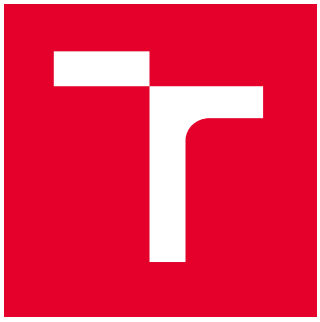


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

IMPLEMENTACE MECHANISMŮ ZAJIŠŤUJÍCÍCH SEGMENT ROUTING FOR IPV6 (SRV6) POMOCÍ PLATFORMY FD.IO VPP

IMPLEMENTATION OF MECHANISMS PROVIDING SEGMENT ROUTING FOR IPV6 (SRV6) USING THE FD.IO VPP PLATFORM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Igor Mjasojedov**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Aneta Koláčková**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Igor Mjasojedov ***ID:*** 204378

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Implementace mechanismů zajišťujících Segment Routing for IPv6 (SRv6) pomocí platformy FD.io VPP

POKYNY PRO VYPRACOVÁNÍ:

Diplomová práce se bude zabývat implementací technologie Segment Routing for IPv6 (SRv6) na platformě FD.io VPP s důrazem na použití v rámci mobilních sítí 5G NR (New Radio). Teoretická část práce bude obsahovat detailní popis technologie SRv6 a také rozbor platformy FD.io. Praktická část bude zahrnovat vytvoření komunikačního scénáře odpovídající transportní části mobilních sítí 5G NR pomocí platformy FD.io. Ve vytvořeném scénáři budou ověřeny dostupné techniky SRv6 a také koncept síťového programování SRv6. Výstupy práce budou přehledně prezentovány a diskutovány.

DOPORUČENÁ LITERATURA:

[1] FD.io, The World's Secure Networking Data Plane Accessed: Sep. 3, 2021. [Online]. Dostupné z: https://fd.io/.

[2] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: a comprehensive survey of research activities, standardization efforts and implementation results," IEEE Communications Surveys Tutorials, pp. 1–1, 2020.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 24.5.2022

Vedoucí práce: Ing. Aneta Koláčková

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

ABSTRAKT

Rozvíjajúce sa mobilné siete piatej generácie (5G) kladú dôraz na zabezpečenie stále vyšších nárokov dynamických služieb. Zvyšujúce sa požiadavky prenášaných služieb na rýchlosť, spoľahlivosť a flexibilitu prenosu sa musia zabezpečiť vývojom technológii na všetkých úrovniach počítačových sietí. Táto diplomová práca sa zameriava na transportnú časť mobilných 5G sietí a predstavenie konceptu Segment Routing (SR) pre jej zefektívnenie. SR umožňuje integráciu sieťového programovania do tradičného dizajnu sietí. SR je možné implementovať nad protokolom MPLS a IPv6. Inštancia implementovaná nad IPv6 sa nazýva Segment Routing over IPv6. Práve ňou sa zaoberá táto práca z dôvodu jej možnosti rozširovania a ďalších výhod vyplývajúcich z natívnych vlastností protokolu IPv6. Cieľom práce je vytvoriť komunikačný scenár v ktorom je možné simulovať nasadenie protokolu SRv6 v transportnej časti 5G siete. V tejto práci sa využíva aj platforma Docker, ktorá slúži k vytvoreniu jednotlivých uzlov siete. Využitou platformou k zavedeniu protokolu SRv6 do topológie je platforma FD.io VPP. Platformu VPP je možné konfigurovať cez protokol NETCONF vďaka Honeycomb agentu, ktorý umožňuje spracovávanie NETCONF správ. Konfigurácia VPP cez tento protokol je v tejto práci zabezpečená prostredníctvom programu Ansible. Testovanie a overovanie celej topológie s príslušnými SRv6 scenármi je zabezpečené cez generátor sieťovej prevádzky TRex.

KĽÚČOVÉ SLOVÁ

Ansible, Docker, FD.io, Honeycomb, hc2vpp, IPv6, NETCONF, Segment Routing, SR, SRv6, TRex, VPP, 5G Network Slicing

ABSTRACT

The development of the mobile networks of fifth-generation (5G) accomplishes the requirements of transported services. New applications and services put high pressure on bandwidth, reliability and flexibility of the data transport. Due to this trend, network technologies need to be developed to address it and ensure transmission quality. This master's thesis aims at the Segment Routing (SR) concept and its commitment to improving transport quality in the transport part of 5G mobile networks. SR allows the integration of network programming into the traditional design of transport networks. SR uses either MPLS (SR-MPLS) or IPv6 (SRv6) on the forwarding plane. This thesis aims at SRv6. The main goal of the thesis is to design and implement a communication scenario for the transport part of the network with the SRv6 protocol. Docker platform is used to create a topology with all its nodes. The VPP platform is used to enable SRv6 on top of this topology. VPP has the ability to be configured via NETCONF protocol thanks to the Honeycomb agent, which can process NETCONF messages and propagate them to the VPP configuration. This configuration is done by the program Ansible, which can send NETCONF configurations to all SR nodes with Honeycomb installed. Testing and verification of the topology with all SRv6 policies is handled by the TRex traffic generator.

KEYWORDS

Ansible, Docker, FD.io, Honeycomb, hc2vpp, IPv6, NETCONF, Segment Routing, SR, SRv6, TRex, VPP, 5G Network Slicing

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

MJASOJEDOV, Igor. *Implementace mechanismů zajišťujících Segment Routing for IPv6 (SRv6) pomocí platformy FD.io VPP*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022, 163 s. Diplomová práca. Vedúci práce: Ing. Aneta Koláčková

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Igor Mjasojedov
VUT ID autora:	204378
Typ práce:	Diplomová práca
Akademický rok:	2021/22
Téma závěrečnej práce:	Implementace mechanismů zajišťujících Segment Routing for IPv6 (SRv6) pomocí platformy FD.io VPP

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 bych poděkoval vedoucímu diplomové práce pani Ing. Anete Koláčkovej za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Rapídna digitalizácia všetkých odvetví a neodmysliteľná súčasť moderných technológií v každodennom živote kladie stále vyššie nároky na prenosové technológie. Zvyšujúce sa požiadavky prenášaných služieb na rýchlosť, spoľahlivosť a flexibilitu prenosu majú za následok vývoj aj nových architektúr v oblasti sieťových technológií. Posledné roky sa dostáva do trendu možnosť programovateľnosti sieťovej infraštruktúry. Softvérovo definované siete umožňujú jednoduchšiu a presnejšiu optimalizáciu siete, ale aj centralizovanú konfiguráciu a správu jednotlivých častí rozsiahlych sietí. Zmena architektúry z tradičného prístupu riadenia sietí na plne centralizovanú, softvérovo definovanú architektúru však nie je jednoduchá a nie vždy aj úplne vyžadovaná. Z tohto dôvodu sa zavádza nový koncept, Segment Routing (SR). SR umožňuje jednoducho implementovať výhody programovateľnosti siete nad jej tradičnou implementáciu bez nutnosti masívneho zásahu do jej infraštruktúry. Koncept SR sa ešte len postupne zavádza do praxe a je stále vo fáze vývoja aj napriek tomu, že ho podporujú viaceré platformy a aj komerčné firmy. Jednou z platforiem, ktorá implementuje koncept SR je platforma FD.io VPP.

Táto diplomová práca sa zameriava na podrobný popis nového konceptu SR a taktiež platformy FD.io VPP. Cieľom práce je vytvorenie komunikačného scenára odpovedajúceho transportnej časti mobilných sietí 5G NR. V rámci vytvoreného komunikačného scenára sa simuluje protokol Segment Routing for IPv6 (SRv6) pomocou platformy FD.io VPP.

Prvá kapitola sa venuje popisu konceptu Segment Routing. Následne je detailne popísaná inštancia SR nad protokolom IPv6. Táto kapitola sa taktiež venuje platforme FD.io VPP, ktorá sa využíva v praktickej časti tejto diplomovej práci pre spojazdnenie SRv6 nad vytvorenou topológiou.

V druhej kapitole sú popísané ostatné platformy a programy potrebné pre vytvorenie funkčného simulačného scenáru. Popisovanou platformou v tejto kapitole je platforma Docker, vďaka ktorej sa vytvára základ topológie. Následne je popisovaný generátor sieťovej prevádzky TRex, ktorým je možné generovať reálnu sieťovú prevádzku a tak testovať funkčnosť topológie a jednotlivých scenárov. Posledná sekcia tejto kapitoly sa venuje programu Ansible pre konfiguráciu platformy VPP prostredníctvom protokolu NETCONF.

Tretia kapitola sa zameriava na návrh logickej topológie a jednotlivým scenárom SRv6, ktoré sa nad touto topológiou vytvárajú.

V štvrtej kapitole je popisovaný prehľad topológie nad platformou Docker. Taktiež sa venuje manuálnej konfigurácii VPP a spojazdneniu generovania sieťovej prevádzky generátorom TRex. Jednotlivé sekcie obsahujú všetky potrebné príkazy a úkony potrebné pre vytvorenie kompletnej topológie. Posledná sekcia tejto ka-

pitoly sa venuje automatizácii procesu tvorby topológie, ako aj konfigurácii VPP prostredníctvom programu Ansible.

Piata kapitola sa venuje podrobnejšiemu popisu niektorých z vytváraných scenárov. Popis sa zameriava na demonštrovanie prechodu jednotlivých paketov naprieč vytvorenou sieťou.

V závere práce sa zhrňujú všetky nadobudnuté výsledky a poznatky.

1 Teoretický rozbor

V tejto kapitole sú zhrnuté a teoreticky vysvetlené hlavné technológie a protokoly, ktoré táto diplomová práca vyžaduje. Prvá podkapitola sa venuje architektúre *Segment Routing* (SR), konceptu od ktorého sa odvíja, jej výhodám a dôvodu stále väčšej popularity. Ďalšia podkapitola sa zameriava na popis konkrétneho protokolu založeného na princípoch SR. Jedná sa o protokol *Segment Routing over IPv6* (SRv6). Keďže cieľom práce je demonštrovať praktické využitie relatívne nového protokolu SRv6 a vytvorenie funkčného scenáru jeho nasadenia v transportnej časti siete, nutnou súčasťou teoretického rozboru musí byť aj platforma na ktorej je možné takýto scenár simulovať. Tou je platforma FD.io *Vector Packet Processing* (VPP), ktorá podporuje architektúru SR a taktiež protokol SRv6.

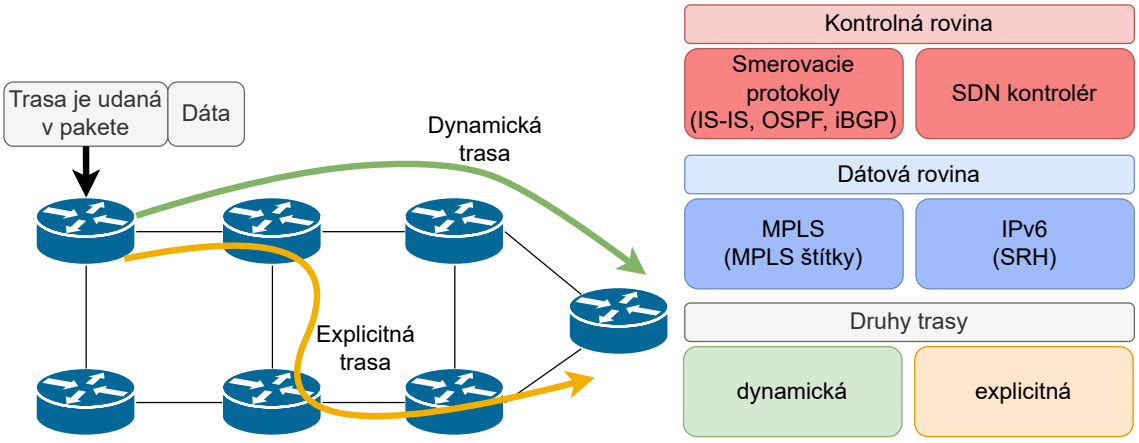
1.1 Segment Routing

Na vývoj mobilných sietí, ako aj sieťových technológií všeobecne, sa kladie obrovský dôraz z dôvodu zvyšujúcich sa požiadaviek na ich rýchlosť so stúpajúcou integráciou miliárd zariadení a miliónov služieb. Nastupujúci trend prechodu k softvérovo definovaným sieťam, ktorý umožňuje presnejšiu optimalizáciu a dôslednejšie spravovanie požiadaviek na jednotlivé služby vyžaduje zmenu aj v architektúre tradičných sieťových štruktúr a postupnú vzájomnú integráciu. Práve z dôvodu plynulej integrácie novej architektúry plne softvérovo definovaných sietí sa vytvorila architektúra sieťovej správy, ktorou je Segment Routing.

SR umožňuje zabezpečiť jednotlivé zvyšujúce sa požiadavky a ich integráciu do tradičného dizajnu sietí. Architektúra SR je založená na koncepte zdrojového smerovania, ktorým odľahčuje sieťové zariadenia od nutného udržiavania prílišného množstva stavových informácií pre podporu náročných služieb a ich požiadaviek pri smerovaní ich dát [41]. Nasadenie SR v tradičných sieťach je vyhovujúce a relatívne priamočiare, keďže jedinou podmienkou k prechodu k SR v transportných častiach sietí je softvérová podpora danými zariadeniami [16]. Preto nie je nutná investícia do nových zariadení a kompletná zmena celého dizajnu sietí. Začiatkom minulej dekády sa sformovala pracovná skupina „Source Packet Routing in Networking“ (SPRING WG) pod záštitou *Internet Engineering Task Force* (IETF) pre vývoj SR architektúry [41]. Postupná štandardizácia tejto architektúry vyvolala záujem u poskytovateľov internetových služieb ako aj korporácií akými sú Google, alebo Facebook [16]. Vhodná implementácia architektúry SR je v prípade transportných a prístupových častí sietí, dátových centier a 5G sietí. Z hľadiska dátovej roviny (data plane) sa vytvorili dve inštancie SR architektúry a to *Segment Routing over MPLS* (SR-MPLS) a Segment Routing over IPv6(SRv6). Práve druhej zmienenej

inštancii, SRv6, sa venuje táto diplomová práca. Podrobnejšiemu popisu SRv6 bude venovaná podkapitola 1.2. V nasledujúcej časti sú však popísané základné pojmy architektúry SR, ako aj základ jej fungovania.

Ako už bolo spomenuté, SR je založený na základe konceptu zdrojového smerovania (Source Routing). Vďaka tomu sa prechod jednotlivých paketov cez danú oblasť siete, tzv. SR doména, určuje na vstupných uzloch, smerovačoch. Takéto hraničné uzly podporujúce SR môžu vložiť zoradený zoznam inštrukcií do hlavičiek paketu. Tieto jednotlivé inštrukcie koordinujú preposielanie a spracovávanie paketu naprieč celou cestou v danej oblasti siete, SR doméne. Samostatné inštrukcie sú nazývané segmenty a sled týchto inštrukcií, resp. zoznam segmentov sa nazýva SR politika (SR policy). Každý segment dokáže presadiť požiadavku na topológiu, ako napríklad nutný prechod cez určitý uzol, alebo servisnú požiadavku v prípade nutnosti vykonania určitej operácie nad daným paketom [41]. Každý segment je označovaný svojimi identifikátorom, tzv. Segment ID, alebo *Segment Identifier* (SID). Keďže segmenty sú vkladané do paketov na zdrojových uzloch, zvyšné uzly naprieč SR doménou jednoducho čítajú jednotlivé značky, resp. identifikátory segmentov v hlavičkách paketov a vykonávajú špecifikované inštrukcie. To znamená, že tieto uzly si nemusia udržiavať žiaden stav dátových tokov, ktoré preposielajú, čo výraznou mierou odľahčuje a zjednodušuje kontrolnú rovinu riadenia sete (control plane) [35]. V nasledujúcom obrázku Obr. 1.1 je znázornená topológia s dvomi trasami. Jedna z nich je vypočítaná dynamicky na základe najkratšej cesty a druhá je explicitne definovaná pre daný tok prostredníctvom zoznamu segmentov, teda inštrukcií. Okrem toho sa v pravej časti obrázku nachádza sumárne rozdelenie kontrolnej a dátovej roviny s možnosťami trasy naprieč sieťou pre jednotlivé pakety.



Obr. 1.1: Segment Routing architektúra [35].

SR dátová rovina

Z pohľadu dátovej roviny sa segmenty, resp. zoznam segmentov prenáša cez *Multiprotocol Label Switching* (MPLS) štítky v inštancii SR-MPLS, viď RFC 8660 [3]. V prípade inštancie SRv6 sa zoznam segmentov prenáša v novej rozširujúcej hlavičke IPv6 protokolu nazývanej *Segment Routing Header* (SRH) definovanej v RFC 8754 [23] vo forme IPv6 adries.

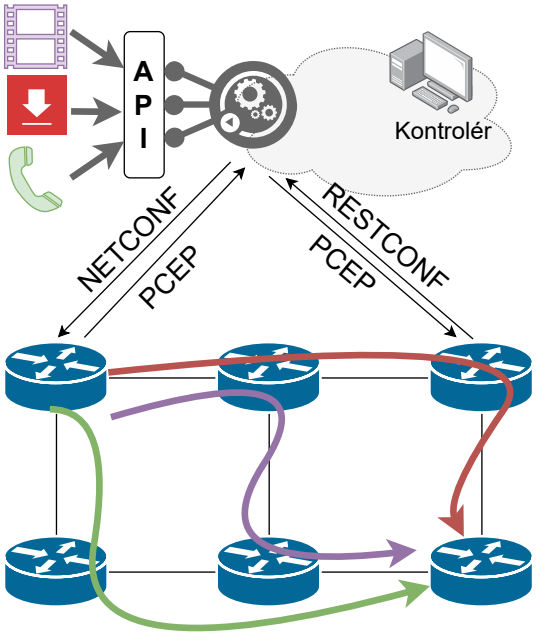
SR kontrolná rovina

Kontrolná rovina sa môže rozdeliť na distribuovanú, centralizovanú a hybridnú architektúru. V prípade distribuovaného prístupu sa využívajú *Interior Gateway Protocols* (IGP) smerovacie protokoly k signalizácií alokácie segmentov a jednotlivé vstupné uzly SR domény nezávisle rozhodujú o vkladaní segmentových zoznamov do prenášaných paketov. Centralizovaná architektúra využíva tzv. *Software-defined networking* (SDN) kontrolér, ktorý prideľuje segmenty a určuje to, ktoré pakety sa majú spracovávať jednotlivými SR politikami. Následne sa uzly danej SR domény automaticky konfigurujú prostredníctvom tohto kontroléru. Hybridná architektúra kombinuje predošlé dve architektúry, kedy smerovače distribuované v sieti vypĺňajú určitú funkciu a SDN kontroléry napríklad definujú Segment Routing politiky a vypočítavajú medzi-doménové trasy. V praxi sa hybridný prístup využíva najviac [24, 35].

Software Defined Networks

Na pojmy Software-defined networking (SDN) a SDN kontrolér sa vzťahuje množstvo interpretácií ich významov a ich úlohy v sieti. V niektorých prípadoch sa SDN interpretuje ako kompletná automatizácia siete vrátane jej plne centralizovanej správy. V iných prípadoch sa jedná len o správu tokov v danej sieti. Segment Routing nepotrebuje pre svoju funkčnosť externý kontrolér. Avšak v prípade, ak sa SR politika rozrastá a stáva sa komplexnejšou, alebo sieť ako taká sa rozširuje a presahuje jednu doménu, je vhodné uvažovať nad implementáciou takéhoto centralizované riešenie prostredníctvom SDN kontorléru [35]. Nasledujúci diagram Obr. 1.2 zobrazuje výhody SR, ako logickú topológiu s využitím SDN kontroléru.

Ako už bolo spomenuté, moderné siete sa potrebujú adaptovať k službám a preto sa vyvíja nový prístup správy siete, ktorý umožňuje upravovať tok dát na sieti v závislosti od požiadaviek jednotlivých služieb. Adaptácia k požiadavkám služieb implementovaná cez tradičnú architektúru sietí je zložitá a náročná na jej správu. Diagram Obr.1.2 zobrazuje službami vedenú sieť, kde sa explicitné cesty pre jednotlivé služby vyrátavajú na základe požiadaviek týchto služieb. Takáto sieť sa dokáže dynamicky

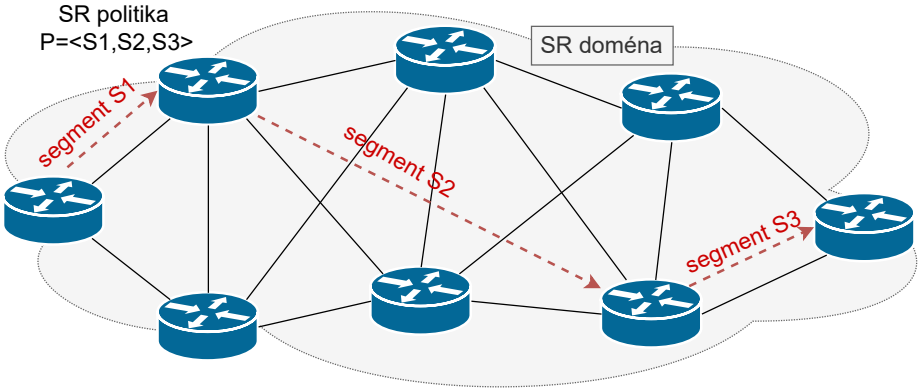


Obr. 1.2: Architektúra SR topológie [27]

adaptovať a upravovať v reálnom čase, čím sa výrazným spôsobom zvyšuje spôsobilosť zabezpečiť kvalitnú prepravu pre individuálne toky jednotlivých služieb a ich dynamických zmien. Tento koncept rozdeľovania sieťových zdrojov sa v 5G nazýva Network Slicing. V diagrame sú zobrazené 3 typy dátových prenosov. Prvým typom je prenos statických dát, ako prenos veľkých súborov, ktorý vyžaduje hlavne vysokú prenosovú rýchlosť siete. Druhým typom prenosu je prenos online videa. Takýto prenos vyžaduje hlavne nízke oneskorenie siete kvôli plynulému prenosu. Tretím zobrazeným dátovým tokom, resp. službou je audio prenos. Tento prenos je vhodné posielať takými časťami siete, kde dochádza k najnižšej strate paketov. Pre jednotlivé služby a ich dátové toky sú v SDN kontroléri dynamicky volené rôzne trasy naprieč SR sieťou na základe informácií z aplikačnej vrstvy, ktoré sú prijímané cez svoje API. Dynamické inštrukcie pre jednotlivé smerovače v sieti sprostredkúva kontrolér prostredníctvom manažmentových protokolov ako PCEP, NETCONF, alebo prostredníctvom protokolu BGP-LS, ktorý umožňuje preposielať link-stavové informácie k jednotlivým smerovačom [27].

Fungovanie SR

Nasledujúca časť sa venuje architektúre SR a jeho fungovaniu. Jednotlivé koncepty a základná terminológia je vysvetlená vo všeobecnosti a platí pre obe inštancie SR a teda pre SR-MPLS, ako aj SRv6. Vysvetľované pojmy sa postupne odkazujú na nasledujúcu topológiu Obr. 1.3.



Obr. 1.3: Topológia s využitím SR [41].

Na obrázku Obr. 1.3 je znázornená topológia, nad ktorou je implementovaný Segment Routing. Takáto skupina uzlov zúčastňujúcich sa v SR sa nazýva SR doména. Tieto uzly sú zväčša pripojené do rovnakej fyzickej infraštruktúry, avšak môže sa jednať aj o uzly prepojené medzi sebou vzdialene prostredníctvom *Virtual private network* (VPN). Predpokladá sa avšak, že všetky uzly sú spravované jednou administratívou. V obrázku Obr. 1.3 je taktiež znázornená tzv. SR Politika P=<S1,S2,S3>, pozostávajúca z troch segmentov, ktoré sú označené svojimi SID. Vstupný uzol, resp. uzol cez ktorý jednotlivé pakety vstupujú do SR domény asociuje určitú SR politiku k danému paketu, v tomto prípade k politike P. K takejto správe segmentov rozlišujeme tri základné operácie nad segmentami: PUSH, NEXT, CONTINUE.

`PUSH`

Operácia PUSH umožňuje vkladanie segmentu na vrchol zoznamu segmentov. V prípade SR-MPLS sa zoznamom segmentov myslí halda MPLS štítkov. Pri inštancii SRv6 je vrchol zásobníku reprezentovaný prvým segmentom v IPv6 hlavičke SRH [23].

