT IFACENETMASK
[1] = dest:10.2.123.1  gw:*  mask:255.255.255.248  metric:3 if:eth0  DIRECT IFACENETMASK
[2] = dest:10.2.1.1  gw:*  mask:255.255.255.0  metric:1 if:eth1  DIRECT IFACENETMASK
[3] = dest:10.2.123.0  gw:*  mask:255.255.255.248  metric:3 if:eth0  DIRECT OSPF
[4] = dest:10.2.1.0  gw:*  mask:255.255.255.0  metric:1 if:eth1  DIRECT OSPF
[5] = dest:10.2.2.0  gw:*  mask:255.255.255.0  metric:4 if:eth0  DIRECT OSPF
[6] = dest:10.0.13.0  gw:*  mask:255.255.255.252  metric:4 if:eth0  DIRECT OSPF
[7] = dest:10.0.23.0  gw:*  mask:255.255.255.252  metric:5 if:eth0  DIRECT OSPF
[8] = dest:10.0.34.0  gw:*  mask:255.255.255.252  metric:4 if:eth0  DIRECT OSPF
[9] = dest:10.0.14.0  gw:*  mask:255.255.255.252  metric:6 if:eth0  DIRECT OSPF
[10] = dest:10.1.13.0  gw:*  mask:255.255.255.252  metric:7 if:eth0  DIRECT OSPF
[11] = dest:10.0.24.0  gw:*  mask:255.255.255.252  metric:5 if:eth0  DIRECT OSPF
[12] = dest:10.3.14.0  gw:*  mask:255.255.255.252  metric:7 if:eth0  DIRECT OSPF
[13] = dest:10.3.24.0  gw:*  mask:255.255.255.252  metric:7 if:eth0  DIRECT OSPF
[14] = dest:10.1.24.0  gw:*  mask:255.255.255.252  metric:8 if:eth0  DIRECT OSPF
[15] = dest:10.1.34.0  gw:*  mask:255.255.255.252  metric:10 if:eth0  DIRECT OSPF
[16] = dest:10.1.3.0  gw:*  mask:255.255.255.0  metric:8 if:eth0  DIRECT OSPF
[17] = dest:10.3.12.0  gw:*  mask:255.255.255.252  metric:10 if:eth0  DIRECT OSPF
[18] = dest:10.3.1.0  gw:*  mask:255.255.255.0  metric:8 if:eth0  DIRECT OSPF
[19] = dest:10.3.2.0  gw:*  mask:255.255.255.0  metric:8 if:eth0  DIRECT OSPF
[19] = dest:10.3.2.0  gw:*  mask:255.255.255.0  metric:8 if:eth0  DIRECT OSPF
[20] = dest:10.1.4.0  gw:*  mask:255.255.255.0  metric:9 if:eth0  DIRECT OSPF
```

Obrázok 4.5: Smerovacie tabuľky smerovača R21



Obrázok 4.6: Smerovacie tabuľky smerovača R31

je metrika danej cesty. U smerovačov Cisco indikuje metriku druhé číslo uvedené v hranatej zátvorke. V simulátore je to číslo uvedené za slovom metric. Po analýze všetkých záznamov zistíme, že metrika u ciest zo smerovačov Cisco sa zhoduje s metrikou odpovedajúcich ciest v simulácii s modulom OSPFRouting.

Na základe tohto pozorovania môžeme prehlásiť výber ciest pomocou modulu OSPFRouting za korektný.

4.3 Výkonnostné testy modulu OSPFRouting

Pri výbere smerovacieho protokolu je veľký dôraz kladený na rýchlosť konvergencie. Je dosť obtiažne rozhodnúť, ktorý z protokolov konverguje rýchlejšie, lebo svoju úlohu tu zohrávajú nastavenia časovačov či topológia siete.

Iná situácia nastáva, keď meriame rýchlosť konvergencie jednotlivých implementácii toho istého protokolu. Porovnanie sa stáva objektívnejšie, keďže správanie algoritmu na rovnakej topológii je deterministické.

Aby bolo možné porovnávať implementácie jednotlivých výrobcov, bola pre smerovací protokol OSPF vytvorená sada výkonnostných testov. Testy sú popísané v RFC 4061[4] a sú zamerané na konvergenciu jedného testovaného smerovača v rámci oblasti.

Otázne je, prečo testovať výkonnosť protokolu OSPF v simulácii. V skutočnosti nejde o to, dokázať, že OSPF v simulátore konverguje rýchlejšie alebo pomalšie ako na smerovačoch Cisco. Cieľom je skôr porovnanať, do akej miery sú časy simulácie hodnoverné, a či je vôbec možné simulátor použiť na skúmanie rýchlosti konvergencie v OSPF sieti.

Z pohľadu simulátora, na rýchlosť konvergencie nevplýva doba výpočtu algoritmu na procesore. Doba výpočtu v simulátore trvá nula sekúnd simulačného času. Aby tento čas zodpovedal realite, musela by simulácia obsahovať model procesora. S tým súvisí aj zavedenie plánovania a prideľovania procesora jednotlivým procesom. Každá akcia na smerovači by musela najprv požiadať o pridelenie procesora, a až tak by sa mohla uskutočniť. Aktuálne však INET Framework model procesora neobsahuje, a zatiaľ sa o jeho vytvorení ani neuvažuje.

RFC 4061 obsahuje dva typy testov nad OSPF protokolom. Prvý typ súvisí s rýchlosťou výpočtu na procesore, ako napr. čas potrebný pre spracovanie LSA, či dĺžka výpočtu SPF stromu. Ako bolo uvedené, INET Framework neobsahuje model procesora, preto nie je

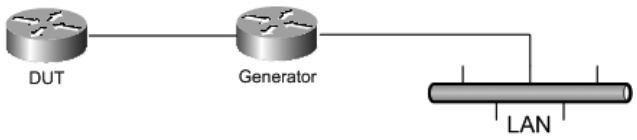
možné tieto testy realizovať. Druhým typom sú testy merajúce výkon OSPF implementácie pri základných úlohách v rámci jednej oblasti. Ide o tzv. black box testy, teda merania vychádzajú zo sledovania komunikácie testovaného zariadenia s ostatnými zariadeniami.

V nasledujúcej časti budú prezentované výsledky jednotlivých testov.

4.3.1 Test: Formovanie susedstva na point-to-point spoji s inicializáciou

Test meria čas potrebný na vytvorenie OSPF susedstva medzi dvoma smerovačmi od vytvorenia spojenia na linkovej vrstve OSI.

Pre testovanie sa využíva topológia podľa obrázka 4.7. Obsahuje testovaný smerovač (DUT) a smerovač, ktorý zachytáva jednotlivé časové okamihy (Generator).



Obrázok 4.7: Topológia testu na point-to-point spoji s inicializáciou

Na začiatku je na oboch smerovačoch nakonfigurované OSPF a linka medzi nimi je deaktivovaná. Po aktivovaní linky sa na Generatoru meria čas od vytvorenia spojenia na linkovej vrstve, po prijatie posledného potvrdzovacieho paketu na LSA zaslané Generatorom.

