VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2023 David Hamran

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

TVORBA IPV6 PAKETŮ POMOCÍ PROGRAMU SCAPY PRO POUŽITÍ V GENERÁTORU PROVOZU TREX

CREATING IPV6 PACKETS WITH SCAPY FOR USE IN THE TREX TRAFFIC GENERATOR

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE David Hamran

AUTHOR

VEDOUCÍ PRÁCE Ing. Aneta Koláčková

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: David Hamran ***ID:*** 230253

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Tvorba IPv6 paketů pomocí programu Scapy pro použití v generátoru provozu Trex

POKYNY PRO VYPRACOVÁNÍ:

Bakalářská práce se bude zabývat programem Scapy, který je určen k manipulaci síťového provozu. Úkolem bude zaměřit se na tvorbu paketů využívající Segment routing pro IPv6 (SRv6) a možnosti vkládat do nich dodatečné informace, zejména pro použití u dalších technologií. Dále pak bude nutné vytvořené pakety implementovat a otestovat pomocí generátoru reálného provozu Trex v předem určené topologii. V práci bude požadována důkladná analýza programu Scapy a popis způsobu vytváření paketů v tomto programu. Teoretická část práce bude obsahovat detailní popis paketů pro IPv6 a SRv6 a jejich možností. Výstupy práce budou přehledně prezentovány a diskutovány.

DOPORUČENÁ LITERATURA:

[1] Scapy - Packet crafting for Python2 and Python3: Sep. 11, 2022. [Online]. Dostupné: https://scapy.net/

[2] Deering, Steve, and Robert Hinden. Internet protocol, version 6 (IPv6) specification. No. rfc8200. 2017. Dostupné: https://www.rfc-editor.org/rfc/rfc8200

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Aneta Koláčková

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Bakalárska práca sa venuje tvorbe paketov v programe Scapy pre použitie v sieťach schopných pracovať s SRv6. Práca obsahuje päť hlavných častí popisujúcich *Internet protokol verzie 6* (IPv6), *Segmentové smerovanie skrz IPv6* (SRv6), program Scapy, skript pre prácu s paketmi a program pre generovanie sieťovej premávky Trex.

KĽÚČOVÉ SLOVÁ

IPv6 IP adresa Segmentové smerovanie Segment segmentové IPv6 Scapy Paket Záhlavie/Hlavička Skript Python Splice Identifikátor

ABSTRACT

The bachelor work is devoted to the creation of packets in the Scapy for use in networks capable of Segment Routing over IPv6. The work contains five main parts describing Internet Protocol version 6, Segment Routing over IPv6, Scapy, script for easier work with packets and the program for generating network traffic Trex.

KEYWORDS

Internet protocol version 6 IP address Segment routing Segment routing over IPv6 Scapy Packet Header Script Python Splice Identifier

Vysadené pomocou balíčku `thesis` verzie 4.07; `http://latex.feec.vutbr.cz`

HAMRAN, David. *Tvorba IPv6 paketů pomocí programu Scapy pro použití v generátoru provozu Trex*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 94 s. Bakalárska práca. Vedúci práce: Ing. Aneta Koláčková,

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	David Hamran
VUT ID autora:	230253
Typ práce:	Bakalárska práca
Akademický rok:	2022/2023
Téma závěrečnej práce:	Tvorba IPv6 paketů pomocí programu Scapy pro použití v generátoru provozu Trex

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som poďakoval vedúcej bakalárskej práce pani Ing. Aneta Koláčková za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Obsah

Úvod **23**

1 Internet protokol verzie 6 (IPv6) **25**

1.1 Vznik IPv6 . . . 25

1.2 Charakteristika IPv6 . . . 25

1.3 Rozbor IPv6 paketu . . . 26

1.3.1 Fixné záhlavie . . . 26

1.3.2 IPv6 adresa . . . 28

1.4 Rozširujúce záhlavie (EH) . . . 29

1.4.1 Záhlavie medziľahlých možností (Hop-by-Hop Options) . . . 30

1.4.2 Smerovacie Záhlavie (RH) . . . 31

1.4.3 Fragmentovacie záhlavie (Fragment header) . . . 32

1.4.4 Hlavička možností destinácie (DOH) . . . 34

1.4.5 Autentizačné záhlavie (AH) . . . 34

1.4.6 Encapsulating security Payload (ESP) . . . 34

1.4.7 Absencia ďalšieho záhlavia . . . 34

1.5 Smerovanie . . . 35

1.6 Vkladanie vlastných informácii do IPv6 paketu . . . 35

2 Segmentové smerovanie skrz IPv6 (SRv6) **37**

2.1 Segmentové smerovanie (SR) . . . 37

2.2 Segmentová smerovacia hlavička (SRH) . . . 38

2.2.1 SRH TLV . . . 39

2.2.2 Vyplňovacie TLV . . . 40

2.3 SR uzly . . . 41

2.3.1 SR zdrojový uzol . . . 41

2.3.2 Tranzitný uzol . . . 42

2.3.3 Uzol koncového bodu segmentu SR . . . 42

2.4 Proces posielania paketov v sieti . . . 42

2.4.1 SR zdrojový uzol . . . 42

2.4.2 Tranzitný uzol bez podpory SR . . . 43

2.4.3 Uzol koncového bodu SR . . . 43

2.4.4 SRv6 Sieťové programovanie . . . 45

2.4.5 SRv6 Segment . . . 45

3 Program Scapy **47**
3.1 Základná charakterisitka programu Scapy 47
3.1.1 Rýchly dizajn paketov 47
3.1.2 Vizualizácia dát 48
3.1.3 Dekódovanie dát 48
3.2 Tvorba IPv6 paketov pomocou programu Scapy 50
3.2.1 Vrstvenie 50
3.2.2 Triedy v Scapy 50
3.2.3 Práca s paketmi 52
3.2.4 Zoskupenia paketov v Scapy 53
3.2.5 Import a export paketov 54
3.2.6 Modifikácia paketov 56

4 Skript pre manipuláciu a analýzu paketov **57**
4.1 Tvorba vlastných nástrojov pomocou knižnice Scapy 57
4.2 Požiadavky na spustenie skriptu 58
4.2.1 Python 58
4.2.2 PiP 58
4.3 Metódy 59
4.3.1 Metóda `install` 59
4.3.2 Metóda `check_packages` 59
4.3.3 Metóda `export_dict_to_file` 60
4.3.4 Metóda `import_dict_from_file` 61
4.3.5 Metóda `packet_creator(templates)` 61
4.3.6 Metóda `iter` 62
4.3.7 Metóda `create_packet` 63
4.3.8 Metóda `browse_templates` 63
4.3.9 Metóda `load_packet` 64
4.3.10 Metóda `save_packet` 65
4.3.11 Metóda `print_packet` 65
4.3.12 Metóda `export_packet` 66
4.3.13 Metóda `join_pcap_files` 66
4.3.14 Metóda `modify_packet` 67
4.3.15 Metóda `add_id` 68
4.3.16 Metóda `generate_random_ipv6_pcap` 69
4.3.17 Metóda `analyze` 69
4.3.18 Metóda `launch_scapy_console()` 70
4.4 Packet Playground 71
4.4.1 Funkcie 71

4.4.2 Tvorba paketov pomocou funkcie `Packet creator` 72

5 Program Trex **79**

5.1 Generátor sieťovej prevádzky 79

5.2 Základná charakteristika programu Trex 79

5.2.1 Vytvorenie konfiguračného súboru 81

5.3 Topológia pre implementáciu paketov 81

Závěr **85**

Literatúra **87**

Zoznam symbolov a skratiek **89**

Zoznam príloh **91**

A Program pre tvorbu paketov **93**

A.1 Trieda pkt časť 1. 93

A.2 Trieda pkt časť 2. 94

Zoznam obrázkov

1.1 Všeobecná forma IPv6 paketu 26
1.2 IPv6 záhlavie 27
1.3 IPv6 adresa 29
1.4 Typy IPv6 adries 30
1.5 Smerovacie záhlavie 31
1.6 Fragmentovacie záhlavie 33
1.7 Hlavička možností destinácie 34
1.8 Vlastné rozširujúce záhlavie 36
2.1 Segmentová smerovacia hlavička 38
2.2 TLV 39
2.3 Pad1 40
2.4 PadN 41
2.5 SRv6 v sieti 44
2.6 SRv6 Segment Identifikátor (SID) 45
3.1 Úvodná obrazovka programu Scapy 47
3.2 Vizualizácia vrstvenia 50
3.3 Vytvorenie základného IPv6 paketu s (SRH) pomocou argumentov. . 52
4.1 Konzolové rozhranie skriptu Packet Playground 57
4.2 Ukážka použitia metódy `packet_creator()` 61
4.3 Príklad použitia metódy `iter` pre voľbu odosielateľa paketu. 62
4.4 Príklad použitia metódy `iter` pre voľbu paketu z databázy. 64
4.5 Príklad použitia metódy `print` na výpis načítaného paketu. 66
4.6 Príkklad použitia metódy analyze 70
4.7 Príkklad použitia metódy launch scapy console. 70
4.8 Analýza paketov. 72
4.9 Výber odosielateľa 73
4.10 Výber identifikátora paketu. 73
4.11 SR politiky pre IPv6 dátové prevádzky na N1 hraničnom uzle[16]. . . 74
4.12 Výpis parametrov paketu. 74
4.13 Menu pre modifikáciu paketu. 75
4.14 Ukážka výpisu predlohy. 75
4.15 Ukážka načítania paketu z uložených predlôh. 76
4.16 Ukážka výpisu načítaného paketu. 76
5.1 Zapojenie TRex generátoru [16] 80
5.2 Topológia pre implementáciu paketov vybraná z diplomovej práce ktorú spracoval Igor Mjasojedov [16]. 82

5.3 Topológia pre implementáciu paketov vybraná z diplomovej práce ktorú spracoval Igor Mjasojedov [16]. 83

Zoznam tabuliek

3.1 Predvolene hodnoty pre IP 49
5.1 Zoznam sietí medzi kontajnermi s ich adresovaním [16]. 82

Zoznam výpisov

3.1 Príklad definovania paketu(DA=192.168.1.1, TTL=10), Protokol IP . 48
3.2 Spustenie programu Scapy 50
3.3 Polia v triede IPv6 a ich predvolené hodnoty. 51
3.4 Polia v triede IPv6ExtHdrSegmentRouting a ich predvolené hodnoty. 51
3.5 Polia v triede IPv6ExtHdrSegmentRoutingTLVPadN a ich predvolené hodnoty. 52
3.6 Vytvorenie jednoduchého paketu so smerovacím záhlavím (SRH). . . 53
3.7 Vytvorenie množiny dvoch paketov so smerovacím záhlavím (SRH). . 53
3.8 Vytvorenie zoznamu paketov (packet list) s SRH. 54
3.9 Použitie príkazov wrpcap a rdpcap 55
3.10 Príklad vytvorenia novej vrstvy s názvom „Disney“ [5]. 55
3.11 Príklad použitia vlastnej vrstvy vytvorenej vo výpise 3.10 [5]. 56
3.12 Ukážka nastavenia tlv_object v smerovacom záhlaví 56
4.1 Ukážka importácie Scapy. 57
4.2 Inštalácia pip pomocou ensurepip. 58
4.3 Inštalácia pip pomocou get-pip.py. 59
5.1 TRex profil sieťovej premávky v bezstavovom móde. 81

Úvod

Táto práca sa venuje tvorbe paketov pomocou programu Scapy pre použitie v sieti podporujúcej *Segmentové smerovanie skrz IPv6* (SRv6). SRv6 spája *Internet protokol verzie 6* (IPv6) a *Segmentové smerovanie* (SR). SRv6 využíva technológiu IPv6 pre smerovanie a implementuje sieťové programovanie prostredníctvom IPv6 rozširujúcich záhlaví. SRv6 redukuje počet potrebných typov protokolov a umožňuje rozšíriteľnosť a programovateľnosť siete. Ponúka taktiež vysokú spoľahlivosť a rozširujúci potenciál aplikácii cloudových služieb.

Cieľom tejto práce je oboznámiť čitateľa s problematikou tvorby paketov v programe Scapy, pridanie vlastných záhlaví a informácii do vytvorených paketov a následná implementácia v sieti, ktorá podporuje SRv6. Implementáciu uskutočňujeme pomocou programu Trex pre generovanie premávky v sieti.

Tento dokument je rozdelený do častí, ktoré popisujú jednotlivé štruktúry potrebné pre pochopenie tejto práce a jej výstupu. Hlavné časti sú popis a analýza IPv6, SRv6, programu Scapy, skriptu pre prácu s paketmi a programu pre generovanie sieťovej premávky Trex.

Výstupom práce je skript pre tvorbu, analýzu a manipuláciu paketov popísaný v kapitole 4.

1 Internet protokol verzie 6 (IPv6)

V tejto sekcii sa budeme zaoberať analýzou protokolu IPv6, jeho vlastnosťami, využitím, výhodami a zakomponovanie SRH do IPv6 paketov ako rozširujúci header.

1.1 Vznik IPv6

Hlavným dôvodom, pre vznik nového protokolu bol predpoklad že IPv4 adresy budú v budúcnosti vyčerpané aj napriek ich počtu, ktorý je 2^{32} možných adries. Táto myšlienka začala byť jasná v 90. rokoch kedy aj napriek prepracovaniu adresovacieho systému bolo jasné že vyčerpanie IPv4 adries je nevyhnuteľné a začalo sa tak pracovať na náhrade. Ďalšími problémami, ktoré postupne narastali, sú rozsah internetových smerovacích tabuliek a neexistencia skutočného end-to-end modelu komunikácie (kvôli technike NAT a jejmu veľkému rozšíreniu).

Aktivity spojené s riešením týchto problémov vyústili do IPv6, ktorý v roku 1995 predstavilo *Internet Engineering Task Force* (IETF). IPv6 nie je len novým protokolom sieťovej vrstvy, ale celou sadou protokolov, podobne ako pôvodný TCP/IP, s tým rozdielom, že IPv6 rieši hlavne úlohy sieťovej vrstvy. IPv6 kombinuje zvýšené množstvo adries s efektívnejším záhlavím protokolu. Počet možných IPv6 adries je približne 2^{128} možných adries.

V súčastnosti sa stále vo veľkej miere používa hlavne IPv4. IPv4 zatiaľ nehrozí úplná náhrada za IPv6. Oba protokoly koexistujú a môžeme len predpokladať, že v budúcnosti IPv6 úplne nahradí IPv4 [15].

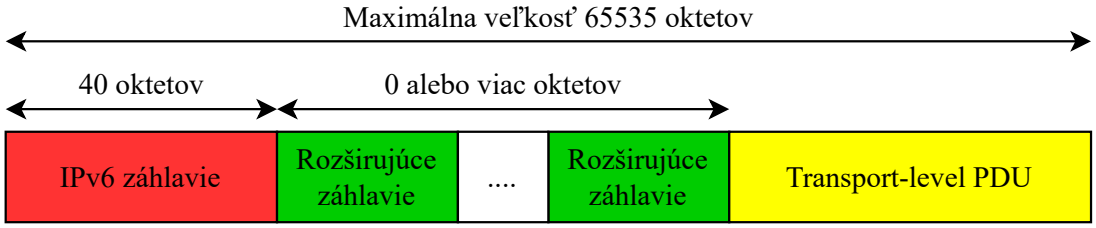
1.2 Charakteristika IPv6

Okrem rozšíreného adresného priestoru došlo aj k diskusii nad ďalšími vlastnosťami IPv6 ako napríklad:

- Zjednodušenie formátu záhlavia - menej povinných polí.
- Snaha o zredukovanie veľkosti smerovacích tabuliek globálnej úrovne v smerovačoch.
- Nové podporné protokoly, hlavne ICMPv6.
- Jednotné adresné schéma pre celý Internet aj vnútorné siete.
- Tri druhy adries - individuálne, skupinové a výberové(unicast, multicast, anycast).
- Rozšírenie adresného priestoru z 32 bitov na 128 bitov, teda z 2^{32} na 2^{128} adries.

1.3 Rozbor IPv6 paketu

IPv6 paket je najmenšia možná entita prenesená pomocou IPv6. IPv6 paket v jeho základnej podobe je zobrazený na obrázku 1.1. Jediné potrebné záhlavie sa nazýva IPv6 záhlavie s dĺžkou 40 oktetov. Potom následujú nepovinné rozširujúce záhlavia. Štruktúra základného záhlavia IPv6 nie je rovnaká ako tá u IPv4. Detail jednotlivých polí je naznačený na 1.2. Každé z rozširujúcich záhlaví obsahuje pole „next header“, ktoré slúži na identifikáciu prípadného následujúceho rozširujúceho záhlavia. V prípade ak už nenasleduje žiadne rozširujúce záhlavie, tak toto pole obsahuje identifikátor protokolu vyššej vrstvy, ktorý používa IPv6 (typicky to je protokol transportnej vrstvy, napr. TCP/UDP)[17] [15].



Obr. 1.1: Všeobecná forma IPv6 paketu [17]

1.3.1 Fixné záhlavie

Fixná alebo nemenná časť IPv6 paketu sa nazýva „Fixed header“.

Pre zvýšenie výkonnosti a za predpokladu že technológie Spojovej vrstvy ISO/OSI modelu poskytujú dostatočnú detekciu chýb, tak IPv6 záhlavie neobsahuje kontrolný súčet pre ochranu [15].

Popis polí IPv6 hlavičke

Verzia (version) – 4 bity, rovnako ako u IPv4 obsahuje verziu a zaisťuje aby ostatné systémy, ktoré spracovávajú datagram počas prenosu, mohli rôzne pole datagramu správne použiť. Pole obsahuje IPv6 respetíve hodnotu 6 [18] [15].

Trieda prevádzky (traffic class) – 8 bitov, toto pole umožňuje nastaviť prioritu paketu - prepravnú triedu, ktorá môže byť použitá pre DiffServ. Využitie tohoto poľa však ešte nie je presne definované, využitie je preto minimálne. Pole je podobné polu ToS v IPv4 [18] [15].

Identifikácia toku dát (flow label) – 20 bitov, označenie toku dát, umožňuje zjednodušenie smerovania, experimentálna záležitosť [15].