`NEXT`

Pre vysvetlenie operácie NEXT je nutné vysvetliť pojem „aktívny segment“. Aktívny segment je segment, ktorý sa používa tým uzlom, ktorý ho bude práve spracovávať. V prípade SR-MPLS je to vrchný štítok a v prípade SRv6 sa jedná o cieľovú IPv6 adresu. Operácia NEXT sa vykonáva po spracovaní aktívneho segmentu. Pozostáva z inšpekcie nasledujúceho segmentu v zozname segmentov a vložení nasledujúceho segmentu do role aktívneho. Pri SR-MPLS sa operácia NEXT vykoná odstránením vrchného štítku v zásobníku a pri SRv6 sa skopíruje nasledujúci segment z hlavičky SRH do poľa cieľovej adresy IPv6 hlavičky, keďže všetky segmenty v inštancii SRv6 sú reprezentované IPv6 adresou.

`CONTINUE`

Operácia CONTINUE označuje situáciu, kedy sa aktívny segment nedokončí, resp. nevykoná a teda zostáva stále aktívnym. V SR-MPLS je táto operácia reprezentovaná operáciou MPLS SWAP. V prípade SRv6 sa paket preposiela ďalej klasickým smerovaním na základe aktuálnej cieľovej IPv6 adresy bez jej zmeny.

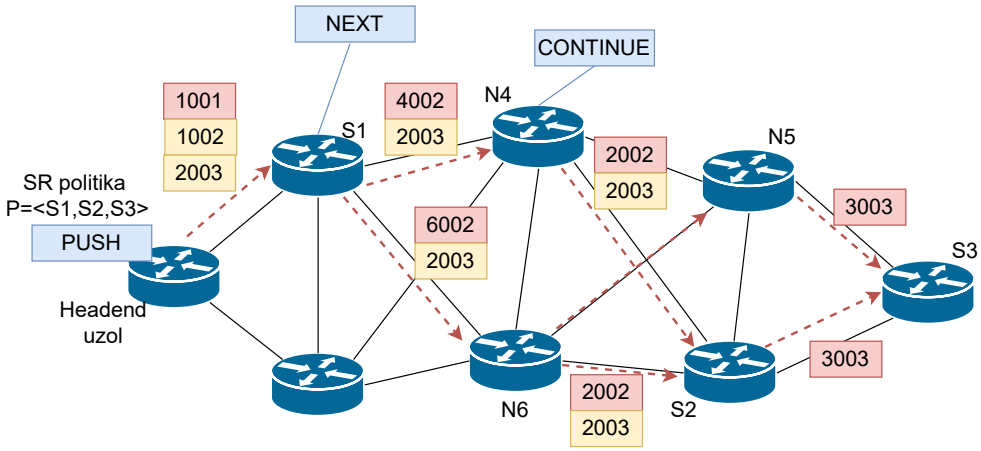
Význam segmentu môže byť lokálny, len pre určitý SR uzol, alebo globálny naprieč celou SR doménou. Ďalej rozlišujeme tzv. IGP segmenty. Tieto segmenty označujú určitú informáciu o sieti, ktorá je propagovaná link-stavovými IGP protokolmi, ako sú OSPF, alebo IS-IS. Hlavnými troma IGP segmentami sú IGP-Prefix, IGP-node a IGP adjacency segment [24]. IGP-Prefix segment je IGP segment reprezentujúci IGP prefix. Tento segment je typicky globálny a označuje inštrukciu k preposlaniu paketu naprieč vypočítanej ceste pomocou používaného smerovacieho algoritmu. IGP-Node segment označuje konkrétny smerovač, resp. uzol v danej SR doméne a má teda globálny význam. IGP-Adjacency segment je naopak segment lokálny a označuje susedstvo s druhým uzlom. Tento segment môže byť vykonaný pri potrebe zaslaní paketu na určité rozhranie iba určitým uzlom. Kombináciou týchto troch segmentov a ich sekvenčným vložením do zoznamu segmentov môžeme zostrojiť ľubovoľnú cestu naprieč danou sieťou, resp. SR doménou [27].

Segment Routing MPLS

SR inštancia dátovej roviny cez MPLS je definovaná v RFC8660 [3]. SR politika je definovaná cez MPLS zásobník značiek. Jednotlivé segmenty, resp. identifikátory segmentov (SID), sú vkladané do listu segmentov ako MPLS značky. Klasické smerovacie funkcie pre MPLS vyhovujú aj pri implementácií SR. Funkcia PUSH je reprezentovaná v SR-MPLS ako Label Push a teda funkciu vkladania značky do zásobníku. Funkcia NEXT je reprezentovaná SR-MPLS funkciou Label POP, ktorá odstraňuje segment, resp. značku na vrchole zásobníku. Poslednou generickou SR funkciou je funkcia CONTINUE a tá je reprezentovaná v SR-MPLS ako funkcia Label Swap. Táto funkcia asociuje značku na vrchole zásobníku prijímaného paketu s rozhraním, cez ktorý sa má daný paket ďalej posielať. Enkapsulácia IP paketu sa v SR-MPLS vykonáva na hraničnom SR uzle, resp. smerovači a teda na hranici SR domény na základe SR politiky pre daný dátový tok.

Hlavný proces, z pohľadu dátovej roviny v MPLS-SR, je mapovanie segmentov na MPLS značky. Jednotlivé smerovače môžu mať k dispozícii rôzne voľné rozsahy značiek, ktoré môžu byť použité pre Segment Routing. Práve preto môže každý smerovač propagovať svoj voľný rozsah značiek pre ich použitie v globálnych segmentoch. Takýto rozsah voľný značiek pre segmenty sa nazýva *Segment Routing*

Global Block (SRGB). Z tohto dôvodu sa globálne segmenty v MPLS SR doméne definujú ako indexy. Tieto indexy sú následne menené na MPLS značky a táto zmena indexu na značku berie do úvahy to, ktorý uzol bude daný segment, resp. značku spracovávať. Tento proces označenia globálnych segmentov indexmi a ich transformácia na značky za využitia SRGB je zobrazený na nasledujúcich obrázkoch Obr. 1.4 a Obr. 1.5. V prvom z nich sa zobrazuje situácia, kde má každý uzol rôzny SRGB a v druhom zas situácia, kedy je SRGB rovnaký pre všetky uzly.

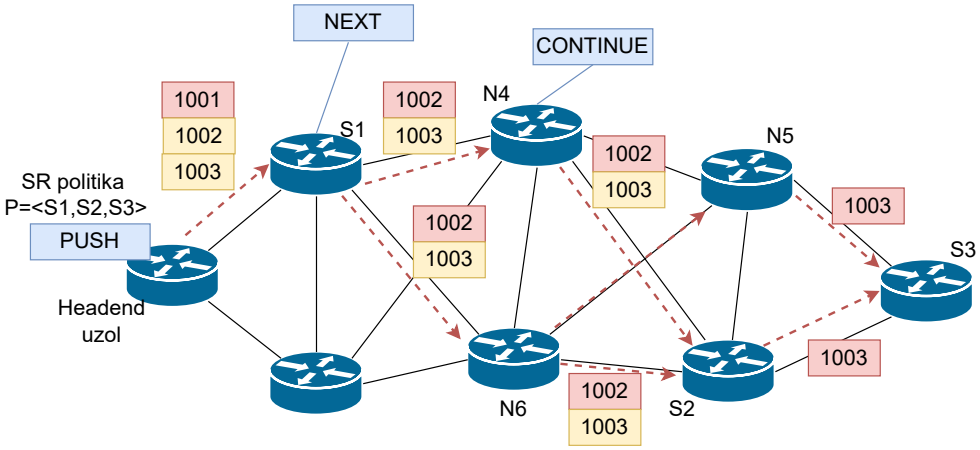


Obr. 1.4: SR-MPLS topológia s rôznymi SRGB [41].

V obrázku Obr. 1.4 vyššie predpokladáme, že každý uzol má rôzne SRGB. Jednotlivé segmenty sú označené indexmi 1 až 3. Počiatočný uzol prijme paket a musí doň vložiť zoznam segmentov. Avšak, pred tým musí zistiť, ktoré uzly majú vykonávať NEXT operáciu. Na základe toho upravuje značky spôsobom, kedy sa sčíta hodnota indexu segmentu a hodnota SRGB pre daný uzol, ktorý má funkciu NEXT vykonať. Tento výpočet sa vykoná pre každý segment v danej SR politike. V prípade preposielania paketu medzi jednotlivými susednými uzlami funkciou CONTINUE, musí každý smerovač zmeniť značku na základe toho, ktorému susednému uzlu sa bude paket posielať. Zmena značky podlieha rovnakému princípu ako na vstupnom uzle a teda sčítaním SRGB susedného uzlu a pričítaním indexu segmentu na vrchole zásobníku. Takýmto spôsobom sa paket zaručene prepraví cez uzly špecifikované cez SR politiku a doručí sa až k poslednému uzlu.

Proces zmeny značiek na každom uzle naprieč SR doménou komplikuje funkčnosť a v neposlednom rade aj riešenie prípadných problémov administrátormi siete. Preto sa odporúča využívať rovnaký rozsah značiek (SRGB) pre všetky uzly, smerovače. Nasledujúci obrázok Obr. 1.5 vystihuje práve tento ideálny prípad.

Každý globálny segment je reprezentovaný rovnakou MPLS značkou, rovnakým SID. Tým sa zabezpečí konzistencia naprieč celej SR domény.



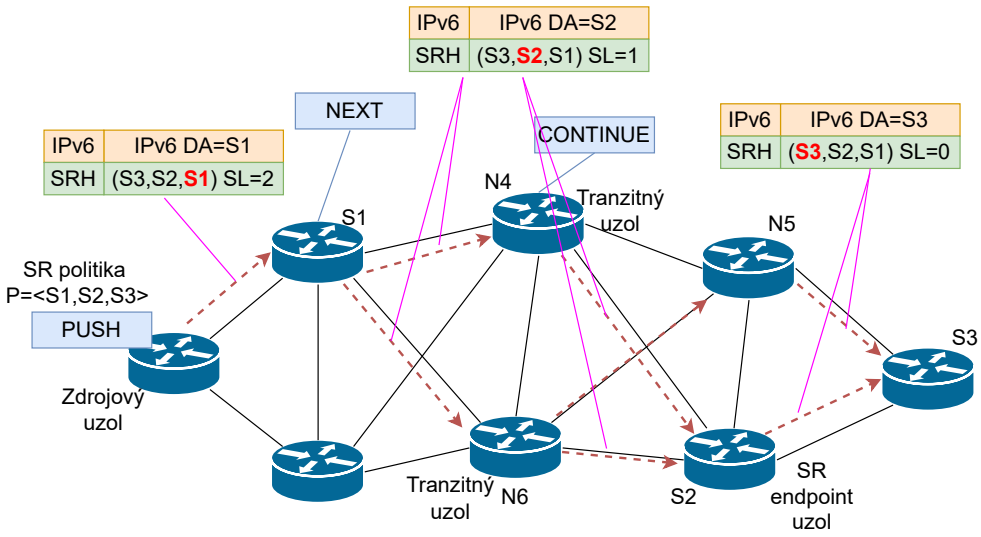
Obr. 1.5: SR-MPLS topológia s totožnými SRGB pre všetky uzly [41].

Segment Routing s využitím IPv6

Pre podporu SR prostredníctvom IPv6 bol vytvorený nový typ IPv6 rozširujúcej hlavičky nazývaný Segment Routing Header (SRH) [23]. V SRH sa prenáša zoznam segmentov určitej SR politiky ako zoznam IPv6 adries, kde každá IPv6 adresa v zozname reprezentuje identifikátor segmentu. Okrem toho sa v SRH hlavičke prenáša aj pole s názvom `Segments left`, ktoré odkazuje na aktívny SID v zozname segmentov. Podrobnejšej špecifikácii každej hodnoty prenášanej v rozširujúcej IPv6 hlavičke SRH sa venuje nasledujúca podkapitola 1.2.3. Táto časť sa venuje princípu fungovania SRv6 a jeho rozdielu od fungovania SR-MPLS popísaného v predošlej časti 1.1.

Nasledujúci obrázok Obr. 1.6 zobrazuje topológiu s SRv6 a prechod paketu naprieč touto sieťou. V topológii je možné vidieť 3 typy SR uzlov a to `SR source node`, `SR Endpoint node` a `Transit node`.

- `SR source node` je hraničný zdrojový uzol a je to uzol, ktorý vkladá zoznam segmentov do paketu po jeho príchode do SR domény. Vkladanie zoznamu segmentov sa deje väčšinou prostredníctvom enkapsulácie pôvodne prijatého paketu (môže sa jednať o IPv4, IPv6, alebo aj L2 rámec) do IPv6 paketu s rozširujúcou hlavičkou SRH. Vkladanie SRH je však možné aj priamo do pôvodne prijatého IPv6 paketu. Tak ako aj v predošlej sekcii, aj tu sa vyskytujú 3 segmenty, S1, S2 a S3. Tieto segmenty sú vkladané ako zoznam segmentov do SRH a pole `Segments left - SL` odkazuje na prvý, aktívny segment v tomto zozname. Následne sa zdrojovým uzlom nastaví cieľová IPv6 adresa nového paketu ako SID prvého segmentu, keďže je reprezentovaný určitou IPv6 adresou. Takto zostrojený paket sa odosiela ďalej. Tento proces je vykonaný na zdrojovom uzle korešponduje s SR funkciou PUSH.



Obr. 1.6: SRv6 topológia s prechodom paketu [41].

- Druhým spomínaným typom uzlu je `SR Endpoint node`. V obrázku Obr. 1.6, prijíma práve tento typ uzlu, uzol N1, paket odoslaný zdrojovým uzlom, keďže cieľová adresa prenášaného IPv6 paketu je lokálne implementovaná ako segment na ňom.
- Ďalším typom uzlu je tranzitný uzol. V topológii je to uzol N4, alebo N6 a ďalšie. Paket prijatý tranzitným uzlom sa preposiela ďalej ako obyčajný IPv6 paket keďže sa cieľová adresa nezhoduje so žiadnym segmentom implementovaným na ňom. Tento proces na tranzitom uzle pre danú SR politiku korešponduje s funkciou CONTINUE. Každý IPv6 smerovač dokáže fungovať ako tranzitný uzol, aj v prípade, že nepodporuje SRv6.

Nižšie uvedená tabuľka Tab. 1.1 zobrazuje prehľad funkcii a základnú terminológiu SR v porovnaní pre obe inštancie SR a to SR-MPLS a SRv6.

Generické SR	SR-MPLS	SRv6
SR politika	Zásobník štítkov	Zoznam segmentov v SRH
Aktívny segment	Štítok na vrchole zásobníku	IPv6 adresa v poli IPv6 cieľovej adresy
Operácia PUSH	MPLS Push	Pridanie SID do zoznamu segmentov v SRH
Operácia NEXT	MPLS Pop	Dekrementácia hodnoty poľa `Segments Left` a kopírovanie aktívneho segmentu do cieľovej IPv6 adresy
Operácia CONTINUE	MPLS Swap	Preposlanie paketu na základe IPv6 cieľovej adresy

Tab. 1.1: Porovnanie SR-MPLS a SRv6.

1.2 Segment Routing over IPv6

Segment Routing over IPv6 (SRv6) je protokol novej generácie, ktorý kombinuje koncept Segment Routing (SR) a protokol IPv6. S využitím flexibilných rozširovacích hlavičiek protokolu IPv6 je možné implementovať sieťové programovanie. SRv6 redukuje počet rôznych protokolov potrebných pri dnešnej implementácií sietí, k zabezpečeniu novodobých nárokov ako Traffic Engineering, QoS, sieťové programovanie, 5G network slicing atď.. [45].

SRv6 koncept sietí je už nasadený vo verejných, komerčných sieťach ako Softbank, Iliad, China Telecom, LINE Corporation atď.. Výrobcovia sieťových zariadení ako Cisco systems, Huawei, Juniper pracujú taktiež na ich podpore SRv6. Vedie sa vývoj aj na open-source úrovni v rámci rôznych platforiem a aplikácií. SRv6 sa stretáva s podporou už aj v jadre Linuxu, projekte FD.io VPP, ale aj v P4, Wireshark, tcpdump, iptables, snort a ďalších sieťových nástrojov, aplikácií a platforiem. Projektu FD.io a platforme VPP sa venuje ďalšia podkapitola tejto práce, keďže cieľom práce je jej využitie pri simulácií funkčného scenáru implementácie SRv6 v transportnej časti siete [11].

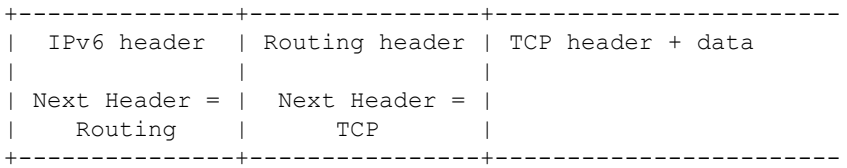
Aj na základe vyššie spomínaných informácií je zjavné, že konceptu SR a predovšetkým jeho inštancii SRv6 sa prikladá stále väčší dôraz. Dôkazom toho je aj postupná štandardizácia SRv6 organizáciou IETF. Konceptu SR a jeho inštancii SRv6 sa venujú nasledujúce štandardy: RFC8402 [24], RFC8754 [23], RFC8986 [22]. Predpokladá sa, že SRv6 postupne nahradí smerovanie segmentov cez MPLS, kvôli svojim výhodám kombinácie zdrojového smerovania, jednoduchosti a hlavne rozšíriteľnosti protokolu IPv6. SRv6 má rovnaké schopnosti a výhody ako aj MPLS. Avšak okrem toho má aj dodatočné niekoľké výhody, ktoré sú sumarizované v nasledovných bodoch:

- SRv6 ponúka široké možnosti sieťového programovania.
  Vďaka troj-dimenzionálnemu programovaciemu priestoru, resp. štruktúre segmentov umožňuje definíciu sieťovej trasy aj definíciu rôznych funkcií, ktoré majú byť vykonávané na konkrétnych uzloch podporujúcich SR. Týmto spôsobom SRv6 podporuje rôznorodé požiadavky služieb a tým pádom je SRv6 vhodná voľba pre tzv. service-driven časti sietí.
- Podporuje SDN architektúru a výpĺňa tak medzeru medzi sieťou a aplikáciami. Sprostredkovaním aplikačných požiadaviek umožňuje prispôsobenie siete a jej optimalizáciu na základe týchto informácií.
- Vďaka SRv6 sa zjednodušuje správa a implementácia siete, keďže už nie je nutné využívanie nadmerného množstva protokolov ako LDP, RSVP-TE, VxLAN, SR-TE MPLS, NSH.
- Jednoduché nasadenie. Keďže SRv6 informácie sa posielajú prostredníctvom natívneho IPv6 protokolu a to pomocou rozširujúce IPv6 hlavičky, nie je nutná zmena pôvodnej štruktúry IPv6 hlavičky. Vďaka tomu je možné preposielať SRv6 pakety aj naprieč zariadeniami, ktoré ešte SR nepodporujú. Preto sa nasadenie SRv6 výrazne zjednodušuje.

Ako už bolo spomenuté, SRv6 môže fungovať nad IPv6 vďaka konceptu rozširujúcich hlavičiek, ktoré sú podporované natívnou implementáciou IPv6 [23]. Konkrétnou rozširujúcou hlavičkou IPv6, ktorá vznikla pre potreby integrácie SR a IPv6 je smerovacia rozširujúca hlavička Segment Routing Header (SRH). Hlavičke SRH sa venuje nasledujúca časť.

1.2.1 Smerovacia hlavička segmentov

Sieťový protokol L3 vrstvy OSI modelu IPv6 disponuje možnosťou prenášať voliteľné informácie kódované v separátnych hlavičkách, ktoré môžu byť umiestnené medzi L3 IPv6 hlavičkou a hlavičkou L4 protokolu prenášaného paketu. Existuje viacero typov týchto rozširujúcich hlavičiek a každý jeden typ je identifikovaný číslom prenášaným v poli `Next Header`, viď obrázok Obr. 1.7



Obr. 1.7: Enkapsulácia rozširujúcej hlavičky protokolu IPv6 [26].

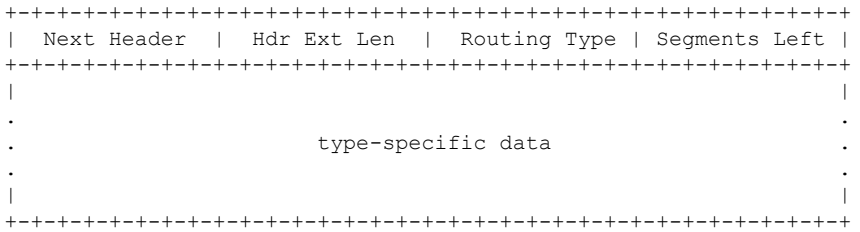
Tieto rozširujúce hlavičky nie sú spracovávané, menené, alebo mazané uzlami naprieč sieťou až kým sa prenášaný paket neprijme uzlom, ktorý je identifikovaný

cieľovou adresou v IPv6 hlavičke. Implementácia IPv6 obsahuje implementáciu nasledovných typov rozširujúcich hlavičiek 1.2.

Identifikačné číslo	**Popis**
0	IPv6 Hop-By-Hop možnosť
43	Smerovacia hlavička pre IPv6
44	Fragmentačná hlavička pre IPv6
50	Bezpečnostná enkapsulácia
51	Autentifikačná hlavička
60	Cieľové možnosti pre IPv6
135	Hlavička mobility
139	Protokol identifikácie hostiteľa
140	Shim6 protokol
253	Rezervované pre experimenty a testovanie
254	Rezerované pre experimenty a testovanie

Tab. 1.2: Zoznam typov rozširujúcich hlavičiek protokolu IPv6 [28].

Keďže sa tento text venuje SRv6, z vyššie uvedených typov hlavičiek sa budeme venovať typu: *Routing Header* (RH). Jedná sa o smerovaciu hlavičku, ktorá sa využíva zdrojovým IPv6 uzlom. Tento uzol v RH špecifikuje zoznam tranzitných uzlov, cez ktoré musí prenášaný paket navštíviť a teda musí byť týmito špecifikovanými uzlami spracovaný [26]. Rozširujúca smerovacia hlavička (RH) je identifikovaná hodnotou 43 v poli `Next Header`. Na nasledujúcom obrázku Obr. 1.8 je zobrazený formát RH.



Obr. 1.8: Formát rozširujúcej smerovacej hlavičky [26].

Next Header – 8-bitový identifikátor typu rozširujúcej hlavičky

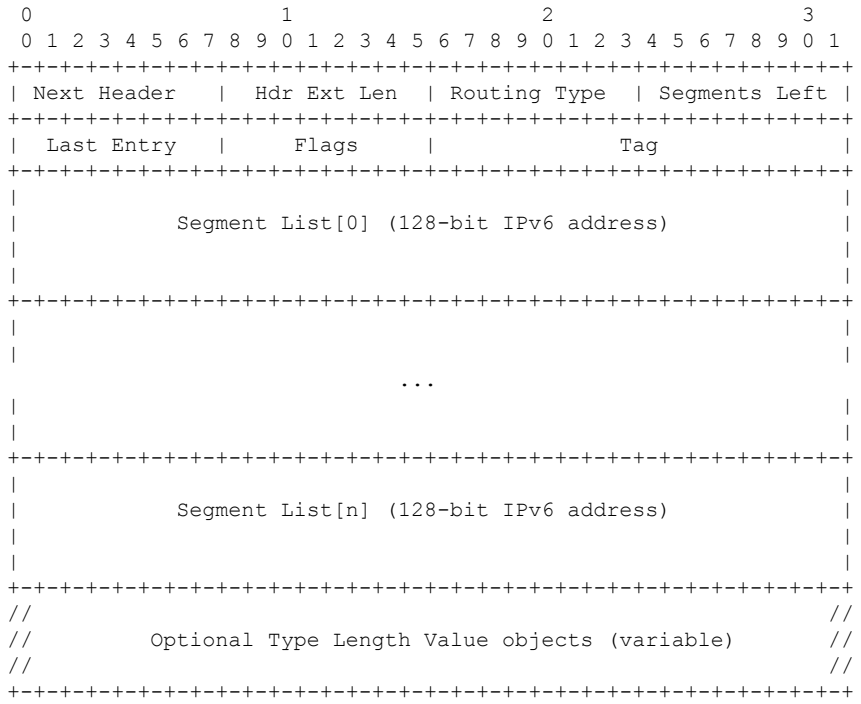
Hdr Ext Len – 8-bitová hodnota. Hodnota určuje dĺžku RH bez zahrnutia prvých 8 oktetov.

Routing Type – 8-bitový identifikátor konkrétnej varianty smerovacej hlavičky. Hodnota pre SRH je 4.

Segments Left – 8-bitové hodnota, ktorá označuje počet zostávajúcich smerovacích segmentov. Označuje počet tranzitných uzlov, ktorými sa musí prenášaný paket spracovať pred dosiahnutím pôvodného cieľa.

type-specific data - Jedná sa o pole, ktoré sa mení na základe zvolenej varianty RH špecifikovanej v poli Routing Type.

