ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

IMPLEMENTÁCIA SMEROVACIEHO PROTOKOLU EIGRP V BALÍKU QUAGGA, TRANSPORTNÁ ČASŤ

Diplomová práca

Študijný odbor:	Aplikované sieťové inžinierstvo – sieťová infraštruktúra
Katedra:	Katedra informačných sietí
Vedúci diplomovej práce:	Ing. Peter Palúch, PhD.

ŽILINA 2015

Bc. Ján Janovic

Abstrakt

JANOVIC, Ján: Implementácia smerovacieho protokolu EIGRP v balíku Quagga, transportná časť [diplomová práca]

Žilinská Univerzita v Žiline, Fakulta riadenia a informatiky, Katedra informačných sietí.

Vedúci: Ing. Peter Palúch, PhD.

Stupeň odbornej kvalifikácie: Inžinier v odbore Aplikované sieťové inžinierstvo – Sieťová infraštruktúra

FRI ŽU v Žiline, 2015 - 73 s.

Cieľom diplomovej práce je implementácia pôvodne proprietárneho smerovacieho protokolu EIGRP, ako nového modulu pre softvérový balík Quagga. V prvej časti práce je detailne opisaný protokol EIGRP, jeho história a základná funkcionalita. V druhej časti je rozobratá architektúra softvérového balíku Quagga, jeho inštalácia a konfigurácia, ako aj licencia, pod ktorou je balík distribuovaný. Nasleduje popis samotnej implementácie a návrhu nového protokolového modulu. Posledná kapitola sa zaoberá testovaním hotového riešenia.

Kľúčové slová: Protokol, EIGRP, Smerovanie, Quagga, Linux, Unix, Implementácia, Softvér, C, GIT

Abstract

JANOVIC, Ján: Implementation of EIGRP routing protocol in Quagga, transport part [Diploma Thesis]

University of Zilina, Zilina, Faculty of Management Science and Informatics, Department of Infocom Networks.

Supervisor: Ing. Peter Palúch, PhD.

Level of specialized Qualification: Master in Applied Network Engineering – Network Infrastructure

FRI, University of Zilina, 2015 - 73 s.

The main goal of this diploma thesis is to create implementation of the EIGRP routing protocol as a new softvare module for Quagga routing suite. In the first part, there is EIGRP described and explained in deatail, its history and the basic functionality. In the second part, the software architecture of Quagga routing suite is broken down as well as its installation/configuration process and its distribution license. As the next part, the design and implementation of the new routing module is described. The last chapter is about testing phase of the resulting code.

Keywords: Protocol, EIGRP, Routing, Quagga, Linux, Unix, Implementation, Software, C, GIT

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že som zadanú diplomovú prácu vypracoval samostatne, pod odborným vedením vedúceho diplomovej práce Ing. Petra Palúcha, PhD. a používal som len literatúru uvedenú v práci

Súhlasím so zapožičiavaním a zverejnením práce**.**

V Žiline dňa 6. 05. 2015 Bc. Ján Janovic

POĎAKOVANIE

Chcem sa úprimne poďakovať Ing. Petrovi Palúchovi PhD. za pomoc pri príprave a realizácií tejto diplomovej práce. Za všetky pripomienky a odborné vedenie, bez ktorého by žiadny z dosiahnutých výsledkov nebol možný. Rovnako tak svojej rodine a snúbenici Lenke za nekonečnú podporu, aj vďaka ktorej som vždy nachádzal nadšenie a chuť do ďalšej práce.

Obsah

Zoznam obrázkov

Zoznam skratiek

BGP	Border Gateway Protocol
BSD	Berkeley Software Distribution
CCIE	Cisco Certified Internetwork Expert
CLI	Command Line Interface
CLNS	Connectionless Network Service
DARPA	Defense Advanced Research Projects Agency
DUAL	Diffusing Update Algorithm
EIGRP	Enhanced Interior Gateway Routing Protocol
EIGRP AS	EIGRP Autonomous Systems
FC	Feasible Condition
FD	Feasible Distance
FS	Feasible Successor
GIT	GNU Interactive Tools
GNU	GNU's Not Unix
GPL	General Public License
HMAC	Hash Message Authentication Code
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IPX	Internetwork Packet Exchange
IS-IS	Intermediate System to Intermediate System
M2M	Machine to Machine

MD5	Message Digest 5
OLSR	Optimized Link State Routing
OSPF	Open Shortest Path First
OSR	Open Source Routing
PID	Process Identifier
RD	Reported Distance
RFC	Request for Comments
RIP	Routing Information Protocol
SHA	Secure Hash Algorithm
SIA	Stuck in Active
SPX	Sequence Packet Exchange
TCP	Transmission Control Protocol
TLV	Type Length Value
UDP	User Datagram Protocol
VLSM	Variable Length Subnet Masking
VPN	Virtual Private Network

Úvod

V dnešných počítačových sieťach sú smerovacie protokoly jednou z najdôležitejších technológií, ktorá zabezpečuje, aby všetky smerovače mali správne informácie a mohli tak doručovať pakety do svojich cieľov. Niektoré zo smerovacích protokolov sú otvorené a ich špecifikácie, či zdrojové kódy sú voľne dostupné, za ďalšími však stoja komerčné spoločnosti a spadajú pod ich vlastníctvo. Jedným z takýchto protokolov je aj EIGRP, vytvorený spoločnosťou Cisco Systems. Donedávna boli schopné týmto protokolom komunikovať výhradne smerovače Cisco a napriek svojim výborným vlastnostiam nemohol byť protokol nasadený v nehomogénnych sieťach so zariadeniami od rôznych výrobcov.

V roku 2013, keď spoločnosti Cisco vypršal kľúčový patent na EIGRP, rozhodli sa uvoľniť jeho špecifikáciu v podobe voľne dostupného EITF dokumentu, tzv. Internet Draft. Odborníkom v oblasti sietí táto udalosť otvorila úplne nové možnosti. Samotné Cisco od ich rozhodnutia očakáva rozšírenie podpory ich doteraz proprietárneho protokolu na ďalšie sieťové zariadenia, nezávisle od špecifického výrobcu.

Cieľom tejto diplomovej práce je vytvorenie plne otvorenej implementácie smerovacieho protokolu EIGRP na platforme Quagga pre akékoľvek zariadenie s operačným systémom Unix/Linux. Zadaná práca obnáša okrem vytvorenia funkčného softvéru aj jeho otestovanie na viacerých operačných systémoch a overenie kompatibility s niekoľkými smerovačmi Cisco. Výsledný zdrojový kód potom bude poskytnutý správcom balíka Quagga a začlenený do oficiálnej distribúcie tohto balíka. Následne očakávame pokračovanie vývoja v spolupráci s už existujúcou komunitou vývojárov.

1 Smerovací protokol EIGRP

História protokolu EIGRP siaha až do polovice 80-tych rokov 20. storočia, kedy spoločnosť Cisco Systems vyvinula predchodcu EIGRP - protokol IGRP. Išlo o alternatívu k vtedy široko nasadzovanému smerovaciemu protokolu RIP, verzie 1. Pri IGRP sa jednalo o classfull distance-vektor protokol, no oproti RIP prinášal isté výhody. Hlavnými z nich boli použiteľnosť a škálovateľnosť pre väčšie siete (max. 255 hopov, oproti 16 v RIP), efektívnejší prenos smerovacích informácií vďaka menšiemu update paketu, či schopnosť rozkladať toky dát medzi viacero aj nerovnocenných ciest. Už IGRP priniesol formu modulárnej architektúry, vďaka ktorej bol schopný prenášať informácie viacerých sieťových protokolov, konkrétne IPv4, ISO CLNP, Novell IPX alebo AppleTalk. Neskôr túto dôležitú vlastnosť prebral aj jeho nástupca, protokol EIGRP. Napriek množstvu výhod však IGRP stále obsahoval obmedzujúce vlastnosti brániace ďalšiemu vývoju. Patrili medzi ne opakované posielanie celej smerovacej databázy, nemožnosť používať VLSM podsieťovanie, pomalá rýchlosť konvergencie a chýbajúci mechanizmus garantujúci bezslučkovosť.

Práve takýto mechanizmus priniesol v roku 1989 profesor J.J Garcia-Luna-Aceves pracujúci v spoločnosti SRI International. Vyvinul a matematicky popísal stavový automat nazývaný Diffusing Update Algorithm, DUAL. DUAL spolu s využitím distribuovaného Bellman-Fordovho algoritmu výpočtu najkratších ciest v grafe, difúznych výpočtov podľa Dijkstru a Scholtena a Garciovho vlastného výskumu v oblasti kritérií pre výber acyklických trás predstavoval v tom čase unikátny a nový prístup k úlohe garantovane acyklického smerovania po najkratších cestách v sieti. Spoločnosť Cisco Systems použila DUAL ako rozhodovaciu logiku pre nový smerovací protokol EIGRP, ktorý vznikol rozšírením, respektívne úplným prepracovaním predchádzajúceho protokolu IGRP.

EIGRP patrí do kategórie distance-vector protokolov, ktoré si ako z názvu vyplýva, vymieňajú a udržiavajú polia (vektory) vzdialeností do jednotlivých známych sietí. Zo získaných informácií si EIGRP vytvára topologickú tabuľku, nad ktorou sa vykonávajú výpočty a z ktorej sa vyberá next-hop smerovač pre cieľovú sieť v smerovacej tabuľke. Viac informácii o topologickej tabuľke môžeme nájsť v časti 1.1.

Oproti IGRP, ktorý nevytváral medzi smerovačmi susedské vzťahy a namiesto toho periodicky oznamoval zoznam všetkých známych sietí, EIGRP proces udržovania susedstva

oddeľuje od prenosu informácií. Protokol zavádza mechanizmus posielania Hello paketov, ktoré sa podieľajú na inicializácii nového susedstva ale aj na udržiavaní už existujúceho. Vďaka tomu stačí, ak si smerovače vymenia všetky smerovacie informácie len pri vytvorení susedstva a ďalej budú oznamovať už iba zmeny, ktoré v sieti nastali. To má za následok výrazné zníženie zaťaženia jednotlivých liniek, aj šetrenie výpočtových zdrojov smerovačov.

Na zabezpečenie spoľahlivého prenosu smerovacích informácii bol v EIGRP použitý nový, pomocný transportný podprotokol - RTP. Viac informácii o jeho použití môžeme nájsť v sekcií 1.4.

EIGRP oproti svojmu predchodcovi dokáže garantovať bezslučkovosť vďaka zavedeniu "feasible condition". Týmto pojmom sa v angličtine označuje podmienka, pomocou ktorej vie lokálny smerovač identifikovať suseda na ceste do cieľa, ktorý zaručuje bezslučkové smerovanie. V nadväznosti na "feasible condition" navyše EIGRP používa difúzne výpočty. Pokiaľ smerovač stratí z akýkoľvek dôvodov svoj najlepší next-hop do cieľovej siete, aktívne sa pýta svojho okolia na novú cestu a po získaní odpovedí od všetkých susedov aktualizuje svoje informácie. Susedia, ktorých daná udalosť v sieti neovplyvní, pýtajúcemu sa smerovaču obratom pošlú odpoveď a otázku ďalej nešíria. Ak však bol pýtajúci sa smerovač next-hop smerovačom aj pre nich, musia posunúť otázky svojmu vlastnému okoliu a až po získaní všetkých odpovedí odpovedať za seba. Takýmto spôsobom sa šíria otázky len do tých častí siete, ktoré sú reálne výpadkom, či zmenou postihnuté. Všetky tieto mechanizmy a ich paralelné vykonávanie pre viacero položiek v topologickej tabuľke sú zabezpečené práve spomínaným stavovým automatom DUAL.

Ďalšou prevenciou proti vzniku smerovacích slučiek pri vzájomnej redistribúcii informácií medzi EIGRP a iným smerovacím protokolom je použitie rôznych administratívnych vzdialeností. Pre interné smery EIGRP používa hodnotu 90 a pre redistribuované, externé smery 170. Týmto je zabezpečené, aby nikdy nebola preferovaná redistribuovaná cesta pred rovnakou, ktorú už poznáme interne.

V prípade, že do určitej siete existuje viacero ciest a EIGRP vyhodnotí, že sú zaručene bezslučkové, dokáže pri výpadku jednej linky na novú trasu konvergovať takmer okamžite, prípadne súčasne využívať všetky dostupné cesty na rozdelenie dátového prenosu. Navyše dokáže využívať aj smery, ktorých metrika nie je rovnaká.

Všetky tieto vlastnosti ho robia vo svojej kategórii veľmi unikátnym a výkonným smerovacím protokolom, ktorý si môže v budúcnosti nájsť miesto v mnohých sieťových aplikáciách.

Keďže bol EIGRP od počiatku vyvinutý a implementovaný súkromnou firmou, jeho súčasti boli zabezpečené sériou patentov, ktoré zabraňovali iným výrobcom v jeho implementácii. Bolo ho preto možné využívať výlučne v homogénnych sieťach so smerovačmi od spoločnosti Cisco. Aj keď je známe, že sa niekoľko ďalších firiem snažilo získať určitý druh licencie na jeho používanie, reálne sa protokol neobjavil u žiadneho z ďalších výrobcov sieťových smerovačov.

1.1 Topologická tabuľka a DUAL

Získané informácie od susedných smerovačov si protokol EIGRP ukladá do tzv. topologickej tabuľky. Okrem prefixu so sieťovou maskou sa v nej nachádzajú aj informácie ako metrika ohlásená od daného susedom, výsledná metrika po zohľadnení ceny linky ku susedovi, jeho IP adresa, feasible distance, stav danej cieľovej siete v rámci DUAL mechanizmu a ďalšie prídavné informácie o sieti.

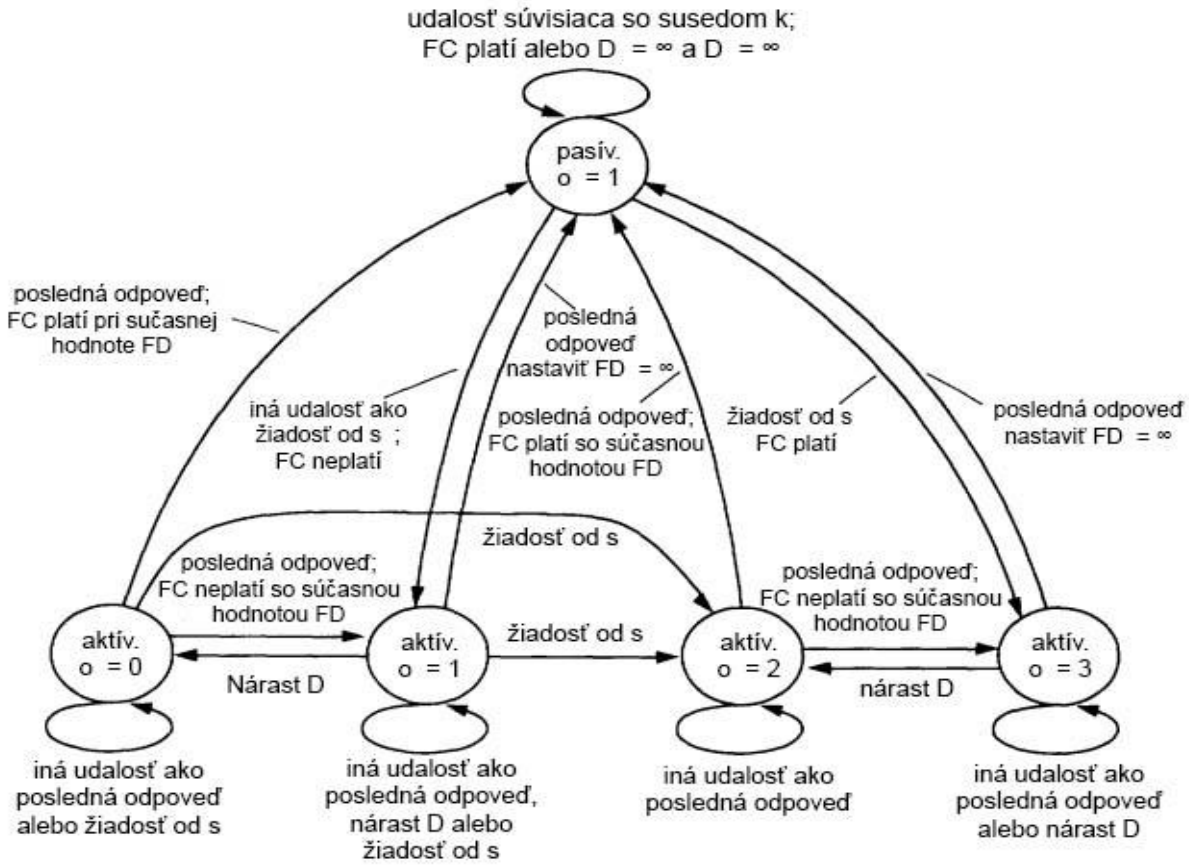
Jednotlivé prefixy sa do tejto tabuľky môžu dostať buď ako siete na priamo pripojených rozhraniach, ak sú zaradené do protokolu EIGRP, lokálne redistribuované siete, alebo po prijatí informácií z Update, Query, Reply, SIA-Query alebo SIA-Reply paketov.

EIGRP si k cieľovej sieti v topologickej tabuľke okrem štandardnej metriky zaznamenáva niekoľko ďalších číselných hodnôt:

- Reported distance – vzdialenosť, ktorú oznamuje sused
- Computed distance – výsledná vzdialenosť po zohľadnení ceny linky k susedovi
- Feasible distance - najkratšia zaznamenaná vzdialenosť od posledného prechodu topologického záznamu z aktívneho do pasívneho stavu

Ako J. J. Garcia špecifikoval vo svojom dokumente o bezslučkovom smerovaní [1], na to aby smerovač mohol považovať daného suseda za zaručene bezslučkového, musí preň platiť tzv. feasibility condition, alebo podmienka bezslučkovosti. Táto podmienka hovorí, že ak je hodnota reported distance pre daný prefix od konkrétneho suseda nižšia ako naša feasible distance, smerovanie dát cez tohto suseda zaručene nespôsobí smerovaciu slučku. Prefix je možné nainštalovať do smerovacej tabuľky jedine v prípade, že platí táto podmienka bezslučkovosti.