Priepustnosť linky	**Smerovače Cisco**	**INET OSPFRouting**
64 Kbps	17,089s	6,00338s
128 Kbps	16,793s	6,00169s
256 Kbps	17,054s	6,00084s
512 Kbps	17,049s	6,00042s
2048 Kbps	17,054s	6,00011s

Tabuľka 4.1: Výsledky testu na point-to-point spoji s inicializáciou

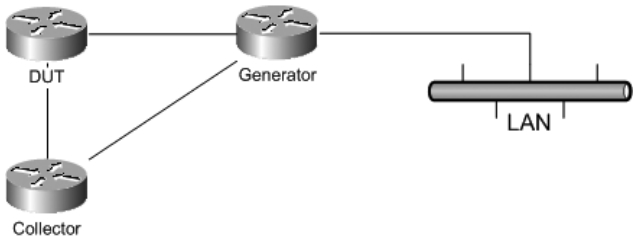
Hodnoty pre smerovače Cisco sú prevzaté z článku [10] a čiastočne overené na smerovačoch v školskom laboratóriu. Meranie bolo uskutočnené na smerovačoch rady 2500 a 2600 s operačným systémom IOS 12.0 a vyšším.

Ako je vidieť z výsledkov merania, rozdiel medzi reálnym zariadením a simuláciou je rádovo 11 sekúnd. Tento rozdiel je tak priepastný, že určite nie je spôsobený dĺžkou výpočtu OSPF na procesore. Najpravdepodobnejším vysvetlením je, že OSPF v smerovači Cisco používa neštandardný časovač tzv. SPF Interval, ktorý nie je definovaný v RFC 2328. Tento časovač určuje minimálnu dobu medzi dvoma výpočtami SPF stromu. Implicitne je nastavený na 10 sekúnd, čo by čiastočne zodpovedalo zistenému rozdielu.

4.3.2 Test: Formovanie susedstva na point-to-point spoji s už inicializovanou databázou

V tomto teste má testovaný smerovač informácie o celej topológii, a pri vytváraní susedstva už nemusí tieto informácie získavať.

Topológia použitá pre tento test je zobrazená na obrázku 4.8.



Obrázok 4.8: Topológia testu na point-to-point spoji s už inicializovanou databázou

Na začiatku je na všetkých smerovačoch nakonfigurované OSPF a linka medzi DUT a Genretorom je deaktivovaná. DUT má informácie o topológii siete získané cez Collector. Po aktivovaní linky sa na Generatoru meria čas od vytvorenia spojenia na linkovej vrstve, po prijatie posledného potvrdzovacieho paketu na LSA zaslané Generatorom.

Priepustnosť linky	**Smerovače Cisco**	**INET OSPFRouting**
64 Kbps	13,156s	1,12725s
128 Kbps	13,129s	1,11543s
256 Kbps	13,101s	1,10953s
512 Kbps	13,057s	1,10658s
2048 Kbps	13,055s	1,10436s

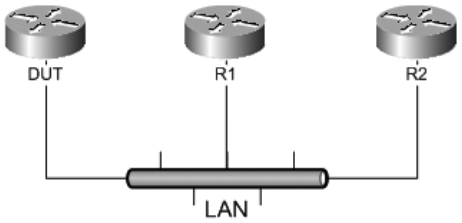
Tabuľka 4.2: Výsledky testu na point-to-point spoji s už inicializovanou databázou

Hodnoty pre smerovače Cisco sú prevzaté z článku [10].

Rozdiel medzi nameranými hodnotami ostal vysoký. Rádovo to činí 12 sekúnd. Opäť to môže byť spôsobené už spomínaným časovačom SPF Interval. Dôležité je však zistenie, že rozdiel medzi vytvorením susedstva s už aktuálnou databázou je u reálnych smerovačov rádovo o 4 sekundy a v simulácii o 5 sekúnd kratší, ako bez inicializovanej databázy.

4.3.3 Test: Počiatočná konvergencia na broadcastovej linke

Test meria čas potrebný pre inicializáciu OSPF na broadcastovej linke. K tomu je použitá topológia z obrázka 4.9. Táto topológia obsahuje testovaný smerovač DUT, ktorý je súčasne DR (designated router), smerovač R1, na ktorom sú zachytávané jednotlivé časové okamihy a smerovač R2, ktorý má úlohu BDR (backup designated router).



Obrázok 4.9: Topológia testu počiatočnej konvergencie na broadcastovej linke

Na začiatku je na všetkých smerovačoch nakonfigurované OSPF. Broadcastové linky

smerovačov DUT a R2 sú aktívne a medzi oboma smerovačmi je vytvorené susedstvo. Linka smerovača R1 je neaktívna. Po aktivovaní tejto linky sa na smerovači R1 meria čas od prvého Hello paketu, prijatého od DUT, po čas prijatia prvého Network LSA.

Priepustnosť linky	**Smerovače Cisco**	**INET OSPFRouting**
10 Mbps	3,068s	10,10691s
100 Mbps	3,028s	10,10648s

Tabuľka 4.3: Výsledky testu počiatočnej konvergencie na broadcastovej linke

Hodnoty pre smerovače Cisco boli merané na produktovej rade 2800 s operačným systémom IOS verzie 12.4. Výsledky, oproti predchádzajúcim dvom testom, sú iné v tom, že simulácia s OSPFRouting konverguje pomalšie ako na smerovačoch Cisco.

Podľa definície v RFC 2328, je pre ustanovenie susedstva na broadcastovej linke potrebná výmena minimálne dvoch Hello paketov. Jeden paket na prechod do stavu *Init* a druhý na prechod do stavu *TwoWay/ExStart* (diagram z obrázka 4.2). Z toho vyplýva, že pri implicitných hodnotách Hello intervalu, by konvergencia mala trvať minimálne 10 sekúnd. Zodpovedá to približne hodnotám nameraným v simulácii. Podľa všetkého, implementácia na smerovačoch Cisco v čase počiatočnej konvergencie zasiela Hello pakety aj v iregulárnych intervaloch. Túto skutočnosť potvrdzuje výpis OSPF aktivity na smerovači, kde v krátkom časovom okamihu je možné pozorovať prijatie dvoch Hello paketov. Práve tento fakt spôsobuje rozdiel v nameraných hodnotách.

4.4 Implementovaná funkcionalita v module OSPFRouting

Obsahom tejto podkapitoly je zhrnutie funkcií obsiahnutých v module OSPFRouting. Taktiež je tu vymenovaných pár funkcií, ktoré sú v praxi dosť často používané a v module OSPFRouting chýbajú.

4.4.1 Podporované funkcie

- **4 typy liniek:**
  - **Broadcast** – na tomto type linky sa na jednom segmente môžu vyskytovať viac ako dva OSPF smerovače, preto dochádza k výberu DR, resp. BDR. Komunikácia prebieha pomocou multicastu.
  - **Point-to-Point** – na linke komunikujú dva OSPF smerovače pomocou multicastu. Výber DR a BDR nie je potrebný.
  - **Non-Broadcast-Multiple-Access** – komunikujú viac ako dva smerovače. Nie je však podporované broadcastové vysielanie. Komunikácia prebieha unicastom a dochádza k výberu DR a BDR.
  - **Virtual** – nejde o fyzický typ linky. Linka je virtuálna a slúži na prepojenie oblasti s chrbticovou oblasťou cez nechrbticovú oblasť.
