ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

DIPLOMOVÁ PRÁCA

Študijný odbor: Aplikované sieťové inžinierstvo – Sieťová infraštruktúra

Matej Perina

Implementácia smerovacieho protokolu EIGRP v balíku Quagga, časť DUAL

Vedúci DP: Ing. Peter Palúch, PhD.

Reg. č. 406/2014 **Október 2014**

Ministerské číslo práce: **28360320152406**

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

zadanie

Abstrakt

PERINA, MATEJ: Implementácia smerovacieho protokolu EIGRP v balíku Quagga, časť DUAL [Diplomová práca]

- Žilinská univerzita v Žiline. Fakulta riadenia a informatiky
- Vedúci: Ing. Peter Palúch, PhD.
- Stupeň odbornej kvalifikácie: Inžinier

Žilina: Fakulta riadenia a informatiky, Žilinská univerzita v Žiline, 2015. 60 strán.

Cieľom diplomovej práce je vytvoriť pre implementáciu protokolu EIGRP do balíka smerovacieho softvéru Quagga časť DUAL a pridružené moduly. Prvá kapitola obsahuje teoretické informácie o možnostiach prenosu údajov v rôznych typoch TLV, forme a mieste ich uloženia, spracovanie údajov do výpočtu klasických aj širokých metrík a nakoniec ich využitie pri výbere optimálnych trás. V druhej kapitole sú uvedené informácie o implementačnej platforme, balíku Quagga. Praktická časť je v tretej kapitole, kde sú rozoberané vzniknuté problémy a požiadavky na funkcionalitu spolu s ich riešením a príslušným implementovaním.

Kľúčové slová: EIGRP, Quagga, DUAL, TLV, metrika

Abstract

PERINA, MATEJ: EIGRP Routing Protocol Implementation in the Quagga Suite, DUAL Part [Diploma thesis]

- The university of Žilina. Faculty of Management, Science and Informatics
- Header: Ing. Peter Palúch, PhD.
- Qualification level: Engineer

Žilina: Faculty of Management, Science and Informatics, The university of Žilina, 2015, 60 pages.

The goal of the diploma thesis is to create DUAL and associated parts for the EIGRP protocol implementation in Quagga routing suite. First chapter contains theoretical information about data transmission possibilities in different types of TLVs, form and place of their storing, their processing into classic and wide metric computation and at the end their usage in process of choosing the optimal routes. Second chapter provides information about implementation platform, Quagga suite. Practical part is in the third chapter, were are discussed encountered problems and functionality requirements along with their solutions and corresponding implementations.

Key words: EIGRP, Quagga, DUAL, TLV, metrics

Poďakovanie

Touto cestou by som rád poďakoval najmä vedúcemu práce Ing. Petrovi Palúchovi, PhD. za jeho odborné rady, venovaný čas a hlavne za prejavenú dôveru a umožnenie podieľania sa na projekte takéhoto významu a dosahu. Ďalej by som rád poďakoval všetkým, ktorý ma podporovali nielen pri doterajšom štúdiu, ktorého vyvrcholením je táto práca.

ČESTNÉ VYHLÁSENIE

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

V Žiline dňa 6. 5. 2015 _____

podpis

Obsah

Zoznam obrázkov

Zoznam ukážok

Zoznam vzorcov

Zoznam použitých skratiek

AFI – Address Family Identifier

API – Application Programming Interface

AS – Autonomous System

BGP – Border Gateway Protocol

CLI – Command Line Interface

D – Distance

DUAL – Diffusing Update Algorithm

EIGRP – Enhanced Interior Gateway Routing Protocol

FC – Feasibility Condition

FD – Feasible Distance

FSM – Finite State Machine

GPL – General Public License

IDRP – Inter-Domain Routing Protocol

IGRP – Interior Gateway Protocol

IP – Internet Protocol

IS-IS – Intermediate System to Intermediate System

MTU – Maximum Transmission Unit

OSPF – Open Shortest Path First

RD – Reported Distance

RID – Router Identificator

RIP – Routing Information Protocol

SAF – Service Advertisement Framework

TID – Topology Identifier

TLV – Type Length Value

TCP – Transmission Control Protocol

VID – Virtual Identifier

ÚVOD

V súčasnej informačnej dobe, keď dátové siete sú všadeprítomné a stávajú sa skôr nevyhnutnosťou ako prepychom pre bežné fungovanie firiem, spoločností alebo jedincov, keď ich nefunkčnosť môže spôsobiť obrovské peňažné škody, sú samozrejme vysoké aj požiadavky na ich spoľahlivosť, a to v oblastiach ako dostupnosť, bezpečnosť, škálovateľnosť či variabilita služieb. Nad všetkými týmito požiadavkami je však jedna rozhodujúca, a to schopnosť doručiť dáta želanému koncovému zariadeniu. Pre tento účel boli vyvinuté smerovače a smerovacie protokoly. Ako už názov napovedá, tieto protokoly sa starajú o vytvorenie znalostnej bázy pre nasmerovanie toku dát správnym smerom, aby dorazili ku zariadeniu ktorému boli určené. Aby však bolo možné spraviť správne rozhodnutie, treba najprv mať informácie o dostupných sieťach a potenciálnych cestách ku nim. Je teda potrebné, aby si zariadenia tieto informácie odovzdávali a šírili ich ku všetkým zariadeniam, ktoré ich potrebujú pre správnu činnosť. A nakoniec, keď už máme všetky potrebné informácie, treba mať jednotný systém hodnotenia kvality ciest, aby sme sa mohli rozhodnúť pre najvhodnejšiu. Toto všetko má každý smerovací protokol uvedené vo svojej špecifikácii, návode, ako naprogramovať zariadenie, aby vedelo daný protokol používať a komunikovať s ostatnými zariadeniami. Špecifikácia protokolu definuje a upresňuje, aké informácie posielať, aký typ paketu použiť pre danú povahu informácie, ako určiť komu ich posielať, akým spôsobom kvantifikovať cesty a veľa ďalších.

Najznámejšími a najpoužívanejšími smerovacími protokolmi sú dnes RIP, OSPF, EIGRP, IS-IS a BGP. Dlhú dobu existovali otvorené špecifikácie pre všetky z nich okrem EIGRP. V roku 2013 sa firma Cisco rozhodla zverejniť špecifikáciu protokolu EIGRP, ktorý bol dovtedy výlučne ich duševným vlastníctvom a bol chránený viacerými patentmi. Práve vypršanie jedného z kľúčových patentov bolo podnetom k uvoľneniu špecifikácie a dať tak možnosť vzniku ďalším, otvoreným implementáciám a rozšíriť tým počet výrobcov zariadení, na ktorých bude možné nasadzovanie tohto protokolu.

Táto práca popisuje implementáciu časti DUAL protokolu EIGRP spolu s niekoľkými ďalšími modulmi do balíka smerovacieho softvéru Quagga. Uvedené sú nevyhnutné teoretické informácie potrebné pre pochopenie vypracovania všetkých častí a základné informácie o smerovacom softvére.

Ďalej sú rozobrané jednotlivé okruhy zadania a popri ich analýze sú okamžite uvedené aj riešenia vzniknutých problémov a ich konkrétna implementácia. Práca pokrýva len funkcionalitu častí nevyhnutných pre vypracovanie.

1 EIGRP

Všetky smerovače používajú v procese smerovania paketov smerovaciu tabuľku, ktorá obsahuje smerovacie informácie o dostupných sieťach. Ak sa v smerovacej tabuľke nenachádza platný záznam o ceste k cieľovej sieti, prechádzajúce pakety do tejto siete budú zahodené. Protokol EIGRP je dynamický *distance-vector* smerovací protokol, pomocou ktorého si smerovače medzi sebou vymieňajú smerovacie informácie. Okrem smerovacej tabuľky si EIGRP vytvára aj ďalšie dve tabuľky:

- Tabuľku susedov
- Topologickú tabuľku

Tieto dve tabuľky sú hlavným zdrojom informácií v procese rozhodovania, ktorú cestu resp. ktorého suseda použiť ako *next-hop* do jednotlivých cieľových sietí. Samotnú logiku rozhodovania predstavuje konečný automat DUAL. Ten zabezpečuje aj to, že preferovaná trasa je zaručene acyklická, že v prípade straty konektivity rozošle potrebné „otázky“ svojmu okoliu a že po ukončení rozhodovania po nastaní zmeny informuje svoje okolie o novovzniknutej situácii.

EIGRP v čase svojho vzniku prevzal dve vlastnosti, ktoré dovtedy boli charakteristické len pre rodinu *link-state* protokolov. Sú to udržiavanie susedských vzťahov a rozposielanie aktualizovaných informácií len na začiatku komunikácie so susedom a v momente detegovania zmien v topológii a nie ich periodické rozposielanie, ako je to v prípade naivných *distance-vector* protokolov. Kvôli tejto skutočnosti býva niekedy nesprávne označovaný ako hybridný. Povahou prenášanej smerovacej informácie, zoznamom sietí a ich metrík bez akejkoľvek dodatočnej topologickej informácie, však EIGRP stále zostáva protokolom typu distance-vector a pri výbere najkratšej cesty sa riadi výlučne metrikou. V tomto prípade ide o číselné vyjadrenie vzdialenosti do cieľovej siete. EIGRP však do metriky započítava aj údaje o rýchlosti a kvalite linky.

1.1 Typy správ

EIGRP pre svoje potreby požíva viacero typov paketov podľa ich účelu. Štruktúra paketov sa však nemení a skladá sa z dvoch hlavných častí. Prvou je EIGRP hlavička.

0	8	16	32
Header Version	Opcode	Checksum	
Flags			
Sequence Number			
Acknowledgement number			
Virtual Router ID		Autonomous system number	

Obrázok 1.1 EIGRP hlavička

Header Version – verzia hlavičky paketu. Aktuálne používaná verzia je 2. Nehovorí o verzii použitých TLV, ale o verzii protokolu ako celku (predchodca EIGRP, protokol IGRP, používal v tomto poli hodnotu 1).

Opcode – určuje typ paketu a povahu prenášaných informácií. Definovaných je 9 typov, používa sa však len 7 z nich:

- 1 Update – obsahuje informácie o sieťach a ich metrikách
- 3 Query – žiadosť o zaslanie prijímateľových informácií o daných sieťach po spracovaní informácií v predmetnej Query
- 4 Reply – odpoveď, obsahuje aktuálne informácie o sieťach požadovaných prostredníctvom Query
- 5 Hello – periodicky posielaný paket, ktorý slúži na objavovanie susedov a neustálom informovaní o vlastnej dostupnosti
- 5 Acknowledgement - používa sa na potvrdzovanie prenosu paketov, má rovnaký *opcode* ako *hello* paket, odlišuje sa nenulovou hodnotou poľa *acknowledgement number*, ktorá je v prípade *hello* vždy nulová
- 10 SIA Query a 11 SIA Reply – špeciálny typ Query a Reply, ktorý sa posiela v prípade dlhého čakania na klasický Reply

Checksum – kontrolná suma pre celý obsah paketu, slúži na kontrolu správnosti prenosu a prijatia paketu.

Flags – určuje špeciálnu obsluhu paketu.

Sequence Number, Acknowledgement number – sekvenčné a potvrdzovacie číslo, používa sa na spoľahlivý prenos paketov.

Virtual Router ID – identifikuje virtuálnu inštanciu smerovača. Pre bežný unicast routing je inštancia virtuálneho smerovača vždy 0. Špeciálne použitia EIGRP, ako napríklad podpora multicastového smerovania v sieťach s multitopologickým smerovaním alebo použitie EIGRP ako protokolu pre synchronizáciu distribuovanej databázy v SAF

(Service Advertisement Framework) využívajú iné hodnoty Toto číslo nie je totožné s číslom Router ID, ktoré je pre každý EIGRP smerovač v EIGRP doméne unikátne

Autonomous system number – číslo autonómneho systému v ktorom sa nachádza odosielateľ paketu, používa sa aj ako nepriama autentifikácia. [1]

Samotné dáta sa prenášajú v druhej časti paketu, ktorou je pole premenlivej dĺžky obsahujúce jeden alebo viacero záznamov typu TLV. Každé TLV sa skladá z troch častí, a to typ (Type), dĺžka (Length) a dáta (Value). (Pozri obrázok 1.2)

V pakete sa môže objaviť viacero TLV rôznych typov a v rôznom poradí, pretože medzi nimi nie sú žiadne závislosti a každé TLV je samostatnou jednotkou.

0	8	16	32
Type high	Type low	Length	
Value (variable length)			

Obrázok 1.2 TLV formát

Type – dvojbajtové pole, v ktorom vyšší bajt slúži na definovanie základného typu TLV a nižší na jeho upresnenie.

Length – pole indikujúce dĺžku celého TLV.

Value – samotné dáta, ich čítanie a spracovávanie závisí od typu TLV. [1]

1.1.1 Typy TLV

Každý typ TLV v sebe prenáša odlišné dáta a využíva sa v rozličných situáciách. V súčasnosti existujú štyri verzie sád TLV, 1.2, 2.0, 3.0 a 4.0. V každej verzii TLV je zavedená samostatná sada (množina) TLV typov. EIGRP smerovače oznamujú podporovanú verziu TLV vo svojich Hello paketoch. Podľa hodnoty poľa *Type high* sa TLV rozdeľujú na nasledujúce kategórie (nasledujúci zoznam neuvádza zastaralé typy TLV pre IPX, AppleTalk a iné):

- Všeobecné typy 0x00 zavedené v 1.2
- IPv4 typy 0x01 zavedené v 1.2, tzv. Classic IPv4 TLV
- IPv6 typy 0x04 zavedené v 1.2, tzv. Classic IPV6 TLV
- SAF 0x05 zavedené v 3.0
- Multi-Protocol typy 0x06 zavedené v 2.0

Verzia TLV 1.2 je spoločným menovateľom všetkých existujúcich EIGRP implementácií. Hoci je verzia 1.2 zastaralá, nebudú do nej pridávané nijaké nové typy TLV

a má sa od jej používania v nových implementáciách EIGRP upustiť, predsa je jej podpora nevyhnutná najmä pre spätnú kompatibilitu so staršími implementáciami EIGRP, ktoré inú verziu TLV nepodporujú. Vo verzii 1.2 majú jednotlivé TLV pre rôzne typy adresových rodín rôzne hodnoty poľa *Type high*.

Verzia TLV 2.0, tzv. Multi-Protocol TLVs, bola zavedená v EIGRP Release 8 okolo verzie operačného systému Cisco IOS 15.1(3)S a 15.2(2)T (verziu „Release" je možné na smerovačoch Cisco overiť výpisom **show eigrp plugins**) a je v súčasnosti preferovanou verziou TLV záznamov pre prenos všetkých smerovacích informácií. Súčasťou TLV 2.0 je aj podpora tzv. širokých metrík (Wide Metrics). Všetky TLV špecifické pre v2.0 nesú v *Type high* hodnotu 0x06. Konkrétny podtyp TLV a formát adresy je vyjadrený v *Type low* a v dátovej časti konkrétneho TLV.