Algoritmus DUAL v procese EIGRP zabezpečuje reagovanie na zmeny v sieti a zaručuje vďaka kontrole podmienky bezslučkovosti výber takej trasy do cieľovej siete, ktorá nemôže spôsobiť smerovaciu slučku. Implementačne je DUAL navrhnutý ako konečný deterministický automat pre jednotlivé prefixy v topologickej tabuľke. Jeho fungovanie môžeme vidieť na nasledujúcom obrázku:



Obrázok 1.1 - Stavový automat DUAL, podľa [1]

Každý z prefixov v topologickej tabuľke sa môže nachádzať v jednom z 5 stavov. Záznam v pasívnom stave označuje, že smerovač identifikoval najlepší next-hop smerovač do danej siete. Záznam zostáva v pasívnom stave i po akejkoľvek topologickej zmene, o ktorej platí, že ten smerovač, ktorý po zohľadnení topologickej zmeny poskytuje najmenšiu výslednú vzdialenosť do cieľovej siete, spĺňa feasibility condition. V opačnom prípade musí sieť prejsť do niektorého z aktívnych stavov a smerovač začne rozosielať svojmu okoliu otázky, aby koordinovane vyhľadal náhradnú cestu do cieľa, ak existuje. Záznam následne prechádza späť do pasívneho stavu až po prijatí všetkých odpovedí a vybratí nového next-hop smerovača.

1.2 EIGRP metrika

Oproti ostatným smerovacím protokolom, EIGRP využíva zložitejší, kompozitný vzorec na výpočet výslednej metriky pre cieľovú sieť. Okrem štandardných parametrov linky

ako bandwidth (šírka pásma) do nej vstupujú aj ďalšie, konkrétne delay (oneskorenie), reliability (spoľahlivosť linky) a load (zaťaženie linky). V EIGRP paketoch sa popri týchto údajoch prenáša aj hodnota MTU, no nijak nevstupuje do výsledného výpočtu metriky.

$$\left[\left(K1 * \frac{10^7}{BW_{min}} + \frac{K2 * BW_{min}}{256 - LD_{max}} + K3 * \sum DL\right) * \frac{K5}{K4 + RL_{min}}\right] * 256$$

kde:

BW_{min} –šírka pásma najpomalšej linky po celej trase do cieľovej siete

LD_{max} – zaťaženie najviac zaťaženej linky po celej trase do cieľovej siete

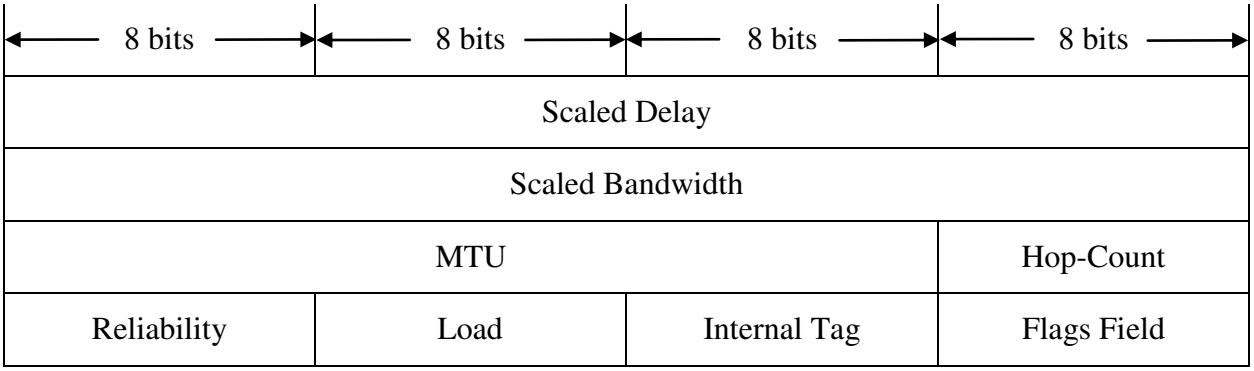
DL – oneskorenie linky

RL_{min} – spoľahlivosť najmenej spoľahlivej linky po celej trase do cieľovej siete

V štandardnej konfigurácii má smerovač nastavené na hodnotu 1 len koeficienty K1 a K3, ostatné sú nulové, takže do výsledného výpočtu vstupujú 2 parametre – najnižšia šírka pásma po celej trase a súčet oneskorení na všetkých linkách. Po úprave predchádzajúceho vzorca dostávame klasickú metriku, vypočítanú nasledovne:

$$\left(\frac{10^7}{BW_{min}} + \sum DL\right) * 256$$

V EIGRP paketoch sú pri použití klasickej metriky jednotlivé čiastkové údaje prenášané v nasledovnom formáte ako súčasť Internal a External TLV:



Obrázok 1.2 Vector Metric Section v Internal a External IPv4 TLV [2]

Okrem klasickej kompozitnej metriky popísanej vyššie, EIGRP pozná aj tzv. širokú metriku (wide metrics), ktorou tvorcovia reagujú na neustále sa zvyšujúce rýchlosti sieťových rozhraní. Šírka rozhrania v klasickej metrike totiž nedokáže odzrkadliť rozdiely pri použití rýchlejších rozhraní ako 10 Gb/s. Aby bolo možné lepšie rozlíšiť tieto dva typy metrík, rozšírená metrika mierne upravuje aj názvoslovie niektorých komponentov použitých na výpočet. Namiesto šírky pásma hovoríme o priepustnosti (throughput, T) a namiesto oneskorenia o latencii (latency, La) Samotný vzorec na výpočet rozšírenej metriky vyzerá nasledovne:

$$\left(K1 * T_{min} + K2 * \frac{T_{min}}{256 - LO_{max}} + K3 * LA_{Sum} + K6 * ExtM\right) * \left(\frac{K5}{K4 + R_{min}}\right)$$

pričom

$$T_{min} = \frac{65536 * 10^7}{BWmin}$$

$$LA_{sum} = \sum \frac{65536 * DL}{10^6}$$

Vo vzorci na výpočet latencie LA_{sum} je použité oneskorenie rozhrania DL, ktoré sa v prípade rozšírených metrík počíta 3 rôznymi spôsobmi

- **Štandardné oneskorenie [pikosekunda]** – ak ide o rozhranie s rýchlosťou pod 1Gbps alebo v prípade, že je nakonfigurovaný príkaz bandwidth

- $\frac{\mathbf{10^{13}}}{\mathbf{Def.\ BW}}$ – pre rozhrania nad 1 Gbps
- $\mathbf{10^7 * delay}$ - ak je na rozhraní nakonfigurovaná hodnota oneskorenia príkazom delay.

1.3 Typy EIGRP paketov a príslušných TLV

Ako každý protokol, aj EIGRP pre svoje fungovanie potrebuje sériu správ s dohodnutým formátom, ktoré si medzi sebou smerovače vymieňajú. EIGRP je navyše špecifické tým, že podporuje variabilné množstvo prídavných polí v každom z paketov, nazývaných TLV.

V nasledujúcej časti si popíšeme jednotlivé formáty paketov, im prislúchajúce TLV a ich úlohu v procese EIGRP.

1.3.1 Hlavička EIGRP paketu

8 bits	8 bits	8 bits	8 bits
Header Version	Opcode	Checksum	
Flags			
Sequence Number			
Acknowledgement Number			
Virtual Router ID		Autonomous System Number	

Obrázok 1.3 Hlavička EIGRP paketu [2]

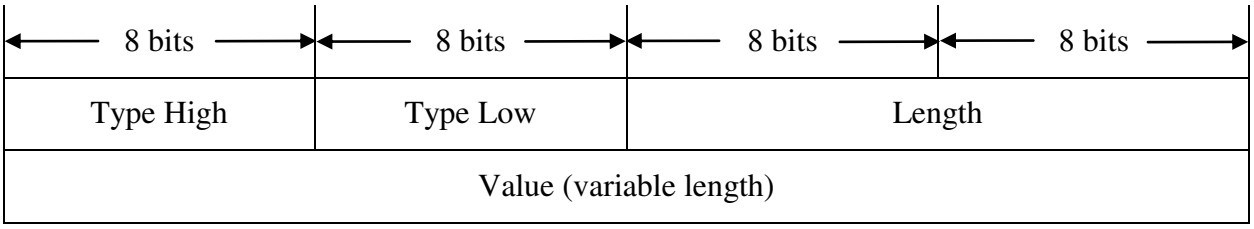
Všetky prenášané pakety v EIGRP musia obsahovať hlavičku, znázornenú na obrázku 1.3. Ide o spoločné polia o celkovej veľkosti 20 bajtov, ktoré obsahujú všetky potrebné informácie určujúce typ celého paketu, polia pre spoľahlivé doručovanie paketov (mechanizmus RTP), číslo autonómneho systému, či ďalšie prídavné značky.

Vysvetlenie jednotlivých polí:

- Header Version – 4-bitové pole, popísujúce verziu formátu EIGRP hlavičky. Od počiatku EIGRP v tomto poli uvádza hodnotu 2
- Opcode – 4-bitové pole, označuje konkrétny typ paketu s nasledujúcimi možnými hodnotami: (1) Update, (2) Request, (3) Query, (4) Reply, (5) Hello alebo Ack, (7) Probe, (10) SIA-Query, (11) SIA-Reply
- Checksum – 24-bitové pole, ktorého hodnota je vypočítaná z obsahu celého EIGRP paketu. Pokiaľ je kontrolná suma nesprávna, paket je smerovačom zahodený.
- Flags – 32-bitové pole, definujúce špeciálne zaobchádzanie s paketmi. Jednotlivé bity tohto poľa môžu byť nastavené nasledovne:
    - Init (0x01) – bit sa nastavuje pri inicializácii susedstva s ďalším smerovačom v UPDATE pakete. Vyžadujeme ním poslanie celej topologickej tabuľky od suseda.
    - CR (0x02) – bit indikuje, že prijímateľ môže akceptovať tento paket len ak sa nachádza v režime podmieneného prijímania. Do tohto režimu sa môže smerovač dostať po predchádzajúcom prijatí HELLO paketu so Sequence TLV
    - RS (0x04) – Restart bit, ktorý je nastavený v HELLO a počiatočnom UPDATE pakete hovorí smerovaču, že sa jeho sused reštartuje, ale chce si udržať vytvorené susedstvo. Na to, aby reštart úspešne prebehol je potrebná spolupráca oboch strán, čo znamená, že RS bit musí následne nastaviť aj lokálny smerovač.
    - EOT (0x08) – End-of-Table bit indikuje pri inicializácii nového susedstva, že smerovač poslal všetky topologické informácie, ktoré obsahuje jeho topologická tabuľka. V prípade reštartujúcich sa smerovačov je tento príznak použitý na zistenie, či už máme všetky potrebné smerovacie informácie z UPDATE paketov od okolitých smerovačov.
- Sequence number – smerovač nastavuje toto 32-bitové pole pri posielaní paketu na unikátnu hodnotu. Pomocou neho je zabezpečený spoľahlivý prenos informácií a paket daného poradového čísla musí byť vždy potvrdený druhou stranou. V prípade, že je pole nastavené na 0, daný paket nevyžaduje potvrdenie.
- Acknowledgement number – 32-bitové pole využívané na potvrdenie prijatia paketu so sekvenčným číslom v rovnakej hodnote, aká je nastavená v tomto poli.

- Virtual RouterID (VRID) – 16-bitová hodnota, identifikujúca virtuálny smerovač, s ktorým je tento daný paket spojený. Ak je v tomto poli nepodporovaná alebo neznáma hodnota, paket je automaticky zahodený.
- Autonomous system number – Identifikuje číslo autonómneho systému v ktorom pracuje EIGRP proces. Nepriamo sa dá považovať za určitý druh veľmi jednoduchej autentifikácie, nakoľko sa susedstvo medzi dvoma smerovačmi vytvorí len za predpokladu, že je táto hodnota v hlavičkách ich paketov rovnaká.

1.3.2 Všeobecný formát TLV



Obrázok 1.4 Všeobecný formát TLV [2]

EIGRP pakety môžu obsahovať variabilný počet týchto prídavných polí. Každé z nich je identifikované svojim typom, má špecifikovanú dĺžku a nesie samotné dáta daného TLV. TLV záznamy prinášajú do EIGRP modulárnosť jednotlivých paketov, prenášaných informácií, efektívne využívanie prenosu a zlepšenú kompatibilitu s viacerými verziami protokolu. Pokiaľ totiž smerovač nepozná dané TLV, môže jeho spracovanie jednoducho preskočiť. Táto situácia môže nastať pri používaní spätnej kompatibility so staršími zariadeniami.

Horný oktet v type TLV určuje, informácie ktorého protokolu sú prenášané v jeho vnútri. Pole môže nadobúdať nasledujúce hodnoty:

- 0x00 Všeobecné
- 0x01 IPv4
- 0x04 IPv6
- 0x05 SAF
- 0x06 Multiprotokol

Ak horný oktet typu TLV obsahuje hodnotu všeobecného TLV 0x00, dolný oktet potom bližšie špecifikuje typ samotného TLV s nasledujúcimi hodnotami:

- 0x01 Parameter TLV
- 0x02 Authentication TLV
- 0x03 Sequence TLV
- 0x04 Software Version TLV
- 0x05 Multicast Sequence TLV
- 0x06 Peer Information TLV
- 0x07 Peer Termination TLV
- 0x08 Peer Topology ID TLV

Pre IPv4 siete existujú špeciálne vyhradené Typy TLV:

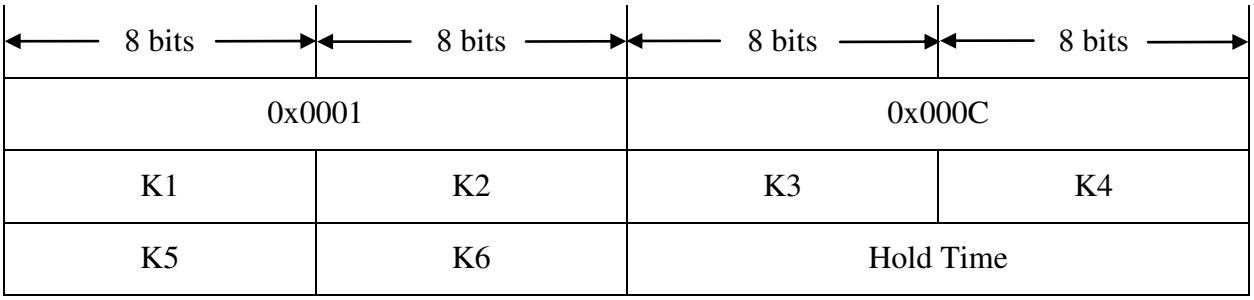
- 0x0102 Internal IPv4 TLV
- 0x0103 External IPv4 TLV
- 0x0104 Community TLV

Tieto čísla predstavujú hodnoty celého poľa TLV, teda Type High spolu s Type Low. Treba poznamenať, že v súčasnosti sú už tieto typy zastaralé a má sa prechádzať sa multiprotokolové TLV pre prenos smerovacích informácii.

1.3.3 Hello paket

Hello mechanizmus je jedna z významných zmien oproti, IGRP. Používa sa ako mechanizmus na dynamické objavovanie susedných smerovačov, overenie ich vzájomnej kompatibility a následné udržiavanie aktívneho susedstva.

Hlavička Hello paketu je špecifická tým, že okrem správne nastaveného poľa Opcode (5), jej sekvenčné aj potvrdzovacie pole musí mať nulovú hodnotu. Okrem hlavičky musí paket obsahovať minimálne Parameter TLV v nasledujúcom formáte:



Obrázok 1.5 Parameter TLV [2]

Parameter TLV obsahuje konfiguračne nastaviteľné hodnoty jednotlivých K parametrov, spolu s hodnotou Hold Time. Toto číslo hovorí prijímajúcemu smerovaču, po koľkých sekundách od prijatia posledného platného EIGRP bude považovať daného suseda za nedostupného. Hodnoty prenášané v Hello paketoch sú používané na overenie, či je možné vytvoriť susedstvo medzi dvoma smerovačmi.

Okrem Parameter TLV sa v hello paketoch môžu nachádzať aj ďalšie, konkrétne Sequence TLV, Software Version TLV a Peer Topology ID TLV.

Hello pakety nevyžadujú potvrdenie a posielajú sa na multicast adresu 224.0.0.10 pre IPv4 a FF02::A pre IPv6 v dvoch rôznych intervaloch:

- **5 sekúnd** - pre siete s podporou broadcast, point-to-point sériové linky, ATM, či vysokorýchlostné multipoint linky
- **60 sekúnd** - pre multipoint rozhrania so šírkou pásma T1 alebo menej, siete NMBA, napríklad Frame Relay multipoint rozhrania

1.3.4 Ack paket

Ack pakety sa používajú na potvrdzovanie prijatia paketov vyžadujúcich spoľahlivé doručenie. Svojim formátom sa v princípe zhodujú s bežným Hello paketom. Rovnako majú nastavené aj pole hlavičky Opcode na 5. Rozdiel je v nenulovom poli Acknowledgement, ktorého hodnota sa nastavuje na sekvenčné číslo paketu vyžadujúceho potvrdenie.

Pokiaľ smerovač nedostane v stanovenom intervale potvrdenie na svoj spoľahlivo odoslaný paket, EIGRP odošle paket znova. Hodnota tohto intervalu je dynamicky a periodicky počítaná z údajov pre konkrétne rozhranie smerovača.

Ack sa posiela vždy ako unicast na adresu príjemcu, ktorý potvrdenie vyžaduje.

1.3.5 Update paket

EIGRP Update paket je určený na samotný prenos smerovacích informácií medzi smerovačmi. Vyplýva z toho, že tieto pakety je nutné doručovať spoľahlivo a vyžadujú potvrdenie . Vo svojej hlavičke majú nastavené špecifické sekvenčné číslo a Opcode (1). Tieto pakety môžu byť posielané ako unicast, tak aj multicast.