- **Interface cost** – každému rozhraniu je možné priradiť ohodnotenie linky, na základe ktorého bude vybratá najkratšia cesta.

- **OSPF priority** – na rozhraniach, u ktorých dochádza k výberu DR a BDR, je možné tento výber ovplyvňovať nastavením priority.
- **Hello/Dead interval** – časovače, ktoré určujú frekvenciu zasielania Hello paketov, resp. čas, po ktorom je zrušené susedstvo v prípade neprijatia Hello paketu. Nastavením týchto časovačov je možne ovplyvňovať rýchlosť konvergencie v OSPF sieti.
- **Autentizácia** – zabezpečenie OSPF komunikácie pred podvrhnutím nepravých OSPF paketov.
- **Oblasti** – členenie OSPF siete do menších segmentov, v rámci ktorých si smerovač udržiava informácie o celej topológii.

4.4.2 Nepodporované funkcie

- **Sumarizácia** – ide o zgrupovanie sietí do tzv. supersietí. V oblasti sa potom namiesto ciest pre všetky siete šíri len jedna cesta pre celú supersieť. K sumarizácii môže dochádzať na rozhraní oblastí (na ABR), alebo na rozhraní autonómnych systémov (na ASBR).
- **Stub oblasti** – v stub oblastiach je zakázané šírenie akýchkoľvek LSA obsahujúcich cesty externých autonómnych systémov.
- **No-so-stuby** – do no-so-stuby oblastí sa nešíria LSA obsahujúce cesty externých autonómnych systémov. Oblasť však môže byť na hranici autonómnych systémov a je povolený prenos LSA o externých cestách z ASBR do ABR.
- **Redistribúcia** – šírenie informácii v rámci OSPF o cestách, ktoré boli naučené iným smerovacím protokolom alebo boli ručne nakonfigurované ako statické.

Kapitola 5

Využitie OSPF simulácie

Kapitola je venovaná využitiu modulu OSPFRouting pre simulovanie, získavanie a vyhodnocovanie informácií o rôznych sieťových topológiach. V prvej časti kapitoly budú najprv predstavené zmeny v INET Framework a module OSPFRouting, ktoré bolo treba implementovať, aby bolo možné simulovať zmeny stavu rozhraní smerovača (Up / Down). Následne bude na konkrétnych ukážkach demonštrované využitie simulácie so smerovacím protokolom OSPF.

Predmetom skúmania bude správanie OSPF v prostredí siete s rôznou chybovosťou liniek, možnosti zrýchlenia konvergencie siete nastavením hello, resp. dead intervalu a závislosť počtu opakovaných výpočtov SPF algoritmu na počte zmien v topológii.

5.1 Zmeny v INET Framework a module OSPFRouting

OSPF je dynamický smerovací protokol. Znamená to, že sa dokáže prispôsobiť rôznym zmenám v topológii siete. Aby bolo možne toto správanie simulovať, je potrebné, aby simulátor umožňoval tieto zmeny plánovať. Ako už bolo uvedené v sekcii 3.2.2, OMNeT++ nemá implicitnú podporu skriptovania priebehu simulácie. INET Framework obsahuje modul ScenarioManager, ktorý rieši tento nedostatok. Avšak ScenerioManger aktuálne podporuje iba moduly pre simulovanie bezdrôtových sietí. Preto bolo potrebné rozšíriť existujúci modul smerovača, aby dokázal spolupracovať s modulom ScenarioManager.

Z hľadiska implementácie sa naskytovali dve možnosti riešenia. Prvou možnosťou bolo rozšíriť modul ScenarioManager tak, aby dokázal interpretovať príkazy o zmene stavu rozhrania načítavaných zo XML súboru. Následne by vykonal všetky kroky potrebné k tomu, aby sa zmena aplikovala, ako napr. zakázať posielanie paketov cez dané rozhranie, informovať smerovací protokol OSPF a zmeniť statické záznamy smerovacej tabuľky. Tiež by si musel uchovávať ukazovatele na tabuľky rozhraní všetkých smerovačov v simulácii a všetky moduly, ktoré musia na zmenu reagovať, by museli vedieť s modulom ScenarioManager komunikovať (moduly by museli dediť triedu IScriptable). Implementácia týmto spôsobom by bola pomerne zložitá a vyžadovala by si radikálnejšie zásahy do už existujúcich modulov.

Oproti tomu druhá a implementovaná varianta ponecháva modul ScenarioManager nezmenený. Celá obsluha príkazov je presunutá do novo vytvoreného modulu s názvom InterfaceStateManager, ktorý je súčasťou každého smerovača. ScenarioManager príkazu o zmene stavu rozhrania nerozumie, dokáže len zistiť, pre ktorý smerovač je príkaz určený a následne poslať celý XML element s príkazom modulu InterfaceStateManager daného smerovača. InterfaceStateManager už vie príkaz interpretovať a zariadi potrebné kroky k jeho vykonaniu.

Celý priebeh zmeny stavu rozhrania zahŕňa nasledujúcich 5 krokov (obrázok 5.1):

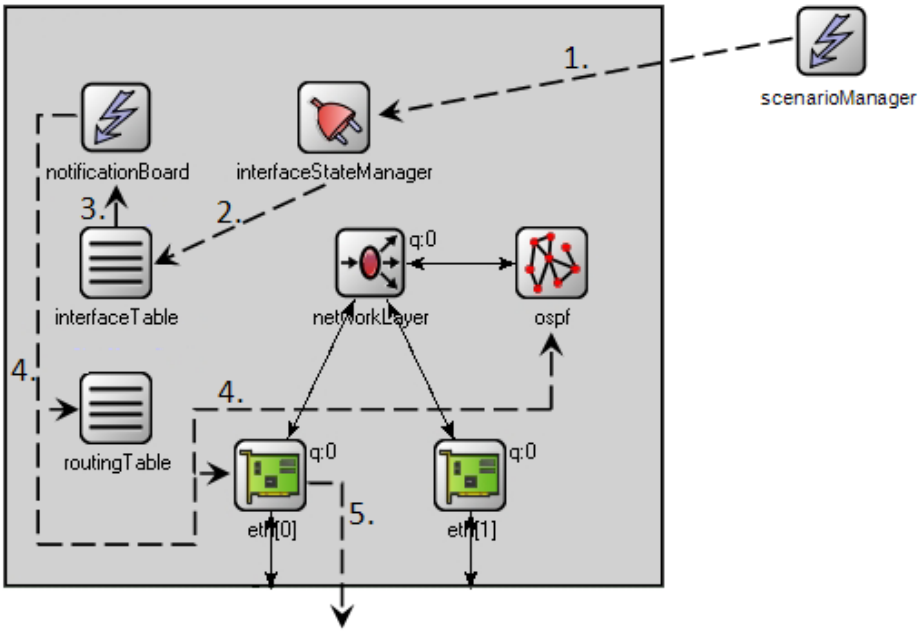
1. ScenarioManager identifikuje smerovač, ktorého linka má zmeniť stav a predá XML element s príkazom modulu InterfaceStateManager daného smerovača.

