ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

DIPLOMOVÁ PRÁCA

Študijný odbor: Počítačové inžinierstvo

Bc. Peter Orság

Implementácia smerovacieho protokolu EIGRP

v balíku Quagga, časť riadenia

Vedúci DP: Ing. Peter Palúch PhD.

Reg. č. 407/2014 **Október 2014**

Ministerské číslo práce: **28360320152407**

Dátum zadania práce: 30.10.2014 *Dátum odovzdania práce: 6.5.2015*

zadanie

Abstrakt

ORSÁG, PETER: Implementácia smerovacieho protokolu EIGRP v balíku Quagga, časť riadenia [Diplomová práca]
- Žilinská univerzita v Žiline. Fakulta riadenia a informatiky
- Vedúci: Ing. Peter Palúch, PhD.
Žilina: Fakulta riadenia a informatiky Žilinská univerzita v Žiline, 2015. 58 strán.

Témou diplomovej práce je vytvoriť si dostatočný prehľad o fungovaní balíka Quagga a následne implementovať príkazové a riadiace rozhranie dynamického smerovacieho protokolu EIGRP do tohto balíka. Práca sa venuje základom fungovania protokolu EIGRP a detailom týkajúcim sa jednotlivých častí práce. V štvrtej kapitole práca dokumentuje vytvorenie samotného rozhrania a programy použité pri jeho tvorbe. Piata kapitola sa zaoberá implementáciou príkazov na riadenie protokolu EIGRP a jeho súčastí, tvorbou príkazov na výpis informácií o fungovaní protokolu EIGRP, tvorbou subsystému na získavanie ladiacich informácií a implementácii exportu prevádzkových informácii o chode protokolu EIGRP pomocou SNMP. V tejto kapitole sú ponúknuté príklady kódu vytvorených funkcií a detaily týkajúce sa jednotlivých bodov zadania.

Kľúčové slová: EIGRP, smerovanie, Quagga, SNMP

Abstract

ORSÁG, PETER: EIGRP routing protocol implementation in the Quagga suite, management part [Diploma thesis]
- University of Žilina. Faculty of Management, Science and Informatics
- Supervisor: Ing. Peter Palúch, PhD.
Žilina: Faculty of Management, Science and Informatics, University of Žilina, 2015. 58 pages.

A topic of the diploma thesis is to make an overview of Quagga suite functionality, and to implement command and management interface for the dynamic routing protocol EIGRP in Quagga suite. This work is dedicated to describe basics of EIGRP functionality and details about each part of the diploma thesis. Fourth part documents creation of the interface itself and programs used in its creation. Fifth chapter specifically describe the implementation of management commands for EIGRP and its components, creation of commands that show functionality of EIGRP protocol, creation of subsystem that gather debug information and implementation of export of operating information about EIGRP protocol using SNMP. This chapter is offering examples of created functions and describe details regarding each point of request.

Keywords: EIGRP, routing, Quagga, SNMP

Poďakovanie:

Tu by som rád poďakoval všetkým, ktorí mi pomáhali pri vypracovaní bakalárskej práce, hlavne vedúcemu mojej práce Ing. Petrovi Palúchovi PhD. za pomoc pri riešení problémov a za poskytnutie cenných informácií a pripomienok.

ČESTNÉ VYHLÁSENIE

Vyhlasujem, že som zadanú diplomovú prácu vypracoval samostatne, pod odborným vedením vedúceho diplomovej práce Ing. Petrom Palúchom PhD. a používal som len literatúru uvedenú v práci.

V Žiline dňa 6. 5. 2015

_____

podpis

Obsah

Zoznam príkladov a tabuliek

Zoznam skratiek

BGP – Border Gateway Protocol
BSD – Berkeley Software Distribution
CEF – Cisco Express Forwarding
CIDR – Classless Inter-Domain Routing
CLNS – ConnectionLess-mode Network Service
DDP – Datagram Delivery Protocol
DTLS – Datagram Transport Layer Security
DUAL – Diffusing Update Algorithm
EIGRP – Enhanced Interior Gateway Protocol
FIB – Forwarding Information Base
GNU – Gnu is Not Unix
HMAC – keyed-Hash Message Authentication Code
IETF – Internet Engineering Task Force
IGP – Interior Gateway Protocol
IGRP – Interior Gateway Routing Protocol
IOS – Internetwork Operating System
IP – Internet Protocol
IPX – Internetwork Packet Exchange
IS-IS – Intermediate System to Intermediate System
MD5 – Message Digest 5
MIB – Management Information Base
MPLS – Multiprotocol Label Switching
MTU – Maximum Transmission Unit
OID – Object IDentifier
OLSR – Optimized Link State Routing
OSI – Open Systems Interconnection
OSPF – Open Shortest Path First
PDM – Protocol Dependent Modules
PDU – Protocol Data Unit
RFC – Request For Comments
RIB – Routing Information Base

RIP – Routing Information Protocol

SHA – Secure Hash Algorithm

SNMP – Simple Network Management Protocol

SRTT – Smooth Round Trip Time

TCL – Tool Command Language

TTL – Time To Live

TLS – Transport Layer Security

UDP – User Datagram Protocol

VPN – Virtual Private Network

VTY – Virtual terminal line

VTYSH – Virtual terminal line shell

Úvod

Quagga je v jednoduchosti softvérový balík zjednocujúci rôzne smerovacie protokoly, ktoré nepodliehajú súkromnému vlastníctvu a sú voľne šíriteľné. Vývoj smerovacích protokolov prebieha už od osemdesiatych rokov a v súčasnosti existuje mnoho alternatívnych protokolov, založených na rozdielnych princípoch od rôznych tvorcov. EIGRP je jedným z protokolov dynamického smerovania, ale ešte nie je oficiálnou súčasťou balíka Quagga.

Táto diplomová práca vznikla ako časť spoločného projektu implementácie smerovacieho protokolu implementácie smerovacieho protokolu EIGRP do balíka Quagga, na ktorom som sa podieľal spolu s Bc. Jánom Janovicom a Bc. Matejom Perinom. Každý člen nášho tímu bol zodpovedný za samostatnú časť celkovej implementácie. Ako vyplýva zo zadania diplomovej práce, našou úlohou bolo najskôr si spraviť prehľad o samotnom balíku Quagga, jeho funkčnosti a syntaxi, ktorú používa. Okrem toho sme museli otestovať aj správanie sa skutočných Cisco zariadení, aby sme správanie sa našej verzie zladili s existujúcimi zariadeniami. Následne som mal navrhnúť a implementovať funkcie, ktorými by bolo možné riadiť funkciu jednotlivých vytvorených komponentov a súčastí protokolu EIGRP ako aj funkcie, ktoré by umožňovali zobraziť výpis o jeho fungovaní a získavať štatistiky ohľadom rôznych aspektov jeho činnosti. Riadiace funkcie EIGRP by mali využívať spôsob zadávania príkazov metódou tzv. *Named Mode*. Všetky tieto funkcie by boli súčasťou klienta Quaggy. Okrem týchto funkcií bolo mojou úlohou oboznámiť sa s fungovaním protokolu SNMP v už existujúcich implementáciách iných sieťových protokolov a ak to bude možné, pokúsiť sa o implementáciu SNMP aj do našej verzie EIGRP. Táto práca je jednou z mnohých klientských aplikácií, ktoré sú súčasťou balíka Quagga, takže pri vypracovávaní práce bolo nutné zladiť našu prácu s už existujúcimi implementáciami sieťových protokolov, keďže v našej práci používame viacero knižničných funkcií, spoločných pre viacero protokolov. Okrem zladenia nášho kódu po funkčnej stránke sme museli používať aj GNU syntax používanú v balíku Quagga, keďže náš projekt bude neskôr zaradený do oficiálnej implementácie Quaggy. Protokol EIGRP, ako už bolo spomínané, existuje na trhu už dlhú dobu a naša implementácia by sa mala čo najviac podobať už existujúcej oficiálnej verzii.

1. EIGRP

Protokol EIGRP je dynamický smerovací protokol vyvinutý firmou Cisco pôvodne ako proprietárny protokol. Jeho špecifikácia bola v roku 2013 zverejnená ako IETF Internet Draft, čo bol aj námet na vypracovanie tejto práce. EIGRP je distance vector protokol, ktorý je vylepšenou verziou svojho predchodcu IGRP, nahradeného v roku 1993.

1.1. Základné vlastnosti protokolu EIGRP

EIGRP podporuje smerovanie typu CIDR a používanie sieťových masiek s rozdielnou dĺžkou. Trasy v smerovacej tabuľke nie sú sumarizované na hranici sieťových tried kým nie je zapnutá automatická sumarizácia. Ak je k dispozícii viac ciest od odosielateľa paketu k príjemcovi, je možné na týchto paralelných linkách vykonávať rozkladanie záťaže. Na zabezpečenie komunikácie používa EIGRP autentifikáciu typu MD5 alebo HMAC-SHA-256. EIGRP ďalej používa rozdielne smerovacie procesy pre rozdielne protokoly ako napríklad IP, IPv6, IPX a Apple Talk (hoci posledné dva protokoly sú spomenuté len kvôli historickej hodnote), použitím protokolovo závislých modulov PDM. Takisto je spätne kompatibilný s IGRP. EIGRP používa na riadenie svojej činnosti algoritmus DUAL, ktorý v súčinnosti s kritériom pre výber bezslučkovej trasy zároveň zabezpečuje že táto trasa je bez smerovacích slučiek.

Oproti ostatným smerovacím protokolom má EIGRP niekoľko výhod. Jednou zo zásadných výhod EIGRP je rýchly čas konvergencie siete pri zmene topológie, jeden z najkratších medzi existujúcimi IGP protokolmi. Ďalšou výhodou je nízke využívanie zdrojov v sieti pri normálnej prevádzke, keďže v stabilnej sieti sú odosielané iba Hello pakety. Ak nastane zmena v sieti, posielané sú iba zmeny v topológii a nie zoznam všetkých sietí. Vďaka týmto výhodám je protokol EIGRP dobre škálovateľný na väčší počet zariadení.

1.2. DUAL

Jednou zo základných súčastí protokolu EIGRP je algoritmus DUAL – principiálne sa jedná o konečný automat so špecifickými reakciami na možné udalosti v sieti. Tento algoritmus sa používa pri výpočtoch najlepšej trasy v sieti, pričom zaisťuje svojimi podmienkami bezslučkovosť. Za cenu mierne zvýšenej prevádzky na sieti je protokol EIGRP s použitím tohto automatu schopný dosahovať najlepšie časy konvergencie siete spomedzi protokolov typu distance vector. Na rozdiel od smerovacích protokolov využívajúcich iba distribuovanú formu Belman-Fordovho algoritmu najkratších ciest, DUAL používa koordinované výpočty medzi smerovačmi, ktoré poznáme ako difúzne výpočty. Difúzne výpočty sa rozrastajú čím viac smerovačov je ovplyvnených zmenou v sieti, a zmenšujú sa vyčlenením zariadení, ktoré ovplyvnené nie sú. Vďaka týmto dynamickým zmenám dokáže mať protokol EIGRP dobrú škálovateľnosť [1].

Cesty v topologickej tabuľke smerovača môžu mať v EIGRP dva stavy, aktívny a pasívny. Cesta môže zmeniť svoj stav pri zmene v topologickej tabuľke. Táto zmena môže byť spôsobená buď chybou na linke do cieľa, chybou zariadenia alebo zvýšením celkovej metriky – dôležité je poznamenať, že ľubovoľná takáto zmena je modelovateľná ako zmena vzdialenosti. V pasívnom stave neprebieha výpočet najkratšej cesty, cesta do cieľa je použiteľná a next hop uvedený do daného cieľa je považovaný za plne použiteľný. Naopak v aktívnom stave sa cesta nachádza, ak smerovač v spolupráci so svojimi susednými smerovačmi hľadá novú najkratšiu cestu do cieľa. Kým je existujúca cesta v aktívnom stave, príjma smerovač naďalej informácie o zmenách do nej, ale nerobí na ich základe smerovacie rozhodnutia kým sa cesta nevráti do pasívneho stavu.