Verzia TLV 3.0 sa nazýva Multi-Topology TLVs a bola istý čas využívaná v Cisco implementáciách EIGRP na oznamovanie smerovacích informácií, ako aj dostupných služieb pre Service Advertisement Framework. Identifikácia TLV 3.0 je o niečo menej priamočiara: *Type high* nesie hodnotu 0x00, ako keby sa malo jednať o všeobecné TLV, avšak *Type low* nesie hodnotu v rozmedzí od 0xf0 do 0xff. Konkrétny formát adresy je vyjadrený v dátovej časti konkrétneho TLV. Od EIGRP Release 8 sú TLV 3.0 považované za zastaralé a nemajú sa v nových implementáciách viac používať s výnimkou spätnej kompatibility.

Verzia TLV 4.0 sa v súčasnosti používa pre Service Advertisement Framework (SAF), využíva Multi-Protocol typy TLV a podľa dostupných zdrojov nie je relevantná pre prenos smerovacích informácií.

Mierne prekvapujúcim dôsledkom je, že odporúčaná verzia pre implementáciu TLV je verzia 2.0. Pôvodná verzia 1.2, ako aj verzia 3.0 sú považované za zastaralé a verzia 4.0 v súčasnosti nemá nijaké relevantné typy TLV pre prenos smerovacích informácií. Samotné typy TLV sú číslované unikátne, t.j. konkrétny typ TLV má len jeden a ten istý význam. Interpretácia konkrétneho typu TLV nezávisí od verzie. Znalosť verzie je však potrebná preto, aby si dva EIGRP smerovače navzájom posielali len také typy TLV, ktoré oba podporujú. Hoci prijatie nepodporovaného TLV v žiadnom prípade nespôsobí chybu, informácia v ňom prenesená bude nevyhnutne ignorovaná.

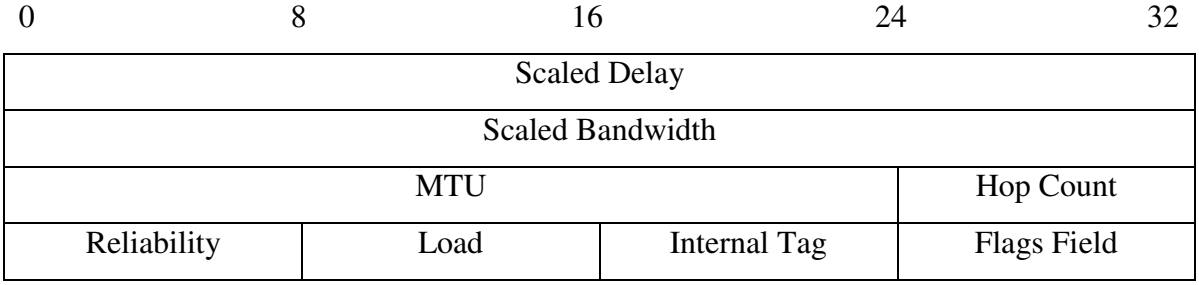
Každá kategória má ešte ďalšie typy definované poli *Type low*. Táto práca pre svoje vypracovanie využíva najmä IPv4 a IPv6 typy, a preto sú popísané bližšie.

Jednotlivé TLV záznamy prenášajúce smerovacie informácie sú v časti Value (dáta) vnútorne štruktúrované a prenášajú v nej položky viacerých typov. V nasledujúcich

podkapitolách budú postupne popísané jednotlivé tieto čiastkové položky, ako aj formát celých TLV typov pre konkrétne typy adresových rodín.

1.1.1.1 Formát klasických metrík

Pod pojmom „klasická metrika“ sa v EIGRP chápe prvotne implementovaná metrika. Jej výpočet v sebe zahŕňa niekoľko parametrov (oneskorenie, šírku pásma, spoľahlivosť a zaťaženie), ktoré si smerovače medzi sebou vymieňajú. Parametre sa prenášajú v klasickom TLV spolu s ďalšími, ktoré sa priamo vo výpočte metriky nepoužívajú. Parametre oneskorenia a šírky pásma sa prenášajú v „*scaled*“ forme, v hodnote, ktorá zodpovedá 256-násobku hodnoty príslušnej metriky použitej vo výpočtovom vzorci. Výpočtový vzorec pre kompozitnú metriku v EIGRP totiž násobí celú vypočítanú hodnotu konštantou 256 a v EIGRP sa čiastkové faktory zodpovedajúce šírke pásma a oneskoreniu prenášajú už vynásobené touto hodnotou.



Obrázok 1.3 Formát klasickej metriky

Scaled Delay – administratívny parameter priraďovaný rozhraniu, formálne vyjadrený v desiatkach mikrosekúnd. Hoci je nazývaný ako „oneskorenie“, nemá nijaký reálny vzťah k prenosovému oneskoreniu na rozhraní daného typu a je len číslom, ktoré môže administrátor modifikovať podľa vlastného uváženia. V poli *Scaled Delay* sa prenáša celkový súčet parametrov Delay rozhraní pozdĺž cesty do cieľa, vynásobený hodnotou 256.

Scaled Bandwidth – obsahuje hodnotu podielu 10^7 / minimálna šírka pásma v kb/s na ceste do cieľa, následne vynásobenú hodnotou 256 .

Hop Count – počet smerovačov na ceste do cieľovej siete.

Reliability –hodnota spoľahlivosti pre najmenej spoľahlivú linku na ceste do cieľovej siete, meraná v mierke od 1 do 255. Hodnota 255 vyjadruje 100% spoľahlivosť.

Load – hodnota zaťaženia najviac zaťaženej linky na ceste k cieľovej sieti meraná v mierke od 1 do 255. Hodnota 255 indikuje 100% zaťaženie.

Internal Tag – hodnota pridelená administrátorom, ktorá je pre EIGRP nedotknuteľná. To umožňuje administrátorovi filtrovať smery na inom EIGRP smerovači na základe tejto hodnoty.

Flag Field – pozri nižšie

Formát **Flags Field -** Má veľkosť jeden bajt a používajú sa spodné tri bity. Prvý bit označovaný aj ako *Source Withdraw* nastavením na jednotku indikuje, že smerovač, ktorý je pôvodným autorom informácie o danej sieti, odvoláva možnosť dostať sa do danej siete..

Druhý bit (*Candidate Default*) indikuje, že smer je kandidátom na *default network*. Výsledná EIGRP *default network* je vybraná spomedzi všetkých oznamovaných *default network* kandidátov na základe metriky. Koncept *default network* je špecifický pre smerovače Cisco a nie je relevantný pre otvorenú implementáciu EIGRP.

Nastavenie tretieho bitu indikuje, že daná sieť je v aktívnom stave. [1]

1.1.1.2 Formát externej sekcie

Tento formát v sebe prenáša dodatočné informácie o cieľových sieťach mimo autonómneho systému EIGRP. Pridáva sa pre siete, ktoré sa do EIGRP dostali redistribúciou.

0 16 24 32

Router Identification		
Autonomous System Number		
Administrator Tag		
External Protocol Metric		
Reserved	Extern Protocol	Flags Field

Obrázok 1.4 Formát externého smeru

Router Identification (RID) – unikátne 32 bitové číslo identifikujúce redistribujúci smerovač, ktorý oznamuje cieľovú sieť do autonómneho systému. Na smerovačoch Cisco sa toto číslo vyberá ako najvyššia IP adresa spomedzi loopback rozhraní a v prípade ich neprítomnosti ako najvyššia IP adresa na zostávajúcich rozhraniach.

Autonomous System Number – číslo pôvodného autonómneho systému, v ktorom sa sieť nachádza.

Administrator Tag - hodnota pridelená administrátorom siete, ktorá je pre EIGRP nedotknuteľná. To umožňuje administrátorovi filtrovať smery na inom EIGRP smerovači na základe tejto hodnoty.

External Protocol Metric – hodnota pôvodnej metriky naučená z cudzieho protokolu. Ak je externým protokolom IGRP alebo iný EIGRP proces, toto pole môže prenášať kompozitnú metriku pôvodného protokolu alebo hodnotu nula, keďže komponenty pôvodnej metriky budú vyjadrené v sekcii TLV s klasickou metrikou podľa Obr. 1.3.

Extern Protocol – definuje externý protokol, z ktorého bola cesta prevzatá. Môže byť:

- IGRP 1
- EIGRP 2
- Static 3
- RIP 4
- HELLO 5
- OSPF 6
- ISIS 7
- EGP 8
- BGP 9
- IDRP 10
- Pripojené 11

Flags Field – pozri kapitolu 1.1.1.1 [1]

1.1.1.3 Formát sieťového prefixu

EIGRP prenáša cieľové siete v komprimovanej forme, kde počet príznačných bitov v poli o premenlivej dĺžke je indikovaný pomocou poľa masky.

0	8
Subnet Mask Bit Count	Destination Address (variable length) ((Bit Count - 1) / 8) +1

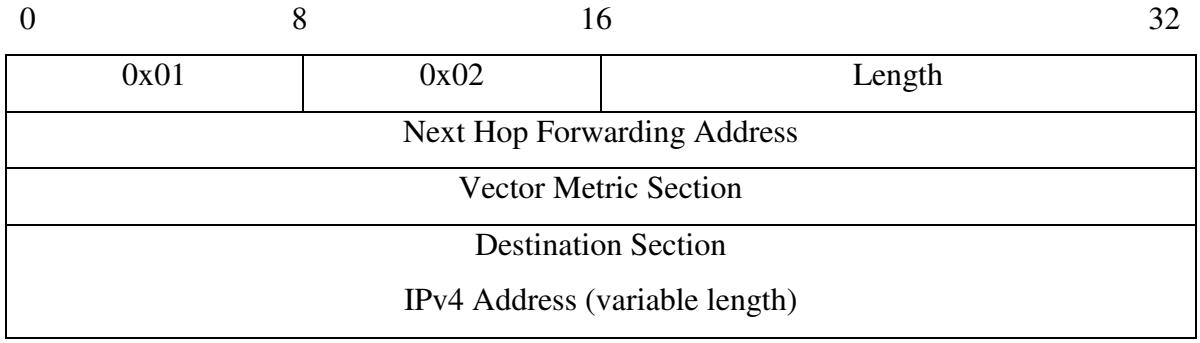
Obrázok 1.5 Formát cieľovej siete

Subnet Mask Bit Count – 8-bitová hodnota používaná na indikáciu počtu bitov v maske adresy. Hodnota 0 hovorí o tzv. default route a žiadna adresa nie je prítomná.

Destination Address – pole o premenlivej dĺžke použité na prenos cieľovej adresy siete. Dĺžka je určená počtom bitov masky zaokrúhlených hore na najbližší oktet.[1]

1.1.1.4 IPv4 INTERNAL_TYPE

Toto TLV prenáša internú IPv4 cieľovú sieť a k nej prislúchajúcu metriku. Siete oznamované v tomto TLV sú siete, v ktorých má smerovač rozhrania, a tiež interné siete naučené od iných EIGRP smerovačov. Hodnota typu pre toto TLV je 0x0102.



Obrázok 1.6 IPv4 INTERNAL_TYPE

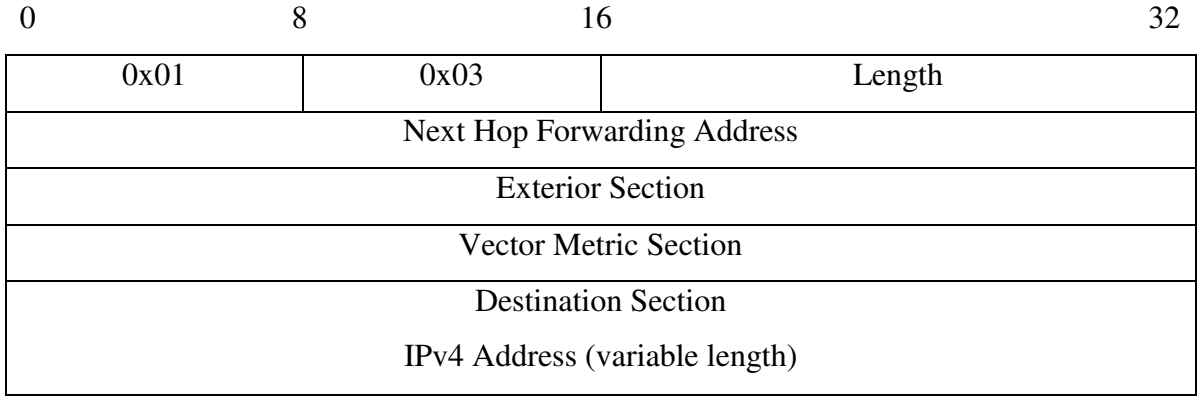
Next Hop Forwarding Address – IPv4 *next-hop* adresa pre cieľovú sieť. Ak je hodnota poľa nulová, ako *next-hop* sa použije IPv4 adresa odosielateľa tejto informácie.

Vector Metric Section – metrika siete. Pozri kapitolu 1.1.1.1

Destination Section – samotná cieľová sieť. Pozri kapitolu 1.1.1.3 [1]

1.1.1.5 IPv4 EXTERNAL_TYPE

TLV prenášajúce externú cieľovú sieť a jej metriku, naučenú z iného smerovacieho protokolu a vloženú do EIGRP. Spolu s touto informáciou sa prenáša identita pôvodného protokolu, externá metrika, číslo AS a administrátorská značka (tag). Hodnota typu pre toto TLV je 0x0103.



Obrázok 1.7 IPv4 EXTERNAL_TYPE

Next Hop Forwarding Address – IPv4 *next-hop* adresa pre cieľovú sieť. Ak je hodnota poľa nulová, ako *next-hop* sa použije adresa odosielateľa tejto informácie.

Exterior Section – prídavné smerovacie informácie pre siete mimo EIGRP AS. Pozri kapitolu 1.1.1.2

Vector Metric Section – metrika siete. Pozri kapitolu 1.1.1.1

Destination Section – samotná cieľová sieť. Pozri kapitolu 1.1.1.3 [1]

1.1.1.6 IPv6 typy

Tieto typy prenášajú cieľovú IPv6 sieť a od IPv4 typov sa líšia iba v dĺžke poľa pre *next-hop* adresu a cieľovú sieť a vyšší bajt poľa *type* má pre IPv6 hodnotu 0x04. Ostatné polia ostávajú rovnaké ako pre IPv4 (obrázky 1.8 a 1.9).