2. InterfaceStateManager si udržiava ukazovateľ na tabuľku rozhraní smerovača (objekt InterfaceTable). Z príkazu identifikuje rozhranie a akciu, ktorá sa má vykonať (zmena na Up / Down). Následne získa ukazovateľ na objekt daného rozhrania a zavolá funkciu, ktorá zmení logickú hodnotu premennej rozhrania *down*.

3. Po zmene, InterfaceTable vygeneruje notifikáciu, ktorú zašle modulu NotificationBoard. Notifikácia je typu NF_INTERFACE_STATE_CHANGED, a ako parameter nesie ukazovateľ na rozhranie, ktoré zmenilo stav.

4. Po tom, ako modul NotificationBoard príjme notifikáciu, rozposiela ju všetkým modulom, ktoré sa prihlásili k odberu tohto typu notifikácie. V prípade zmeny stavu rozhrania sú to moduly EthernetInterface, RoutingTable a OSPFRouting. EthernetInterface musí zablokovať priechod paketov cez rozhranie. RoutingTable musí vymazať cesty súvisiace s daným rozhraním. A OSPFRouting musí vygenerovať LSA o zmenene topológie.

5. V prípade, že rozhranie spája dva priamo susediace smerovače, musí sa zmeniť aj stav rozhrania na druhom smerovači. O toto sa už stará modul fyzickej vrstvy, teda modul EthernetInterface alebo modul PPPInterface.



Obrázok 5.1: Priebeh zmeny stavu rozhrania

V nasledujúcom texte budú popísané konkrétne zmeny vykonané v jednotlivých moduloch.

5.1.1 Nový modul InterfaceStateManager

Definícia modulu je popísaná v súbore *InterfaceStateManager.ned*. Ide o jednoduchý modul bez parametrov a vstupných či výstupných brán. S ostatnými modulmi komunikuje priamo volaním funkcií. V súboroch *InterfaceStateManager.cc* a *InterfaceStateManager.h* sa nachádza C++ implementácia modulu. Trieda *InterfaceStateManager* dedí vlastnosti triedy *IScriptable*, aby modul dokázal prijímať príkazy od modulu ScenarioManager.

Trieda obsahuje nasledovné 4 funkcie:

- *initialize()* – v rámci inicializácie získa ukazovateľ do tabuľky rozhraní.
- *handleMessage()* – každý jednoduchý modul v OMNeT++ musí mať implementovanú funkciu pre príjem správ. V tomto prípade však neimplementuje žiadnu funkcionalitu, keďže modul neobsahuje žiadne brány.
- *processCommand()* – funkcia ako pramater prijíma XML element od modulu ScenarioManager. XML element rozparsuje a na základe jeho obsahu vyhľadá rozhranie a zavolá funkciu *changeInterfaceState()*
- *changeInterfaceState()* – parametrami funkcie sú ukazovateľ na rozhranie a logická hodnota nového stavu rozhrania. Ak je logická hodnota premennej rozhrania *down* opačná ako parameter nového stavu, tak sa zmení jej hodnota.

5.1.2 Úprava modulu OSPFRouting

Hlavnú triedu *OSPFRouting* bolo treba upraviť tak, aby dedila od triedy *INotifiable*. To umožňuje, aby modul dokázal prijímať notifikácie od modulu NotificationBoard. V inicializácii bolo doplnené prihlásenie k odberu notifikácií o zmene stavu rozhrania NF_INTERFACE_STATE_CHANGED. Pre implementáciu reakcie na prijatú notifikáciu bola doplnená funkcia *receiveChangeNotification()*. Funkcia na základe prijatej notifikácie identifikuje OSPF rozhranie, na ktorom nastala zmena, a podľa stavového automatu zobrazeného na obrázku 4.1 zašle rozhraniu signál (InterfaceUp / InterfaceDown), ktorým sa zmení jeho stav.

Pre správnu reakciu modulu OSPFRouting na zmenu stavu rozhrania, by popísané zmeny mali postačovať. Avšak boli odhalené chyby implementácie, ktoré po niektorých zmenách stavu spôsobovali buď pád celej simulácie, alebo nekorektnosť smerovacieho algoritmu. Pravdepodobne to bolo spôsobené tým, že autor modulu nemohol tieto dynamické vlastnosti otestovať, keďže mu to simulátor nepovoľoval.

Odhalené boli nasledovné 3 chyby:

- Pri prechode rozhrania do stavu Down sa musia zrušiť všetky susedstvá, ktoré sú cez toto rozhranie vytvorené. Pri tomto rušení však dochádzalo k nesprávnemu generovaniu a rozposielaniu LSA, ktoré mali za následok, že ostatné smerovače nezaregistrovali zmenu v topológii. Riešením bolo rozšíriť triedu *OSPF::Interface* o premennú *isGoingDown*, ktorá indikuje rozhranie vo fáze prechodu do stavu Down. Funkcia pre odstraňovanie susedstiev bola zmenená tak, že keď je nastavená premenná *isGoingDown*, negeneruje žiadne LSA.
- Na rozhraní typu broadcast, sa po prechode rozhrania do stavu Down, musí vygenerovať Network LSA, ktoré zneplatní aktuálne Network LSA pre danú sieť. Následné sa rozposiela v rámci celej oblasti a každý smerovač ho vymaže zo svojej databázy. Tento postup bol implementovaný správne. Problém bol pri mazaní LSA z databázy.

Smerovač správne vymazal Network LSA, no nechal nezmenené ostatné LSA, ktoré prišli cez danú sieť. Pri následnom výpočte SPF stromu sa tieto LSA odkazovali na informácie, ktoré už boli vymazané z databázy, čo spôsobovalo prístup do už uvoľnenej pamäte a následný pád celej simulácie. Riešením bolo pridanie funkcie *RemoveParentFromRoutingInfo()*. Funkcia pri odstraňovaní Network LSA zaistí aj vynulovanie všetkých ukazovateľov na záznam tohto LSA.

- Na rozhraní dvoch oblastí smerovač ABR generuje Summary LSA informujúce jednu oblasť o sieťach druhej oblasti. Každé LSA má obmedzenú platnosť MAX_AGE, implicitne je to 60 minút. V polovici tohto času, teda v čase 30 minút, pôvodca LSA vygeneruje nové LSA, ktoré predlžuje platnosť pôvodného a rozpošle ho do celej oblasti. V prípade ABR však nedochádzalo ku generovaniu LSA, ktoré nepredlžovalo platnosť, ale naopak nové LSA zneplatnilo pôvodné, čo malo za následok jeho okamžité vymazanie z databázy. Problém bol vo vyhodnotení informácie o tom, či oblasť ešte stále obsahuje sieť z daného Summary LSA. Pre vyriešenie problému stačilo prezátvorkovať podmienený vyraz vo funkcii *OriginateSummaryLSA()*.

5.1.3 Úprava modulu EthernetInterface

Pôvodná implementácia modulu EthernetInterface už dokázala komunikovať s modulom NotificationBoard. Preto stačilo len pridať prihlásenie k odberu notifikácii o zmene stavu linky NF_INTERFACE_STATE_CHANGED a doplniť reakciu na túto notifikáciu do funkcie *receiveChangeNotification()*. Modul na notifikáciu zareaguje tým, že nastaví premennú *Disabled* a v prípade, že aj na opačnej strane linky je smerovač, nastaví aj jeho rozhranie do stavu Down. Premenná *Disabled* riadi prechod správ cez modul. V prípade, že obsahuje logickú hodnotu true, sú všetky správy zahadzované.