Unicast posielaný update:

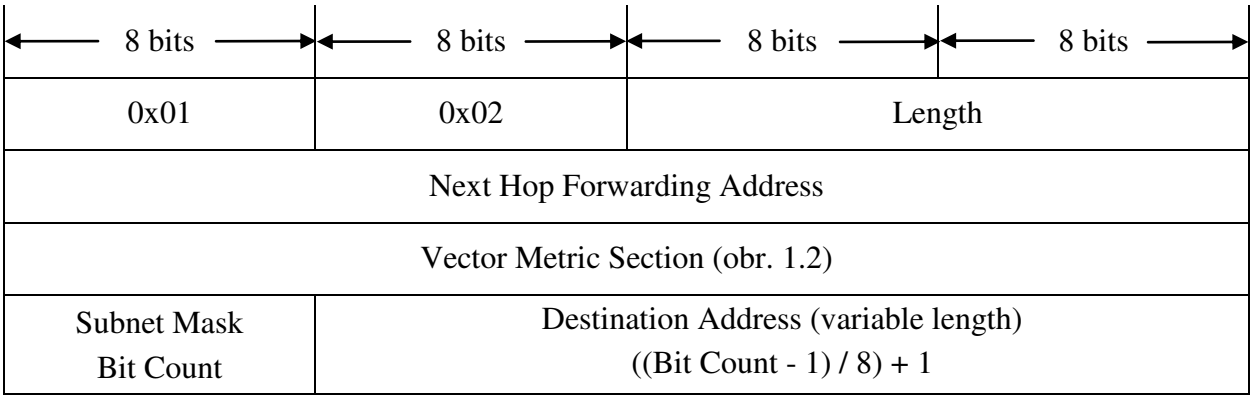
- Na point-to-point prepojoch a pri staticky nakonfigurovaných susedoch
- Ak smerovač nedostal potvrdenie na svoj predchádzajúci multicastový update paket, odošle ho opäť už ako unicast priamo svojmu konkrétnemu susedovi.
- Pri vytváraní nového susedstva a synchronizácii smerovacích informácií

Multicast posielaný update:

- Po úspešnom vytvorení susedstva a pri oznamovaní zmien v sieti
- Pri špecifických situáciách nasadenia EIGRP nad VPN a Hub-and-Spoke topológii

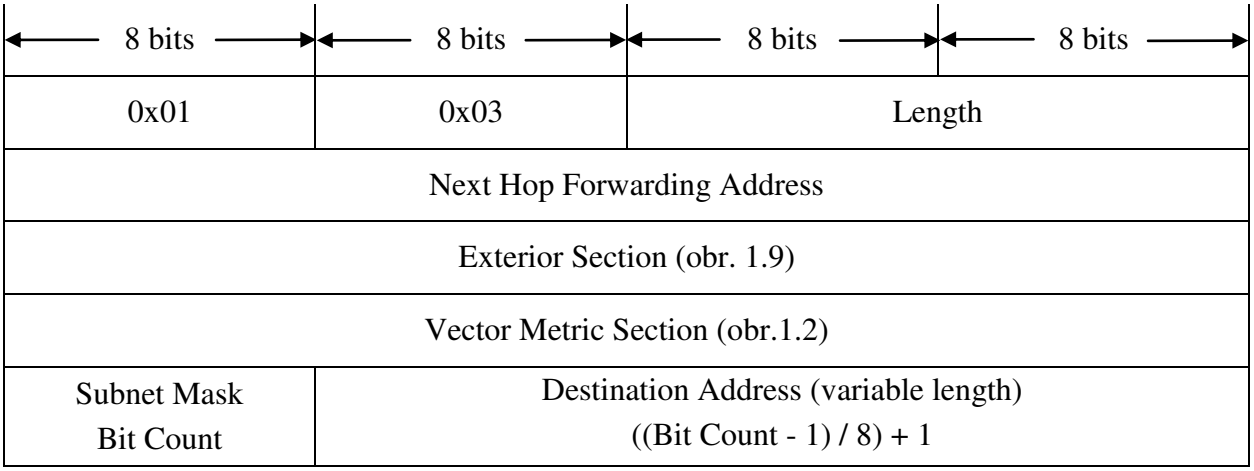
Smerovacie informácie sú v Update paketoch prenášané ako variabilný počet Internal TLV, ak ide o siete v rámci jedného autonómneho systému a External TLV, pokiaľ sa jedná o redistribuované siete z iného protokolu. Navyše jeden EIGRP Update paket môže obsahovať oba typy TLV.

Internal TLV



Obrázok 1.6 Internal IPv4 TLV [2]

External TLV



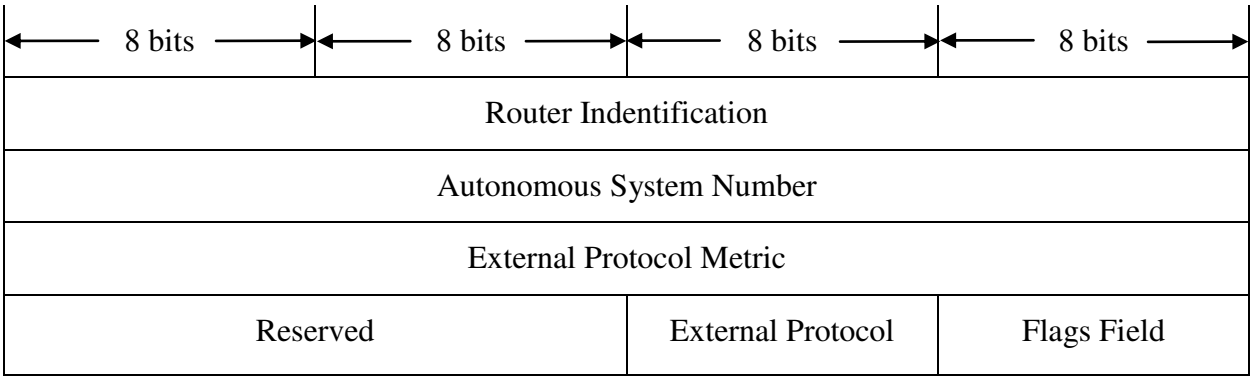
Obrázok 1.7 External IPv4 TLV [2]

Zaujímavosťou pri Internal aj External TLV je ich variabilná dĺžka v poli pre prenos IP prefixu. Na základe hodnoty masky siete, prenášanej v poli Subnet Mask Bit Count je dĺžka poľa Destination Address nasledovná:

Subnet Mask Bit Count (BC)	Dĺžka Destination Address
BC <= 8	1 bajt
8 < BC <= 16	2 bajty
16 < BC <= 24	3 bajty
24 < BC	4 bajty

Obrázok 1.8 Variabilná dĺžka poľa Destination Address v Internal a External IPv4 TLV

External TLV navyše v sebe nesie aj detailnejšie informácie o pôvode prenášanej externej siete. ktoré môžeme vidieť na obrázku 1.9.



Obrázok 1.9 Exterior Section v External IPv4 TLV [2]

1.3.6 Query paket

Pokiaľ v sieti nastane z akýchkoľvek dôvodov situácia, že smerovač, ktorý aktuálne ponúka najkratšiu cestu do istej siete, nespĺňa feasibility condition, DUAL spustí proces aktívneho pýtania sa okolitých smerovačov na túto sieť. Na tieto účely je určený Query paket, ktorý v sebe nesie Internal alebo External TLV s popisom danej siete. Podobne ako Update je aj Query doručovaný spoľahlivo a za použitia multicast aj unicast IP adries.

1.3.7 Reply paket

Reply pakety sú odosielané ako odpoveď na prijatú Query, vždy ako unicast. V hlavičke majú nastavené Opcode pole na 4. Paket musí byť doručovaný spoľahlivo a obsahuje informácie o aktuálnej vzdialenosti do dotazovanej siete, po zohľadnení informácií z prijatého paketu Query. Vo svojom tele prenáša opäť Internal a External TLV ako u predchádzajúcich typov.

1.3.8 SIA Query/SIA Reply

Pri spustení Query procesu sa niekedy, obzvlášť pri veľkej a komplexnej sieti môže stať, že získanie všetkých odpovedí od okolitých smerovačov na poslanú Query trvá dlhší čas. EIGRP smerovače majú pre tento prípad mechanizmus, ktorý kontroluje, či daný sused skutočne stále očakáva odpovede na svoje vlastné otázky a preto neodpovedá, alebo nastala nejaká neštandardná situácia. Po uplynutí časovača na dĺžku difúzneho výpočtu je susedom, od ktorých sme stále neprijali Reply odoslaná správa SIA Query. Ak je susedný smerovač stále aktívny a očakáva ukončenie svojho vlastného Query procesu, okamžite spätne odošle SIA-Reply.

1.4 RTP

Prenos paketov smerovacieho protokolu, najmä paketov prenášajúcich riadiacu a smerovaciu informáciu, vo všeobecnosti vyžaduje spoľahlivý transport. V IP sieťach však neexistuje univerzálny spoľahlivý transportný protokol s podporou multicastingu, a naviac, protokol EIGRP vznikal so snahou byť nezávislý od konkrétneho sieťového a transportného protokolu, keďže jeho použitie bolo zamýšľané aj pre siete s IPX/SPX či AppleTalk. EIGRP si preto implementuje svoj vlastný spoľahlivý transportný protokol Reliable Transport Protocol (RTP). Nejde však o samostatné PDU, do ktorého by boli EIGRP pakety zabalené. RTP polia sú vložené priamo v hlavičke EIGRP paketu, čím je docielené značné zníženie dátovej réžie potrebnej na prenos. Ide o už spomínané polia Flags, Sequence a Acknowledgement (obrázok 1.3). Logika používania týchto polí, najmä Sequence a Acknowledgement, funguje analogicky

ako v TCP. Každý EIGRP paket vyžadujúci spoľahlivé doručenie vo svojej hlavičke nesie unikátne sekvenčné číslo. Na potvrdenie daného paketu smerovač vyžaduje prijať od suseda paket s poľom Acknowledgement v hlavičke nastaveným na hodnotu odoslaného sekvenčného čísla. Je dôležité poznamenať, že tento paket nemusí byť len Ack paket, ale hocijaký platný unicastový EIGRP paket. Ak teda náš susedný smerovač potrebuje súčasne potvrdiť prijatie nášho paketu a súčasne odoslať nám nejaký unicastový paket, môže oboje realizovať naraz pomocou jediného paketu. V tomto zmysle je RTP príbuzné protokolu TCP, ktoré takisto nerozlišuje medzi samostatne stojacimi potvrdzovacími segmentmi a segmentmi pre prenos užitočnej informácie. Každý TCP segment môže súčasne potvrdzovať prijaté dáta od druhej strany a súčasne prenášať vlastné dáta druhej strane. Na rozdiel od TCP sa v RTP nečíslujú bajty, ale odoslané spoľahlivé pakety, a RTP nemá koncept posuvného okna premenlivej veľkosti. Po odoslaní jedného spoľahlivého paketu musí RTP čakať, kým dostane potvrdenia od všetkých jeho príjemcov, až potom môže pokračovať v odosielaní ďalšieho multicastového paketu. Výnimku z tohto pravidla tvorí implementácia tzv. podmieneného príjmu (Conditional Receive), ktorá však nemení spôsob potvrdzovania a posúvania sa na ďalšie spoľahlivé pakety čakajúce na odoslanie, len dovoľuje RTP pokračovať v ich odosielaní takým spôsobom, aby si ich zaostávajúce smerovače nevšímali.

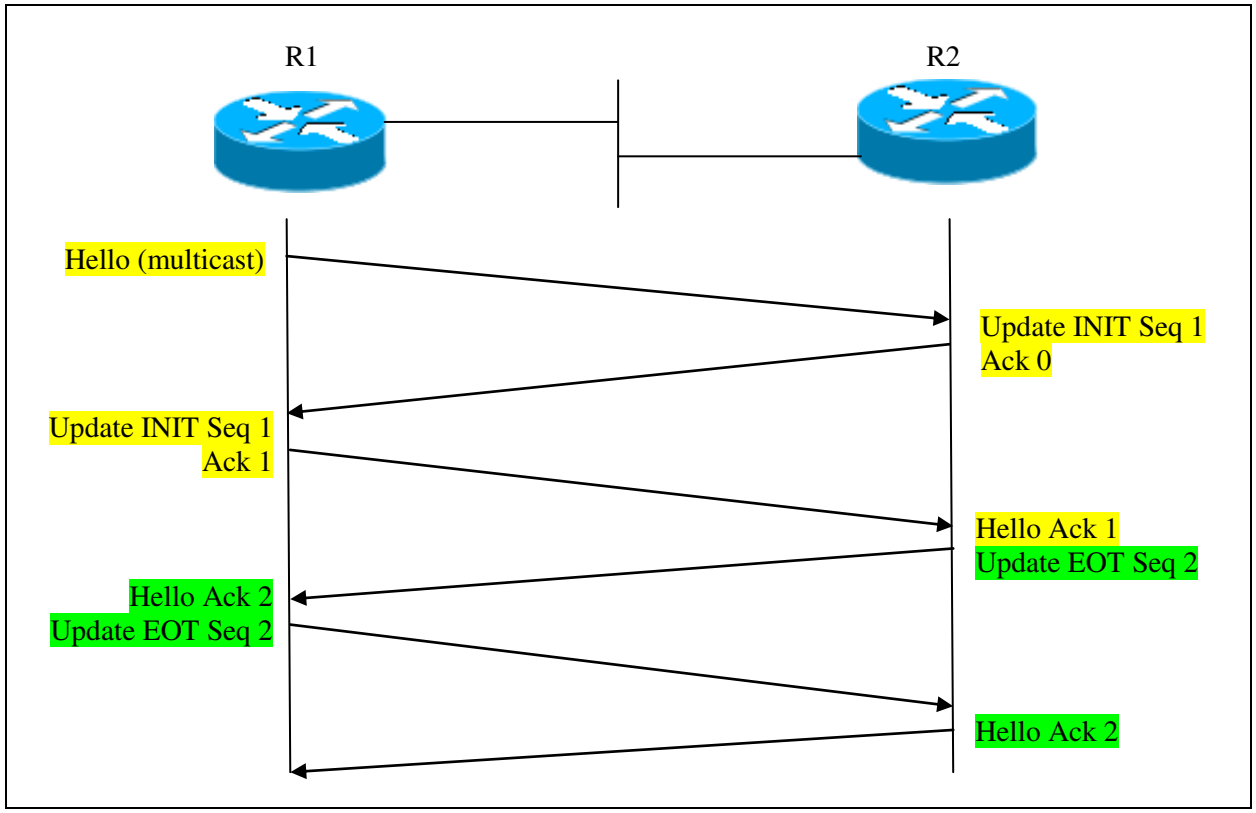
Každý smerovač dynamicky na základe vlastností linky pre jednotlivé pakety spúšťa časovač, nazývaný retransmit časovač. Pokiaľ nedostane potvrdenie na svoj odoslaný paket pred uplynutím tohto času, paket je poslaný znova. Štandardne po 16-tich neúspešných opakovaných pokusoch o doručenie, sa susedstvo s nekomunikujúcim smerovačom zruší. EIGRP po odoslaní paketu na jednu z multicast adries 224.0.0.10, či FF02::A spúťa aj tzv. Multicast Flow Timer. Ak smerovač dostane všetky potvrdenia svojho multicastového paketu skôr, než Multicast Flow Timer vyprší, pokračuje štandardne odoslaním ďalšieho paketu. Ak však od niektorého zo susedov potvrdenie nedostane, takýto paket smerovač začne posielať opätovne na základe spomínaného retramsmit časovača, priamo na IP adresu neodpovedajúceho suseda formou unicastového paketu.

1.5 Nadväzovanie a udržiavanie susedstiev

EIGRP umožňuje vytvárať susedstvá buď automaticky, použitím Hello paketov posielaných cez konkrétne rozhranie na EIGRP multicast adresy 224.0.0.10 v prípade IPv4 a FF02::A pre IPv6, alebo je možné susedov definovať manuálne v konfigurácii smerovača. V takomto prípade je pre dané rozhranie vypnutá podpora multicastovej EIGRP prevádzky a nie je možné objaviť ďalších susedov za daným rozhraním dynamicky.

Aby sa dva EIGRP smerovače mohli stať susedmi, musia byť splnené všetky nasledujúce podmienky:

- rozhrania oboch smerovačov pripojené na spoločnej IP podsieti
- zhodné číslo autonómneho systému v hlavičkách paketov
- zhodné EIGRP K hodnoty
- zhodujúce sa parametre autentifikácie (ak je použitá)



Obrázok 1.10 Nadväzovanie susedstva medzi EIGRP smerovačmi

Na obrázku 1.10 môžeme vidieť podrobnejšie znázornený proces vytvárania susedstva medzi dvoma smerovačmi

Proces začína prijatím multicast Hello paketu na smerovači R2. Pre pokračovanie je potrebné skontrolovať zhodné K hodnoty a číslo autonómneho systému v Parameter TLV. Rovnako tak musí byť správna autentifikačná suma v príslušnom TLV, pokiaľ je autentifikácia aktivovaná. V momente prijatia Hello paketu je nový sused v stave Down.

Nasleduje výmena tzv. "Null Update", alebo inak povedané prázdneho Update paketu s nastaveným INIT príznakom. Susedný smerovač sa presúva do stavu Pending. Ide o počiatočnú signalizáciu a výzvu na zaslanie topologických informácií. Ako každý Update, aj tento je doručovaný spoľahlivo, a pre úspešné vytvorenie susedstva je potrebné, aby druhá strana potvrdila jeho sekvenčné číslo v akomkoľvek svojom pakete. Zvyčajne môžeme vidieť toto potvrdenie obratom pomocou susedovho Null Update. Sused sa presúva do stavu Up a susedstvo sa dá považovať v tejto fáze za aktívne.

Posledným krokom je zaslanie všetkých informácií z topologickej tabuľky. Toto prebieha štandardne pomocou Internal a External TLV v Update paketoch. Posledný z Update paketov nesie príznak EOT signalizujúci, že obsahuje posledné smerovacie informácie a proces výmeny je ukončený.

1.6 Redistribúcia

EIGRP umožňuje do svojej topologickej tabuľky vložiť nie len siete prenášané v rámci jedného autonómneho systému, ale aj redistribuovať informácie z iných smerovacích protokolov.

Konfigurácia redistribúcie vyžaduje špecifikovanie zdrojového protokolu a počiatočnej metriky po jednotlivých komponentoch. Redistribuované siete sa do smerovacej tabuľky vkladajú s administratívnou vzdialenosťou 170, čo zamedzuje vzniku smerovacích slučiek, pri vzájomnej redistribúcii EIGRP s ďalším smerovacím protokolom. Medzi jednotlivými smerovačmi sú tieto siete prenášané opäť pomocou Update paketov, vo formáte External TLV, ktorý môžeme vidieť na obrázku 1.6.