Využitie konceptu SR umožňuje nový variant smerovacej rozširujúcej IPv6 hlavičky, nazývaný Segment Routing Header (SRH). Tento variant smerovacej hlavičky (RH) má identifikačnú hodnotu 4, ktorá sa prenáša v poli `Routing Type`. Nasledujúci obrázok Obr. 1.9 zobrazuje kompletný formát smerovacej hlavičky SRH.



Obr. 1.9: Formát rozširujúcej smerovacej hlavičky variantu SRH [23].

Význam prvých štyroch 8-bitových polí bol už vysvetlený v predošlej časti. Zvyšné hodnoty sú špecifikované nasledovne:

Last Enty – obsahuje index posledného záznamu, resp. segmentu v liste segmentov.

Flags – 8 bitov, z ktorých každý jeden má definovaný osobitný význam. Väčšina z nich je však ešte rezervovaná pre budúce použitie.

Tag – označenie paketu ako súčasti triedy, alebo skupiny paketov.

Segment List[0..n] – zoznam 128-bitových IPv6 adries, kde každá adresa reprezentuje jeden segment. `Segment list` je zoradený od posledného segmentu určitej SR politiky. To znamená, že prvý segment zoznamu segmentov (Segment List[0]) obsahuje posledný segment danej SR politiky. Toto opačné poradie je

z dôvodu ukazovateľa na aktuálny segment prostredníctvom poľa `Segments Left`.

TLV – variabilné pole, ktoré umožňuje prenášať dáta vo formáte typ-dĺžka-hodnota.

Nasledujúca sekcia sa venuje formátu segmentov, ktoré sú prenášané v SRH.

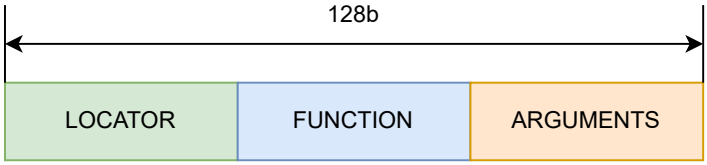
1.2.2 Formát segmentov

Vstupný uzol vkladá do paketu IPv6 hlavičku SRH, v ktorej je špecifikovaný zoznam inštrukcií. Tieto inštrukcie sú nazývané segmenty. Každá inštrukcia reprezentuje funkciu, ktorá sa vykonáva na určitom uzle v rámci SR domény. Táto funkcia je lokálne definovaná na uzle, kde sa bude vykonávať a môže sa jednať o jednoduchú funkciu preposlania paketu na ďalší špecifický uzol, alebo o funkciu aplikačnej úrovne, ako napríklad odoslanie paketu do sieťového analyzátoru. Takýmto spôsobom sa dosahuje sieťové programovanie nad tradičnou sieťovou architektúrou.

Jednotlivé segmenty sa rozlišujú svojim identifikátorom, SRv6 Segment Identifier (SID). Tento identifikátor má formát IPv6 adresy. Pre každý lokálny SID si SR uzol vytvára záznam do tabuľky *Forwarding Information Base* (FIB). Ak sa SRv6 SID nachádza v poli cieľovej adresy v IPv6 hlavičke, tak sa takýto paket posiela ďalej cez tranzitné uzly ako pri klasickom IPv6 smerovaní. Ak uzol podporujúci SRv6 prijme IPv6 paket, vykoná vyhľadávanie v tabuľke FIB na základe najdlhšej zhody prefixu nad cieľovou adresou paketu. Toto vyhľadávanie môže mať nasledujúce výsledky:

- záznam FIB, ktorý reprezentuje lokálny SRv6 SID
- záznam FIB, ktorý reprezentuje lokálne rozhranie, ak pre danú IPv6 adresu nie je implementovaný lokálny SID
- záznam FIB reprezentujúci cieľovú sieť, ktorá nie je lokálna
- bez zhody s FIB

Aj napriek tomu, že identifikátor segmentu (SID) má formát IPv6 adresy, jednotivé jej časti sa reprezentujú inak než klasická IPv6 adresa. Jeho dĺžka je 128 bitov. Formát SID pozostáva z troch častí a to LOCATOR:FUNCTION:ARGUMENTS ako je možné vidieť na nasledujúcom obrázku Obr. 1.10.



Obr. 1.10: Formát identifikátoru segmentu - SID [45].

- LOCATOR – identifikuje umiestnenie sieťového uzlu a umožňuje IPv6 smerovanie daného SID k uzlu, ktorý ho lokálne implementuje a na ktorom sa má daný SID vykonať. Toto pole môže byť reprezentované aj ako dvojica B:N, kde B označuje SRv6 SID blok a N je identifikátor uzlu, ktorý implementuje daný SID.
- FUNCTION – identifikuje funkciu, ktorá sa má vykonať na uzle, ktorý implementuje daný SID. Niektoré z najčastejšie používaných funkcií budú spomenuté v nasledujúcej podsekcii. Niektoré funkcie môžu vyžadovať doplňujúce informácie pre svoju funkčnosť. Tieto informácie sú v tom prípade obsiahnuté v ďalšej časti SID formátu.
- ARGUMENTS – jedná sa o voliteľné pole. Obsahuje podporné informácie pre pole FUNCTION. Využíva sa pre definovanie parametrov funkcie, ktorá sa má vykonávať. V prípade, že funkcia nevyžaduje žiadne argumenty, tak je toto pole prázdne.

LOCATOR časti jednotlivých SID v rámci SR domény musia byť konzistentné s celkovým adresovaním IPv6 naprieč celej siete. Preto sa vyhradzuje blok adries B::/48 pre celú SR doménu a následne bloky B:N::/64 pre jednotlivé SR uzly v rámci SR domény.

Keď sa implementuje SID na určitom SR uzle, špecifikuje sa hodnota SID ako B:N:FUNCTION a všeobecný kód správania (SRv6 Endpoint Behaviour codepoint) z registra uvedeného v RFC8986. Tento uzol následne rozposiela svoje SID vo formáte B:N:FUNCTION cez kontrolnú rovinu aj s všeobecným kódom správania typu implementovanej funkcie. Zdrojový SR uzol nedokáže vyvodiť typ správania jednotlivých SID len na základe FUNCTION hodnoty v nich. Preto je nutné rozposielať lokálne implementované SID aj s týmto kódom. Zdrojový SR uzol tak dokáže mapovať prijaté SID (B:N:FUNCTION) z SR domény k jednotlivým typom správania. Následne tak zdrojový SR uzol môže vybrať požadovaný typ správania na určitom uzle vďaka výberu SID propagovaného daným SR uzlom.

Príklad adresovania, propagácie SID a následného výberu konkrétneho SID zdrojovým uzlom je uvedený v nasledujúcich bodoch.

1. Administrátor siete priradí SRv6 SID blok adries 2001:db8:bbbb::/48 pre SRv6 infraštruktúru spravovanej siete, resp. pre SR doménu.
2. Priradia sa SRv6 LOCATOR adresné bloky pre jednotlivé SR uzly v SR doméne. Napríklad adresný blok 2001:db8:bbbb:3::/64 sa priradí pre smerovač R3.
   (a) Na jednotlivých smerovačoch sa následne priraďujú konkrétne adresy pre lokálne implementované funkcie (pole FUNCTION). Napríklad adresa

0x0100 sa s typom správania End.X (Endpoint with L3 cross-connect) priradí pre prepojenie smerovača R3 a susedného smerovača R4. Takáto funkcia je kódovaná 16 bitovou hodnotou bez argumentov (FUNCTION=16, ARGUMENTS=0).

(b) Takto zostrojený segment je propagovaný kontrolnou rovinou ako SID 2001:db8:bbbb:3:100:: s hodnotou typu správania 5 (SRv6 Endpoint Behavior codepoint pre End.X je 5).

1.2.3 SRv6 modely správania

SRv6 sieťový programovací model definuje dva druhy SRv6 správaní a to správanie SR politiky zdrojového uzlu a koncového uzlu (SR policy Headend / Endpoint behaviors). Pre sprehľadnenie textu sa budú používať názvy headend a endpoint modely správania. SR headend správanie priraďuje prijaté pakety do SRv6 politiky. Každá SRv6 politika má definovaný zoznam segmentov, reprezentujúcich sa cez ich SID. Tento zoznam sa následne pripája do prijatého paketu hraničným uzlom SR domény. SR endpoint správanie reprezentuje funkciu, ktorá sa má vykonať nad SRv6 paketmi na špecifickom uzle v sieti. Endpoint správanie sa klasifikuje na decap a no-decap. Rozdiel spočíva v tom, či sa má, alebo nemá vykonať dekapsulácia SRv6 enakpsulácie prenášaného paketu. Funkcie s headend typmi správania sa vykonávajú na zdrojovom SR uzle (SR Headend node) a endpoint typy sa vykonávajú na SR endpoint uzlami. Keďže tranzitné uzly neimplementujú žiadne lokálne SID, nie je nimi nutné vykonávať inšpekciu SRH [1]. Nasledujúca tabuľka Tab. 1.3 zobrazuje zoznam niektorých modelov správania SRv6.

Kategória	Model správania	Linux	VPP
Headend	H.Insert	✓	✓
	H.Encaps	✓	✓
	H.Encaps.L2	✓	✓
Endpoint (no-decap)	End	✓	✓
	End.T	✓	✓
	End.X	✓	✓
Endpoint (decap)	End.DT4	✓	✓
	End.DT6	✓	✓
	End.DX2	✓	✓
	End.DX4	✓	✓
	End.DX6	✓	✓
Binding SID	End.B6.Insert	✓	✓
	End.B6.Encaps	✓	✓
	End.DX2	✓	✓
Proxy	End.AS		✓
	End.AD		✓
	End.AM		✓

Tab. 1.3: Zoznam modelov správania SRv6 a ich podporou v Linuxe a na platforme VPP [1].

Nasledujúca časť sa venuje bližšiemu popisu tých najčastejšie používaných typov správaní. V prvom rade sú predstavené niektoré z Headend správaní. H.Encaps.L2 model správania je rovnaký ako H.Encaps s tým rozdielom, že sa prijatý paket zapuzdruje celý, vrátane L2 vrstvy OSI modelu a nie len od L3 vrstvy. Správanie H.Insert vkladá hlavičku SRH do prijatého IPv6 paketu medzi pôvodnú hlavičku IPv6 L3 vrstvy a hlavičku L4 transportného protokolu. Pôvodná IPv6 hlavička je modifikovaná tiež a to zámenou pôvodnej cieľovej IPv6 adresy za prvý segment zo zoznamu segmentov. Pôvodná cieľová adresa je prenášaná v poslednom SID v zozname segmentov.

End

End správanie reprezentuje najzákladnejšiu SRv6 funkciu medzi Endpoint modelmi správania. Nahradzuje IPv6 cieľovú adresu v pakete s nasledujúcim SID zo zoznamu segmentov. Následne daný paket odosiela na základe vyhľadania aktualizovanej cieľovej adresy vo FIB. Nasledujúci výpis zobrazuje pseudokód implementácie tohto modelu správania.

```
S01. Spracúvanie SRH hlavičky {
S02.   If (Segments Left == 0) {
S03.      Zastavenie spracúvania SRH a prechod k nasledujúcej hlavičke v pakete, ktorej typ
            označuje pole Next Header v smerovacej hlavičke.
S04.   }
S05.   If (IPv6 Hop Limit <= 1) {
S06.      Odoslanie ICMP Time Exceeded správy na zdrojovú adresu s kódom
            Code 0 (Hop limit exceeded in transit),
            zastavenie spracúvania paketu a následné zahodenie paketu.
S07.   }
S08.   max_LE = (Hdr Ext Len / 2) - 1
S09.   If ((Last Entry > max_LE) or (Segments Left > Last Entry+1)) {
S10.      Zaslanie ICMP Parameter Problem k zdrojovej adrese s kódom
            Code 0 (Erroneous header field encountered),
            Ukazovateľ nastavený na pole Segments Left,
            zastavenie spracúvania paketu a následné zahodenie paketu.
S11.   }
S12.   Zníženie hodnoty poľa IPv6 Hop Limit o 1
S13.   Zníženie hodnoty poľa Segments Left o 1
S14.   Aktualizácia IPv6 DA (cieľovej adresy) s hodnotou Segment List[Segments Left]
S15.   Odoslanie paketu na základe smerovacích informácií z IPv6 FIB
S16. }
```

V prípade spracovávania hlavičky vyššej úrovne paketu zodpovedajúcim FIB záznamom pre lokálny End SID sa vykonáva nasledujúci pseudokód.

```
S01.   If (Ak je hlavička vyššej úrovne paketu povolená lokálnou konfiguráciou) {
S02.      Prechod k spracovaniu hlavičky vyššej úrovne.
S03.   } Else {
S04.      Zaslanie ICMP Parameter Problem k zdrohovej adrese s kódom
            Code 4 (SR Upper-layer Header Error),
            Ukazateľ nastavený na offset hlavičky vyššej úrovne,
            zastavenie spracúvania paketu a následné zahodenie paketu.
S05  }
```

End.DX6

End.DX6 odstraňuje SRv6 zapuzdrenie (pridanú IPv6 hlavičku) z paketu a odosiela dekapsulovaný paket definovanému L3 uzlu.

```
S01. Spracúvanie SRH hlavičky {
S02.   If (Segments Left != 0) {
S03.      Odoslanie ICMP Time Exceeded správy na zdrojovú adresu s kódom
            Code 0 (Erroneous header field encountered),
            Ukazovateľ nastavený na pole Segments Left,
            zastavenie spracúvania paketu a následné zahodenie paketu.
S04.   }
S05.   Pokračovanie so spracovávaním hlavičky vyššej úrovne.
S06. }
```

Ak sa spracovávanie nasledujúcej hlavičky paketu zhoduje so záznamom v lokálnej FIB pre segment End.DX6, vykonáva sa nasledujúci pseudokód.

```
S01.   If (Upper-Layer header type == 41(IPv6) ) {
         Odstránanie vonkajšej IPv6 hlavičky s jej všetkými rozširujúcimi hlavičkami
           Odoslanie výsledného IPv6 paketu.
```

```
S03.   } Else {
S04.     If (Ak je hlavička vyššej úrovne paketu povolená lokálnou konfiguráciou) {
S05.       Prechod k spracovaniu hlavičky vyššej úrovne.
S06.     } Else {
S07.       Zaslanie ICMP Parameter Problem k zdrohovej adrese s kódom
             Code 4 (SR Upper-layer Header Error),
             Ukazateľ nastavený na offset hlavičky vyššej úrovne,
             zastavenie spracúvania paketu a následné zahodenie paketu.
S08      }
S09.   }
```

End.DX4

End.DX4 odstraňuje SRv6 enkapsuláciu (pridanú IPv6 hlavičku) z paketu a odosiela výsledný paket definovanému L3 uzlu. Pri tomto modeli správania je potrebné aby výsledný paket, bol L3 protokolu IPv4. Nasledujúci výpis kódu zobrazuje pseudokód implementácie End.DX4.

```
S01. Spracúvanie SRH hlavičky {
S02.   If (Segments Left != 0) {
S03.      Odoslanie ICMP Time Exceeded správy na zdrojovú adresu s kódom
            Code 0 (Erroneous header field encountered),
            Ukazovateľ nastavený na pole Segments Left,
            zastavenie spracúvania paketu a následné zahodenie paketu.
S04.   }
S05.   Pokračovanie so spracovávaním hlavičky vyššej úrovne.
S06. }
```

Ak sa spracovávanie nasledujúcej hlavičky paketu zhoduje so záznamom v lokálnej FIB pre segment End.DX4, vykonáva sa nasledujúci pseudokód.

```
S01.   If (Upper-Layer header type == 4(IPv4) ) {
         Odstránanie vonkajšej IPv6 hlavičky s jej všetkými rozširujúcimi hlavičkami
           Odoslanie výsledného IPv4 paketu.
S03.   } Else {
S04.     If (Ak je hlavička vyššej úrovne paketu povolená lokálnou konfiguráciou) {
S05.       Prechod k spracovaniu hlavičky vyššej úrovne.
S06.     } Else {
S07.       Zaslanie ICMP Parameter Problem k zdrohovej adrese s kódom
             Code 4 (SR Upper-layer Header Error),
             Ukazateľ nastavený na offset hlavičky vyššej úrovne,
             zastavenie spracúvania paketu a následné zahodenie paketu.
S08      }
S09.   }
```

Ostatné modely správania

End.T správanie je variant End, kedy sa vyhľadávanie vykonáva v špecifickej smerovacej tabuľke asociovanej s daným SID a nie v hlavnej smerovacej tabuľke. End.X je ďalšou variantou End, kedy sa paket odosiela priamo L3 susednému uzlu asociovanému s daným SID, bez nutnosti vyhľadávania cieľovej adresy vo FIB. End.DT6 odstraňuje SRv6 enkapsuláciu paketu a vykonáva vyhľadávanie záznamu vo FIB pre

pôvodnú IPv6 adresu v špecifickej IPv6 smerovacej tabuľke asociovanej k danému SID. End.DT4 je IPv4 varianta End.DT6. End.DX2 je využívaný pri paketoch enkapsulovaných aj s L2 vrstvou pôvodného paketu. V tabuľke Tab. 1.3 je možné vidieť aj ďalšie modely správania, ktoré sú však mimo zamerania tejto diplomovej práce.

1.3 FD.io VPP

Softvérové prepínače a smerovače získavajú čoraz stále väčšiu pozornosť v rapídne sa vyvíjajúcej ére založenej na architektúre Network Function Virtualization (NFV) a Software-defined networking (SDN). Ich nasadenie sa predpokladá v budúcich mobilných sieťach 5G. V poslednej dobe sa výkonnosť NFV/SDN riešení značne zvyšuje. To je dôsledok hlavne vývoja vysoko výkonných technológií pre spracovávanie paketov ako *Data Plane Development Kit* (DPDK), NetMap, Kamuee, alebo Vector Packet Processing (VPP). Táto podkapitola sa venuje práve posledne spomínanej technológii VPP [29, 12].

Projekt FD.io

NFV sa zameriava na výkonnostné problémy tradičných sieťových zaradení. Tieto problémy adresuje odstraňovaním priamej väzby hardvéru od softvéru. Podporuje rýchle nasadenie sieťových funkcií na generické hardvérové platformy. Požiadavky na vysokorýchlostné spracovávanie paketov s nízkym oneskorením sú teda vhodne adresované vďaka architektúre NFV. Pre neustálu potrebu zvyšovania výkonnosti NFV sa vytvoril open-source projekt Fast Data I/O (FD.io).

Projekt FD.io sa započal spoločnosťami Intel a Cisco zameriavajúc sa na vysokú priepustnosť, nízke oneskorenie a optimalizované využívanie I/O zdrojov pri spracovávaní paketov. Jadrom tohto projektu je framework Vector Packet Processing (VPP). Jedná sa o vysokovýkonný framework s vektorovým modelom spracovávania paketov a rôznymi optimalizačnými technikami. Výkonnosť spracovávania IP sieťovej prevádzky cez platformu VPP dosahuje až 15 miliónov paketov za sekundu bez jedinej straty paketu v jednom jadre systému [44].

1.3.1 Vector Packet Processing

FD.io VPP je vysokorýchlostný, škálovateľný a multipatformný sieťový framework operujúci na vrstvách L2 až L4 OSI modelu. Jeho nasadenie je možné v prostredí Linuxového užívateľského priestoru a na viacerých architektúrach vrátane x86, ARM a Power. VPP je neustále vyvíjaný a svoje vlastnosti dosahuje aj vďaka využívaniu rôznych podporujúcich modulov. Jedným z nich je aj Data Plane Development Kit

(DPDK), ktorý dodáva niektoré dôležité funkcie a ovládače pre VPP. VPP umožňuje vytvorenie rôznych sieťových inštancií smerovačov, prepínačov, firewallov, alebo inštanciu zariadenia na vyvažovanie sieťovej prevádzky [32].

FD.io VPP je vyvíjaný cez vektorové spracovávanie paketov namiesto skalárneho spracovávania. Podrobnejší rozdiel medzi spomínanými dvoma prístupmi spracovávania paketov je popísaný v nasledujúcej časti.

Skalárne spracovávanie paketov

Pri skalárnom spracovávaní paketov sa spracúva každý paket samostatne. Na základe prerušení sa vyberie jeden paket zo zásobníka sieťového rozhrania a následne sa postupne sekvenčne spracúva skupinou funkcií ako je vidno na nasledujúcom obrázku Obr. 1.11.

```
+---> fooA(packet1) +---> fooB(packet1) +---> fooC(packet1)
+---> fooA(packet2) +---> fooB(packet2) +---> fooC(packet2)
...
+---> fooA(packet3) +---> fooB(packet3) +---> fooC(packet3)
```

Obr. 1.11: Skalárne spracovávanie paketov [32].

Skalárne spracovávanie je vhodné využiť v situáciách, kde nie je až tak nutná výkonnosť spracovávania [32]. Aj keď sa jedná o relatívne jednoduchý prístup spracovávania paketov, čelí viacerým výkonnostným nedostatkom.

Vektorové spracovávanie paketov

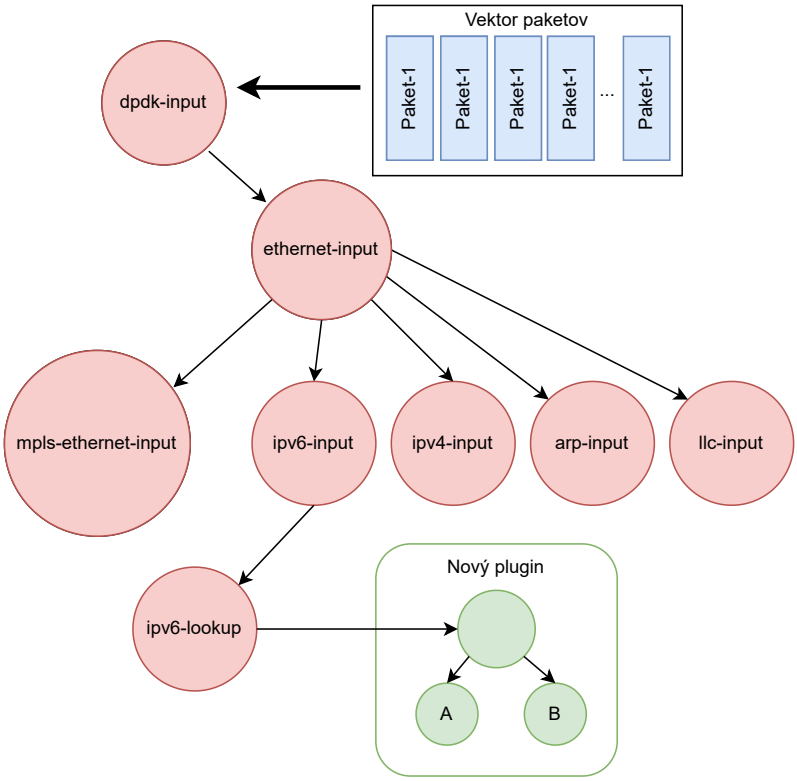
Na rozdiel od skalárneho spracovávania paketov, vektorový spôsob spracúva viacero paketov naraz. Tento súbor paketov sa nazýva aj "vektor", alebo "vektor paketov" a môže obsahovať až 256 paketov. Vektorové spracovávanie paketov je štandardný prístup pri vývoji vysokorýchlostných platforiem ako je VPP, alebo DPDK.

```
+---> fooA([packet1, +---> fooB([packet1, +---> fooC([packet1, +--->
           packet2,           packet2,           packet2,
           ...                ...                ...
           packet256])        packet256])        packet256])
```

Obr. 1.12: Vektorové spracovávanie paketov [32].

1.3.2 Graf spracovávania paketov

Platforma VPP je založená na grafe spracovávajúcom pakety. Tento modulárny prístup umožňuje pridávanie nových modulov, resp. uzlov do grafu. Tým pádom je možné prispôsobiť spracovávanie paketov k špecifickým potrebám. Nasledujúci obrázok Obr. 1.13 zobrazuje graf ktorý sa využíva pri spracovávaní paketov cez VPP.



Obr. 1.13: Graf spracovávania paketov s VPP [21].

VPP platforma sformuje vektor z príchodzích paketov. Namiesto prechodu paketov jednotlivo naprieč celým grafom, VPP inštancia spracuje celý vektor paketov v prvom uzle grafu a až potom odosiela daný vektor do druhého uzlu grafu, atď. [19]. Keďže takýmto spôsobom sa prvým paketom vo vektore pripraví inštrukčná vyrovnávacia pamäť, zvyšné pakety vo vektore sa dokážu spracovať násobne vyššou rýchlosťou. Na základe chybovosti a rýchlosti spracovávania VPP dokáže dynamicky meniť veľkosť vektoru a teda počet paketov spracovávaných naraz. Tým sa dosahuje vysoká miera stability, rýchlosti spracovávania a nízka miera odozvy. [21].