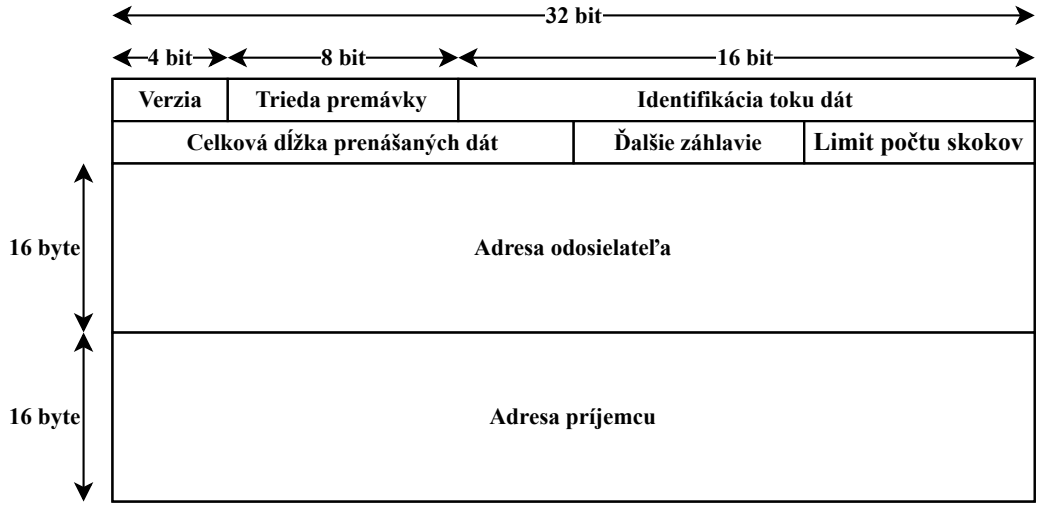
Celková dĺžka prenášaných dát (payload length) – 16 bitov, dĺžka prenášaných dát bez veľkosti základného záhlavia. Informuje o počte bajtov. Maximálna dĺžka môže teda byť teoreticky až 65kB [18] [15].

Ďalšie záhlavie (next header) – 8 bitov, informácia o vnorenom záhlaví, typicky informácia o protokole vyššej vrstvy (často TCP alebo UDP) alebo o rozširujúcom záhlaví [15].

Limit počtu skokov (hop limit) – 8 bitov, odpovdeá položke TTL u IPv4. Špecifikuje maximálny počet smerovačov, ktorými môže paket prejsť predtým než začne byť považovaný za neplatný. Hodnota poľa sa s každým prechodom smerovačom znižuje o jeden [18] [15].

IPv6 adresa odosielateľa (source address) – 128 bitov, pole obsahuje adresu odosielateľa z ktorého bol paket odoslaný. Toto pole sa počas svojej cesty od odosielateľa k príjemcovi nemení [18] [15].

IPv6 adresa príjemcu (destination address) – 128 bitov ,pole obsahuje adresu príjemcu. Táto adresa môže byť unicast či multicast. Toto pole sa počas svojej cesty od zdroja k cieľu nemení, až na výnimku. V prípade Smerovacieho Záhlavia Typu 0, obsahuje pole „Adresa príjemcu“ adresu ďalšieho skoku, ktorá nemusí byť konečná [18] [15].



Obr. 1.2: IPv6 záhlavie [18].

Z popisu je zrejmé, že celková dĺžka fixného záhlavia je 40 B. Táto dĺžka je pevne daná, narozdiel od protokolu IPv4, kde bola dĺžka premenná. Základná časť záhlavia IPv4 má dĺžku 20B, čo je polovica veľkosti IPv6 záhlavia. Vzhľadom k štyri-krát dlhším adresám u IPv6 to však neznamená veľký nárast [15].

Malý prírastok veľkosti záhlavia je daný vyradením nadbytočných položiek z fixného záhlavia. Konkrétne sa jedná o pole *rozširujúcich volieb, dĺžku záhlavia, kontrolný súčet a fragmentácia.* Niektoré operácie je možné uskutočniť pomocou rozširujúcich záhlaví [15].

Fragmentácia je v súčasnej dobe pomerne málo častý jav, ktorý naviac komplikuje fungovanie smerovania. Fragmentácia sa bežne nepredpokladá a bola odsunutá (pre špeciálne prípady) do špeciálneho rozširujúceho záhlavia [15].

Kontrolný súčet na IP vrstve verzie 6 sa už nerobí vôbec. Výpočet a jeho kontrola v každom uzle zbytočne spomaľovali smerovací proces. Za dostatočnú je považovaná kontrola, ktorá je štandardne vykonávaná na spojovej vrstve. Pokiaľ by táto kontrola nestačila, je nutné implementovať ešte ďalšiu a na vyššej vrstve ako je sieťová, čo je napríklad u transportných protokolov bežné [15].

Druhy adresovania IPv6

V IPv6 sú definované tri druhy adresovania, ktoré majú odličné chovanie:

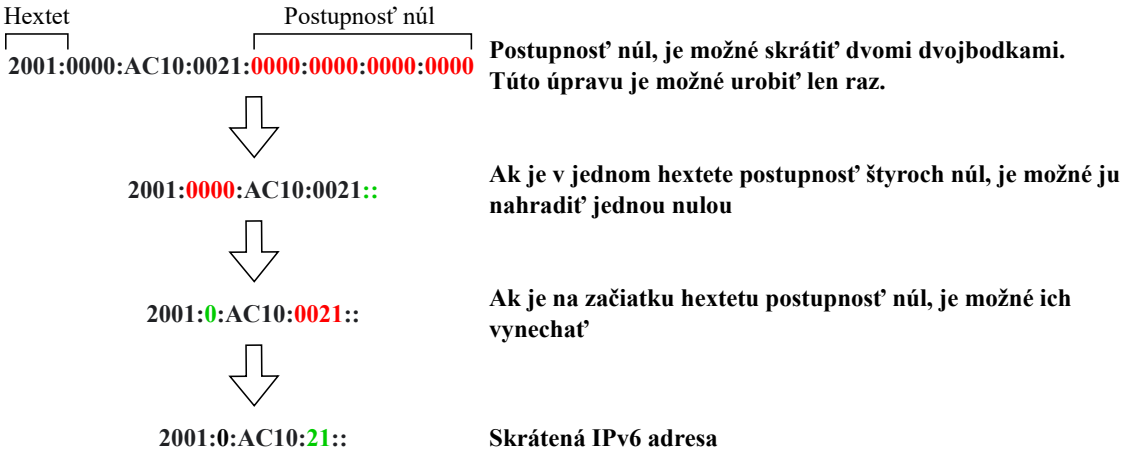
- **individuálne (unicast)** – adresy identifikujúce jednotlivé sieťové rozhrania, tak aby im mohli byť odoslané pakety [15].
- **skupinové (multicast)** – sú určené pre adresovanie skupín. Platí, že pakety odoslané na túto adresu by mali byť doručené všetkým členom skupiny. Tieto adresy zastupujú aj **všesmerové** (broadcast) adresy, ktoré nie sú v rámci IPv6 samostatne. V rámci adresného priestoru sú definované aj niektoré špeciálne skupiny [15].
- **výberové (anycast)** – takisto označujú skupinu adresátov, rozdiel je však v tom, že pakety sa posielajú len jednému z členov, z pravidla tomu, ktorý je „najbližšie“. Tento typ existuje aj v IPv4 [15].

1.3.2 IPv6 adresa

Adresa IPv6 je 128-bitová adresa, ktorá sa skladá z ôsmych hextetov, skupín 16 bitov. Jednotlivé hextety sa oddeľujú dvojbodkou. Zápis IPv6 adresy je možné skrátiť podľa určitých pravidiel [15].

Typy adries

V rámci adresného rozsahu IPv6 sú definované špeciálne typy a podtypy adries. Pre ilustráciu je na obrázku 1.4 uvedené grafické znázornenie jednotlivých typov a ich začlenenie do základných kategórii [15].



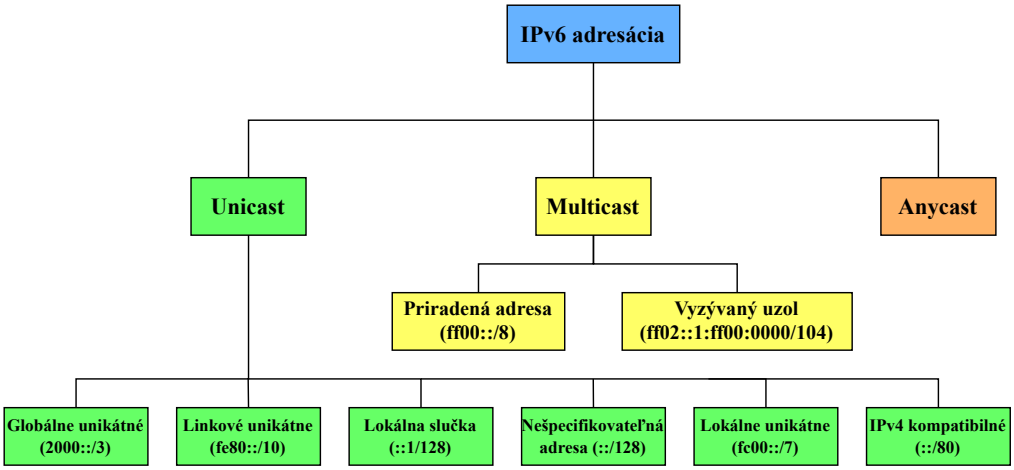
Obr. 1.3: IPv6 adresa

1.4 Rozširujúce záhlavie (EH)

V IPv6 sú voliteľné informácie internetovej vrstvy zakódované v samostatných záhlaviach, ktoré môžu byť umiestnené medzi hlavičkou IPv6 a hlavičkou hornej vrstvy v pakete. Existuje malý počet takýchto hlavičiek rozšírenia, pričom každá z nich je identifikovaná odlišnou hodnotou v poli „Ďalšie záhlavie“ (Next Header). Rozširujúce záhlavia sú očíslované podľa čísel protokolu IP IANA, čo sú rovnaké hodnoty, aké sa používajú pre protokoly IPv4 a IPv6. Pri spracovaní sekvencie hodnôt „Next Header“ v pakete prvá, ktorá nie je hlavičkou rozšírenia, označuje, že ďalšia položka v pakete je záhlavím hornej vrstvy. Ak neexistuje hlavička hornej vrstvy, použije sa špeciálna hodnota „No Next Header“. IPv6 záhlavie môže obsahovať ľubovolný počet rozširujúcich záhlaví. Úplná implementácia IPv6 zahrnuje implementáciu nasledujúcich záhlaví [6]:

- Záhlavie medziľahlých možností (Hop-by-Hop Options Header)
- Fragmentovacie záhlavie (Fragment header)
- Hlavička možností destinácie (DOH)
- Smerovacie Záhlavie (RH)
- Autentizačné záhlavie (AH)
- Encapsulating security Payload (ESP)

Rozširujúce záhlavia (okrem hlavičky Hop-by-Hop Options) nie sú spracované, vložené ani vymazané žiadnym uzlom počas cesty paketu, kým paket nedosiahne uzol (alebo každý z množiny uzlov, v prípade multicastu) identifikovaný v poli „Adresa príjemcu“ v IPv6 záhlaví [18] [6].



Obr. 1.4: Grafické znázornenie základných typov IPv6 adries a ich príslušnosti do troch existujúcich kategórii [15]

Poradie EH

Ak sa v tom istom pakete používa viac ako jedna hlavička rozšírenia, odporúča sa aby sa tieto hlavičky boli v nasledujúcom poradí:

1. IPv6
2. Záhlavie medziľahlých možností (Hop-by-Hop Options Header)
3. DOH[1]
4. Smerovacie Záhlavie
5. Fragmentovacie záhlavie
6. Autentizačné záhlavie
7. Encapsulating Security Payload
8. DOH[2]
9. Záhlavie vyššej vrstvy

Každé rozširujúce záhlavie by sa malo vyskytovať maximálne raz, s výnimkou hlavičky Destination Options, ktorá by sa mala vyskytovať maximálne dvakrát (raz pred hlavičkou Smerovania a raz pred hlavičkou hornej vrstvy) [6].

1.4.1 Záhlavie medziľahlých možností (Hop-by-Hop Options)

Záhlavie „Hop-by-Hop Options“ sa používa na prenášanie voliteľných informácií, ktoré môže preskúmať a spracovať každý uzol pozdĺž cesty doručenia paketu. Detailnejší popis záhlavia je k dispozícii v dokumente *RFC 8200* [6].

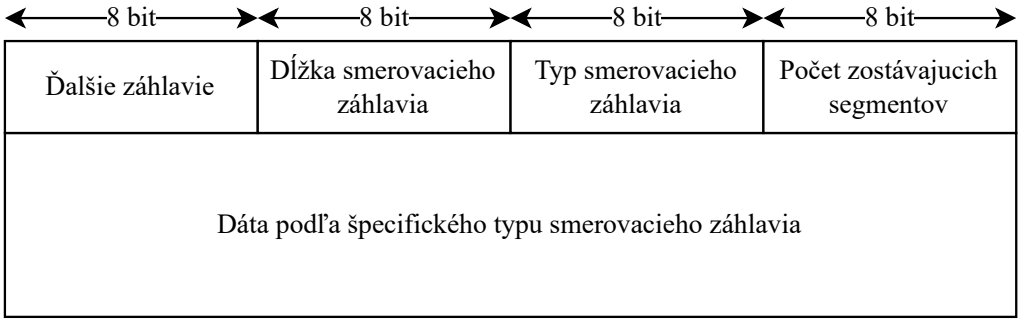
[1]Pre možnosť spracovania prvým príjemcom, ktorý je zapísaný v poli „Adresa príjemcu“ v IPv6 záhlaví a ostatnými adresátmi v RH.

[2]Pre možnosť spracovania len posledným adresátom paketu.

Záhlavie sa nevkladá ani nemaže, ale môže byť preskúmané alebo spracované ktorýmkoľvek uzlom počas cesty paketu, kým paket nedosiahne uzol (alebo každý z množiny uzlov, v prípade multicastu) identifikovaný v poli „Adresa príjemcu“ v IPv6 záhlaví. Záhlavie Hop-by-Hop Options, ak je prítomné, musí bezprostredne nasledovať po hlavičke IPv6. Jeho prítomnosť je indikovaná hodnotou 0 v poli „Ďalšie záhlavie“ hlavičky IPv6 [6].

1.4.2 Smerovacie Záhlavie (RH)

RH používa odosielateľ na uvedenie jedného alebo viacerých medziľahlých uzlov, ktoré majú byť „navštívené“ na ceste k príjemcovi. Táto funkcia je veľmi podobná funkcii „Loose Source“ v IPv4 a možnosti „Record Route“. Smerovacie záhlavie je označená pomocou hodnoty 43 v predchádzajúcom záhlaví v poli ďalšie záhlavie (Next Header) a má nasledujúci formát [6]:



Obr. 1.5: Smerovacie záhlavie [6].

Ďalšie záhlavie – 8 bitov, identifikuje typ hlavičky bezprostredne za hlavičkou Smerovania. Používa rovnaké hodnoty ako pole „Protokol“ v hlavičke IPv4.

Dĺžka smerovacieho záhlavia – 8 bitové celé číslo, označuje dĺžku RH 8–oktetových jednotkách, bez zahrnutia prvých ôsmych oktetov.

Typ smerovacieho záhlavia – 8 bitový identifikátor typu RH.

Počet zostávajúcich záhlaví – 8 bitové celé číslo bez znamienka. Označuje počet zostávajúcich úsekov trasy, t.j. počet explicitne uvedených medziľahlých uzlov, ktoré treba ešte navštíviť pred dosiahnutím konečného adresáta.

Dáta podľa špecifického typu smerovacieho záhlavia – Pole s premenlivou dĺžkou, formátu určeného typom smerovania a dĺžky tak, že úplná hlavička smerovania je celočíselný násobok ôsmych oktetov [6].

Typy RH:

- **Type 0 (Zastaralé)** - Tento typ záhlavia je z bezpečnostných hrozieb ako napríklad jednoduchá možnosť denial of service útoku od roku 2007 zastaraný [18].
- **Type 1 (Zastaralé)** - Používané pre Nimrod projekt financovaný DARPA. V roku 2009 bol označený za zastaraný [18].
- **Type 2** - Používané v *Mobile IPv6* (MIPv6). Definované tak, aby umožňovalo špecifické filtrovanie proti RH typu 0 [18].
- **Type 3** - *Routing Protocol for Low-Power and Lossy Networks* (RPL) Source Route Header pre nízkoenergetické a stratové siete [18].
- **Type 4** - *Segmentová smerovacia hlavička* (SRH) [18].
- **Type 253 (Súkromné účely)** - Možno použiť na testovanie, nie na skutočné implementácie. RFC3692-style Experiment 1 [18].
- **Type 254 (Súkromné účely)** - Možno použiť na testovanie, nie na skutočné implementácie. RFC3692-style Experiment 2 [18].

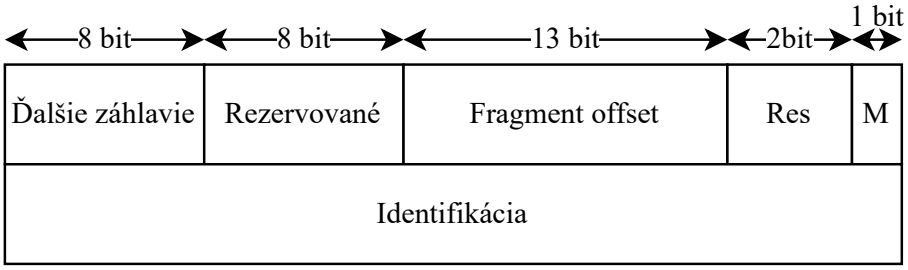
Ak počas spracovania prijatého paketu uzol narazí na RH s nerozpoznanou hodnotou typu smerovania, požadované správanie uzla závisí od hodnoty poľa „Počet zostávajúcich uzlov“ takto:

- Ak „Počet zostávajúcich uzlov“ je 0, uzol musí ignorovať RH a pokračovať v spracovaní ďalšej hlavičky v pakete, ktorej typ je identifikovaný poľom „Ďalšie záhlavie“ v RH [6].
- Ak je pole „Počet zostávajúcich uzlov“ nenulové, uzol musí zahodiť paket a poslať správu „ICMP Parameter Problem“ kód 0 na zdrojovú adresu paketu, ukazujúcu na nerozpoznaný typ smerovania [6].