DUAL používa na zabezpečenie bezslučkovosti v sieti pri svojich výpočtoch podmienku s anglickým názvom Feasible Condition (tento názov, prípadne jeho skratku FC budeme používať v texte diplomovej práce, keďže neexistuje priliehavý slovenský preklad). Principiálnou nevýhodou dynamických smerovacích protokolov typu distance vektor je totiž ten fakt, že smerovače realizujú svoje rozhodnutia len na základe svojej obmedzenej znalosti o sieťovej topológii. Vstupnou informáciou pre výpočet najkratších ciest v bežných protokoloch typu distance vector je len zoznam susedov daného smerovača, ceny rozhraní k týmto susedom a zoznam vzdialeností týchto jednotlivých susedov od jednotlivých cieľových sietí. Výpočty najkratších ciest v bežných protokoloch

typu distance vector prebiehajú bez koordinácie medzi smerovačmi. Ak sa v sieti udeje topologická, vzniká riziko, že smerovač, ktorý o tejto topologickej zmene už má informáciu, bude svoj prepočet najkratších ciest zakladať na existujúcich informáciách od svojich susedov, ktorí však ešte o tejto topologickej zmene možno nevedia. Dôsledkom takéhoto rozhodnutia môžu byť prechodné smerovacie slučky. Algoritmus DUAL sa usiluje smerovacím slučkám predchádzať, a práve preto všetky svoje rozhodnutia podmieňuje kontrolou voči podmienke FC. Tým garantuje, že akékoľvek smerovacie rozhodnutie, ktoré urobí, nespôsobí smerovaciu slučku [1].

V súvislosti s touto podmienkou bezslučkovosti je zavedených niekoľko pojmov, ktorými sa riadi.

- Feasible Distance (FD) – Je to historické minimum známej vzdialenosti do konkrétnej cieľovej siete od posledného jej prechodu z aktívneho do pasívneho stavu, vyjadrené celkovou metrikou od aktuálneho smerovača do konkrétnej cieľovej siete. Povedané inak, Feasible Distance je údaj o najmenšej vzdialenosti do daného cieľa, ktorú smerovač doposiaľ zaznamenal od posledného prechodu tejto siete do pasívneho stavu.
- Reported Distance (RD) – Je to aktuálna vzdialenosť vyjadrená celkovou metrikou od konkrétneho next hopu do konkrétnej siete.
- Feasibility Condition (FC)– Postačujúca podmienka pre výber suseda, ktorý preukázateľne nespôsobí smerovaciu slučku pri doručovaní paketov do zvolenej siete. Formulácia FC znie: Každý sused, pre ktorého platí nerovnosť RD < FD, garantovane poskytuje acyklickú trasu do cieľa. Voľnejšie sa FC dá formulovať slovami: Každý sused dane´ho smerovača, ktorý *je* k danému cieľu bližšie, než daný smerovač *je alebo bol* od posledného prechodu daného cieľa do pasívneho stavu, garantovane poskytuje acyklickú trasu do cieľa.
- Successor – Je to next hop, ktorý spĺňa podmienku bezslučkovosti a poskytuje do cieľovej siete najkratšiu cestu, ktorá je momentálne k dispozícii.
- Feasible successor – Je to next hop, ktorý spĺňa podmienku bezslučkovosti, ale nemusí poskytovať najkratšiu cestu do danej siete. Takýto sused môže stále byť použitý na posielanie paketov pri rozkladaní sieťovej záťaže viacerými cestami do tej istej siete. Feasible successor sa môže stať successorom ak dôjde k takej

topologickej zmene, že po zohľadnení zmenených vzdialeností bude nová najkratšia cesta poskytovaná práve súčasným feasible successorom [1].

1.3. Metriky protokolu EIGRP

Pojem „distance vector", alebo vektor vzdialeností, ktorým sa charakterizuje trieda dynamických smerovacích protokolov vrátane EIGRP, vyjadruje fakt, že smerovače v týchto protokoloch si navzájom posielajú vektory, t.j. jednorozmerné polia vzdialeností od jednotlivých známych sietí. Rôzne smerovacie protokoly používajú odlišný spôsob výpočtu vzdialeností, takisto nazývanej aj metrika, do jednotlivých cieľových sietí. EIGRP používa na ohodnotenie vzdialenosti do cieľa 4 čiastkové metriky a tými sú:

- Bandwidth, - Najnižšia hodnota šírky pásma v kilo-bitoch za sekundu na ceste od zdroja k cieľu.
- Delay – Celkové oneskorenie v desiatkach mikrosekúnd na ceste od zdroja k cieľu.
- Reliability – Hodnota od 1 do 255, ktorá predstavuje spoľahlivosť linky (pomer úspešne prijatých ku všetkým prijatým paketom), kde 255 predstavuje spoľahlivú linku.
- Load – Hodnota od 1 do 255, ktorá predstavuje využitie linky, kde 255 predstavuje 100% využitú kapacitu linky.

Piatou čiastkovou metrikou v EIGRP je MTU, ale vo výpočte sa nepoužíva. Pri nastavovaní smerovania v sieti sa neodporúča ovplyvňovať celkovú metriku pomocou iných metrík ako Bandwidth a Delay, pričom parameter Bandwidth by mal vždy zodpovedať reálnej prenosovej kapacite rozhrania, na ktorom je nastavený. Vpľne manipulovateľným parametrom metriky, ktorý môže používať administrátor pre vlastné ovplyvňovanie výberu ciest v EIGRP, zostáva teda Delay. Úplný vzorec na výpočet celkovej metriky vyzerá nasledovne:

$$Metrika = \left(\left[K1 * Bandwidth + \frac{(K2 * Bandwidth)}{256 - Load} + K3 * Delay \right] * \left[\frac{K5}{Reliability + K4} \right] \right) * 256$$

Základné hodnoty nastavené pre jednotlivé K premenné sú:

K1 = 1 K2 = 0 K3 = 1 K4 = 0 K5 = 0

Ak je premenná K5 nastavená ako nula, vo vzorci bude dosadená namiesto nuly jednotka. Metriku bandwitdh EIGRP vypočíta ako bandwidth = (10000000/bandwidth(i)) * 256, kde bandwidth(i) predstavuje najnižšiu hodnotu šírky pásma, nastavenej na všetkých rozhraniach na ceste k cieľu. Metriku delay EIGRP vypočíta ako delay = delay(i), kde delay(i) predstavuje celkové oneskorenie na ceste k cieľu. Pri ponechaní pôvodných K hodnôt na výpočet celkovej metriky sa vzorec na výpočet zjednodušuje na:

$$Metrika = (\ Bandwidth + Delay\ ) * 256$$

Momentálne je snaha do EIGRP zaviesť aj šiestu K premennú, ktorá predstavuje tzv. rozšírené metriky. Pod týmito metrikami sa v súčasnosti rozumie jitter a energy. Jitter predstavuje skreslenie siete v mikrosekundách a je to rozdiel medzi najpomalším a najrýchlejším doručením paketu do cieľa. Energy predstavuje možnosť výberu cesty v sieti podľa spotreby energie. EIGRP však v súčasnosti nevie určiť ani jednu z týchto dvoch metrík, preto je K6 v základnom nastavení nastavené na nulu [1].

1.4. Typy EIGRP paketov

EIGRP používa sedem typov paketov :

- Hello – Hello je paket, ktorý slúži na detekciu EIGRP susedov, udržiavanie vzťahov s nimi a detekciu ich zániku. Hello paket je obvykle odosielaný na multicastovú adresu 224.0.0.10 (IPv4) alebo FF02::A (IPv6). Výnimku tvoria situácie, kedy sú susedia konfigurovaný staticky – v takom prípade sa posiela

unicastový Hello paket na adresu každého staticky definovaného suseda. Hello pakety nie sú potvrdzované.

- Ack – Paket slúžiaci na potvrdenie prijatia iného paketu. Paket Ack v skutočnosti obsahuje iba hlavičku EIGRP paketu a prázdne telo, preto sa niekedy označuje aj ako špeciálny typ Hello paketu – oba totiž majú ten istý číselný kód (tzv. opcode). Paket Ack sú zásadne posielané ako unicast. Pomocou paketov Ack sa potvrdzuje prijatie paketov Update, Query, Reply, SIA-Query a SIA-Reply. Tieto potvrdzované pakety sa preto nazývajú spoľalivé pakety.
- Update – Update je paket na prenos smerovacích informácií – adresa a maska siete a jednotlivé komponenty metriky. Update pakety sa môžu posielať ako unicast aj ako multicast. Pri úvodnej synchronizácii dvoch smerovačov sa Update pakety medzi nimi prenášajú ako unicast pakety. Po úvodnej synchronizácii sa informácie o zmenách posielajú ako multicastové Update pakety. Unicastové Update pakety sa používajú v prípade statických susedov a naviac v prípade tzv. retransmisií – opakovaného odoslania Update paketu susedovi, ktorý jeho prijatie nepotvrdil.
- Query – Query sa používa keď cieľová sieť prejde do aktívneho stavu. Pakety Query sa odosielajú zvyčajne na multicastovú adresu. Unicastová adresa Query paketu je použitá v prípade retransmisií, alebo v prípade statických susedov. Query pakety sú posielané potvrdzovane, ich prijatie teda musí smerovač potvrdiť paketom Ack.
- Reply – Reply paket je odpoveďou na paket Query. V pakete Reply jeho odosielateľ uvádza svoju aktuálnu vzdialenosť do cieľovej siete po zohľadnení informácie, ktorú dostal v pakete Query. Pakety Reply sa posielajú len ako unicasty. Prijatie Reply paketu musí prijímajúci smerovač potvrdiť paketom Ack.
- SIA-Query a SIA-Reply – Pakety používané počas aktívneho stavu a difúzneho výpočtu na overenie, či dlhotrvajúca absencia odpovede Reply je spôsobená komunikačnými problémami medzi konkrétnou dvojicou smerovačov. Ak smerovač v aktívnom stave nedostane do určitého času od konkrétneho suseda odpoveď Reply na svoju žiadosť Query, odošle mu paket SIA-Reply a to i v prípade, že sa sám stále nachádza v aktívnom stave. Ak medzi dvojicou týchto susedov existuje komunikačný problém, na odoslanú SIA-Query sa odpoveď SIA-Reply nevráti, čo odosielateľovi SIA-Query indikuje príčinu problému. Paket SIA-Query aj SIA-Reply sú zásadne unicastové a potvrdzované [1].

2. QUAGGA

Quagga je smerovací softvér založený na princípe daemonov (služieb bežiacich na pozadí operačného systému). V jeho jadre je daemon zebra, ktorý komunikuje priamo s jadrom operačného systému, na ktorom Quagga beží a sprostredkúva mu najkratšie cesty z jednotlivých smerovacích protokolov. V súčasnosti je do balíka Quagga implementovaných 7 smerovacích protokolov:

- RIP (v1, v2, ng)
- OSPF (v2 a v3)
- IS-IS
- BGP
- BABEL
- OLSR
- MPLS

Niektoré implementácie ako napríklad OSPF verzia 3 a IS-IS pre IPv6 majú ešte nedostatky a sú vo fáze vývoja. Quagga obsahuje tiež rozsiahlu knižnicu pre vývoj ďalších klientskych daemonov, ktoré sú správaním podobné už existujúcim implementáciám smerovacích protokolov. Jednotlivé klientské daemony sú prístupné cez sieťovú konzolu VTY. Okrem toho je v Quagge aj nástroj na spoločnú správu všetkých smerovacích protokolov VTYSH, ktorý dokáže nastaviť takmer všetky vlastnosti jednotlivých protokolov na jednom mieste [2].

Projekt Quagga je softvér určený pre operačné systémy unixového typu ako napríklad Linux, Solaris, FreeBSD alebo NetBSD a je rozširovaný pod softvérovou licenciou GNU. Tento projekt dostal svoje meno po vyhynutom druhu zebry. Vznikol ako fork už neexistujúceho projektu GNU Zebra vytvoreného Kunihirom Ishigurom. Základným rozdielom je otvorenosť obidvoch projektov voči prispievateľom do týchto balíkov. Kým Zebra bola uzatvoreným centralizovaným projektom, Quagga je zameraná na otvorený prístup ku komunite, ktorá môže zdieľať a prispievať do balíka. To umožnilo aj vytvorenie tejto práce.