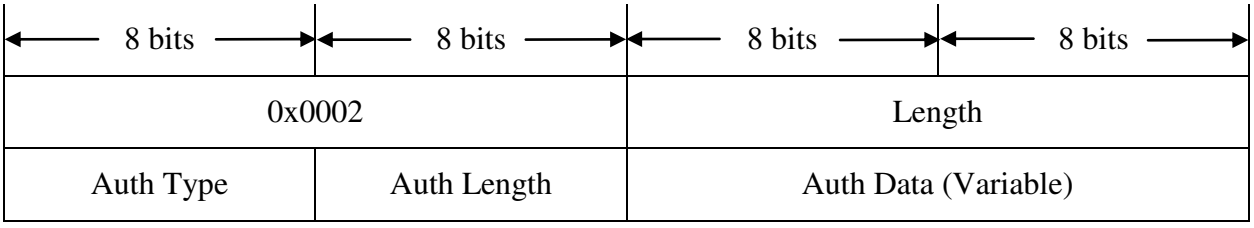
1.7 Sumarizácia

Jednou z vlastností EIGRP ako každého classless distance-vector smerovacieho protokolu je podpora sumarizácie prenášaných prefixov v ktoromkoľvek mieste v sieti. Výhodou sumarizácie je zmenšenie počtu smerovacích záznamov v topologickej tabuľke a tým zníženie nárokov na pamäťové, aj výpočtové zdroje smerovača a zníženie zaťaženia linky pri prenose smerovacích informácií. Sumarizácia v EIGRP navyše znižuje hĺbku šírenia Query paketov v sieti, čo má za následok rýchlejšie ukončenie Query procesu a nedochádza k Stuck-in-Active stavu, popísanému v časti 1.3.8.

1.8 Autentifikácia prenášaných informácií

Dôležitým aspektom fungovania smerovacích protokolov je zabezpečenie prenosu smerovacích informácii pred útokmi typu tzv. „man-in-the-middle“. Takýmto spôsobom môže dôjsť k podvrhnutiu sfalšovaných, nepravdivých informácii smerovaču, čo bude mať za následok nesprávne smerovanie v sieti.

Tvorcovia EIGRP vyvinuli za účelom ochrany informácií autentifikačné TLV, zobrazené na obr. 1.11, ktoré sa pridáva a prenáša v každom EIGRP pakete po nakonfigurovaní procesu autentifikácie.



Obrázok 1.11 Autentifikačné TLV [2]

EIGRP v súčasnosti podporuje 2 typy zabezpečovacích algoritmov:

MD5

MD5 algoritmus je jedna z najrozšírenejších kryptografických hashovacích funkcií, produkujúca 128-bitovú hodnotu. Zvyčajne býva reprezentovaná v textovom formáte ako 32 znakové hexadecimálne číslo.

V protokole EIGRP sa MD5 suma využíva na kontrolu integrity prenášaných dát v pakete a keďže sa do výslednej hodnoty započítava aj spoločné, vopred dohodnuté heslo, dá sa táto suma použiť zároveň ako autentifikačný mechanizmus. Heslo je potrebné nakonfigurovať na všetkých smerovačoch prepojených spoločnou sieťou.

Quagga implementácia MD5 zabezpečenia využíva na výpočet tejto hodnoty dátovú štruktúru md5_ctxt. S touto štruktúra pracujú nasledujúce funkcie:

- `void md5_init (md5_ctxt *)`
- `void md5_loop (md5_ctxt *, const void *, u_int)`
- `void md5_pad (md5_ctxt *)`
- `void md5_result (uint8_t *, md5_ctxt *)`

Inicializačná funkcia md5_init nastavuje počiatočné hodnoty jednotlivým poliam v štruktúre md5_ctxt. Funkcia md5_loop následne prepočítava MD5 hash za použitia dát, ktoré sa do funkcie predávajú ako druhý argument. Tretí parameter tejto funkcie obmedzuje, koľko bajtov dát z druhého argumentu má funkcia zobrať do úvahy pri výpočtoch. MD5_pad sa používa na zarovnanie spracovávaných dát pred finalizáciou pomocou funkcie md5_result. Táto zo štruktúry md5_ctxt vytvára výsledné 16-bajtové pole znakov, ktoré sa prenáša v paketoch.

Po dlhotrvajúcom prieskume spôsobu použitia MD5 mechanizmu v oficiálnej Cisco EIGRP implementácii boli zistené rozdielne postupy pri počítaní jednotlivých výsledných súm, a to nasledovne:

- Pri zabezpečení **Hello** paketu do výpočtu vstupuje len jeho prvých 44 bajtov. Navyše, ak je použité heslo kratšie ako 16 znakov, po jeho započítaní funkciou md5_loop, MD5 dáta prechádzajú opätovným prepočítaním za použitia umelo vytvoreného hesla, pozostávajúceho výlučne z núl. Počet núl predstavuje doplnok dĺžky hesla do hodnoty 16.

Výsledný zdrojový kód potom vyzerá nasledovne:

```
unsigned char digest[16];
md5_ctxt *ctx;
md5_init(&ctx);
md5_loop(&ctx, packet, 44);
md5_loop(&ctx, key->string, strlen(key->string));
if(strlen(key->string) < 16)
  md5_loop(&ctx, '0', 16 - strlen(key->string));
md5_pad(&ctx);
md5_result(digest, &ctx);
```

Obrázok 1.12 Počítanie MD5 sumy pre Hello paket

- Pri zabezpečení počiatočného **Null Update** paketu do výpočtu nijak nevstupuje nakonfigurované heslo a MD5 suma sa počíta len z prvých 40-tich bajtov.

```
unsigned char digest[16];
md5_ctxt *ctx;
md5_init(&ctx);
md5_loop(&ctx, packet, 40);
md5_pad(&ctx);
md5_result(digest, &ctx);
```

Obrázok 1.13 Počítanie MD5 sumy pre Null Update paket

- MD5 suma v bežnom **Update** pakete, ktorý nesie smerovacie informácie, používa rovnaký princíp pre započítavanie hesla ako Hello paket. Rovnako tak je aj na začiatku započítaných len prvých 44 bajtov z paketu. Výpočet však pokračuje ďalej, kedy sú vstupnými dátami všetky Internal a External TLV v pakete, avšak len v dĺžke ich celkovej veľkosti mínus 20 bajtov

```
unsigned char digest[16];
int offset = HEADER_SIZE + AuthTLV_SIZE; /* 60 bajtov */
md5_ctxt *ctx;
md5_init(&ctx);
md5_loop(&ctx, packet, 44);
md5_loop(&ctx, key->string, strlen(key->string));
if(strlen(key->string) < 16)
  md5_loop(&ctx, '0', 16 - strlen(key->string));

if(packet_size > offset))
{
  md5_loop(&ctx, paket + offset, backup_end - 20 - offset);
}
md5_pad(&ctx);
```

```
md5_result(digest, &ctx);
```

Obrázok 1.14 Počítanie MD5 sumy pre Update, Query, SIA-Query a SIA-Reply

- U ostatných typoch paketov, ktoré nesú smerovacie informácie (**Query, Reply, SIA-Query, SIA-Reply**), výpočet MD5 sumy prebieha rovnako, ako v prípade štandardných Update paketov.

Pri počítaní MD5 sumy musí platiť, že do výpočtov vstupuje celý paket, v ktorom hneď za EIGRP hlavičkou nasleduje autentifikačné TLV. V čase výpočtu je potrebné, aby mal paket nulovú hodnotu polí Checksum v hlavičke a MD5 Digest v autentifikačnom TLV.

Tieto špecifiká doposiaľ neboli nikde publikované a táto diplomová práca je prvým dokumentom, ktorý na ne poukazuje.

HMAC-SHA 256

HMAC je špeciálnym typom zabezpečovacieho kódu (Message Authentication Code, MAC), kedy výpočet výslednej zabezpečovacej hodnoty obsahuje aj kryptografickú hash funkciu v kombinácii s vopred dohodnutým heslom. Rovnako, ako v predchádzajúcom prípade sa v EIGRP využíva zároveň na kontrolu integrity dát, ako aj na samotnú autentifikáciu. Oproti MD5 algoritmu je v Cisco implementácii jednoduchší postup pri výpočte, kedy sa ako vstupné dáta používa vždy celý EIGRP paket, vrátane Authentication TLV.

2 Balík Quagga

Mnoho dnešných smerovačov je ponúkaných zákazníkom v hardvérovom prevedení, kedy samotné zariadenia, aj operačný systém vyrába jeden konkrétny výrobca. Existujú však aj softvérové varianty, ktorých zdrojové kódy môžu byť dokonca otvorené, voľne dostupné pod hlavičkou open-source.

Príkladom takéhoto otvoreného riešenia je Quagga. Ide o balík smerovacieho softvéru, určený pre akékoľvek zariadenia postavené na operačných systémoch Unix/Linux. Quagga vznikla už v roku 1996, vtedy ešte pod názvom Zebra Project. Jej tvorca Kunihiro Ishiguro pracoval na projekte s cieľom vytvoriť prepojenie sietí poskytovateľov internetového pripojenia British Telecom a Marubeni. Jeho vízia o kvalitnom smerovacom softvéri po finančnej podpore od investora Yoshinari Yoshikawa prerástla do reálneho projektu, ktorý sa stal vo svojom čase prvým softvérovým multifunkčným smerovacím balíkom.

V súčasnosti Quagga podporuje protokoly RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, IS-IS, BGP-4 a BGP-4+, BABEL, OLSR a neustále pribúdajú ďalšie. [8]

Quagga poskytujú aktuálne plnú podporu na nasledujúcich platformách

- GNU/Linux (akákoľvek distribúcia, i386)
- FreeBSD
- NetBSD
- OpenBSD
- Solaris

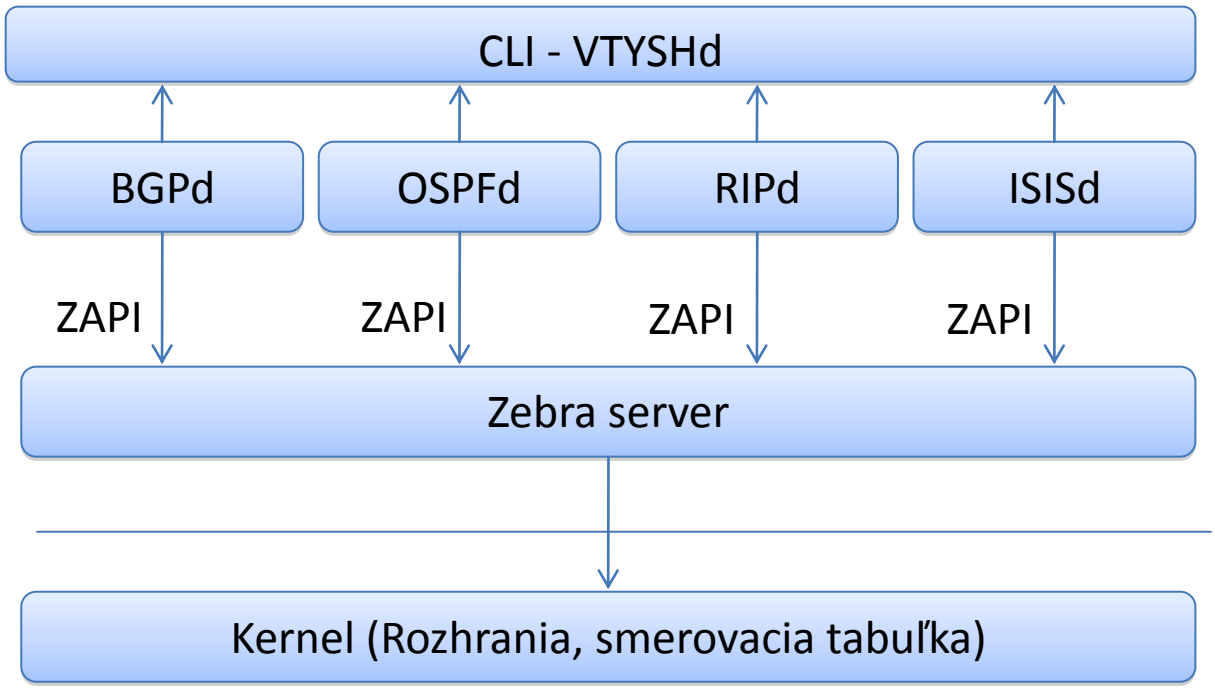
Na vývoj balíka Quagga je od začiatku používaný programovací jazyk C.

2.1 Softvérová architektúra balíka Quagga

Niektoré z existujúcich smerovacích nástrojov a softvérov sú implementované ako jednoprocesné programy, ktoré v sebe spájajú a poskytujú všetku funkcionalitu. Tvorcovia Quaggy na tieto účely zvolili odlišnú architektúru. Balík je zložený z niekoľkých démonov,

pričom každý z nich predstavuje samostatný smerovací protokol a je spúšťaný ako samostatný proces.

Tieto démony sú pomocou interného protokolu Quagga ZAPI prepojené so zebra serverom, ktorý tvorí abstrakčnú vrstvu medzi volaniami interných funkcií kernelu operačného systému a jednotlivými protokolovými procesmi. Žiaden z protokolových démonov vďaka tomu nemusí implementovať celú funkcionalitu pre prácu so smerovacou tabuľkou, či fyzickými sieťovými rozhraniami. Pre ilustráciu môžeme vidieť činnosť balíka Quagga na obrázku 2.1



Obrázok 2.1 Architektúra balíka Quagga

ZAPI jednotlivým démonom umožňuje pri svojej inicializácii získať štruktúry s informáciami o pripojených sieťových rozhraniach zariadenia, dynamicky reagovať na zmeny v stave rozhraní (IP konfigurácia, funkčnosť linky), či získavať a vkladať smerovacie informácie do smerovacej tabuľky operačného systému.

V čase, keď Quagga vznikala ešte neexistovala známa a v súčasnosti vo veľkej miere využívaná knižnica **pthread** pre tvorbu viacvláknových aplikácií na platforme Unix/Linux.

Reálne preto všetky moduly fungujú ako jednovláknové procesy. Tvorcovia balíka však prišli aspoň s ich vlastnou implementáciou pseudo-vlákien, ktoré vývojárom umožňujú do určitej miery ovplyvniť prioritu vykonávania funkcií vo vnútri balíka. Toto nachádza využitie napríklad pri garantovaní odozvy konfiguračného rozhrania jednotlivých protokolov, bez ohľadu na to, čo vykonávajú vo svojom vnútri. Pseudo-vlákno pre užívateľský vstup a spracovávanie zadaných príkazov má najvyššiu prioritu.

2.2 Distribučná open source licencia

Aj keď je otvorený softvér voľne prístupný pre všetkých užívateľov, jeho šírenie sa riadi podľa podmienok stanovených rôznymi neziskovými organizáciami, ktoré sa zaoberajú podporou a ochranou projektov typu open-source. Zdrojový kód balíka Quagga je ošetrený GNU GPLv2 licenciou, vytvorenou organizáciou Free Software Foundation v roku 2007.

GNU GPLv2 je najrozšírenejšia licencia pre voľne šíriteľný softvér, ktorá garantuje koncovým používateľom voľnosť v používaní, študovaní, zdieľaní, úpravách, či dokonca v predaji dotknutého softvéru. Každý používateľ alebo distribútor však musí pri šírení softvéru zabezpečiť, aby aj noví majitelia mali rovnako dostupné všetky práva, tak ako sú definované v licencii. Pre Quaggu to znamená, že balík musí byť voľne šírený a pokiaľ na ňom aj výrobcovia postavia svoje sieťové zariadenia, mali by používateľom umožniť plný prístup k zdrojovému kódu, prípadne možnosť tento kód upraviť a prispôsobiť ho pre svoje potreby [4].

2.3 Nasadenie Quagga smerovačov

Mnohí výrobcovia sieťových zariadení a smerovačov stavili na otvorené riešenie Quagga ako na nástroj pre vykonávanie a podporu základných smerovacích protokolov. Quagga prináša značné výhody z finančného hľadiska, keďže ide o otvorený, zdarma šíriteľný softvér a vďaka výbornej podpore od komunity vývojárov, starajúcich sa o balík.

Quagga BGP môžeme nájsť v mnohých uzloch internetu, kedy sú ako smerovače v určitom autonómnom systéme použité celé servery. Veľkosť BGP smerovacej tabuľky sa môže pri súčasnom počte používaných IP sietí dostať až na hodnoty niekoľkých stoviek MB. To si vyžaduje množstvo systémových zdrojov smerovača, ktoré sú u väčších výrobcov veľmi

drahé. Zoznam viacerých voľne dostupných BGP smerovačov postavených na balíku Quagga môžeme nájsť napríklad na adrese https://www.bgp4.net/. Ide o „smerovač pod lupou“ (z angl. looking glass), v ktorom si môže ktokoľvek po pripojení sa pomocou protokolu telnet pozrieť jeho základnú konfiguráciu a obsah smerovacej tabuľky.

Príkladmi ďalších výrobcov, stavajúcich svoje zariadenia na balíku Quagga sú:

- Mellanox Technologies
- ZyXEL
- Soecris

2.4 Spravovanie a vývoj balíka Quagga

Zdrojové kódy Quagga sú udržiavané v oficiálnom GIT repozitári, ktorý je možné nájsť na adrese git://git.savannah.nongnu.org/quagga.git. O správu a pridávanie nového kódu do repozitára sa starajú niekoľkí správcovia, vždy po dôkladnej kontrole správnej funkčnosti a jeho formálnej stránky.

Za účelom zvýšenia povedomia o existencii otvoreného riešenia Quagga bola v roku 2011 vďaka iniciatívne spoločností Internet Systems Consortium a Google, založená skupina Open Source Routing (OSR). OSR bola neskôr, v roku 2014 presunutá pod neziskovú organizáciu Network Device Education Foundation (NetDEF) sídliacu v Silicon Valley. V súčasnosti sa OSR zameriava na podporu vývojárskej komunity Quagga, vydávanie oficiálnych verzií kódu, či podporu nezávislých prispievateľov a akademických inštitúcií, ktorým takto ponúka priestor a pomoc pri vytváraní vysokokvalitného a stabilného kódu pre balíka Quagga [11].

2.5 Inštalácia a konfigurácia

Inštaláciu balíka Quagga je možné vykonať buď z balíčkovacích systémov na jednotlivých operačných systémoch, alebo ručne kompiláciou zo zdrojového kódu. Keďže na účely tejto práce je potrebné pristupovať a upravovať priamo zdrojové kódy, v nasledujúcej časti budú popísané všetky potrebné kroky pre úspešné skompilovanie a spustenie balíka touto cestou.