0	8	16	32
0x04	0x02	Length	
Next Hop Forwarding Address (16 octets)			
Vector Metric Section			
Destination Section IPv6 (variable length)			

Obrázok 1.8 IPv6 INTERNAL_TYPE

0	8	16	32
0x04	0x03	Length	
Next Hop Forwarding Address (16 octets)			
Exterior Section			
Vector Metric Section			
Destination Section IPv6 (variable length)			

Obrázok 1.9 IPv6 EXTERNAL_TYPE

1.1.1.7 Multi Protocol typy

Za *multiprotokolové* typy, niekedy prenesene označované aj *wide metrics*, sa považujú TLV zavedené vo verzii 2.0. Tieto typy sú určené na to, aby sa pomocou

jediného typu TLV dali preniesť cieľové siete rôznych smerovaných protokolov (IPv4 a IPv6). Paket obsahujúci TLV verzie 2.0 sa skladá z EIGRP hlavičky a z troch čiastkových častí.

0	8	16	32
Header Version	Opcode	Checksum	
Flags			
Sequence Number			
Acknowledgement number			
Virtual Router ID		Autonomous system number	
TLV Header Encoding			
Wide Metric Encoding			
Destination Descriptor (variable length)			

Obrázok 1.10 Paket obsahujúci TLV 2.0

1.1.1.8 Hlavička TLV 2.0

Hlavička TLV 2.0 je oproti predchádzajúcej verzii 1.2 komplexnejšia a obsahuje dodatočné informácie potrebné pre implementáciu pokročilejších služieb nad EIGRP.

0	8	16	32
Type high	Type low	Length	
AFI		TID	
Router Identifier			
Value (variable length)			

Obrázok 1.11 Hlavička TLV 2.0

Type – vyšší bajt má v prípade *wide metrics* TLV hodnotu 0x06. Nižší bajt môže hovoriť o troch typoch TLV:

- REQUEST_TYPE 0x0601
- INTERNAL_TYPE 0x0602
- EXTERNAL_TYPE 0x0603

Address Family Indetifier (AFI) – definuje typ a formát cieľovej siete. Tento koncept je podobný konceptu, ktorý využíva MP-BGP.

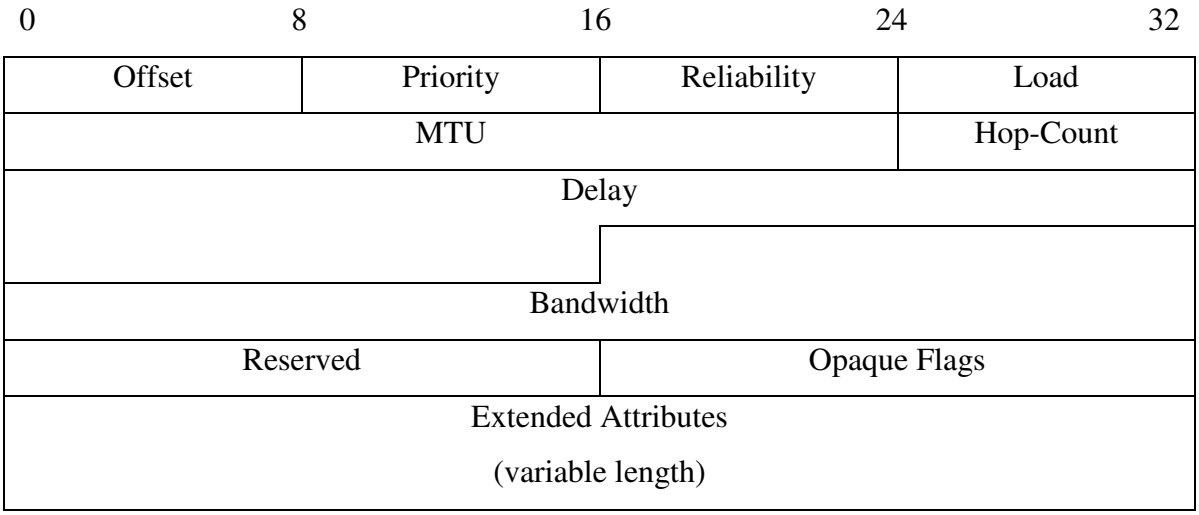
Topology Identifier (TID) – identifikátor topológie pri podpore technológie Multi Topology Routing (MTR).

Router Identifier (RID) – unikátne 32-bitové číslo, ktoré identifikuje zdrojový smerovač oznamujúci cieľovú sieť do autonómneho systému.[1]

1.1.1.9 Formát širokých metrík

TLV 2.0 poskytuje možnosť prenášať okrem bežných EIGRP metrík aj rozšírené sekcie metriky, ktoré nie sú použité v klasickom výpočte metriky. Prídavné informácie sú zahrnuté pre umožnenie počítania metriky z nových, zatiaľ nedefinovaných veličín v budúcnosti.

Dôležitým faktom pri *wide metrics* je, že všetky komponenty metriky sú uvádzané vo svojom „surovej“, t.j. neprepočítanej hodnote, bez násobenia či inej aritmetickej úpravy.



Obrázok 1.12 Formát wide metriky

Offset – počet 16-bitových slov v poli *Extended Attributes*, používa sa na určenie začiatku sekcie s cieľovou sieťou, ktorá nasleduje za týmto záznamom.

Priority – priorita prefixu pri prenose skupiny cieľových adries k susedným smerovačom. Nulová priorita indikuje, že priorita nebola nastavená. V súčasnosti prenášané ako 0.

Reliability –spoľahlivosť pre daný smer meraná v mierke od 1 do 255. Hodnota 255 hovorí o 100% spoľahlivosti.

Load – miera zaťaženia trasy ku cieľovej sieti meraná v mierke od 1 do 255. Hodnota 255 znamená 100% využitie.

MTU – maximálna prenosová veľkosť pre cestu do cieľovej siete. Vo výpočte metriky sa nepoužíva.

Hop-count – počet smerovačov na ceste ku cieľovej sieti.

Delay – súčet oneskorení rozhraní pozdĺž cesty do cieľa vyjadrený v pikosekundách. Podobne ako pri klasických metrikách, hoci je tento parametre nazývaný ako „oneskorenie“, nemá nijaký reálny vzťah k prenosovému oneskoreniu na rozhraní daného typu a je len číslom, ktoré môže administrátor modifikovať podľa vlastného uváženia.

Bandwidth – minimálna šírka pásma pozdĺž cesty do cieľa uvedená v kilobitoch za sekundu. *Reserved* – prenášané ako 0x0000.

Opaque Flags – 16-bitová značka špecifická podľa protokolu.

Extended Attributes – (voliteľné) toto poľa sa za normálnych okolností neprenáša, ale ak je prítomné, obsahuje rozšírené atribúty pre cestu do cieľovej siete. Možné sú:

- NoOp – pre prípadné zarovnanie sekcie na 32 bitov
- *Scaled* metrika – pre spätnú kompatibilitu
- Adminstrátorská značka
- Kolísanie oneskorenia
- Energia
- Pridané cesty – umožňuje EIGRP oznamovať niekoľkonásobné najlepšie cesty (tzv. funkcia Add-Path).[1]

1.2 Topologická tabuľka

Topologická tabuľka obsahuje všetky známe cieľové siete. Do topologickej tabuľky sa cieľová sieť môže dostať dvoma spôsobmi: Buď je priamo pripojená, a teda smerovač má v nej niektoré zo svojich rozhraní, a spadá do rozsahu určeného príkazom *network*, alebo mu ju oznámi sused. [2]

Oficiálna forma záznamu v topologickej tabuľke nie je nikde uvádzaná a závisí od konkrétnej implementácie. Najlepší príklad záznamov nám ponúka výstup príkazu **show ip eigrp topology all-links** na smerovači Cisco.

```
R1# show ip eigrp topology all-links
IP-EIGRP Topology Table for AS(1)/ID(192.168.1.1)
Codes: P - Passive, A - Active, U - Update, Q - Query, R
- Reply, r - reply Status, s - sia Status
P 10.0.0.0/8, 1 successors, FD is 3586560, serno 8
        via 192.168.1.5 (3586560/3074560), Serial0/0
        via 192.168.1.14 (6144000/5632000), Serial0/1
P 192.168.1.8/30, 2 successors, FD is 6144000, serno 12
        via 192.168.1.5 (6144000/5632000), Serial0/0
        via 192.168.1.14 (6144000/5632000), Serial0/1
...
```

Obrázok 1.13 Topologická tabuľka EIGRP na smerovači Cisco

Na obrázku 1.13 je možné si všimnúť základné údaje, ktoré by mal topologický záznam obsahovať.

- Sieť ktorej sa záznam týka,
- jej stav,
- *Feasible distance* (FD),
- zoznam susedov, ktorý nám sieť oznámili,
- ich oznamované vzdialenosti,
- výsledné vzdialenosti cez jednotlivých susedov a ich príznaky,

Údaj *serno* používa interne Cisco pre zefektívnenie odosielania viacerých aktualizovaných informácií naraz, aby sa posielali len údaje o tých sieťach v ktorých nastala zmena.

Údaje o susedoch poskytujúcich cestu ku cieľovej sieti sa menia vždy pri prijatí správy s aktuálnymi informáciami. Vzdialenosti a stav siete má však právo meniť iba DUAL.

1.3 Metrika

V redundantných sieťach má smerovač do cieľovej siete viacero možných ciest. Aby si mohol vybrať optimálnu cestu, potrebuje každej z nich priradiť metriku, ktorá reprezentuje jej výhodnosť. EIGRP priraďuje cestám rad čiastkových metrík, z ktorých vypočítava výslednú kompozitnú metriku. Vzorec na jej výpočet sa líši podľa toho, či sa používajú klasické alebo široké metriky. V oboch prípadoch však platí, že výhodnejšia, a teda preferovanejšia, je cesta s nižšou metrikou.

1.3.1 Štandardná metrika

Vo svojom výpočte používa štyri základné parametre linky, a to:

- Šírku pásma (Bw)
- Oneskorenie (D)
- Zaťaženie (Lo)
- Spoľahlivosť (R)

Všetky sa prenášajú v TLV klasickej metriky (pozri kapitolu 1.1.1.1). Spolu s nimi sa používajú aj K hodnoty, ktoré sú konfiguračne upraviteľné. Vzorec samotný je prevzatý z predchádzajúceho protokolu IGRP a vyzerá nasledovne:

$$Metrika = \left(K1 * Bw_s + \left(\frac{K2 * Bw_s}{256 - Lo_{max}}\right) + K3 * D_s\right) * \left(\frac{K5}{R_{min} + K4}\right)$$

Vzorec 1.1 Štandardná metrika [3]

Aby sa však odlíšilo od predchádzajúcej formy a zväčšila sa citlivosť na jednotlivé parametre používajú sa v nej hodnoty šírky pásma (Bw_s) a oneskorenia (D_s) v takzvanej *scaled* forme. Rozdiel spočíva vo vynásobení parametrov Bw a D konštantou 256 a výsledná metrika má veľkosť 32 bitov a nie 24 ako v prípade IGRP.

Scaled forma šírky pásma je počítaná ako prevrátená hodnota najmenšej šírky pásma pozdĺž celej cesty do cieľovej siete v kilobitoch za sekundu zväčšená 256 krát a vynásobená referenčnou šírkou pásma, ktorá má hodnotu 10^7 (vzorec 1.2).

$$Bw_s = \frac{256 * 10^7}{Bandwidth_{min}}$$

Vzorec 1.2 Scaled šírka pásma [3]

Výsledná hodnota *scaled* oneskorenia vo vzorci je počítaná ako súčet všetkých oneskorení výstupných rozhraní v desiatkach mikrosekúnd po celej dĺžke trasy zväčšený 256 krát (vzorec 1.3). Hodnota oneskorenia 16 777 215 sa používa na indikovanie nedostupnej siete.

$$D_s = 256 * Delay_{summed}$$

Vzorec 1.3 Scaled oneskorenie [3]

Zvyšné dva parametre reprezentujú najväčšie zaťaženie (Lo_{max}) a najmenšiu spoľahlivosť (R_{min}) na linkách po ceste do cieľovej siete. Do výpočtu sa berú bez ďalších zmien.

Konštanty K1-K5 (K hodnoty) sú celočíselné váhové koeficienty od 0 do 255 a ich zmenou ovplyvňujeme dopad jednotlivých komponentov metriky na celkovú metriku. Je nutné, aby všetky smerovače v jednom AS počítali metriku rovnakým spôsobom, a teda aby používali rovnaké K hodnoty. Ak sa zhodovať nebudú, smerovače nebudú schopné nadviazať medzi sebou susedstvo. Predvolené hodnoty sú 1 pre K1 a K3 a 0 pre všetky ostatné. V prípade že je hodnota K5 nulová, pravá časť vzorca, K5/(K4 + R_{min}), sa vo výpočte nepoužije. [3]

Po dosadení *scaled* výrazov zo vzorcov 1.2 a 1.3 a východzích hodnôt koeficientov K môžeme celý výraz upraviť na jednoduchší tvar.

$$Metrika = 256 * \left(\frac{10^7}{Bandwidth_{min}} + Delay_{summed}\right)$$

Vzorec 1.4 Upravená štandardná metrika

1.3.2 Wide metrika

Ako je zrejmé zo vzorca 1.4, linky s rýchlosťou vyššou ako 10 Gbps sa pomocou parametra šírky pásma nedajú diferencovať. Takisto, východzia hodnota oneskorenia na rozhraní s rýchlosťou 1 Gbps je 1 (10 mikrosekúnd) a na rýchlejších rozhrania už neklesá. Šírka pásma aj oneskorenie sa v EIGRP paketoch pri klasickej metrike prenášajú v *scaled* forme (pozri kapitolu 1.1.1.1). To pre smerovače znamená, že pred výpočtom metriky z nich musia spätne vypočítať hodnoty $Bandwidth_{min}$ a $Delay_{summed}$, aby bolo možné vykonať nutnú minimalizáciu šírky pásma a súčet oneskorení. Následne je treba ich previesť späť na *scaled* formy, aby ich bolo možné použiť vo vzorci a oznamovať susedom.

Z týchto dôvodov vývojový tím EIGRP navrhol *wide* metriku, ktorá umožňuje používať viacej parametrov a pomocou prenosu dát v surovej forme sa vyhnúť strate presnosti pri výpočtoch. [3] *Wide* metrika pozostáva z nasledovných parametrov.