Modulárna architektúra VPP postavená na teórií grafov umožňuje jednoduchú rozšíriteľnosť vlastností spracovávania paketov. Vytvorením nových rozšírení a ich následným nasadením do grafu v podobe separátneho uzlu je možné dosiahnuť zavedenie nových vlastností grafu bez nutnosti zmeny kódu samotného jadra VPP. Okrem softvérových rozšírení je možné pridať, prípadne nahradiť za existujúce uzly,

aj hardvérové rozšírenia. Tieto rozšírenia sa môžu pridať tiež ako uzol do grafu, ktorý vykonáva určitú funkciu.

Platforma FD.io VPP je pre potreby tejto diplomovej práce vyhovujúca aj z toho dôvodu, že do výraznej miery podporuje koncept Segment Routing, ako aj jeho inštanciu SRv6. Vďaka tejto platforme je možné implementovať komunikačný scenár protokolu SRv6, v ktorom je možné simulovať nasadenie tohto protokolu do transportnej časti siete 5G. Nasledujúca kapitola 2 sa venuje ďalším platformám a programom, ktoré sa využívajú k vytvoreniu komunikačného scenáru SRv6.

2 Využívané platformy a programy

Okrem teoretického rozboru konceptu Segment Routing 1.1, jeho inštancie SRv6 1.2 a hlavnej technológie VPP 1.3, je nutné venovať pár viet aj ďalším využívaným platformám a programom. Pre vytvorenie praktickej časti tejto diplomovej práce, a teda pre vytvorenie funkčných simulačných scenárov bolo nutné využiť aj platformu ako Docker 2.1, alebo programy TRex 2.2 a Ansible 2.3. Táto kapitola sa venuje podrobnejšiemu popisu jednotlivých využívaných programov a platforiem.

2.1 Platforma Docker

Docker je softvér s otvoreným zdrojovým kódom. Umožňuje izoláciu aplikácii do tzv. kontajnerov a zabezpečuje jednotné vývojové prostredie pre systémy Windows, Linux a aj MacOS. Jedná sa v základe o sadu nástrojov, ktoré uľahčujú vytváranie, nasadzovanie a spúšťanie kontajnerov s použitím jednoduchých príkazov, prípadne automatizáciou cez API. Izolácia jednotlivých kontajnerov umožňuje chod viacerých kontajnerov na hostiteľskom systéme súbežne. Kontajnery sú odľahčené, samostatne spustiteľné inštancie tzv. obrazov, ktoré obsahujú všetko potrebné pre chod daného virtualizovaného systému, resp. aplikácie. Vďaka tomu sú jednoducho prenositeľné a nie sú závislé na aktuálnych programoch a ich verziách v hostiteľskom systéme [31]. Nasledujúca časť sa venuje architektúre platformy Docker.

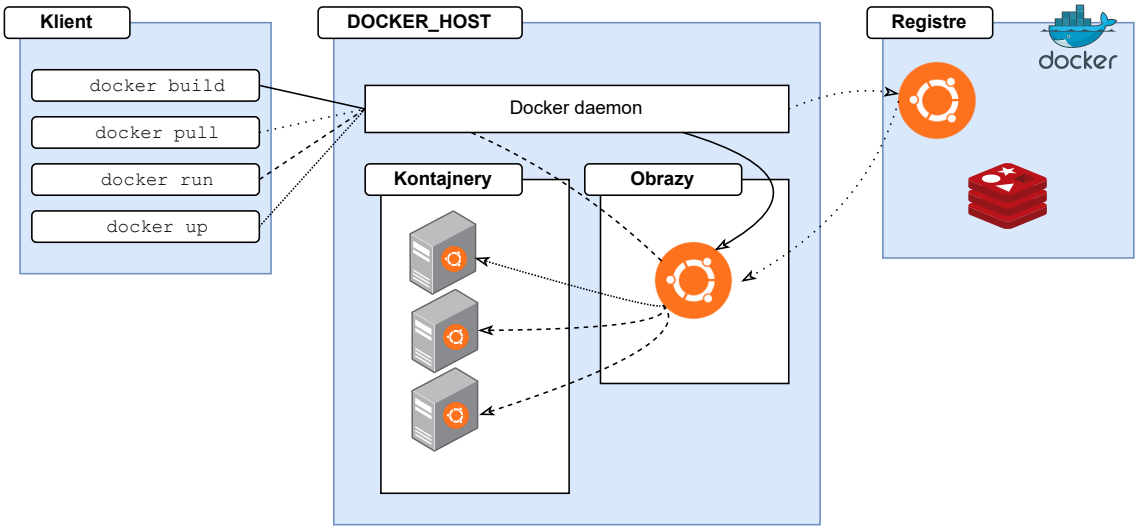
Docker využíva klient-server architektúru. Klient komunikuje s Docker démonom, ktorý zabezpečuje vytváranie, chod a distribúciu Docker kontajnerov. Docker klient aj démon môžu fungovať na tom istom systéme, alebo sa môže využívať vzdialený prístup k Docker démonu. Docker klient komunikuje s démonom cez REST API, UNIX sokety, alebo cez sieťové rozhranie. Špeciálnym Docker klientom je aj Docker Compose, ktorý dokáže pracovať s viacerými kontajnermi prepojenými navzájom súčasne. Na nasledujúcom obrázku Obr. 2.1 je znázornená Docker architektúra s jej hlavnými komponentami.

V obrázku Obr. 2.1 sú znázornené okrem iného aj bloky ”registre” a ”obraz”. Jednotlivé komponenty Docker platformy sú vysvetlené v nasledovne [13].

Registre (Docker Registry) – Docker registre obsahujú Docker obrazy (images). Z verejne prístupných registrov ako Docker Hub [1] je možné stiahnuť a následne použiť tieto obrazy k vytvoreniu konkrétnych kontajnerov.

Obrazy (Docker Images) – Docker obraz je šablóna inštrukcií pre vytvorenie Docker kontajneru. Často je obraz závislý na ďalšom obraze s určitými pris-

[1] Docker Hub - `https://hub.docker.com`



Obr. 2.1: Architektúra platformy Docker [13].

pôsobeniami. Napríklad je možné vytvoriť obraz webového serveru Apache, avšak v závislosti na základom systéme Ubuntu s už špecifickou konfiguráciou, ktorá umožňuje plynulý chod vyvíjanej aplikácie. Je možné vytvoriť aj vlastný obraz. Pre vytvorenie vlastného obrazu je nutné vytvoriť súbor Dockerfile so syntaxou definujúcou kroky k prekladu obrazu do kontajneru s jeho spustením. Bližší popis štruktúry a formátu tohto súboru sa nachádza v nasledujúcej pod-sekcii 2.1.

Kontajnery (Docker containers) – Docker kontajner je inštancia jeho obrazu. Kontajnery je možné vytvárať, spúšťať, zastavovať, vymazávať použitím Docker API, alebo cez príkazy v CLI. Jednotlivé kontajnery sa dajú prepájať medzi sebou a vytvárať tak aj komplexnejšie siete. Práve táto možnosť prepájania kontajnerov a následného vytvárania ich sietí sa plne využíva v tejto diplomovej práci prostredníctvom klienta Docker Compose. Viac o konkrétnom vytváraní takýchto prepojených kontajnerov sa píše v kapitole 4.

Formát súboru Dockerfile

Docker umožňuje vytváranie obrazov na základe inštrukcií zo súboru Dockerfile. Súbor Dockerfile je textový dokument pozostávajúci zo sérií príkazov, ktoré je nutné spustiť v CLI pre zhotovenie Docker obrazu. Vďaka tomu je možné automatizovane vytvárať Docker obrazy a následne z nich vytvárať jednotlivé inštancie vo forme Docker kontajnerov [14].

Pre vytvorenie obrazu zo súboru Dockerfile je nutné spustiť príkaz `docker build` v konkrétnom kontexte, resp. adresári kde sa daný Dockerfile nachádza. Poprípade

je možné špecifikovať tento súbor cez argument "-f" tohto príkazu a teda napríklad príkaz `docker build -f ../Dockerfile`. Preklad súboru Dockerfile a následné vytvorenie obrazu zabezpečuje Docker démon cez postupné na sebe nezávislé spúšťanie jednotlivých v ňom definovaných inštrukcií. Každá inštrukcia v súbore Dockerfile sa spúšťa nezávisle a jej vykonanie nemá efekt na ďalšie inštrukcie, takže napríklad príkaz `RUN cd /tmp` v súbore Dockerfile nebude mať žiaden vplyv na v poradí ďalej definované inštrukcie.

Nasledujúca časť sa venuje už formátu súboru Dockerfile a niektorým jeho inštrukciám. Celý súbor Dockerfile pozostáva z inštrukcii vo formáte: `INŠTRUKCIA argumenty`. Riadky začínajúce so znakom "#" predstavujú komentáre. Dockerfile musí začínať z inštrukcie `FROM`. Táto a niekoľko ďalších inštrukcií, ktoré sa využívajú v praktickej časti tejto diplomovej práce, je popísaných v nasledujúcom zozname.

`FROM [--platform=<platform>] <image>[:<tag>]` – Touto inštrukciou sa špecifikuje rodičovský obraz, z ktorého sa nový obraz vytvára. Voliteľný parameter `--platform` môže byť použitý pre špecifikáciu platformy cieľového obrazu. Môžu byť použité napríklad hodnoty ako linux/amd64, linux/arm64, alebo windows/amd64. Hodnota `tag` je taktiež voliteľná a jej prednastavená hodnota je `latest`.

`ARG <name>[=<default value>]` – Inštrukcia ARG je jedinou inštrukciou, ktorej definícia môže byť pred inštrukciou `FROM`. Definuje premennú, ktorá sa môže ďalej používať v ostatných inštrukciách, alebo ju môže užívateľ definovať pri spúšťaní prekladu cez prepínač `-build-arg <meno_premennej>=<hodnota>`.

`RUN <command>` – Inštrukcia RUN vykonáva príkaz zapísaný ako jej argument. Tento príkaz je vykonaný nad aktuálnym obrazom. Po vykonaní príkazu cez inštrukciu RUN sa výsledný obraz ukladá a používa sa pre ďalšie kroky, resp. inštrukcie v súbore Dockerfile. Túto inštrukciu je možné použiť niekoľko krát. V prípade príliš dlhého argumentu je možné zapísať tento príkaz na viacero riadkov a to za pomoci znaku "\"na konci riadku s pokračovaním príkazu na ďalšom riadku.

`COPY <src>... <dest>` – Ďalšou inštrukciou je inštrukcia COPY. Táto inštrukcia kopíruje súbory, alebo adresáre zo `src` cesty hostiteľského systému do súborového systému vytváraného kontajnera, na základe cieľovej cesty špecifikovanej v argumente `dst`. Je možné definovať viacero zdrojových súborov, resp. adresárov. Taktiež je možné definovať masku zahŕňajúcu viacero zdrojových súborov zhodujúcu sa s ich názvami.

`CMD ["executable", "param1", "param2"]` – V súbore Dockerfile sa môže inštrukcia CMD nachádzať len raz. Hlavnou úlohou tejto inštrukcie je definovať príkaz, ktorý sa vykoná po spustení kontajnera vytvoreného na základe obrazu vytvoreného prekladom daného súboru Dockerfile. Ďalším využitím inštrukcie

CMD je poskytnutie preddefinovaných argumentov pre inštrukciu ENTRYPOINT.

`ENTRYPOINT ["executable", "param1", "param2"]` – Tak ako inštrukcia CMD, tak aj inštrukcia ENTRYPOINT definuje príkaz, ktorý sa má vykonať po spustení konrajneru. Súbor Dockerfile musí špecifikovať minimálne jednu z týchto dvoch inštrukcií. Inštrukcia ENTRYPOINT by mala byť definovaná v prípade, že sa kontajner bude vyžívať a spúšťať ako spustiteľný súbor, resp. program.

Formát súboru docker-compose.yaml

Docker Compose je nástroj pre definovanie, konfiguráciu a spúšťanie viacerých kontajnerov súčasne. Umožňuje spustenie viackontajnerového prostredia na jednom hostiteľskom systéme. Jednotlivé kontajnery následne môžu medzi sebou komunikovať cez definované siete. Pre konfiguráciu a prepojenie jednotlivých kontajnerov je nutné vytvorenie textového konfiguračného súboru vo formáte YAML s názvom docker-compose.yaml [15]. Vytváranie komplexných aplikácií nad platformou Docker za využitia nástroja Docker Compose možno zhrnúť do nasledujúcich troch bodov.

1. Vytvorenie súborov Dockerfile pre zhotovenie jednotlivých obrazov a následne kontajnerov.
2. Vytvorenie súboru docker-compose.yaml s definíciou jednotlivých služieb, resp. kontajnerov a ich sieťových prepojení pre spustenie izolovaného prostredia.
3. Preklad, spustenie a ďalšie operácie nad celým prostredím a všetkými spolupracujúcimi kontajnermi a službami.

Konfiguračný súbor nástroja Docker Compose je YAML súbor definujúci jeho verziu, služby, siete a iné informácie, ktorých použitie sa však v tejto práci nevyžaduje. Štruktúra súboru docker-compose.yaml je nasledovná.

Príklad súboru docker-compose.yaml

```
version: '2.1'
services:
    ubuntu1:
        build: .
        ports: ["8301:2831", "4431:8445"]
        privileged: true
        sysctls:
          net.ipv6.conf.all.disable_ipv6: 0
          net.ipv4.ip_forward: 1
          net.ipv6.conf.all.forwarding: 1
        networks:
          ubuntu1_ubuntu2:
            ipv4_address: 192.168.12.101
            ipv6_address: fd12::101
            priority: 200
```

```
# --- #
    ubuntu2:
        build: .
        ports: ["8302:2831", "4432:8445"]
        privileged: true
        sysctls:
          net.ipv6.conf.all.disable_ipv6: 0
          net.ipv4.ip_forward: 1
          net.ipv6.conf.all.forwarding: 1
        networks:
          ubuntu1_ubuntu2:
            ipv4_address: 192.168.12.102
            ipv6_address: fd12::102
            priority: 200
networks:
    ubuntu1_ubuntu2:
        driver_opts:
          com.docker.network.driver.mtu: 2000
        enable_ipv6: true
        ipam:
          config:
            - subnet: 192.168.12.0/24
            - subnet: fd12::/64
```

Jednotlivé parametre konfiguračného súboru docker-compose.yaml zobrazeného vyššie sú vysvetlené nasledovne [2].

version – Označenie verzie s príslušnou syntaxou tohto súboru.

services – Tento koreňový element musí daný konfiguračný súbor obsahovať. Označuje definície služieb, resp. kontajnerov s konfiguračnými informáciami. Vo výpise vyššie sa špecifikujú dva kontajnery s názvami `ubuntu1` a `ubuntu2`. V nasledujúcej časti sú popísané elementy tretej úrovne, príslušné k jednotlivým službám kontajnerom.

- **build** – Parameter `build` špecifikuje adresár s konfiguračnými súbormi pre daný kontajner ako napríklad umiestnenie súboru Dockerfile.
- **ports** – Definícia mapovania portov pre daný kontajner.
- **priviledged** – Nastavenie tohto elementu na kladnú hodnotu umožňuje beh kontajneru v privilegovanom móde.
- **sysctls** – V prípade nutnosti definície parametrov kernelu kontajnera je možné tieto parametere špecifikovať už v tejto sekcii, čím sa pri preklade a spustení tohto kontajneru automaticky nastavia.
- **networks** – Element `networks` pod príslušným kontajnerom umožňuje špecifikáciu sietí, ku ktorým je tento kontajner pripojený. K jednotlivým sieťam sa následne definujú nastavenia, ako IP adresy kontajnera pre konkrétnu sieť. Takto špecifikované siete musia byť definované aj na globálnej úrovni a teda pod elementom `networks` koreňovej úrovne konfiguračného súboru.

networks – Pod týmto elementom koreňovej úrovne sa definujú virtuálne siete, ktoré slúžia k prepojeniu jednotlivých kontajnerov medzi sebou. V rozoberanom príklade sa definuje jedna virtuálna sieť s názvom `ubuntu1_ubuntu2`, ktorá slúži k prepojeniu kontajnerov `ubuntu1` a `ubuntu2`, ktoré si túto sieť špecifikujú pod elementom `networks` tretej úrovne.

driver_opts – Pri tomto elemente je možné definovať nastavenie ovládača pre danú sieť. V zobrazenom príklade sa nastavuje hodnota maximálnej prenosovej jednotky - *Maximum Transmission Unit* (MTU) pre sieť na hodnotu 2000 bajtov.

enable_ipv6 – Povolenie protokolu IPv6.

ipam – Tento element špecifikuje IPAM nastavenia. V tomto príklade sa nastavujú rozsahy IPv4 adries v CIDR formáte, ako aj rozsahy IPv6 adries určených pre použitie danou sieťou.

2.2 TRex generátor sieťovej prevádzky

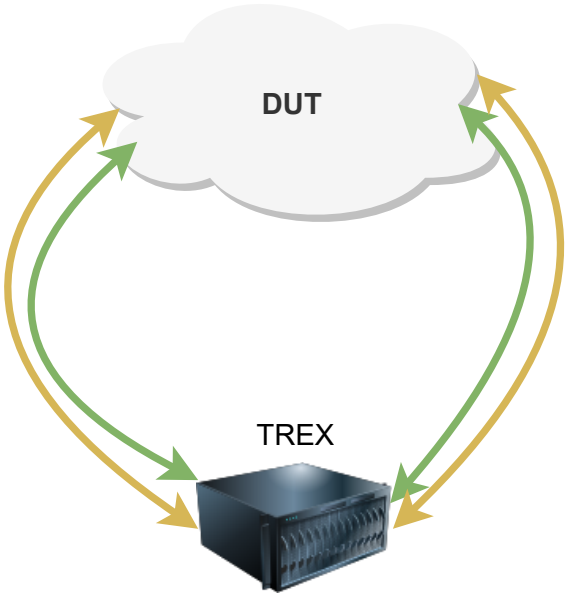
Pre vytvorenie a otestovanie plnohodnotného simulačného scenáru je vhodné poslať vytvorenou sieťou aj relevantnú sieťovú prevádzku. Z tohto dôvodu bol zvolený ako generátor sieťovej prevádzky program TRex. Táto sekcia sa venuje podrobnejšiemu popisu jeho funkcii.

TRex je vysoko výkonný open-source softvérový generátor reálnej sieťovej prevádzky vyvíjaný spoločnosťou Cisco Systems [9, 25]. Hlavnými funkciami a parametrami tohto generátoru sú:

- podpora DPDK rozhraní
- výkonnosť až do 200Gb/s na špecializovanej UCS Cisco platforme
- stavový aj bezstavový režim
- emulovanie sieťovej prevádzky na úrovni L7 OSI modelu s plnou podporou TCP/UDP
- replikácia zachytenej reálnej sieťovej prevádzky
- generovanie a amplifikácia 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. Tieto šablóny sú určitými profilmi sieťovej prevádzky a je možné ich definovať pomocou YAML konfiguračných súborov spolu s príslušnými PCAP súbormi. Taktiež je možnosť tieto profily definovať prostredníctvom vytvoreného aplikačného rozhrania v jazyku Python. Základné zapojenie TRex generátoru do siete je znázornené

na nasledujúcom obrázku Obr. 2.2.



Obr. 2.2: Zapojenie TRex generátoru [9]

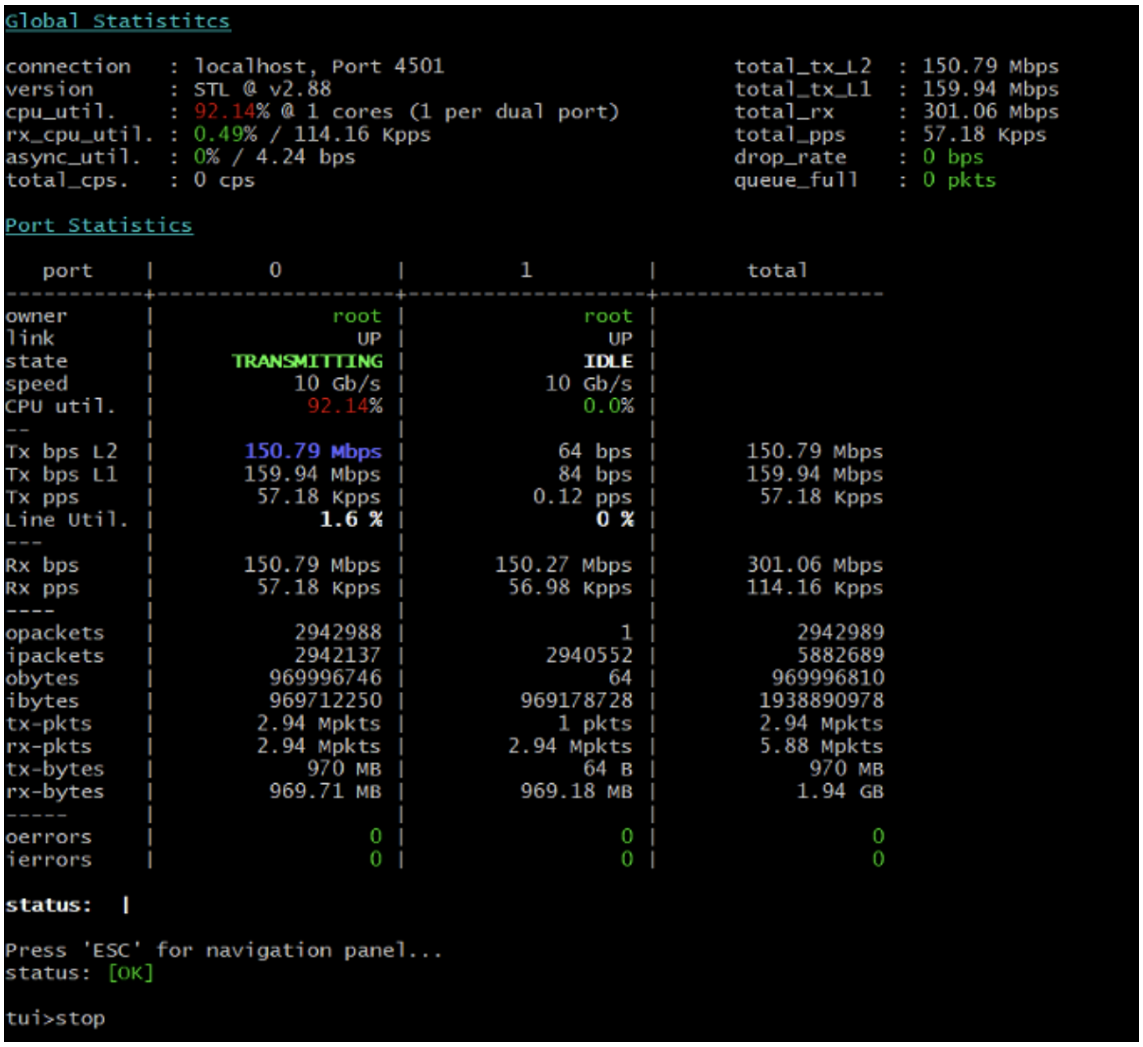
V obrázku Obr. 2.2 je znázornené zapojenie zariadenia s TRex generátorom a zariadenie s označením *Device Under Test* (DUT). DUT označuje samostatné zariadenie, alebo celú sieť, ktorá je vystavená testovaniu TRex generátorom. Táto diplomová práca bude ako uzol DUT využívať celú sieť reprezentujúcu transportnú časť 5G sietí. DUT je nutné konfigurovať takým spôsobom, aby pakety, odosielané z klientskej časti TRex generátoru, boli smerované na serverovú časť generátoru a opačne. Toto zapojenie TRex generátoru je najčastejšie.

Po ukončení generovania sieťovej prevádzky, ako aj počas generovania (v prípade spustenia interaktívnej Trex konzoly), je možné sledovať štatistiky o prenesenom množstve bajtov, paketov, CPU utilizácii, počte errorov a ďalšie informácie. V nasledujúcom obrázku Obr. 2.3 je znázornený výpis štatistík pri generovaní sieťovej prevádzky TRex generátorom.

Najpodstatnejšími informáciami z pohľadu tejto diplomovej práce bola celková úspešnosť prenesených paketov. TRex generátor je možné spustiť v dvoch základných módoch a to v bezstavovom a stavovom, ktoré sa bližšie opisujú v nasledujúcej časti.

Bezstavový mód

TRex v bezstavovom móde podporuje testovanie nad L2/L3 vrstvou OSI modelu. Tento mód umožňuje definovanie sieťového toku na základe jedného paketu a ná-



Obr. 2.3: TRex výstup počas generovania sieťového toku.

sledne celej šablóny, profilu sieťovej prevádzky. Profil môže definovať prevádzku v troch módoch:

- plynulý mód (continuous)
- dávkový mód (burst)
- viacdávkový mód (multi-burst)

Profil sieťovej prevádzky sa definuje v jazyku Python s možnosťou zostavenia ľubovolného paketu s využitím nástroja Scapy [2]. Scapy umožňuje vytváranie paketov postupným reťazením jednotlivých hlavičiek protokolov od vrstvy L2, až po L7 vrstvu OSI modelu. TRex však disponuje tzv. Field Engine modulom, ktorý dokáže dynamicky meniť ľubovoľné polia v danom pakete, alebo jeho celkovú veľkosť. Je možné napríklad definovať rozsah zdrojových a cieľových adries, ktoré následne postupne využívajú pri generovaní jednotlivých paketov a teda každý paket môže mať

[2] `https://scapy.net`

rôznu dvojicu IP adries ako aj veľkosť [10].