Quagga pozostáva z procesov predstavujúcich smerovacie protokoly, ktoré majú názvy ako ospfd, isisd, eigrpd a podobne. S týmito procesmi sa komunikuje cez interaktívny príkazový riadok VTY a informácie o vykonávaných akciách posúvajú ďalej procesu zebra, ktorý je prostredníkom medzi týmito jednotlivými procesmi a jadrom operačného systému.

Proces zebra udržiava okrem iného kópiu sieťových rozhraní a tabuľku aktuálne aktívnych ciest (FIB). Normálne je za tieto informácie zodpovedné jadro operačného systému, takže zebra komunikuje s týmto jadrom. Toto správanie je možné prestaviť v Quagge tak aby proces zebra komunikoval s inými forwardovacími jadrami ako napríklad s konkrétnym hardvérom. Toto je možné s pomocou tzv. Forwarding Plane manažéra. Okrem toho si proces zebra robí kópiu smerovacích informácií zo smerovacích protokolov a spolu s kópiou FIB tvorí svoju vlastnú smerovaciu databázu RIB. Proces zebra môže taktiež zavádzať statické cesty do svojej RIB. V takom prípade je zebra zodpovedná za výber najlepšej cesty a aktualizovanie svojej FIB databázy. Informácie o tejto FIB databáze dokáže taktiež oznamovať jednotlivým procesom smerovacích protokolov. Rovnako sú klientske procesy Quaggy informované o zmenách stavov svojich sieťových rozhraní [2].

3. SNMP

SNMP je protokol pôvodne zamýšľaný na riadenie zariadení v IP sieťach. Hoci má chopnosťi, aby ním bolo možné zariadenia vzdialene konfigurovať, túto možnosť vzhľadom na rôznorodosť konfiguračných prvkov a ťažkopádny prístup k jednotlivým elementom konfigurácie podporuje iba málo výrobcov sieťových zariadení. Používaný je preto väčšinou iba na vzdialený dohľad a monitorovanie zariadení.

SNMP je komponent programovej sady IP definovanej podľa IETF. Skladá sa zo štandardov na ovládanie siete zahrňujúcich protokol aplikačnej vrstvy a sadu dátových objektov. SNMP poskytuje riadiace dáta vo forme objektov, ktoré reprezentujú resp. modelujú jednotlivé komponenty riadeného systému a hodnota ktorých reprezentuje stav týchto komponentov. O hodnotu týchto objektov môžu riadiace aplikácie požiadať a niekedy ju aj nastaviť ak na to majú oprávnenie [3].

3.1. Základné koncepty fungovania SNMP

V štandardnom použití SNMP vystupuje jeden alebo viac zariadení ako *manažér*. Manažér má za úlohu monitorovanie a riadenie skupiny klientskych zariadení v počítačovej sieti. V každom monitorovanom zariadení beží v každom okamihu softvérový komponent nazývaný *agent*, ktorý ohlasuje informácie manažérovi pomocou SNMP. Protokol SNMP ponúka okrem monitorovania aj možnosť aktívne ovládať správanie sa zariadení v sieti, pomocou zmien objektov, ktoré sprístupňuje. Poskytované objekty sú hierarchicky usporiadané. Toto usporiadanie, ako aj ostatné metadáta, sú popísané v databáze MIB. MIB definuje štruktúru riadiacich dát, ktoré používajú hierarchický priestor obsahujúci objektové identifikátory OID. Každé OID identifikuje objekt, ktorý môže byť čítaný alebo nastavovaný pomocou SNMP. Z vyššie uvedeného vyplýva, že každá sieť v ktorej je použité SNMP, sa skladá z troch základných komponentov: monitorovaného zariadenia, ďalej agenta, ktorý je zapnutý na monitorovanom zariadení a zo softvéru na ovládanie siete, ktorý je zapnutý na manažérovi [4].

SNMP samotné nedefinuje, aké informácie by mal systém poskytovať na monitorovanie. Používa na to rozšíriteľný návrh kde sú dostupné informácie definované databázou MIB. MIB používa zápis definovaný štruktúrou riadiacich informácií vo verzii 2, definovanou v RFC 2578.

3.2. Detaily protokolu SNMP

SNMP pracuje na siedmej (aplikačnej) vrstve OSI modelu. SNMP agent prijíma požiadavky na UDP porte 161. Manažér môže poslať požiadavky z ktoréhokoľvek portu na agentov port 161, ktorý odošle odpoveď späť na zdrojový port manažéra. Manažér dostáva oznamy na porte 162 a agent môže generovať oznamy na ktoromkoľvek dostupnom porte. Ak je pri prevádzke použité TLS alebo DTLS požiadavky a oznamy sú prijímané na portoch 10161 a 10162 [5].

SNMP verzia 1 špecifikuje päť základných typov PDU, vo verzii 2 boli pridané ďalšie dva typy a posledný bol pridaný vo verzii 3. Všetky typy PDU majú nasledovnú základnú štruktúru:

IP hlavička	UDP hlavička	Verzia	Komunita	Typ PDU	ID požiadavky	Chybový status	Chybový index	Objekty

Tabuľka 3.2.1: Základný formát PDU v SNMP

3.3. Typy PDU

- GetRequest – Je to správa od manažéra agentovi, ktorou manažér žiada agenta o odoslanie hodnôt konkrétnych objektov. Odovzdanie požadovaných hodnôt objektov je operácia agenta, ktorá je vykonaná správou Response [6].

- SetRequest – Je to správa od manažéra agentovi, ktorou manažér žiada agenta o zmenu hodnôt konkrétnych objektov. Požadovaná premenná je špecifikovaná v poli premenných a zmeny sú definované v tele požiadavky. Všetky zmeny sú vykonané ako atomické operácie agenta. Odozva s novými hodnotami objektov je vykonaná správou Response[6].

- GetNextRequest – Je to správa od manažéra agentovi, ktorou manažér žiada agenta o odoslanie hodnoty objektu, ktorý lexikograficky existuje za objektom uvedeným v žiadosti. Odozva je správa typu Response s ďalšou premennou v lexikografickom poradí z databázy MIB, ktorá sa nachádza v poli objektov. Celá MIB databáza agenta môže byť preskúmaná opätovným volaním GetNextRequest, pričom prvý GetNextRequest vyžiada objekt s OID 0 a každý ďalší GetNextRequest vyžiada objekt s OID prevzatým z bezprostredne predchádzajúcej odpovede Response [6].

- GetBulkRequest – Optimalizovaná verzia GetNextRequest. Je to správa od manažéra agentovi, ktorá požaduje viacero iterácií GetNextRequest. Odozva je správa typu Response s viacerými iteráciami, špecifikovanými v poli objektov. Toto PDU má 2 špecifické polia a to Non-repeaters a Max-repetitions, ktoré bližšie špecifikujú správanie sa pri návrate hodnôt. Pole Non-repeaters špecifikuje aby bolo z daného objektu vrátená len aktuálna hodnota, keďže niektoré objekty ako napríklad SysUpTime sú pravidelne aktualizované. Max-repetitions špecifikuje koľko hodnôt daného objektu má byť vrátených. Ak je zoznam hodnôt premennej príliš dlhý, bude pri vracaní odozvy orezaný aby sa zmestil do maximálnej dĺžky správy. GetBulkRequest bol pridaný vo verzii 2 [6].

- Response – Je to správa od agenta manažérovi, ktorá obsahuje odozvu na správy GetRequest, SetRequest, GetNextRequest, GetBulkRequest a InformRequest. Hlásenia o chybách, ktoré sa môžu vyskytnúť sa nachádzajú v poliach chybový status a chybový index. Vo verzii 1 bolo toto pole nazvané GetResponse, aj keď fungovalo ako odozva pre Get aj Set príkazy [6].

- Trap – Je to asynchrónne hlásenie od agenta manažérovi. SNMP Trap umožňuje agentovi oznámiť manažéra o dôležitých udalostiach pomocou nevyžiadanej SNMP správy. Obsahuje aktuálnu hodnotu SysUpTime, OID identifikujúce typ trapu

a ďalšie možné nastavenia, ktoré sa nachádzajú v poli objektov. Adresovanie príjemcu je riešené odlišne pre rôzne aplikácie, typicky pomocou objektov na riadenie trapov v databáze MIB. Formát správy trap sa vo verzii SNMP 2 zmenil [6].

- InformRequest – Je to potvrdený asynchrónny oznam. Toto PDU bolo zavedené v SNMP verzii 2 a bolo pôvodne vymyslené na komunikáciu medzi dvomi manažérmi. Neskoršie implementácie rozšírili pôvodný úmysel a povolili komunikáciu od agenta manažérovi [7]. Komunikácia medzi dvomi manažérmi bola možná už vo verzii 2 použitím trapov, ale keďže SNMP funguje obyčajne cez UDP komunikáciu, doručenie správ a ani informovanie o strate paketov nie je garantované. InformRequest ponúka riešenie v tom, že posiela naspäť potvrdenie o doručení správy [6].

3.4. AgentX

AgentX alebo plným názvom Agent Extensibility Protocol je sieťový protokol, ktorý dovoľuje ovládanie SNMP objektov, definovaných rôznymi procesmi, pomocou jedného nadradeného agenta. Agenti, ktorí exportujú objekty cez AgentX sa nazývajú subagenti. Štandard AgentX nedefinuje len protokol AgentX ale aj procedúry používané pri spracúvaní SNMP správ subagentami.

3.5. Verzie a vývoj SNMP

Verzia 1

Je pôvodnou implementáciou SNMP protokolu. Funguje nad protokolmi UDP, IP, CLNS, DDP a IPX. Prvá verzia bola veľmi kritizovaná za svoju slabú bezpečnosť [8]. Prihlasovanie sa klientov bolo vykonávané iba použitím tzv. komunitného reťazca, ktorý fungoval ako heslo a bol prenášaný v textovom tvare. Protokol SNMP bol pôvodne navrhovaný ako dočasný protokol, ktorý potrebuje ďalší vývoj aby mohol byť použiteľný vo väčšej miere cez internet.

Verzia 2

SNMP verzia 2 obsahuje zlepšenia v oblastiach výkonnosti, bezpečnosti, diskrétnosti a komunikácie medzi dvomi manažérmi. Obsahuje dva nové príkazy GetBulkRequest a GetNextRequest na získavanie veľkého množstva dát jediným príkazom. Takisto obsahuje nový bezpečnostný systém založený na tzv. účastníkoch (parites), ale tento systém bol všeobecne považovaný za veľmi zložitý [8] a v praxi sa neujal.Do SNMPv2 bol preto kvôli svojej jednoduchosti a rozšírenosti prenesený spôsob autentifikácie pomocou komunitných reťazcov zo SNMPv1. Variant SNMPv2 využívajúci komunitné reťazce pre autentifikáciu sa označuje ako SNMPv2c. SNMPv2u je akýmsi kompromisom medzi dvomi spomínanými verziami, ktorá sa snaží ponúkať vyššiu bezpečnosť ako komunitný model z verzie 1 a zároveň má nižšiu zložitosť ako účastnícky model z pôvodnej verzie 2. Tento mechanizmus bol neskôr použitý ako jedna z dvoch bezpečnostných štruktúr použitých vo verzii 3.

Verzie 1 a 2 sú vzájomne nekompatibilné, keďže používajú rozdielne hlavičky a PDU formáty. SNMP verzia 2 takisto obsahuje dve protokolové operácie naviac oproti verzii 1. Jediné možnosti na komunikáciu medzi týmito dvomi protokolmi je buď s použitím proxy agentov alebo bilinguálnych sieťových systémov.