- Priepustnosť (T)
- Latencia (La)
- Spoľahlivosť (R)
- Zaťaženie (Lo)
- Rozšírené parametre (ExtM)

Samotný vzorec po pridaní parametrov znie:

$$WideMetrika = \left(K1 * T_{min} + K2 * \frac{T_{min}}{256 - Lo_{max}} + K3 * La_{summed} + K6 * ExtM\right) * \left(\frac{K5}{K4 + R_{min}}\right)$$

Vzorec 1.5 Wide metrika [3]

Analógiou ku šírke pásma v klasickej metrike je priepustnosť (T_{min}). Výsledný tvar používaný vo vzorci je aj tu odvodený od najmenšej šírky pásma prenosových liniek nasledovne:

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

Vzorec 1.6 Priepustnosť [3]

Latencia, čo je parameter analogický ku oneskoreniu, sa tiež po ceste zvyšuje a ešte sa dodatočne upravuje pre potreby diferencovania rýchlejších liniek.

$$La_{summed} = \sum \frac{65536 * Delay_{interface}}{10^6}$$

Vzorec 1.7 Latencia

V prípade *wide* metriky sa ešte líši aj hodnota $Delay_{interface}$ v závislosti od rýchlosti a konfigurácie rozhrania. Môže teda nadobúdať tieto hodnoty:

- Pre rozhrania s rýchlosťou ≤ 1 Gbps alebo v prípade použitia príkazu *bandwidth* je to východzia hodnota oneskorenia rozhrania v pikosekundách
- Pre rozhrania s rýchlosťou > 1Gbps sa počíta ako 10^{13}/východzia šírka pásma rozhrania
- Ak je použitý príkaz *delay,* počíta sa ako 10^7*konfigurované oneskorenie. [3]

Zaťaženie a spoľahlivosť ostávajú rovnaké ako pri štandardnej metrike. Pridaný parameter *ExtM* (rozšírené parametre) v sebe zahŕňa kolísanie a energiu. Ich vplyv riadi parameter K6. EIGRP neposiela tieto metriky, ak ich smerovač nevie používať.

K hodnoty a ich využitie zostávajú rovnaké ako v prípade klasickej metriky. Po dosadení zo vzorcov 1.6 a 1.7 a východzích K hodnôt do vzorca 1.5 dostávame teda zjednodušený vzorec pre výpočet *wide* metriky.

$$WideMetrika = 65536 * \left(\frac{10^7}{Bandwidth_{min}} + \sum \frac{Delay_{interface}}{10^6}\right)$$

Vzorec 1.8 Upravená wide metrika

Východzie hodnoty šírky pásma a oneskorení pre jednotlivé typy rozhraní a metrík sú uvedené vo vydanom drafte [1].

1.4 Feasibility Condition, Feasible Distance, Feasible Successor, Successor

V súčasnosti sa v bežnej praxi nasadzujú redundantné zariadenia a záložné prepoje pre potreby zabezpečenia bezvýpadkového chodu sietí. Spolu s týmito opatreniami však vzniká aj ďalšia neželaná vlastnosť, a to potenciál pre smerovacie slučky. Prítomnosť redundantných prepojov môže viesť ku situácii, že paket sa po niekoľkých smerovacích rozhodnutiach vráti spať ku jednému zo smerovačov, ktorým už prešiel. Znamená to, že cieľové stanice medzi sebou nebudú schopné komunikovať a navyše takéto pakety môžu spôsobiť zahltenie siete.

V smerovacích protokoloch je preto spravidla prítomný mechanizmus, ktorý sa snaží, s rôznou mierou úspešnosti, zabrániť vzniku takýchto situácií. V prípade EIGRP sa tento mechanizmus zakladá na niekoľkých metrikách. Aby bolo možné pochopiť všetky súvislosti, treba poznať niekoľko základných pojmov používaných v EIGRP.

Computed Distance (CD) – označuje výslednú vzdialenosť do cieľovej siete. Dostávame sa k nej pomocou vzorca 1.1 alebo 1.5 podľa typu používanej metriky a počíta sa pre každý susediaci smerovač.

Reported Distance (RD) – je vzdialenosť do cieľovej siete. Pri oznamovaní vzdialenosti susedným smerovačom sa vždy používa aktuálna hodnota RD.

Stavy siete – Cieľová sieť sa môže nachádzať v dvoch typoch stavov, aktívnom a pasívnom. Cieľová sieť je v pasívnom stave ak už bol ukončený proces výberu acyklickej trasy do nej. Aktívny stav hovorí o tom, momentálne nevieme určiť acyklickú trasu a prebieha proces rozhodovania (pozri kapitolu 1.5).

Feasible Distance (FD) – je historicky najmenšej vzdialenosť, ktorú smerovač do cieľovej siete zaznamenal od posledného prechodu siete do pasívneho stavu. V niektorých prípadoch sa môže líšiť od CD.

Feasibility Condition (FC) - je takzvaná podmienka bezslučkovosti. FC využíva koncept RD a FD ako postačujúcu podmienka pre výber acyklickej trasy do cieľa:

„Každý susedný smerovač, ktorého aktuálna vzdialenosť do cieľovej siete je menšia než doposiaľ najnižšia zaznamenaná vzdialenosť aktuálneho smerovača od posledného prechodu tejto siete do pasívneho stavu, poskytuje do tejto siete zaručene acyklickú trasu.“

Presnejšie, susedný smerovač *k* z pohľadu aktuálneho smerovača pre danú cieľovú sieť *j* spĺňa FC, ak jeho momentálna oznamovaná vzdialenosť do siete *j* je menšia ako FD aktuálneho smerovača, čiže ak $RD_j^k < FD_j$.

FC je samotný mechanizmus ktorý zabraňuje vzniku smerovacích slučiek. Je dokázané, že pri dodržiavaní FC pri výbere *next-hop* smerovača pre jednotlivé siete sa nemôže stať, aby vznikla smerovacia slučka. V sieťach s rôznymi typmi prenosových médií alebo rýchlosťami liniek však môže nastať situácia, pri ktorej síce smerovač poskytuje bezslučkovú cestu, avšak FC nespĺňa. Táto situácia sa v prípade potreby rieši tzv. difúznym výpočtom.

Všetky smerovače, od ktorých sme sa dozvedeli o tej istej cieľovej sieti, môžeme podľa spĺňania FC rozdeliť na tri typy.

Feasible Successor (FS) – je označenie pre smerovač, ktorý spĺňa FC.

Successor – je smerovač, ktorý spomedzi všetkých FS poskytuje celkovo najkratšiu cestu. *Successorov* môže byť aj viac. V takom prípade môže smerovač rozkladať toky dát do tej istej cieľovej siete medzi viacerých sucessorov. Tento proces sa nazýva *equal cost load balancing*. Hovoríme o ňom teda v prípade, že viacero smerovačov naraz poskytuje rovnakú najmenšiu vzdialenosť do cieľovej siete. EIGRP však vďaka FC a schopnosti identifikovať FS dokáže sprostredkovať aj *unequal cost load balancing*, čiže rozkladanie tokov dát medzi cesty s rôznymi cenami. Koeficient hovoriaci o tom, koľkokrát môže byť vzdialenosť cez FS horšia než súčasná najmenšia vzdialenosť, aby cestu cez tohto suseda EIGRP ešte vždy použilo, sa nastavuje príkazom **variance**, a to v rozmedzí od 1 do 128. Východzia hodnota maximálneho počtu paralelných ciest v smerovacej tabuľke do toho istého cieľa je 4 a konfiguračne je upraviteľná príkazom **maximum-paths** podľa typu smerovača a verzie operačného systému IOS až na 32.

Posledný typ nemá vlastné označenie a zahŕňa smerovače, ktoré nespĺňajú FC.

1.5 DUAL

Neoddeliteľnou časťou smerovacích protokolov je rozhodovacia logika. EIGRP je principiálne, rovnako ako ostatné smerovacie protokoly typu *distance-vector*, založené na distribuovanej forme Bellman-Fordovho algoritmu najkratších ciest v grafe. V EIGRP je však na rozdiel od iných *distance-vector* protokolov Bellman-Fordov algoritmus len súčasťou väčšieho, nadradeného algoritmu s názvom *Diffusing Update Algorithm* (DUAL).

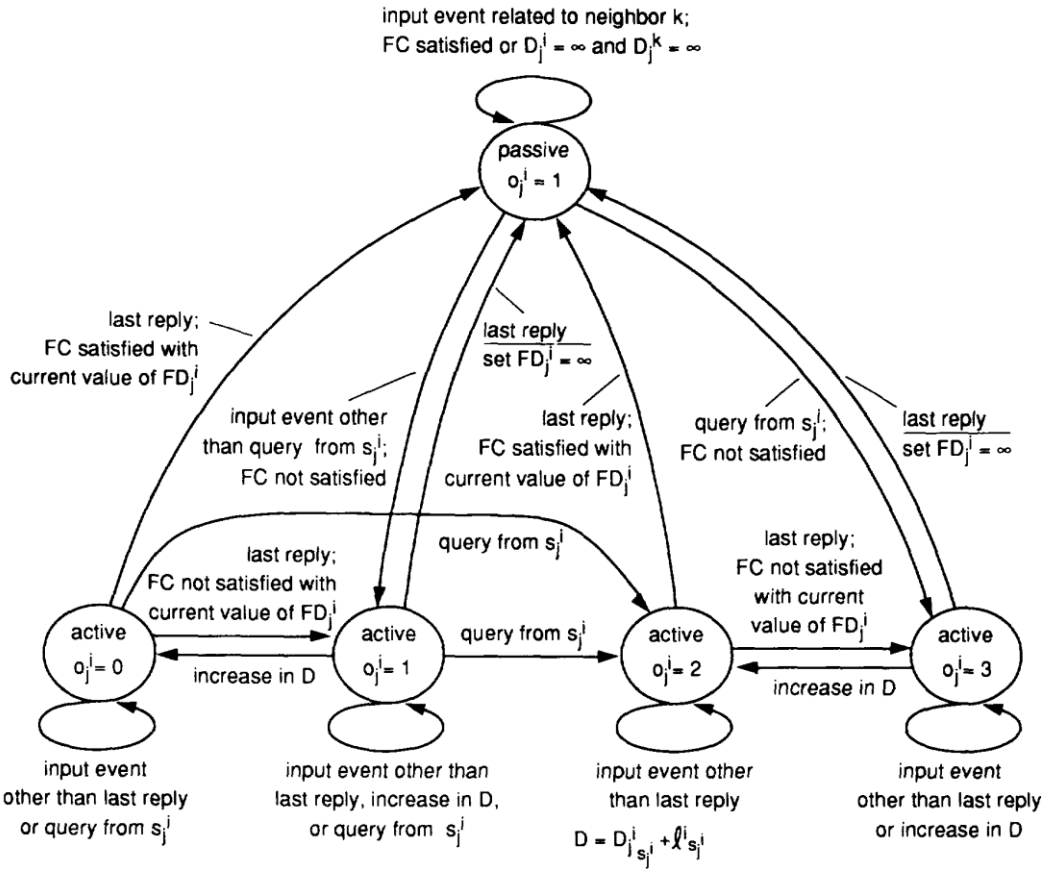
Je to centrum celého protokolu, miesto, kde sa skĺbia všetky vyššie rozoberané časti. Vo svojej povahe je DUAL jednoduchý konečný automat (*finite state machine*).

Úlohou rozhodovacej logiky je vybrať z možných ciest do cieľovej siete tú najvýhodnejšiu a riadiť priebeh celého výberu. Informácie o cestách sú v EIGRP preto sústredené v topologickej tabuľke. Tá je pre DUAL pracovným priestorom. Všetky rozhodnutia, ktoré DUAL spraví, sú výlučne na základe informácií z topologickej tabuľky. Keďže je to programová logika a DUAL robí rozhodnutia na základe operátorov <, >, =, ako rozhodujúci údaj sa používajú čísla, čiže metrika.

DUAL pre svoju činnosť využíva hlavne tri typy správ: *Update*, *Query* a *Reply*. Pomocou *Update* správ smerovač oznamuje okoliu nové informácie o sieti v prípade, že nastala zmena v topológii: zmenili sa vzdialenosti, parametre rozhraní, existujúca sieť zanikla alebo sa objavila nová sieť. Správa *Query* slúži ako žiadosť. Smerovač si pomocou nej vyžiadava aktuálne vzdialenosti o danej sieti od svojho okolia a zároveň oznamuje vlastnú vzdialenosť, ktorú musí okolie dopytujúceho sa smerovača zohľadniť pred tým, ako samo odpovie. Odpovedá sa správou *Reply*.. Každá správa môže odkazovať na viacero sietí naraz.

Rozhodujúce údaje z topologickej tabuľky pre DUAL sú výsledná vzdialenosť (ďalej D), oznamovaná vzdialenosť (ďalej RD) susedných smerovačov a *feasible distance* (ďalej FD). Na základe týchto údajov smerovač robí smerové rozhodnutia a má výhradné právo ich meniť počas svojej činnosti. Hodnota D je vždy aktuálna vzdialenosť do cieľovej siete cez *successora*. Rovnaký postup rozhodovania sa používa v prípade IPv4 aj IPv6 sietí.

Do DUAL-u teda vstupuje cieľová sieť spolu so všetkými aktuálnymi údajmi o možných cestách a na výstupe po spracovaní dostávame popis činnosti, ktorú treba vykonať ako korektnú reakciu na udalosť. Z pohľadu DUAL-u môže byť cieľová sieť v niekoľkých stavoch. Zmenu stavu môžu vyvolať zmeny v topológii, ale nie každá topologická zmena vždy znamená zmenu stavu. Ako už bolo spomenuté, DUAL je vo svojej podstatne konečný automat a jeho stavový diagram je predstavený na obrázku 1.14:



Obrázok 1.14 Stavový diagram automatu DUAL [4]

Autorom tohto diagramu je J. J. Garcia-Lunes-Aceves, ktorý vyvinul celý mechanizmus DUAL-u. Znázorňuje päť stavov, v ktorých sa môže cieľová sieť nachádzať: jeden pasívny a štyri aktívne stavy (0 – 3). V Cisco dokumentoch a vydanom drafte sa môžeme stretnúť s podobným diagramom, odvodeným z tohto, ktorý má rovnaký počet stavov, ale líši sa v počte vstupných udalostí. Funkcionalita je však rovnaká. Implementácia DUAL-u v tejto práci vychádza z pôvodného diagramu hlavne z dôvodu väčšej jednoduchosti a efektívnosti oproti Cisco implementácii.

Spúšťače prechodu medzi stavmi sa nazývajú udalosti. Sú vyvolávané zmenami v topológii. Za udalosť sa považuje napríklad prijatie paketu typu *Update, Query* alebo *Reply* a spracovanie jeho informácií. Samotný DUAL mechanizmus je teda tiež spúšťaný po nastaní niektorej udalosti.

Pre potreby lepšieho pochopenia práce v neskorších kapitolách sú teraz jednotlivé význami stavov a funkcie prechodov popísané detailne.

Pasívny stav - Sieť sa v ňom nachádza, ak o nej máme všetky informácie od susedných smerovačov a bol skončený rozhodovací proces. Je to želaný stav, ktorý hovorí, že cieľová sieť je dostupná. V pasívnom stave môžu nastať tri typy udalostí.