Nasledujúca časť popisuje príklad vytvorenia profilu sieťovej prevádzky pre bezstavový mód. V nasledovnom výpise kódu sa definuje tok plynulého módu s paketom IP/UDP s dátami o veľkosti 10 bajtov.

TRex profil sieťovej prevádzky v bezstavovom móde.

```
from trex_stl_lib.api import *

class STLS1(object):
    def create_stream (self):
        return STLStream(
            packet =
                    STLPktBuilder(
                        pkt = Ether()
                            /IP(src="16.0.0.1",dst="48.0.0.1")
                            /UDP(dport=12,sport=1025)/(10*'x')
                    ),
             mode = STLTXCont())

    def get_streams (self, direction = 0, **kwargs):
        # create 1 stream
        return [ self.create_stream() ]

# dynamic load - used for TRex console or simulator
def register():
    return STLS1()
```

Metódou `create_stream()` sa definuje IP/UDP paket s dátovou časťou o veľkosti 10 bajtov. Následne sa metódou `SRLTXCont()` určuje plynulý mód streamu. Každý profil sieťovej prevádzky je nutné registrovať metódou `SRLTXCont()`, ktorá vracia triedu `STLS1`. Týmto kódom je definovaný jednoduchý UDP plynulý tok, ktorý je následne použitý pre generovanie sieťovej prevádzky.

TRex však umožňuje aj vytvorenie profilu sieťovej prevádzky na základe už zachytených paketov a teda z PCAP súborov [39]. Ponúkajú sa dve možnosti vytvorenia takéhoto profilu a to:

- **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. Výhodami tejto metódy je však možnosť využitia Field Engine modulu a teda doplňujúcej manipulácie polí jednotlivých paketov.
- **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 dodatočnej úpravy modulom Field Engine. Úpravu pa-

ketov je v tomto prípade nutné vykonať priamo pre daný PCAP súbor. Táto metóda je podobná utilite `tcpreplay` [3].

Stavový mód

Trex v stavovom móde využíva ku generovaniu sieťovej prevádzky PCAP súbory, ktoré obsahujú TCP, alebo UDP toky. Tieto sieťové toky následne replikuje s novými IP adresami a portami pre klientskú aj serverovú stranu komunikácie. Aj napriek tomu, že tento mód generuje stavovú prevádzku, Trex neimplementuje plný TCP/IP zásobník. Dokáže plne fungovať ako klient, alebo server, avšak nedokáže interagovať s inými TCP/IP implementáciami. Plne implementovaný TCP/IP zásobník je až v móde Advanced Stateful [9, 30].

Profil sieťovej prevádzky sa v tomto móde definuje v súbore s formátom YAML. Príklad tohto súboru je zobrazený na nasledovnom výpise kódu.

TRex profil sieťovej prevádzky v stavovom móde.

```
- duration : 10.0
  generator :
          distribution : "seq"
          clients_start : "16.0.0.1"
          clients_end   : "16.0.0.254"
          servers_start : "48.0.0.1"
          servers_end   : "48.0.0.254"
cap_info :
   - name: cap2/dns.pcap
     cps : 1.0          # Generovanie 1 spojenia za sekundu
     ipg : 10000
```

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.

Pokročilý stavový mód – Stavový mód má však ešte svoje rozšírenie vo forme pokročilého stavového módu (Advanced Stateful mode). Toto rozšírenie sa môže pokladať za samostatný mód keďže na rozdiel od stavového módu implementuje kompletný TCP/IP zásobník v užívateľskom priestore a dokáže tak emulovať L7 protokoly. Tak ako aj pri bezstavovom móde, aj pri pokročilom stavovom móde sa profil sieťovej prevádzky definuje v jazyku Python [8].

[3] `https://tcpreplay.appneta.com/wiki/tcpreplay-man.html`

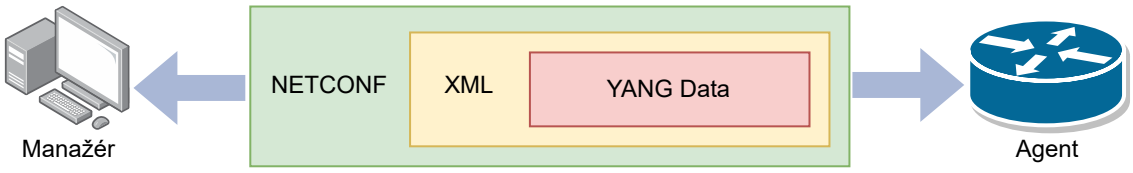
V tejto diplomovej práci sa využíva bezstavový mód generátoru TRex. Dôvod využitia práve tohto módu je jeho jednoduchosť vo vytváraní profilov sieťovej prevádzky Python modulom Scapy. Taktiež je výhodou interaktívny mód TRex konzoly (`tui`) zobrazujúci real-time štatistiky odoslaných a prijatých paketov. Pre potreby testovania vytváranej topológie bol využitý jeden profil pre generovanie troch IPv6 sieťových tokov a jeden profil pre generovanie jedného IPv4 sieťového toku.

2.3 NETCONF pomocou Ansible

Máme rôzne možnosti konfigurácie sieťových zariadení. Jednou z možností je pripojenie cez protokol telnet, alebo *Secure Shell Protocol* (SSH) k sieťovému zariadeniu a sprístupnenie príkazového riadku s možnosťou zadávania príkazov s potrebnou konfiguráciou. Ďalšou z možností je protokol *Simple Network Management Protocol* (SNMP), s ktorým je možná aj konfigurácia zariadení, avšak tento protokol sa hlavne využíva pre monitorovanie týchto zariadení. Novým, do budúcna perspektívnym spôsobom konfigurácie sieťových zariadení je *Network Configuration Protocol* (NETCONF).

NETCONF

Jedná sa o spôsob konfigurácie sieťových zariadení s použitím programu namiesto manuálnej konfigurácie cez príkazový riadok zadávaním jednotlivých príkazov. NETCONF je IETF štandard, ktorý bol po prvýkrát vydaný v roku 2006. Cieľom vytvorenia tohto štandardu a spôsobu konfigurácie je hlbšia integrácia aplikácií a siete ako takej. NETCONF umožňuje konfiguráciu zariadení jednotne, bez závislostí na ich konkrétnej platforme. Napríklad pridanie IP adresy k určitému sieťovému rozhraniu by bolo v prípade rôznych výrobcov zariadení vykonané s rôznou syntaxou príkazov. NETCONF umožňuje jednotný spôsob konfigurácie zariadení bez závislosti na ich konkrétnom výrobcovi. Na obrázku Obr. 2.4 je zobrazená komunikácia cez protokol NETCONF.



Obr. 2.4: Fungovanie NETCONF.

Ide o komunikáciu typu klient – server. Vymieňané správy medzi manažérom a spravovaným agentom fungujú na princípe *Remote Procedure Call* (RPC). Tieto

RPC správy sa posielajú v textovej podobe a sú zakódované v jazyku XML. Vďaka tomuto serializačnému formátu je možné jednoducho zasielať hierarchické štruktúry správ. Na manažérovi beží určitý program, ktorý dokáže posielať žiadosti na tzv. Agentov. Agentom môže byť sieťové zariadenie ako smerovač, alebo prepínač. Žiadosťou môže byť napríklad žiadosť o zmenu konfigurácie agenta, alebo len žiadosť o zaslanie určitých informácií z agenta. Vo volaní RPC sa nachádza XML dokument ktorý obsahuje príkaz na agenta k vykonaniu určitej operácie [17]. V nasledujúcom výpise je znázornená RPC správa operácie get-config.

RPC správa get-config.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <startup/>
  </get-config>
</rpc>
```

Pre vyjadrenie štruktúry a sémantiky konfiguračných dát sa vyvinul modelovací jazyk *Yet Another Next Generation* (YANG). Ten bol štandardizovaný v roku 2010 pod RFC6020 [4]. Jeho cieľom je vytváranie modelov, resp. schém k jednotnej konfigurácii a správe sieťových zariadení pomocou protokolu NETCONF. Tak ako je znázornené aj na obrázku Obr. 2.4, šablóna tejto žiadosti je dátový model YANG. Tento model je následne enkódovaný do XML formátu. NETCONF dokáže takto enkódované žiadosti prenášať medzi programom manažéra a agentmi s využitím typicky SSH protokolu pre komunikáciu medzi zariadeniami. Nasledujúca pod-sekcia sa venuje bližšie formátu YANG modelov.

YANG modely

Prostredníctvom NETCONF je možné ako konfigurovať, tak aj získavať informácie z jednotlivých sieťových zariadení. Pre komunikáciu so sieťovými zariadeniami sa využíva protokol SSH. Pre jednotlivé NETCONF operácie sa využívajú XML dotazy, resp. konfigurácie, ktoré využívajú dátové modely v jazyku YANG. YANG modely neobsahujú konkrétne konfiguračné dáta, ale slúžia ako šablóny pre štruktúrované dáta v iných jazykoch, ako napríklad XML, alebo JSON. Tieto modely obsahujú štruktúru konfiguračných dát, stavové dáta, RPC a notifikácie pre hlbšiu integráciu s protokolom NETCONF [5]. Niektoré YANG dátové modely sú vytvorené organizáciou IETF a niektoré sú vytvorené priamo výrobcami zariadení. Sú identifikované menom a *Uniform Resource Identifier* (URI). Príkladom štandardného dátového YANG modelu je model `ietf-interfaces` s URI urn:ietf:params:xml:ns:yang:ietf-interfaces, definovaný v RFC8343 [6] a je zobrazený nižšie.

Príklad dátového YANG modelu ietf-interfaces [6].

```
module: ietf-interfaces
    +--rw interfaces
       +--rw interface* [name]
          +--rw name                          string
          +--rw description?                  string
          +--rw type                          identityref
          +--rw enabled?                      boolean
          +--rw link-up-down-trap-enable?     enumeration {if-mib}?
          +--ro admin-status                  enumeration {if-mib}?
          +--ro oper-status                   enumeration
          +--ro last-change?                  yang:date-and-time
          +--ro if-index                      int32 {if-mib}?
          +--ro phys-address?                 yang:phys-address
          +--ro higher-layer-if*              interface-ref
          +--ro lower-layer-if*               interface-ref
          +--ro speed?                        yang:gauge64
          +--ro statistics
             +--ro discontinuity-time    yang:date-and-time
             +--ro in-octets?            yang:counter64
             +--ro in-unicast-pkts?      yang:counter64
             +--ro in-broadcast-pkts?    yang:counter64
             +--ro in-multicast-pkts?    yang:counter64
             +--ro in-discards?          yang:counter32
             +--ro in-errors?            yang:counter32
             +--ro in-unknown-protos?    yang:counter32
             +--ro out-octets?           yang:counter64
             +--ro out-unicast-pkts?     yang:counter64
             +--ro out-broadcast-pkts?   yang:counter64
             +--ro out-multicast-pkts?   yang:counter64
             +--ro out-discards?         yang:counter32
             +--ro out-errors?           yang:counter32
```

Jednotlivé uzly v jazyku YANG majú svoje meno a aj svoju prislúchajúcu hodnotu, prípadne sadu uzlov zanorenú pod nimi. V príklade YANG dátového modelu ietf-interfaces zobrazeného vyššie je vidieť, že jednotlivé uzly tohto modelu majú okrem svojho názvu a prislúchajúceho typu ich hodnoty, tak aj indikátor `rw`, alebo `ro`. Tieto indikátory označujú možnosti získavania danej hodnoty uzlu, prípadne aj možnosť zápisu novej hodnoty doň. Formát YANG definuje niekoľko typov uzlov a niektoré z nich sú vysvetlené v nasledujúcom zozname [42]:

leaf - Obsahuje samostatnú hodnotu špecifického typu.

leaf-list - Obsahuje zoznam uzlov typu leaf.

container - Tento uzol zgrupuje spolu súvisiace uzly, ktoré sú reprezentované ako jeho potomkovia.

list - Každý takýto uzol reprezentuje štruktúru, ktorá je identifikovaná unikátnym kľúčom. Môže obsahovať ľubovoľný počet potomkov, resp. podradených uzlov a to rôzneho typu.

rpc - Slúži pre definíciu RPC pre protokol NETCONF.

noticifaction - Definícia notifikácií pre protokol NETCONF.

ANSIBLE

Ansible je open-source automatizačný nástroj. Od 2015 roku je tento nástroj pod správou spoločnosti RedHat. Umožňuje konfiguráciu systémov, správu sietí, nasadzovanie softvéru a organizáciu náročnejších úloh automatizovaným spôsobom [38].

Pre orchestráciu jednotlivých spravovaných uzlov v sieti sa využívajú Ansible moduly, ktoré sa na uzly dopravujú cez protokol SSH. Moduly sa dočasne ukladajú na uzly a komunikujú s riadiacim strojom pomocou protokolu JSON na štandardnom výstupe. V okamihu, kedy Ansible uzly neriadi a nespravuje, tak žiadne výpočtové prostriedky sa nespotrebúvajú, keďže na spravovaných uzloch nebežia žiadne programy, alebo démoni. Tento prístup je výhodnejší od iných systémov pre správu konfigurácií ako Puppet, alebo CFEngine, ktoré pre svoje fungovanie využívajú aktívnych agentov, resp. démonov na spravovaných uzloch. Moduly sú v Ansible považované za jednotky činnosti. Jednotlivé moduly fungujú samostatne a sú napísané v bežných skriptovacích jazykoch ako Python, Perl, Ruby, atď. [43]. Výhodou týchto modulov je tiež aj ich vlastnosť opakovania operácie v prípade prvotných neúspechov ich vykonania a prípadne aj návrat do pôvodnej konfigurácie spravovaného uzlu. Takýmto modulom je aj modul netconf_config. Tento modul umožňuje administrátorom zasielať XML konfiguračné súbory k spravovaným zariadeniam schopných spracovávať NETCONF správy [34].

Moduly umožňujú vykonávať určité úlohy na spravovaných zariadeniach, avšak až tzv. Ansible playbooky definujú čo konkrétne sa má vykonať. Playbooky popisujú samotné konfigurácie. Vytvárajú sa vo formáte YAML. Poskytujú taktiež jednoznačné a prehľadné informácie o vykonávanom procese [33, 36]. Príklad takéhoto konfiguračného playbooku je zobrazený vo výpise nižšie.

Príklad Ansible playbooku s netconf_config modulom [33].

```
tasks:
  - name: NTP config
    netconf_config:
      host: 10.0.0.1
      username: admin
      password: admin
      xml: |
```