Ak po spracovaní smerovacej hlavičky prijatého paketu medziľahlý uzol určí, že paket má byť presmerovaný na linku, ktorej MTU linky je menšia ako veľkosť paketu, uzol musí paket zahodiť a poslať správu „ICMP Packet Too Big“ odosielateľovi paketu [6].

1.4.3 Fragmentovacie záhlavie (Fragment header)

Fragmentovacie záhlavie sa používa na odoslanie paketu, ktorý by bol väčší ako MTU. Uzol rozdelí takýto paket na časti, fragmenty a koncový uzol je podla tohoto záhlavia schopný zostaviť paket do pôvodnej nefragmentovanej podoby (Poznámka: na rozdiel od IPv4 sa fragmentácia v IPv6 vykonáva iba zdrojovými uzlami, nie smerovačmi počas cesty paketu) [18] [6].



Obr. 1.6: Fragmentovacie záhlavie [6].

Ďalšie záhlavie – 8 bitov, identifikuje typ hlavičky bezprostredne za hlavičkou Smerovania. Používa rovnaké hodnoty ako pole „Protokol“ v hlavičke IPv4.

Rezervované – 8 bitové rezervované pole. Inicializované ako nula pre prenos, ignorované pri prijatí.

Fragment Offset – 13–bitové bezznamienkové číslo. offset v 8–oktetových jednotkách

Res – 2–bitové rezervované pole. Inicializované ako nula pre prenos, ignorované pri prijatí.

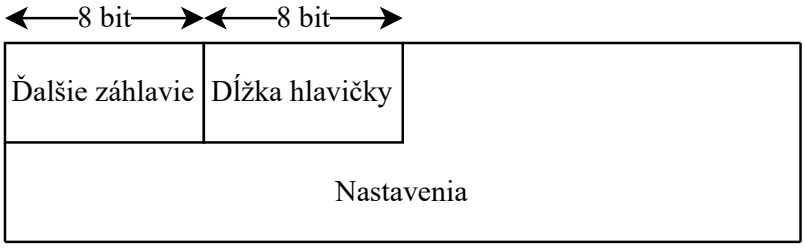
M – 1 = viac fragmentov; 0 = posledný fragment.

Identifikácia – 32 bitov. Pre každý paket, ktorý sa má fragmentovať, zdrojový uzol vygeneruje hodnotu identifikácie. Identifikácia sa musí líšiť od identifikácie akéhokoľvek iného fragmentovaného paketu odoslaného nedávno[3] s rovnakou zdrojovou adresou a cieľovou adresou. Ak je prítomné RH, cieľová adresa, o ktorú ide, je cieľová adresa konečného cieľa [6].

[3]maximálna pravdepodobná živnotnosť paketu, zahrňujúca čas cesty od zdroja k cieľu a čas znovu–zloženia paketu z fragmentov.

1.4.4 Hlavička možností destinácie (DOH)

DOH sa používa na prenášanie voliteľných informácií, ktoré musia byť spracované iba cieľovým uzlom (uzlami) paketu. Hlavička Možnosti destinácie je identifikovaná hodnotou Next Header 60 v bezprostredne predchádzajúcej hlavičke a má nasledujúci formát:



Obr. 1.7: *Hlavička možností destinácie* [6].

Ďalšie záhlavie – 8 bitov, identifikuje typ hlavičky bezprostredne za hlavičkou Smerovania. Používa rovnaké hodnoty ako pole „Protokol“ v hlavičke IPv4.

Dĺžka smerovacieho záhlavia – 8 bitové celé číslo, označuje dĺžku DOH 8–oktetových jednotkách, bez zahrnutia prvých ôsmych oktetov.

Nastavenia – Pole s premenlivou dĺžkou tak, že DOH je celočíselný násobok dĺžky 8 oktetov. Obsahuje jednu alebo viac možností *Type length value* (TLV) [6].

1.4.5 Autentizačné záhlavie (AH)

IP AH slúži pre poskytnutie bezspojovej integrity, overenia pôvodu dát pre IP diagramy a ochranu pred replikátmi. Všetky byty v IP AH budú spracované. Veľkosť tohto záhlavia je počítaná na základe pola Payload Length, ktoré sa nachádza v záhlaví [18].

1.4.6 Encapsulating security Payload (ESP)

ESP je navrhnutý aby poskytovať viaceré bezpečnostné služby pre IPv6 ako napríklad diskrétnosť, overenie pôvodu dát, bezspojovú integritu, anti-replikačné služby. AH a Encapsulating Security Payload sa využívajú rovnako v IPv6 aj IPv4. Sú súčasťou IPsec [18].

1.4.7 Absencia ďalšieho záhlavia

Hodnota 59 v poli „Ďalšie záhlavie“ v hlavičke IPv6 alebo v akejkoľvek rozširujúcej hlavičke indikuje, že už nenasleduje žiadne ďalšie záhlavie. Ak pole „Celková

dĺžka prenášaných dát“ hlavičky IPv6 indikuje prítomnosť oktetov za koncom hlavičky, ktorej pole „Ďalšie záhlavie“ obsahuje 59, tieto oktety musia byť ignorované a odovzdané nezmenené, ak je paket preposielaný [18].

1.5 Smerovanie

Smerovanie v IPv6 je založené na totožných princípoch ako v IPv4, ale pracuje s dlhšími adresami. Tomu sa museli prispôsobiť aj smerovacie protokoly, preto sa prakticky u všetkých IGP protokolov stretáme s verziami pre IPv6. Sú to hlavne:

- **RIPng** (*Router Information Protocol Next Generation*) – verzia semrovacieho protokolu RIP pre IPv6,
- **EIGRP for IPv6** (*Enhanced Interior Gateway Routing Protocol*) – smerovací protokol firmy Cisco vo verzii pre IPv6,
- **OSPFv3** (Open shortest Path First) – OSPF pre IPv6 siete,
- **IS-IS for IPv6** (Intermediate System to Intermediate System) – verzia IS-IS pre IPv6.

V oblasti smerovania medzi autonómnymi systémami sa používa *Border Gateway protocol* (BGP). Pre IPv4 siete sa využíva verzia 4, označovaná ako BGP4, pre IPv6 existuje modifikovaná verzia označovaná ako **BGP4+** [15]

1.6 Vkladanie vlastných informácii do IPv6 paketu

Vkladanie vlastných informácii do hlavičky IPv6 môžeme realizovať definovaním vlastného rozširujúceho záhlavia.

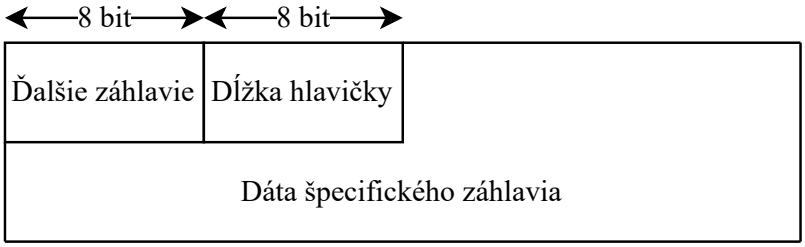
Definovanie nového EH IPv6 sa neodporúča, pokiaľ neexistujú žiadne existujúce EH IPv6, ktoré možno použiť zadaním novej možnosti pre tento EH IPv6. Návrh na špecifikáciu novej hlavičky rozšírenia IPv6 musí obsahovať podrobné technické vysvetlenie, prečo nie je možné použiť existujúce EH IPv6 pre požadovanú novú funkciu. Ďalšie informácie nájdete v dokumente *RFC6564*.

Nové EH, ktoré požadujú správanie „Hop-by-hop“ nesmú byť definované, pretože jediné EH, ktoré má „Hop-by-hop“ správanie je *Záhlavie medziľahlých možností*.

Nové záhlavie takéhoto typu nie sú odporúčané, pretože uzly by mohli byť nakonfigurované tak aby, tento typ záhlaví ignorovali alebo zahadzovali pakety obsahujúce takýto typ záhlavia. Dizajnéri, ktorí zvažujú definovanie nových možností hop-by-hop, si musia byť vedomí tohto pravdepodobného správania. Musí existovať veľmi jasné odôvodnenie, prečo je potrebná akákoľvek nová možnosť hop-by-hop predtým, ako bude štandardizovaná.

Namiesto definovania nových hlavičiek rozšírenia sa odporúča, aby bola použitá „Hlavička možností destinácie“ na prenášanie vlastných informácií, ktoré musia byť spracované iba cieľovým uzlom (uzlami) paketu, pretože poskytujú lepšiu manipuláciu a spätnú kompatibilitu [6].

Ak sú definované nové EH, musia používať nasledujúci formát:



Obr. 1.8: Vlastné *Rozširujúce záhlavie* [6].

2 Segmentové smerovanie skrz IPv6 (SRv6)

2.1 Segmentové smerovanie (SR)

Segmentové smerovanie, SR, môžeme aplikovať pre IPv6 použitím rozširujúceho Smerovacie Záhlavie, *Segmentová smerovacia hlavička* (SRH). Táto časť práce popisuje SRH a ho spracovávajú uzlamy, ktoré podporujú SR. SR-MPLS delí cestu paketu na segmenty, prideluje SID segmentom a zapisuje do paketov informácie o segmetoch pri odoslaní a tým „vedie“ paket po predom definovanej ceste. SRv6 sa považuje za nástupcu SR-MPLS. Segment je IPv6 adresa. SRH sa pridáva k IPv6 záhlaviu a každý uzol preposiela paket na základe cesty definovanej v SRH [11].

Medzi hlavné výhody SRv6 patria:

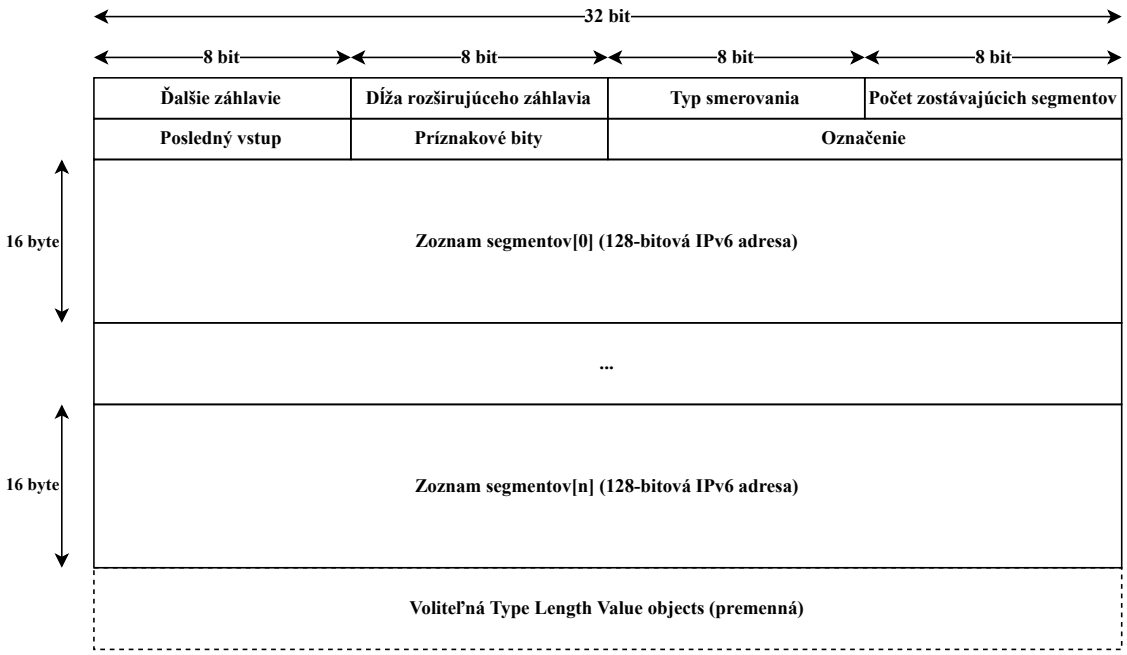
- Jednoduchosť
  - Pri použití s MPLS ponúka možnosť tunelovania služieb MPLS (VPN, VPLS a VPWS) z krajného poskytovateľa vstupu na krajného poskytovateľa výstupu bez akéhokoľvek iného protokolu ako IGP (IS-IS alebo OSPF).
  - Jednoduchšia obsluha bez samostatných protokolov na distribúciu štítkov (napríklad bez LDP alebo RSVP).
  - Lepšie využitie existujúcej infraštruktúry, pre nízko–kapitálové výdavky.
- Spätná kompatibilita – SRv6 sme schopný nasadiť v akejkoľvek sieti podporujúcej IPv6.
- Škálovateľnosť
  - Možnosť vyhnúť sa veľkému množstvu štítkov v LPD databáze.
  - Možnosť vyhnúť sa veľkému množstvu MPLS Traffic Engineering LSPs v sieti.
  - Nie je potreba konfigurovať veľké množstvo tunelov.
- Široké možnosti riadenia prevádzky

Pri SRv6:

- Prefix-SID je adresa IPv6.
- Operátor MUSÍ explicitne vytvoriť inštanciu SRv6 SID. Adresy uzlov IPv6 nie sú štandardne SID SRv6 [11].

2.2 Segmentová smerovacia hlavička (SRH)

SRH je RH typu 4 ako bolo spomenuté v časti 1.4.2. SRH špecifikuje explicitnú cestu k cieľu a uchováva informácie o zozname segmentov [11]. SRH pridáva do paketu IPv6 niektoré metaúdaje so zoznamom prvkov prenosovej cesty (napr. uzly, linky, služby atď.), ktorými musí paket prejsť a ktoré sú reprezentované protokolom IPv6. Kompletná zdrojovo-smerovaná cesta môže byť obsiahnutá v pakete pomocou jediného segmentu [13].



Obr. 2.1: Segmentová smerovacia hlavička (SRH) [11]

Ďalšie záhlavie (Next Header) – 8 bitov, slúži k identifikácii typu záhlavia ktoré bude nasledovať po IPv6 záhlaví. Používa rovnaké hodnoty ako IPv4 Protocol field [11].

Dĺžka rozširujúceho záhlavia (Hdr Ext Len) – 8 bitov, obsahuje celé číslo bez znamienka, ktoré popisuje dĺžku SRH záhlavia. Pokrýva dĺžku od Segment List [0] do Segment List [n] [11].

Typ smerovania (routing Type) – 8 bitov, obsahuje hodnotu 4 (Segment Routing) [11].

Počet zostávajúcich segmentov (Segments Left) – 8 bitov, obsahuje celé číslo bez znamienka, ktoré obsahuje počet medziľahlých uzlov ktorými bude musieť paket prejsť aby sa dostal ku koncovej (cieľovej) stanici [11].

Posledný vstup (last entry) – 8 bitov, reprezentuje index posledného segmentu v zozname segmentov (Segment List[n]) [11].

Príznakové bity (flags) – 8 bitov, obsahuje identifikátory v dátovom pakete [11].

Označenie (tag) – 16 bitov, slúži na označenie paketov patriacich do rovnakej skupiny alebo triedy. Napríklad pakety s rovnakými vlastnosťami. Ak sa pole Tag nepoužíva, musí ho zdroj nastaviť na 0 pred odoslaním [11].

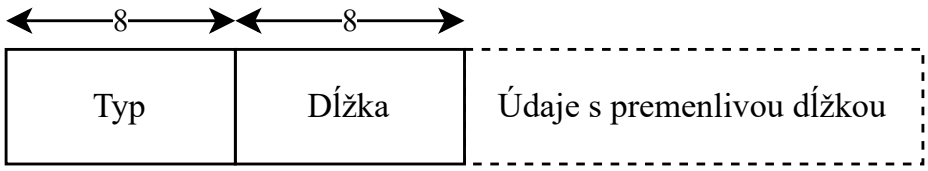
Zoznam segmentov (segment List[0. . . n)] – 128-bit bitov, obsahuje hodnotu reprezentujúcu IPv6 adresu n-tého segmentu v zozname segmentov (Segment List) [11].

Type length value (TLV) – TLV poskytujú metadáta pre spracovanie segmentov. Počas spracovania SID sú všetky TLV ignorované, pokiaľ lokálna konfigurácia neuvádza inak. Tým pádom je toto pole v prípade implementácie voliteľné. TLV sú prítomné, ak dĺžka rozširujúceho záhlavia (Hdr Ext Len) prekročí prvok v poli posledný vstup (last entry) v zozname segmentov. Pri spracovávaní TLV v koncovom bode segmentu MUSIA byť TLV plne obsiahnuté v SRH, ako to určuje Dĺžka rozširujúceho záhlavia. Detekcia TLV prekračujúcich hranicu SRH dĺžku rozširujúceho záhlavia má za následok problém s parametrom ICMP, kód 0, správu na odosielateľovu adresu, ktorá ukazuje na pole dĺžky rozširujúceho záhlavia v SRH, a paket sa zahodí [11, 12].

2.2.1 SRH TLV

TLV sú prítomné, keď je dĺžka rozširujúceho záhlavia väčšia ako (Posledný vstup+1)*2.

Počas spracovania TLV v koncovom bode segmentu MUSIA byť TLV plne obsiahnuté v SRH, ako to určuje Hdr Ext Len. Detekcia TLV prekračujúcich hranicu SRH Hdr Ext Len má za následok problém s parametrom ICMP, kód 0, správu na zdrojovú adresu, ktorá ukazuje na pole Hdr Ext Len v SRH, a paket sa zahodí [11, 12].



Obr. 2.2: TLV

Typ – 8-bitový kódový bod z "Segment Routing Header TLV". Nerozpoznané typy musia byť pri prijatí ignorované.

Dĺžka – Dĺžka dátového poľa s premenlivou dĺžkou v bajtoch.

Údaje s premenlivou dĺžkou – údaje, ktoré sú špecifické pre daný typ.

Položky typu TLV (Type Length Value) obsahujú VOLITEĽNÉ informácie, ktoré môže použiť uzol identifikovaný v cieľovej adrese (DA) paketu.

Každý TLV má svoju vlastnú dĺžku, formát a sémantiku. Kódový bod pridelený (IANA) každému typu TLV definuje formát aj sémantiku informácií prenášaných v TLV. V rovnakom SRH môže byť zakódovaných viacero TLV.