- Prijatie *Query* paketu od successora spôsobujúceho, že najvýhodnejší sused (t.j. smerovač, ktorý ponúka najlepšiu cestu) nespĺňa FC. Táto udalosť znamená, že sám *successor* stratil konektivitu do cieľovej siete. V tom prípade sa musí zmeniť RD, FD a D na aktuálnu vzdialenosť cez súčasného *successora* (prirodzene, tá je zvýšená). Susediacim smerovačom sa posielajú vlastné *Query* pakety obsahujúce aktuálnu zvýšenú RD ako žiadosť o aktuálne informácie a sieť sa presúva do aktívneho stavu 3.
- Udalosť iná ako prijatie *Query* od *successora*, ktorá spôsobí nárast D tak, že najvýhodnejší sused nespĺňa FC. Obdobne ako v predchádzajúcom prípade sa zvýšia RD, FD a D. Sieť sa presúva do aktívneho stavu 1 a rozposiela susedným smerovačom *Query*.
- Pri ostatných udalostiach, ktoré nespôsobia, že najvýhodnejší sused prestane spĺňať FC, sieť ostáva v pasívnom stave. Ak dôjde k poklesu D, klesnú na rovnakú hodnotu aj RD a FD. Ak D narastie, novým *sucessorom* sa stáva najvýhodnejší sused a zvýši sa aj RD. FD sa môže zvýšiť len s prechodom do aktívneho stavu. V obidvoch prípadoch sa nová RD oznamuje všetkým susedom prostredníctvom paketu *Update*.

Aktívny stav 1 – V tomto stave smerovač čaká na prijatie všetkých *Reply* od susedov. Po prijatí posledného *Reply* má smerovač všetky potrebné informácie pre korektný výber novej najkratšej cesty. Najvýhodnejší sused sa stane novým *successorom* a RD, FD a D sa upravia podľa neho. Sieť sa presúva do pasívneho stavu a zmena sa ohlási susedom prostredníctvom paketu *Update*. Ak počas čakania na posledné *Reply* nastane udalosť, ktorá spôsobí opätovné zvýšenie vzdialenosti, sieť sa presúva aj s novou hodnotou D do aktívneho stavu 0. Ak toto zvýšenie spôsobil *Query* paket od *successora*, sieť sa s novou hodnotu D presúva do aktívneho stavu 2. Pri presune do stavov 0 alebo 2 sa RD ani FD nemenia, tie zostávajú na pôvodnej hodnote, ktorú mali pri vstupe do aktívneho stavu 1.

Aktívny stav 3 – Je analogický ku aktívnemu stavu 1. Čaká sa na všetky *Reply*. Ak počas čakania nastane udalosť, ktorá spôsobí ďalší nárast vzdialenosti, sieť sa upravenou hodnotou D presúva do aktívneho stavu 2. Po prijatí posledného *Reply* má smerovač všetky údaje pre korektný výber novej najkratšej cesty. Sieť sa vracia do pasívneho stavu a najlepší sused sa stáva novým *successorom*. Podľa neho sa upravia aj hodnoty D, RD a FD. Ak počas čakania na posledné *Reply* nastane udalosť, ktorá spôsobí opätovné

zvýšenie vzdialenosti, sieť sa presúva aj s novou hodnotou D do aktívneho stavu 2, zatiaľ čo RD a FD zostávajú na pôvodnej hodnote, ktorú mali pri vstupe do aktívneho stavu 3.

Aktívny stav 0 – Sieť v tomto stave čaká na prijatie všetkých *Reply* od susedov. Každý *Reply* sa po prijatí ihneď spracuje a aktualizujú sa údaje v topologickej tabuľke. Po prijatí poslednej *Reply* môžu nastať dve situácie.

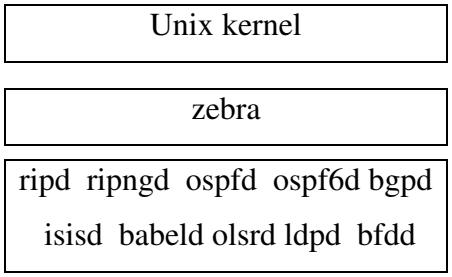
- Najvýhodnejší sused spĺňa FC s aktuálnou hodnotou FD. Vtedy sa sieť vracia do pasívneho stavu a tento sused sa stáva *successorom*. RD sa nastaví na hodnotu novej D a zmena sa oznámi susedom. V prípade, že konečné D je menšie ako FD, nastaví sa aj FD na hodnotu D.
- Najvýhodnejší sused nespĺňa FC s aktuálnou hodnotou FD. V tom prípade sa RD mení na hodnotu D a rozposielajú sa nové *Query* susedom. Sieť prechádza do aktívneho stavu 1.

Ak počas čakania na prijatie posledného *Reply* príde *Query* od *successora*, aktualizuje sa D a sieť sa presúva do aktívneho stavu 2.

Aktívny stav 2 – Ak je sieť v tomto stave, znamená to, že nastali súčasne dve udalosti: Bola prijatá *Query* od *successora* a ešte sa zvýšila hodnota D. Čaká sa na prijatie všetkých *Reply* od susedov. Po prijatí posledného *Reply* môžu nastať dve situácie rovnaké ako v aktívnom stave 0. Najvýhodnejší sused buď spĺňa alebo nespĺňa FC s aktuálnou hodnotu FD. Postupuje sa rovnako ako v aktívnom stave 0 s tým rozdielom, že ak najvýhodnejší sused nespĺňa FC, sieť ide do aktívneho stavu 3.

2 QUAGGA

Quagga je otvorený balík smerovacieho softvéru. Poskytuje implementácie protokolov smerovacích OSPF, RIP, BGP a IS-IS pre unixové platformy, špeciálne pre FreeBSD, Linux, Solaris a NetBSD. Je odvodená od GNU Zebra, ktorú vyvinul Kunihiro Ishiguro. Quagga vetva si však vybudovala viac zainteresovanú komunitu ako centralizovaný model GNU Zebra. Je distribuovaná pod licenciou GNU GPL, ktorá garantuje koncovým používateľom slobodu v používaní, študovaní, zdieľaní a úprave software-u.



Obrázok 2.1 Quagga vrstvový model

Architektúra pozostáva z daemona jadra, zebry a samostatných daemonov jednotlivých protokolov. Zebra vystupuje ako abstraktná vrstva unixového kernelu a poskytuje vyšším protokolom prístup k nemu pomocou Zserv API a TCP tokov. K jednotlivým protokolom sa pristupuje pomocou telnet pripojenia na definované porty a konfigurovateľné sú cez CLI, ktoré sa snaží čo najviac priblížiť CLI, aké poznáme u ostatných smerovacích softvérov.

Pri prispievaní do balíka je potrebné, aby konečné predložené návrhy na zmenu alebo doplnky boli v jazyku C a taktiež je nutné dodržiavať rovnaký programátorský štýl, ako je použitý vo zvyšku balíka. Po schválení zmien a doplnení do oficiálnej distribúcie ich už bude môcť vidieť a využívať každý koncový používateľ. Celý vývoj balíku funguje v systéme na správu verzií Git. Zdrojové súbory sú dostupné v repozitáry na adrese git://git.savannah.nongnu.org/quagga.git. [5]

Oficiálnu wiki stránku o vývoji protokolu EIGRP pre Quaggu môžete nájsť na https://wiki.quagga.net/wiki/index.php/Quagga_EIGRP_daemon_development.

3 VYPRACOVANIE ZADANIA

Zo zadania je zrejmé, že implementácia má byť pre balík Quagga, a preto bola práca vypracovaná v jazyku C. Ako vývojové prostredie bol použitý program Eclipse z dôvodu existencie verzie pre linuxové operačné systémy, podpory pracovného prostredia pre jazyk C a vstavaný plugin pre prácu s repozitármi Git. Použité ukážky teda obsahujú farebnú schému totožnú s týmto programom. Práca využívala vlastnú vetvu oficiálnej distribúcie a vždy aktuálne zdrojové súbory sú dostupné na adrese https://github.com/janovic/Quagga-EIGRP.git.

Zadanie práce sa dá rozdeliť na štyri okruhy.

- Metriky rozhraní
- Topologická tabuľka
- Výmena informácií s pomocou Classic/Wide Metrics TLV
- DUAL

Nasledujúce kapitoly analyzujú každý okruh z hľadiska požiadaviek na vypracovanie, uvádzajú, ako boli riešené vzniknuté problémy a ako vyzerá samotná implementácia.

Spoločnou časťou pre všetky okruhy sú tri hlavičkové súbory, ktoré sú načítavané na začiatku každého súboru. Sú to eigrp_structs.h, ktorý obsahuje všetky použité štruktúry, eigrp_const.h, v ktorom sú definované všetky použité konštanty, a eigrp_macros.h obsahujúci všetky makrá. Všetky súbory, funkcie, konštanty a makrá, ktoré patria do našej implementácie, by sa podľa štýlu kódu používaného v Quagge mali začínať reťazcom eigrp_. V prípade, že to tak nie je, sa jedná buď o knižničné funkcie alebo o ešte nezrevidované časti kódu. Pri začiatku vytvárania implementácie sme použili už existujúci kód na obsluhu niektorých častí z protokolu OSPF preto je možné nájsť kód „bez využitia“. Pri konečnom revidovaní je plánované takéto riadky kódu odstrániť.

3.1 Metrika rozhraní

Cieľom tejto časti bolo vytvoriť funkčný systém, ktorý by umožňoval každému rozhraniu definovať jeho parametre, pristupovať k nim, meniť ich, použiť ich vo výpočte metriky cez jednotlivé rozhrania a realizovať samotný výpočet metriky.

V rámci balíka Quagga existujú dva typy sieťových rozhraní. Jeden typ je poskytovaný zebrou, ktorý predstavuje samotné fyzické rozhranie, a druhý virtuálny, s ktorým pracuje samotný protokol a vnútri sa odkazuje na fyzické rozhranie, ku ktorému

patrí. Parametre linky sú z dôvodu dostupnosti pre protokol uložené na virtuálnom rozhraní. Obsluha rozhraní a ich parametrov je záležitosť všetkých už implementovaných protokolov, preto z veľkej časti je ich obsluha prenesená z OSPF a mierne upravená pre naše potreby.

V prvom kroku bolo potrebné vytvoriť úložisko všetkých parametrov linky. My sme použili nasledovnú štruktúru s názvom *eigrp_if_params* typu *struct*.

```
struct eigrp_if_params
{
  /* EIGRP Interface is passive: no sending or receiving (no need to
     join multicast groups) */
  DECLARE_IF_PARAM (u_char, passive_interface);
  DECLARE_IF_PARAM (u_int32_t, v_hello); /* Hello Interval */
  DECLARE_IF_PARAM (u_int16_t, v_wait);  /*Router HoldTime Interval */
  DECLARE_IF_PARAM (u_char, type);       /* type of interface */
  DECLARE_IF_PARAM (u_int32_t, bandwidth); /* interface bandwidth */
  DECLARE_IF_PARAM (u_int32_t, delay);     /* interface delay  */
  DECLARE_IF_PARAM (u_char, reliability);  /* interface reliability */
  DECLARE_IF_PARAM (u_char, load);         /* interface load */
  DECLARE_IF_PARAM (char *, auth_keychain );  /* Associated keychain
                                                 with interface*/
  DECLARE_IF_PARAM (int, auth_type);   /* EIGRP authentication type */
};
```

Ukážka 3.1 Štruktúra eigrp_if_params

Jednotlivé položky štruktúry sú vytvárané pomocou jedného zo skupiny prebraných makier slúžiacich na prácu s parametrami rozhrania. Použité makro vytvára v štruktúre položky s definovaným typom a názvom.

Všetky prebrané makrá sú uvedené v nasledujúcej ukážke. Aj napriek tomu, že sú implementované, nevyužívame všetky, ale sú k dispozícii pre ďalší vývoj. Nepotrebné makrá budú nakoniec odstránené.

```
#define DECLARE_IF_PARAM(T, P) T P; u_char P##__config:1
#define IF_EIGRP_IF_INFO(I) ((struct eigrp_if_info *)((I)->info))
#define SET_IF_PARAM(S, P) ((S)->P##__config) = 1
#define IF_DEF_PARAMS(I) (IF_EIGRP_IF_INFO (I)->def_params)
#define UNSET_IF_PARAM(S, P) ((S)->P##__config) = 0
#define EIGRP_IF_PARAM_CONFIGURED(S, P) ((S) && (S)->P##__config)
#define EIGRP_IF_PARAM(O, P) \
        (EIGRP_IF_PARAM_CONFIGURED ((O)->params, P)?\
                        (O)->params->P:IF_DEF_PARAMS((O)->ifp)->P)
#define EIGRP_IF_PASSIVE_STATUS(O) \
       (EIGRP_IF_PARAM_CONFIGURED((O)->params, passive_interface) ? \
         (O)->params->passive_interface : \
         (EIGRP_IF_PARAM_CONFIGURED(IF_DEF_PARAMS((O)->ifp), \
            passive_interface) ? \
             IF_DEF_PARAMS((O)->ifp)->passive_interface : \
               (O)->eigrp->passive_interface_default))
```

Ukážka 3.2 Makrá pre prácu s parametrami rozhrania

Makro, ktoré slúži na prístup k jednotlivým parametrom rozhrania pomocou ich názvov, sa používa nasledovne.

```
IF_DEF_PARAMS(struct interface * interface ) -> parameter
```

Ukážka 3.3 Prístup k parametru rozhrania

V ukážke je dôležité si všimnúť, že ako argument makra sa používa *struct interface*, ktorý sa vzťahuje na fyzické rozhranie poskytované zebrou a nie na virtuálne rozhranie protokolu. Tieto dve rozhrania sa totiž pri spustení daemona protokolu úzko spoja. To sa realizuje funkciou, ktorá musí byť vopred definovaná a v nej je potrebné určiť, kam sa parametre linky zapíšu. V našom prípade je to funkcia *eigrp_if_new_hook* (eigrp_interface.c).

```
int
eigrp_if_new_hook (struct interface *ifp)
{
  ifp->info = XCALLOC (MTYPE_EIGRP_IF_INFO,
               sizeof (struct eigrp_if_info));
  /* creating parameters */
  IF_DEF_PARAMS (ifp) = eigrp_new_if_params ();
  /* initalizing parameters to default values */


  SET_IF_PARAM (IF_DEF_PARAMS (ifp), bandwidth);
  IF_DEF_PARAMS (ifp)->bandwidth = (u_int32_t)
               EIGRP_BANDWIDTH_DEFAULT;

  SET_IF_PARAM (IF_DEF_PARAMS (ifp), delay);
  IF_DEF_PARAMS (ifp)->delay = (u_int32_t) EIGRP_DELAY_DEFAULT;

  SET_IF_PARAM (IF_DEF_PARAMS (ifp), reliability);
  IF_DEF_PARAMS (ifp)->reliability = (u_char)
               EIGRP_RELIABILITY_DEFAULT;

  SET_IF_PARAM (IF_DEF_PARAMS (ifp), load);
  IF_DEF_PARAMS (ifp)->load = (u_char) EIGRP_LOAD_DEFAULT;

  return 0;
}
```