```
<!-- XML konfigurácia -->
            <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
                <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
                    <ntp>
                        <enabled>true</enabled>
                        <server>
                            <name>ntp1</name>
                            <udp><address>127.0.0.1</address></udp>
                        </server>
                    </ntp>
                </system>
            </config>
```

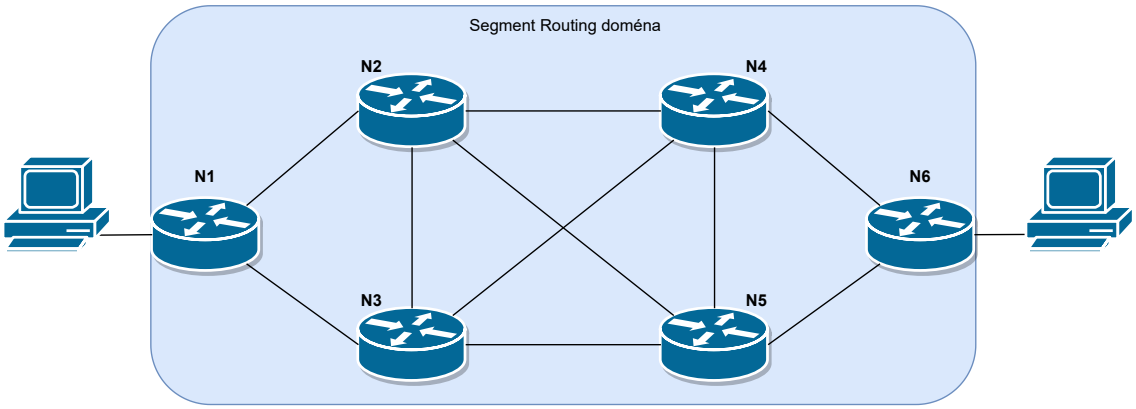
Playbook môže obsahovať viacero cieľových uzlov, ako aj viacero úloh, ktoré sa na nich majú vykonať. Zobrazený výpis demonštruje playbook pre konfiguráciu *Network Time Protocol* (NTP) nastavení cez NETCONF pre uzol s IP adresou `10.0.0.1`. Po vykonaní tejto konfigurácie sa vracia informácia o úspešnosti tejto akcie na štandardný výstup. Pre spustenie a vykonanie tohto playbooku s prischlúchajúcou konfiguráciou je nutné zadať príkaz `ansible-playbook [názov playbook YAML súboru]`. Konkrétnejšie použitie Ansible playbookov pre potreby tejto diplomovej práce bude znázornené v kapitole implementácia 4.

3 Návrh topológie a simulačných scenárov

Cieľom tejto kapitoly je vytvorenie konceptuálneho návrhu topológie, ktorá využíva technológiu SRv6. Topológia má reprezentovať transportnú časť 5G NR sietí a jej tvorba sa má zamerať na využitie platformy FD.io VPP. Prvá sekcia 3.1 sa venuje návrhu topológie a v druhej sekcii 3.2 sú zobrazené SR politiky naprieč SR doménou, ktoré reprezentujú koncept Network Slicing v 5G sieťach.

3.1 Návrh logickej topológie

Návrh logickej topológie sa odvíjal od jedného z možných variant zapojení transportných častí 5G sietí. Na obrázku Obr.3.1 je zobrazený návrh logickej topológie používaný v tejto diplomovej práci.



Obr. 3.1: Návrh logickej topológie.

Topológia 3.1 zobrazuje Segment Routing doménu, ktorá pozostáva zo šiestich smerovačov (N1 až N6) podporujúcich Segment Routing. K hraničnými smerovačom N1 a N2 sú pripojené uzly vo forme počítačov, ktoré reprezentujú externé siete, prípadne lokálne siete. Na základe týchto externých sietí sa nastavujú SR politiky na hraničných uzloch. Smerovanie v rámci SR domény je zabezpečené cez dynamický smerovací protokol a statické smerovacie cesty. Konfigurácia zariadení sa vykonáva buď manuálne, alebo prostredníctvom protokolu NETCONF s využitím programu Ansible. Detailom konfigurácie a vytváraniu virtualizovanej topológie sa venuje nasledujúca kapitola Implementácia 4.

Tvorba fyzickej topológie

Topológia je vytvorená cez platformu Docker 2.1. S využitím nástroja Docker Compose je vytvorená virtualizovaná topológia pozostávajúca z kontajnerov, ktoré re-

prezentujú jednotlivé smerovače v sieti. Okrem toho, je vytvorený kontajner pre generovanie reálnej sieťovej prevádzky za použitia generátora TRex 2.2 a kontajner pre spúšťanie NETCONF operácií pre vzdialenú konfiguráciu VPP jednotlivých smerovačov.

Adresovanie

Adresovanie v SR doméne a teda medzi jednotlivými smerovačmi je vytvorené cez protokol IPv6 a IPv4. Adresovanie externých sietí je taktiež zabezpečené cez protokol IPv6, ale aj IPv4.

Smerovanie

Smerovanie naprieč danou topológiou je možné dvoma spôsobmi. Buď cez statické trasy, alebo cez dynamický protokol. Z dôvodou komplexnosti topológie sa využíva dynamické smerovanie ako primárny spôsob smerovania. Konkrétne sa využíva dynamický smerovací protokol OSPFv3 pre smerovanie IPv6 sieťovej prevádzky. Spojazdnenie tohto dynamického smerovania je zabezpečené cez program FRRouting [1].

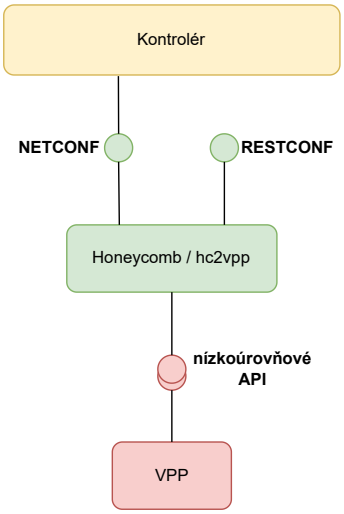
Konfigurácia SRv6

Konfigurácia SRv6 sa vykonáva nad platformou VPP, ktorá sa spúšťa nad každým kontajnerom reprezentujúcim smerovač. Konfigurácia VPP zahŕňa adresovanie rozhraní, statické smerovanie, vytvorenie lokálnych SRv6 segmentov, politík a "steeringu" paketov do jednotlivých SR politík. Túto konfiguráciu je možné zabezpečiť dvomi spôsobmi. Prvým spôsobom je manuálna konfigurácia cez príkazový riadok VPP a teda cez utilitu `vppctl`. Druhým spôsobom je implementácia VPP konfigurácie prostredníctvom protokolu NETCONF a to s využitím Docker kontajnera s programom Ansible 2.3. Jedná sa o vzdialenú konfiguráciu a nebude nutný žiaden manuálny zásah do konfigurácie priamo na jednotlivých smerovačoch, resp. ich VPP inštanciách. Takáto vzdialená konfigurácia je umožnená vďaka manažmentovnému agentu Hc2vpp.

Hc2vpp je manažmentový Java agent, ktorý je možné spojazdniť na každej vytvorenej VPP inštancii. Umožňuje YANG konfiguračným modelom upravovať konfiguráciu VPP inštancii cez NETCONF, alebo RESTCONF protokol. Hc2vpp je nástupca projektu Honeycomb a je priamo vyvíjaný pre účely podpory vzdialenej správy VPP inštancii. Prostredníctvom hc2vpp je možné konfigurovať mnohé časti VPP inštancie, ale pre potreby tejto diplomovej práce sa bude využívať len pre

[1] FRRouting `https://frrouting.org`

konfiguráciu SRv6. Hc2vpp je možné prepojiť aj s open-source SDN kontrolérom Opendailight [20]. Architektúry Hc2vpp, resp. Honeycomb je možné vidieť v nasledujúcom diagrame.



Obr. 3.2: Architektúra Hc2vpp [20].

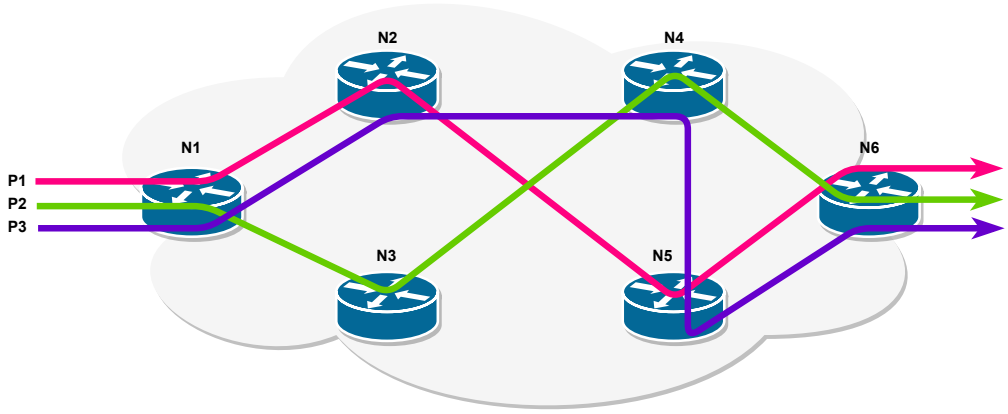
Nasledujúca sekcia sa venuje jednotlivým scenárom vytvoreným pre demonštráciu SR politík a niektorých SR modelov správania podporovaných platformou FD.io VPP.

3.2 Návrh SRv6 politík

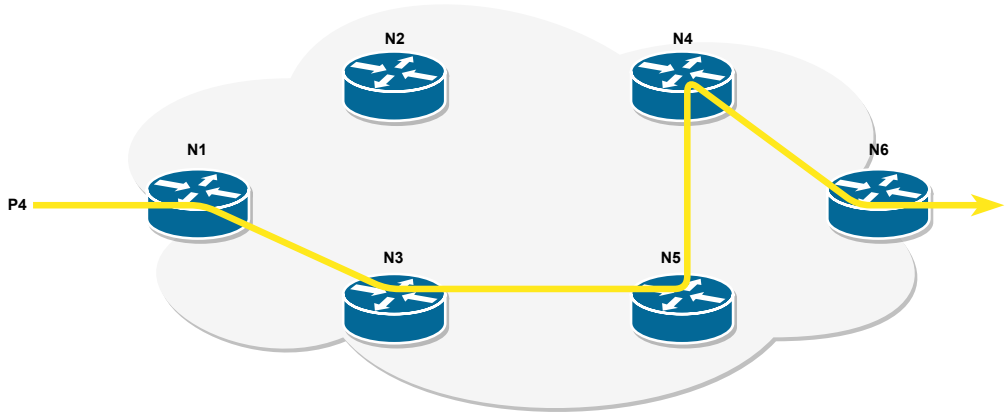
V tejto sekcii sú zobrazené SR politiky, ktoré sú implementované nad topológiou zobrazenej na obrázku Obr. 3.1. Pomocou implementácie SR politík na hraničných uzloch (N1 a N6) SR domény a lokálnych Endpoint segmentov v rámci celej SR domény je možné programovať ľubovolný prechod jednotlivých dátových tokov. Vytvorené sú politiky pre IPv4 a IPv6 dátovú prevádzku na oboch hraničných uzloch N1 a N6.

SR politiky na hraničnom uzle N1

Na hraničnom uzle N1 sú implementované štyri SR politiky. Tri z nich sú SR politiky pre IPv6 dátovú prevádzku a jedna je pre IPv4 dátovú prevádzku. V nasledujúcich dvoch obrázkoch Obr. 3.3 a Obr. 3.4 je zobrazený logický prechod dátovej prevádzky pre jednotlivé SR politiky.



Obr. 3.3: SR politiky pre IPv6 dátové prevádzky na N1 hraničnom uzle.



Obr. 3.4: SR politika pre IPv4 dátovú prevádzku na N1 hraničnom uzle.

Jednotlivé politiky pozostávajú z niekoľkých kľúčových informácií. Jedná sa o nasledujúce informácie.

1. Číselný identifikátor SR politiky (tzv. color).
2. Binding SID, resp. headend segment, ktorý identifikuje SR politiku.
3. Model správania tejto politiky. Platforma VPP podporuje tri modely Headend správania, ktoré sú popísané v teoretickej časti, viď tabuľka Tab. 1.3. V tejto práci sa z nich využili dve, a to H.Insert a H.Encaps. H.Insert model správania neenkapsuluje pôvodný paket a nepridáva celú IPv6 hlavičku aj s SR informáciami, ale namiesto toho pridáva SRH do pôvodného paketu. Tento paket však musí byť v tomto prípade paket protokolu IPv6. Naopak, H.Encaps model správania enkapsuluje pôvodný paket do novej IPv6 hlavičky spolu s SRH.
4. Zoznam segmentov v rámci SR domény v exaktnom poradí.

Prehľadný zápis aj s prislúchajúcimi hodnotami je zapísaný v nasledujúcej tabuľke Tab. 3.1.

#	BSID	Typ	Segmenty
1	fd11:1066::1	H.Encaps	End(N2)=>End(N5)=>End.DX6(N6) fd22::100=>fd55::100=>fd66::106
2	fd11:1066::2	H.Encaps	End(N3)=>End(N4)=>End.DX6(N6) fd33::100=>fd44::100=>fd66::106
3	fd11:1166::3	H.Insert	End(N2)=>End(N4)=>End(N5)=>End(N6) fd22::100=>fd44::100=>fd55::100=>fd66::100
4	fd11:1046::4	H.Encaps	End(N3)=>End(N5)=>End(N4)=>End.DX4(N6) fd33::100=>fd55::100=>fd44::100=>fd66::104

Tab. 3.1: Definícia politík na hraničnom uzle N1.

Všetky kľúčové hodnoty pre vytvorenie SR politiky sú zobrazené v tabuľke Tab. 3.1. Stĺpec so segmentami je zobrazený jednak s ich konkrétnymi adresami ako aj s ich modelom správania na príslušnom uzle. Jedinou politikou s modelom správania H.Insert je tretia politika s BSID fd11:1166::3.

Je nutné zadefinovať aj cieľové adresy, ktoré sa majú smerovať cez SRv6. K tomu slúži riadenie prevádzky (traffic steering) pomocou definície pravidiel. Tieto pravidlá sú zobrazené v nasledujúcej tabuľke Tab. 3.2.

#	BSID	IP Protokol	Cieľová sieť
1	fd11:1066::1	IPv6	AAAA::/16
2	fd11:1066::2	IPv6	BBBB::/16
3	fd11:1166::3	IPv6	CCCC::/16
4	fd11:1046::4	IPv4	48.0.0.0/24

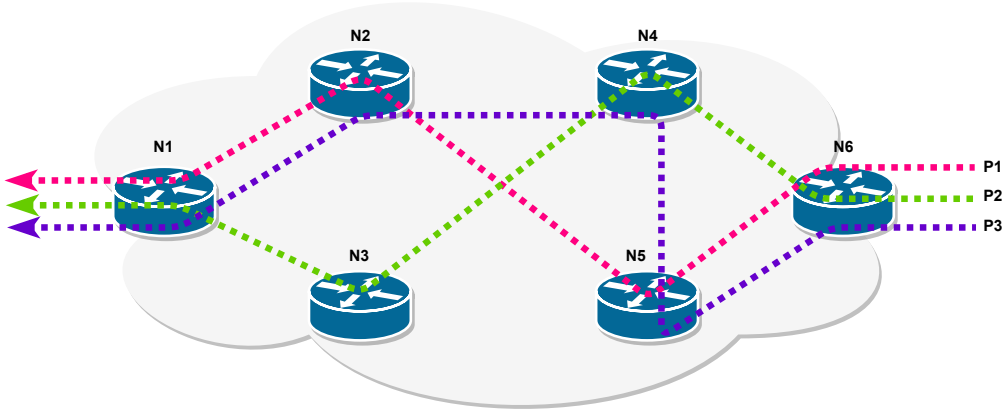
Tab. 3.2: Pravidlá smerovania IP prevádzky cez SR politiky na uzle N1.

Tabuľka Tab. 3.2 definuje to, ktorá SR politika sa má využívať pre ktoré špecifické cieľové IP adresy. Prvé tri politiky sa využívajú pre smerovanie IPv6 sieťovej prevádzky a to konkrétne pre cieľové siete AAAA::/16, BBBB::/16, CCCC::/16 a posledná, štvrtá politika sa využíva pre smerovanie IPv4 sieť 48.0.0.0/24. Nasledujúca časť sa venuje návrhu SR politík implementovaných na hraničnom uzle N6.

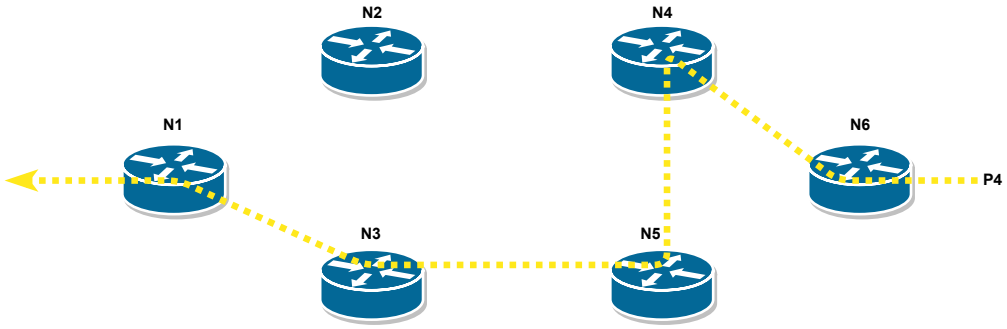
SR politiky na hraničnom uzle N6

Tak ako aj na uzle N1 tak aj na N6 sú implementované štyri SR politiky. Jedná sa v podstate o ekvivalenty politík implementovaných na uzle N1 avšak v opačnom

smere. V nasledujúcich dvoch obrázkoch Obr. 3.5 a Obr. 3.6 sú zobrazené politiky v tomto smere a ich logický prechod naprieč SR doménou.



Obr. 3.5: SR politiky pre IPv6 dátové prevádzky na N6 hraničnom uzle.



Obr. 3.6: SR politika pre IPv4 dátové prevádzky na N6 hraničnom uzle.

Informácie potrebné pre vytvorenie týchto SR politík sú zobrazené v nasledujúcej tabuľke Tab. 3.3. Aj keď sú jednotlivé politiky na uzle N6 vytvorené zrkadlovo k politikám na uzle N1, neznamená to, že je to povinnosť. Tento návrh politík bol zvolený z dôvodu prehľadnej prezentácie generovanej sieťovej prevádzky generátorom TRex.

#	BSID	Typ	Segmenty
1	fd66:6061::1	H.Encaps	End(N5)=>End(N2)=>End.DX6(N1) fd55::100=>fd22::100=>fd11::106
2	fd66:6061::2	H.Encaps	End(N4)=>End(N3)=>End.DX6(N1) fd44::100=>fd33::100=>fd11::106
3	fd66:6161::3	H.Insert	End(N5)=>End(N4)=>End(N2)=>End(N1) fd55::100=>fd44::100=>fd22::100=>fd11::100
4	fd66:6041::4	H.Encaps	End(N4)=>End(N5)=>End(N3)=>End.DX4(N1) fd44::100=>fd55::100=>fd33::100=>fd11::104

Tab. 3.3: Definícia politík na hraničnom uzle N6.

Ďalšia tabuľka Tab. 3.4 zobrazuje pravidlá smerovania IP prevádzky cez jednotlivé SR politiky.

#	BSID	IP Protokol	Cieľová sieť
1	fd66:6061::1	IPv6	A000::/16
2	fd66:6061::2	IPv6	B000::/16
3	fd66:6161::3	IPv6	C000::/16
4	fd66:6041::4	IPv4	16.0.0.0/24

Tab. 3.4: Pravidlá smerovania IP prevádzky cez SR politiky na uzle N6.

Táto kapitola zhrnula návrh logickej topológie, kľúčové body vytvorenia tejto topológie a návrh SR politík implementované nad ňou. Nasledujúca kapitola 4 sa venuje podrobnému popisu implementácie tejto topológie, ako aj popisu generovania sieťovej prevádzky a automatizácie celého konfiguračného procesu.

4 Implementácia topológie

V tejto kapitole sú kompletne zhrnuté postupy vytvárania topológie, konfigurácie VPP, generovania sieťovej prevádzky generátorom TRex a aj automatizácie celého procesu. Pre úplnosť dokumentácie, sa nasledujúca časť venuje prvému spôsobu vytvárania topológie. Avšak tento spôsob sa ukázal ako nevhodný pre cieľ tejto diplomovej práce. Funkčný a použitý spôsob vytvárania topológie je opisovaný v jednotlivých sekciách tejto kapitoly začínajúci sekciou 4.1.

Prvá verzia vytvárania topológie

Prvotné pokusy vytvorenia topológie nad platformou VPP vychádzali z oficiálnej dokumentácie VPP[1]. Cieľová topológia sa vytvárala cez virtualizované prostredie s operačným systémom Ubuntu. V tomto prostredí sa nainštalovala platform FD.io VPP. Jednotlivé smerovače boli vytvorené spustením viacerých inštancií VPP v danom samostatnom virtualizovanom systéme. Tieto inštancie VPP, reprezentujúce jednotlivé smerovače v topológii boli prepojené nasledovne:

- Prepojenia medzi smerovačmi, resp. VPP inštanciami boli realizované cez virtuálne rozhrania memif.
- Prepojenia medzi externými, resp. lokálnymi sieťami a smerovačmi do ktorých sú pripojené, boli realizované cez rozhranie typu veth.

Po vytvorení topológie a zabezpečení prepojenia jednotlivých externých sietí a smerovačov, bol rad na spojazdnení manažmentového agenta Honeycomb pre umožnenie konfigurácie VPP inštancií cez protokol NETCONF. Honeycomb však podporuje len jediné jeho spustenie nad systémom, v tomto prípade nad hostiteľskom systéme Ubuntu. Z tohto dôvodu nebolo možné spustiť Honeycomb pre jednotlivé VPP inštancie bežiace nad tým istým hostiteľským systémom.

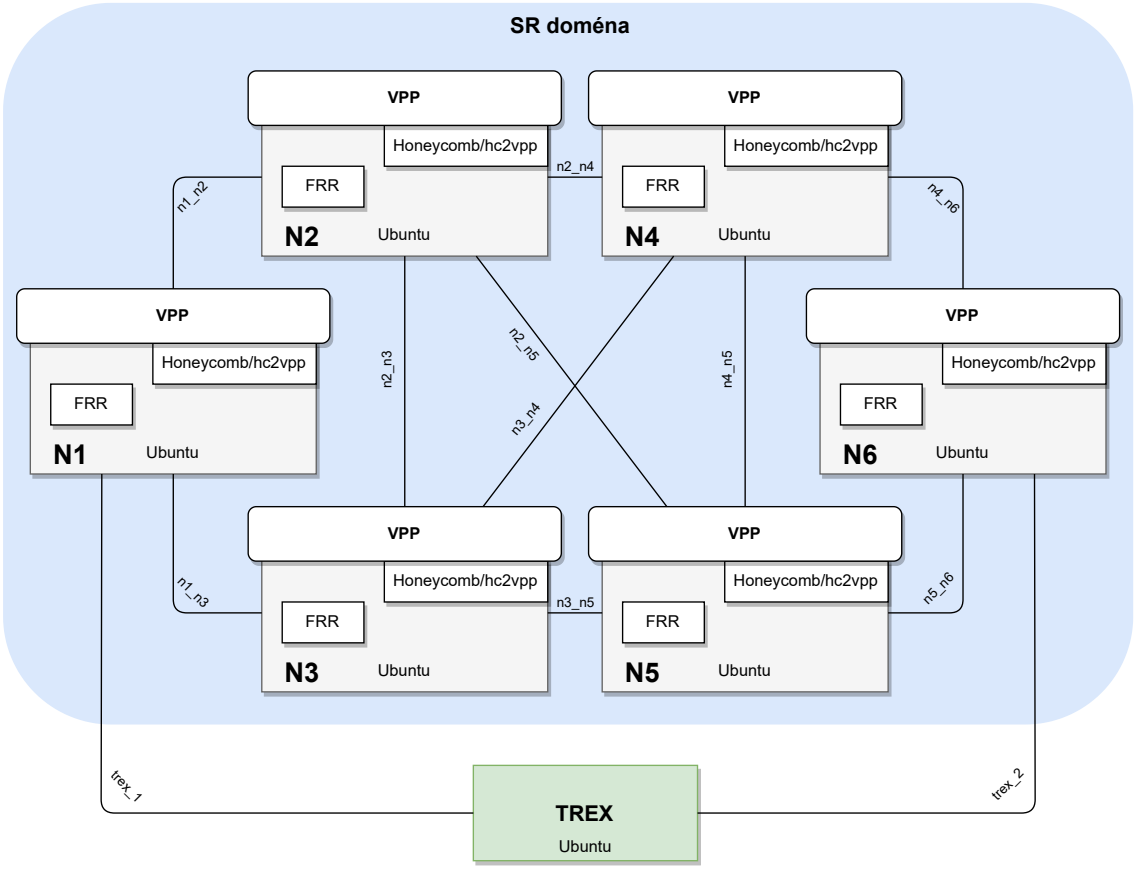
Práve preto bolo nutné vytvoriť topológiu s platformou VPP, k implementácii protokolu SRv6, iným spôsobom. Spôsob, ktorý sa javí ako najvhodnejší je cez platformu Docker. Nasledujúce sekcie tejto kapitoly opisujú finálnu a funkčnú verziu implementácie topológie so simulačnými scenármi SRv6 s využitím platformy VPP.

4.1 Prehľad topológie

Topológia je vytvorená nad platformou Docker. Jednotlivé SR uzly, resp. smerovače sú vytvorené ako Docker kontajnery s operačným systémom Ubuntu. Kontajnery sú navzájom prepojené linkami vytvorenými cez Docker Compose klienta. V nasledujúcom obrázku Obr. 4.1 je zobrazený diagram pozostávajúci z SR uzlov, Docker

[1] `https://s3-docs.fd.io/vpp/22.06/gettingstarted/progressivevpp/index.html`

kontajnerov, ktoré tvoria SR doménu. V tomto diagrame je zobrazený aj kontajner TREX, ktorý slúží k emulácií externých sietí. Tento kontajner je pripojený k hraničným uzlom N1 a N6 a slúži ku generovaniu sieťovej prevádzky v obidvoch smeroch.



Obr. 4.1: Topológia vytvorená nad platformou Docker.

Pre vytvorenie a z časti aj konfiguráciu topológie sa v prostredí Docker vytvárajú tri druhy Docker obrazov a následne ich inštancií v podobe kontajnerov. Jedná sa o nasledovné druhy obrazov.

Obraz "node" - Tento obraz slúži pre vytváranie inštancií SR uzlov N1 až N6.

Obraz "trex" - Jedná sa o obraz k vytvoreniu TREX kontajneru pre generovanie sieťovej prevádzky. Kontajnery vytvorené z tohto aj z vyššie popisovaného obrazu "node" sú spustené po celú dobu simulácie.

Obraz "ansible" - Obraz "ansible" je obraz k vytvoreniu spustiteľného kontajneru vo forme programu, ktorý slúži ku konfigurácií platformy VPP v jednotlivých "node" kontajneroch. V topológii zobrazenej v obrázku Obr. 4.1 nie je uvedený tento kontajner. To je z dôvodu, že vytvorenie tohto kontajneru nie je potrebné pre fungovanie tejto topológie, keďže slúži len ku konfigurácii VPP cez protokol NETCONF.

Pre vytvorenie obrazu je nutné v prvom rade vytvoriť súbor Dockerfile. Tieto súbory pre tvorbu jednotlivých obrazov sú detailnejšie popísané v nasledovnej časti.

Obraz "node"

V topológii je nutné vytvoriť šesť kontajnerov z obrazu node. Každý takýto kontajner musí disponovať nainštalovanou platformou VPP, agentom Honeycomb pre umožnenie prijímania NETCONF dotazov a aj programom FRRouting pre spojazdnenie dynamického smerovacieho protokolu, v tomto prípade OSPFv3 medzi jednotlivými kontajnermi. V nasledujúcom výpise je zobrazený textový súbor Dockerfile pre tvorbu node obrazu.

Dockerfile pre vytvorenie obrazu node.

```
FROM ubuntu:18.04

ARG VPP=18.10-release
ARG HONEYCOMB=1.18.10-RC2~16
ARG FRRVER=frr-stable

ARG VPP_BASE=https://nexus.fd.io/content \
    /repositories/fd.io.stable.1810.ubuntu.bionic.main/io/fd/vpp/
ARG HONEYCOMB_BASE=https://nexus.fd.io/content/repositories \
    /fd.io.stable.1810.ubuntu.xenial.main/io/fd/hc2vpp/honeycomb/

ARG PKG_VPP=${VPP_BASE}/vpp/${VPP}_amd64/vpp-${VPP}_amd64-deb.deb
ARG PKG_VPP_LIB=${VPP_BASE}/vpp-lib/${VPP}_amd64/vpp-lib-${VPP}_amd64-deb.deb
ARG PKG_VPP_PLUGINS=${VPP_BASE}/vpp-plugins/${VPP}_amd64/vpp-plugins-${VPP}_amd64-deb.deb
ARG PKG_HONEYCOMB=${HONEYCOMB_BASE}/${HONEYCOMB}_all/honeycomb-${HONEYCOMB}_all-deb.deb

RUN apt-get update && apt-get install -y \
    # Utils
    iproute2 iputils-ping net-tools vim-tiny jshon telnet curl wget ethtool vim tcpdump\
    # VPP package dependencies
    libnuma1 libssl1.0.0 libmbedcrypto1 libmbedtls10 libmbedx509-0 \
    # Honeycomb package dependencies
    openjdk-8-jre-headless \
# Install packages
&& mkdir /tmp/deb && cd /tmp/deb \
&& echo $PKG_VPP \\n $PKG_VPP_LIB \\n $PKG_VPP_PLUGINS \\n $PKG_HONEYCOMB > urls \
&& wget -i urls && dpkg --ignore-depends=vpp --ignore-depends=vpp-plugins -i *.deb \
# Reduce image size
&& cd / && rm -rf /var/lib/apt/lists/* /tmp/deb \

# Installation of FFRouting
&& apt-get update && apt-get install -y gnupg2 \
&& curl -s https://deb.frrouting.org/frr/keys.asc | apt-key add - \
&& apt-get update && apt-get install -y lsb-release && apt-get clean all \
&& echo deb https://deb.frrouting.org/frr $(lsb_release -s -c) ${FRRVER} | tee -a
     /etc/apt/sources.list.d/frr.list \
&& apt-get update && apt-get install -y frr frr-pythontools

COPY entrypoint.sh /
CMD ["/entrypoint.sh"]
```

V súbore Dockerfile je prvým príkazom príkaz FROM, ktorý definuje využívanie operačného systému Ubuntu vo verzii 18.04 týmto typom kontajneru. Nasledujúce riadky súboru definujú argumenty pomocou príkazov ARG. Definované argumenty obsahujú verzie platformy VPP, agentu Honeycomb, alebo verziu programu FRRouting. Ďalšími argumentmi sú definície URL adries k archívom použitým pri inštalácií VPP a Honeycomb. Ďalším príkazom je príkaz RUN, ktorý obsahuje širší zoznam príkazov zoradených za sebou. Tieto príkazy inštalujú utility do vytváraného systému Ubuntu, platformu VPP s agentom Honeycomb a následne aj program FRRouting so všetkými ich závislosťami. Predposledným príkazom je príkaz COPY, ktorý kopíruje súbor entrypoint.sh do systému Ubuntu. Posledným príkazom je príkaz CMD, ktorý spúšťa shell skript entrypoint.sh pri spustení kontajnera vytvoreného z obrazu node. Postup prekladu a spúšťania kontajnerov je popísaný v sekcii 4.2.

Obraz "trex"

Obraz trex slúži k vytvoreniu kontajneru, ktorého účelom je generovanie sieťovej prevádzky a k emulovaniu externých sietí v rámci topológie. V nasledujúcom výpise je zobrazený obsah textového súboru Dockerfile pre tento obraz.

Dockerfile pre vytvorenie obrazu trex.

```
FROM ubuntu:focal-20210119

ARG TREX_VERSION=v2.88

RUN apt-get update \
    && apt-get -y install --no-install-recommends \
    iproute2 iputils-ping nano net-tools netbase pciutils \
    python3 python3-distutils strace wget vim tcpdump \
    && apt-get autoclean \
    && apt-get autoremove -y \
    && rm -rf /var/lib/apt/lists/*

RUN wget --no-check-certificate \
    https://trex-tgn.cisco.com/trex/release/${TREX_VERSION}.tar.gz && \
    tar -zxvf ${TREX_VERSION}.tar.gz -C / && \
    chown root:root /${TREX_VERSION}  && \
    rm ${TREX_VERSION}.tar.gz

COPY trex_cfg.yaml /etc/trex_cfg.yaml
WORKDIR /${TREX_VERSION}

CMD tail -f /dev/null
```

Z vyššie uvedeného výpisu je vidieť, že tento kontajner beží taktiež nad operačným systémom Ubuntu. V ďalšom kroku sa inštalujú cez príkazy RUN podporné utility, programy a aj program TRex. Spustenie tohto kontajneru je bližšie popísané v sekcii 4.2 a opis generovania sieťovej prevádzky je popísaný v sekcii 4.4.

Obraz "ansible"

Tento obraz slúži k vytvoreniu kontajneru spúšťaného ako program pre konfiguráciu VPP platforiem na jednotlivých SR uzloch prostredníctvom protokolu NETCONF. Kontajner sa spúšťa len za týmto účelom a po vykonaní potrebných konfigurácií sa kontajner ukončí. Obsah súboru Dockerfile tohto obrazu je zobrazený v nasledujúcom výpise.

Dockerfile pre vytvorenie obrazu ansible.

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y ansible python-ncclient wait-for-it && \
        rm -rf /var/lib/apt/lists/* && \
        echo "[local]\nlocalhost ansible_connection=local" > /etc/ansible/hosts
COPY entrypoint.sh *.yaml /
ENTRYPOINT ["/entrypoint.sh"]
```

Súbor Dockerfile zobrazený vyššie definuje ako operačný systém Ubuntu prostredníctvom príkazu FROM. Následne po inštalácií programu ansible a príslušných závislostí v podobe Python knižnice nccient [2] sa prostredníctvom príkazu RUN kopíruje súbor entrypoint.sh a všetky súbory formátu YAML do koreňového adresára tohto kontajneru. Kopírované súbory sú sa nachádzajú v tom istom adresári ako aj súbor Dockerfile na hostiteľskom systéme, kde je platforma Docker inštalovaná. Posledným príkazom je príkaz ENTRYPOINT, ktorý definuje spustenie skriptu entrypoint.sh po spustení vytvoreného kontajneru.