Bit najvyššieho rádu typu TLV (nultý bit) určuje, či sa údaje TLV tohto typu môžu zmeniť na ceste do konečného cieľa paketu:
0 : Údaje TLV sa počas cesty nemenia.
1 : Údaje TLV sa počas cesty menia.
Všetky TLV špecifikujú svoje požiadavky na zarovnanie pomocou formátu xn+y. Formát xn+y je definovaný podľa [RFC8200]. Zdrojové uzly SR používajú pri konštrukcii SRH požiadavky na zarovnanie xn+y TLV a Padding TLV.

Pole Dĺžka TLV sa používa na preskočenie TLV pri kontrole SRH v prípade, že uzol nepodporuje alebo nerozpozná typ. Dĺžka definuje dĺžku TLV v oktetoch, bez polí Typ a Dĺžka [12].

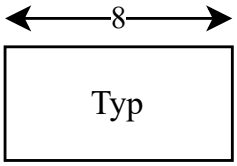
2.2.2 Vyplňovacie TLV

Existujú dva typy vyplňovacích TLV, Pad1 a PadN, a pre oba platí nasledovné:

- Vyplňovacie TLV sa používajú na splnenie požiadavky na zarovnanie nasledujúcich TLV.
- Vyplňovacie TLV sa používajú na doplnenie SRH na násobok 8 oktetov.
- Výplňové TLV sú ignorované uzlom spracovávajúcim SRH TLV.
- V jednom SRH MÔŽE byť použitých viacero výplňových TLV [12].

Pad1

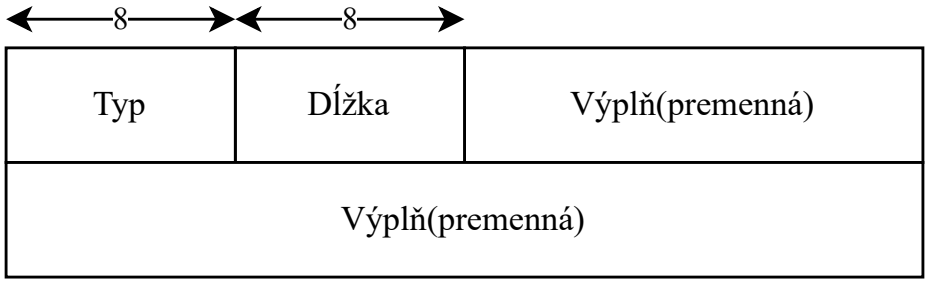
Ak je potrebný jeden bajt výplne, MUSÍ sa použiť jeden Pad1 TLV. Pad1 TLV sa NESMIE použiť, ak sa vyžaduje viac ako jeden po sebe idúci bajt výplne. Pole typ obsahuje hodnotu 0 [12].



Obr. 2.3: Pad1

PadN

PadN TLV sa používa na doplnenie SR hlavičky na násobok 4 bajtov. Obsahuje pevné pole dĺžky, ktoré určuje počet bajtov, ktoré treba pridať k SR hlavičke, aby sa dosiahla požadovaná dĺžka. Hodnota poľa PadN môže byť od 0 do 255 bajtov [12]..



Obr. 2.4: PadN

Typ – obsahuje hodnotu 4
Dĺžka – 0 až 5. Dĺžka poľa „Výplň“ v bajtoch.
Výplň – Výplňové bity nemajú žiadnu sémantiku. MUSIA byť nastavené na 0 pri prenose a ignorované pri prijatí.
PadN TLV sa MUSÍ použiť, ak sa vyžaduje viac ako jeden bajt výplne [12].

2.3 SR uzly

Existujú rôzne typy uzlov, ktoré môžu byť zapojené do sietí smerovania segmentov: zdrojové uzly SR, ktoré vytvárajú pakety so segmentom v cieľovej adrese hlavičky IPv6, tranzitné uzly, ktoré preposielajú pakety smerované do vzdialeného segmentu, a uzly koncových bodov segmentu SR, ktoré spracovávajú lokálny segment v cieľovej adrese hlavičky IPv6 [12].

2.3.1 SR zdrojový uzol

Zdrojový uzol SR je akýkoľvek uzol, ktorý vytvára paket IPv6 so segmentom (t.j. SRv6 SID) v cieľovej adrese hlavičky IPv6. To zahŕňa buď:

- Hostiteľ, ktorý vytvára paket IPv6
- Vstupný smerovač domény SR zapuzdrujúci prijatý paket do vonkajšej hlavičky IPv6, za ktorým nasleduje voliteľný SRH [12].

2.3.2 Tranzitný uzol

Tranzitný uzol je akýkoľvek uzol, ktorý preposiela paket IPv6, pričom cieľová adresa tohto paketu nie je lokálne nakonfigurovaná ako segment alebo lokálne rozhranie. Nie je potrebné, aby bol tranzitný uzol schopný spracovať segment alebo SRH [12].

2.3.3 Uzol koncového bodu segmentu SR

Uzol koncového bodu segmentu SR je akýkoľvek uzol prijímajúci paket IPv6, kde je cieľová adresa tohto paketu lokálne nakonfigurovaná ako segment alebo lokálne rozhranie [12].

2.4 Proces posielania paketov v sieti

Táto časť popisuje spracovanie paketov SRv6 v zdrojových uzloch SR, tranzitných a koncových bodoch segmentu SR. Celý proces preposielania paketov v sieti ktorá vie pracovať s SRv6 nie je zložitý a je popísaný formou sekcii, ktoré informujú ako sa paket v danej situácii spracováva [12].

2.4.1 SR zdrojový uzol

Zdrojový uzol nasmeruje paket do politiky SR. Ak má politika SR za následok zoznam segmentov obsahujúci jeden segment a nie je potrebné pridávať informácie k príznaku SRH alebo pridávať TLV; DA je nastavený na jednu položku zoznamu segmentov a SRH MÔŽE byť vynechaný.

SRH sa vytvára takto:

- Polia Next Header a Hdr Ext Len sú nastavené podľa špecifikácie v [RFC8200].
- Pole Typ smerovania je nastavené na 4.
- DA paketu je nastavené na hodnotu prvého segmentu.
- Prvým prvkom zoznamu segmentov SRH je konečný segment. Druhým prvkom je predposledný segment atď.
- Pole Segments Left je nastavené na n-1, kde n je počet prvkov v politike SR.
- Pole Posledný záznam je nastavené na n-1, kde n je počet prvkov v Politike SR.
- TLV (vrátane HMAC) môžu byť nastavené podľa ich špecifikácie.
- Paket je presmerovaný na cieľovú adresu paketu (prvý segment).

Ak je zdrojový uzol schopný SR, vytvorí *Segmentová smerovacia hlavička* (SRH) do ktorého zapíše zoznam segmentov v opačnom poradí ako je cesta k cieľu (Segment List [0] reprezentuje posledný segment a Segment List [n-1] je prvý segment cesty).

Pole „Segments Left“ nastaví na hodnotu n-1, pričom n reprezentuje počet segmentov. IP Adresa príjemcu sa nastaví na n-1 segment (prvý segment cesty) a následne ho odošle na DA ako v prípade štandardného smerovania IPv6 [12].

Redukovaná SRH

Ak zdroj nevyžaduje, aby bol celý zoznam SID uchovaný v SRH, môže sa použiť redukovaná SRH.

Redukovaná SRH neobsahuje prvý segment súvisiacej politiky SR (prvý segment je ten, ktorý je už v DA hlavičky IPv6) a pole posledný vstup je nastavené na n-2, kde n je počet prvkov. v Politike SR [12].

2.4.2 Tranzitný uzol bez podpory SR

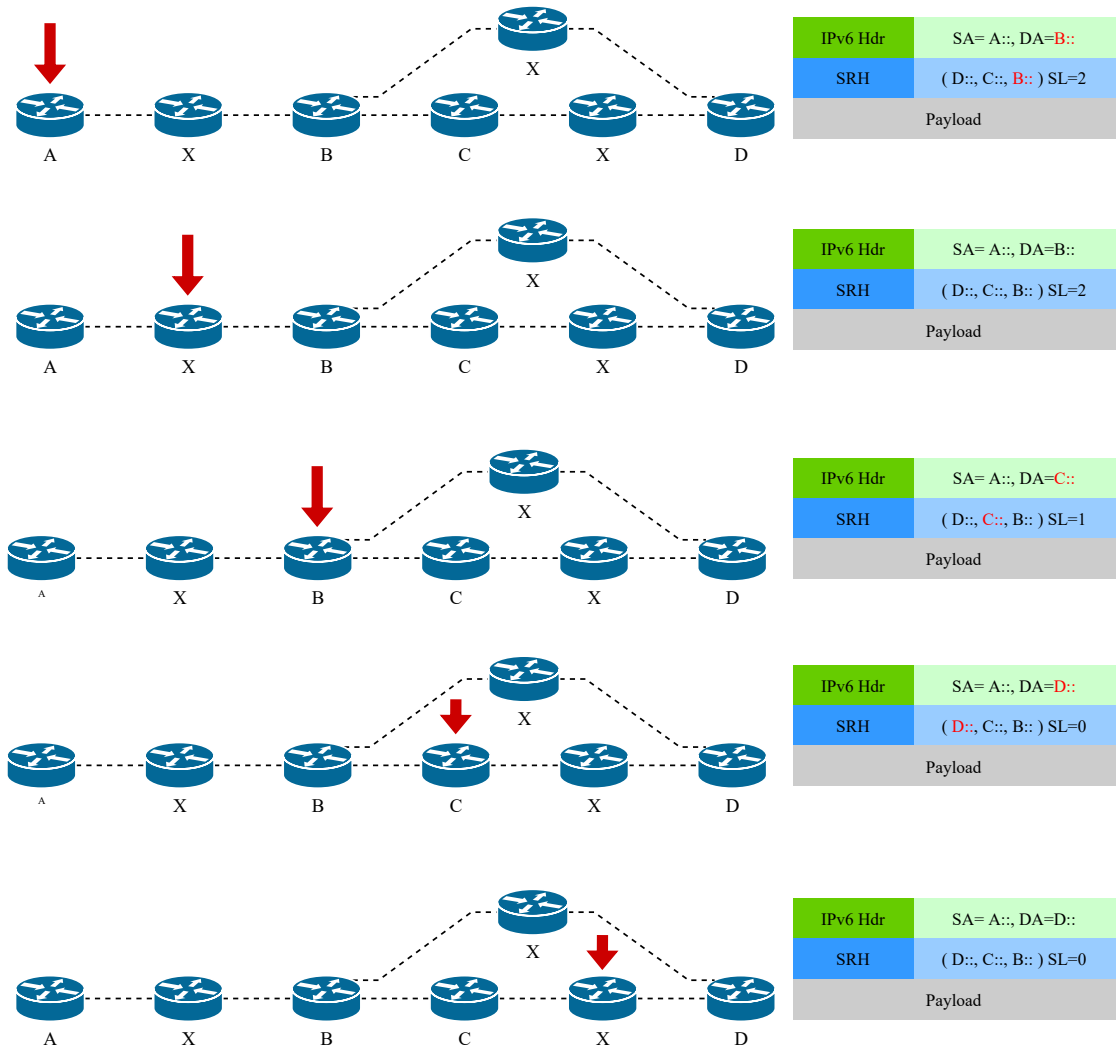
Uzol, ktorý nie je schopný *Segmentové smerovanie skrz IPv6* (SRv6) realizuje smerovanie na základe IPv6 Adresa príjemcu. S Segmentová smerovacia hlavička nevykonáva žiadne úpravy.

Jediný uzol, ktorý môže kontrolovať hlavičku rozšírenia smerovania (a teda aj *Segmentové smerovanie skrz IPv6* (SRv6)), je uzol zodpovedajúci DA paketu. Akýkoľvek iný tranzitný uzol NESMIE kontrolovať spodnú smerovaciu hlavičku a MUSÍ posielať paket smerom k DA podľa svojej smerovacej tabuľky IPv6 [12].

2.4.3 Uzol koncového bodu SR

SR koncové uzly sú uzly podporujúce Segmentové smerovanie, ktorých adresa je IP Adresa príjemcu. SR koncové uzly spracovávajú SRH nasledovne:

- Ak je pole Segments Left > 0 tak uzol jeho hodnotu zníží o 1. Do adresy príjemcu vloží hodnotu Segment List [Segmets Left] a presmeruje paket do novej IP adresy príjemcu.
- Ak je pole Segments Left $= 0$ tak uzol odstráni IP a SR záhlavie a následne šandardne spracováva payload (uzol nemusí podporovať SRv6).



Obr. 2.5: Zjednodušený príklad fungovania SRv6 v sieti

Na obrázku 2.5 je znázornený príklad prevádzky v sieti čiastočne podporujúcej SRv6. Uzly, ktoré SRv6 nepodporujú sú označené ako „X“. Smerovanie prebieha následovne:

1. Zdrojový uzol vytvorí SRH, do ktorého zapíše všetky uzly, ktoré paket musí navštíviť na ceste od zdroja k cieľu, Následne nastaví hodnotu SL (počet zostávajúcich segmetov) na počet týchto uzlov (segmentov) - 1. ako je popísané v 2.4.1.
2. Uzol X, ktorý nie je schopný spracovávať SRH bude smerovať len podľa štandardného IPv6 záhlavia, ktoré má nastavené adresu príjemcu (DA) „B::“.

3. Uzol B je schopný spracovať SRH a nastaví novú hodnotu DA na ďalšiu hodnotu zo zoznamu segmentov, „C::“. Tým zabezpečí že paket nepôjde po najkratšej trase ale po trase nami určenej t.j. cez uzol C a X aby sa dostal k príjemcovi. Následne zníži hodnotu SL o 1 a paket prepošle na DA.
4. Uzol C nastaví hodnotu DA na ďalšiu hodnotu zo zoznamu segmentov, „D::“. Následne nastaví hodnotu SL na 0 a pošle paket ďalej na uzol X.
5. Uzol X bude opäť SRH ignorovať a bude paket smerovať len pomocou DA t.j. do uzla D.

Vďaka tomuto správaniu je SRv6 možné použiť v akejkoľvek sieti bez väčších úprav.

2.4.4 SRv6 Sieťové programovanie

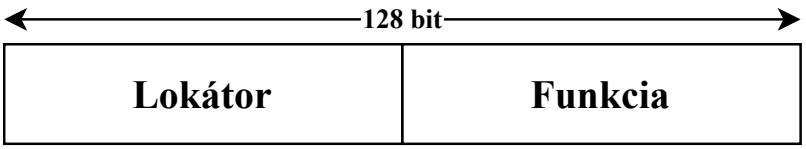
SRv6 sieťové programovanie odkazuje na možnosť aplikácii zakódovať ľubovolne zložitý program ako sériu samostatných funkcii rozosielaných po SRv6 sieti [8] [9].

Sieťové programovanie kombinuje funkcie segmentového smerovania, jednoduché aj zložité, na dosiahnutie sieťového cieľa, ktorý presahuje obyčajné smerovanie paketov [10].

2.4.5 SRv6 Segment

Segment SRv6 je 128-bitové číslo, ktoré sa tiež nazýva SRv6 SID alebo SID. SID je inštanciovaná adresa IPv6, ktorej je priradená jedinečná funkcia. SRv6 SID identifikuje uzol, prepojenie, L2VPN, L3VPN alebo službu. SRv6 SID môže definovať akúkoľvek sieťovú funkciu [9].

SRv6 SID je sieťová inštrukcia, ktorá pozostáva z lokátora a funkcie. Lokátor je zodpovedný za smerovanie a musí byť jedinečný v rámci domény SR. Funkcia identifikuje akúkoľvek funkciu na zariadení. Funkcia napríklad identifikuje akciu preposielania alebo službu. Štruktúra SRv6 SID uľahčuje programovanie siete. Obrázok 2.6 zobrazuje štruktúru SRv6 SID [9].



Obr. 2.6: SRv6 Segment Identifikátor (SID) [9].

SRv6 SID sú rôznych typov. Každý typ SRv6 SID označuje špecifickú funkciu. End SID a End.X SID sú najbežnejšie SRv6 SID. End.DT4 SID znamená IPv4 VPN.

3 Program Scapy

Táto kapitola je venovaná programu Scapy, programu pre manipuláciu paketov. Scapy umožňuje vytvárať alebo dekódovať, posielať alebo zachytávať pakety. Scapy obsahuje aj funkcie ako tracerouting, probing, unit testy, útoky a network discovery.

V našej práci bude slúžiť hlavne ako nástroj pre generovanie IPv6 paketov so záhlavím SRv6, ktoré budeme potom ďalej používať v generátore sieťovej prevádzky Trex. Scapy je program písaný v Pythone a bol vytvorený Philippe Biondim.

Obr. 3.1: Úvodná obrazovka programu Scapy zobrazená v Linux konzole

3.1 Základná charakterisitka programu Scapy

Myšlienka programu je jednoduchá. Scapy robí primárne dve veci, odosiela pakety a prijíma odpovede. Pomocou Scapy je možné definovať množinu paketov, odoslať ich, spárovať požiadavky s odpoveďami a ako návratovú hodnotu dostať zoznam párov paketov a zoznam nespárovaných paketov.

Okrem toho môžeme definovať vlastné funkcie ako napríklad funkciu ktorá pinguje zariadenia v sieti a ako návratovú hodnotu vracia zoznam zariadení, ktoré odpovedali. Alebo funckiu, ktorá robí portscan a vracia LaTeX report [5].

3.1.1 Rýchly dizajn paketov

Iné nástroje pre tvorbu paketov sa držia vzoru programu, v ktorom zadávame príkazy do shellu. Výsledkom toho je obtiažny syntax pre popis paketu.

Scapy k tomuto problému pristupuje predstavením *Domain Specific Language* (DSL), ktorý umožňuje presný a rýchly popis akéhokoľvek paketu. Použitie syntaxe Pythonu a interpreter Pythonu ako syntaxe a interpreter DSL má mnoho výhod. Nie je potreba písať vlastný interpreter, používatelia sa nemusia učiť ďalší jazyk.

Scapy umožňuje jeho používateľovi popísať paket alebo množinu paketov pomocou špecifikovania jednotlivých vrstiev ako sú na sebe naskladané. Polia každej vrstvy majú predvolené hodnoty, ktoré užívateľ môže zmeniť. Užívateľ nie je nútený používať preddefinované metódy alebo šablóny. Vďaka tomu nie je potreba aby sa použil iný nástroj pri rôznych scenároch. Priemerný počet riadkov na popísanie paketu v jazyku C je 60. Pomocou Scapy je používateľ schopný popísať paket v jednom riadku [5].