Pred začatím kompilácie Quagga je potrebné na cieľový systém nainštalovať nasledujúce balíčky:

- GCC, verzia 2.95
- Automake, verzia 1.9.6
- Autoconf, verzia 2.59
- Libtool, verzia 1.5.22
- Texinfo, verzia 4.7
- GNU AWK, verzia, 3.1.5
- Dia, verzia 0.92.2-1

Po úspešnej inštalácii nasleduje spustenie pripraveného skriptu bootstrap.sh, ktorý sa nachádza v koreňovom adresári Quagga. Nakoľko Quagga používa v prvotnej fáze kompilácie infraštruktúru AutoTools, ktorá zabezpečuje automatizované vytváranie konfiguračných a kompilačných súborov pre každý z jej protokolových modulov, tento skript vytvorí v každom z adresárov súbory configure a make.

```
# ./bootstrap.sh
```

Následne je možné pristúpiť k samotnej kompilácii pomocou príkazov:

```
# make
# make install
```

Po úspešne ukončenej inštalácii budú v adresári /usr/local/etc/ vytvorené konfiguračné súbory jednotlivých protokolových modulov. Ide však len o vzorové súbory a pre správne spustenie Quagga procesov z ich názvu treba odstrániť časť „.sample“.

2.6 Import zdrojového kódu Quagga do Eclipse IDE

Aby sme zefektívnili vývoj protokolu EIGRP v balíku Quagga, používame na tieto účely integrované vývoje prostredie Eclipse. V nasledujúcej časti je popísané, ako importovať zdrojový kód z GIT repozitára do vývojového prostredia a ako ho správne zkonvertovať na projekt v jazyku C. Návod je písaný pre anglickú jazykovú lokalizáciu prostredia Eclipse.

Postup pri importovaní zdrojového kódu:

1. Po prvom spustení Eclipse IDE je potrebné doinštalovať doplnky poskytujúce podporu pre vývoj v jazykoch C/C++ a podporu pre repozitáre GIT. V menu Eclipse vyberieme Help →Install New Software.
2. Z rozbalovacej ponuky zvolíme „Update Site“ pre konkrétnu verziu Eclipse.
3. Z kategórie Programming Languages vyberieme „C/C++ Development Tools“ a z kategórie Collaboration „Eclipse EGit“, prípadne „Eclipse JGit“.
4. Následne v ľavom okne Project Explorer po kliknutí pravým tlačítkom na myške, vyberieme Import → Git → Projects from Git. Zvolíme URI repozitár a vyplníme potrebné údaje na stiahnutie git repozitára.
5. Po stiahnutí repozitára vyberieme možnosť „Import as General Project“
6. Po pravom kliknutí na práve importovaný projekt vyberieme New → Other → Convert to a C/C++ Autotools Project. V nasledujúcom okne stačí vybrať importovaný projekt, označiť „Covert to C Project“ a potvrdiť voľbu tlačítkom Finish.

Predchádzajúca procedúra vytvorí všetky potrebné prepojenia častí zdrojového kódu vrámci vývojového prostredia, čo nám značne uľahčí pohyb a vyhľadávanie v zdrojovom kóde.

3 Analýza problému

Hlavnou a najdôležitejšou časťou tejto diplomovej práce je samotná implementácia smerovacieho protokolu EIGRP. Keďže proprietárna verzia od spoločnosti Cisco už existuje, od začiatku nám išlo o vytvorenie open-source verzie, ideálne v rámci niektorého z už existujúcich balíkov smerovacieho softvéru. Patria medzi ne napríklad [9]:

- Quagga Routing Suite
- XORP
- Vyatta (VyOS)
- pfSense

Pri výbere najvhodnejšej platformy pre vývoj protokolu EIGRP padlo rozhodnutie na Quaggu hlavne kvôli podpore širokej škály operačných systémov, pre jej dobrú dokumentáciu, veľmi kvalitný, prehľadný a udržiavaný zdrojový kód a veľmi aktívnu komunitu, s ktorou môže vývojár konzultovať svoj pokrok, či riešiť prípadné problémy.

Vývoj omnoho komplexnejších balíkov XORP a Vyatta už naopak niekoľko rokov stagnuje, bolo by preto náročné hľadať akúkoľvek formu podpory a informácií. Týmto balíkom na komplexnosti pridáva aj funkcionalita, ktorá nesúvisí priamo so smerovaním ako firewalling, IPS, VPN služby a mnohé ďalšie. Nevýhodou balíka Vyatta je jeho spojenie do jedného celku s operačným systémom Debian a nemožnosť nezávislej distribúcie na iné operačné systémy. V prípade pfSense je opäť veľkým obmedzením jeho nasadenie len na platformách s operačným systémom FreeBSD.

Úlohou tejto diplomovej práce bolo vytvoriť a otestovať nový EIGRP modul s kompletnou základnou funkcionalitou, transportnou vrstvou a plným prepojením na Zebra server v balíku Quagga. V tomto protokole bolo potrebné následne implementovať transportnú vrstvu, umožňujúcu vytvárať a spoľahlivo doručovať pakety, vytvárať a spravovať EIGRP susedstvá, redistribuovať externé prefixy z iných smerovacích protokolov, či sumarizovať takto získané smerovacie informácie. Táto úloha obnáša niekoľko oblastí, ktoré bolo potrebné vziať do úvahy ešte pred začatím samotného softvérového vývoja:

- Celý zdrojový kód sa musí prísne riadiť princípmi popísanými v *GNU Coding Style* [10], ktoré softvérový balík Quagga vo svojich moduloch dodržiava. Ide hlavne

o spôsob formátovania kódu a používania špecifík jazyka C, čo zabezpečuje prehľadnosť celého softvérového projektu.

- Nový protokolový démon treba začleniť do infraštruktúry Quagga a urobiť ho tak plnohodnotnou súčasťou balíka pre každého používateľa. Znamená to použitie rovnakého prístupu pri vytváraní zdrojových súborov, ich kompilácii, spúšťaní, či konfigurovaní EIGRP modulu.
- EIGRP proces musí vedieť v rámci balíka Quagga správne komunikovať so Zebra serverom, ktorý je okrem iného zodpovedný za správu smerovacej tabuľky hostiteľského operačného systému.
- EIGRP proces musí byť navrhnutý modulárne, čo mu bude umožňovať súbežne pracovať ako s IPv4 formátom paketov, tak aj IPv6. Aj keď podpora pre IPv6 siete nie je priamo podstatou tejto práce, v budúcnosti sa počíta s jej implementáciou.
- Pri implementácii je potrebné dodržiavať všetky informácie poskytnuté v otvorenom štandarde Internet Draft, aby bola dosiahnutá plná kompatibilita so smerovačmi od spoločnosti Cisco.
- Pokiaľ možno, je potrebné využívať čo najviac hotovej softvérovej infraštruktúry, ktorá už existuje v knižniciach balíka Quagga, aby sme sa vyhli duplicitnej implementácii použitých funkcií.
- Výsledný zdrojový kód je potrebné priebežne ladiť a testovať jeho funkčnosť nie len v rámci jedného operačného systému používaného na vývoj, ale aj na iných operačných systémoch podporovaných balíkom Quagga, ideálne aj na iných procesorových architektúrach ako napríklad ARM.
- Medzi nepovinnú, ale rovnako veľmi dôležitú súčasť projektu patrí postupné vytvorenie dokumentácie protokolu a používateľskej príručky pre používateľov balíka Quagga.

4 Implementácia EIGRP do balíka Quagga

Nakoľko je implementácia nového protokolového modulu nesmierne komplexný projekt a vyžaduje vytvorenie obrovského množstva zdrojového kódu, nie je v rozsahových možnostiach tejto diplomovej práce oboznámiť čitateľov s detailnými popismi celého zdrojového kódu. V tejto kapitole bude bližšie vysvetlené použitie hlavných štruktúr, ako aj niektorých funkcií, zabezpečujúcich základné vykonávanie procesu EIGRP.

Vývoj protokolu EIGRP v balíku Quagga prebieha spolu s kolegami Matejom Perinom a Petrom Orságom ako študentský tímový projekt. Bolo preto potrebné hneď na jeho začiatku nasadiť systém na správu revízií zdrojového kódu. Dokážeme tak zefektívniť tímovú kooperáciu a zálohovať výsledný zdrojový kód. Na tieto účely sme sa rozhodli použiť nástroj GIT a služby verejne dostupného on-line GIT repozitára Github.

Aktuálna verzia zdrojového kódu sa nachádza na adrese:

https://github.com/janovic/Quagga-EIGRP/tree/EIGRP-Development

4.1 Súbory v adresári eigrpd

Balík Quagga, ako už bolo spomínané, sa skladá z oddelených softvérových modulov, ktoré sú v rámci stromovej štruktúry repozitára členené v adresároch prislúchajúcich jednotlivým smerovacím protokolom. Analogicky bol pre EIGRP vytvorený samostatný adresár *eigrpd*. V ňom umiestnené súbory so zdrojovým kódom sú rozdelené podľa ich funkcionality, ktorú si v nasledujúcej časti popíšeme. Hlavičkové súbory s príponou .h zvyčajne obsahujú prototypy jednotlivých funkcií používaných v prislúchajúcom, rovnomennom súbore so zdrojovým kódom s príponou .c

- ***eigrp_const.h*** – definície konštánt používaných v rámci celého softvérového modulu.
- ***eigrp_dump.c / eigrp_dump.h*** – funkcie umožňujúce vykonávať debugovanie, prípadne získavať a zobrazovať na konzolu rôzne informácie o bežiacom procese.

- ***eigrp_filter.c / eigrp_filter.h*** – zdrojový kód určený na filtrovanie prenášaných smerovacích informácií použitím distribučných zoznamov, či smerovacích máp.
- ***eigrp_fsm.c / eigrp_fsm.h*** – implementácia DUALu konečného automatu, ktorý zabezpečuje rozhodovacie funkcie pri detekcii topologických zmien.
- ***eigrp_hello.c*** – zdrojový kód, určený na prijímanie, spracovávanie a posielanie Hello paketov. Prototypy funckii sa nachádzajú v súbore eigrp_packet.h.
- ***eigrp_interface.c / eigrp_interface.h*** – funkcie umožňujúce spravovať jednotlivé rozhrania používané protokolom na komunikáciu s okolím a zebra serverom.
- ***eigrp_macros.c*** – obsahuje definície makier jazyka C, ktoré sa využívajú ako pomocné funkcie v rôznych častiach kódu
- ***eigrp_main.c*** – súbor obsahujúci hlavnú zavádzaciu funkciu main, s množstvom inicializačných funkcií.
- ***eigrp_neighbor.c / eigrp_neighbor.h*** – obslužný kód pre vytváranie a udržiavanie susedstiev, ako aj údajových štruktúr s nimi súvisiacich
- ***eigrp_network.c / eigrp_network.h*** – kód určený na inicializáciu sieťových soketov, softvérovú konfiguráciu sieťových rozhraní a kalkuláciu EIGRP metrík
- ***eigrp_packet.c / eigrp_packet.h*** – zdrojový kód zodpovedný za nízkoúrovňovú prácu s akýmkoľvek typom paketov, tzn. prijímanie, triedenie, kontrolovanie, vytváranie a posielanie.
- ***eigrp_query.c / eigrp_reply.c*** – funkcie umožňujúce spracovávať a posielať EIGRP query a reply pakety. Prototypy funkcíi sa nachádzajú v súbore eigrp_packet.h.
- ***eigrp_siaquery.c / eigrp_siareply.c*** - funkcie umožňujúce spracovávať a posielať EIGRP SIA-Query a SIA-Reply pakety. Prototypy funkcíi sa nachádzajú v súbore *eigrp_packet.h.*
- ***eigrp_snmp.c / eigrp_snmp.h*** – kód zabezpečujúci podporu SNMP funkcionality spojenej s EIGRP
- ***eigrp_structs.h*** – hlavičkový súbor obsahujúci definície všetkých použitých údajových štruktúr v programe
- ***eigrp_topology.c / eigrp_topology.h*** – implementácia topologickej tabuľky a funkcií s ňou spojených

- ***eigrp_update.c*** – kód určený na spracovávanie a posielanie EIGRP update paketov. Prototypy funkcií sa nachádzajú v súbore eigrp_packet.h.
- ***eigrp_vty.c*** – funkcie umožňujúce spracovávať zadané konfiguračné a diagnostické príkazy z terminálu
- ***eigrp_zebra.c / eigrp_zebra.h*** – implementácia prepojenia EIGRP modulu so zebra serverom za účelom výmeny smerovacích, či stavových informácií
- ***eigrpd.c / eigrpd.h*** – obsluha alokácie a dealokácie samotnej inštancie EIGRP protokolu v rámci nového modulu.

4.2 Dátové štruktúry EIGRP

Jednou z kľúčových fáz vývoja protokolu EIGRP bol návrh vhodných dátových štruktúr, ktoré nám umožnia uloženie a prácu s potrebnými informáciami o jednotlivých častiach protokolu. Všetky štruktúry sú definované v súbore eigrp_structs.h a najdôležitejšie z nich si popíšeme v nasledujúcich sekciách. Vysvetlivky k jednotlivým premenným v štruktúre sa nachádzajú nad každou z nich v podobe komentára.

4.2.1 struct eigrp_master

```
struct eigrp_master
{
  /* Zoznam inštancií štruktúr eigrp */
  struct list *eigrp;
  /* Hlavné vlákno procesu */
  struct thread_master *master;
  /* Zoznam dostupných rozhraní systému */
  struct list *iflist;
  /* Záznam o čase spustenia */
  time_t start_time;
  /* Nakonfigurované prepínače procesu  */
  u_char options;
};
```

Ukážka 4.1 – Štruktúra egirp_master

Štruktúra eigrp_master je hlavnou dátovou entitou EIGRP procesu. Navrhnutá bola podľa vzoru ďalších protokolových démonov balíka Quagga. Keďže na jednom systéme je možné spustiť niekoľko inštancií protokolu EIGRP, každý s odlišným číslom autonómneho systému, práve v tejto štruktúre sa ukladá ich zoznam. Obsahuje smerník na hlavné vlákno procesu, ktoré zabezpečuje manažment a plánovanie ďalších spustených vlákien. Ukladá sa do nej aj zoznam dostupných rozhraní systému a informácie o čase spustenia, prípadne nakonfigurované prepínače procesu.

4.2.2 struct eigrp

```
struct eigrp
{
  /* Číslo autonómneho systému */
  u_int16_t AS;
  /* Virtual Router ID */
  u_int16_t vrid;
  /* K-hodnoty */
  u_char k_values[6];
  /* Multiplikátor metrík */
  u_char variance;
  /* Maximum možných ciest do 1 siete */
  u_char max_paths;
  /* Názov EIGRP inštancie */
  char *name;
  /* ID smerovača, automatické */
  u_int32_t router_id;
  /* ID smerovača, manuálne */
  u_int32_t router_id_static;
  /* Zoznam štruktúr eigrp_interface */
  struct list *eiflist;
  /*   Pasívne rozhrania */
  u_char passive_interface_default;
  /* Popisovač súboru */
  unsigned int fd;
  /* Veľkosť vyrovnávacej pamäte */
  unsigned int maxsndbuflen;
  /* Globálna hodnota EIGRP sekvenčného čísla */
  u_int32_t sequence_number;
  /* Pamäť pre prichádzajúce dáta */
  struct stream *ibuf;
  /* Front odchádzajúcich dát */
  struct list *oi_write_q;
  /* Vlákno pre posielanie dát */
  struct thread *t_write;
  /* Vlákno pre prijímanie dát */
  struct thread *t_read;
```

```
  /* Nakonfigurované EIGRP siete */
  struct route_table *networks;
  /* Topologická tabuľka */
  struct list *topology_table;
  /* Sériové číslo zmien v topologickej tabuľke */
  u_int64_t serno;
  /* Naposledy zaznamenané sériové číslo */
  u_int64_t serno_last_update;
  /* Zoznam zmien v topológii potrebných na odoslanie pre interné
prefixy */
  struct list *topology_changes_internalIPV4;
/* Zoznam zmien v topológii potrebných na odoslanie pre externé
prefixy */
  struct list *topology_changes_externalIPV4;
  /* Virtuálny EIGRP sused */
  struct eigrp_neighbor *neighbor_self;
  /* Metrika pre redistribuované prefixy z určitého protokolu*/
  struct eigrp_metrics dmetric[ZEBRA_ROUTE_MAX + 1];
  /* Počet redistribuovaných protokolov */
  int redistribute;
};
```

Ukážka 4.2 – Štruktúra eigrp

Rozsiahla štruktúra eigrp predstavuje samotnú inštanciu protokolu EIGRP a ukladá v sebe všetky hlavné informácie potrebné na vykonávanie jeho činnosti. Nachádzajú sa tu globálne hodnoty špecifické pre konkrétnu inštanciu ako napríklad ID smerovača, číslo autonómneho systému, K-hodnoty, sekvenčné číslo, zoznam rozhraní, topologická tabuľka, či nakonfigurované siete. Štruktúra sa nevytvára hneď po spustení procesu EIGRP, ale až po aktivovaní protokolu pre konkrétny autonómny systém príkazom

```
eigrpd(config)# router eigrp [číslo AS]
```

4.2.3 struct eigrp_interface

```
struct eigrp_interface
{
  /* Smerník na inštanciu EIGRP, pod ktorú rozhranie patrí */
  struct eigrp *eigrp;
  /* Smerník na prislúchajúcu generickú štruktúru rozhania */
  struct interface *ifp;
  /* Výstupná vyrovnávacia pamäť pre pakety */
  struct eigrp_fifo *obuf; /* Output queue */
  /* Konfiguračné parametre rozhrania */
  struct eigrp_if_params *params;
```

```
  /* Príznak aktivovaného multicast prenosu */
  u_char multicast_memberships;
  /* Typ pripojenej siete */
  u_char type;
  /* IP prefix rozhrania */
  struct prefix *address;
  /* Smerník na štruktúru Zebra servera */
  struct connected *connected;
  /* Zoznam susedov za príslušným rozhraním*/
  struct list *nbrs;
  /* Vlákno určené na posielanie hello paketov */
  struct thread *t_hello;
  /* Počet paketov na odoslanie */
  int on_write_q;
  /* Smerník na aplikované access-list štruktúry */
  struct access_list *list[EIGRP_FILTER_MAX];
  /* Smerník na aplikované prefix-list štruktúry */
  struct prefix_list *prefix[EIGRP_FILTER_MAX];
  /* Smerník na aplikované route-map štruktúry */
  struct route_map *routemap[EIGRP_FILTER_MAX];

  /* Štatistické premenné predstavujúce počty jednotlivých
odoslaných a prijatých typov paketov */
  u_int32_t hello_in;
  u_int32_t update_in;
  u_int32_t query_in;
  u_int32_t reply_in;
  u_int32_t hello_out;
  u_int32_t update_out;
  u_int32_t query_out;
  u_int32_t reply_out;
  u_int32_t siaQuery_in;
  u_int32_t siaQuery_out;
  u_int32_t siaReply_in;
  u_int32_t siaReply_out;
  u_int32_t ack_out;
  u_int32_t ack_in;
  /* Sekvenčné číslo pre vykonávanie autentifikácie */
  u_int32_t crypt_seqnum;
};
```

Ukážka 4.3 – Štruktúra eigrp_interface

Štruktúra eigrp_interface sa vytvára pri spustení démona pre každé z rozhraní, o existencii ktorého nás informuje zebra server. Štruktúra eigrp_interface v sebe ukladá jednak smerník na generickú štruktúru rozhrania používanú v Quagge struct interface, ale aj množstvo ďalších informácií, ktoré sú špecifické len pre protokol EIGRP.