5.2 Simulácia: Chybovosť liniek

Cieľom tejto simulácie je ukázať správanie smerovacieho protokolu OSPF v prostredí siete s chybovými linkami.

Medzi OSPF smerovačmi na jednej linke sa vytvárajú susedstvá, tie zaisťujú informáciu o tom, či sused na linke je stále dostupný. V prípade, že na linke dochádza k strate paketov, môžu byť tieto susedstvá na istý časový okamih rozviazané a následne opäť nadviazané. Dochádza k tzv. flapovaniu ciest.

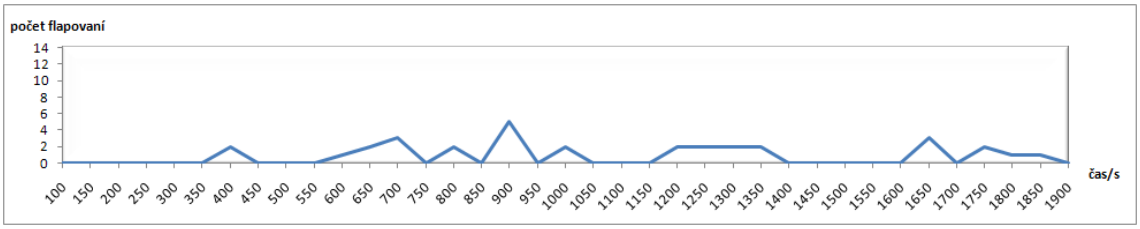
OMNeT++ umožňuje pre každú linku nastaviť v definícii NED premennú *per* (packet error). Je to pravdepodobnosť (od 0 po 1) určujúca zahadzovanie paketov na linke. Pre vyhodnotenie simulácie bolo treba rozšíriť modul OSPFRouting, aby do súboru skalárnych štatistík zapisoval hodnoty o prijatých a odoslaných paketoch, a tiež detekoval flapovanie a zapísal informáciu o ňom do logovacieho súboru. Pre čítanie a následné grafické zobrazenie štatistických súborov má OMNeT++ vstavaný nástroj. Jeho nedostatkom je, že dokáže vyhodnotiť štatistiky len pre samostatné moduly. Nedokáže ich sčítať a vyhodnotiť komplexne pre celú simuláciu. Riešením je spracovať súbory pomocou vlastných skriptov a graficky zobraziť v externom programe. V prípade tejto simulácie bol použitý skriptovací jazyk Python a pre grafické zobrazenie program MS Excel.

Na topológii z obrázka 4.3 použitej v prechádzajúcej kapitole bola postupne simulovaná rôzna chybovosť liniek od 0% až do 81% a vyhodnotený počet flapovaní. Simulácia bola v každom behu spúšťaná po dobu 1900 sekúnd simulačného času, pričom počítanie

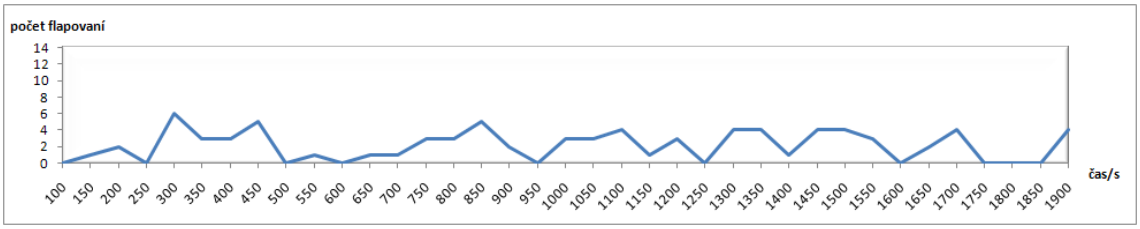
flapovania začínalo od času 100 sekúnd. Prvých 100 sekúnd je doba, počas ktorej daná topológia určite ukončí počiatočnú konvergenciu. V nasledujúcej tabuľke a grafoch budú prezentované výsledky meraní.

	0%	**10%**	**20%**	**30%**	**40%**	**50%**	**81%**
Zaslané Hello pakety	5890	5890	5890	5890	5890	5890	5890
Prijaté Hello pakety	5320	4652	4034	3458	2906	2357	822
Zaslané OSPF pakety	7776	11415	15377	19201	20482	18695	8727
Prijaté OSPF pakety	7408	9703	11345	12292	10769	7794	1345
Počet flapovaní	0	32	80	198	227	178	0

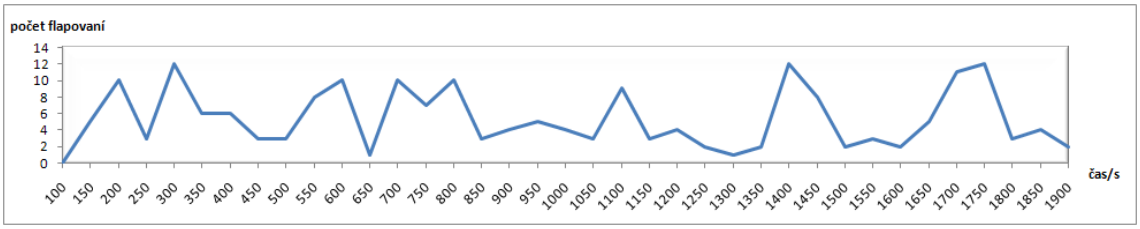
Tabuľka 5.1: Výsledky meraní flapovania na linkách s rôznou chybovosťou



Obrázok 5.2: Flapovanie pri chybovosti 10%

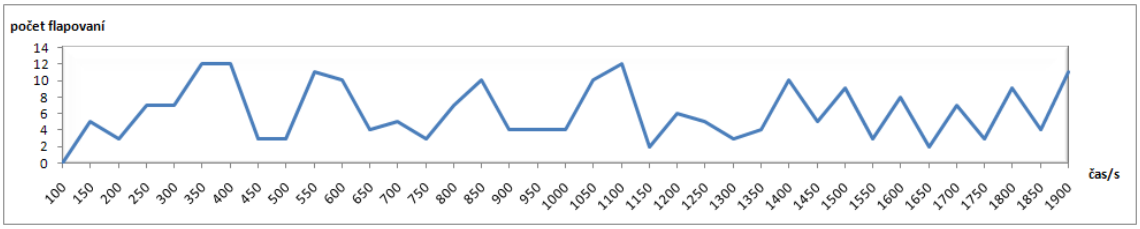


Obrázok 5.3: Flapovanie pri chybovosti 20%

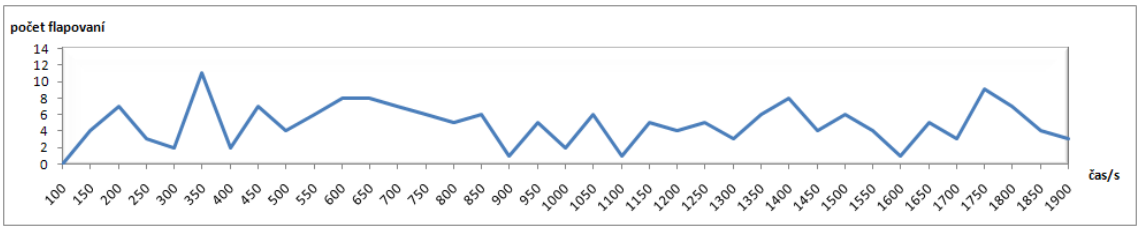


Obrázok 5.4: Flapovanie pri chybovosti 30%

Ako je vidieť z výsledkov simulácií, počet flapovaní rastie s chybovosťou liniek len do 40% chybovosti. Pri chybovosti 50% je už počet flapovaní menší. Príčinou je veľké množstvo stratených paketov, ktoré spôsobuje predĺženie času potrebného na znovuobnovenie



Obrázok 5.5: Flapovanie pri chybovosti 40%



Obrázok 5.6: Flapovanie pri chybovosti 50%

susedstva. Pri hodnote chybovosti 81%, dokonca, už nedochádza k žiadnemu flapovaniu. Nie že by v tomto prípade bola už sieť stabilná, práve naopak, pri tejto chybovosti už nedokážu smerovače medzi sebou ani za 1900 sekúnd nadviazať susedstvo.