Výpis 3.1: Príklad definovania paketu(DA=192.168.1.1, TTL=10), Protokol IP

```
1 >>> a=IP(ttl=10)
2 >>> a.dst="192.168.1.1"
3 >>> a
4 < IP ttl=10 dst=192.168.1.1 |>
```

3.1.2 Vizualizácia dát

Keď sondujeme sieť, posielame veľa správ zatiaľ čo odpoveď dostane len málo. Narozdiel od iných nástrojov, Scapy dáva všetky informácie, napríklad všetky poslané správy a všetky prijaté odpovede. Skúmaním týchto dát používateľ nájde potrebnú informáciu. Ak je dataset malý tak ju môže jednoducho hľadať. V ostatných prípadoch bude interpretácia dát závisieť od uhla pohľadu. Väčšina programov zvolí uhoľ pohľadu a všetky dáta, ktoré nepotrebuje pre interpretáciu zahodí. Vďake tomu že Scapy poskytuje raw dáta, tak používateľ je schopný tieto dáta použiť pre viacero uhlov pohľadu. Napríklad, používateľ urobí TCP port sken a zobrazí dáta ako výsledok port skenu. Následne môže použiť tie isté dáta pre zobrazenie s ohľadom na TTL odpovede. Nemusí znovu probovať sieť aby mohol upraviť uhol pohľadu [5].

3.1.3 Dekódovanie dát

Problém s väčšinou sieťových nástrojov je ten že sa snažia interpretovať prijaté odpovedi namiesto toho aby ich len dekódovali a dali k dispozícii používateľovi. Hlásenie **Received a TCP Reset on port 80** nie je náchylné na interpretačnú chybu. Hlásenie **Port 80 is closed** je interpretácia, ktorá môže byť správna väčšinu času, ale zlá v špecifickom kontexte s ktorých autor daného programu nepočítal. Napríklad niektoré skenery občas hlásia **Filtered TCP port**, keď dostanú **ICMP destination unreachable** paket. To môže byť správne ale v niektorých prípadoch

to nemusí znamenať že paket neprešiel cez firewall ale že nebol host, ktorému by sa mal paket posunúť [5].

Interpretovanie dát môže pomôcť používateľom, ktorý nevedia čo je port sken ale zároveň môže spôsobiť škody, nakoľko do výsledkov vnáša zaujatosť. Skúsení používatelia sa môžu pokúsiť spätne prísť na to ako nástroj interpretuje dáta aby mohli zistiť prečo dostali danú interpretáciu dát. Nanešťastie sa takýmto procesom stráca veľa dát [5].

Predvolené hodnoty

Scapy sa bude snažiť použiť predvolené hodnoty pre všetky polia paketu, ak ich užívateľ neprepíše [5].

Ostatné polia sa vyplnia hodnotami, ktoré sa považujú za najviac užitočné:

- TCP source port bude 20, destination port bude 80.
- UDP source a destination porty budú 53.
- Typ ICMP bude echo request.

Tab. 3.1: Príklad: Predvolené hodnoty pre IP [5]

Name	**Type**	**Default value**
version	BitField	4
ihl	BitField	none
tos	XByteField	0
len	ShortField	none
id	ShortField	1
flags	FlagsField	0
frag	BitField	0
TTL	ByteField	64
proto	ByteEnumField	0
chksum	XShortField	none
src	Emph	none
dst	Emph	('127.0.0.1')
options	IPoptionsField	(")

3.2 Tvorba IPv6 paketov pomocou programu Scapy

Tvorenie paketov v programe Scapy je jednoduché a intuitívne. Program Scapy beží prostredníctvom terminálu.Scapy má k dispozícii voliteľné balíčky s prídavnými funkciami, ktoré nám umožňujú použiť nové funkcie. O neprítomnosti balíčkov nás Scapy pri zapnutí informuje prostredníctvom správ:

- INFO: Can't import python matplotlib wrapper. Won't be able to plot.
- INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

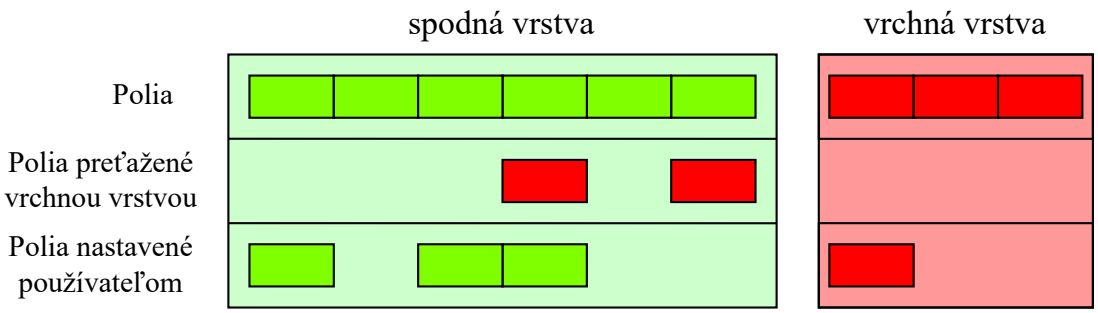
Základné funkcie, odosielanie a prijímanie paketov by malo však fungovať stále [5].

Výpis 3.2: Spustenie programu Scapy

```
virtual-machine:~$ sudo scapy -H
Welcome to Scapy (2.5.0rc1) using IPython 8.5.0
>>>
```

3.2.1 Vrstvenie

Scapy využíva symbol „/“ ako operátor pre oddelenie jednotlivých vrstiev. Pritom „spodná“ vrstva môže mať jedno alebo viac svojich predvolených polí preťažených „vrchnou“ vrstvou. Názornejšie zobrazenie tohoto procesu je na 3.2 [5].



Obr. 3.2: Vizualizácia vrstvenia [5].

3.2.2 Triedy v Scapy

Scapy pre tvorbu paketu využíva objekty, kotré majú rôzne parametry. Tieto parametry je potom možné individuálne nastavovať a z objektov (jednotlivé vrstvy) je možné vytvoriť paket pomocou vrstvenia [5].

Trieda IPv6

Trieda IPv6 v Scapy reprezentuje základné záhlavie IPv6 uvedené na obr. 1.2. Jednotlivé polia môžeme vyplniť ak ich predáme ako argument konštruktoru alebo jednotlivé polia plníme explicitne ako je ukázané vo výpise 3.6 [5].

Výpis 3.3: Polia v triede IPv6 a ich predvolené hodnoty.

```
1 >>> ls(IPv6)
2 version    : BitField  (4 bits)                  = ('6')
3 tc         : BitField  (8 bits)                  = ('0')
4 fl         : BitField  (20 bits)                 = ('0')
5 plen       : ShortField                          = ('None')
6 nh         : ByteEnumField                       = ('59')
7 hlim       : ByteField                           = ('64')
8 src        : SourceIP6Field                      = ('None')
9 dst        : DestIP6Field                        = ('None')
```

Trieda IPv6ExtHdrSegmentRouting

Trieda IPv6ExtHdrSegmentRouting v Scapy reprezentuje *Smerovacie Záhlavie* (RH), ktoré sa delí na rôzne typy, ako to je uvedené v 1.4.2. Jednotlivé polia môžeme vyplniť ak ich predáme ako argument konštruktoru alebo jednotlivé polia plníme explicitne ako je ukázané vo výpise 3.6. Pre naše potreby bude hodnota „type“ vždy natavená na 4 (SRH). Hodnota „addresses“ je pole IPv6 adries, do ktorého budeme vkladať adresy pre SRv6 smerovanie. Toto pole odpovedá poľu „zoznam segmentov“, ktoré je popísané v 2.2. Pole „segleft“ odpovedá poľu „zostávajúce segmenty“, ktoré je popísané v 2.2 a pri exporte do súboru „cap“ alebo „pcap“ sa nastaví automaticky [5].

Výpis 3.4: Polia v triede IPv6ExtHdrSegmentRouting a ich predvolené hodnoty.

```
1 >>> ls(IPv6ExtHdrSegmentRouting)
2 nh         : ByteEnumField                       = ('59')
3 len        : FieldLenField                       = ('None')
4 type       : ByteField                           = ('0')
5 segleft    : ByteField                           = ('None')
6 reserved   : BitField  (32 bits)                 = ('0')
7 addresses  : IP6ListField                        = ('[]')
```

IPv6ExtHdrSegmentRoutingTLVPadN

Trieda IPv6ExtHdrSegmentRoutingTLVPadN v Scapy reprezentuje SRH TLV typu PadN, ktoré je popísané v sekcii 2.2.2. Túto triedu budeme v práci ďalej používať pre tvorbu vlsatného nástroja na analýzu a manipuláciu paketov.

Výpis 3.5: Polia v triede IPv6ExtHdrSegmentRoutingTLVPadN a ich predvolené hodnoty.

```
1 >>> ls(IPv6ExtHdrSegmentRoutingTLVPadN)
2 type        : ByteEnumField                    = ('4')
3 len         : FieldLenField                    = ('None')
4 padding     : StrLenField                      = ("b'\\x00'")
```

3.2.3 Práca s paketmi

Scapy umožňuje jednoduchú tvorbu a manipuláciu s paketmi. Ako je ukázané na obrázku 3.3, celý paket môžeme vytvoriť v podstate v jednom riadku tak, že hodnoty ktoré chceme zapísať do príslušných polí predáme ako argumenty. Na obrázku 3.3 predávame *IPv6ExtHdrSegmentRouting()* ako argument *src*, *dst*, *addresses* a *type*. Rovnakým spôsobom by sme mohli nastaviť aj ostatné hodnoty polí [5].

```
>>> paket = IPv6(src='::1', dst='::9')/IPv6ExtHdrRouting(type = 4, addresses=['::2','::3'])
>>> paket.display()
###[ IPv6 ]###
  version   = 6
  tc        = 0
  fl        = 0
  plen      = None
  nh        = Routing Header
  hlim      = 64
  src       = ::1
  dst       = ::9
###[ IPv6 Option Header Routing ]###
     nh        = No Next Header
     len       = None
     type      = 4
     segleft   = None
     reserved  = 0
     addresses = [ ::2, ::3 ]
```

Obr. 3.3: Vytvorenie základného IPv6 paketu s (SRH) pomocou argumentov.

Hodnoty polí však môžeme aj samostatne explicitne nastavovať ako je to ukázané vo výpise 3.6, kde je najprv vytvorený paket s dvomi vrstvami a následne priraďujeme jednotlivým poliam hodnoty.

Výpis 3.6: Vytvorenie jednoduchého paketu so smerovacím záhlavím (SRH).

```
1 >>> paket = IPv6()/IPv6ExtHdrSegmentRouting()
2 >>> paket.src = "::1"
3 >>> paket.dst = "::9"
4 >>> paket.type = 4
5 >>> paket.addresses = ["::2","::3"]
6 >>> paket
7 <IPv6  nh=Routing Header src=::1 dst=::9 |<IPv6ExtHdrSegmentRouting
  type=4 addresses=[ ::2, ::3 ] |>>
```

3.2.4 Zoskupenia paketov v Scapy

Väčšinou chceme naraz vytvoriť a poslať viac ako len jeden paket a preto budeme využívať možností Scapy tvoriť zoskupenia paketov pre generovanie ľubovolného počtu paketov.

Množina (set) paketov

Každé pole celého paketu môže byť množina. Toto implicitne definuje množinu paketov, generovaných pomocou karteziánskeho súčinu medzi všetkými poľami ako je ukázané vo výpise 3.7 [5].

Výpis 3.7: Vytvorenie množiny dvoch paketov so smerovacím záhlavím (SRH).

```
1 >>> paket = IPv6(src='::1', dst='::9')/IPv6ExtHdrSegmentRouting(
2 type = [4, 4],
3 addresses=['::2', '::3']
4 )
5 >>> [p for p in paket]
6 [<IPv6  nh=Routing Header src=::1 dst=::9 |<IPv6ExtHdrSegmentRouting
  type=4 addresses=[ ::2, ::3 ] |>>,
7  <IPv6  nh=Routing Header src=::1 dst=::9 |<IPv6ExtHdrSegmentRouting
  type=4 addresses=[ ::2, ::3 ] |>>]
```

Tvorba zoznamu paketov

V Scapy je zoznam paketov (PacketList) objekt, ktorý nám umožňuje vykonávať operácie s množinou paketov. Vo výpise 3.8 je ukázané ako sme použili množinu paketov z výpisu 3.7 a vložili ju do zoznamu paketov [5].

Výpis 3.8: Vytvorenie zoznamu paketov (packet list) s SRH.

```
1 >>> paketList = PacketList([p for p in paket])
2 >>> paketList.summary()
3 ::1 > ::9 (43) / IPv6ExtHdrSegmentRouting
4 ::1 > ::9 (43) / IPv6ExtHdrSegmentRouting
5 >>> [p for p in paketList]
6 [<IPv6  nh=Routing Header src=::1 dst=::9 |<IPv6ExtHdrSegmentRouting
  type=4 addresses=[ ::2, ::3 ] |>>,
7  <IPv6  nh=Routing Header src=::1 dst=::9 |<IPv6ExtHdrSegmentRouting
  type=4 addresses=[ ::2, ::3 ] |>>]
```

3.2.5 Import a export paketov

Aby sme s vygenerovanými paketmi mohli pracovať ďalej budeme ich musieť najprv dať do správneho formátu.

Scapy umožňuje exportovať zoznam paketov (Packet List) do súboru „cap“ alebo „pcap“. Takýto typ súboru sme potom schopný otvoriť v iných programoch ako je napríklad aj *WireShark*.

Scapy umožňuje aj načítať pakety do zoznamu paketov a ďalej ich potom upravovať.

Príkaz „wrpcap“ – Tento príkaz nám umožňuje exportovať zoznam paketov do ľubovolného súboru, väčšinou formát „cap“ alebo „pcap“. Pre naše potreby nám stačia dva argumenty pre tento príkaz. Prvý argument bude meno súboru (respektíve cesta), do ktorého budeme zoznam exportovať a druhý argument bude meno zoznamu paketov vytvoreného v Scapy [5].

Príkaz „rdpcap“ – Tento príkaz nám umožňuje načítať zoznam paketov zo súboru. Jeho použitie je podobné ako pri príkaze „wrpcap“ ale bude nám stačiť len jeden argument, a to bude cesta k súboru, ktorý budeme chcieť importovať [5].

Výpis 3.9: Použitie príkazov wrpcap a rdpcap

```
1 >>> wrpcap('<CESTA_K_SÚBORU>', '<MENO_ZOZNAMU_PAKETOV>')
2 >>> paketList = rdpcap('<CESTA_K_SÚBORU>')
```

Tvorba vlastnej vrstvy

Okrem iného nám Scapy umožňuje aj relatívne jednoduché definovanie nových tried (vrstiev). Vrstva je podtrieda triedy **Packet**. Všetka logika pre manipuláciu s vrstvami je uložená v triede **Packet**. Jednoduchá vrstva je zložená zo zoznamu polí, ktoré budú buď zreťazené pri zostavovaní vrstvy, alebo rozrezané jedno po druhom pri rozoberaní reťazca. Zoznam polí je uložený v atribúte s názvom `fields_desc`. Každé pole je inštanciou triedy poľa:

Výpis 3.10: Príklad vytvorenia novej vrstvy s názvom „Disney“ [5].

```
1     class Disney(Packet):
2     name = "DisneyPacket␣"
3     fields_desc=[ ShortField("mickey",5),
4                   XByteField("minnie",3) ,
5                   IntEnumField("donald" , 1 ,
6                   { 1: "happy", 2: "cool" , 3: "angry" })]
```

V príklade 3.10 má vytvorená vrstva tri polia. Prvé je 2-bytové celočíselné pole s menom `mickey` a predvolenou hodnotou 5. Druhé pole je 1-bytové celočíselné pole s menom `minnie` a predvolenou hodnotou 3. Rozdiel medzi `ByteField` a `XByteField` je o preferencii používateľa aby bolo pole zobrazované v hexadecimálnej forme. Posledné pole je 4-bytové celočíselné pole s menom `donald`. Od obyčajného `IntField` sa líši tým, že niektoré z možných hodnôt poľa majú doslovnú reprezentáciu. Pre príklad, ak je hodnota 3, tak hodnota bude zobrazená ako `angry` [5].

Výpis 3.11: Príklad použitia vlastnej vrstvy vytvorenej vo výpise 3.10 [5].

```
>>> paket = Disney(mickey=1)
>>> ls(paket)
mickey : shortField = 1 (5)
minnie : XByteField = 3 (3)
donald : IntEnumField = 1 (1)
>>> paket.show()
mickey = 1
minnie = 0x3
donald = happy
>>> paket.donald = "cool"
>>> raw(paket)
'\x00\x01\x03\x00\x00\x00\x02'
Disney(_)
<Disney mickey=1 minnie=0x3 donald=cool |>
```

3.2.6 Modifikácia paketov

Scapy obsahuje množstvo funkcií na modifikáciu paketov, vrátane modifikácie hlavičiek a tlv_objects v záhlaví IPv6ExtHdrSegmentRouting, ktoré v našej práci budeme používať pre vloženie vlastných identifikátorov.

Výpis 3.12: Ukážka nastavenia tlv_object v smerovacom záhlaví

```
>>> padN = IPv6ExtHdrSegmentRoutingTLVPadN(
type=4,
len=5,
padding=chosen_id
)
>>> packet[IPv6ExtHdrSegmentRouting].tlv_objects = [padN]
```

V nasledujúcom príklade je ukázané ako pomocou Scapy môžeme meniť doplňujúce TLV pre našu potrebu, pričom „chosen_id“ je nami zvolená hodnota identifikátora. V tomto príklade sa predpokladá že paket s názvom „packet“ už obsahuje SRH(IPv6ExtHdrSegmentRouting).