Ukážka 3.4 eigrp_if_new_hook

Ako počiatočné hodnoty sú použité nasledujúce konštanty.

```
#define EIGRP_BANDWIDTH_DEFAULT             10000000
#define EIGRP_DELAY_DEFAULT                 1000
#define EIGRP_RELIABILITY_DEFAULT           255
#define EIGRP_LOAD_DEFAULT                  1
```

Ukážka 3.5 Východzie parametre rozhraní

Ďalej bolo potrebné uložené parametre použiť vo výpočte metriky. Výpočet však používa parametre v *scaled* forme a takisto TLV klasickej metriky prenáša viac parametrov, než aké sú pre rozhranie definované. Okrem parametrov razhrania aj *hop count*, *MTU*, *internal tag* a *flags field* (pozri obrázok 1.3). Bolo teda výhodné vytvoriť ďalšiu štruktúru, v ktorej sa ukladajú parametre priamo použiteľné pre výpočet. Vyzerá nasledovne:

```
struct eigrp_metrics
{
  u_int32_t delay;
  u_int32_t bandwith;
  unsigned char mtu[3];
  u_char hop_count;
  u_char reliability;
  u_char load;
  u_char tag;
  u_char flags;
};
```

Ukážka 3.6 eigrp_metrics

Môžeme si všimnúť, že poradie položiek v štruktúre je zhodné s poradím parametrov v TLV klasickej metriky (obrázok 1.3). Je to z dôvodu, aby sa celá štruktúra dala efektívne využiť pri odosielaní týchto TLV. Aby sme však mohli položky štruktúry priamo použiť vo výpočte, je potrebné hodnoty *delay* a *bandwidth* ukladať do *scaled* formy. To by nebol problém, ak by sme používali iba hodnoty prijaté cez TLV od susedov. Keďže je ale potrebné najprv zvýšiť hodnotu *delay* a určiť najmenšiu šírku pásma, je potrebné prijaté hodnoty previesť do surovej formy, pomocou parametrov liniek ich upraviť a nakoniec ich previesť späť do *scaled* formy. Na prevod medzi jednotlivými formami sa používajú nasledujúce funkcie (eigrp_interface.c) odvodené zo vzorcov 1.2 a 1.3:

```
u_int32_t
eigrp_bandwidth_to_scaled (u_int32_t bandwidth)
{
  u_int64_t temp_bandwidth = (256ull * 10000000) / bandwidth;

  temp_bandwidth =
      temp_bandwidth < EIGRP_MAX_METRIC ? temp_bandwidth :
EIGRP_MAX_METRIC;

  return (u_int32_t) temp_bandwidth;

}

u_int32_t
eigrp_scaled_to_bandwidth (u_int32_t scaled)
{
  u_int64_t temp_scaled = scaled * (256ull * 10000000);

  temp_scaled =
      temp_scaled < EIGRP_MAX_METRIC ? temp_scaled : EIGRP_MAX_METRIC;

  return (u_int32_t) temp_scaled;
}

u_int32_t
eigrp_delay_to_scaled (u_int32_t delay)
{
  return delay * 256;
}

u_int32_t
eigrp_scaled_to_delay (u_int32_t scaled)
{
  return scaled / 256;
}
```

Ukážka 3.7 Premena parametrov z a do scaled formy

K výpočtu potrebujeme ešte K hodnoty, ktoré sú globálne a rovnaké pre všetky rozhrania. Sú preto definované v štruktúre celého eigrp a cez ňu sa k nim aj pristupuje.

```
struct eigrp
{
  ...
  u_char    k_values[6];    /*Array for K values configuration*/
  ...
};
```

Ukážka 3.8 K hodnoty

Samotný výpočet metriky podľa vzorca 1.1 je rozdelený po častiach. Ak je niektorá K hodnota nulová, výpočet ku nej sa vzťahujúci sa nevykonáva. Funkcia sa nachádza v súbore eigrp_network.c.

```
u_int32_t
eigrp_calculate_metrics(struct eigrp *eigrp, struct eigrp_metrics
*metric)
{
  u_int64_t temp_metric;
  temp_metric = 0;

  if(metric->delay == EIGRP_MAX_METRIC)
    return EIGRP_MAX_METRIC;

  // EIGRP Metric = {K1*BW+[(K2*BW)/(256-load)]
  // +(K3*delay)}*{K5/(reliability+K4)}

  if (eigrp->k_values[0])
    temp_metric += (eigrp->k_values[0] * metric->bandwith);
  if (eigrp->k_values[1])
    temp_metric += ((eigrp->k_values[1] * metric->bandwith)
        / (256 - metric->load));
  if (eigrp->k_values[2])
    temp_metric += (eigrp->k_values[2] * metric->delay);
  if (eigrp->k_values[4]) {
    temp_metric *= eigrp->k_values[4];
    temp_metric /= (metric->reliability + eigrp->k_values[3]);
  }

  if (temp_metric <= EIGRP_MAX_METRIC)
    return (u_int32_t) temp_metric;
  else
    return EIGRP_MAX_METRIC;
}
```

Ukážka 3.9 Výpočet metriky

Vy výpočte je zohľadnená skutočnosť, že nedostupná sieť sa oznamuje maximálnou hodnotou oneskorenia. V takom prípade sa okamžite vracia hodnota maximálnej metriky. Takisto bolo treba vziať do úvahy skutočnosť, že pri výpočte by mohlo dôjsť k pretečeniu hodnôt. Aby taká situácia nenastala je dočasná premenná, v ktorej sa spočítava výsledok, typu u_int64_t, ktorý má 64 bitov. V prípade, že výsledná hodnota metriky v dočasnej premennej je väčšia ako maximálna možná hodnota EIGRP metriky, vracia sa hodnota maximálna hodnota, v opačnom prípade sa vracia vypočítaná metrika, ale už o veľkosti 32 bitov. Maximálne hodnoty používané pri výpočte metriky sú definovaná nasledovne.

```
#define EIGRP_MAX_METRIC   0xffffffffU    /*4294967295*/
#define EIGRP_MAX_DELAY    0xffffffU      /*16777215*/
```

Ukážka 3.10 Maximálne hodnoty

Táto funkcia však spočítava iba údaje zahrnuté v štruktúre eigrp_metrics. Skutočný výpočet celkovej vzdialenosti zohľadňujúci aj sumáciu oneskorení a minimalizáciu šírky pásma, ktorý využíva výpočet z ukážky 3.9, je nasledujúci (eigrp_network.c):

```
u_int32_t
eigrp_calculate_total_metrics(struct eigrp *eigrp,
    struct eigrp_neighbor_entry *entry)
{
  entry->total_metric = entry->reported_metric;

  u_int64_t temp_delay = (u_int64_t) entry->total_metric.delay
      + (u_int64_t) eigrp_delay_to_scaled(EIGRP_IF_PARAM (entry->ei,
             delay));
  entry->total_metric.delay =
      eigrp_scaled_to_delay(temp_delay) > EIGRP_MAX_DELAY ?
           eigrp_delay_to_scaled(EIGRP_MAX_DELAY) :
                 (u_int32_t)temp_delay;

  u_int32_t bw = eigrp_bandwidth_to_scaled(EIGRP_IF_PARAM (entry->ei,
                bandwidth));
  entry->total_metric.bandwith =
      entry->total_metric.bandwith > bw ? bw :
          entry->total_metric.bandwith;

  return eigrp_calculate_metrics(eigrp, &entry->total_metric);
}
```

Ukážka 3.11 Výpočet celkovej metriky

V tomto vzorci je rovnako ako v predchádzajúcom ošetrený prípad pretečenia pri sumácií oneskorení dočasnou 64 bitovou premennou.

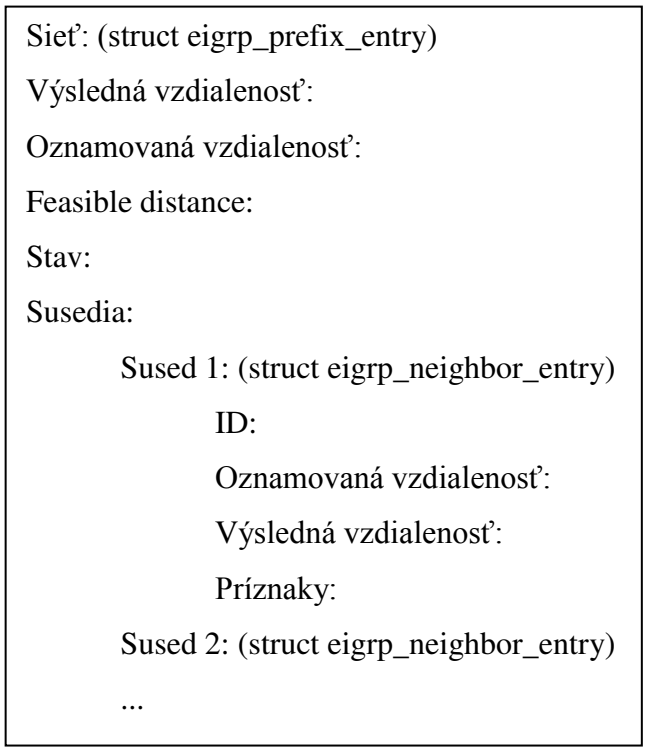
Po implementovaní funkcionality TLV pre *wide* metriky bude možné analogicky pridať aj funkcie pre výpočet *wide* metrík z aktuálnych a doplnkových parametrov.

3.2 Topologická tabuľka

Táto časť práca sa zameriava na implementáciu topologickej tabuľky, jej štruktúr a všetkých funkcií, ktoré umožňujú rýchlu a efektívnu prácu s jej položkami. Na topologickú tabuľku sú kladené vysoké nároky, pretože poskytuje pre DUAL pracovný priestor. Všetky informácie o známych sieťach v nej budú uložené, a preto musí byť prispôsobená týmto požiadavkám

Z obrázku 1.13 vyplýva, že topologická tabuľka je vytváraná ako zreťazený zoznam záznamov. Každý záznam hovorí o cieľovej sieti ako celku a obsahuje ďalší zoznam záznamov, ktoré hovoria o vzdialenostiach do cieľovej siete cez jednotlivých

susedov. Náš záznam o cieľovej sieti sa nazýva *struct eigrp_prefix_entry* a záznam o susedovi *struct eigrp_neighbor_entry*. Jeden záznam si môžeme predstaviť nasledovne:



Obrázok 3.1 Štruktúry eigrp_prefix_entry a eigrp_neighbor_entry v topologickej tabuľke

Aby bolo možné vytvoriť celý záznam, potrebujeme najskôr špecifikovať štruktúru záznamov o susedoch. Počas doby implementácie prišli z časti projektu obstarávajúcej spracovávanie a odosielanie paketov požiadavky ukladať okrem informácií z obrázka 3.1 aj dodatočné informácie. Aktuálne implementovaná štruktúra vyzerá nasledovne:

```
struct eigrp_neighbor_entry
{
  struct eigrp_prefix_entry *prefix;  // parent prefix
  u_int32_t reported_distance;     //distance reported by neighbor
  u_int32_t distance;              //D -sum of reported distance and
                                   // link cost to advertised neighbor

  struct eigrp_metrics reported_metric; //RD in eigrp_metrics form
  struct eigrp_metrics total_metric;    //D in eigrp_metrics form
      //ip address of advertising neighbor
  struct eigrp_neighbor *adv_router;
     //used for marking successor and FS
  u_char flags;

  struct eigrp_interface *ei;    //pointer for case of connected entry

};
```

Ukážka 3.12 eigrp_neighbor_entry

Význam polí by mal byť zrejmý z komentárov v ukážke 3.12. Pole *flags* má veľkosť 8 bitov a ich využitie ako príznaky popisuje nasledujúci obrázok.

8 0

				External Flag	In Routing Table Flag	Feasibe Successor Flag	Successor Flag

Obrázok 3.2 Pole flags

Výsledný záznam obsahujúci všetky informácie o cieľovej sieti je definovaný nasledovne:

```
struct eigrp_prefix_entry
{
  struct list *entries,                    // eigrp_neighbor_entry list
              *rij;
  u_int32_t fdistance;                      // FD
  u_int32_t rdistance;                      // RD
  u_int32_t distance;                       // D
  struct eigrp_metrics reported_metric;     // RD for sending

  u_char nt;                                //network type
  u_char state;                             //route fsm state
  u_char af;                                // address family
  u_char req_action;                        // required action

  struct prefix_ipv4 *destination_ipv4;     // pointer to ipv4 address
  struct prefix_ipv6 *destination_ipv6;     // pointer to ipv6 address

  /*If network type is REMOTE_EXTERNAL, pointer will have reference to
    its external TLV*/
  struct TLV_IPv4_External_type *extTLV;
  /*Serial number for this entry. Increased with each change of entry*/
  u_int64_t serno;
};
```

Ukážka 3.13 eigrp_prefix_entry

Celý zoznam záznamov o susedoch pre danú sieť sa nachádza pod premennou *entries*. Pre vytvorenie tohoto zoznamu sme použili štruktúru *list* definovanú v knižniciach Quaggy. Táto ponúka okrem možnosti využitia ku nej definovaných makier a funkcií aj možnosť zoraďovania jednotlivých záznamov pri pridávaní do zoznamu podľa vlastne stanoveného kritéria. Túto vlastnosť sa podarilo veľmi zaujímavo využiť. Pri bližšom pohľade na fungovanie stavového automatu v DUALe totiž prídeme na skutočnosť, že kritická podmienka pri rozhodovaní je, či sused s najlepšou poskytovanou metrikou spĺňa FC. Určite by teda bolo potrebné vytvárať funkciu, ktorá by prechádzala zoznamom záznamov o susedoch a hľadala ho. Ak však využijeme možnosť zoraďovania záznamov a ako kritérium stanovíme, že záznam s menšou metrikou bude vždy vyššie, dosiahneme,

že záznam obsahujúci suseda s najlepšou metrikou bude vždy na vrchu zoznamu a môžeme k nemu pristupovať priamo. Tým, že nebudeme opakovane prehľadávať zoznam, sa rýchlosť vykonávania niekoľkokrát zvýši. Je definované, že podmienka porovnávajúca záznamy môže vracať iba tri hodnoty.