Počet zaslaných OSPF paketov tiež rastie do chybovosti 40%. Stojí za tým znovuzasielanie nepotvrdených OSPF paketov. Pri väčšej chybovosti už nedochádza k tak častému znovuobnoveniu susedstva, preto aj počet prenášaných paketov, ktoré vyžadujú potvrdenie, je menší.

Možno trochu zarážajúci je rozdiel medzi zaslanými a prijatými Hello paketmi pri 0% chybovosti. Dôvodom sú stub siete na niektorých smerovačoch. Do týchto stub sietí sú zasielané Hello pakety, ale nie je tam žiadny iný smerovač, ktorý by ich prijal.

5.3 Simulácia: Zmeny v topológii

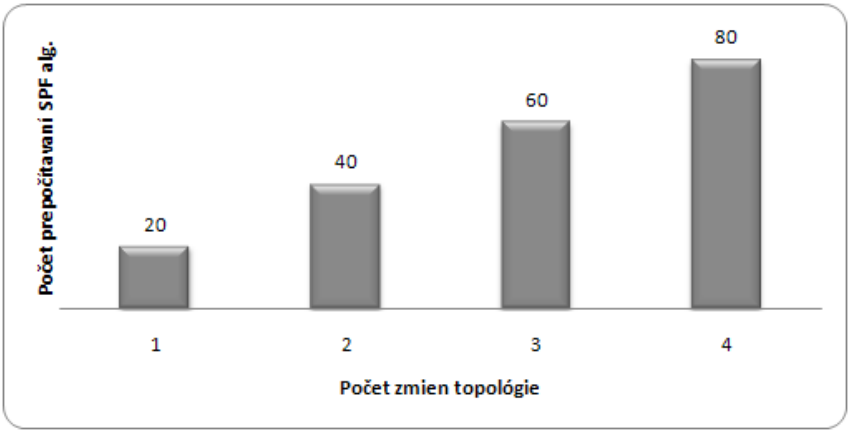
Simulácia ukazuje, ako môže počet rýchlo po sebe nasledujúcich zmien v topológii vyťažiť smerovač s veľkým množstvom prepočítavaní SPF algoritmu.

V simulácii bola použitá topológia z obrázka 4.3. Postupne bola simulácia spúšťaná s rôznym počtom simulovaných pádov linky (1 až 4). Časový interval medzi jednotlivými pádmi bol vždy 0,2 sekúnd. Pre získanie počtu kalkulácii SPF algoritmu bolo potrebné vytvoriť skript, ktorý od času prvej zmeny v topológii spočítal výskyt frázy "Routing table was rebuilt." v logoch simulácie.

Na obrázku 5.7 je znázornený graf s nameranými hodnotami.

Ako je z grafu vidieť, počet kalkulácií algoritmu je lineárne závislý na počte zmien v topológii. Použitá topológia obsahuje 10 smerovačov, čo v prepočte znamená, že na jednu zmenu v topológii musí smerovač prepočítať SPF strom dvakrát. Dvakrát preto, lebo pri zlyhaní point-to-point linky zareagujú smerovače na oboch koncoch linky vygenerovaním nového LSA.

Takáto lineárna závislosť môže byť v reálnych sieťach veľmi nebezpečná. Neustále fla-

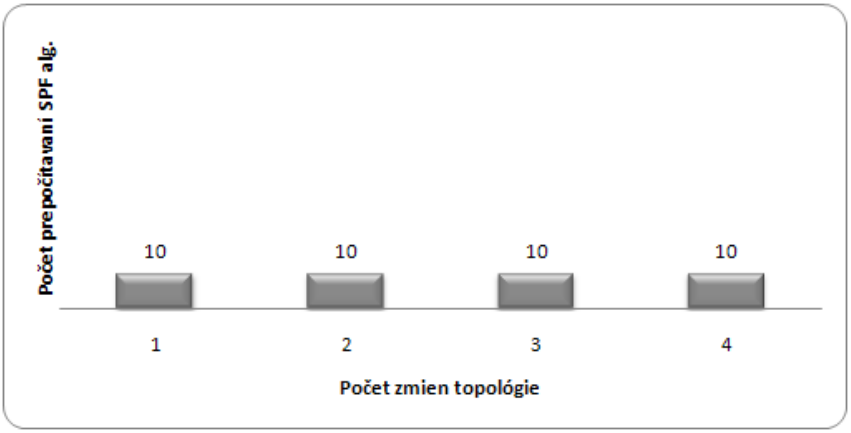


Obrázok 5.7: Prepočítavanie SPF algoritmu v závislosti na zmenách topológie

povanie niekoľkých fyzických liniek môže spôsobiť také množstvo kalkulácií SPF algoritmu, že procesory smerovačov, najmä tých menej výkonných, budú tak vyťažené, že nebudú schopné vykonávať ich základnú funkciu, a to smerovať pakety.

Niektorí výrobcovia smerovačov prišli s riešením a implementovali nový časovač - SPF delay. SPF delay určuje čas medzi prijatím LSA a aktiváciou SPF algoritmu. Implicitná hodnota je 5 sekúnd. Treba však poznamenať, že tento časovač nie je súčasťou definície RFC 2328.

V prípade implementovaného SPF delay by graf merania vyzeral podľa obrázku 5.8.



Obrázok 5.8: Prepočítavanie SPF algoritmu s implementovaným SPF delay

Aj keď v topológii došlo k štyrom zmenám, tieto štyri zmeny prebehli do jednej sekundy. Čiže LSA o všetkých zmenách prišli na každý smerovač do piatich sekúnd. Smerovač, teda z pôvodných ôsmich kalkulácií, vykoná iba jednu. Zaručí to šetrenie výkonom a väčšiu stabilitu siete. Na druhej strane použitie SPF delay prináša jednu nevýhodu. Aj keď v topológii nastane len jedna zmena, ku konvergencii nedôjde skôr, ako po vypršaní piatich sekúnd časovača.