4 Skript pre manipuláciu a analýzu paketov

V tejto kapitole sa budeme venovať skriptu, ktorý bol vytvorený s cieľom uskutočniť analýzu, tvorbu a modifikáciu paketov a zároveň umožniť používateľom vkladať vlastné identifikátory do týchto paketov. Výsledok sme dosiahli s pomocou knižnice Scapy a jazyku Python.

```
==================================================
                Packet Playground
==================================================
  [C]    Packet creator
  [A]    Add identifier to packet(s)
  [T]    Create a test pcap file
  [a]    Analyze packet(s)
  [S]    Launch Scapy
  [J]    Join PCAP files
  [Q]    Quit the program
==================================================
Enter your choice:
```

Obr. 4.1: Konzolové rozhranie skriptu Packet Playground

4.1 Tvorba vlastných nástrojov pomocou knižnice Scapy

Scapy sa dá použiť aj pre tvorbu vlastných automatizovaných nástrojov pomocou jazyka Python. V našej práci budene používať Scapy na vytvorenie skriptu pre pohodlnejšiu a rýchlejšiu prácu s paketmi. Na začiatok musíme importovať knižnicu Scapy do nášho skriptu a potom môžeme začať vytvárať a upravovať pakety pomocou Scapy. Vieme vytvárať nové pakety pomocou Scapy konštruktorov, ktoré umožňujú definovať jednotlivé časti paketu, ako sú IP adresy, porty, typ protokolu a ďalšie [5].

Výpis 4.1: Ukážka importácie Scapy.

```
import sys
from scapy.all import *
```

Scapy tiež nám umožňuje modifikovať zachytené pakety. Môžeme použiť Scapy na zachytávanie paketov v reálnom čase a analyzovať ich obsah. Vieme si prezrieť všetky časti paketu, ako sú zdrojová a cieľová IP adresa, porty, dĺžka paketu a ďalšie. Ak je to potrebné, môžeme upraviť obsah paketu, napríklad zmeniť IP adresu, port alebo dáta. Scapy taktiež umožňuje aplikovanie filtrov na zachytené pakety, aby sme mohli analyzovať iba určité časti sieťového prenosu.

4.2 Požiadavky na spustenie skriptu

Táto krátka sekcia popisuje požiadavky nutné pre spustenie skriptu.

4.2.1 Python

Ako programovací jazyk k vytvoreniu skriptu sme zvolili jazyk Python. Hlavným dôvodom bola jednoduchá syntax. Široká ponuka knižníc a halvne knižnica Scapy, ktorá je potrebná pre manipulácia paketov.

4.2.2 PiP

Pip je inštalátor balíkov pre Python. Pip je možné použiť na inštaláciu balíkov z „Python Package Index“ a ďalších indexov. V našom skripte ho využívame na ištaláciu potrebných knižníc pre správnu funkciu skriptu[1].

Inštalácia

Pip sa zvyčajne nainštaluje automaticky, ak ste:

- pracujete vo virtuálnom prostredí
- používate Python stiahnutý z python.org
- používate Python, ktorý nebol modifikovaný redistribútorom

Podporované metódy

Ak vaše prostredie Python nemá nainštalovaný pip, existujú 2 spôsoby na inštaláciu pipu podporované priamo správcami pipu:

- ensurepip
- get-pip.py

ensurepip

Python sa dodáva s modulom securepip, ktorý dokáže nainštalovať pip v prostredí Pythonu.

Výpis 4.2: Inštalácia pip pomocou ensurepip.

```
python -m ensurepip --upgrade
```

get-pip.py

Toto je Python skript, ktorý na inštaláciu pip používa určitú logiku bootstrappingu.

- skript je dostunpný na webovej stránke `https://bootstrap.pypa.io/get-pip.py`.
- otvorte terminálový/príkazový riadok, cd do priečinka obsahujúceho súbor get-pip.py a spustite [1]:

Výpis 4.3: Inštalácia pip pomocou get-pip.py.

```
python get-pip.py
```

4.3 Metódy

V tejto sekcii sú popísané využité metódy, ktoré sú v skripte využívané. Jednotlivé definície t.j. kódy metód sa nachádzajú v zozname príloh.

4.3.1 Metóda `install`

Táto metóda slúži na inštaláciu Python balíčkov z repozitárov PyPI.

Argumenty

- `package`: názov balíčka, ktorý sa má nainštalovať.

Použitie

```
install('názov_balíčka')
```

Popis metódy

Metóda používa knižnicu `subprocess` na spustenie príkazu `pip install` v aktuálnom Python prostredí. Ak inštalácia prebehla úspešne, metóda vypíše správu `package installed`. V prípade chyby v inštalácii vypíše správu `package failed to install`.

4.3.2 Metóda `check_packages`

Argumenty

- Nie sú potrebné žiadne argumenty

Použitie

```
check_packages()
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Metóda `check_packages()` slúži na overenie, či sú všetky požadované knižnice uvedené v súbore `requirements.txt` nainštalované v aktuálnom prostredí. Ak niektorá z týchto knižníc chýba, metóda ju nainštaluje pomocou funkcie `install` a vypíše správu o nainštalovaní. Po dokončení inštalácie všetkých knižníc metóda vypíše správu o úspešnom dokončení. Táto metóda nám umožňuje spustiť skript na ľubovolnom prostredí bez potreby inštalácie balíčkov manuálne.

4.3.3 Metóda `export_dict_to_file`

Argumenty

- `data`: Slovník, ktorý sa má uložiť do súboru
- `file_name`: Názov súboru, do ktorého sa má slovník uložiť

Použitie

```
export_dict_to_file(data, file_name)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Metóda `export_dict_to_file` slúži na uloženie slovníka do binárneho súboru s použitím knižnice `pickle`. Na začiatku metódy sa otvorí súbor s názvom `file_name` a do neho sa pomocou funkcie `pickle.dump()` zapíše obsah slovníka `data`. Po úspešnom zapísaní sa vypíše správa o vytvorení súboru s názvom `templates.pickle`. Pomocou tejto metódy je používateľ schopný si uložiť slovník obsahujúci predlohy paketov. Po spustení skriptu budú tieto predlohy automaticky načítané pomocou metódy popísanej v sekcii 4.3.4.

4.3.4 Metóda `import_dict_from_file`

Argumenty

- `file_name`: názov súboru, z ktorého sa má importovať slovník.

Použitie

```
templates = import_dict_from_file('templates.pickle')
```

Návratové hodnoty

- slovník, ktorý bol importovaný zo súboru

Popis metódy

Táto metóda slúži na importovanie slovníka zo súboru. Používa knižnicu `pickle` na deserializáciu dát zo súboru. Ak súbor existuje, tak sa z neho načítajú dáta a funkcia vráti slovník. Ak súbor neexistuje, tak sa vypíše správa o tom, že súbor neexistuje a funkcia vráti `None`.

4.3.5 Metóda `packet_creator(templates)`

Táto metóda vytvára rozhranie pre tvorbu, úpravu a exportovanie paketov. V rámci cyklu sú užívateľovi ponúknuté rôzne možnosti pre prácu s paketmi, ako napríklad vytvorenie nového paketu, úprava existujúceho paketu, výpis predloh a export paketu do súboru. Používateľ môže zadať svoj výber a program potom vykoná príslušnú funkciu.

```
==================================================
                  Packet creator
==================================================
  [1]    Create a packet
  [2]    Modify loaded packet
  [3]    Print templates
  [4]    Load packet from a template
  [5]    Save packet as a template
  [6]    Print loaded packet
  [7]    Export packet to pcap file
  [Q]    Quit the packet creator
==================================================
Enter your choice:
```

Obr. 4.2: Ukážka použitia metódy `packet_creator()`

4.3.6 Metóda `iter`

Metóda `iter()` je pomocná metóda, ktorá slúži na voľbu hodnoty z poľa hodnôt.

Argumenty

- `object`: názov hodnôt v poli ktoré iterujeme (e.g. IP adresy).
- `array`: pole hodnôt z ktorého volíme jednu hodnotu na základe indexu.

Použitie

```
packet_sender = iter("packet sender", ipv6_pool)
```

Návratové hodnoty

- `chosen_value`: hodnota vybraná z poľa.

Popis metódy

Metóda najskôr vypíše zoznam hodnôt z poskytnutého poľa a následne čaká na vstup od používateľa. Po zvolení vráti hodnotu odpovedajúcu zvolenej hodnote z poľa. Ak táto hodnota presahuje rozsah hodnôt v poli, používateľ bude upozornený chybou a následne bude požadovaný nový vstup.

```
Choose a packet sender
1. fd12::
2. fd13::
3. fd23::
4. fd24::
5. fd25::
6. fd34::
7. fd35::
8. fd45::
9. fd46::
10. fd56::
11. fd91::
12. fd92::
Enter your choice (1-12):
```

Obr. 4.3: Príklad použitia metódy `iter` pre voľbu odosielateľa paketu.

4.3.7 Metóda `create_packet`

Metóda `create_packet` slúži k vytvoreniu paketu ako objektu triedy `packet`.

Argumenty

- Nie sú potrebné žiadne argumenty

Použitie

```
p = create_packet()
```

Návratové hodnoty

- `packet`: Vytvorený paket(objekt).

Popis metódy

Táto funkcia vytvára paket (packet) s IPv6 adresami a niekoľkými ďalšími parametrami. Konkrétne si vyžaduje od používateľa názov paketu, počiatočnú a konečnú IPv6 adresu paketu, počet zostávajúcich segmentov paketu, zoznam adries pre každý segment a ID sčítavania paketu. Následne vytvorí a vráti tento paket.

4.3.8 Metóda `browse_templates`

Metóda `browse_templates` slúži k prechádzaniu a voľbe paketu z predlôh. Jej funkcia sa podobá metóde `iter` popísanej v sekcii 4.3.6.

Argumenty

- `browsing`: Argument typu Boolean, ktorý určuje či metóda vráti vybraný paket alebo sa metóda využíva len na prechádzanie predlôh a nie je potreba návratovej hodnoty.

Použitie

```
browse_templates(False,templates)
```

Návratové hodnoty

- `packet`: Vytvorený paket(objekt)(Ak je aktívna t.j. rovná True hodnota „Browsing“).

Popis metódy

Táto funkcia umožňuje používateľovi prehliadať slovník šablón a zobraziť vybranú hodnotu podľa kľúča. Ak v slovníku nie sú žiadne šablóny, vypíše sa hlásenie a funkcia sa ukončí. Používateľovi sa vypíše zoznam kľúčov šablón s priradeným číslom, ktoré používateľ zadá pre výber šablóny. Ak používateľ zadá 'Q', funkcia sa ukončí. Pokiaľ je zadaná neplatná hodnota, funkcia

vypíše chybové hlásenie a čaká na ďalší vstup. Ak bola vybraná platná šablóna, funkcia vypíše jej hodnotu. Ak bola táto funkcia zavolaná s argumentom „browsing“, funkcia vráti hodnotu vybranej šablóny.

```
Available templates:
1. test
2. test_2
3. IDtest
Enter a number to select a template (or 'Q' to quit): 2

Selected template: test_2
==================================================
                      test_2
==================================================
Sender: fd12::
Receiver: fd35::
Routing header type:  4
Segments left:  5
Segment list:  ['fd13::', 'fd23::', 'fd24::', 'fd25::', 'fd34::']
Slice ID:  c000


Load this packet? (Y/N)
```

Obr. 4.4: Príklad použitia metódy `iter` pre voľbu paketu z databázy.

4.3.9 Metóda `load_packet`

Metóda `load_packet` načítanie paketu z databázy. Využíva pritom metódu `browse_templates` s argumentom „browsing“, popísanú v sekcii 4.3.8.

Argumenty

- `packet`: Vybraný paket(objekt).

Použitie

```
packet = browse_templates(True, templates)
```

Návratové hodnoty

- `packet`: Vybraný paket(objekt).

Popis metódy

Táto metóda umožňuje používateľovi vybrať paket zo slovníka šablón a načítať ho. Funkcia najprv zavolá funkciu `browse_templates` s argumentom „True“, aby používateľ mohol vybrať paket zo zoznamu šablón, ako je to popísané v sekcii 4.3.8. Ak bol vybraný platný paket, používateľ bude vyzvaný na potvrdenie, či chce tento paket načítať. Ak používateľ potvrdí výber, funkcia vráti vybraný paket a ukončí sa. Ak používateľ odmietne výber paketu, funkcia sa vráti na začiatok a používateľ môže vybrať ďalší paket. Ak nebola vybraná žiadna platná predloha, cyklus sa ukončí a funkcia sa skončí.

4.3.10 Metóda `save_packet`

Metóda `save_packet` slúži na uloženie načítaného alebo vytvoreného paketu ako predlohu. Tento paket bude následne automaticky uložený medzi predlohy po ukončení skriptu.

Argumenty

- `packet`: Načítaný paket(objekt).

Použitie

```
save_packet(packet)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Táto metóda umožňuje uložiť paket ako šablónu do slovníka predlôh. Ak bola poskytnutá platná hodnota paketu, táto metóda uloží paket pod jeho menom ako kľúč a priradí k nemu hodnotu paketu. V opačnom prípade, ak nebola poskytnutá žiadna platná hodnota paketu, táto metóda vypíše správu o tom, že nie je načítaný žiadny paket.

4.3.11 Metóda `print_packet`

Argumenty

- `packet`: paket(objekt).

Použitie

```
print_packet(packet)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Táto metóda slúži na výpis informácií o pakete, ktorý bol poskytnutý ako vstupný parameter. Ak bola poskytnutá platná hodnota paketu, táto metóda vypíše informácie o pakete, ako sú meno paketu, jeho odosielateľa, prijímateľa, typ routovacieho hlavičky, počet zostávajúcich segmentov, zoznam segmentov a identifikátor slajsov. V opačnom prípade, ak nebola poskytnutá žiadna platná hodnota paketu, táto metóda vypíše správu o tom, že nie je načítaný žiadny paket.

```
==================================================
                      IDtest
==================================================
Sender: fd12::
Receiver: fd35::
Routing header type:  4
Segments left:  5
Segment list:  ['fd13::', 'fd23::', 'fd24::', 'fd25::', 'fd34::']
Slice ID:  a000

Press enter to continue...
```

Obr. 4.5: Príklad použitia metódy `print` na výpis načítaného paketu.

4.3.12 Metóda `export_packet`

Argumenty

- `packet`: paket(objekt).

Použitie

```
export_packet(packet)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Táto metóda slúži na exportovanie paketu do súboru „pcap“. Metóda využíva knižnice scapy pre vytvorenie potrebných záhlaví a následný export do súboru. Metóda je schopná vytvoriť ľubovolné množstvo paketov. V prípade ak by žiadny paket nebol načítaný, tak metóda vypíše chybovú hlášku a ukončí sa. Používateľ je vyzvaný na zadanie množstva paketov, ktoré majú byť exportované do súboru, ako aj na názov výsledného súboru. Názov súboru bude automaticky doplnený príponou „.pcap“ ak ju používateľ nezadal do názvu.

4.3.13 Metóda `join_pcap_files`

Argumenty

- Nie sú potrebné žiadne argumenty

Použitie

```
join_pcap_files()
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Táto metóda slúži na spojenie viacerých súborov príponou „.pcap“ do jedného výstupného súboru. metóda najprv načíta názvy vstupných súborov od používateľa a uloží ich do zoznamu „input_files“. Ak názov vstupného súboru nekončí na '.pcap', metóda automaticky pridá toto rozšírenie. Taktiež skontroluje, či súbor s daným názvom existuje v aktuálnom adresári. Ak sa názov vstupného súboru rovná „done“, cyklus pre vstupné súbory sa ukončí.

Potom používateľ zadá názov výstupného súboru do premennej „output_file“, kde metóda tiež automaticky pridá rozšírenie '.pcap', ak nie je uvedené.

Následne metóda prejde cez každý súbor v zozname „input_files“ a načíta pakety pomocou funkcie „rdpcap“ z knižnice „scapy“. Načítané pakety sa ukladajú do zoznamu „packets“.

Nakoniec metóda zapíše všetky načítané pakety do výstupného súboru pomocou funkcie „wrpcap“.

4.3.14 Metóda `modify_packet`

Argumenty

- `packet`: Načítaný paket(objekt).

Použitie

```
modify_packet(packet)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Táto metóda slúži na modifikáciu načítaného paketu. Metóda využíva metódy triedy `pkt` na modifikáciu parametrov objektu. Metódy triedy `pkt` sú ukázané v prílohe A.

4.3.15 Metóda `add_id`

Argumenty

- Nie sú potrebné žiadne argumenty

Použitie

```
add_id(packet)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Popis metódy

Metóda „add_id()“ slúži na pridanie identifikátora do paketov s protokolom IPv6 v súbore s formátom „.pcap“. Ak vstupný súbor neexistuje, metóda vypíše hlásenie „File does not exist“. Najprv sa metóda spýta používateľa na názov vstupného súboru a skontroluje, či súbor existuje. Ak nie je názov vstupného súboru ukončený príponou „.pcap“, metóda automaticky pridá túto príponu.

Následne metóda načíta pakety zo vstupného súboru pomocou funkcie „rdpcap“. Používateľa sa potom požiada o názov výstupného súboru a ak nie je názov súboru ukončený príponou „.pcap“, metóda ju automaticky pridá.

Používateľa potom požiada o výber ID zo zoznamu možností, ktorý sa uloží do premennej „chosen_id“. Následne sa vytvorí nový „IPv6ExtHdrSegmentRoutingTLVPadN“ objekt s používateľom zvoleným ID, ktorý sa uloží do premennej „padN“.

Potom metóda prejde cez každý paket v súbore a skontroluje, či má paket IPv6 záhlavie. Ak áno, skontroluje, či má paket rozširujúcu hlavičku „IPv6ExtHdrSegmentRouting“. Ak áno, paket sa upraví s používateľom zvoleným ID a zapíše sa do výstupného súboru pomocou funkcie „wrpcap“ z knižnice „Scapy“. Ak paket nemá IPv6 protokol alebo rozširujúcu hlavičku „IPv6ExtHdrSegmentRouting“, metóda prejde k nasledujúcemu paketu.

4.3.16 Metóda `generate_random_ipv6_pcap`

Argumenty

- `packet_count`: požadovaný počet paketov.
- `output_filename`: názov výstupného súboru.