Verzia 3

Tretia verzia nezavádza okrem kryptografickej bezpečnosti z hľadiska protokolu nič nové, avšak kvôli novým textovým dohodám a terminológii pôsobí dojmom radikálne nového protokolu [3]. Verzia 3 zavádza primárne zlepšenia v oblasti bezpečnosti a zlepšenie diaľkového ovládania. Vyriešené boli hlavne problémy týkajúce sa globálneho využívania SNMP, účtovníctva a spravovania chýb. Čo sa týka bezpečnosti, novinky obsahujú silnú autentifikáciu a šifrovaný prenos dát. Okrem toho boli pridané novinky týkajúce sa administrácie a to označovanie pôvodcov oznámení a proxy forwardery. Každá SNMP správa vo verzii 3 obsahuje bezpečnostné parametre zakódované do reťazca bajtov, ktorého význam závisí od použitého bezpečnostného modelu. Od roku 2004 IETF považuje SNMPv3 za úplný internetový štandard, čo je najvyšší stupeň vývoja pre RFC.

4. Programovací jazyk a vývojové prostredie

Balík Quagga je písaný v jazyku C a používa štýl formátovania kódu GNU, ktorého sme sa pri tvorbe svojej práce držali aj my. Pri práci som používal vývojové prostredie Eclipse od neziskovej open source komunity Eclipse Foundation. Prostredie Eclipse ponúka možnosť prepojiť projekt s internetovým repozitárom a zosúladiť tak prácu viacerých ľudí na tom istom projekte, čo sme pri práci na našom projekte využili aj my. Ako repozitár používame GitHub, cez ktorý sme si vytvorili vlastnú vetvu balíka Quagga pod názvom EIGRP Developement. Náš projekt môžete nájsť na adrese: https://github.com/janovic/Quagga-EIGRP.

4.1. Programy použité pri tvorbe diplomovej práce

Okrem vývojového prostredia Eclipse a repozitára GitHub som pri svojej práci používal viacero programov, niektoré sú nutné na chod nami vytvoreného protokolu, niektoré slúžia na kontrolu a overenie funkčnosti kódu. Na preklad a použitie balíka Quagga sú potrebné niektoré balíky, ktoré sa dajú stiahnuť a nainštalovať v každej verzii Linuxu. Týmito balíkmi sú:

- autoconf – Balík na generovanie konfiguračných skriptov.
- automake – Nástroj na generovanie špeciálnych make súborov používaných v balíku Quagga.
- libtools – Podporný knižničný skript.
- texinfo – Oficiálny formát dokumentácií ku GNU projektom.
- gawk – Nástroj na špeciálne formátovanie textu.
- dia – Nástroj na tvorbu diagramov.

Jedným z najdôležitejších súčastí nášho vývojového prostredia je softvér Dynamips. Je to program na emuláciu Cisco smerovačov. Podporuje Cisco IOS pre hardvérové platformy 1700, 2600, 3600, 3700 a 7200. Ako doplnok k tomuto programu používame add-on Dynagen, ktorý umožňuje načítanie sieťovej topológie cez konfiguračné súbory.Jednou zo

súčastí mojej práce je aj protokol SNMP, preto k jeho chodu potrebujeme SNMP agenta AgentX, ktorý sa dá takisto voľne stiahnuť ako balík v každej verzii Linuxu.

Na testovanie funkčnosti samotného protokolu používame program Wireshark, ktorým odchytávame komunikáciu medzi zariadeniami. Okrem toho je Wireshark užitočný aj pri zisťovaní správania sa skutočných Cisco zariadení, podľa ktorých sme vytvárali tento projekt.

5. Vlastná implementácia

Náš projekt EIGRP obsahuje v súčasnosti 39 súborov, ktoré obsahujú buď priamo zdrojový kód, súbory potrebné pre preklad projektu, alebo súbory potrebné pre chod niektorých súčastí našej práce, ako napríklad databáza MIB, ktorú používam pri svojej implementácii protokolu SNMP pre EIGRP. Moja práca je prevažne obsiahnutá v zdrojových súboroch:

- eigrp_dump.c
- eigrp_dump.h
- eigrp_snmp.c
- eigrp_snmp.h
- eigrp_vty.c
- eigrp_vty.h

Okrem väčšiny kódu obsiahnutého v týchto súboroch som občas potreboval pridať časť kódu aj do iných súborov. Napríklad niektoré konštanty a štruktúry používané v mojej časti kódu sú definované v súboroch eigrp_structs.h a eigrp_const.h, výpisy ladiacich informácií sú v súbore eigrp_packet.c alebo príkazy vytvorené v rámci mojej práce bolo treba doplniť do niektorých knižničných funkcií a definícií v adresári lib. Jednotlivé súčasti mojej práce ako aj ich rozdelenie je podrobne popísané v nasledujúcich kapitolách.

5.1. Implementácia konzolových príkazov pre riadenie činnosti protokolu EIGRP

V prvom rade bolo potrebné vytvorenie príkazového rozhrania (Command Line Interface, CLI), čiže základných príkazov na ovládanie protokolu EIGRP. Keďže som s balíkom Quagga nemal pred tvorbou tejto práce žiadne skúsenosti, musel som najskôr preskúmať a otestovať funkčnosť kódu na už vytvorených sieťových protokoloch, ako napríklad OSPF a pochopiť syntax vytvorených funkcií. Časť protokolu týkajúca sa konzoly a konzolových príkazov je v balíku Quagga obsiahnutá v súboroch vty.c (napríklad konzolové príkazy protokolu OSPF sú súčasťou súboru ospf_vty.c), preto som ponechal konvenciu a konzolové príkazy pre náš protokol sa nachádzajú taktiež v súbore eigrp_vty.c. Všetky implementované príkazy na riadenie činnosti protokolu EIGRP a jeho komponentov, ako aj node do ktorého patria, sú v zozname nižsie.

show_ip_eigrp_interfaces_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_neighbors_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_topology_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_neighbors_detail_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_interfaces_detail_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_topology_all_links_cmd	ENABLE_NODE/ VIEW_NODE
show_ip_eigrp_topology_detail_cmd	ENABLE_NODE/ VIEW_NODE
eigrp_if_delay_cmd	INTERFACE_NODE*
eigrp_if_bandwidth_cmd	INTERFACE_NODE*
eigrp_if_ip_holdinterval_cmd	INTERFACE_NODE*
no_eigrp_if_ip_holdinterval_cmd	INTERFACE_NODE*
eigrp_if_ip_hellointerval_cmd	INTERFACE_NODE*
no_eigrp_if_ip_hellointerval_cmd	INTERFACE_NODE*
interface_desc_cmd	INTERFACE_NODE*
no_interface_desc_cmd	INTERFACE_NODE*
eigrp_authentication_mode_cmd	INTERFACE_NODE*
no_eigrp_authentication_mode_cmd	INTERFACE_NODE*
eigrp_authentication_keychain_cmd	INTERFACE_NODE*

no_eigrp_authentication_keychain_cmd	INTERFACE_NODE*
eigrp_ip_summary_address_cmd	INTERFACE_NODE*
no_eigrp_ip_summary_address_cmd	INTERFACE_NODE*
eigrp_redistribute_source_metric_cmd	INTERFACE_NODE
no_eigrp_redistribute_source_metric_cmd	INTERFACE_NODE
router_eigrp_cmd	CONFIG_NODE
no_router_eigrp_cmd	CONFIG_NODE
eigrp_address_family_cmd	EIGRP_NAMED_NODE
no_eigrp_address_family_cmd	EIGRP_NAMED_NODE
eigrp_address_family_interface_cmd	EIGRP_NAMED_AF_NODE
no_eigrp_address_family_interface_cmd	EIGRP_NAMED_AF_NODE
eigrp_address_family_interface_settings_cmd	EIGRP_NAMED_AF_INTERFACE_NODE
eigrp_network_cmd	EIGRP_NODE**
no_eigrp_network_cmd	EIGRP_NODE**
eigrp_variance_cmd	EIGRP_NODE
no_eigrp_variance_cmd	EIGRP_NODE
eigrp_router_id_cmd	EIGRP_NODE
no_eigrp_router_id_cmd	EIGRP_NODE
eigrp_passive_interface_cmd	EIGRP_NODE*
no_eigrp_passive_interface_cmd	EIGRP_NODE*
eigrp_timers_active_cmd	EIGRP_NODE**
no_eigrp_timers_active_cmd	EIGRP_NODE**
eigrp_metric_weights_cmd	EIGRP_NODE**
no_eigrp_metric_weights_cmd	EIGRP_NODE**
eigrp_maximum_paths_cmd	EIGRP_NODE
no_eigrp_maximum_paths_cmd	EIGRP_NODE
eigrp_neighbor_cmd	EIGRP_NODE**
no_eigrp_neighbor_cmd	EIGRP_NODE**

Príkazy označené hviezdičkou sú prístupné taktiež cez Named Mode pod EIGRP_NAMED_AF_INTERFACE_NODE a príkazy označené dvomi hviezdičkami pod EIGRP_NAMED_AF_NODE.

Na príklade 5.1.1 je možné vidieť ukážku kódu z jednej mnou vytvorených funkcií, ktorá predstavuje jeden konzolový príkaz. Každý konzolový príkaz musí mať definovaný node smerovača, v ktorom bude tento príkaz sprístupnený. Nody smerovača predstavujú vrcholy príkazového stromu. Každý z týchto vrcholov má k dispozícii osobitný zoznam príkazov a dá sa preň nastaviť vlastná úroveň privilégií. Napríklad Enable Node, ktorý slúži na nastavovanie globálnych nastavení klienta zariadenia, alebo náš EIGRP Node, ktorý obsahuje všetky nastavenia týkajúce sa protokolu EIGRP a podobne. DEFUN je makro používané v balíku Quagga a predstavuje definíciu základnej verzie príkazu. Skladá sa z 5 základných častí a tými sú:

- Názov samotného príkazu. Pod týmto názvom je príkaz používaný v zdrojových súboroch.

- Názov konkrétnej verzie príkazu. Niektoré príkazy môžu mať totiž viacero variantov. Pre každý z týchto variantov musí platiť, že majú rovnaký názov príkazu, ale rozličný názov verzie príkazu aby sa vedelo rozlíšiť čo má daný príkaz vykonávať a aké možnosti má ponúkať. Takisto je možné každý z variantov použiť v inom node smerovača. Ako som spomenul vyššie DEFUN je makro pre definovanie základnej verzie príkazu s funkčnou časťou. Pre definovanie iných verzií toho istého príkazu sa používa makro ALIAS, ktoré neobsahuje funkčnú časť, ako je možné vidieť na príklade 5.1.1.

```
ALIAS (show_ip_eigrp_topology,
       show_ip_eigrp_topology_detail_cmd,
       "show ip eigrp topology
       (A.B.C.D|A.B.C.D/nn|detail|summary)",
       SHOW_STR
       IP_STR
       "IP-EIGRP show commands\n"
       "IP-EIGRP topology\n"
       "Netwok to display information about\n"
       "IP prefix <network>/<length>, e.g.,
       192.168.0.0/16\n"
       "Show all links in topology table\n"
       "Show a summary of the topology table\n")
```

```
DEFUN (show_ip_eigrp_topology,
        show_ip_eigrp_topology_cmd,
        "show ip eigrp topology",
        SHOW_STR
        IP_STR
        "IP-EIGRP show commands\n"
        "IP-EIGRP topology\n")
        { ... }
```

Príklad 5.1.1:Ukážka základného typu príkazu DEFUN a alternatívneho typu toho istého príkazu ALIAS

- Definície možností pre daný príkaz. Tretí riadok predstavuje znenie konkrétnych možností ponúknutých v konzole pri písaní príkazov. Podľa tejto definície vie klient dopĺňať príkazy a ponúkať nápovedu pre dané príkazy. Ako je možné vidieť na predchádzajúcom príklade, hlavný príkaz `show_ip_eigrp_topology_cmd` má presne zadefinované znenie bez možnosti modifikácie, kým alternatívny príkaz `show_ip_eigrp_topology_detail_cmd` ponúka možnosť po zadaní základného príkazu, výber konkrétnych sietí v topológii alebo detailný výpis. Keďže alternatívne nastavenia sú zaradené pod makrom ALIAS, bude príkaz platný aj bez výberu jednej z ďalších možností.
- Popis k daným príkazom. Štvrtý riadok predstavuje informácie o príkazoch, alebo nápovedu k formátu zadávania údajov do príkazu.
- Funkčná časť. Jadro príkazu v ktorom sa vykonávajú funkcie k daným príkazom. Je ohraničené ako klasická funkcia zloženými zátvorkami.