- 1 v prípade, že záznam 1 by sa mal zaradiť pod záznam 2.
- -1 ak záznam 1 má byť nad záznamom 2.
- 0 ak sú obidva záznamy rovnaké, vtedy sa záznam 1 zaradí pod záznam 2.

Definovaná podmienka pre porovnávanie záznamov o susedoch vyzerá nasledovne (eigrp_topology.c):

```
static int
eigrp_neighbor_entry_cmp(struct eigrp_neighbor_entry *entry1,
    struct eigrp_neighbor_entry *entry2)
{
  if (entry1->distance < entry2->distance)  // comparison of entries
    return -1;                    // actually set to sort by distance
  if (entry1->distance > entry2->distance)
    return 1;

  return 0;
}
```

Ukážka 3.14 Zoraďovanie záznamov o susedoch

Rovnaký spôsob zoraďovania používame aj pri záznamoch o cieľovej sieti. I keď sa zatiaľ nenašlo využitie zoradenia aj týchto záznamov, dá sa ním detegovať pridávanie už existujúcej cieľovej siete a odhaliť tak chyby kódu. Záznamy o sieťach teda zoraďujeme na základe prefixu, čím menší prefix, tým vyššie bude záznam v zozname. Zároveň na základe protokolu sú záznamy o IPv4 sieťach vyššie ako záznamy IPv6 sietí. Podmienku sme definovali nasledovne (eigrp_topology.c):

```
static int
eigrp_prefix_entry_cmp(struct eigrp_prefix_entry *node1,
    struct eigrp_prefix_entry *node2)
{
  if (node1->af == AF_INET)
    {
      if (node2->af == AF_INET)
        {
          if (node1->destination_ipv4->prefix.s_addr
              < node2->destination_ipv4->prefix.s_addr)
            {
              return -1; // if it belong above node2
            }
          else
            {
              if (node1->destination_ipv4->prefix.s_addr
                  > node2->destination_ipv4->prefix.s_addr)
                {
                  return 1; //if it belongs under node2
                }
              else
                {
                  return 0; // same value... ERROR...in case of adding
                            // same prefix again
                }
            }
        }
      else
        {
          return 1;
        }
    }
  else
    {
      return 1; // add to end
    }
}
```

Ukážka 3.15 Zoraďovanie záznamov o sieťach

Samotná topologická tabuľke je nakoniec už len zreťazený zoznam záznamov o sieťach a keďže je len jedna, je definovaná globálne v štruktúre celého eigrp.

```
struct eigrp
{
  ...
  struct list *topology_table;
  ...
};
```

Ukážka 3.16 Definícia topologickej tabuľky

Aktuálne funkcie, ktoré poskytujú nástroje pre prácu s prvkami topologickej tabuľky, môžeme nájsť v súbore eigrp_topology.c. Na čo sú jednotlivé funkcie určené, sa dá intuitívne zistiť z ich názvov. Niektoré funkcie určené na aktualizáciu údajov o sieťach

podľa údajov o susedoch sú využívané hlavne DUALom a sú bližšie popísané v kapitole 3.4.

```
extern struct list *eigrp_topology_new (void);
extern void eigrp_topology_init (struct list*);
extern struct eigrp_prefix_entry *eigrp_prefix_entry_new (void);
extern struct eigrp_neighbor_entry *eigrp_neighbor_entry_new (void);
extern void eigrp_topology_free (struct list *);
extern void eigrp_topology_cleanup (struct list *);
extern void eigrp_prefix_entry_add (struct list *, struct
                eigrp_prefix_entry *);
extern void eigrp_neighbor_entry_add (struct eigrp_prefix_entry *,
struct eigrp_neighbor_entry *);
extern void eigrp_prefix_entry_delete (struct list *, struct
                eigrp_prefix_entry *);
extern void eigrp_neighbor_entry_delete (struct eigrp_prefix_entry *,
struct eigrp_neighbor_entry *);
extern void eigrp_topology_delete_all (struct list *);
extern unsigned int eigrp_topology_table_isempty (struct list *);
extern struct eigrp_prefix_entry *eigrp_topology_table_lookup_ipv4
(struct list *, struct prefix_ipv4 *);
extern struct list *eigrp_topology_get_successor (struct
               eigrp_prefix_entry *);
extern struct eigrp_neighbor_entry *eigrp_prefix_entry_lookup (struct
               list *, struct eigrp_neighbor *);
extern void eigrp_topology_update_all_node_flags (struct eigrp *);
extern void eigrp_topology_update_node_flags (struct
                eigrp_prefix_entry *);
extern int eigrp_topology_update_distance ( struct
                eigrp_fsm_action_message *);
extern void eigrp_update_routing_table(struct eigrp_prefix_entry *);
extern void eigrp_topology_neighbor_down(struct eigrp *, struct
                eigrp_neighbor *);
extern void eigrp_update_topology_table_prefix(struct list *, struct
                eigrp_prefix_entry * );
```

Ukážka 3.17 Implementované funkcie topologickej tabuľky

3.3 Výmena informácií s pomocou Classic/Wide Metrics TLV

V tejto časti bolo cieľom zabezpečiť korektné čítanie a vysielanie klasických aj *wide metrics* TLV. O vývoj tejto časti implementácie prejavil veľký záujem aj Donnie Savage, inžinier zo spoločnosti Cisco, ktorý dlhodobo spolupracoval pri vývoji a implementácii protokolu EIGRP do operačného systému Cisco IOS. Realizoval sa najmä v implementovaní TLV 2.0, ktoré sa využívajú pri *wide* metrikách. Ohľadom tejto časti stále prebieha komunikácia a nie je úplne dokončené nasadenie v protokole.

Implementácia klasických TLV spočíva v definovaní spôsobu, akým sa údaje posielajú a čítajú z dátového toku. Poradie, akým sú údaje posielané, je zrejmé z obrázkov

1.3 až 1.7. Definované sú teda v prvom rade štruktúry, do ktorých sa ukladajú všetky polia TLV:

```
struct TLV_IPv4_Internal_type
{
  u_int16_t type;
  u_int16_t length;
  struct in_addr forward;

  /*Metrics*/
  struct eigrp_metrics metric;

  u_char prefix_length;

  unsigned char destination_part[4];
  struct in_addr destination;
}__attribute__((packed));

struct TLV_IPv4_External_type
{
  u_int16_t type;
  u_int16_t length;
  struct in_addr next_hop;
  struct in_addr originating_router;
  u_int32_t originating_as;
  u_int32_t administrative_tag;
  u_int32_t external_metric;
  u_int16_t reserved;
  u_char external_protocol;
  u_char external_flags;

  /*Metrics*/
  struct eigrp_metrics metric;

  u_char prefix_length;
  unsigned char destination_part[4];
  struct in_addr destination;
}__attribute__((packed));
```

Ukážka 3.18 TLV štruktúry

Tieto štruktúry sa potom postupným čítaním z dátového toku napĺňajú údajmi. Ako je možné si všimnúť, štruktúry obsahujú o jednu položku navyše oproti obrázkom 1.6 a 1.7, konkrétne *unsigned char destination_part[4]*. Je to preto, že posledné pole má premenlivú dĺžku, ktorú nie je možné popísať v statickej deklarácii štruktúry. Dĺžka adresy cieľovej siete, ktorá sa v tomto poli posiela, je totiž zaokrúhlená na najbližších 8 bitov nahor a posielaná v opačnom poradí bajtov. Prijímanie siete 192.168.0.0/16 vyzerá asi tak, že pri čítaní z toku po 8 bajtov sa najprv prijme číslo 16, čo predstavuje masku, potom číslo 168 a nakoniec 192. Treba preto ukladať tieto čísla samostatne a nakoniec ich zložiť v správnom poradí práve do poslednej štruktúry s názvom destination. Funkcia pre čítanie TLV interného typu vyzerá nasledovne (eigrp_packet.c):

```
struct TLV_IPv4_Internal_type *
eigrp_read_ipv4_tlv (struct stream *s)
{
  struct TLV_IPv4_Internal_type *tlv;

  tlv = eigrp_IPv4_InternalTLV_new ();

  tlv->type = stream_getw(s);             // read type
  tlv->length = stream_getw(s);           // read length
  tlv->forward.s_addr = stream_getl(s);   // read forwarding address
  tlv->metric.delay = stream_getl(s);     // read delay
  tlv->metric.bandwith = stream_getl(s); // read bandwidth
  tlv->metric.mtu[0] = stream_getc(s);    // read first byte of MTU
  tlv->metric.mtu[1] = stream_getc(s);    // read second byte of MTU
  tlv->metric.mtu[2] = stream_getc(s);    // read third byte of MTU
  tlv->metric.hop_count = stream_getc(s); // read hop count
  tlv->metric.reliability = stream_getc(s); // read reliability
  tlv->metric.load = stream_getc(s);      // read load
  tlv->metric.tag = stream_getc(s);       // read tag
  tlv->metric.flags = stream_getc(s);     // read flags

  tlv->prefix_length = stream_getc(s);    // read prefix length
  // choosing variant of reading acording to prefix length
  if (tlv->prefix_length <= 8)
    {
      tlv->destination_part[0] = stream_getc(s);
      tlv->destination.s_addr = (tlv->destination_part[0]);

    }
  .
  .
  .
  else if (tlv->prefix_length > 24 && tlv->prefix_length <= 32)
    {
      tlv->destination_part[0] = stream_getc(s);
      tlv->destination_part[1] = stream_getc(s);
      tlv->destination_part[2] = stream_getc(s);
      tlv->destination_part[3] = stream_getc(s);
      tlv->destination.s_addr = ((tlv->destination_part[3] << 24)
          + (tlv->destination_part[2] << 16)
          + (tlv->destination_part[1] << 8)
          + tlv->destination_part[0]);
    }
  return tlv;
}
```

Ukážka 3.19 Prijímanie TLV

Odosielanie TLV sa realizuje presne opačným postupom. Posielané dáta sa však pred odoslaním nedávajú do jednej štruktúry, ale čítajú sa priamo z príslušného miesta topologickej tabuľky. Ukážka funkcie vykonávajúcej odosielanie je skrátená z dôvodu dĺžky a podobných častí kódu:

```
u_int16_t
eigrp_add_internalTLV_to_stream (struct stream *s,
    struct eigrp_prefix_entry *pe)
{
  u_int16_t length;

  stream_putw(s, EIGRP_TLV_IPv4_INT);          // send type
  // send length acording to prefix length
  if (pe->destination_ipv4->prefixlen <= 8)
    {
      stream_putw(s, 0x001A);
      length = 0x001A;
    }
  if ((pe->destination_ipv4->prefixlen > 8)
      && (pe->destination_ipv4->prefixlen <= 16))
    .
    .
    .
  if (pe->destination_ipv4->prefixlen > 24)
    {
      stream_putw(s, 0x001D);
      length = 0x001D;
    }

  stream_putl(s, 0x00000000);  // use next hop addres from ip header

  /*Sending metric*/
  stream_putl(s, pe->reported_metric.delay);
  stream_putl(s, pe->reported_metric.bandwith);
  stream_putc(s, pe->reported_metric.mtu[2]);
  stream_putc(s, pe->reported_metric.mtu[1]);
  stream_putc(s, pe->reported_metric.mtu[0]);
  stream_putc(s, pe->reported_metric.hop_count);
  stream_putc(s, pe->reported_metric.reliability);
  stream_putc(s, pe->reported_metric.load);
  stream_putc(s, pe->reported_metric.tag);
  stream_putc(s, pe->reported_metric.flags);
      // send prefix length
  stream_putc(s, pe->destination_ipv4->prefixlen);
      // choose way of destinat address according to prefix length
  if (pe->destination_ipv4->prefixlen <= 8)
    {
      stream_putc(s, pe->destination_ipv4->prefix.s_addr & 0xFF);
    }
  .
  .
  .
  if (pe->destination_ipv4->prefixlen > 24)
    {
   stream_putc(s, pe->destination_ipv4->prefix.s_addr & 0xFF);
   stream_putc(s, (pe->destination_ipv4->prefix.s_addr >> 8) & 0xFF);
   stream_putc(s, (pe->destination_ipv4->prefix.s_addr >> 16) & 0xFF);
   stream_putc(s, (pe->destination_ipv4->prefix.s_addr >> 24) & 0xFF);
    }
  return length;
}
```

Ukážka 3.20 Odosielanie TLV

Prijímanie a odosielanie TLV externého formátu je analogické ku týmto funkciám a v konečnej verzii aj prijímanie a odosielanie *wide metrics* TLV.

3.4 DUAL

Je najobsiahlejšia a najkomplexnejšia časť práce. Jej cieľom bolo implementovať rozhodovaciu logiku a stavový automat DUALu s dôrazom na protokolovú nezávislosť.

Aby mohol proces rozhodovania pracovať čo najrýchlejšie, rozdelili sme celý proces na dve časti. Prvá časť predstavuje rozhodovaciu logiku a ako výstup podáva číslo udalosti, ktorá nastala. Druhá časť vykonáva samotnú obsluhu jednotlivých udalostí. Treťou časťou, ktorá nie je priamo súčasťou DUALu, je špeciálny typ správy používanej pre komunikáciu s DUALom. Celý proces nakoniec vyzerá tak, že nastanie zmeny sa prostredníctvom špeciálnej správy oznámi DUALu. DUAL s pomocou rozhodovacej logiky určí, aká udalosť pre danú sieť nastala, a posunie túto informáciu obsluhe, ktorá nad danou sieťou vykoná potrebné operácie. Pri takomto prístupe je rozhodovacia logika dostupná ostatným sieťam, kým na inej sa vykonáva obsluha, čím sa zvyšuje efektivita. Protokolová nezávislosť je zabezpečená rozhodovaním výlučne na základe vzdialeností a nie adries alebo typu sietí.

Implementácia teda vyžadovala vytvorenie špeciálnej správy, pomocou ktorej by sa DUALu oznamovali informácie nevyhnutné pre určenie typu nastanej udalosti, vytvorenie rozhodovacej logiky a nakoniec samotných funkcií obsluhy pre každú udalosť. Obslužné funkcie musia vo vysokej miere využívať funkcie pre prácu s topologickou tabuľkou, nakoľko v nej vykonávajú všetky zmeny.

Správa - pri komunikácii s DUALom treba vytvoriť špeciálnu správu, ktorá musí obsahovať informácie o:

- type paketu, ktorý spôsobil zmenu,
- sieti ktorej sa zmena týka,
- susedovi od ktorého správa prišla,
- a samotné TLV paketu.