Použitie

```
generate_random_ipv6_pcap(packet_count, output_filename)
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Táto metóda slúži na generovanie náhodných IPv6 paketov. Argumenty sú počet paketov, ktoré chce používateľ vygenerovať. Súťasťou týchto pakeov sú aj slice identifikátory definované v poli „slice_ids“. Paket sa skladá zo záhlaví Ethernet, IPv6, SRv6, TCP a náhodné dáta. Na konci funkcie je paket vyexportovaný pomocou funkcie „wrpcap“.

4.3.17 Metóda `analyze`

Argumenty

- Nie sú potrebné žiadne argumenty.

Použitie

```
analyze()
```

Návratové hodnoty

- Metóda nemá návratovú hodnotu.

Táto metóda slúži na analýzu paketov v súbore s príponou ".pcap".

Najskôr sa používateľa vyzve na zadanie názvu vstupného súboru pcap. Ak názov súboru neobsahuje príponu ".pcap", automaticky sa pridá. Ak súbor existuje, pakety sa prečítajú pomocou funkcie „rdpcap“ a uložia sa do zoznamu „packet_list“.

Potom sa používateľovi zobrazí menu, v ktorom sú vypísané informácie o paketoch v zozname „packet_list“ a používateľ sa môže rozhodnúť, ktorý paket chce zobraziť podrobnejšie. Po zadaní čísla indexu paketu sa overí, či ide o platné celé číslo a či sa nachádza v rozmedzí indexov „packet_list“. Ak ide o platné číslo, zvolený paket sa zobrazí na obrazovke a používateľ sa môže rozhodnúť, či pokračovať v analýze alebo ukončiť program.

```
==================================================
                 Packet analysis
==================================================
[0] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[1] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[2] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[3] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[4] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
Enter the index of the packet you want to view (Q to quit): Q

Press enter to continue...
```

Obr. 4.6: Príkklad použitia metódy analyze

4.3.18 Metóda `launch_scapy_console()`

Táto metóda slúži len na spustenie Scapy konzole. Môžeme tak pomocou nej zadávať príkazy z knižnice Scapy bez toho, aby sme museli pridávať samostatné metódy do skriptu alebo skript uknončovať.

```
==================================================
                Packet Playground
==================================================
  [A]    Add identifier to packet(s)
  [T]    Create a test pcap file
  [a]    Analyze packet(s)
  [S]    Launch Scapy
  [J]    Join PCAP files
  [Q]    Quit the program
==================================================
Enter your choice: S
Scapy >>> ls(IPv6)
version    : BitField  (4 bits)                = ('6')
tc         : BitField  (8 bits)                = ('0')
fl         : BitField  (20 bits)               = ('0')
plen       : ShortField                        = ('None')
nh         : ByteEnumField                     = ('59')
hlim       : ByteField                         = ('64')
src        : SourceIP6Field                    = ('None')
dst        : DestIP6Field                      = ('None')
Scapy >>>
```

Obr. 4.7: Príkklad použitia metódy launch scapy console.

4.4 Packet Playground

Táto sekcia popisuje samotnú funkcionalitu skriptu pre analýzu, tvorbu a modifikáciu paketov. Užívateľské rozhranie je typu konzolová aplikácia.

4.4.1 Funkcie

Pridávanie identifikátoru paketom

Táto funkcia slúži na pridávanie identifikátoru paketom v súbore „pcap“. Používateľ musí najskôr zadať názov vstupného súboru, ktorý obsahuje pakety, ktorým chce priradiť identifikátory a potom názov výstupného súboru, do ktorého sa zapíšu nové pakety s identifikátormi. V poslednom kroku používateľ vyberie z piatich dostupných identifikátorov a voľbu potvrdí tlačidlom enter. Vstupný súbor sa musí nachádzať v rovnakom adresári ako skript. Výstupný súbor bude vytvorený takisto v rovnakom adresári v akom sa nachádza skript.

Generátor testovacích „pcap“ súborov

Táto funkcia slúži na vytvorenie testovacieho súboru „pcap“. Súbor obsahuje pakety s náhodnými dátami a identifikátormi. Súbor sa vytvorí v adresári v ktorom sa nachádza skript. Používateľ musí zadať aký počet paketov si praje vytvoriť a voľbu potvrdiť tlačidlom enter.

Analýza paketov

Táto funkcia slúži na analýzu paketov. Používateľ zadá názov súboru „pcap“, ktorý si praje analyzovať a následne mu bude vypísaný obsah súboru spolu s indexmi paketov, tak ako je to zobrazené na obrázku 4.8. Používateľ si potom môže zvoliť jeden z paketov, ktorý si praje analyzovať a svoju voľbu potvrdí tlačidlom enter. Používateľovi sú nasledovne zobrazené podrobnosti o vybranom pakete. Cyklus pre analýzu paketov sa končí až keď používateľ zadá „Q“.

Scapy konzola

Táto funkcia slúži pre interpretovanie príkazov Scapy za behu skriptu. Používateľ môže zadávať ľubovolné príkazy z knižnice Scapy. Pre ukončenie tejto funkcie je potrebné zadať slovo „exit“.

```
==================================================
                 Packet analysis
==================================================
[0] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[1] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[2] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[3] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
[4] Ether / IPv6 / IPv6ExtHdrSegmentRouting / TCP ftp_data > http S / Raw
Enter the index of the packet you want to view (Q to quit): Q

Press enter to continue...
```

Obr. 4.8: Analýza paketov.

Zlučovanie „pcap“ súborov

Táto funkcia umožňuje používateľovi zlúčiť „pcap“ súbory. Používateľ v cykle zadáva názvy súborov na zlúčenie a cyklus ukončuje slovom „done“. Vstupné súbory sa musia nachádzať v rovnakom adresári ako skript. Výstupný súbor bude vytvorený takisto v rovnakom adresári v akom sa nachádza skript.

4.4.2 Tvorba paketov pomocou funkcie `Packet creator`

Skript obsahuje časť pre tvorbu vlastného paketu ako je to popísané v sekcii 4.3.5. Používateľ má na výber z 8 možností:

- Vytvoriť paket
- Modifikovať načítaný paket
- Vypísať predlohy
- Načítať paket z predlôh
- Uložiť načítaný paket ako predlohu
- Vypísať načítaný paket
- Exportovať paket do súboru „pcap“
- Ukončiť „Packet Creator“

Používateľ volí funkcionalitu pomocou zadania čísla do konzoly.

Vytvorenie paketu

Pri tvorbe paketu musí používateľ najskôr zadať meno paketu. Následne sa mu zobrazí rozsah IPv6 adries dostupných v topológii pre ktorú sú pakety určené, rozsah je dostupný v tabuľke 5.1. Z tohoto rozsahu najprv volí adresu odosielateľa a potom adresu príjemcu.

Po výbere odosielateľa a príjemcu bude od používateľa požadovaný počet zostávajúcich segmentov, t.j. počet zostávajúcich medziuzlov, ktorými bude musieť paket

```
Choose a packet sender
1. fd12::
2. fd13::
3. fd23::
4. fd24::
5. fd25::
6. fd34::
7. fd35::
8. fd45::
9. fd46::
10. fd56::
11. fd91::
12. fd92::
Enter your choice (1-12):
```

Obr. 4.9: Výber odosielateľa

prejsť. Po zadaní počtu bude znovu požadované vybrať adresy príslušných uzlov ktorými má paket prejsť, tak ako tomu bolo u výberu odosielateľa a príjemcu.

V poslednej časti má používateľ na výber z piatich splice identifikátorov, ktoré symbolizujú jednotlivé cesty v topológii.

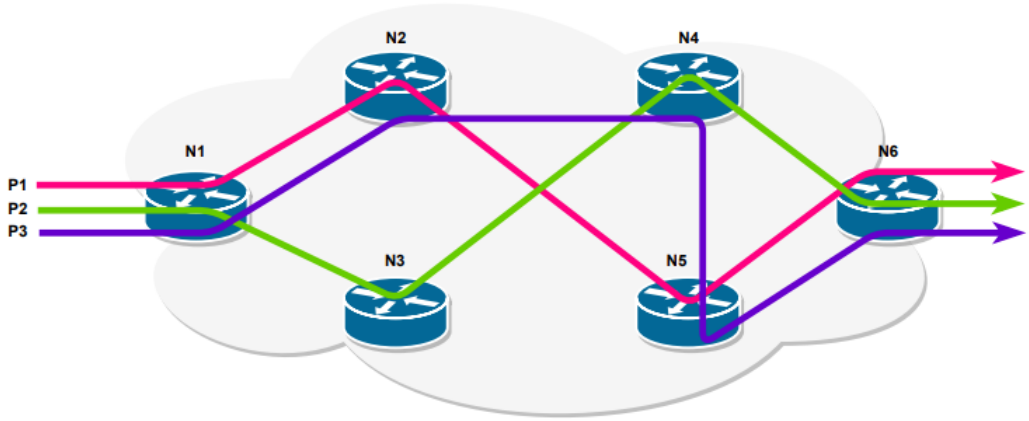
```
Choose a slice ID
1. a000
2. b000
3. c000
4. e000
5. d000
Enter your choice (1-5): 1
Packet created!
```

Obr. 4.10: Výber identifikátora paketu.

Na obrázku 4.11 sú zobrazené tri z piatich dostupných splicov, kde:

- a000 – Ružová,
- b000 – zelená,
- c000 – fialová

Po vytvorení paketu nasleduje výpis parametrov paketu ako je to na obrázku 4.12. Následne sa po stlačení tlačidla enter používateľ dostane naspäť



Obr. 4.11: SR politiky pre IPv6 dátové prevádzky na N1 hraničnom uzle[16].

```
==================================================
                        test
==================================================
Sender: fd12::
Receiver: fd23::
Routing header type:  4
Segments left:  2
Segment list:  ['fd13::', 'fd24::']
Slice ID:  a000

Press enter to continue...
```

Obr. 4.12: Výpis parametrov paketu.

Modifikácia načítaného paketu

Pri modifikácii paketu sa používateľovi najskôr zobrazí menu v ktorom si musí zvoliť akú vlastnosť paketu si praje modifikovať, tak ako je ukázané na obrázku 4.13. Dostupné možnosti pre modifikáciu paketu sú:

- Meno
- Odosielateľ
- Príjemca
- Identifikátor
- Zoznam segmentov

Výpis uložených predlôh

Pri výpise predlôh paketov, ktoré boli v minulosti vytvorené je používateľovi vypísaný zoznam predlôh, z ktorého si následovne môže vybrať, ktorú predlohu si praje zobraziť. Po zadaní výberu sa používateľovi vypíšu vlastnosti predlohy, tak ako je

```
==================================================
              Modify Packet Attribute
==================================================
  [1]    Name
  [2]    Sender
  [3]    Receiver
  [4]    Identifier
  [5]    Segment List
==================================================
Enter your choice:
```

Obr. 4.13: Menu pre modifikáciu paketu.

to ukázané na obrázku 4.14. Cyklus pre výpis predlôh sa končí až keď používateľ zadá „Q“.

```
Available templates:
1. test
2. test_2
3. IDtest
Enter a number to select a template (or 'Q' to quit): 1

Selected template: test
==================================================
                        test
==================================================
Sender: fd12::
Receiver: fd23::
Routing header type:  4
Segments left:  5
Segment list:  ['fd12::', 'fd13::', 'fd23::', 'fd24::', 'fd25::']
Slice ID:  d000



Press enter to continue...
```

Obr. 4.14: Ukážka výpisu predlohy.

Načítanie paketu z uložených predlôh

Pri načítaní paketu z predlôh je používateľovi vypísaný zoznam predlôh, z ktorého si následovne môže vybrať, ktorú predlohu si praje načítať. Po zadaní výberu sa používateľovi vypíšu vlastnosti predlohy spolu s otázkou či si danú predlohu praje načítať. Ak by si používateľ vybral že si zvolenú predlohu nepraje načítať, tak mu bude opäť vypísaný zoznam predlôh. Cyklus pre výpis predlôh sa končí až keď používateľ zadá „Q“ alebo načíta jednu z predlôh.

```
Available templates:
1. test
2. test_2
3. IDtest
Enter a number to select a template (or 'Q' to quit): 1

Selected template: test
==================================================
                        test
==================================================
Sender: fd12::
Receiver: fd23::
Routing header type:  4
Segments left:  5
Segment list:  ['fd12::', 'fd13::', 'fd23::', 'fd24::', 'fd25::']
Slice ID:  d000


Load this packet? (Y/N)Y
Packet loaded

Press enter to continue...
```

Obr. 4.15: Ukážka načítania paketu z uložených predlôh.

Uloženie načítaného paketu do predlôh

Pre uloženie paketu nie je potrebný žiadny zásah od používateľa okrem voľby uloženia načítaného paketu. Ak sa používateľ pokúsi uložiť predlhou bez načítaného paketu tak skript vyhodí chybu.

Výpis načítaného paketu

Táto funkcia slúži na zobrazenie načítaného paketu. Príklad je uvedený na obrázku 4.16.

```
==================================================
                        test_2
==================================================
Sender: fd12::
Receiver: fd35::
Routing header type:  4
Segments left:  5
Segment list:  ['fd13::', 'fd23::', 'fd24::', 'fd25::', 'fd34::']
Slice ID:  c000

Press enter to continue...
```

Obr. 4.16: Ukážka výpisu načítaného paketu.

Export paketov do súboru „pcap“

Posledná funkcia časti „Packet creator“ je export paketu alebo paketov do súboru „pcap“. Používateľ najprv zadá koľko paketov si praje vytvoriť a následne zadá

názov výstupného súboru. Výstupný súbor bude potom vyexportovaný do adresára v ktorom sa skript nachádza.

5 Program Trex

Táto kapitola sa popisuje čo je to generátor sieťovej prevádzky a následne opisuje program TRex. Pomocou programu TRex budeme testovať nami vytvorené pakety pomocou programu Scapy v predom určenej topológii, ktorá je na obr. 5.3.

5.1 Generátor sieťovej prevádzky

Smerovače sa tradične testovali pomocou komerčných generátorov prenosu, zatiaľ čo výkon sa zvyčajne meral pomocou metrík paketov za sekundu (PPS). Keďže funkčnosť a služby smerovača sa stávajú zložitejšími, generátory stavovej prevádzky sa stali nevyhnutnými na poskytovanie realistickejších scenárov prevádzky.

Výhody realistických generátorov prevádzky:

- Presné metriky výkonu.
- Objavovanie prekážok v realistických dopravných scenároch[2].

5.2 Základná charakteristika programu Trex

TRex je vysoko výkonný open-source softwarový generátor reálnej sieťovej prevádzky, vyvíjaný spoločnosťou Cisco Systems.

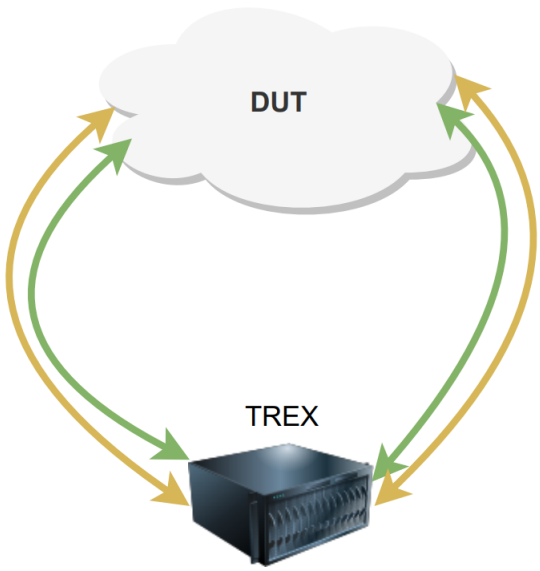
Hlavné funkcie a parametre programu:

- podpora DPDK rozhraní
- výkonnosť až do 200Gb/s na špecializovanej UCS Cisco platforme
- stavový aj bezstavový režim
- emulácia sieťovej prevádzky zo strany klienta ako aj servera
- rozsiahle štatistiky o generovanej prevádzke
- interaktívny mód sledovania aktuálneho generovania
- možnosť spustenia aj vo virtualizovanom prostredí

Generovanie a simulácia reálnej sieťovej prevádzky sa vykonáva na základe šablón. Šablóny je možné definovať pomocou konfiguračných súborov YAML s príslušnými PCAP súbormi. Profily je možné definovať aj prostredníctvom vytvoreného aplikačného rozhrania v jazyku Python [16].

Obrázok 5.1 je znázorňuje zapojenie TRex generátoru a zariadenia označeného *Device Under Test* (DUT). DUT môže označovať samostatné zariadenie alebo celú sieť.

Po ukončení generovania sieťovej prevádzky alebo aj počas generovania je možné sledovať štatistiky o prenesenom množstve bajtov, paketov, CPU utilizácii, počte chýb a ďalšie informácie.



Obr. 5.1: Zapojenie TRex generátoru [16]

Bezstavový mód

V bezstavovom móde program TRex podporuje testovanie nad L2/L3 vrstvou OSI modelu. Tento mód umožňuje definovanie sieťového toku na základe jedného paketu a následne celej šablóny, profilu sieťovej prevádzky. Profil sieťovej prevádzky sa definuje v jazyku Python s možnosťou zostavenia ľubovolného paketu s využitím nástroja Scapy, opísaného v kapitole 3.

Ako už bolo spomenuté TRex umožňuje aj vytvorenie profilu sieťovej prevádzky na základe už zachytených paketov a teda z PCAP súborov.

Možnosti vytvorenia profilu pomocou PCAP:

- **Local PCAP push** – Pri tejto metóde sa PCAP súbor lokálne načíta Python klientom a je následne transformovaný do zoznamu tokov. Odoslanie jedného toku spúšťa odosielanie ďalšieho. Každý tok obsahuje iba jeden paket zo súboru. Aj z tohto dôvodu je veľkosť PCAP súboru pri tejto variante obmedzený na veľkosť 1,5MB.
- **Server-based push** – Táto metóda už umožňuje použitie PCAP súborov neobmedzenej veľkosti. Jednotlivé pakety z PCAP súboru sa postupne replikujú a to bez možnosti ich úpravy. Úpravu paketov je v tomto prípade nutné vykonať priamo pre daný PCAP súbor.

Stavový mód

Trex v stavovom móde využíva ku generovaniu sieťovej premávky PCAP súbory, ktoré obsahujú TCP alebo UDP toky. Tieto toky následne replikuje s novými IP adresami a portami pre klientskú aj serverovú stranu komunikácie. Profil sieťovej prevádzky v tomto móde definujeme YAML súborom. Pri spúšťaní TRex generátoru sa špecifikuje cez argument aj tento konfiguračný YAML súbor. V ňom je možné okrem iného nájsť rozsah zdrojových a cieľových adries, ktoré sa budú vkladať do jednotlivých paketov. Taktiež sa tu nachádza umiestnenie PCAP súboru, avšak je možné špecifikovať aj väčšie množstvo PCAP súborov a tým vytvoriť reálnejšiu sieťovú prevádzku pozostávajúcu z paketov prenášajúcich dáta rôznych služieb súbežne.

Pre naše účely budeme používať bezstavový mód generátoru TRex, kvôli jednoduchosti vytvárania profilov sieťovej prevádzky pomocou Scapy. Výhodou je aj interaktívny mód TRex konzoly zobrazujúci štatistiky o odoslaných a prijatých paketoch v reálnom čase [16].

5.2.1 Vytvorenie konfiguračného súboru

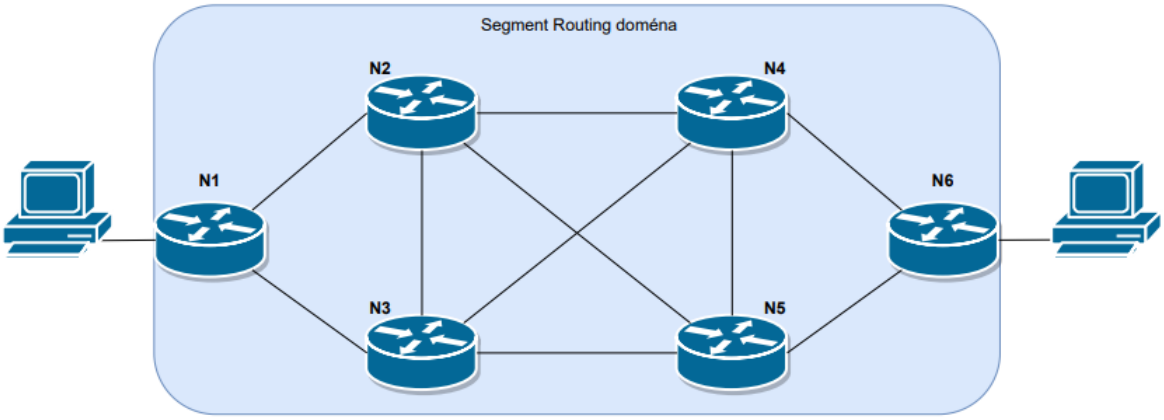
Výpis 5.1: TRex profil sieťovej premávky v bezstavovom móde.