4.2 Vytvorenie topológie

Pre vytvorenie topológie je využívaný klient Docker Compose. To z dôvodu jednoduchej manipulácie s celou topológiou, resp. so všetkými kontajnermi. Pre preklad jednotlivých obrazov, vytvorenie kontajnerov a následné spustenie celej topológie je nutné vytvorenie súboru docker-compose.yaml. Táto sekcia sa venuje obsahu tohto súboru ako aj následným Docker Compose príkazom k vytvoreniu topológie na úrovni kontajnerov.

Celkovo sa vytvára osem kontajnerov. Šesť z nich je určených pre reprezentáciu smerovačov v SR doméne, jeden kontajner pre TRex generátor a jeden kontajner pre spúšťanie Ansible konfigurácií. V súbore docker-compose.yaml je preto možné vidieť osem položiek v sekcii 'services'. Vytvárané kontajnery N1 až N6 a trex sú navzájom prepájané linkami definovanými taktiež v tomto súbore. V diagrame na obrázku Obr. 4.1 je možné vidieť názvy jednotlivých liniek, resp. sietí, ktoré prepájajú tieto

[2] https://docs.ansible.com/ansible/latest/network/user_guide/platform_netconf_enabled.html

kontajnery. V nasledujúcej tabuľke Tab. 4.1 je zobrazené adresovanie naprieč týmito linkami.

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

Tab. 4.1: Zoznam sietí medzi kontajnermi s ich adresovaním.

Kompletný obsah súboru docker-compose.yaml je zobrazený v prílohe A.1. Konfigurácia tohto súboru, resp. hodnoty využité v ňom sú do značnej miery vysvetlené v sekcii 2.1. Za zmienku však stojí nasledovná hodnota. Nasledovný výpis textu zo súboru docker-compose.yaml prislúcha ku kontajneru ansible.

```
ansible:
    depends_on: ["node1", "node2", "node3", "node4", "node5", "node6"]
    build: ansible
```

Kľúčová `depends_on` vyjadruje závislosť medzi kontajnermi, resp. službami [3]. Keďže kontajner Ansible je spúšťaný len pri potrebe konfigurácie VPP na jednotlivých uzloch topológie, je nutné aby tieto uzly boli úspešne vytvorené pred jeho spustením. Tvorba kontajneru Ansible je vďaka tomuto riadku zahájená až po úspešnom vytvorení všetkých uzlov N1 až N6.

Po kompletnom vytvorení súboru docker-compose.yaml, ako aj súborov Dockerfile pre príslušné Docker obrazy je možné tieto obrazy vytvoriť, ako aj spustiť ich inštancie vo forme kontajnerov. Nasledovnými príkazmi sa vytvoria Docker obrazy a vytvoria kontajnery N1 až N6 a aj kontajner trex spolu s ich prepojeniami.

```
docker-compose build
docker-compose up -d node1 node2 node3 node4 node5 node6 trex
```

[3] `https://docs.docker.com/compose/compose-file/compose-file-v3/#depends_on`

Tieto príkazy je nutné spustiť v tom adresári, kde sa nachádza aj súbor docker-compose.yaml. Argument '-d' pri príkaze `docker-compose up` spúšťa jednotlivé kontajnery na pozadí. V prípade neuvedenia názvov jednotlivých kontajnerov na konci tohto príkazu, sa vytvoria a spustia všetky kontajnery uvedené v súbore docker-compose.yaml. Keďže automatizovanej konfigurácii sa venuje až sekcia 4.4, kontajner ansible sa v tomto prípade vynechal a teda konfigurácia VPP sa momentálne nevykoná.

V tomto bode je vytvorená topológia tak, ako je zobrazená v diagrame na obrázku Obr. 4.1. Avšak pre spojazdnenie topológie vrátanie smerovania a konfigurácie VPP je nutné doplniť konfiguráciu jednotlivých uzlov. Konfigurácii VPP sa venuje nasledovná sekcia 4.3. Následne je nutné ešte doplniť konfiguráciu v súvislosti s generovaním sieťovej prevádzky z trex kontajneru, čomu sa venuje sekcia 4.4.

4.3 Konfigurácia VPP

Každý z node kontajnerov disponuje inštalovanou platformou VPP. VPP predstavuje virtuálny smerovač nad hostiteľským systémom, v tomto prípade systémom Ubuntu. Platforma VPP sa pre potreby tejto diplomovej práce využíva k spojazdneniu SRv6 nad topológiou zobrazenou na obrázku Obr. 4.1. Jednotlivé pakety prichádzajúce do jednotlivých kontajnerov N1 až N6, sa v prvom rade spracúvajú systémom Ubuntu. Ten ich následne na základe smerovacích pravidiel posiela na platformu VPP k ďalšiemu spracovávaniu cez SRv6, resp. k pridávaniu potrebných SRv6 informácií.

Prepojenie Ubuntu a VPP

Z tohto dôvodu je nutné vytvoriť prepojenie medzi systémom Ubuntu a platformou VPP. To sa docieli vytvorením virtuálneho rozhrania typu tap. Zobrazené nasledujúce príkazy sa vykonávajú nad kontajnerom N1, ale je nutné vykonať ekvivalenty týchto príkazov nad každým kontajnerom typu node.

Manuálna VPP konfigurácia sa vytvára pomocou nástroja `vppctl`. Jedná sa o príkazový riadok vytvorený pre platformu VPP. K spusteniu tohto príkazového riadku je nutné zadať nasledujúci príkaz z hostiteľského systému Docker platformy.

```
docker-compose exec node1 vppctl
```

Vyššie uvedený príkaz prepne terminál do kontextu vppctl. Následne je možné zadávať VPP príkazy ku konfigurácii platformy VPP. Nižšie uvedené VPP príkazy vytvárajú už rozhranie typu tap, k prepojeniu systému Ubuntu a VPP v kontajneri N1.

```
create tap id 0 hw-addr 00:00:00:00:10:10 host-ip6-addr fd10::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd10::10/64
set interface ip address tapcli-0 192.168.10.10/24
```

Tieto príkazy vytvoria rozhranie s názvom `tapcli-0` na strane VPP s virtuálnym prepojením na rozhranie s názvom `vpp` na strane systému Ubuntu kontajneru N1. Na strane Ubuntu je následne nutné doplniť konfiguráciu IPv4 adresy pre toto rozhranie, keďže IPv6 adresa je definovaná už pri vytváraní tohto rozhrania prvým príkazom vo VPP. To zabezpečí nasledujúci príkaz.

```
ip addr add 192.168.10.1/255.255.255.0 dev vpp
```

V tomto momente je vytvorené virtuálne prepojenie medzi hostiteľským systémom Ubuntu a platformou VPP v kontajneri N1. Následne je možné prejsť ku konfigurácií SRv6.

Konfigurácia SRv6

Konfigurácia SRv6 pozostáva z troch častí. V prvej časti sa vytvárajú lokálne segmenty a následne sa na hraničných uzloch vytvárajú SR politiky spolu s pravidlami smerovania prichádzajúcich paketov prostredníctvom SRv6, cez konkrétnu politiku na základe ich cieľovej adresy (traffic steering). Tak ako aj v predošlej časti, tak aj v tejto sa bude jednať o konfiguráciu kontajnera N1.

V prípade kontajneru N1 sa vytvárajú tri lokálne segmenty s modelmi správania End, End.DX4 a End.DX6. Dekapsulačné typy správania End.DX4 a End.DX6 je možné využiť na tomto uzle z dôvodu, že uzol N1 je hraničným uzlom a terminuje politiky konfigurované na uzle N6. V určitých prípadoch sa teda jedná o posledný uzol, ktorý musí odstrániť IPv6 hlavičky pridané z dôvodu úspešného smerovania paketov naprieč SR doménou a následne tak zasiela už originálne pakety smerom k pôvodnému cieľu. Segmenty na uzle N1 sa konfigurujú nasledovnými VPP príkazmi v termináli vppctl.

```
sr localsid address fd11::100 behavior end
sr localsid address fd11::104 behavior end.dx4 tapcli-0 192.168.10.1
sr localsid address fd11::106 behavior end.dx6 tapcli-0 fd10::1
```

Prvým segmentom je segment s identifikátorom fd11::100 a modelom správania End. Ďalším segmentom je segment s modelom správania End.DX4. Keďže tento segment odstraňuje extra IPv6 hlavičku pridanú hraničným uzlom odosiela pôvodný IPv4 paket, je nutné pridať smerovacie informácie. V tomto prípade je definované východzie rozhranie tapcli-0 a next-hop IPv4 adresa hostiteľského systému Ubuntu. Ekvivalentná konfigurácia je vytvorená aj pri poslednom segmente, avšak pre IPv6 sieťovú prevádzku. Pre kontrolu správnosti konfigurácie je možné zobraziť konfigurované lokálne segmenty nasledovným príkazom.

```
show sr local
```

Kompletný zoznam implementovaných segmentov na jednotlivých SR uzloch je zobrazený v nasledujúcej tabuľke Tab. 4.2.

Názov SR uzlu	**SID**	**Model správania**
N1	fd11::100	End
	fd11::104	End.DX4
	fd11::106	End.DX6
N2	fd22::100	End
N3	fd33::100	End
N4	fd44::100	End
N5	fd55::100	End
N6	fd66::100	End
	fd66::104	End.DX4
	fd66::106	End.DX6

Tab. 4.2: Zoznam implementovaných segmentov.

Segmenty sú implementované na všetkých SR uzloch. SR politiky sú však implementované len na hraničných uzloch N1 a N6. Nasledujúca časť popisuje implementáciu SR politík na uzle N1. Detailnejší popis všetkých politík je v návrhovej sekcii 3.2.

Konfigurácie štyroch SR politík na uzle N1 je zobrazená v nasledujúcom výpise.

```
sr policy add bsid fd11:1066::1 next fd22::100 next fd55::100 next fd66::106 encap
sr policy add bsid fd11:1066::2 next fd33::100 next fd44::100 next fd66::106 encap
sr policy add bsid fd11:1166::3 next fd22::100 next fd44::100 next fd55::100 next fd66::100 insert
sr policy add bsid fd11:1046::4 next fd33::100 next fd55::100 next fd44::100 next fd66::104 encap
```

Každá politika má definovaný svoj identifikátor v podobe tzv. Binding SID (bsid) a zoznam segmentov naprieč SR doménou. Tri z týchto politík majú argumentom `encaps` definovaný model správania H.Encaps a jedna z nich má definované správania H.Insert argumentom `insert`. V prípade H.Encaps správania sa cieľová adresa novej IPv6 hlavičky pridávanej k pôvodnému paketu získava z aktuálne aktívneho segmentu. Čo sa týka zdrojovej IPv6 adresy tejto novo pridanej IPv6 hlavičky, je nutné túto adresu definovať nasledujúcim príkazom.

```
set sr encaps source addr fd10::1
```

Tento príkaz má účinok nad všetkými takto novo vytvorenými paketmi a teda sa jedná o zdrojovú adresu, ktorá sa bude využívať pri všetkých SR politikách s H.Encaps správaním. V prípade H.Insert správania sa zdrojová adresa nemení a zostáva pôvodná, ako aj pôvodná IPv6 hlavička. V tomto prípade sa mení len cieľová

adresa a rozširujúca SRH hlavička sa pridáva priamo medzi pôvodu IPv6 hlavičku a hlavičku transportného protokolu daného paketu. Pre kontrolu konfigurácie SR politík je možné využiť nasledujúci príkaz ich zobrazenia.

```
show sr policies
```

Po definícií SR politík, je možné definovať pravidlá, ktoré smerujú príchodzie pakety prostredníctvom SR politík naprieč SR doménou na základe ich pôvodných cieľových adries. Jedná sa o tzv. traffic steering. V nasledujúcom výpise sú zobrazené implementované pravidlá v kontajneri N1.

```
sr steer l3 AAAA::/16 via bsid fd11:1066::1
sr steer l3 BBBB::/16 via bsid fd11:1066::2
sr steer l3 CCCC::/16 via bsid fd11:1166::3
sr steer l3 48.0.0.0/24 via bsid fd11:1046::4
```

Z vyššie uvedeného výpisu je vidieť, že prvé tri politiky sa využívajú pre IPv6 prevádzku do sietí AAAA::/16, BBBB::/16, CCCC::/16 a posledná politika sa využíva pre cieľovú IPv4 sieť 48.0.0.0/24. Zobrazenie týchto pravidiel je možné cez nasledujúci príkaz.

```
show sr steering-policies
```

Táto časť sa venovala konfigurácii SRv6, avšak je nutné spojazdniť aj smerovanie naprieč SR doménou, pre úspešné smerovanie paketov s cieľovými adresami nastavenými konkrétnymi identifikátormi segmentov.

Smerovanie v rámci SR domény

Z pohľadu smerovania v rámci SR domény a teda medzi jednotlivými smerovačmi N1 až N6 je nutné zabezpečiť viditeľnosť všetkých segmentov konfigurovaných na týchto uzloch. V rámci SR domény nie je nutné zabezpečiť smerovanie generovaných paketov na základe ich pôvodných adries, keďže v SR doméne sú cieľovými adresami vždy identifikátory segmentov. Toto je možné docieliť dvoma spôsobmi.

1. Prvou možnosťou je konfigurácia statických trias, ktoré VPP priamo podporuje. Pri tejto variante je možné vytvoriť virtuálne Ethernet rozhrania, ktoré priamo prepájajú VPP inštancie s rozhraniami ich kontajneru. Jedná sa o VPP rozhrania typu `host`, alebo `AF_PACKET`. Po vytvorení týchto rozhraní je možné vkladať pakety na linky medzi kontajnermi priamo z platformy VPP. Konfigurácia takýchto rozhraní vo VPP je znázornená v nasledovných príkazoch.

   ```
   create host-interface name eth0 hw-addr 00:00:00:00:12:10
   create host-interface name eth1 hw-addr 00:00:00:00:13:10
   ```

   Kompletná konfigurácia musí obsahovať aj doplňujúce príkazy s príslušnými IP adresami a prípadne ďalšími konfiguráciami týchto rozhraní. Následne je možné definovať statické smerovacie trasy priamo vo VPP. Pre komplexnosť

topológie ale nie je táto varianta ideálnou a flexibilnou možnosťou a vhodnejšou možnosťou sa javí dynamické smerovanie, ktoré sa aj využilo v tejto diplomovej práci.

2. Druhou možnosťou je implementácie smerovania v rámci SR domény prostredníctvom dynamického smerovania. VPP však nepodporuje dynamické smerovacie protokoly a teda je nutné dynamické smerovanie spojazdniť na úrovni kontajnerov, resp. na úrovni systémov Ubuntu. K tomu sa využil program FRRouting. Dynamickým smerovacím protokolom využitým v tejto práci je smerovací protokol *Open Shortest Path First routing protocol for IPv6* (OSPFv3). Konfigurácií tejto varianty smerovania v SR doméne sa zvyšok tejto sekcie.

Keďže vybranou variantou smerovania pre túto diplomovú prácu je smerovanie dynamické a to cez program FRRouting, je nutné smerovať sieťovú prevádzku z VPP späť na systém Ubuntu. VPP teda spracuje prijatú sieťovú prevádzku, pridá do jej jednotlivých paketov potrebné SRv6 informácie a takto upravené pakety posiela na svoj hostiteľský systém Ubuntu. Konfigurácia tohto správania je zabezpečená konfiguráciou statickej trasy nasledujúcim VPP príkazom. Opäť, jedná sa o konfiguráciu na N1 uzle a ekvivalentná konfigurácia sa vykonáva na všetkých SR uzloch.

```
ip route ::/0 via fd10::1
```

Vyššie zobrazený VPP príkaz pridáva východziu smerovaciu trasu pre všetky cieľové IPv6 adresy smerom k systému Ubuntu. Všetky upravené pakety s SR informáciami odosielané z platformy VPP majú cieľovú adresu jedného z implementovaných segmentov v rámci SR domény. Smerovacia tabuľka systému Ubuntu teda musí obsahovať záznam trasy k všetkým týmto segmentom. Pre zabezpečenie tejto nutnosti je spustený a konfigurovaný program FRRouting na každom uzle N1 až N6.

Inštalácia FRRouting sa vykonala už pri vytváraní kontajnerov typu node. Jednotlivé príkazy inštalácie FRRouting je teda možné nájsť v príslušnom súbore Dockerfile. FRRouting podporuje rozsiahle množstvo dynamických smerovacích protokolov. Po spustení FRRouting ako služby, bežia jednotlivé protokoly ako démoni na danom systéme. V tomto prípade sa využíva len protokol OSPFv3 a pre jeho aktiváciu je nutné ešte pred samotným spustením FRRouting povoliť tento protokol v súbore `/etc/frr/daemons`, keďže všetky dynamické protokoly sú spočiatku zakázané. Následne je možné spustiť FRRouting nasledovným príkazom v príkazovom riadku pre systém Ubuntu.

```
service frr start
```

Po spustení služby FRRouting je možné konfigurovať systém cez príkazový riadok FRRouting, cez utilitu vtysh, ktorá sa spúšťa príkazom `vtysh` v systéme Ubuntu. Cieľom dynamického smerovania v SR doméne je propagácia IPv6 adresných priestorov vyhradených pre identifikátory segmentov. Na každom SR uzle majú všetky

identifikátory lokálne definovaných segmentov spoločný IPv6 prefix, rovnakú 'LOCATOR' časť 1.2. Avšak systém Ubuntu o tomto prefixe nemá doposiaľ informáciu a preto je nutné definovať statickú trasu smerom k VPP nasledujúcim príkazom.

```
ip route add fd11::/64 via fd10::10 dev vpp
```

V tomto momente je v systéme Ubuntu definovaná trasa k lokálne definovaným segmentom vo VPP. Ďalším krokom je konfigurácia FRRouting k propagácii prefixov vyhradených pre definíciu lokálnych segmentov. V nasledujúcom výpise je zobrazená kompletná konfigurácia FRRouting na uzle N1 cez utilitu vtysh.

```
conf t
ipv6 prefix-list node1_SIDs permit fd11::/64
route-map VPP_SIDs permit 1
  match ipv6 address prefix-list node1_SIDs
router ospf6
  ospf6 router-id 1.1.1.1
  redistribute kernel route-map VPP_SIDs
int eth0
  ipv6 ospf6 area 0
int eth1
  ipv6 ospf6 area 0
```

FRRouting má prístup k všetkým sieťovým komponentom hostiteľského systému Ubuntu. Konfigurácia cez utilitu vtysh pripomína príkazový riadok Cisco zariadení. Tiež sa jednotlivé časti konfigurujú prostredníctvom kontextovo oddelených častí. Prvým riadkom sa kontext príkazového riadku prepína do konfiguračného kontextu. Následne sa definuje prefix-list pre IPv6 prefix fd11::/64 vyhradený pre segmenty na uzle N1. Následne sa tento prefix-list vkladá do route-map. Route-map sa vytvorila z dôvodu propagácie tohto prefixu zo smerovacích trias definovaných systémom Ubuntu v SPFv3 procese medzi jednotlivými uzlami. OSPFv3 identifikátor uzla N1 je definovaný na 1.1.1.1. Poslednou časťou konfigurácie je priradenie rozhraní kontajneru N1 do procesu OSPFv3. Celá topológia a všetky jej uzly spadajú do hlavnej oblasti 0 (area 0). Obdobná konfigurácia sa vykonáva na všetkých uzloch N1 až N6.

Pre overenie funkčnosti konfigurácie OSPFv3 je možné skontrolovať nadviazané OSPFv3 susedstvá ako aj úspešnosť propagácie IPv6 prefixov určených pre lokálne segmenty nasledujúcimi príkazmi v príkazovom riadku vtysh.

```
N1# show ipv6 ospf6 neighbor
Neighbor ID     Pri    DeadTime    State/IfState         Duration I/F[State]
2.2.2.2           1    00:00:35     Full/DR              00:01:43 eth0[BDR]
3.3.3.3           1    00:00:34     Full/BDR             00:01:29 eth1[DR]

N1# show ipv6 ospf6 route external-2
*N E2 fd22::/64                      fe80::42:c0ff:fea8:c66      eth0 00:01:59
*N E2 fd33::/64                      fe80::42:c0ff:fea8:d67      eth1 00:01:50
*N E2 fd44::/64                      fe80::42:c0ff:fea8:c66      eth0 00:01:45
                                     fe80::42:c0ff:fea8:d67      eth1
*N E2 fd55::/64                      fe80::42:c0ff:fea8:c66      eth0 00:01:36
                                     fe80::42:c0ff:fea8:d67      eth1
*N E2 fd66::/64                      fe80::42:c0ff:fea8:c66      eth0 00:01:10
                                     fe80::42:c0ff:fea8:d67      eth1
```

V tomto momente je kompletne konfigurovaná SR doména vrátane celej konfigurácie SRv6 a smerovania. Ďalšia sekcia 4.4 sa venuje doplneniu konfigurácie k zabezpečeniu úspešného generovania sieťovej prevádzky ako aj postupu spustenia generovania tejto prevádzky generátorom TRex.

4.4 Generovanie sieťovej prevádzky

Testovaná sieť generátorom TRex musí generovanú sieťovú prevádzku smerovať z jeho TX rozhrania na jeho RX rozhranie. Je možné generovať sieťovú prevádzku aj obojsmerne a to je prípad v tejto diplomovej práci. Kontajner TRex s operačným systémom Ubuntu je pripojený k hraničným uzlom N1 a N6, tak ako je to znázornené v obrázku Obr. 4.1. Jedným smerom generovania je generovanie cez uzol N1, následne sa generovaná sieťová prevádzka upravuje podľa potrieb SRv6 v platforme VPP na uzle N1. Takto upravená sieťová prevádzka sa posiela naprieč SR doménou podľa vložených segmentov do jednotlivých paketov na základe SR politiky a prijíma sa na uzle N6. Tento uzol následne posiela už pôvodné pakety smerom k RX rozhraniu TRex v tomto prípade. Generovanie opačným smerom je obdobné. TRex počas generovania sieťovej prevádzky zobrazuje štatistiky o odoslaných a prijatých paketoch na oboch svojich rozhraniach. Smerovaniu generovaných paketov sa venuje nasledujúca časť.

Smerovanie generovaných paketov

Pre generovanie sieťovej prevádzky generátorom TRex sa pre potreby tejto diplomovej práce vytvorili tzv. profily sieťovej prevádzky (traffic profiles). Jedná sa o súbory v jazyku Python. V nich sú definované pakety prostredníctvom Python Scapy, ktoré sa majú generovať. Nasledujúca tabuľka Tab. 4.3 zobrazuje zdrojové a cieľové IP adresy generovaných sieťových tokov.

Protokol	Zdrojový prefix	Cieľový prefix
IPv6	A000::/64	AAAA::/64
IPv6	B000::/64	BBBB::/64
IPv6	C000::/64	CCCC::/64
IPv4	16.0.0.0/24	48.0.0.0/24

Tab. 4.3: Zoznam generovaných IPv6 a IPv4 sieťových tokov.

Tabuľka Tab. 4.3 zobrazuje generované sieťové toky v prípade ich generovania cez rozhranie pripojené k uzlu N1. V prípade generovania opačnou stranou, cez rozhranie pripojené k uzlu N6, budú zdrojové a cieľové prefixy vymenené. Kompletný kód súborov s využívanými profilmi je možné vidieť v prílohách tohto dokumentu C.2.

Takto generované toky je nutné vhodne smerovať. Nasledovné výpisy zobrazujú konfiguráciu smerovania týchto tokov na uzle N1 a N6.

Uzol N1 (Ubuntu)

```
ip route add 48.0.0.0/24 via 192.168.10.10 dev vpp
ip route add 16.0.0.0/24 via 192.168.91.99 dev eth3
ip route add aaaa::/16 via fd10::10 dev vpp
ip route add a000::/16 via fd91::99 dev eth3
ip route add bbbb::/16 via fd10::10 dev vpp
ip route add b000::/16 via fd91::99 dev eth3
ip route add cccc::/16 via fd10::10 dev vpp
ip route add c000::/16 via fd91::99 dev eth3
```

Uzol N6 (Ubuntu)

```
ip route add 16.0.0.0/24 via 192.168.60.60 dev vpp
ip route add 48.0.0.0/24 via 192.168.92.99 dev eth3
ip route add a000::/16 via fd60::60 dev vpp
ip route add aaaa::/16 via fd92::99 dev eth3
ip route add b000::/16 via fd60::60 dev vpp
ip route add bbbb::/16 via fd92::99 dev eth3
ip route add c000::/16 via fd60::60 dev vpp
ip route add cccc::/16 via fd92::99 dev eth3
```

Smerovanie týchto tokov je zabezpečené statickými trasami. Lokálne prefixy sú smerované cez rozhranie `eth3`, priamo k TRex kontajneru. Vzdialené prefixy sú smerované na rozhranie `vpp`, smerom k platforme VPP v danom kontajneri. Tým sa zabezpečí spracovávanie týchto tokov cez SRv6, keďže pre dané prefixy sú vo VPP nastavené pravidlá smerovania prevádzky cez konkrétne SR politiky.