4.2.4 struct eigrp_if_params

```
struct eigrp_if_params
{
  /* Definície konfigurovateľných parametrov rozhrania */
  DECLARE_IF_PARAM (u_char, passive_interface);
  DECLARE_IF_PARAM (u_int32_t, v_hello);
  DECLARE_IF_PARAM (u_int16_t, v_wait);
  DECLARE_IF_PARAM (u_char, type);
  DECLARE_IF_PARAM (u_int32_t, bandwidth);
  DECLARE_IF_PARAM (u_int32_t, delay);
  DECLARE_IF_PARAM (u_char, reliability);
  DECLARE_IF_PARAM (u_char, load);
  DECLARE_IF_PARAM (char *, auth_keychain );
  DECLARE_IF_PARAM (int, auth_type);
};
```

Ukážka 4.4 – Štruktúra eigrp_if_params

Pre zvýšenie prehľadnosti EIGRP špecifických parametrov, ktoré je možné nakonfigurovať na rozhraniach smerovača boli tieto oddelené v samostatnej štruktúre eigrp_if_params. Alokuje sa pri vytváraní štruktúr rozhrania eigrp_interface a jej adresa sa priraďuje do smerníka *params. Na prácu s parametrami rozhraní boli podľa už existujúcich implementácií protokolov v Quagge vytvorené niekoľké makrá

- *DECLARE_IF_PARAM* – deklarovanie parametra
- *IF_DEF_PARAMS* – umožňuje prístup k jednotlivým parametrom v štruktúre
- *SET_IF_PARAM* – vytvorenie príznaku, pomocou ktorého je možné kontrolovať formálne aktívnosť daného parametra
- *UNSET_IF_PARAM* – formálna deaktivácia parametra

4.2.5 struct eigrp_neighbor

```
struct eigrp_neighbor
{
  /* Referencia na EIGRP rozhranie, za ktorým sa nachádza sused */
  struct eigrp_interface *ei;
  /* Stav suseda (DOWN, PENDING, UP) */
  u_char state;
  /* Naposledy prijaté sekvenčné číslo */
  u_int32_t recv_sequence_number;
```

```
  /* Sekvenčné číslo v Null Update, použité pri inicializácii
susedstva*/
  u_int32_t init_sequence_number;
  /* Adresa susedného smerovača */
  struct in_addr src;
  /* Verzia operačného systému smerovača – hodnoty použité len vo
výpisoch */
  u_char os_rel_major;
  u_char os_rel_minor;
  /* Verzia podporovaných TLV formátov. Informácia zabezpečujúca
kompatibilitu*/
  u_char tlv_rel_major;
  u_char tlv_rel_minor;
  /* Jednotlivé K-hodnoty prijaté od suseda */
  u_char K1;
  u_char K2;
  u_char K3;
  u_char K4;
  u_char K5;
  u_char K6;
  /* Hold timer hodnota prijatá od suseda a použitá v náležitom
EIGRP časovači */
  u_int16_t v_holddown;
  /* Vlákno EIGRP hold time časovača */
  struct thread *t_holddown;
  /* Front pre pakety, ktoré je potrebné opätovne poslať */
  struct eigrp_fifo *retrans_queue;
  /* Front pre posielanie multicastových paketov*/
  struct eigrp_fifo *multicast_queue;
  /* Prijaté autentifikačné sekvenčné číslo */
  u_int32_t crypt_seqnum;
};
```

Ukážka 4.5 – Štruktúra eigrp_neighbor

Štruktúra neighbor predstavuje dátové úložisko pre všetky informácie týkajúce sa jednotlivých susedných smerovačov v rámci EIGRP procesu. Vytvára sa pri dynamickom objavení nového suseda, po prijatí hello paketu, spracovanie ktorého môžeme vidieť v časti 4.6. Každá zo štruktúr v sebe ukladá smerníky na fronty určené pre retransmisie nepotvrdených paketov od konkrétneho suseda. Do budúcna sa myslí na implementáciu špecifického kódu na zabezpečenie kompatibility so všetkými verzia protokolu EIGRP, preto sa v rámci štruktúry ukladajú aj verzie operačného systému a použitých TLV.

4.2.6 EIGRP pakety

```
struct eigrp_packet
{
  /* Smerník na nasledujúci paket vrámci zreťazeného zoznamu */
  struct eigrp_packet *next;
  /* Smerník na predchádzajúci paket vrámci zreťazeného zoznamu */
  struct eigrp_packet *previous;
  /* Pointer na dáta v pakete */
  struct stream *s;
  /* Cieľová IP adresa paketu */
  struct in_addr dst;
  /* Vlákno zodpovené za opakované posielanie nepotvrdeného paketu*/
  struct thread *t_retrans_timer;
  /* Počet opakovaných poslaní paketu */
  u_char retrans_counter;
  /* Sekvenčné číslo paketu */
  u_int32_t sequence_number;
  /* Dĺžka paketu v bajtoch */
  u_int16_t length;
};
```

Ukážka 4.6 – Štruktúra eigrp_packet

```
struct eigrp_header
{
  /* Jednotlivé polia hlavičky, tak ako boli popísané v časti 1.3.1
*/
  u_char version;
  u_char opcode;
  u_int16_t checksum;
  u_int32_t flags;
  u_int32_t sequence;
  u_int32_t ack;
  u_int16_t vrid;
  u_int16_t ASNumber;
  /* Smerník na príslušné TLV v pakete */
  char *tlv[0];

}__attribute__((packed));
```

Ukážka 4.7 – Štruktúra eigrp_header

Vo všeobecnosti sa obsah paketov v balíku Quagga ukladá bajt po bajte ako tok v štruktúre stream. V EIGRP module je navyše vytvorená obalová štruktúra eigrp_packet, ktorá k samotnému obsahu paketu pridáva ďalšiu funkcionalitu zabezpečujúcu preposielanie,

pridáva mu sekvenčné číslo, cieľovú IP adresu a začleňuje ho do lineárneho zreťazeného zoznamu paketov, určených na poslanie niektorým z rozhraní. Pre vytváranie obsahu paketu sa v súbore eigrp_structs.h nachádzajú definované štruktúry EIGRP hlavičky a všetkých ostatných typov TLV, ktoré je možné kombinovať vrámci špecifických typov paketov.

4.3 Spustiteľný EIGRP démon

Vykonávanie každého programu napísaného v jazyku C začína vo funkcii main. Podľa vzoru ostatných protokolových modulov Quagga bola funkcia main v EIGRP navrhnutá za účelom spustenia EIGRP démona, spracovania všetkých zadaných prepínačov z príkazového riadku a inicializácie celej softvérovej infraštruktúry. Ako môžeme vidieť v ukážke 4.8, program začína nastavením potrebným prístupových práv pre ním vytvorené súbory, načítaním svojho mena a analýzou zadaných prepínačov. Pomocou nich vieme špecifikovať cestu ku konfiguračným súborom, upraviť číslo PID procesu, či definovať používateľa a skupinu, pod ktorým sa program spúšťa. Nasleduje veľké množstvo inicializačných funkcií, ktoré sa starajú o vytvorenie potrebných dátových štruktúr programu, vyžiadanie si informácií o rozhraniach systému, spustenie príkazového riadku, či registráciu všetkých použiteľných príkazov v module. Nakoniec je vykonávanie procesu presunuté do pozadia pomocou funcie daemon() a spustený manažment vlákien. O plánovanie vlákien sa v tomto prípade stará funkcia thread_fetch, bežiaca v nekonečnom cykle while. Platí, že najvyššiu prioritu v procese majú systémové signály, za nimi nasledujú štandardné udalosti vyvolané samotným procesom, ako konzolový vstup a výstup. Za vlákna s nízkou prioritou sú považované časovače a vstupno-výstupné funkcie, starajúce sa o prijímanie/odosielanie paketov. Najnižšiu prioritu majú udalosti a časovače bežiace „na pozadí“.

```
int
main (int argc, char **argv)
{
  ...
  /* Nastavenie prístupových práv pre novo vytvorené súbory */
  umask (0027);
  /* Načítanie názvu programu*/
  progname = ((p = strrchr (argv[0], '/')) ? ++p : argv[0]);
  /* Analýza prepínačov */
  while (1)
```

```
    {
      int opt;
      opt = getopt_long (argc, argv, "df:i:z:hA:P:u:g:vC", longopts,
                         0);
      if (opt == EOF)
        break;
      switch (opt)
        {
        ...
        }
    }
...
  zlog_default = openzlog (progname, ZLOG_EIGRP,
                           LOG_CONS|LOG_NDELAY|LOG_PID, LOG_DAEMON);
  /* Inicializácia celého balíka */
  eigrp_master_init ();
  master = eigrp_om->master;
  zprivs_init (&eigrpd_privs);
  signal_init (master, array_size (eigrp_signals), eigrp_signals);
  cmd_init (1);
  vty_init (master);
  memory_init ();
  eigrp_if_init ();
  eigrp_zebra_init ();
  eigrp_debug_init ();
  eigrp_vty_init ();
  keychain_init();
  eigrp_vty_show_init ();
  eigrp_vty_if_init ();
#ifdef HAVE_SNMP
  eigrp_snmp_init ();
#endif /* HAVE_SNMP */
  access_list_init ();
  access_list_add_hook (eigrp_distribute_update_all_wrapper);
  access_list_delete_hook (eigrp_distribute_update_all_wrapper);
  prefix_list_init ();
  prefix_list_add_hook (eigrp_distribute_update_all);
  prefix_list_delete_hook (eigrp_distribute_update_all);
  distribute_list_init (EIGRP_NODE);
  distribute_list_add_hook (eigrp_distribute_update);
  distribute_list_delete_hook (eigrp_distribute_update);
  /* Načítanie konfiguračného súboru */
  vty_read_config (config_file, config_default);
  if (dryrun)
    return (0);
  /* Presunúť program do pozadia ako démon */
  if (daemon_mode && daemon (0, 0) < 0)
    {
      zlog_err ("EIGRPd daemon failed: %s", strerror (errno));
      exit (1);
    }
  /* Vytvorenie identifikátora procesu */
```

```
  pid_output (pid_file);
  /* Socket pre VTY server shell */
  vty_serv_sock (vty_addr, vty_port, EIGRP_VTYSH_PATH);
  zlog_notice ("EIGRPd %s starting: vty@%d", QUAGGA_VERSION,
               vty_port);
  /* Spustenie plánovania vlákien */
  while (thread_fetch (master, &thread))
    thread_call (&thread);

  return (0);
}
```

Ukážka 4.8 – Vstupná funkcia modulu EIGRP - main

4.4 Prepojenie EIGRP modulu na zebra server

Aby mohla byť zabezpečená komunikácia protokolového modulu so zebra serverom, je potrebné softvérovo prepojiť tieto dva bežiace procesy. Realizuje sa to pomocou vytvorenia inštancie štruktúry zclient a použitím funkcií z knižničného hlavičkového súboru *zebra.h*. Zclient obsahuje smerníky na funkcie, zabezpečujúce informovanie protokolových démonov o zmenách v sieti, o zmenách v stave rozhraní, prípadne pomocou nich dokáže protokolom zaslať redistribuované siete. Vytvorený systém je zaujímavý tým, že zebra server dokáže spúšťať špecifické funkcie, ktoré sú rozdielne implementované v jednotlivých protokoloch, no v každom z nich sú priradené rovnakým smerníkom náležitej inštancie *zclient*.

```
void
eigrp_zebra_init (void)
{
  zclient = zclient_new ();
  zclient_init (zclient, ZEBRA_ROUTE_EIGRP);
  zclient->router_id_update = eigrp_router_id_update_zebra;
  zclient->interface_add = eigrp_interface_add;
  zclient->interface_delete = eigrp_interface_delete;
  zclient->interface_up = eigrp_interface_state_up;
  zclient->interface_down = eigrp_interface_state_down;
  zclient->interface_address_add = eigrp_interface_address_add;
  zclient->interface_address_delete =
eigrp_interface_address_delete;
  zclient->ipv4_route_add = eigrp_zebra_read_ipv4;
  zclient->ipv4_route_delete = eigrp_zebra_read_ipv4;
}
```

Ukážka 4.9 – Inicializačná funkcia klientskej štruktúry zebra.

4.5 Reliable Transport Protocol

Dôležitou funkciou EIGRP je zabezpečenie spoľahlivého doručovania paketov. Keďže protokol nepoužíva niektorý z už existujúcich transportných protokolov, bolo potrebné implementovať vlastný mechanizmus. V ukážke 4.10 môžeme vidieť funkciu zodpovednú za posielanie paketov vyžadujúcich spoľahlivé doručenie. Tieto pakety sú dopredu vytvorené vo funkciách prislúchajúcich ich konkrétnym typom a pridané do frontu paketov retrans_queue, ktorý obsahuje každá štruktúra susedného smerovača. Pri posielaní je vytvorená kópia paketu, ktorá sa umiestni do výstupného frontu rozhrania a je spustený časovač, zabezpečujúci sledovanie potvrdenia paketu. Po jeho uplynutí dôjde k opätovnému odoslaniu nepotvrdeného paketu a zvýšeniu počítadla neúspešných retransmisií. Štandardne, ak sa nepodarí prijať potvrdenie po 16tich retransmisiách, susedstvo so smerovačom je považované za neaktívne a proces jeho nadväzovania prebieha od začiatku.

```
void
eigrp_send_packet_reliably (struct eigrp_neighbor *nbr)
{
  struct eigrp_packet *ep;

  ep = eigrp_fifo_tail(nbr->retrans_queue);

  if (ep)
    {
      struct eigrp_packet *duplicate;
      duplicate = eigrp_packet_duplicate(ep, nbr);
      /* Vloženie paketu do výstupného frontu rozhrania */
      eigrp_fifo_push_head(nbr->ei->obuf, duplicate);

      /* Spustenie retransmit časovača */
      THREAD_TIMER_ON(master, ep->t_retrans_timer,
eigrp_unack_packet_retrans,
           nbr, EIGRP_PACKET_RETRANS_TIME);

      /* Zvýšenie globálneho sekvenčného čísla */
      nbr->ei->eigrp->sequence_number++;

      /* Volanie vlákna pre zápis paketov */
      if (nbr->ei->on_write_q == 0)
        {
          listnode_add(nbr->ei->eigrp->oi_write_q, nbr->ei);
          nbr->ei->on_write_q = 1;
```

```
        }
      if (nbr->ei->eigrp->t_write == NULL)
        nbr->ei->eigrp->t_write =
            thread_add_write(master, eigrp_write, nbr->ei->eigrp,
nbr->ei->eigrp->fd);
    }
}
```

Ukážka 4.10 – Funkcia zodpovedná za odosielanie paketov vyžadujúcich spoľahlivé doručenie

V ukážke 4.11 môžeme vidieť časť funkcie, ktorou prechádza paket po prijatí. Tento kód kontroluje pole Ack v hlavičkách jednotlivých EIGRP paketov. Jednak slúži na detekciu úspešného vytvorenia nového susedstva, kedy je potrebné identifikovať potvrdenie nášho odoslaného Null Update paketu, no v tomto mieste sa vykonáva aj bežná kontrola prijatých Ack paketov. Ak sa hodnota poľa Ack v prijatom pakete zhoduje s hodnotou sekvenčného čísla v spoľahlivo odoslanom pakete, považujeme tento paket za potvrdený a jeho štruktúra je odstránená z frontu paketov čakajúcich na potvrdenie. Ak sa v tomto fronte nachádzajú ďalšie pakety, môžeme po potvrdení predchádzajúceho, odoslať nasledujúci z nich.

```
int
eigrp_read (struct thread *thread)
{
  ...
  /* Kód spracuvávajúci prijaté potvrdenia paketov */
  if (ntohl(eigrph->ack) != 0)
    {
      nbr = eigrp_nbr_get(ei, eigrph, iph);
      assert(nbr);

     struct eigrp_packet *ep;
     ep = eigrp_fifo_tail(nbr->retrans_queue);
     if (ep != NULL)
       {
         if (ntohl(eigrph->ack) == ep->sequence_number)
           {
             if((nbr->state == EIGRP_NEIGHBOR_PENDING) &&
(ntohl(eigrph->ack) == nbr->init_sequence_number))
               {
                 eigrp_nbr_state_set(nbr, EIGRP_NEIGHBOR_UP);
                 zlog_info("Neighbor adjacency became full");
                 nbr->init_sequence_number = 0;
                 nbr->recv_sequence_number = ntohl(eigrph->sequence);
```

```
                  eigrp_update_send_EOT(nbr);
                }
              ep = eigrp_fifo_pop_tail(nbr->retrans_queue);
              if (nbr->retrans_queue->count > 0)
                {
                  eigrp_send_packet_reliably(nbr);
                }
            }
        }
      ep = eigrp_fifo_tail(nbr->multicast_queue);
      if (ep != NULL)
        {
          if (ntohl(eigrph->ack) == ep->sequence_number)
            {
              ep = eigrp_fifo_pop_tail(nbr->multicast_queue);
              eigrp_packet_free(ep);
              if (nbr->multicast_queue->count > 0)
                {
                  eigrp_send_packet_reliably(nbr);
                }
            }
        }
    }
...
}
```