Každý príkaz musí byť definovaný pre nejaký konkrétny node smerovača, aby klient vedel, kde má byť daný príkaz použiteľný. Takisto treba definovať vzhľad príkazového riadku pre každý node. Na príklade 5.1.2 je možné vidieť ukážku zavádzania spomínaného príkazu `show_ip_eigrp_topology_cmd` do klienta Quaggy a na príklade 5.1.3 definíciu nodu a jeho vzhľadu v príkazovom riadku pre EIGRP node.

```
install_element(ENABLE_NODE, &show_ip_eigrp_topology_cmd);
install_element(VIEW_NODE, &show_ip_eigrp_topology_cmd);
```

Príklad 5.1.2: Zavedenie konzolových príkazov do nodov smerovača

```
static struct cmd_node eigrp_node =
{
  EIGRP_NODE,
  "%s(config-router)# ",
  1
};
```

Príklad 5.1.3: Definovanie nodu smerovača a jeho vzhľadu v príkazovom riadku

5.1.1. EIGRP named mode

Špeciálnou požiadavkou pri tvorbe riadiacich príkazov bolo aj využitie tzv. EIGRP Named Mode. V klasickom prístupe sa pri aktivácii protokolu EIGRP na smerovači zadáva hneď autonómny systém, v ktorom bude daná inštancia EIGRP fungovať. Named Mode ponúka možnosť definovať EIGRP proces pomocou unikátneho mena a sprehľadniť tak do určitej miery konfiguráciu. Pre klasický mód tiež platí, že z EIGRP node sa dajú nastaviť len globálne parametre protokolu. Named Mode ponúka možnosť EIGRP nastavení na rozhraniach, v topológii alebo pre celú rodinu adries či už IPv4, alebo IPv6. Na Cisco smerovačoch je v IOS od verzie 15 a ďalej stále možné používať obidva prístupy, starý aj nový, preto som túto možnosť ponechal aj v našom prípade. Pri aktivovaní prístupu EIGRP Named Mode musí reťazec predstavujúci názov EIGRP procesu obsahovať text, ktorý nezačína číslom, keďže číslo na danom mieste pôsobí ako identifikátor autonómneho systému v klasickom móde.

```
eigrpd(config)# router eigrp
  <1-65535>  Autonomous system
  WORD           EIGRP Virtual-instance name
eigrpd(config-router)#
```

```
  address-family  Enter Address Family command node
  end             End current mode and change to enable
                  mode
  exit            Exit current mode and down to previous
                  mode
  help            Description of the interactive help
                  system
  list            Print command list
  no              Negate a command or set its default
  quit            Exit current mode and down to previous
                  mode
  show            Show running system information
  write           Write running configuration to memory,
                  network, or terminal
eigrpd(config-router)# address-family
  ipv4            Address family IPv4
  ipv6            Address family IPv6

eigrpd(config-router)# address-family ipv4
  autonomous-system           Specify address-family autonomous
                              system number
eigrpd(config-router)# address-family ipv4 autonomous-system
  <1-65535> Autonomous system
eigrpd(config-router-af)#
  af-interface    Enter Address Family interface configuration
  end             End current mode and change to enable
                  mode
  exit            Exit current mode and down to previous
                  mode
  help            Description of the interactive help
                  system
  list            Print command list
  no              Negate a command or set its default
  quit            Exit current mode and down to previous
                  mode
  show            Show running system information
  write           Write running configuration to memory,
                  network, or terminal
```

Príklad 5.1.1.1: Ukážka nastavení EIGRP Named Mode v konzole

Na príklade 5.1.1.1 je možné vidieť spôsob špecifikovania nastavení v Named Mode cez príkazový riadok. Na rozdiel od pôvodného módu sa v Named Mode nastavuje číslo autonómneho systému až po zadaní rodiny adries. Ako je možné vidieť na príklade 5.1.1.1, rodina adries ako následne aj rozhranie pre danú rodinu adries má vlastný node.

5.2. Implementácia príkazov pre získavanie informácií o stave a činnosti protokolu EIGRP

Pre správnu činnosť každého protokolu je nutné, mať k dispozícii nástroj na kontrolu činnosti daného protokolu. Na tento účel slúžia príkazy `show`, ktoré poskytujú rôzne informácie o chode protokolov. V našom prípade som vytvoril tri základné príkazy, ktoré sú detailne popísané v tejto časti.

show ip eigrp neighbors

Tento príkaz poskytuje informácie o EIGRP susedoch s ktorými je toto zariadenie vo vzťahu full adjacency. Za príkazom je možné zadať číslo autonómneho systému EIGRP, ktorého susedia sa majú zobraziť. Tento výpis poskytuje nasledovné informácie:

```
EIGRP neighbors for AS(1)

H  Address  Interface  Hold   Uptime   SRTT   RTO   Q   Seq
                       (sec)           (ms)         Cnt  Num
0 10.0.0.2  veth0       11      0       0      2    0     1
```

Príklad 5.2.1: Ukážka výpisu príkazu show ip eigrp neighbors na konzole

- Autonómny systém EIGRP, v ktorom máme spojenie so susedmi uvedenými v tabuľke nižšie.
- IP adresu suseda.
- Rozhranie cez ktoré poznáme suseda.
- Holdtime, alebo čas, za ktorý sa musí tento sused ozvať smerovaču Hello paketom, inak bude označený ako nedostupný a vyradený.
- Uptime, alebo čas, ktorý uplynul od vytvorenia susedstva s daným susedom.
- SRTT, alebo čas v milisekundách, ktorý uplynie od odoslania potvrdzovanej EIGRP správy susedovi a prijatia Ack správy potvrdzujúcej doručenie.
- RTO, alebo čas v milisekundách, ktorý bude smerovač čakať kým znovu odošle neúspešne odoslanú EIGRP správu.
- Poradovník paketov, ktorý označuje počet EIGRP paketov, odoslaných danému susedovi, ktoré ešte neboli potvrdené.
- Sekvenčné číslo poslednej prijatej potvrdzovanej EIGRP správy od suseda.

Údaje zobrazované v tomto výpise sú obsiahnuté v štruktúre `struct eigrp_neighbor` uvedenej na príklade 5.2.2.

```
struct eigrp_neighbor
{
  /* EIGRP rozhranie pod ktorým je tento sused */
  struct eigrp_interface *ei;

  u_char state; /* stav suseda */

  u_int32_t recv_sequence_number; /* Posledné prijaté sequence Number */
  u_int32_t init_sequence_number;

  /* Ak je paket nepotvrdený snažíme sa ho poslať 16 krát */
  u_char retrans_counter;

  struct in_addr src; /* Zdrojová adresa suseda */

  u_char os_rel_major;        // major číslo verzie
  u_char os_rel_minor;        // minor číslo verzie
  u_char tlv_rel_major;       // major číslo verzie tlv
  u_char tlv_rel_minor;       // minor číslo verzie tlv
```

```
  /* K hodnoty */
  u_char K1;
  u_char K2;
  u_char K3;
  u_char K4;
  u_char K5;
  u_char K6;

  /* Hodnoty časovačov */
  u_int16_t v_holddown;

  /* Vlákna */
  struct thread *t_holddown;

  struct eigrp_fifo *retrans_queue;
  struct eigrp_fifo *multicast_queue;

  u_int32_t crypt_seqnum;           /* Kryptografické sekvenčné číslo */
};
```

Príklad 5.2.2: Ukážka štruktúry s informáciami o EIGRP susedovi

show ip eigrp interfaces

Tento príkaz zobrazuje zoznam rozhraní, na ktorých je aktivovaný protokol EIGRP. Tak isto ako v predchádzajúcom prípade aj za týmto príkazom môže nasledovať číslo autonómneho systému EIGRP. V takom prípade budú zobrazené rozhrania pre daný konkrétny autonómny systém. Ako súčasť tohto príkazu som pridal oproti oficiálnej Cisco implementácii dva ďalšie údaje. Sú to prvé dva údaje a to Bandwidth a Delay, keďže tieto parametre sú z hľadiska EIGRP rozhrania podstatné a ich zaradenie do show výpisu je vylepšenie pôvodnej funkcionality. Kompletný výpis poskytuje informácie zobrazené na príklade 5.2.3.

```
EIGRP interfaces for AS(1)

Interface  Bandwidth  Delay  Peers  Xmit Queue   Mean
                                   Un/Reliable  SRTT
veth0      10000000   1000   0      0 / 0        0

Pacing Time    Multicast    Pending  Hello    Holdtime
Un/Reliable    Flow Timer   Routes
0              0            0        5        15
```

Príklad 5.2.3: Ukážka výpisu show ip eigrp interfaces na konzole

- Autonómny systém EIGRP, v ktorom máme nastavené dané rozhrania.
- Konkrétne rozhranie, pre ktoré je nastavený protokol EIGRP.
- Hodnota šírky pásma daného rozhrania.
- Hodnota oneskorenia nastaveného na danom rozhraní.
- Peers, alebo počet priamo pripojených EIGRP susedov k danému rozhraniu.
- Transmit queue alebo front potvrdzovaných a nepotvrdzovaných EIGRP správ na danom rozhraní čakajúcich na odoslanie.
- Mean SRTT alebo priemerný čas v milisekundách, ktorý uplynie od odoslania potvrdzovanej EIGRP správy ľubovoľnému susedovi na tomto rozhraní a prijatia potvrdzujúcej správy Ack od neho naspäť.
- Pacing time, alebo intervaly, v ktorých by sa mali odosielať potvrdzované a nepotvrdzované správy na danom rozhraní, aby bolo možné riadiť celkový objem EIGRP prevádzky na rozhraní.
- Multicast flow timer alebo maximálny čas, do ktorého sa očakávajú od všetkých susedov potvrdenia na odoslaný multicastový, potvrdzovaný. Pending routes, alebo počet neoznámených ciest, ktoré sú obsiahnuté v paketoch čakajúcich na odoslanie vo fronte paketov pre dané rozhranie.
- Hello interval, ktorý predstavuje čas, po ktorom sú odosielané na rozhraní Hello pakety.
- Hold time, alebo interval v ktorom očakáva smerovač od suseda Hello paket. Po uplynutí tohto intervalu považuje smerovač suseda za nefunkčného.

Údaje zobrazované v tomto výpise sú obsiahnuté v štruktúre `struct eigrp_interface` uvedenej na príklade 5.2.4.

```
struct eigrp_interface
{
  /* Inštancia EIGRP pre toto rozhranie */
  struct eigrp *eigrp;

  /* Informácie o rozhraní zo Zebry */
  struct interface *ifp;

  /* Zásobník správ na odoslanie */
  struct eigrp_fifo *obuf;

  /* To which multicast groups do we currently belong? */
  u_char multicast_memberships;


  /* Nastavené premenné */
  struct eigrp_if_params *params;

  /* Typ EIGRP siete */
  u_char type;

  struct prefix *address; /* Prefix rozhrania */
  struct connected *connected; /* Smerník na pripojené rozhrania */
  struct list *nbrs; /* Zoznam EIGRP susedov */
  struct thread *t_hello; /* Hello časovač */
  int on_write_q;

  /* Access-list */
  struct access_list *list[EIGRP_FILTER_MAX];

  /* Prefix-list */
  struct prefix_list *prefix[EIGRP_FILTER_MAX];

  /* Route-mapa */
  struct route_map *routemap[EIGRP_FILTER_MAX];
```

```
  u_int32_t hello_in; /* Počítadlo prijatých Hello správ */
  u_int32_t update_in; /* Počítadlo prijatých Update správ */
  u_int32_t query_in; /* Počítadlo prijatých Querry správ */
  u_int32_t reply_in; /* Počítadlo prijatých Reply správ */
  u_int32_t hello_out; /* Počítadlo odoslaných Hello správ */
  u_int32_t update_out; /* Počítadlo odoslaných Update správ */
  u_int32_t query_out; /* Počítadlo odoslaných Query správ */
  u_int32_t reply_out; /* Počítadlo odoslaných Reply správ */
  u_int32_t siaQuery_in; /* Počítadlo prijatých SIA-Query správ */
  u_int32_t siaQuery_out; /* Počítadlo odoslaných SIA-Query správ */
  u_int32_t siaReply_in; /* Počítadlo prijatých SIA-Reply správ */
  u_int32_t siaReply_out; /* Počítadlo odoslaných SIA-Reply správ */
  u_int32_t ack_out; /* Počítadlo odoslaných Ack správ */
  u_int32_t ack_in; /* Počítadlo prijatých Ack správ */

  u_int32_t crypt_seqnum;           /* Kryptografické sekvenčné číslo */
};
```