```
def get_streams (self,
                 ipg_usec = 10.0,
                 loop_count = 1):

    profile = STLProfile.load_pcap(self.pcap_file,
                                   ipg_usec = ipg_usec,
                                   loop_count = loop_count)
```

5.3 Topológia pre implementáciu paketov

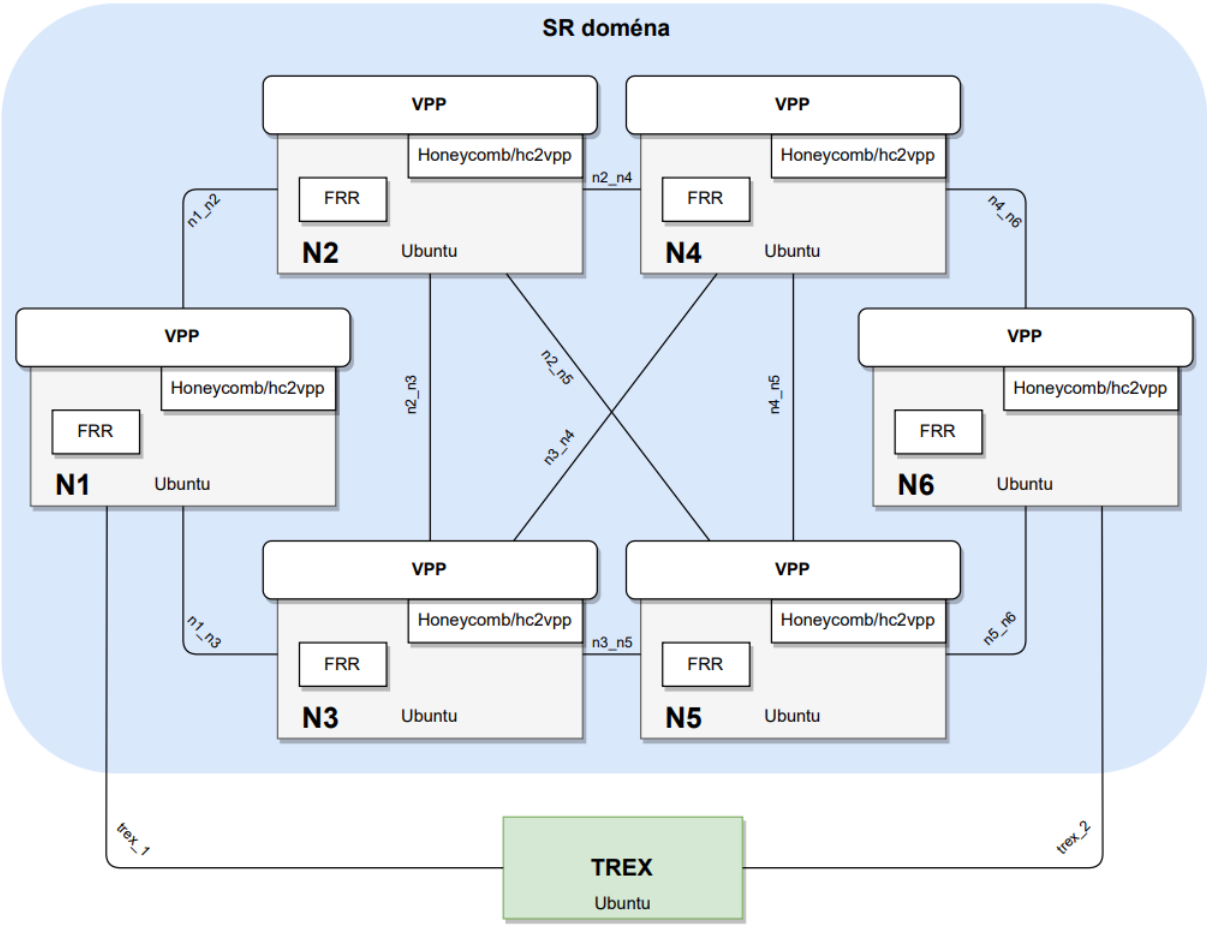
Pomocou možností popísaných v podkapitole 4.1 budeme tvoriť pakety s vlastnými identifikátormi a údajmi potrebnými pre implementáciu paketov do topológie zobrazenej na obrázku 5.2, ktorú navrhol Ing. Igor Mjasojedov pre implementáciu SRv6 v sieti. Detailnejší popis vytvorenia a fungovania topológie je dostupný v diplomovej práci v kapitole **3** a **4** [16].



Obr. 5.2: Topológia pre implementáciu paketov vybraná z diplomovej práce ktorú spracoval Igor Mjasojedov [16].

Tab. 5.1: Zoznam sietí medzi kontajnermi s ich adresovaním [16].

Názov siete	IPv4	IPv6
node1_node2	192.168.12.0/24	fd12::/64
node1_node3	192.168.13.0/24	fd13::/64
node2_node3	192.168.23.0/24	fd23::/64
node2_node4	192.168.24.0/24	fd24::/64
node2_node5	192.168.25.0/24	fd25::/64
node3_node4	192.168.34.0/24	fd34::/64
node3_node5	192.168.35.0/24	fd35::/64
node4_node5	192.168.45.0/24	fd45::/64
node4_node6	192.168.46.0/24	fd46::/64
node5_node6	192.168.56.0/24	fd56::/64
trex-1	192.168.91.0/24	fd91::/64
trex-2	192.168.92.0/24	fd92::/64



Obr. 5.3: Topológia pre implementáciu paketov vybraná z diplomovej práce ktorú spracoval Igor Mjasojedov [16].

Záver

Cieľom tejto práce, bolo analyzovať *Internet protokol verzie 6* (IPv6), SRv6, program Scapy a popísať proces tvorby paketov v Scapy. Následne tvorba prvých paketov v programe pre implementáciu do programu Trex. Pre tieto účely bol v rámci práce vytvorený aj skript pre tvorbu, manipuláciu a analýzu paketov.

V prvej kapitole je analýza IPv6. V kapitole sú popísané základné vlastnosti protokolu IPv6 spolu s dôvodom vzniku a porovnaním s jeho predchodcom IPv4, rozbor IPv6 paketu, popis jednotlivých polí IPv6 hlavičky, druhy adresovania, popis jednotlivých rozširujúcich záhlaví. Rozširujúce záhlavie typu 4, Smerovacie Záhlavie je nutnou súčasťou pre fungovanie SRv6.

V druhej kapitole práce je rozoberané samotné *Segmentové smerovanie skrz IPv6* (SRv6), *Segmentová smerovacia hlavička* (SRH) a proces smerovania paketov v sieti, ktorá je pracuje SRv6. Zavedenie segmentového smerovania v IPv6 značne zjednodušuje smerovanie a otvára široké možnosti riadenia premávky v sieti. Výhodou je aj spätná kompatibilita, vďaka ktorej je možné nasadiť SRv6 v podstate v každej IPv6 sieti. Pre naše účely sme určili že pre prenos vlastných informácii v tomto záhlaví, ako napríklad identifikátorov bude vhodné záhlavie PadN, ktoré je v tejto kapitole popísané a je aj ďalej využívané v skripte pre tvorbu paketov.

V tretej kapitole je rozoberaný program Scapy, jeho schopnosť vizualizácie a dekódovania dát a následne proces tvorby paketov. Pomocou Scapy sme schopný veľmi rýchlo tvoriť množstvo paketov s vlastnými parametrami pre použitie v generátore premávky. V kapitole sú popísané vrstvy (triedy), ktoré sa používajú pre tvorbu paketu, tvorba množín paketov pre jednoduchú tvorbu viacerých paketov naraz. V kapitole je ďalej popísaná schopnosť Scapy exportovať pakety do súboru „cap“ alebo „pcap“. Scapy umožňuje aj tvorbu vlastných nástrojov pomocou importu Scapy do vlastného skriptu. Vďaka tejto vlastnosti sme boli schopný vytvoriť skript pre pohodlnejšiu a rýchlejšiu tvorbu,analýzu paketov a ich export.

Štvrtá kapitola sa zaoberá samotným skriptom pre tvorbu, manipuláciu a analýzu paketov vytvoreným prostredníctvom jazyka Python a knižnice Scapy. Tento skript umožňuje mnohé funkcie pre prácu s paketmi a na jeho spustenie používateľ potrebuje mať nainštalovaný len Python. Skript disponuje zoznamom požiadavkov na spustenie a sám si ich pomocou modulu PiP nainštaluje. V týchto požiadavkoch je zahrnutá aj knižnica Scapy. V kapitole sú ďalej popísané jednotlivé metódy, ktoré sa v skripte využívajú, ako aj samotná funkcionalita skriptu a požiadavky na spustenie. Súčasťou skriptu je aj tzv. „Packet Creator“, ktorý umožňuje tvorbu vlastných paketov.

Piata kapitola sa zaoberá generátorom sieťovej prevádzky Trex. V kapitole základná charakteristika programu ako aj popis stavového a bezstavového módu prog-

ramu. Súčasťou tejto kapitoly je potom popis možností vytvorenia profilu pomocou súborov „pcap“, ktoré dokáže vytvárať skript spomenutý v predošlej časti. Súčasťou kapitoly je aj topológia a adresovacia tabuľka pre samotnú implementáciu vytvorených paketov v sieti. Detailnejší popis topológie a jej funkcionality je potom popísaný v diplomovej práci, ktorú vypracoval Igor Mjasojedov[16].

Pri implementácii paketov sme narazili na otvorený problém, kde identifikátory, ktoré sme pomocou skriptu paketom priradili sú spolu s celým SRH prepisované. Krajné riešenie v tejto situácii by bolo vložiť identifikátory do payloadu, nakoľko tam nedochádza k prepisu. Lepším riešením by bolo použiť hlavičku možností destinácie v hlavičke IPv6.

Literatúra

[1] *Pip documentation v23.1.2* [https://pip.pypa.io/en/stable/].

[2] *TRex*. Cisco, Jan 2023. Dostupné z: `https://trex-tgn.cisco.com/trex/doc/trex_manual.html`.

[3] Abley, J., Savola, P. a Neville Neil, G. *Deprecation of type 0 routing headers in IPv6*. 2007.

[4] Bashandy, A., Filsfils, C., Previdi, S., Decraene, B., Litkowski, S. et al. Segment Routing with MPLS data plane. *RFC 8660*. 2019.

[5] Biondi, P. Scapy documentation (!). *Vol.* 1. vyd. 2010, zv. 469, s. 155–203.

[6] Deering, S. a Hinden, R. *Internet protocol, version 6 (IPv6) specification*. 2017.

[7] EBALARD, P. B. A. IPv6 Routing Header Security. *Innovation Works*. 2007.

[8] Filsfils, C., Camarillo, P., Leddy, J., Voyer, D., Matsushima, S. et al. SRv6 network programming. *Internet-Draft*. 2017.

[9] Filsfils, C., Camarillo, P., Leddy, J., Voyer, D., Matsushima, S. et al. Segment routing over IPv6 (SRv6) network programming. *RFC 8986*. 2021.

[10] Filsfils, C., Camarillo, P., Leddy, J., Voyer, D., Matsushima, S. et al. *Segment Routing over IPv6 (SRv6) Network Programming* [RFC 8986]. RFC Editor, február 2021. DOI: 10.17487/RFC8986. Dostupné z: `https://www.rfc-editor.org/info/rfc8986`.

[11] Filsfils, C., Dukes, D., Previdi, S., Leddy, J., Matsushima, S. et al. IPv6 segment routing header (SRH). *IETF InternetDraft, available: https://tools.ietf. org/html/draft-ietf-6man-segment-routing-header-18*. 2019.

[12] Filsfils, C., Dukes, D., Previdi, S., Leddy, J., Matsushima, S. et al. *IPv6 Segment Routing Header (SRH)* [RFC 8754]. RFC Editor, marec 2020. DOI: 10.17487/RFC8754. Dostupné z: `https://www.rfc-editor.org/info/rfc8754`.

[13] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S. et al. *Segment Routing Architecture* [RFC 8402]. RFC Editor, júl 2018. DOI: 10.17487/RFC8402. Dostupné z: `https://www.rfc-editor.org/info/rfc8402`.

[14] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S. et al. *Segment routing architecture*. 2018.

[15] Jeřábek, J. *Komunikační technologie*. 2021. vyd. Brno: Vysoké učení technické v Brně, 2021. 174 s. ISBN 978-80-214-4713-4.

[16] Mjasojedov, I. *Implementace mechanismů zajišťujících Segment Routing for IPv6 (SRv6) pomocí platformy FD.io VPP*. 2022. Dizertačná práca. Brno University of Technology.

[17] Stallings, W. IPv6: the new Internet protocol. *IEEE Communications Magazine*. 1996, zv. 34, č. 7, s. 96–108. DOI: 10.1109/35.526895.

[18] Strayer, W., Jones, C., Tchakountio, F. a Hain, R. SPIE-IPv6: single IPv6 packet traceback. In: *29th Annual IEEE International Conference on Local Computer Networks*. 2004, s. 118–125. DOI: 10.1109/LCN.2004.118.

Zoznam symbolov a skratiek

DiffServ Differentiated Servieces

TLV Type length value

IGP Interior gateway protocol

RIP Router Information Protocol

BGP Border Gateway protocol

EH Rozširujúce záhlavie

DOH Hlavička možností destinácie

DUT Device Under Test

PyPI Python Package Index

IPv6 Internet protokol verzie 6

IPv4 Internet protokol verzie 4

IP Internet Protokol

AH Autentizačné záhlavie

DA Adresa príjemcu

SA Adresa odosielatela

TCP Transmission Control Protocol

UDP User Datagram Protocol

ICMP Internet Control Message Protocol

IPsec Internet Protocol secutiry

ESP Encapsulating security Payload

SRH Segmentová smerovacia hlavička

SR Segmentové smerovanie

SID Segment Identifikátor

SRv6 Segmentové smerovanie skrz IPv6

IANA	Internet Assigned Numbers Authority
DARPA	Defense Advanced Research Projects Agency
MIPv6	Mobile IPv6
RH	Smerovacie Záhlavie
RPL	Routing Protocol for Low-Power and Lossy Networks
DSL	Domain Specific Language
TTL	Doba života
MTU	Maximálna prenosová jednotka
LSSR	Loose Source and Record Route
IETF	Internet Engineering Task Force
NAT	Network address translation
ICMPv6	Internet control message protocol verzie 6

Zoznam príloh

A Program pre tvorbu paketov **93**
A.1 Trieda pkt časť 1. 93
A.2 Trieda pkt časť 2. 94

A Program pre tvorbu paketov

Príloha obsahuje len triedu pkt, ktorá slúži na tvorbu paketov nakoľko samotný skript pre tvorbu paketov je príliš velký.

A.1 Trieda pkt časť 1.

Trieda paket, ktorá slúži pre tvorenie objektov, pre budúcu implementáciu funkcie predvoľby paketov.

```
class pkt:
    def __init__(self,
    name,
    sender,
    receiver,
    type,
    segmentsLeft,
    segmentList,
    identifier):
        self.__name = name
        self.__sender = sender
        self.__receiver = receiver
        self.__type = type
        self.__segmentsLeft = segmentsLeft
        self.__segmentList = segmentList
        self.__identifier = identifier

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    def get_sender(self):
        return self.__sender

    def set_sender(self, sender):
        self.__sender = sender

    def get_receiver(self):
        return self.__receiver
```

A.2 Trieda pkt časť 2.

```
def set_receiver(self, receiver):
        self.__receiver = receiver

    def get_type(self):
        return self.__type

    def set_type(self, type):
        self.__type = type

    def get_segmentsLeft(self):
        return self.__segmentsLeft

    def set_segmentsLeft(self, segmentsLeft):
        self.__segmentsLeft = segmentsLeft

    def get_segmentList(self):
        return self.__segmentList

    def set_segmentList(self, segmentList):
        self.__segmentList = segmentList

    def get_identifier(self):
        return self.__identifier

    def set_segmentList(self, identifier):
        self.__identifier = identifier
```