Ukážka 4.11 – Funkcia určená na spracovanie prichádzajúcich EIGRP paketov

4.6 Manažment EIGRP susedstiev

Protokol EIGRP používa dynamickú identifikáciu nových susedov pomocou posielania a prijímania multicast Hello paketov. V Quagga implementácii sa o pravidelné posielanie Hello paketov stará funkcia znázornená v ukážke 4.12. Na vykonávane pravidelne sa opakujúcich udalostí sa v balíku Quagga používajú špeciálne typy vlákien nazývané timer. Pri ich spustení prijímajú ako jeden z parametrov počet sekúnd, po ktorých sa má funkcia vlákna vykonať. V rámci nej je v tomto prípade poslaný Hello paket a vlákno znovu inicializované.

```
int
eigrp_hello_timer (struct thread *thread)
{
  struct eigrp_interface *ei;

  ei = THREAD_ARG(thread);
  ei->t_hello = NULL;
```

```
  if (IS_DEBUG_EIGRP(0, TIMERS))
    zlog (NULL, LOG_DEBUG, "Start Hello Timer (%s) Expire [%u]",
         IF_NAME(ei), EIGRP_IF_PARAM(ei, v_hello));

  /* Posielanie Hello paketu */
  eigrp_hello_send(ei, EIGRP_HELLO_NORMAL);

  /* Časovač pre odosielanie Hello */
  ei->t_hello = thread_add_timer(master, eigrp_hello_timer, ei,
                                 EIGRP_IF_PARAM(ei, v_hello));

  return 0;
}
```

Ukážka 4.12 – Časovač zodpovedný za pravidelné posielanie EIGRP Hello paketov

Ukážka 4.12 zabezpečuje posielanie Hello paketov. Detekcia nového susedného smerovača však prebieha naopak po prijatí tohto paketu. Pri spracovaní tejto udalosti je vytvorená štruktúra nového suseda pomocou funkcie v ukážke 4.13. Po alokácii miesta v pamäti, je vytvorený odkaz na rozhranie, za ktorým sa tento smerovač nachádza a jeho stav je nastavený na DOWN.

```
struct eigrp_neighbor *
eigrp_nbr_new (struct eigrp_interface *ei)
{
  struct eigrp_neighbor *nbr;
  /* Alokácia štruktúry pre nového suseda */
  nbr = XCALLOC (MTYPE_EIGRP_NEIGHBOR, sizeof (struct
eigrp_neighbor));
  /* Vytvorenie odkazu na rozhranie, za ktorým sa sused nachádza */
  nbr->ei = ei;
  /* Stav suseda po vytvorení je DOWN*/
  eigrp_nbr_state_set (nbr, EIGRP_NEIGHBOR_DOWN);
  return nbr;
}
```

Ukážka 4.13 – Funkcia vytvárajúca štruktúru susedného EIGRP smerovača

Vo funkcii znázornenej v ukážke 4.14 sa vykonáva spracovanie Parameter TLV z prijatého Hello paketu. Zhoda v K-hodnotách, prenášaných v tomto TLV je jednou

z hlavných podmienok pre úspešné vytvorenie susedstva. Po kontrole proces pokračuje odoslaním zrýchleného Hello paketu a Null Update. Stav susedného smerovača sa v tomto bode nastaví na PENDING. Pre dokončenie procesu vytvárania susedstva je potrebné prijať Ack paket, potvrdzujúci nami odoslaný Null Update. Vo funkcii eigrp_read z ukážky 4.11 môžeme vidieť spracovanie tohto potvrdenia.

```
static void
eigrp_hello_parameter_decode (struct eigrp_neighbor *nbr,
                              struct eigrp_tlv_hdr_type *tlv)
{
  struct eigrp *eigrp = nbr->ei->eigrp;
  struct TLV_Parameter_Type *param = (struct TLV_Parameter_Type
*)tlv;
  /* Kontrola zhody K-hodnôt potrebnej pre vytvorenie susedstva */
  if ((eigrp->k_values[0] == nbr->K1) &&
      (eigrp->k_values[1] == nbr->K2) &&
      (eigrp->k_values[2] == nbr->K3) &&
      (eigrp->k_values[3] == nbr->K4) &&
      (eigrp->k_values[4] == nbr->K5))
    {
      if (eigrp_nbr_state_get(nbr) == EIGRP_NEIGHBOR_DOWN)
       {
         zlog_info("Neighbor %s (%s) is pending: new adjacency",
                   inet_ntoa(nbr->src), ifindex2ifname(nbr->ei->ifp-
>ifindex));
           /* Odoslanie zrýchleného Hello */
           eigrp_hello_send(nbr->ei, EIGRP_HELLO_NORMAL);
           eigrp_update_send_init(nbr);

         eigrp_nbr_state_set(nbr, EIGRP_NEIGHBOR_PENDING);
       }
    }
...
```

Ukážka 4.14 – Funkcia analyzujúca prijaté Parameter TLV

4.7 Výmena smerovacích informácií

V ukážke 4.15 vidíme funkciu zodpovednú za vytváranie a posielanie Update paketu, ktorý v sebe nesie samotné smerovacie informácie. Po jeho alokácii a vytvorení hlavičky sa do dátového toku, z ktorého je paket tvorený, pridáva v prípade potreby autentifikačné TLV. Nasleduje vloženie Internal a External IPv4 TLV, nesúcich samotnú smerovaciu informáciu.

Aby sme dosiahli oddelenie rozhodovacej roviny DUAL od prenosu dát, funkcia neprehľadáva zakaždým celú topologickú tabuľku, ani nepríjíma ako argument konkrétnu položku topologickej tabuľky. Namiesto toho sú v spojovaných zoznamoch topology_changes_internalIPV4 a topology_changes_externalIPV4 predpripravené smerníky len na tie položky, u ktorých nastala zmena po spracovaní udalosti v DUAL. Nasleduje vypočítanie MD5 sumy, pokiaľ je použitá autentifikácia a vypočítanie *checksum* hodnoty do hlavičky paketu. Posledný krok predstavuje priradenie správnej cieľovej adresy paketu (v tomto prípade ide o multicast adresu EIGRP protokolu), nastavenie správneho sekvenčného čísla a pridanie paketu do frontu spoľahlivo odosielaných paketov.

```
void
eigrp_update_send (struct eigrp_interface *ei)
{
  struct eigrp_packet *ep, *duplicate;
  struct listnode *node, *nnode, *node2, *nnode2;
  struct eigrp_neighbor *nbr;
  struct eigrp_prefix_entry *pe;
  u_char has_tlv;
  u_int16_t length = EIGRP_HEADER_LEN;
  ep = eigrp_packet_new(ei->ifp->mtu);
  /* Príprava Null Update hlavičky */
  eigrp_packet_header_init(EIGRP_OPC_UPDATE, ei, ep->s, 0,
                           ei->eigrp->sequence_number, 0);
  /* Ak je zapnutá autentifikácia, vloženie AuthTLV */
  if((IF_DEF_PARAMS (ei->ifp)->auth_type == EIGRP_AUTH_TYPE_MD5) &&
(IF_DEF_PARAMS (ei->ifp)->auth_keychain != NULL))
    {
      length += eigrp_add_authTLV_MD5_to_stream(ep->s,ei);
    }
  has_tlv = 0;
  for (ALL_LIST_ELEMENTS(ei->eigrp->topology_changes_internalIPV4,
node, nnode, pe))
    {
      if(pe->req_action & EIGRP_FSM_NEED_UPDATE)
        {
          length += eigrp_add_internalTLV_to_stream(ep->s, pe);
          has_tlv = 1;
        }
    }
  for (ALL_LIST_ELEMENTS(ei->eigrp->topology_changes_externalIPV4,
node, nnode, pe))
    {
      if(pe->req_action & EIGRP_FSM_NEED_UPDATE)
        {
          length += eigrp_add_externalTLV_to_stream(ep->s, pe);
```

```
            has_tlv = 1;
          }
      }
    if(!has_tlv)
      {
        eigrp_packet_free(ep);
        return;
      }
    if((IF_DEF_PARAMS (ei->ifp)->auth_type == EIGRP_AUTH_TYPE_MD5) &&
(IF_DEF_PARAMS (ei->ifp)->auth_keychain != NULL))
      {
        eigrp_make_md5_digest(ei,ep->s, EIGRP_AUTH_UPDATE_FLAG);
      }
    /* EIGRP kontrolná suma */
    eigrp_packet_checksum(ei, ep->s, length);
    ep->length = length;
    ep->dst.s_addr = htonl(EIGRP_MULTICAST_ADDRESS);
    /* Nastavenie sekvenčného čísla */
    ep->sequence_number = ei->eigrp->sequence_number;
    if (IS_DEBUG_EIGRP_PACKET(0, RECV))
      zlog_debug("Enqueuing Update length[%u] Seq [%u]",
                 length, ep->sequence_number);
    for (ALL_LIST_ELEMENTS(ei->nbrs, node, nnode, nbr))
      {
        if (nbr->state == EIGRP_NEIGHBOR_UP)
          {
            /* Vloženie paketu do fronty na retransmisie */
            eigrp_fifo_push_head(nbr->retrans_queue, ep);

            if (nbr->retrans_queue->count == 1)
              {
                eigrp_send_packet_reliably(nbr);
              }
          }
      }
}
```

Ukážka 4.15 – Funkcia využívaná na vytváranie a posielanie EIGRP update paketov

4.8 EIGRP autentifikácia

Autentifikácia, ako bolo popísané v časti 1.8, predstavuje pridanú bezpečnosť pri posielaní EIGRP paketov jednotlivým susedom. Je založená na pridaní autentifikačného TLV do celej komunikácie medzi dvoma smerovačmi. Dôležitou úlohou pre zabezpečenie prenosu, a zároveň aj kompatibility s Cisco smerovačmi, bolo implementovať rovnaký postup pri počítaní MD5 zabezpečovacej sumy. Ako môžeme vidieť v ukážke 4.16, tieto výpočty sa líšia podľa toho, o aký typ paketu sa jedná.

MD5 suma sa počíta až po zostavení celého obsahu paketu a keďže sú jeho dáta interne uložené ako celistvý bajtový tok, sťažuje nám to prácu s jednotlivými poliami paketu. Pre vykonanie správnych výpočtov, alebo zápisu výslednej hodnoty na korektné miesto v rámci paketu musíme niekoľko krát umelo presúvať kurzor a označenie konca dátového toku. V ukážke 4.16 na to slúžia funkcie stream_set_getp a stream_set_endp.

```
int
eigrp_make_md5_digest (struct eigrp_interface *ei, struct stream *s,
u_char flags)
{
  struct key *key;
  struct keychain *keychain;
  unsigned char digest[EIGRP_AUTH_TYPE_MD5_LEN];
  MD5_CTX ctx;
  void *ibuf;
  size_t backup_get, backup_end;
  struct TLV_MD5_Authentication_Type *auth_TLV;
  ibuf = s->data;
  backup_end = s->endp;
  backup_get = s->getp;
  auth_TLV = eigrp_authTLV_MD5_new();
  stream_set_getp(s,EIGRP_HEADER_LEN);
  stream_get(auth_TLV,s,EIGRP_AUTH_MD5_TLV_SIZE);
  stream_set_getp(s, backup_get);

  keychain = keychain_lookup(IF_DEF_PARAMS (ei->ifp)-
>auth_keychain);
   if(keychain)
     key = key_lookup_for_send(keychain);
   memset(&ctx, 0, sizeof(ctx));
   MD5Init(&ctx);
  /* Generovanie MD5 sumy. Každý paket vyžaduje iný postup */
   if(flags & EIGRP_AUTH_BASIC_HELLO_FLAG)
     {
       MD5Update(&ctx, ibuf, EIGRP_MD5_BASIC_COMPUTE);
       MD5Update(&ctx, key->string, strlen(key->string));
       if(strlen(key->string) < 16)
         MD5Update(&ctx, zeropad, 16 - strlen(key->string));
     }
   else if(flags & EIGRP_AUTH_UPDATE_INIT_FLAG)
     {
       MD5Update(&ctx, ibuf, EIGRP_MD5_UPDATE_INIT_COMPUTE);
     }
   else if((flags & EIGRP_AUTH_UPDATE_FLAG)
           || (flags & EIGRP_AUTH_QUERY_FLAG))
     {
       MD5Update(&ctx, ibuf, EIGRP_MD5_BASIC_COMPUTE);
```

```
        MD5Update(&ctx, key->string, strlen(key->string));
        if(strlen(key->string) < 16)
          MD5Update(&ctx, zeropad, 16 - strlen(key->string));
        if(backup_end > (EIGRP_HEADER_LEN + EIGRP_AUTH_MD5_TLV_SIZE))
          {
            MD5Update(&ctx, ibuf + (EIGRP_HEADER_LEN +
EIGRP_AUTH_MD5_TLV_SIZE),
                backup_end - 20 - (EIGRP_HEADER_LEN +
EIGRP_AUTH_MD5_TLV_SIZE));
          }
      }
    MD5Final(digest, &ctx);
  /* Vloženie vytvorenej MD5 sumy do AuthTLV */
  memcpy(auth_TLV->digest,digest,EIGRP_AUTH_TYPE_MD5_LEN);
  stream_set_endp(s,EIGRP_HEADER_LEN);
  stream_put(s,auth_TLV,EIGRP_AUTH_MD5_TLV_SIZE);
  stream_set_endp(s, backup_end);
  eigrp_authTLV_MD5_free(auth_TLV);
  return EIGRP_AUTH_TYPE_MD5_LEN;
}
```

Ukážka 4.16 – Funkcia zabezpečujúca počítanie MD5 súm za účelom autentifikácie

4.9 Implementácia redistribúcie

Redistribúcia smerovacích informácií v EIGRP je vykonávaná pomocou External IPv4 TLV. Jeho formát môžeme vidieť na obrázku 1.6. Spracovanie prijatého a odosielanie vytvoreného TLV sa vykonáva analogicky s internými cieľovými sieťami. Rozdiel nastáva pri vykonávaní redistribúcie na Quagga smerovači.

Pre získanie informácii o sieťach z iných protokolov v rámci Quagga infraštruktúry je potrebné poslať požiadavku na zebra server pomocou funkcie eigrp_redistribute_set. Táto funkcia sa volá pri zadaní príkazu redistribute v CLI konfiguračnom móde smerovača Quagga EIGRP. Následne zebra server viacnásobným volaním funkcie v ukážke 4.17, postupne pošle na spracovanie vyžiadané informácie. Je potrebné vytvoriť záznam v topologickej tabuľke a odoslať Update s novými informáciami svojmu okoliu.

```
static int
eigrp_zebra_read_ipv4 (int command, struct zclient *zclient,
                       zebra_size_t length)
{
  struct stream *s;
  struct zapi_ipv4 api;
```

```
  unsigned long ifindex;
  struct in_addr nexthop;
  struct prefix_ipv4 p;
  struct TLV_IPv4_External_type *external_tlv;
  struct eigrp *eigrp;
  s = zclient->ibuf;
  ifindex = 0;
  nexthop.s_addr = 0;
  /* Type, flags, message. */
  api.type = stream_getc (s);
  api.flags = stream_getc (s);
  api.message = stream_getc (s);
  /* IPv4 prefix */
  memset (&p, 0, sizeof (struct prefix_ipv4));
  p.family = AF_INET;
  p.prefixlen = stream_getc (s);
  stream_get (&p.prefix, s, PSIZE (p.prefixlen));
  if (IPV4_NET127(ntohl(p.prefix.s_addr)))
    return 0;
  /* Next-hop, index rozhrania, vzdialenosť, metrika */
  if (CHECK_FLAG (api.message, ZAPI_MESSAGE_NEXTHOP))
    {
      api.nexthop_num = stream_getc (s);
      nexthop.s_addr = stream_get_ipv4 (s);
    }
  if (CHECK_FLAG (api.message, ZAPI_MESSAGE_IFINDEX))
    {
      api.ifindex_num = stream_getc (s);
      ifindex = stream_getl (s);
    }
  if (CHECK_FLAG (api.message, ZAPI_MESSAGE_DISTANCE))
    api.distance = stream_getc (s);
  if (CHECK_FLAG (api.message, ZAPI_MESSAGE_METRIC))
    api.metric = stream_getl (s);
  eigrp = eigrp_lookup ();
  if (eigrp == NULL)
    return 0;
  if (command == ZEBRA_IPV4_ROUTE_ADD)
    {
       ...
    }
  else /* if (command == ZEBRA_IPV4_ROUTE_DELETE) */
    {
       ...
    }

  return 0;
}
```

Ukážka 4.17 – Redistribúcia externých prefixov v Quagga EIGRP

4.10 Ukladanie konfigurácie Quagga EIGRP

Ukladanie konfigurácie síce nie je priamo podstatou tejto práce, no ide o jednu z funkcionalít, ktorá by nemala chýbať na akomkoľvek smerovači. Každý smerovací protokol v rámci balíka Quagga podporuje vytvorenie a uloženie nakonfigurovaných príkazov do externého súboru. Funkcia eigrp_config_write, v ukážke 4.18, sa využíva na 2 účely:

- vypísanie aktuálne bežiacej konfigurácie po zadaní príkazu show running-config
- zapísanie aktuálne bežiacej konfigurácie do externého súboru príkazom write

Výstup sa veľmi podobá na konfiguračný súbor zo zariadení od spoločnosti Cisco, čo značne uľahčuje jeho následné čítanie a používanie.

```
static int
eigrp_config_write (struct vty *vty)
{
  struct eigrp *eigrp;
  int write = 0;

  eigrp = eigrp_lookup ();
  if (eigrp != NULL)
    {
     /* Vypís časti router eigrp */
      vty_out (vty, "router eigrp %d%s", eigrp->AS, VTY_NEWLINE);
      write++;
      if (!eigrp->networks)
        return write;
      /* Router ID */
      if (eigrp->router_id_static != 0)
      {
       struct in_addr router_id_static;
       router_id_static.s_addr = htonl(eigrp->router_id_static);
        vty_out (vty, " eigrp router-id %s%s",
                 inet_ntoa (router_id_static), VTY_NEWLINE);
      }
      /* Výpis sieťovej konfigurácie */
      config_write_network (vty, eigrp);
      /* Výpis konfigurácie na rozhraniach */
      config_write_interfaces (vty, eigrp);
};
```