Príklad 5.2.4: Ukážka štruktúry s informáciami o EIGRP rozhraní

show ip eigrp topology

Tento príkaz slúži na vypísanie informácií o stave topologickej tabuľky a ciest v nej. Rovnako ako v predchádzajúcich príkazoch je možné zadať konkrétny autonómny systém, pre ktorý sa cesty zobrazia. Tento výpis poskytuje nasledujúce informácie:

```
EIGRP Topology Table for AS(1)/ID(192.168.0.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R -
Reply
       r - reply Status, s - sia Status

P  10.0.0.0/24, 1 successors, FD is 256256, serno: 0
       via Connected, veth0
```

Príklad 5.2.5: Ukážka výpisu show ip eigrp topology na konzole

- Zoznam označení pre siete. Passive označuje stabilnú sieť, pre ktorú sa nevykonávajú EIGRP výpočty. Naopak active označuje sieť, do ktorej sa hľadá cesta a prebiehajú EIGRP výpočty. Update, query a reply znamenajú, že bol odoslaný tento typ paketu do danej siete. Reply status znamená, že bol odoslaný query paket do danej siete a čaká sa na odozvu.
- Konkrétnu sieť a jej masku.
- Successors, alebo počet susedov, cez ktorých sa dá dostať do konkrétnej siete.
- Feasible distance, alebo najlepšia metrika, ktorú má smerovač do danej siete cez niektorého zo susedov.
- Via označuje cez akého suseda nám bola oznámená cesta do danej siete.
- Metriky do danej siete. Prvé číslo označuje metriku tohto smerovača do danej siete, druhé označuje metriku suseda, ktorý túto cestu ohlásil.
- Rozhranie, cez ktoré poznáme informácie o týchto cestách.

Údaje zobrazované v tomto výpise sú obsiahnuté v štruktúre `struct eigrp_topology` uvedenej na príklade 5.2.6.

```
struct eigrp_prefix_entry
{
  struct list *entries, *rij;
  u_int32_t fdistance;                    // Feasible Distance
  u_int32_t rdistance;                    // Reported Distance
  u_int32_t distance;                     // Distance
  struct eigrp_metrics reported_metric;   // Posielaná Reported Distance

  u_char nt;                              // Typ siete
  u_char state;                           // Stav FSM
  u_char af;                              // Address family
  u_char req_action;                      // Požadovaná akcia

  struct prefix_ipv4 *destination_ipv4;   // Smerník na štruktúru s IPv4
                                             adresou
  struct prefix_ipv6 *destination_ipv6;   // Smerník na štruktúru s IPv6
                                             adresou

  // Ak je typ siete REMOTE_EXTERNAL, smerník bude mať odkaz na jej
     externé TLV //
```

```
  struct TLV_IPv4_External_type *extTLV;

  u_int64_t serno; /* Sériové číslo tohto záznamu. Zvyšuje sa s každou
                      zmenou záznamu */
};
```

Príklad 5.2.6: Ukážka štruktúry s informáciami o topológii

Na príklade 5.2.7 je možné vidieť implementáciu príkazu `show_ip_eigrp_topology` popisovaného v predchádzajúcej časti kapitoly. Na začiatok si funkcia overí stav EIGRP a v prípade, že nie je aktívne, príkaz vypíše oznam a končí. Nasleduje výpis hlavičky daného výpisu, ktorý obsahuje nápovedu ku kódom, ako bolo opísané na predchádzajúcich stranách a hlavičky jednotlivých stĺpcov. Nakoniec sa tabuľka naplní údajmi z topologickej tabuľky, ktorú máme zoradenú do zoznamov. V jednom sa nachádza zoznam všetkých známych sietí, ktoré boli smerovaču oznámené, v druhom všetci successori a feasible successori do každej z nich.

```
{
  struct eigrp *eigrp;
  struct listnode *node, *nnode, *node2, *nnode2;
  struct eigrp_prefix_entry *tn;
  struct eigrp_neighbor_entry *te;

  eigrp = eigrp_lookup ();
  if (eigrp == NULL)
  {
    vty_out (vty, " EIGRP Routing Process not enabled%s",
             VTY_NEWLINE);
    return CMD_SUCCESS;
  }

  show_ip_eigrp_topology_header (vty, eigrp);


  for(ALL_LIST_ELEMENTS(eigrp->topology_table,node,nnode,tn))
  {
    show_ip_eigrp_prefix_entry (vty,tn);
    for (ALL_LIST_ELEMENTS(tn->entries,node2,nnode2,te))
```

```
    {
            if (((te->flags & EIGRP_NEIGHBOR_ENTRY_SUCCESSOR_FLAG) ==
                  EIGRP_NEIGHBOR_ENTRY_SUCCESSOR_FLAG)||
                ((te->flags & EIGRP_NEIGHBOR_ENTRY_FSUCCESSOR_FLAG) ==
                  EIGRP_NEIGHBOR_ENTRY_FSUCCESSOR_FLAG))
                        show_ip_eigrp_neighbor_entry (vty, eigrp, te);
      }
    }
  return CMD_SUCCESS;
}
```

Príklad 5.2.7: Ukážka kódu jednej z funkcií na získavanie informácií o stave EIGRP

Samotný výpis sa nakoniec vykonáva funkciami `show_ip_eigrp_prefix_entry` a `show_ip_eigrp_neighbor_entry`, ktoré vypisujú na konzolu VTY spôsobom podobným klasickému výpisu pomocou `print()`. Príklad 5.2.8 predstavuje výpis jedného riadku, v ktorom musí byť na začiatku zadefinované kam má daný výpis smerovať (`vty`) a na zabezpečenie ďalšieho pokračovania konzoly na novom riadku slúži `VTY_NEWLINE` na konci.

```
vty_out (vty, "%-7s%s, %s%s"," ","via
Connected",eigrp_if_name_string (te->ei), VTY_NEWLINE);
```

Príklad 5.2.8: Ukážka kódu funkcie na výpis textu na konzolu VTY

5.3. Tvorba subsystému pre získavanie ladiacich informácií o činnosti protokolu EIGRP

Ďalšou dôležitou súčasťou tejto práce je získavanie ladiacich informácií o činnosti protokolu, tzv debugging. Tieto informácie sa používajú či už na kontrolu správneho fungovania protokolu a jeho nastavení, alebo na zisťovanie príčin nesprávneho fungovania. V našej práci používame dva základné príkazy pre ladenie, a to `debug eigrp packet` a `debug eigrp transmit`. Pri ladení paketov má užívateľ možnosť vybrať si niektorý z typov EIGRP správ a smerovač bude následne vypisovať informácie o danom type na konzolu vždy, keď nastane situácia definovaná príkazom. Pri ladení prenosu (`transmit`), budú vypisované informácie o prenosových údajoch všetkých EIGRP paketov, ako napríklad zdrojová adresa, alebo TTL paketu. Ako nastavenia rôznych situácií pri ladení sa dá nastaviť výpis pri odoslaní daného typu paketu, jeho prijatí, alebo pri obidvoch situáciách. Ak nie je zadaná špecifická situácia, automaticky je nastavený stav výpisu pri prijatí aj odoslaní daného typu paketu. Pri obidvoch typoch ladenia je možnosť detailného aj obyčajného výpisu, ktorým možno ovládať rozsah a zložitosť ladenia.

Ako prvé bolo treba vytvoriť funkciu ktorá by zapínala a vypínala ladiace výpisy podľa predvolených parametrov. Na to slúžia makrá `conf_debug_eigrp_packet`, ktoré sa používa pri zadaní príkazu v config node a `term_debug_eigrp_packet`, ktoré sa používa pri zadaní príkazu v enable node. Tieto makrá používajú dva parametre, ktoré predstavujú príznaky pre ladenie. Prvý z nich je príznak označujúci typ EIGRP paketu a druhý označuje situáciu, pri ktorej má výpis nastať. Pri príkaze `debug eigrp transmit`, ktorý nemá špeciálne nastavenie typu paketu, keďže vypisuje informácie o všetkých paketoch, sa prvý z parametrov nastaví na 0. Príklad funkcií ktorá zapína a vypína jeden typ ladenia je na príklade 5.3.1.

```
#define CONF_DEBUG_PACKET_ON(a, b)
      conf_debug_eigrp_packet[a] |= (b)
#define CONF_DEBUG_PACKET_OFF(a, b) conf_debug_eigrp_packet[a] &= ~(b)
#define TERM_DEBUG_PACKET_ON(a, b)
      term_debug_eigrp_packet[a] |= (b)
```

```
#define TERM_DEBUG_PACKET_OFF(a, b) term_debug_eigrp_packet[a] &= ~(b)
#define DEBUG_PACKET_ON(a, b) \
    do { \
      CONF_DEBUG_PACKET_ON(a, b); \
      TERM_DEBUG_PACKET_ON(a, b); \
    } while (0)
#define DEBUG_PACKET_OFF(a, b) \
    do { \
      CONF_DEBUG_PACKET_OFF(a, b); \
      TERM_DEBUG_PACKET_OFF(a, b); \
    } while (0)
```

Príklad 5.3.1: Ukážka kódu funkcií na zapínanie a vypínanie jedného typu debugu

Okrem týchto funkcií bolo treba vytvoriť ďalšiu funkciu, ktorá kontroluje či je zapnutý správny druh ladenia a príznakov pre daný výpis. Na to slúži makro `IS_DEBUG_EIGRP_PACKET`, ktoré má rovnaké parametre ako predchádzajúce funkcie.

```
#define IS_DEBUG_EIGRP_PACKET(a, b) \
      (term_debug_eigrp_packet[a] & EIGRP_DEBUG_ ## b)
```

Príklad 5.3.2: Ukážka kódu funkcie na kontrolu aktivácie konkrétneho typu ladenia a jeho príznakov

Ako už bolo spomínané, funkcie na zapínanie, vypínanie a kontrolu ladenia používajú dva príznaky na špecifikáciu konkrétneho nastavenia. Na príklade 5.3.3 sú predstavené príznaky paketov a na príklade 5.3.4 príznaky označujúce konkrétne situácie a detailné zobrazenie.

```
extern unsigned long term_debug_eigrp_packet[];
#define EIGRP_DEBUG_UPDATE              0x01
#define EIGRP_DEBUG_REQUEST             0x02
#define EIGRP_DEBUG_QUERY               0x04
```

```
#define EIGRP_DEBUG_REPLY                0x08
#define EIGRP_DEBUG_HELLO                0x10
#define EIGRP_DEBUG_PROBE                0x40
#define EIGRP_DEBUG_ACK                  0x80
#define EIGRP_DEBUG_SIAQUERY             0x200
#define EIGRP_DEBUG_SIAREPLY             0x400
#define EIGRP_DEBUG_STUB                 0x800
#define EIGRP_DEBUG_PACKETS_ALL          0xfff
```

Príklad 5.3.3: Ukážka zadefinovania príznakov pre pakety

```
extern unsigned long term_debug_eigrp_transmit;
#define EIGRP_DEBUG_SEND                 0x01
#define EIGRP_DEBUG_RECV                 0x02
#define EIGRP_DEBUG_SEND_RECV            0x03
#define EIGRP_DEBUG_PACKET_DETAIL        0x04
```

Príklad 5.3.4: Ukážka zadefinovania flagov pre špecifické situácie a detailný výpis

Na príklade 5.3.5 je príklad jedného z výpisov ladiacich informácií. Je to výpis ktorý sa zobrazí pri prijatí paketu a zapnutom výpise prenosových informácií `debug eigrp transmit`. Výpis obsahuje informácie o type prijatého paketu, jeho veľkosti, rozhranie cez ktoré bol prijatý, zdrojovú adresu a cieľovú adresu.

```
if (IS_DEBUG_EIGRP_TRANSMIT(0, RECV))
    zlog_debug("Received [%s] length [%u] via [%s] src [%s] dst [%s]",
             LOOKUP(eigrp_packet_type_str, opcode), length,
             IF_NAME(ei), inet_ntoa(iph->ip_src, inet_ntoa(iph->ip_dst));
```

Príklad 5.3.5: Ukážka výpisu prenosových ladiacich informácií o prijatom pakete

V nasledujúcom zozname sú uvedené všetky implementované príkazy pre ladenie protokolu EIGRP a možnosti, ktoré ponúkajú.