Štruktúra správy je definovaná nasledovne:

```
struct eigrp_fsm_action_message
{
  u_char packet_type;             //UPDATE, QUERY, SIAQUERY, SIAREPLY
  struct eigrp *eigrp;                    // which thread sent mesg
  struct eigrp_neighbor *adv_router;     //advertising neighbor
  struct eigrp_neighbor_entry *entry;
  struct eigrp_prefix_entry *prefix;
  int data_type;                  // internal or external tlv type
  union{
    struct TLV_IPv4_External_type *ipv4_ext_data;
    struct TLV_IPv4_Internal_type *ipv4_int_type;
  }data;
};
```

Ukážka 3.21 eigrp_fsm_action_mesasage

Rozhodovacia logika – riadi sa podmienkami vyplývajúcimi zo stavového automatu popísaného v kapitole 1.5. Všetky udalosti sú očíslované. Tie udalosti, ktoré sa nachádzajú v diagrame z obrázka 1.14 viacnásobne, majú rovnaké číslo a nakoniec ich obsluhuje tá istá funkcia. Čísla jednotlivých funkcií je možné nájsť v hlavičke súboru eigrp_fsm.c.

Proces rozhodovania vždy začína aktualizovaním záznamu o susedovi v topologickej tabuľke údajmi z prijatého paketu. To sa vykonáva funkciou, ktorá zisťuje, či došlo k zmene vzdialenosti, a ak áno, či došlo ku zvýšeniu, zníženiu alebo len zmene parametrov bez zmeny výslednej metriky. Nakoniec presunie záznam o susedovi na správnu pozíciu v zozname. Samotná funkcia vyzerá nasledovne (eigrp_topology.c):

```
int
eigrp_topology_update_distance(struct eigrp_fsm_action_message *msg)
{
  struct eigrp *eigrp = msg->eigrp;
  struct eigrp_prefix_entry *prefix = msg->prefix;
  struct eigrp_neighbor_entry *entry = msg->entry;
  int change = 0;

  struct TLV_IPv4_External_type *ext_data = NULL;
  struct TLV_IPv4_Internal_type *int_data = NULL;
  if (msg->data_type == EIGRP_TLV_IPv4_INT)
    {
      int_data = msg->data.ipv4_int_type;
      if (eigrp_metrics_is_same(&int_data->metric,
            &entry->reported_metric))
        {
          return 0; // No change
        }
      change =
          entry->reported_distance
              < eigrp_calculate_metrics(eigrp,&int_data->metric) ? 1 :
          entry->reported_distance
              > eigrp_calculate_metrics(eigrp, &int_data->metric) ? 2
                  : 3; // Increase : Decrease : No change
      entry->reported_metric = int_data->metric;
      entry->reported_distance = eigrp_calculate_metrics(eigrp,
          &int_data->metric);       // Updating metric
      entry->distance = eigrp_calculate_total_metrics(eigrp, entry);
    }
  else
    {
      ext_data = msg->data.ipv4_ext_data;
    }
  /* Move to correct position in list according to new distance */
  listnode_delete(prefix->entries, entry);
  listnode_add_sort(prefix->entries, entry);
  return change;
}
```

Ukážka 3.22 eigrp_topology_update_distance

Následne sa rozhodovací proces delí na základe aktuálneho stavu siete. Celý rozhodovací proces je zhrnutý vo funkcii *eigrp_get_fsm_event*. Okrem vzdialeností je druhý najdôležitejší údaj o tom, ktorý zo susedov spĺňa FC a ktorý je vybraný za *successora*. Tento údaj sa nachádza v zázname o susedovi v poli *flags* (obrázok 3.2). Ukážka je z dôvodu rozsiahlosti obmedzená na rozhodovací proces v rámci jedného stavu.

```
int eigrp_get_fsm_event(struct eigrp_fsm_action_message *msg) {
    // Loading base information from message
    //struct eigrp *eigrp = msg->eigrp;
    struct eigrp_prefix_entry *prefix = msg->prefix;
    struct eigrp_neighbor_entry *entry = msg->entry;
    u_char actual_state = prefix->state;

    if (entry == NULL) {
        entry = eigrp_neighbor_entry_new();
        entry->adv_router = msg->adv_router;
        entry->ei = msg->adv_router->ei;
        entry->prefix = prefix;
        msg->entry = entry;
    }
    // Dividing by actual state of prefix's FSM
    switch (actual_state) {
    case EIGRP_FSM_STATE_ACTIVE_1: {
        int change = eigrp_topology_update_distance(msg);//update dist

        if (msg->packet_type == EIGRP_OPC_QUERY
                && (entry->flags &
            EIGRP_NEIGHBOR_ENTRY_SUCCESSOR_FLAG)) { //query from
                                                    // successor
            return EIGRP_FSM_EVENT_QACT;
        } else if (msg->packet_type == EIGRP_OPC_REPLY) {
            listnode_delete(prefix->rij, entry->adv_router);

            if (change == 1
                    && (entry->flags &
                  EIGRP_NEIGHBOR_ENTRY_SUCCESSOR_FLAG)) {
                return EIGRP_FSM_EVENT_DINC;  //increase in D
            } else if (prefix->rij->count) {
                return EIGRP_FSM_KEEP_STATE;
            } else {                          //last reply received
                zlog_info("All reply received\n");
                return EIGRP_FSM_EVENT_LR;
            }
        } else if (msg->packet_type == EIGRP_OPC_UPDATE && change == 1
                && (entry->flags &
                  EIGRP_NEIGHBOR_ENTRY_SUCCESSOR_FLAG)) {
            return EIGRP_FSM_EVENT_DINC; //increase in D
        }
        return EIGRP_FSM_KEEP_STATE;

        break;
    }
    }
    return EIGRP_FSM_KEEP_STATE;
}
```

Ukážka 3.23 Rozhodovací proces

Návratové hodnoty pre jednotlivé typy udalostí korešpondujú s ich číselným označením.

```
#define EIGRP_FSM_EVENT_NQ_FCN          0 /*input event other than
                                        query from succ, FC not satisfied*/
#define EIGRP_FSM_EVENT_LR              1 /*last reply, FD is reset*/
#define EIGRP_FSM_EVENT_Q_FCN           2 /*query from succ, FC not
                                          satisfied*/
#define EIGRP_FSM_EVENT_LR_FCS          3 /*last reply, FC satisfied
                                          with current value of FDij*/
#define EIGRP_FSM_EVENT_DINC            4 /*distance increase while in
                                          active state*/
#define EIGRP_FSM_EVENT_QACT            5 /*query from succ while in
                                          active state*/
#define EIGRP_FSM_EVENT_LR_FCN          6 /*last reply, FC not
                                    satisfied with current value of FDij*/
#define EIGRP_FSM_KEEP_STATE            7 /*state not changed, usually
                                        by receiving not last reply */
```

Ukážka 3.24 FSM udalosti

Obslužné funkcie – Pre každú udalosť je potrebné vykonať iné operácie, a preto musí mať každá svoju vlastnú funkciu. Popis operácií je podrobne opísaný v kapitole 1.5. Doplnená funkcionalita oproti popisu je obsluha smerovacej tabuľky kernelu operačného systému. Pomocou príznakov z poľa *flags* v zázname o susedovi vieme povedať, ktorý sused je aktuálne v smerovacej tabuľke kernelu operačného systému a v prípade straty statusu *successora* ho z nej prípadne odstrániť. Takisto v opačnom prípade, ak je sused *successorom* a nie je v smerovacej tabuľke, tak sa tam doplní. Pre vkladanie a odstraňovanie položiek zo smerovacej tabuľky kernelu operačného systému slúžia funkcie komunikujúce so zebra daemonom *eigrp_zebra_route_add* a *eigrp_zebra_route_delete*. Ukážka zobrazuje obsluhu udalosti číslo 7 (*keep_state*):

```
int eigrp_fsm_event_keep_state(struct eigrp_fsm_action_message *msg) {

    struct eigrp_prefix_entry *prefix = msg->prefix;
    struct eigrp_neighbor_entry *entry = msg->entry;

    if (prefix->state == EIGRP_FSM_STATE_PASSIVE) {
        if (!eigrp_metrics_is_same(&prefix->reported_metric,
                &((struct eigrp_neighbor_entry *) prefix->entries
                        ->head->data)->total_metric)) {
      // if there was distance change update RD, FD and D
            prefix->rdistance =
                    prefix->fdistance =
                            prefix->distance =
                                    ((struct eigrp_neighbor_entry *)
                             prefix->entries->head->data)->distance;
      //update metrics
            prefix->reported_metric =
                    ((struct eigrp_neighbor_entry *) prefix->entries
                                        ->head->data)->total_metric;
            if (msg->packet_type == EIGRP_OPC_QUERY)
                eigrp_send_reply(msg->adv_router, msg->entry);
            prefix->req_action |= EIGRP_FSM_NEED_UPDATE;
            listnode_add((eigrp_lookup())
                  ->topology_changes_internalIPV4,prefix);
        }
        eigrp_topology_update_node_flags(prefix);
        eigrp_update_routing_table(prefix);
    }

    if (msg->packet_type == EIGRP_OPC_QUERY)
        eigrp_send_reply(msg->adv_router, prefix);

    return 1;
}
```

Ukážka 3.25 Obsluha udalosti

Ktorú funkciu obsluhy použiť určíme spojením aktuálneho stavu siete a čísla udalosti, ktoré sme dostali od rozhodovacej logiky. Pre každú kombináciu je v nami definovanom poli s názvom NSM určená funkcia obsluhy, ktorá sa má použiť. Od pôvodnej myšlienky spúšťať každú funkciu obsluhu v samostatnom vlákne muselo byť upustené, z dôvodu implementácie funkcionality sériových čísiel a neprítomnosti funkcie čakania v Quagga vláknach. Nebolo teda možné zabezpečiť, že v čase spustenia procedúry pre odoslanie hromadnej správy sú už nastavené všetky potrebné príznaky. Pole NSM má nasledovnú štruktúru (eigrp_fsm.c).

```
struct { int (*func)(struct eigrp_fsm_action_message *);
} NSM[EIGRP_FSM_STATE_MAX][EIGRP_FSM_EVENT_MAX] = { {
//PASSIVE STATE
        { eigrp_fsm_event_nq_fcn }, /* Event 0 */
        { eigrp_fsm_event_keep_state }, /* Event 1 */
        { eigrp_fsm_event_q_fcn }, /* Event 2 */
        { eigrp_fsm_event_keep_state }, /* Event 3 */
        { eigrp_fsm_event_keep_state }, /* Event 4 */
        { eigrp_fsm_event_keep_state }, /* Event 5 */
        { eigrp_fsm_event_keep_state }, /* Event 6 */
        { eigrp_fsm_event_keep_state }, /* Event 7 */
}, {
//Active 0 state
        { eigrp_fsm_event_keep_state }, /* Event 0 */
        { eigrp_fsm_event_keep_state }, /* Event 1 */
        { eigrp_fsm_event_keep_state }, /* Event 2 */
        { eigrp_fsm_event_lr_fcs }, /* Event 3 */
        { eigrp_fsm_event_keep_state }, /* Event 4 */
        { eigrp_fsm_event_qact }, /* Event 5 */
        { eigrp_fsm_event_lr_fcn }, /* Event 6 */
        { eigrp_fsm_event_keep_state }, /* Event 7 */

},
.
.
.
};
```

Ukážka 3.26 NSM

Nakoniec sme definovali funkciu, ktorú je možné volať z akéhokoľvek miesta protokolu a ktorá ako parametre prijíma práve správu o udalosti a aktuálny stav a na ich základe vyberá a spúšťa správnu obslužnú funkciu z NSM poľa (eigrp_fsm.c).

```
int eigrp_fsm_event(struct eigrp_fsm_action_message *msg, int event) {

    zlog_info("EIGRP AS: %d State: %d  Event: %d Network: %s\n",
            msg->eigrp->AS, msg->prefix->state, event,
            eigrp_topology_ip_string(msg->prefix));
    (*(NSM[msg->prefix->state][event].func))(msg);

    return 1;
}
```

Ukážka 3.27 eigrp_fsm_event

ZÁVER

Spolu so zvyšnými dvoma časťami projektu sa nám (spolu s Bc. Jánom Janovicom a Bc. Petrom Orságom) úspešne podarilo vytvoriť funkčnú implementáciu protokolu EIGRP v balíku Quagga. Celý projekt sa aktívne vyvíja a neustále doň pribúdajú nové funkcionality. Vzhľadom na mieru pozornosti, akej sa projektu dostalo v širokom okruhu ľudí, môžeme v dohľadnom čase očakávať prvú verziu implementácie v oficiálnych distribúciách Quaggy.

Zadanie bolo vypracované do funkčného stavu a postupne pribúdajú ďalšie funkcie. Komunikácia s Cisco zariadeniami prebieha spoľahlivo, takisto aj výmena smerovacích informácií a reagovanie na zmenu vzdialeností. Implementácia rozhodovacej logiky a ostatných modulov sa ukázala počas tvorby ako efektívna a rýchlo editovateľná. V prípade vzniknutia chýb boli tieto rýchlo identifikovateľné a ľahko opraviteľné. Nestihli sme možno spracovať všetky nápady a myšlienky, ktoré sme mali. Môžeme však povedať, že na veľa vecí do budúcna bolo myslené a značne sa tak zredukovalo množsto času, ktoré bude projektu potrebné venovať.

V rámci projektu sme mali možnosť sa stretnúť s novými typmi problémov a nadobudli sme cenné skúsenosti z oblasti troubleshootingu. V rámci balíka Quagga sme mali možnosť sa stretnúť asi so všetkými možnými nástrojmi, aké jazyk C poskytuje. Mali sme možnosť si precvičiť tímovú prácu, ale aj samostatné jednanie. Najcennejšia však bola možnosť podieľať sa na vývoji niečoho, čo, ako všetci dúfame, bude používané na tisícoch zariadení po celom svete.

POUŽITÁ LITERATÚRA

[1] SAVAGE, D. – SLICE, D. – NG, J. – MOORE, S. – WHITE, R. *Enhanced Interior Gateway Routing Protocol v.02* [online]. 10. 4. 2014. Dostupné na internete: <https://tools.ietf.org/html/draft-savage-eigrp-02>

[2] *Introduction to EIGRP* [online]. 10. 5. 2005. Dostupné na internete: <http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/13669-1.html>

[3] PALÚCH, P. – KOCHARIANS, N. 2014. *CCIE Routing and Switching v5.0 Official Cert Guide, Volume 1, Fifth Edition* Indianapolis: Cisco Press, 2014. 364 - 366 s. ISBN 1-58714-396-8

[4] GARCIA-LUNES-ACEVES, J.J. *Loop-Free Routing Using Diffusing Computations* [online]. IEEE/ACM Transactions on networking, vol. 1, no. 1. Február 1993. Dostupné na internete: <http://www.utdallas.edu/~kxs028100/Papers/loop-free-routing-using-diffusing-computations.pdf>

[5] *Quagga routing suite*. Dostupné na internete: http://www.nongnu.org/quagga/