Ukážka 4.18 – Ukladanie celej konfigurácie smerovača

Konfiguračný súbor protokolov v balíku Quagga sa netvorí priebežne po zadávaní jednotlivých príkazov, ale je vytváraný „na požiadanie“. Úlohou funkcií v ukážkach 4.18 a 4.19 je postupne prezrieť aktuálny stav všetkých častí protokolu a potrebné informácie vypísať na obrazovku, či do súboru vo vopred stanovenom formáte.

```
static int
config_write_network (struct vty *vty, struct eigrp *eigrp)
{
  struct route_node *rn;
  /* Výpis nakonfigurovaných sietí pod EIGRP */
  for (rn = route_top (eigrp->networks); rn; rn = route_next (rn))
    if (rn->info)
      {
        vty_out (vty, " network %s/%d %s",
                 inet_ntoa (rn->p.u.prefix4), rn->p.prefixlen,
VTY_NEWLINE);
      }
  /* Oddelenie od ďalšej konfiguračnej časti pomocou '!' */
  vty_out (vty, "!%s", VTY_NEWLINE);
  return 0;
}
```

Ukážka 4.19 – Funkcia vytvárajúca čiastkový výstup na účely ukladania konfigurácie

5 Testovanie

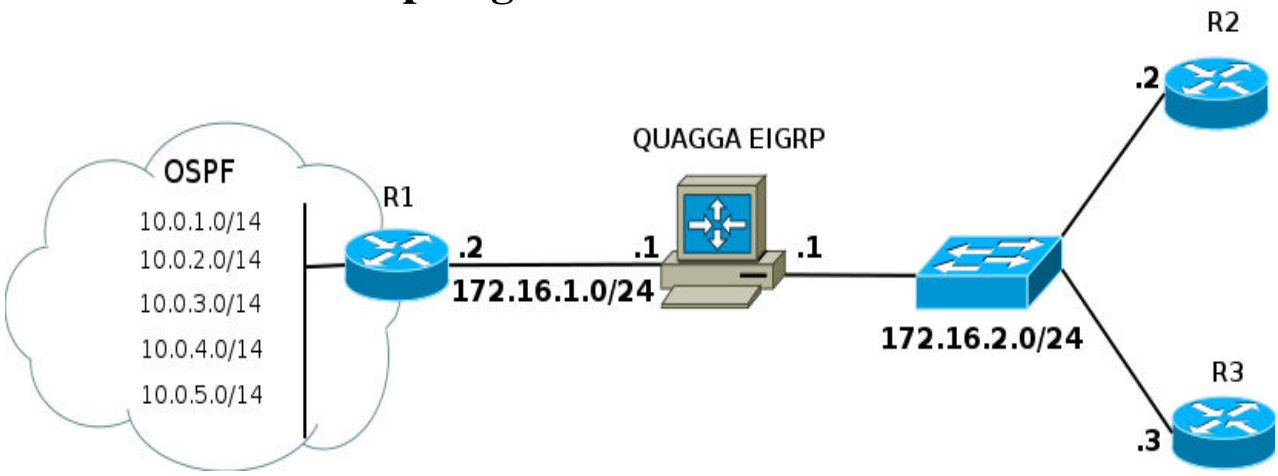
Neoddeliteľnou súčasťou každého softvérového projektu sú fázy priebežného a dôkladného testovania implementovanej funkcionality. Ideálnym spôsobom testovania funkčnosti a kompatibility smerovacieho protokolu EIGRP je využitie samotných hardvérových smerovačov od spoločnosti Cisco. Tie sú však pomerne drahé a nie vždy dostupné v testovacích podmienkach. S cieľom zefektívniť a zrýchliť proces testovania boli využívané aj virtuálne smerovače a topológie, ktoré nám umožňovali okamžité overenie implementovanej funkčnosti v každom štádiu vývoja.

5.1 Dynamips a Dynagen

Dynamips predstavuje softvérový emulátor vytvorený za účelom virtualizovania smerovačov spoločnosti Cisco. Prácu na ňom začal Christophe Fillot v auguste 2005. Aj keď vývoj originálnej verzie sa zastavil koncom roku 2007, zdrojový kód naďalej dobrovoľne rozširuje komunita programátorov združených pod názvom projekt GNS3. Dynamips existuje pre všetky najpoužívanejšie operačné systémy – Windows, Linux a Mac OS.

Dynagen je rozšírením softvéru dynamips o textový príkazový riadok, zjednodušujúci vytváranie a spúšťanie sieťových topológií, či konfiguráciu jednotlivých smerovačov.

5.2 Použitá topológia



Obrázok 5.1 Topológia použitána testovanie Quagga EIGRP

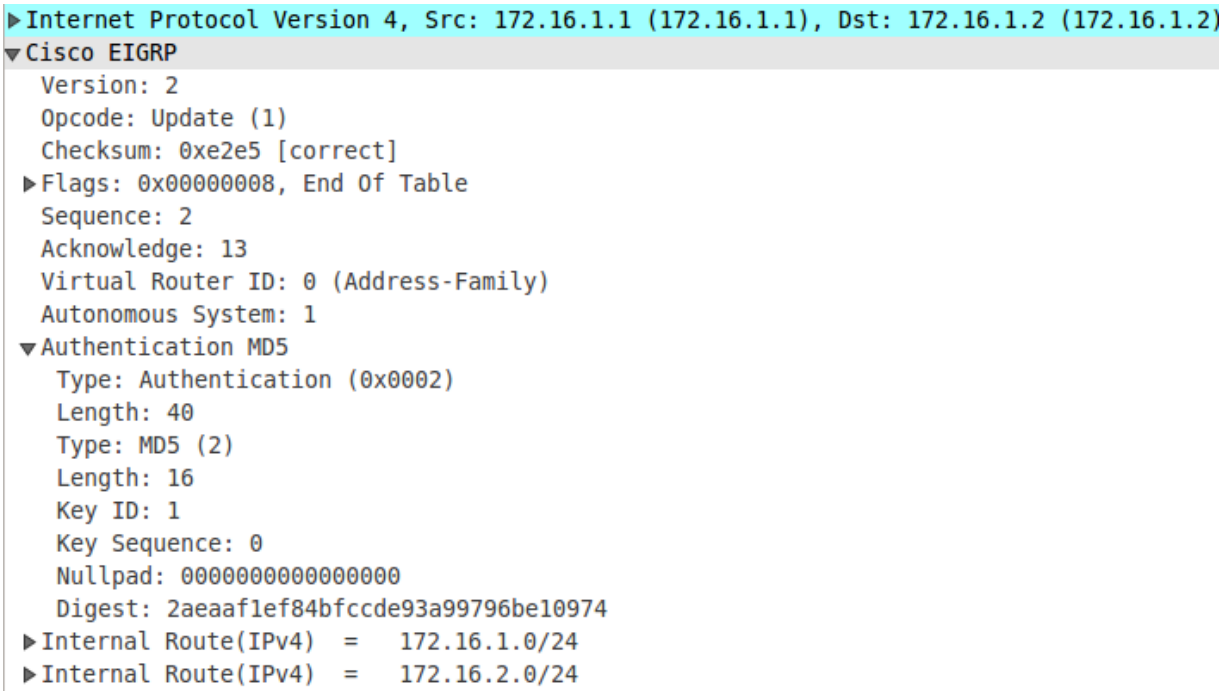
Zjednodušená topológia bola navrhnutá za účelom efektívneho testovania všetkých funkcionalít implementovaných do Quagga EIGRP. Smerovače R1, R2 a R3 predstavujú buď priamo hardvérové zariadenia od spoločnosti Cisco , alebo virtualizované zariadenia Cisco 2400 (pri testovaní IOS verzie 12.4) a Cisco 7200 (pri testovaní IOS verzie 15.2). Na smerovači R1 je spustený aj proces OSPF, čo umožňuje testovanie redistribúcie a správneho prenosu externých smerovacích informácií. Smerovače R2 a R3 sú pripojené na sieti typu broadcast multi-access, čo nám umožňuje testovať správnu funkčnosť multicast prevádzky.

5.3 Ukážka funkčnosti EIGRP modulu

Quagga EIGRP je po mnohých stránkach komplexný softvérový projekt a preto si vyžadoval neustále testovanie postupne pridávaných funkcionalít. Nie je v možnostiach tejto práce predstaviť a detailne popísať celý tento proces. V nasledujúcej časti, na obrázkoch 5.2 až 5.5, môžeme vidieť ukážku výslednej funkčnosti na zapojenej topológii z obr. 5.1. Quagga EIGRP pri zapnutej autentifikácii úspešne vytvorí susedstvo so všetkými Cisco smerovačmi a vymení si smerovacie informácie ako o interných, tak aj o externých sieťach.

604	1307.1474826	172.16.1.2	224.0.0.10	EIGRP	114 Hello
606	1311.9541096	172.16.1.2	224.0.0.10	EIGRP	114 Hello
607	1316.9414276	172.16.1.2	224.0.0.10	EIGRP	114 Hello
609	1319.9425396	172.16.1.1	224.0.0.10	EIGRP	114 Hello
611	1319.9707976	172.16.1.2	224.0.0.10	EIGRP	124 Hello
612	1319.9711126	172.16.1.1	172.16.1.2	EIGRP	94 Update
613	1319.9711516	172.16.1.1	224.0.0.10	EIGRP	114 Hello
614	1319.9810776	172.16.1.2	172.16.1.1	EIGRP	94 Update
617	1321.9717326	172.16.1.1	172.16.1.2	EIGRP	94 Update
618	1321.9913676	172.16.1.2	172.16.1.1	EIGRP	94 Update
619	1321.9916796	172.16.1.1	172.16.1.2	EIGRP	150 Update
620	1322.0216626	172.16.1.2	224.0.0.10	EIGRP	262 Update
621	1322.0218116	172.16.1.1	172.16.1.2	EIGRP	114 Hello (Ack)
622	1322.0418416	172.16.1.2	172.16.1.1	EIGRP	60 Hello (Ack)
623	1322.0519776	172.16.1.2	224.0.0.10	EIGRP	122 Update
624	1322.0522186	172.16.1.1	172.16.1.2	EIGRP	114 Hello (Ack)
625	1324.9440176	172.16.1.1	224.0.0.10	EIGRP	114 Hello
628	1327.0496886	172.16.1.2	224.0.0.10	EIGRP	124 Hello
629	1329.9470636	172.16.1.1	224.0.0.10	EIGRP	114 Hello
631	1331.9353706	172.16.1.2	224.0.0.10	EIGRP	124 Hello

Obrázok 5.2 Vytváranie susedstva medzi smerovačom R1 a Quagga EIGRP démonom



Obrázok 5.3 Ukážka autentifikovaného Update paketu, ktorý posiela Quagga EIGRP

```
root@janovic-N55SF:/home/janovic# 2015/05/04 14:55:25 EIGRP: EIGRPd
0.99.24-rc1 starting: vty@2609
2015/05/04 14:55:25 EIGRP: interface 172.16.1.1 [11] join EIGRP
Multicast group.
2015/05/04 14:55:25 EIGRP: interface 172.16.2.1 [13] join EIGRP
Multicast group.
2015/05/04 14:55:25 EIGRP: Neighbor 172.16.1.2 (veth0) is pending:
new adjacency
2015/05/04 14:55:27 EIGRP: Neighbor adjacency became full
2015/05/04 14:58:38 EIGRP: Vty connection from 127.0.0.1
2015/05/04 14:59:25 EIGRP: Neighbor 172.16.2.2 (veth2) is pending:
new adjacency
2015/05/04 14:59:27 EIGRP: Neighbor adjacency became full
2015/05/04 14:59:48 EIGRP: Neighbor 172.16.2.3 (veth2) is pending:
new adjacency
2015/05/04 14:59:50 EIGRP: Neighbor adjacency became full
```

Obrázok 5.4 Spúšťanie EIGRP procesu a vytváranie susedstiev so smerovačmi R1, R2 a R3

```
eigrpd# sh ip eigrp neighbors

EIGRP neighbors for AS(1)

H   Address           Interface            Hold   Uptime   SRTT  Seq
                                           (sec)           (ms)  Num
0   172.16.1.2        veth0                11     1        0     15
0   172.16.2.2        veth2                12     1        0     5
0   172.16.2.3        veth2                13     1        0     6
```

Obrázok 5.5 Tabuľka aktívnych susedov v Quagga EIGRP

```
eigrpd# sh ip eigrp topology

EIGRP Topology Table for AS(1)/ID(192.168.117.1)

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

P  10.0.1.0/24, 1 successors, FD is 129256,
       via 172.16.1.2 (129256/128256), veth0
P  172.16.1.0/24, 1 successors, FD is 256256
       via Connected, veth0
P  10.0.2.0/24, 1 successors, FD is 129256
       via 172.16.1.2 (129256/128256), veth0
P  192.0.2.0/24, 1 successors, FD is 29160
       via 172.16.1.2 (29160/28160), veth0
P  172.16.2.0/24, 1 successors, FD is 256256
       via Connected, veth2
P  10.0.3.0/24, 1 successors, FD is 129256
       via 172.16.1.2 (129256/128256), veth0
P  10.0.4.0/24, 1 successors, FD is 129256
       via 172.16.1.2 (129256/128256), veth0
P  10.0.5.0/24, 1 successors, FD is 129256
       via 172.16.1.2 (129256/128256), veth0
```

Obrázok 5.6 Topologická tabuľka v Quagga EIGRP

6 Záver

Protokol EIGRP bol do roku 2013 proprietárnym smerovacím protokolom, vytvoreným spoločnosťou Cisco Systems. Napriek jeho výnimočným vlastnostiam ho nebolo možné využiť na zariadeniach ďalších výrobcov. Po ohlásení vydania jeho špecifikácie vo formáte EITF Internet Draft na výročnej konferencii Cisco Live v Londýne sa pre vývojársku komunitu otvorili nové možnosti na jeho implementáciu.

Práve na základe týchto skutočností sme si stanovili za cieľ tejto diplomovej práce vytvoriť svetovo prvú, voľne dostupnú open-source implementáciu smerovacieho protokolu EIGRP. Súčasťou práce bolo podrobné naštudovanie architektúry softvérového balíka Quagga, ktorý bol vybraný ako najvhodnejšia platforma pre implementáciu. Rovnako tak bolo potrebné výsledný softvér dôkladne otestovať.

Ciele diplomovej práce sa nám podarilo naplniť, dokonca prekonať. Nielen že bola vytvorená, v čase písania tejto práce prvá a svetovo jediná funkčná a otvorená verzia smerovacieho protokolu EIGRP, ale svojou prácou sme dokázali zaujať množstvo ľudí z rôznych krajín sveta. Počas vývoja sme dokonca zožali veľké množstvo obdivu a podpory aj od samotnej spoločnosti Cisco Systems. V roku 2014 sme dostali pozvanie na výročnú a najväčšiu sieťovú konferenciu na svete – Cisco Live! US, v kalifornskom San Franciscu, kde boli niekoľkými cestami prezentované výsledky našej práce širšej verejnosti. Jeden z pôvodných autorov Cisco implementácie protokolu EIGRP a držiteľov množstva súvisiacich patentov, Donnie Savage, sa po osobnom stretnutí v San Franciscu rozhodol sám prispieť svojou prácou do nášho projektu. V súčasnosti pôsobí ako softvérový architekt a „technical matter expert“ pre oblasť smerovania. Ide o veľmi unikátnu a nezvyklú kooperáciu medzi študentmi univerzity a svetovou korporáciou, ktorá prejavuje záujem o vytváraný projekt.

Quagga EIGRP má pred sebou ešte množstvo práce, ktorú je potrebné vynaložiť, aby sme dosiahli plnú funkcionalitu, porovnateľnú s aktuálnou verziou protokolu od spoločnosti Cisco. Nakoľko sa však jedná o otvorený softvér, očakávame zapojenie sa množstva ďalších vývojárov, ktorí tvoria komunitu okolo balíka Quagga. Je isté, že EIGRP ako modulárny protokol nájde uplatnenie v IPv4 aj IPv6 sieťach. Ukazuje sa aj však, že vďaka svojej povahe má potenciál preniknúť do iných typov sietí, ktoré nie sú postavené na IP adresovaní. Môže sa jednať o najrôznejšie aplikácie v rýchlo sa rozmáhajúcom sektore M2M, kedy bude protokol

EIGRP použitý na hľadanie najkratších ciest vrámci siete zloženej z rôznych senzorov či menších mikročipov.

Na to, aby mohol ďalší vývoj pokračovať, intenzívne spolupracujeme so správcami balíka Quagga za účelom plného začlenenia kódu do už existujúcej oficiálnej distribúcie.

7 Použité zdroje

[1] GARCIA-LUNES-ACEVES J. J., Loop-free routing using diffusing computations, Journal IEEE/ACM Transactions on Networking (TON) archive, Volume 1 Issue 1, Február 1993

[2] SAVAGE D., SLICE D., NG J., MOORE S., Enhanced Interior Gateway Routing Protocol, 10 Apríl 2014
Dostupné na internete: <https://tools.ietf.org/html/draft-savage-eigrp-02>

[3] RFC 791 [online], Internet Protocol, DARPA Internet Program Protocol Specification, September 1981
Dostupné na internete: <https://www.ietf.org/rfc/rfc791.txt>

[4] GNU General Public License [online], Free Software Foundation Inc., 29 Jún 2007
Dostupné na internete: <https://www.gnu.org/copyleft/gpl.html>

[5] KOCHARIANS Narbik, PALUCH Peter, CCIE Routing and Switching v5.0 Official Cert Guide, Volume 1, Cisco Press, Aug 27, 2014, ISBN-13: 978-1-58714-396-0

[6] Enhanced Interior Gateway Protokol [online], Cisco Systems, Document ID: 16406, 05 Január 2015
Dostupné na internete: <http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html>

[7] URIARTE Yon, Zebra for Dummies, Február 2001
Dostupné na internete: <http://www.nongnu.org/quagga/zhh.html>

[8] ISHIGURO KUNIHIRO, Quagga Routing Suite Manual, 01. Apríl 2011
Dostupné na internete: <http://www.nongnu.org/quagga/docs/docs-info.html>

[9] List of open-source routing platforms[online], 31 Marec 2015
Dostupné na internete: <http://en.wikipedia.org/wiki/List_of_open-source_routing_platforms>

[10] GNU Coding Standards[online], 23. Apríl 2015
Dostupné na internete: <http://www.gnu.org/prep/standards/standards.html>

[11] Open Source Routing[online], Január 2014
Dostupné na internete: <https://www.opensourcerouting.org/about-us/>