- show_debugging_eigrp_cmd – Vypisuje aký druh ladenia a príznakov je aktívny.
- debug_eigrp_packets_all_cmd – Aktivuje ladenie konkrétnych typov paketov uvedených v príklade 5.3.3.
- no_debug_eigrp_packets_all_cmd – Deaktivuje ladenie konkrétnych typov paketov
- debug_eigrp_packets_send_recv_cmd - Aktivuje ladenie konkrétnych typov paketov uvedených v príklade 5.3.3 a ponúka možnosti výberu situácie, pri ktorej nastane výpis (odoslanie alebo prijatie paketu).
- no_debug_eigrp_packets_send_recv_cmd – Deaktivuje ladenie konkrétnych typov paketov pre konkrétne situácie.
- debug_eigrp_packets_send_recv_detail_cmd – Aktivuje ladenie konkrétnych typov paketov uvedených v príklade 5.3.3, ponúka možnosti výberu situácie, pri ktorej nastane výpis (odoslanie alebo prijatie paketu) a ponúka možnosť zapnúť detailný výpis.
- no_debug_eigrp_packets_send_recv_detail_cmd – Deaktivuje detailné ladenie konkrétnych typov paketov pre konkrétne situácie.
- debug_eigrp_transmit_cmd – Aktivuje ladenie všetkých typov paketov a ponúka výpis transportných informácií pri konkrétnej situácii (odoslanie alebo prijatie paketu).
- debug_eigrp_transmit_detail_cmd – Deaktivuje ladenie všetkých typov paketov a konkrétnej situácie.
- no_debug_eigrp_transmit_cmd – Aktivuje ladenie všetkých typov paketov a ponúka detailný výpis transportných informácií pri konkrétnej situácii (odoslanie alebo prijatie paketu).
- no_debug_eigrp_transmit_detail_cmd – Deaktivuje ladenie všetkých typov paketov a konkrétnej situácie s detailným výpisom.

5.4. Implementácia exportu prevádzkových informácií a štatistík cez protokol SNMP

Posledným zadaním práce bolo oboznámiť sa so samotným protokolom SNMP, jeho implementáciou v už existujúcich smerovacích protokoloch zahrnutých do balíka Quagga a pokúsiť sa o vlastnú implementáciu exportu prevádzkových informácií z protokolu EIGRP pomocou SNMP protokolu. Ako už bolo spomenuté v tretej kapitole, najskôr bolo treba nainštalovať a nastaviť SNMP agenta, pomocou ktorého budú informácie medzi zariadeniami prenášané. Tento balík sa v Linuxe dá stiahnuť a nainštalovať pod názvom net-snmp. Jeho súčasťou sú aj dva konfiguračné súbory umiestnené v adresári `/etc/snmp` a sú to `snmp.conf` a `snmpd.conf`, Tieto predstavujú konfiguračné súbory pre manažéra a pre klientov. V týchto súboroch nastavíme všetky potrebné údaje pre chod SNMP ako verziu protokolu, heslá na komunikáciu klientov so serverom, používateľov, adresy s ktorými majú zariadenia nadväzovať spojenie, alebo umiestnenie databázy MIB z ktorej bude protokol čerpať údaje. V našom prípade treba povoliť používanie SNMP. Toto dosiahneme príkazom `configure –enable-snmp=agentx` pri preklade a kompilácii balíka Quagga. Ako ďalší krok treba následne stiahnuť EIGRP MIB do nášho zariadenia. Toto môžeme spraviť buď ručne, alebo pomocou softvéru ako napríklad `snmp-mibs-downloader`. Tieto MIB databázy môžeme umiestniť prakticky hocikde, ale ich štandardné umiestnenie je v adresári `/usr/local/share/snmp/mibs`. V tomto bode máme pripravené SNMP pre použitie v našej práci.

Ako prvé je treba zaregistrovať EIGRP MIB, ktorú bude náš klient používať v zdrojovom súbore Quaggy. MIB je vo svojej podstate schémou databázy – popisuje jednotlivé spravované objekty, ich typy a ich identifikátory (OID), ktoré môže daný agent pomocou SNMP protokolu poskytnúť. Ako som pri svojej práci neskôr zistil, databáza MIB je v niektorých prípadoch nekompromisná a ak nedostane presne ten istý formát údajov, aký si žiada, odmietne dostupné údaje s chybovým hlásením. Hodnoty premenných do SNMP sú vracané buď ako konkrétna hodnota, názov, alebo ako `SNMP_INTEGER`. Tento integer môže okrem skutočného integeru znamenať hodnoty, ktoré majú pre danú premennú vopred zadefinovaný význam. Napríklad pri premennej `router_id_type`,

ktorá má ako návratové hodnoty presne zadefinované hodnoty 0:unknown, 1:ipv4, 2:ipv6, 3:ipv4z, 4:ipv6z a 16:dns. Preto nestačí vedieť akú hodnotu má každý SNMP objekt vracať, ale aj či nemá SNMP pre tento údaj definované špeciálne návratové hodnoty.

V príklade 5.4.1 je možné vidieť kód pre inicializáciu SNMP a zadefinovanie EIGRP MIB do zdrojového kódu Quaggy.

```
void
  eigrp_snmp_init ()
  {
    eigrp_snmp_iflist = list_new ();
    smux_init (eigrp_om->master);

     REGISTER_MIB("ciscoEigrpMIB",eigrp_variables,variable,
                  eigrp_oid);
  }
```

Príklad 5.4.1: Ukážka kódu pre inicializáciu SNMP a zaregistrovanie EIGRP MIB v balíku Quagga

Každý smerovací protokol má tiež zadefinované svoje presné OID, pod ktorým ho vie SNMP identifikovať. EIGRP OID je 1.3.6.1.4.1.9.9.449.1. Za týmto číslom ďalej nasledujú OID jednotlivých súčastí EIGRP, pre ktoré vie SNMP poskytovať informácie. 1 predstavuje informácie o VPN, 2 prenosové štatistiky, 3 informácie o topológii, 4 informácie o EIGRP susedoch a 5 informácie o EIGRP rozhraniach. Každá z týchto konkrétnych typov informácií je ešte podadresárom pod table a entry adresármi takže úplná adresa k jednotlivým položkám bude napríklad 1.3.6.1.4.1.9.9.449.1.1.1.1 pre VPN štatistiky a ďalšie číslo za tým predstavuje konkrétny údaj. Každý z údajov, ktoré sú dostupné cez SNMP, bolo treba zadefinovať v našom zdrojovom súbore aby Quagga vedela poskytnúť správne informácie. V príklade 5.4.2 je možné vidieť príklad definovania premenných pre jeden zo spomínaných typov informácií. Nie je pravidlom, aby premenné nasledovali podobne ako v príklade. V niektorých prípadoch, ako napríklad v premenných pre rozhrania, začínajú OID od trojky a niektoré OID sú v poradí vynechané.

```
/* EIGRP peer entry */
#define EIGRPHANDLE                                 1
#define EIGRPPEERADDRTYPE                           2
#define EIGRPPEERADDR                               3
#define EIGRPPEERIFINDEX                            4
#define EIGRPHOLDTIME                               5
#define EIGRPUPTIME                                 6
#define EIGRPSRTT                                   7
#define EIGRPRTO                                    8
#define EIGRPPKTSENQUEUED                           9
#define EIGRPLASTSEQ                                10
#define EIGRPVERSION                                11
#define EIGRPRETRANS                                12
#define EIGRPRETRIES                                13
```

Príklad 5.4.2: Definícia OID jednotlivých premenných tak, ako sú zadefinované v databáze MIB

```
/* EIGRP peer variables */
  {EIGRPHANDLE,INTEGER,NOACCESS,eigrpPeerEntry,
   4, {4, 1, 1, 1}},
  {EIGRPPEERADDRTYPE,IPADDRESSTYPE,RONLY,eigrpPeerEntry,
   4, {4, 1, 1, 2}},
  {EIGRPPEERADDR,IPADDRESS,RONLY, eigrpPeerEntry,
   4, {4, 1, 1, 3}},
  {EIGRPPEERIFINDEX,INTERFACEINDEXORZERO,RONLY,
   eigrpPeerEntry,
   4, {4, 1, 1, 4}},
  {EIGRPHOLDTIME,INTEGER, RONLY, eigrpPeerEntry,
   4, {4, 1, 1, 5}},
  {EIGRPUPTIME, STRING, RONLY, eigrpPeerEntry,
   4, {4, 1, 1, 6}},
  {EIGRPSRTT, INTEGER, RONLY, eigrpPeerEntry,
   4, {4, 1, 1, 7}},
  {EIGRPRTO, INTEGER, RONLY, eigrpPeerEntry,
   4, {4, 1, 1, 8}},
```

```
{EIGRPPKTSENQUEUED, INTEGER, RONLY, eigrpPeerEntry,
 4, {4, 1, 1, 9}},
{EIGRPLASTSEQ, INTEGER, RONLY, eigrpPeerEntry,
 4, {4, 1, 1, 10}},
{EIGRPVERSION, STRING, RONLY, eigrpPeerEntry,
 4, {4, 1, 1, 11}},
{EIGRPRETRANS, COUNTER, RONLY, eigrpPeerEntry,
 4, {4, 1, 1, 12}},
{EIGRPRETRIES, INTEGER, RONLY, eigrpPeerEntry,
 4, {4, 1, 1, 13}},
```

Príklad 5.4.3: Ukážka definície konkrétnych parametrov pre SNMP premenné

V príklade 5.4.3 je možné vidieť definície EIGRP premenných, ktoré je možné odoslať cez protokol SNMP. Premenné v príklade patria do skupiny informácií o EIGRP susedoch. Všetky kategórie a príkazy patria pod jednu veľkú štruktúru `struct variable eigrp_variables = {...}`. Jednotlivé údaje v zložených zátvorkách prestavujú:

- Názov premennej.
- Typ premennej.
- Typ prístupu. Ten označuje či je danú premennú možné iba čítať, čítať a zapisovať do nej, alebo k nej nie je cez SNMP prístup.
- Funkcia, ktorá bude vracať údaje z Quaggy
- Počet čísel predstavujúcich vrcholy stromu MIB, po ktorých sa ku konkrétnej premennej dá dostať od základného OID protokolu EIGRP
- OID konkrétnej premennej ktoré nasleduje za OID protokolu EIGRP

Každý typ premennej má definovanú osobitnú funkciu, ktorá vracia informácie protokolu SNMP, keďže niektoré typy premennej môžu vyžadovať osobitné parametre. Na príklade 5.4.4 je uvedený príklad funkcie `eigrpPeerEntry`.

```
static u_char *
  eigrpPeerEntry (struct variable *v, oid *name, size_t
                  *length, int exact, size_t *var_len,
                  WriteMethod **write_method)
  {
        struct eigrp *eigrp;
        struct eigrp_interface *ei;
        struct listnode *node, *nnode;
        struct eigrp_neighbor *nbr;
        struct in_addr nbr_addr;
        unsigned int ifindex;
        eigrp = eigrp_lookup ();
        if (smux_header_generic (v, name, length, exact,
            var_len, write_method)== MATCH_FAILED)
            return NULL;

        memset (&nbr_addr, 0, sizeof (struct in_addr));
        ifindex = 0;

        nbr = eigrpNbrLookup (v, name, length, &nbr_addr,
                              &ifindex, exact);
        if (! nbr)
            return NULL;
        ei = nbr->ei;
        if (! ei)
            return NULL;
```

Príklad 5.4.4: Príklad jednej z funkcií, ktorá vracia informácie o smerovači naspäť do SNMP