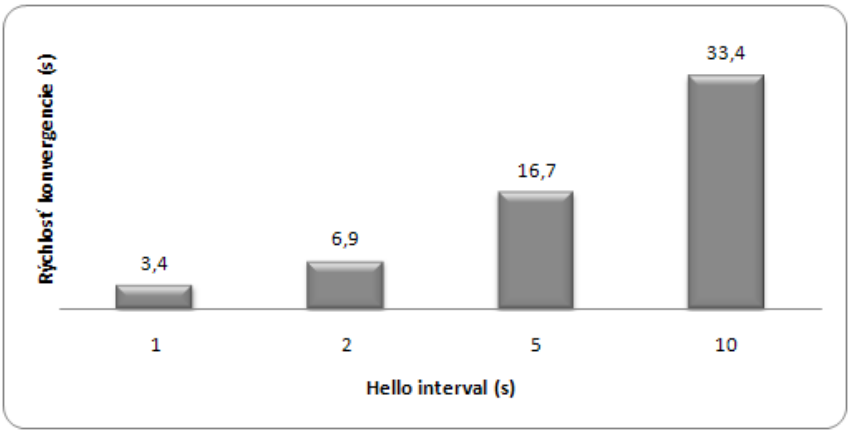
5.4 Simulácia: Rýchlosť konvergencie

Na tejto simulácii bude ukázaná možnosť zrýchlenia konvergencie OSPF siete. Konkrétne sa bude týkať nastavenia Hello resp. Dead intervalu, jeho vplyvu na rýchlosť konvergencie a zaťaženie linky.

Detekcia poruchy siete je v protokole OSPF postavená na dvoch mechanizmoch. Prvým je signalizácia od linkovej vrstvy. Detekcia týmto spôsobom je väčšinou rýchla a OSPF môže na udalosť zareagovať do jednej sekundy od jej vzniku. Obmedzením je, že porucha sa musí týkať linky (vodiča) priamo pripojenej do smerovača. Problém môže nastať u broadcastových sietí, kde sú linky väčšinou zakončované na prepínači a detekcia poruchy linky niektorého zo susedov, pomocou signalizácie linkovej vrstvy, nie je možná. Riešením je druhý mechanizmus, a to zasielanie Hello paketov, ktoré informujú o tom, že sused je stále dostupný. Frekvencia zasielania Hello paketov (Hello interval) je konfigurovateľná, podobne aj doba, počas ktorej musí byť Hello paket prijatý (Dead interval). Implicitne je Hello interval nastavovaný na 10 sekúnd a Dead interval na 40 sekúnd (štvornásobok Hello intervalu).

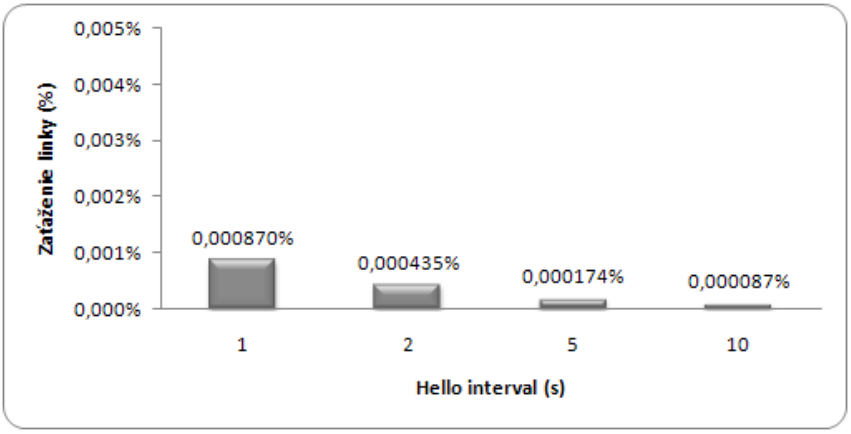
V simulácii bola opäť použitá topológia z obrázku 4.3, kde bol simulovaný pád linky medzi smerovačom R22 a prepínačom. Meraný bol čas od zmeny stavu linky po okamih, kedy všetky smerovače odstránili informáciu o stub sieti, pripojenej na smerovači R22, zo svojej smerovacej tabuľky. K vyhodnocovaniu bol použitý jednoduchý skript v jazyku Python, ktorý z logov simulácie určil požadovaný čas. Simulácia bola postupne spúšťaná so štyrmi rôznymi nastaveniami Hello intervalu. Dead interval bol vždy nastavený na štvornásobok Hello intervalu. Rýchlosť konvergencie do istej miery závisí aj na čase, kedy dôjde k pádu linky. Preto pre každé nastavenie Hello intervalu bola simulácia spúšťaná trikrát, vždy s iným náhodne generovaným časom pádu linky a výsledky boli spriemerované.

Graf na obrázku 5.9 zobrazuje namerané hodnoty rýchlosti konvergencie pri konkrétnych nastaveniach Hello intervalu. Graf na obrázku 5.10 zobrazuje, ako sa mení vyťaženosť FastEthernetovej linky pri rôznej frekvencii zasielania Hello paketov.



Obrázok 5.9: Závislosť rýchlosti konvergencie na Hello intervale

Ako je možné z výsledkov pozorovať, zmenšením Hello Intervalu z 10 sekúnd na sekundu je možné zrýchliť konvergenciu o 30 sekúnd. Pritom zaťaženie linky sa zväčší len rádovo o tisícinu percenta. Sekundový Hello interval by mal byť určite použitý v moderných sieťach, ktoré vyžadujú rýchlu konvergenciu.



Obrázok 5.10: Závislosť vyťaženosti linky na Hello intervale

Implantácia modulu OSPFRouting požíva pre reprezentáciu Hello intervalu celočíselný typ, čo zabraňuje nastaviť interval menší ako jedna sekunda. Naproti tomu, v komerčných implementáciách, je možné nastaviť aj hodnoty rádovo v milisekundách. Ako sa však uvádza v článku [1], hodnoty Hello intervalu pod 500 milisekúnd môžu v reálnych sieťach spôsobovať flapovanie liniek. Dôvodom sú častejšie straty paketov, a tiež zvýšené nároky na procesor pri spracovaní veľkého množstva Hello paketov. V simulátore by táto skutočnosť nebola odhalená, aj keby povoľoval nastavenie Hello intervalu menšieho ako sekunda. Dôvodom je, že v simulácii je za nulový simulačný čas spracované akékoľvek množstvo paketov a k strate dochádza len v prípade, že je nastavená chybovosť na linke.

Kapitola 6

Záver

OMNeT++ je mocný simulačný nástroj s prepracovaným grafickým prostredím. Pomocou nadstavby INET Framework umožňuje simuláciu sietí TCP/IP. INET Framework je bohatý na množstvo rôznych modulov, v niektorých prípadoch však trpí ich nižšou kvalitou a neúplnosťou.

V podkapitole 3.1 bola predstavená architektúra OMNeT++, jeho inštalácia, spôsob modelovania a vytváranie modulov. Podkapitola 3.2 bola venovaná INET Framework – postupu pri vytváraní simulácie a komponentám modulov reprezentujúcich smerovač.