Funkcia `eigrpPeerEntry` najskôr získa informáciu o autonómnom systéme EIGRP. Následne porovná požadované OID s existujúcimi OID a v prípade nezhody prerušuje vykonávanie príkazu, keďže nie je aké informácie vracať. Po vyhľadaní EIGRP suseda, o ktorom budú vyhľadávané informácie, je potrebné vyhľadať podľa OID konkrétnu premennú a vrátiť hodnotu. Na to slúži `switch`, ktorý je uvedený v príklade 5.4.5. Podobným spôsobom sú riešené všetky premenné.

```
switch (v->magic)
{
    ...
case EIGRPHELLOSSENT:           /* 3 */
/* Hello packets output count */
          if (eigrp)
          {
                counter = 0;
                for (ALL_LIST_ELEMENTS (eigrp->eiflist,
                      node, nnode, ei))
                {
                      counter += ei->hello_out;
                }
                return SNMP_INTEGER (counter);
          }
          else
              return SNMP_INTEGER (0);
          break;
     ...
```

Príklad 5.4.5: Príklad príkazu na vrátenie jednej konkrétnej premennej protokolu SNMP

Aktuálne vieme pomocou protokolu SNMP vracať informácie o nasledovných SNMP objektoch:

EIGRPASNUMBER - 2.1.1.1 – Autonómny systém EIGRP.
EIGRPNBRCOUNT - 2.1.1.2 – Počet EIGRP susedov.
EIGRPHELLOSSENT - 2.1.1.3 – Počítadlo odoslaných Hello paketov.
EIGRPHELLOSRCVD - 2.1.1.4 – Počítadlo prijatých Hello paketov.
EIGRPUPDATESSENT - 2.1.1.5 – Počítadlo odoslaných Update paketov.
EIGRPUPDATESRCVD - 2.1.1.6 – Počítadlo prijatých Update paketov.
EIGRPQUERIESSENT - 2.1.1.7 – Počítadlo odoslaných Query paketov.
EIGRPQUERIESRCVD - 2.1.1.8 – Počítadlo prijatých Query paketov.
EIGRPREPLIESSENT - 2.1.1.9 – Počítadlo odoslaných Reply paketov.
EIGRPREPLIESRCVD - 2.1.1.10 – Počítadlo prijatých Reply paketov.
EIGRPACKSSENT - 2.1.1.11 – Počítadlo odoslaných Ack paketov.

EIGRPACKSRCVD - 2.1.1.12 – Počítadlo prijatých Ack paketov.
EIGRPSIAQUERIESSENT - 2.1.1.15 – Počítadlo odoslaných SIA-Query paketov.
EIGRPSIAQUERIESRCVD - 2.1.1.16 – Počítadlo prijatých SIA-Query paketov.
EIGRPASROUTERIDTYPE - 2.1.1.17 – Typ Router ID.
EIGRPASROUTERID - 2.1.1.18 – Router ID.
EIGRPPEERCOUNT - 5.1.1.3 – Počet EIGRP susedov na konkrétnom rozhraní.
EIGRPAUTHKEYCHAIN - 5.1.1.23 – Meno autentifikačného kľúča pre konkrétne rozhranie

Čísla uvedené za názvami objektov sú OID objektov, ktoré nasledujú v OID strome za OID protokolu EIGRP. Objekty ktoré sme schopní zatiaľ vracať, zatiaľ tvoria prevažne počítadlá paketov. Tieto počítadlá nie sú súčasťou protokolu EIGRP a musel som ich implementovať do existujúcich štruktúr. Počítadlá s OID 2.1.1.3 až 2.1.1.16 sú získavané zo štruktúry `struct eigrp_interface`, ktorá je uvedená na príklade 5.2.4.

SNMP ponúka možnosť vypísať informácie zadané konkrétnym OID, ale aj všetky OID ktoré sú dostupné ako nižšia vetva pod iným OID. Takisto je možnosť zadať špecifikáciu pre niektoré OID, ako napríklad jedného konkrétneho suseda pre výpis `EIGRPPEERADDR`, ktorý bez špecifikovania konkrétneho poradového čísla suseda v zozname EIGRP susedov, vypíše všetkých susedov v zozname. Príklad takejto funkcie je uvedený v príklade 5.4.6. Táto funkcia získava zoznam susedov pre SNMP výpisy. Z parametrov zadaných do funkcie sa vie táto funkcia správať dvomi spôsobmi. Funkcia mení svoje správanie podľa premennej `exact`, ktorá má hodnotu 1, ak bol zadaný konkrétny sused a 0 ak nie. Ak bol zadaný konkrétny sused, vráti štruktúru s informáciami tohto suseda pomocou funkcie `eigrp_snmp_nbr_lookup`, ak nie, bude funkcia vracať štruktúry všetkých susedov pomocou funkcie `eigrp_snmp_nbr_lookup_next`, tieto funkcie je možné vidieť na príklade 5.4.7 a 5.4.8.

```
static struct eigrp_neighbor *
eigrpNbrLookup (struct variable *v, oid *name, size_t *length, struct
in_addr *nbr_addr, unsigned int *ifindex, int exact)
{
  unsigned int len;
  int first;
  struct eigrp_neighbor *nbr;
  struct eigrp *eigrp;

  eigrp = eigrp_lookup ();

  if (! eigrp)
    return NULL;

  if (exact)
    {
      if (*length != v->namelen + IN_ADDR_SIZE + 1)
            return NULL;
      oid2in_addr (name + v->namelen, IN_ADDR_SIZE,nbr_addr);
      *ifindex = name[v->namelen + IN_ADDR_SIZE];

      return eigrp_snmp_nbr_lookup (eigrp,nbr_addr,ifindex);
    }
  else
  {
      first = 0;
      len = *length - v->namelen;

      if (len <= 0)
            first = 1;

      if (len > IN_ADDR_SIZE)
            len = IN_ADDR_SIZE;

      oid2in_addr (name + v->namelen, len, nbr_addr);
      len = *length - v->namelen - IN_ADDR_SIZE;
      if (len >= 1)
       *ifindex = name[v->namelen + IN_ADDR_SIZE];
      nbr = eigrp_snmp_nbr_lookup_next (nbr_addr, ifindex,
                                        first);
      if (nbr)
```

```
       {
        *length = v->namelen + IN_ADDR_SIZE + 1;
        oid_copy_addr (name+v->namelen,nbr_addr,IN_ADDR_SIZE);
        name[v->namelen + IN_ADDR_SIZE] = *ifindex;
        return nbr;
       }
     }
  return NULL;
```

Príklad 5.4.6: Ukážka kódu funkcie získavajúcej konkrétneho EIGRP suseda z parametrov SNMP príkazu

```
static struct eigrp_neighbor *
eigrp_snmp_nbr_lookup_next  (struct  in_addr  *nbr_addr,  unsigned  int
*ifindex, int first)
{
  struct listnode *node, *nnode, *node2, *nnode2;
  struct eigrp_interface *ei;
  struct eigrp_neighbor *nbr;
  struct route_node *rn;
  struct eigrp_neighbor *min = NULL;
  struct eigrp *eigrp = eigrp;


  eigrp = eigrp_lookup ();

  for (ALL_LIST_ELEMENTS (eigrp->eiflist, node, nnode, ei))
    {
      for (ALL_LIST_ELEMENTS (ei->nbrs, node2, nnode2, nbr))
          {
                  if (first)
                    {
                        if (! min)
                          min = nbr;
                        else if (ntohl (nbr->src.s_addr) < ntohl (min->
                                  src.s_addr))
                             min = nbr;
                    }
                  else if (ntohl (nbr->src.s_addr) > ntohl (nbr_addr->
```

```
                        s_addr))
                    {
                        if (! min)
                          min = nbr;
                        else if (ntohl (nbr->src.s_addr) < ntohl (min->
                                 src.s_addr))
                          min = nbr;
                    }
            }
      }
    if (min)
      {
        *nbr_addr = min->src;
        *ifindex = 0;
        return min;
      }
    return NULL;
}
```

Príklad 5.4.7: Ukážka kódu funkcie získavajúcej všetkých EIGRP susedov zo všetkých Rozhraní

```
static struct eigrp_neighbor *
eigrp_snmp_nbr_lookup (struct eigrp *eigrp, struct in_addr *nbr_addr,
                   unsigned int *ifindex)
{
  struct listnode *node, *nnode, *node2, *nnode2;
  struct eigrp_interface *ei;
  struct eigrp_neighbor *nbr;

  for (ALL_LIST_ELEMENTS (eigrp->eiflist, node, nnode, ei))
    {
      for (ALL_LIST_ELEMENTS (ei->nbrs, node2, nnode2, nbr))
        {
          if (IPV4_ADDR_SAME (&nbr->src, nbr_addr))
            {
                  return nbr;
            }
        }
```

```
    }
  return NULL;
}
```

Príklad 5.4.8: Ukážka kódu funkcie získavajúcej konkrétneho zadaného EIGRP suseda

Funkcia na príklade 5.4.7 vracia štruktúru `eigrp_neighbor` pre funkciu `eigrpNbrLookup` na príklade 5.4.6. Prehľadá všetky rozhrania a všetkých EIGRP susedov na nich, ktorých následne zoradí podľa adries a vráti susedov. Funkcia `eigrp_snmp_nbr_lookup` na príklade 5.4.8 jednoducho nájde a vráti jedného konkrétne zadaného EIGRP suseda, ak takáto adresa existuje.

Záver

Cieľom tejto práce bolo vytvorenie príkazového a riadiaceho rozhrania dynamického smerovacieho protokolu EIGRP a implementovanie exportu prevádzkových informácií o chode protokolu EIGRP pomocou protokolu SNMP. Doteraz som vo svojej práci implementoval celkovo 57 príkazov, z čoho 39 príkazov tvorí príkazy pre riadenie EIGRP a jeho komponentov a sú súčasťou súboru eigrp_vty.c. Ďalších 7 príkazov tvoria príkazy pre získavanie informácií o stave a činnosti protokolu EIGRP a sú tiež súčasťou súboru eigrp_vty.c. Posledných 11 príkazov tvorí príkazy pre získavanie ladiacich informácií o chode protokolu EIGRP a sú súčasťou súboru eigrp_dump.c. Podarilo sa aj plne implementovať systém exportu prevádzkových informácií a štatistík, ktorý dokáže v súlade s databázou EIGRP MIB vracať nielen informácie o konkrétnom požadovanom OID, ale aj vyčleniť konkrétne zadaného suseda v rámci OID, ktorý normálne vracia informácie o všetkých dostupných susedoch. Súčasťou tohto bodu práce je zadefinovanie všetkých OID, ktoré je schopný smerovač vracať, 5 funkcií pre vracanie objektov z piatich vetiev OID stromu, 3 funkcie pre zisťovanie zadaného suseda a vrátenie informácií o ňom a funkcia pre aktivovanie SNMP a zaregistrovanie EIGRP MIB databázy. Okrem toho som zadefinoval výpisy ladiacich informácií v súbore eigrp_packet.c a niektoré konštanty a štruktúry používané v práci v súboroch eigrp_structs.h, eigrp_const.h a knižničnom adresári Quaggy lib. Celkovo moja práca zasahuje do desiatich súborov a obsahuje zhruba 4000 riadkov kódu.

V rámci tejto práce som sa snažil pokryť každý z bodov zadania do čo najväčšej miery, ale keďže protokol EIGRP je veľmi rozsiahly, stále je priestor pre ďalšie zlepšovanie.

Použitá literatúra

[1] Enhanced Interior Gateway Routing Protocol – Internet draft

[2] Introduction to the Quagga Routing Suite – Paul Jakma, David Lamparter

[3] RFC 3411 – An architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks

[4] Douglas R. Mauro & Kevin J. Schmidt – Essential SNMP (first edition), 2001, Sebastopol, O'Reilly & Associates

[5] RFC 6356 – Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP), Section 10

[6] RFC 1098 - A Simple Network Management Protocol (SNMP)

[7] J. Case, K. McCloghrie, M. Rose, S. Waldbusser – "RFC 1448 - Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)", April 1993

[8] Security in SNMPv3 versus SNMPv1 or v2c, 29.11.2010