Kapitola 4 bola celá venovaná modulu OSPFRouting, ktorý implementuje smerovací protokol OSPF. Implantačne ide o jeden z najrozsiahlejších modulov. Boli predstavené jeho základné triedy a metódy, ako aj členenie zdrojových súborov. Na konkrétnej topológii bola overená správnosť výberu cesty porovnaním s výberom na reálnych smerovačoch. Na implementácii boli prevedené výkonnostné testy, definované v RFC 4061. Testy však nedopadli ideálne, a to z nasledujúcich dôvodov. Jednak kvôli absencii modelu procesora, ktorá má za následok, že každý výpočet má dĺžku trvania nula sekúnd simulačného času. Druhým dôvodom je, že komerčná implementácia OSPF na smerovačoch Cisco, oproti modulu OSPFRouting, používa neštandardné časovače, ktoré nie sú definované v RFC 2328. Zistenia týchto testov treba brať do úvahy pri rôznych simuláciách, u ktorých je potrebné vyhodnocovať čas. Oproti správaniu na reálnych smerovačoch môže byť čas odlišný aj o 10 sekúnd. Využitie pri simuláciách, kde je potrebné len korektné smerovanie, je bezproblémové.

V kapitole 5 boli predstavené možnosti využitia OSPF simulácie. Na začiatku boli prezentované implementované zmeny, potrebné k umožneniu skriptovania priebehu simulácie. Konkrétne ide o možnosť určiť časový okamih, kedy má dôjsť k zmene stavu linky spájajúcej dve zariadenia. Toto rozšírenie má svoje využitie nielen pri OSPF simulácii, ale aj v rade iných simulácií, ako napr. zisťovanie dostupností služieb za určitých podmienok, či možnosti simulovať zlyhanie celého zariadenia. Do simulácii v INET Framework sa týmto rozšírením zanáša dynamickosť, ktorá pri simulácii drôtových sietí chýbala. V zvyšku kapitoly bolo na troch konkrétnych prípadoch demonštrované využitie OSPF simulácie. V prvej simulácii bolo skúmané, ako chybovosť linky môže ovplyvniť stabilitu siete so smerovaním OSPF. Výsledky ukázali, že už pri 10% chybovosti dochádza k nepravidelnému flapovaniu liniek a miernej nestabilite siete. Nárast počtu flapovaní končí pri 40% chybovosti. Pri chybovosti 50%, kvôli množstvu stratených paketov, už nie je tak jednoduché obnoviť susedstvo. A pri chybovosti 81% k vytvoreniu susedstva vôbec nedošlo. Cieľom druhej simulácie bolo zistiť, ako počet zmien v topológii ovplyvní počet kalkulácií SPF algoritmu. Simulácia ukázala, že počet kalkulácií je lineárne závislí na počte zmien v topológii. V priemere pripadajú

na jednu zmenu topológie dve kalkulácie SPF algoritmu. Ukazuje sa tu ako opodstatnená implementácia neštandardného časovača SPF delay, ktorý spôsobí, že kalkulácia SPF bude spozdená implicitne o 5 sekúnd od prijatia LSA. V praxi to znamená, že v rámci tohto intervalu nezáleží na počte zmien topológie, stále dôjde len k jednému výpočtu SPF algoritmu. Tretia simulácia ukázala možnosti zrýchlenia konvergencie nastavením Hello intervalu. Simuláciou bolo zistené, že zmenou Hello intervalu z implicitných 10 sekúnd na 1 sekundu je možné zrýchliť konvergenciu približne o 30 sekúnd. Zvýšenie záťaže na pásmo linky sa zmení len minimálne. Preto záverom tejto simulácie môže byť odporúčanie zmenšiť Hello interval na broadcastových linkách.

Čo sa týka ďalšieho vývoja modulu OSPFRouting, modul je stále v štádiu vývoja. Je známa jeho občasná nestabilita. Čiastočne bola odstránená v rámci tejto práce, no stále ešte ostávajú menšie chyby zdedené nesprávnym návrhom. Na škodu by určite nebolo mať podporu sumarizácie a redistribúcie, no k bežnému simulovaniu nie sú tieto funkcie až tak podstatné.

Z hľadiska vývoja simulácie by určite pomohla implementácia modelu procesora. Simulácia by tak aj z pohľadu času lepšie zodpovedala realite. Ide však o netriviálnu záležitosť, ktorá by vyžadovala radikálny zásah do celého modelu smerovača.

V neposlednom rade, by v budúcnosti bolo užitočné vytvoriť aplikáciu na zjednodušenie vytvárania simulácie. Aktuálne je pre vytvorenie simulácie potrebná ručná editácia množstva textových súborov. Niektoré informácie je potrebné zadávať dvakrát, čím sa zanáša väčšia pravdepodobnosť vzniku chýb. Samozrejme ide aj o užívateľský komfort. Čím bude práca so simuláciou intuitívnejšia a rýchlejšia, tým si získa väčšiu obľubu.

Literatúra

[1] Basu, A.; Riecke, J. G.: Stability Issues in OSPF Routing. `www.acm.org/sigcomm/sigcomm2001/p18-basu.pdf`, 2001.

[2] Dokumentácia: INET Framework for OMNeT++/OMNEST. `http://inet.omnetpp.org/doc/INET/neddoc/index.html`.

[3] Doyle, J.: *Routing TCP/IP, Volume I, Second Edition*. Cisco Press, 2005, iSBN 1-58705-202-4.

[4] Manral, V.; White, R.: Benchmarking Basic OSPF Single router Control Plane Convergence, RFC 4061. `http://www.ietf.org/rfc/rfc4061.txt`, 2005.

[5] Moy, J.: OSPF version 2, RFC 2328. `http://www.ietf.org/rfc/rfc2328.txt`, 1998.

[6] Moy, J. T.: *OSPF Anatomy of an Internet routing protocol*. Addison-Wesley, 1998, iSBN 0-201-63472-4.

[7] Sportack, M. A.: *Směrování v sítích IP*. Computer Press, 2004, iSBN 80-251-0127-4.

[8] Varga, A.: *OMNeT++ Discrete Event Simulation System v4.0 - User Manual*. 2009.

[9] WWW stránky: Cisco System Inc. `http://www.cisco.com`.

[10] Zaballos, A.; Seguí, C.: Analysis and simulation of IGP Routing Protocols. `www.salle.url.edu/ zaballos/opnet/Trame.pdf`, 2006.

Dodatok A

Obsah CD

Umiestnenie na CD	**Popis**
ansa/ethernet	upravený modul EthernetInterface
ansa/InterfaceStateManager	novovytvorený modul InterfaceStateManager
ansa/ospfv2	upravený modul OSPFRouting
ansa/ppp	upravený modul PPPInterface
ansa/ANSARouter.ned	modul smerovača ANSARouter, ktorý využíva všetky upravené a novovytvorené moduly
ansaExamples/MyOSPFCaseStudy	simulácia použitá v sekciách 4.2, 5.2, 5.3 a 5.4
ansaExamples/OSPFBenchmark1	simulácia použitá v sekcii 4.3.1
ansaExamples/OSPFBenchmark2	simulácia použitá v sekcii 4.3.2
ansaExamples/OSPFBenchmark3	simulácia použitá v sekcii 4.3.3
install/omnetpp-4.0rc1.zip	inštalačné súbory OMNeT++
install/INETMANET-20080920.tbz2	inštalačné súbory INET Framework
readme.txt	návod ku správnemu nainštalovaniu súborov z CD

Tabuľka A.1: Obsah CD