Spustenie generátora TRex

V tejto diplomovej práci sa využíva TRex generátor v bezstavovom móde. Spustenie programu TRex v bezstavovom móde je zabezpečené už pri spúšťaní kontajneru

TRex. V súbore docker-compose.yaml je v sekcii služby trex uvedený parameter `command`. Jeho hodnota je `["./t-rex-64", i", -cfg", "/etc/trex_cfg.yaml"]`. Jedná sa o príkaz, ktorý sa vykoná pri spustení TRex kontajneru. To isté je možné vykonať aj manuálne z príkazového riadku systému Ubuntu v tomto kontajneri.

```
./t-rex-64 -i --cfg /etc/trex_cfg.yaml
```

Týmto príkazom sa spúšťa samotný TRex generátor v bezstavovom móde ako služba v danom kontajneri. Avšak samotné generovanie sieťovej pravádzky sa vykonáva z TRex konzoly. TRex konzolu je možné spustiť nasledovným príkazom.

```
root@3091b370e659:/v2.88# ./trex-console
Using 'python3' as Python interpeter
Connecting to RPC server on localhost:4501                   [SUCCESS]
Connecting to publisher server on localhost:4500             [SUCCESS]
Acquiring ports [0, 1]:                                      [SUCCESS]

Server Info:
Server version:   v2.88 @ STL
Server mode:      Stateless
Server CPU:       1 x AMD Ryzen 9 3900X 12-Core Processor
Ports count:      2 x 10.0Gbps @ Unknown

-=TRex Console v3.0=-
Type 'help' or '?' for supported actions

trex>
```

Po spustení tejto konzoly sa sprístupnia príkazy, ktorými je možné spúšťať, dynamicky upravovať, alebo zastavovať generovanie. Niektoré z používaných príkazov sú zobrazené v nižšie uvedenom výpise.

```
start -f /ipv6_stl_profile.py
update -m 50mbps
pause
stats
release
tui
stop
```

Jednotlivé príkazy sú vysvetlené nasledovne.

start – Spúšťa generovanie sieťovej prevádzky definovanej v súbore ipv6_stl_profile.py.

update – Upravuje parametre generovania. V tomto prípade sa jedná o úpravu rýchlosti generovania na 50 Mb/s.

pause – Pozastavuje generovanie.

stats – Zobrazenie štatistík o doposiaľ vygenerovanej prevádzke.

release – Opätovné spustenie pozastaveného generovania.

tui – Spustenie interaktívneho módu, v ktorom sa real-time zobrazujú a aktualizujú štatistiky generovania.

stop – Zastavenie generovania.

clear – Vymazanie štatistík o prenesených paketoch.

Týmto je postup implementácie zavŕšený, ako aj postup spustenia generovania sieťových tokov. K výstupom z generovania ako aj k detailnému pohľadu na niektoré pakety zachytené v určitých bodoch topológie 4.1 sa venuje kapitola 5. Avšak ešte pred ňou je pár viet venovaných automatizácií celého procesu vytvárania funkčnej topológie pripravenej na spustenie generovania sieťovej prevádzky v nasledujúcej sekcii 4.5.

4.5 Automatizácia konfigurácie

Automatizácia vytvárania a konfigurácie tejto topológie je hlavne časovo výhodná. Táto sekcia sa venuje vytvoreniu shellového skriptu, ktorý túto úlohu zabezpečí. Taktiež sa venuje popisu konfigurácie platformy VPP prostredníctvom kontajnera Ansible cez protokol NETCONF.

Skript automatizovaného vytvorenia topológie

Cieľom vytvoreného skriptu `topology-create.sh` je spustenie všetkých kontajnerov topológie a zároveň spustenie aj kontajneru ansible ku konfigurácií platformy VPP jednotlivých SR uzlov prostredníctvom protokolu NETCONF. Taktiež tento skript konfiguruje systémy Ubuntu uzlov N1 až N6 a uzlu trex. Po vykonaní tohto skriptu bude topológia pripravená ku generovaniu sieťovej prevádzky. Jednotlivé príkazy v tomto skripte z väčšej časti kopírujú príkazy uvedené v predošlých sekciách tejto kapitoly. K potrebe konfigurácie dynamického smerovania cez program FRRouting sa pripravili konfiguračné súbory, ktoré sa týmto skriptom kopírujú z hostiteľského systému Windows do jednotlivých kontajnerov. Skript následne spúšťa program FRRouting už s týmito pred pripravenými konfiguráciami OSPFv3. Podstatnejšou časťou automatizovanej konfigurácie je však konfigurácia platformy VPP na každom SR uzle, ktorá sa vykonáva automaticky cez kontajner ansible.

Konfigurácia VPP cez protokol NETCONF

Ku konfigurácií platformy VPP na každom z SR uzlov cez protokol NETCONF sa využíva kontajner ansible. Po jeho vytvorení sa spúšťa skript `entrypoint.sh` špecifikovaný príkazom ENTRYPOINT uvedeným v súbore Dockerfile pre obraz Ansible, viď 4.2. Obsah spúšťaného skriptu je nasledovný.

Skript entrypoint.sh kontajneru ansible.

```
wait-for-it node1:2831 -t 300
wait-for-it node2:2831 -t 300
wait-for-it node3:2831 -t 300
wait-for-it node4:2831 -t 300
wait-for-it node5:2831 -t 300
wait-for-it node6:2831 -t 300

if [ "$#" = 0 ]; then
    for playbook in /*.yaml; do
        ansible-playbook $playbook
    done
else
    for playbook in "$@"; do
        ansible-playbook $playbook
    done
fi
```

Pre konfiguráciu VPP jednotlivých kontajnerov prostredníctvom protokolu NETCONF, je nutné aby bol na každom kontajneri úspešne spustený agent Honeycomb. Inštalácia agentu Honeycomb sa nachádza v súbore Dockerfile pre kontajnery typu node. Jeho spustenie je okamžité po spustení kontajneru. Tento agent otvára a pracuje s portom 2831 a dokáže spracovávať NETCONF konfigurácie. Prvé riadky súboru `entrypoint.sh` zobrazeného vyššie využívajú skript `wait-for-it` [4], ktorým sa docieli pokračovanie v hlavnom skripte až po overení dostupnosti portu 2831 na každom jednom kontajneri typu node.

Druhou časťou vyššie uvedeného skriptu je spúšťanie nástroja `ansible-playbook`. Argumentom tohto nástroja sú jednotlivé playbooky s XML konfiguráciami pre protokol NETCONF. V prípade spustenia kontajneru ansible bez argumentov sa vykonajú všetky dostupné playbooky. To sa docieli nasledujúcim príkazom.

```
docker-compose up ansible
```

V prípade zadania argumentu vo forme názvu konkrétneho playbooku sa vykoná len špecifikovaný playbook.

```
docker-compose run ansible playbook-1-interfaces.yaml
```

Pre potreby tejto diplomovej práce sa vytvorili tri playbooky (súbory formátu YAML). Prvým Súborom je súbor `playbook-1-interfaces.yaml`. Tento súbor definuje XML konfiguráciu tap rozhraní medzi VPP a systémom Ubuntu v každom kontajneri typu node. Nižšie uvedený výpis kódu zobrazuje časť tohto súboru určenú pre konfiguráciu uzlu N1. Konfigurácia ostatných uzlov je obdobná.

[4] `http://manpages.ubuntu.com/manpages/bionic/man8/wait-for-it.8.html`

Časť súboru palybook-1-interfaces.yaml pre konfiguráciu uzlu N1.

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node 1 Interfaces
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
            <interface>
              <name>tap0</name>
              <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">v3po:tap</type>
              <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
                <tap-name>vpp</tap-name>
                <mac>00:00:00:00:10:10</mac>
              </tap>
              <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                <address>
                  <ip>192.168.10.10</ip>
                  <prefix-length>24</prefix-length>
                </address>
              </ipv4>
              <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                <address>
                  <ip>fd10::10</ip>
                  <prefix-length>64</prefix-length>
                </address>
              </ipv6>
            </interface>
          </interfaces>
        </config>
```

Táto časť konfigurácie má svoj ekvivalent v nasledujúcich VPP príkazoch cez utilitu vppctl.

```
create tap id 0 hw-addr 00:00:00:00:10:10 host-ip6-addr fd10::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 192.168.10.10/24
set interface ip address tapcli-0 fd10::10/64
```

Druhým playbook súborom vytvoreným pre konfiguráciu platforiem VPP je súbor s názvom `playbook-2-sid-definition.yaml`. Tento súbor slúži ku konfigurácií statickej trasy smerom k systému Ubuntu a k definícií lokálnych segmentov jednotlivých SR uzlov. Nasledujúci výpis obsahuje časť súboru s konfiguráciou uzlu N1.

Časť súboru playbook-2-sid-definitions.yaml pre konfiguráciu uzlu N1.

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node 1 SRv6 SIDs
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
            <control-plane-protocols>
              <control-plane-protocol>
                <type>static</type>
                <name>learned-protocol-0</name>
                <description>supplementary routing for SRv6 demos node vppA</description>
                <vpp-protocol-attributes xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                    <primary-vrf>0</primary-vrf>
                </vpp-protocol-attributes>

                <static-routes>
                  <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ipv6-unicast-routing">
                    <route>
                      <destination-prefix>::/0</destination-prefix>
                      <description>Default route</description>
                      <next-hop>
                        <outgoing-interface>tap0</outgoing-interface>
                        <next-hop-address>fd10::1</next-hop-address>
                      </next-hop>
                    </route>
                  </ipv6>
                </static-routes>
              </control-plane-protocol>
            </control-plane-protocols>
            <srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
              <encapsulation>
                <source-address>fd10::1</source-address>
                <ip-ttl-propagation>false</ip-ttl-propagation>
              </encapsulation>
              <locators>
                <locator>
                  <name>fd11::</name>
                  <enable>true</enable>
                  <is-default>false</is-default>
                  <prefix>
                    <address>fd11::</address>
                    <length>64</length>-
                  </prefix>
```

```
<!-- ... -->
                <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
                  <local-sids>
                    <sid>
                      <opcode>256</opcode> <!-- ::100 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:xml:ns:yang:ietf-srv6-types">
                          ietf-srv6-types:End
                      </end-behavior-type>
                      <end/>
                    </sid>
                    <sid>
                      <opcode>260</opcode> <!-- ::104 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:xml:ns:yang:ietf-srv6-types">
                          ietf-srv6-types:End.DX4
                      </end-behavior-type>
                      <end-dx4>
                        <paths>
                          <path>
                            <path-index>1</path-index>
                            <interface>tap0</interface>
                            <next-hop>192.168.10.1</next-hop>
                            <weight>1</weight>
                            <role>PRIMARY</role>
                          </path>
                        </paths>
                      </end-dx4>
                    </sid>
                    <sid>
                      <opcode>262</opcode> <!-- ::106 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:xml:ns:yang:ietf-srv6-types">
                          ietf-srv6-types:End.DX6
                      </end-behavior-type>
                      <end-dx6>
                        <paths>
                          <path>
                            <path-index>1</path-index>
                            <interface>tap0</interface>
                            <next-hop>fd10::1</next-hop>
                            <weight>1</weight>
                            <role>PRIMARY</role>
                          </path>
                        </paths>
                      </end-dx6>
                    </sid>
                  </local-sids>
                </static>
                <fib-table xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
                  <table-id>0</table-id>
                  <address-family xmlns:vpp-fib-table-management=
                      "urn:opendaylight:params:xml:ns:yang:vpp-fib-table-management">
                      vpp-fib-table-management:ipv6
                  </address-family>
                </fib-table>
              </locator>
            </locators>
          </srv6>
        </routing>
      </config>
```

Ekvivalentnou konfiguráciou tejto časti súboru sú nasledovné VPP príkazy.

```
ip route ::/0 via fd10::1
sr localsid address fd11::100 behavior end
sr localsid address fd11::104 behavior end.dx4 tapcli-0 192.168.10.1
sr localsid address fd11::106 behavior end.dx6 tapcli-0 fd10::1
```

Posledným, tretím konfiguračným YAML playbook súborom je súbor s názvom `playbook-3-srv6-policices.yaml`. Tento súbor obsahuje konfiguráciu SR politík na uzle N1 a N6 spolu s pravidlami smerovania paketov cez ne. Nasledujúci výpis zobrazuje časť tohto súboru s konfiguráciu prvej SR politiky P1 s príslušným smerovacím pravidlom k tejto politike na uzle N1.

Časť súboru playbook-3-srv6-policices.yaml pre konfiguráciu uzlu N1.

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node1 SRv6 Policies
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <segment-routing xmlns="http://cisco.com/ns/yang/oc-srte-policy">
            <traffic-engineering>
              <named-segment-lists>
                <named-segment-list>
                  <name>fd11:1066::1-1</name>
                  <config>
                    <name>fd11:1066::1-1</name>
                  </config>
                  <segments>
                    <segment>
                      <index>1</index>
                      <config>
                        <index>1</index>
                        <type>type-2</type>
                        <sid-value>fd22::100</sid-value>
                      </config>
                    </segment>
                    <segment>
                      <index>2</index>
                      <config>
                        <index>2</index>
                        <type>type-2</type>
                        <sid-value>fd55::100</sid-value>
                      </config>
                    </segment>
```

```
<!-- ... -->
                    <segment>
                      <index>3</index>
                      <config>
                        <index>3</index>
                        <type>type-2</type>
                        <sid-value>fd66::106</sid-value>
                      </config>
                    </segment>
                  </segments>
                </named-segment-list>
                <policies>
                <policy>
                  <config>
                    <name>fd11:1066::1</name>
                    <color>1</color>
                    <endpoint>fd60::60</endpoint>
                  </config>
                  <color>1</color>
                  <endpoint>fd60::60</endpoint>
                  <candidate-paths>
                    <candidate-path>
                      <name>candidatePath1</name>
                      <provisioning-method>provisioning-method-config</provisioning-method>
                      <preference>1</preference>
                      <distinguisher>0</distinguisher>
                      <config>
                        <name>candidatePath1</name>
                        <provisioning-method>provisioning-method-config</provisioning-method>
                        <computation-method>path-explicitly-defined</computation-method>
                        <preference>1</preference>
                        <distinguisher>0</distinguisher>
                      </config>
                      <binding-sid>
                        <config>
                          <alloc-mode>explicit</alloc-mode>
                          <type>srv6</type>
                          <value>fd11:1066::1</value>
                        </config>
                      </binding-sid>
                      <segment-lists>
                        <segment-list>
                          <name>fd11:1066::1-1</name>
                          <config>
                            <name>fd11:1066::1-1</name>
                            <weight>1</weight>
                          </config>
                        </segment-list>
                      </segment-lists>
                    </candidate-path>
                  </candidate-paths>
```

```
<!-- ... -->
                  <autoroute-include>
                    <config>
                      <metric-type>constant</metric-type>
                      <metric-constant>0</metric-constant>
                    </config>
                    <prefixes>
                      <config>
                        <prefixes-all>false</prefixes-all>
                      </config>
                      <prefix>
                        <ip-prefix>aaaa::/16</ip-prefix>
                        <config>
                          <ip-prefix>aaaa::/16</ip-prefix>
                        </config>
                      </prefix>
                    </prefixes>
                  </autoroute-include>
                  <binding-sid>
                    <config>
                      <alloc-mode>explicit</alloc-mode>
                      <type>srv6</type>
                      <value>fd11:1066::1</value>
                    </config>
                  </binding-sid>
                  <vpp-sr-policy xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
                    <config>
                      <policy-type>Default</policy-type>
                      <policy-behavior>Encapsulation</policy-behavior>
                      <table-id>0</table-id>
                      <address-family
                        xmlns:fib="urn:opendaylight:params:xml:ns:yang:vpp-fib-table-management">
                        fib:ipv6
                      </address-family>
                    </config>
                  </vpp-sr-policy>
                </policy>
```

Ekvivalentou konfiguráciou sú nasledujúce VPP príkazy.

```
set sr encaps source addr fd10::1
sr policy add bsid fd11:1066::1 next fd22::100 next fd55::100 next fd66::106 encap
sr steer l3 AAAA::/16 via bsid fd11:1066::1
```

V prípade konfigurácie 'steer' smerovacích pravidiel pre IPv4 cieľové siete prostredníctvom XML konfigurácie dochádza k nepredvídateľnému správaniu VPP. Namiesto správnej konfigurácie týchto pravidiel s definovanou IPv4 sieťou sa interpretuje príslušná XML konfigurácia ako IPv4 prefix 0.0.0.0, viď nasledujúci výpis z uzlu N1 po vykonaní konfigurácie cez NETCONF.

```
vpp# show sr steering-policies
SR steering policies:
Traffic          SR policy BSID
L3 cccc::/16     fd11:1166::3
L3 0.0.0.0/24    fd11:1046::4
L3 bbbb::/16     fd11:1066::2
L3 aaaa::/16     fd11:1066::1
```

Tento problém nanešťastie ostáva otvorený a nenašli sa žiadne relevantné zdroje popisujúce možnú jeho príčinu, prípadne jeho vyriešenie. Z tohto dôvodu sa do skriptu topology-create.sh doplnila manuálna konfigurácia VPP 'steer' pravidiel pre tieto IPv4 prefixy. Jedná sa o nasledovné dva príkazy.

```
docker-compose exec node1 vppctl sr steer l3 48.0.0.0/24 via bsid fd11:1046::4
docker-compose exec node6 vppctl sr steer l3 16.0.0.0/24 via bsid fd66:6041::4
```

Prvý príkaz sa vykonáva na uzle N1 a druhý príkaz na uzle N2. V prípade pravidiel pre IPv6 cieľové siete k tomuto problému nedochádza, ako ani k žiadnemu problému pri zvyšnej konfigurácií prostredníctvom protokolu NETCONF cez kontajner Ansible.

5 Simulované scenáre

Fungovanie SRv6 je popísané v teoretickej časti tejto diplomovej práce, v sekcii 1.2. Táto kapitola sa venuje praktickému popisu prechodu paketov naprieč implementovanou topológiou. Pozostáva z výsledkov generovania tokov generátorom TRex ako aj s priamymi ukážkami zachytených paketov v tejto topológii. Vzhľadom na charakter jednotlivých SR politík, stačí ak sa detailnejšie vysvetlia len dve. Politika P3, s modelom správania H.Insert a politika P4 s modelom správania H.Encaps.

5.1 Scenár s politikou modelu H.Insert

Politika P3 implementovaná na uzle N1 je graficky znázornená na obrázku Obr. 3.3 a jej parametre sú definované v tabuľke Tab. 3.1. Príkazy implementácie tejto politiky aj s jej príslušným pravidlom pre vkladanie paketov do nej je zobrazená na nasledujúcom výpise.

Politika P3 na uzle N1 s pravidlom vkladania paketov.

```
sr policy add bsid fd11:1166::3 next fd22::100 next fd44::100 next fd55::100 next fd66::100 insert
sr steer l3 CCCC::/16 via bsid fd11:1166::3
```

Sieťová prevádzka pre prvé tri politiky, vrátane politiky P3 je definovaná v súbore `/ipv6_stl_profile.py` a je generovaná súčasne. Nasledujúca časť zobrazuje zachytené pakety a ich zmeny naprieč ich prechodom v sieti. K zachytávaniu paketov sa využíva utilita `tcpdump` [1], ktorá sa spúšťa z príkazového riadku systému Ubuntu, v tomto prípade na uzle N1.

```
tcpdump -nni eth3 dst net cccc::/16 -Q in -w /n1-eth3.pcap
```

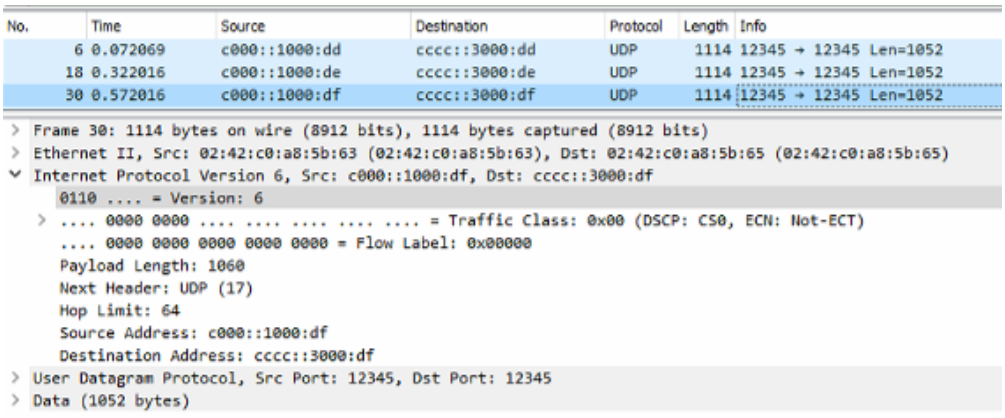
Parameter '-w' ukladá zachytené pakety do PCAP súboru `/n1-eth3.pcap` v rámci systému Ubuntu. K zobrazovaniu zachytených paketov sa využíva program Wireshark [2]. Tento program je inštalovaný na hostiteľskom systéme Docker platformy. Z tohto dôvodu je nutné vytvorený PCAP súbor skopírovať do hostiteľského systému Windows nasledujúcim príkazom vo Windows CMD termináli.

```
docker cp hc2vpp_node1_1:/n1-eth3.pcap ./n1-eth3.pcap
```

Nižšie uvedený obrázok Obr. 5.1 zobrazuje zachytené pakety na rozhraní eth3 kontajneru N1 v programe Wireshark.

Obrázok Obr. 5.1 zobrazuje paket, tak ako bol vygenerovaný, v neupravenom stave, bez SRv6 informácií. Nasledujúce zachytávanie tohto sieťového toku je vykonávané taktiež na uzle N1, avšak už na výstupnom rozhraní, rozhraní eth0 smerom k uzlu N2. Zachytávania sa vykonalo nasledujúcim príkazom.

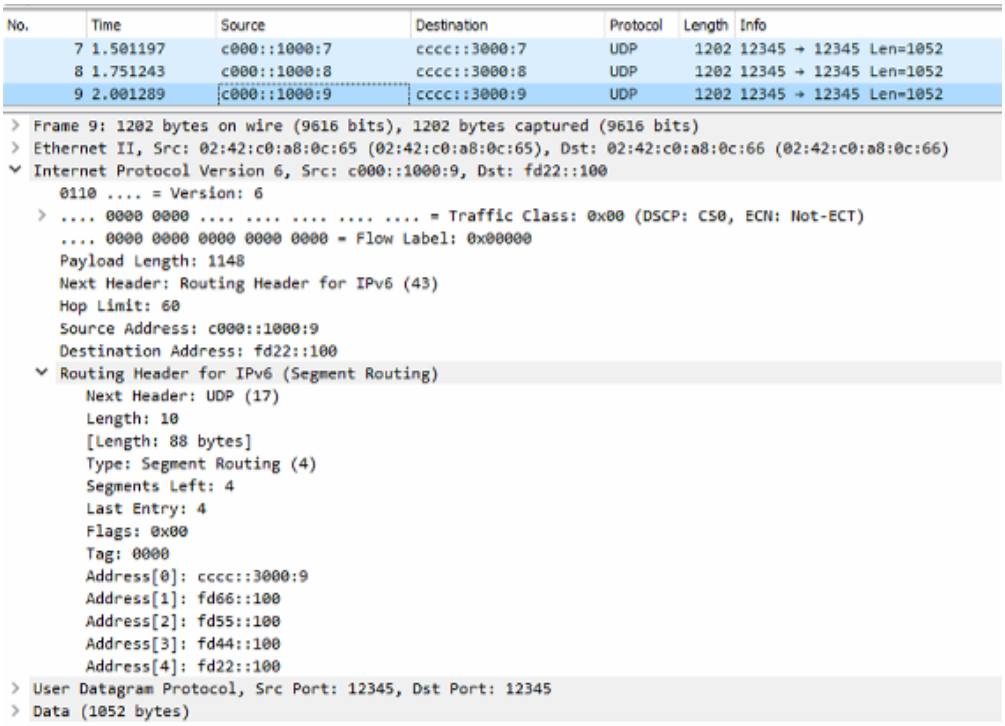
[1] `https://www.tcpdump.org/manpages/tcpdump.1.html`
[2] `https://www.wireshark.org`



Obr. 5.1: Zachytený paket na uzle N1 (sieť trex-1).

```
tcpdump -nni eth0 -Q out src net c000::/16 -w /n1-eth0.pcap
```

Po skopírovaní vytvoreného PCAP súboru `n1-eth0.pcap` je už možné programom Wireshark analyzovať SRv6 informácie, ktoré boli na uzle N1 pridané platformou VPP do jednotlivých paketov. Nasledujúci obrázok Obr. 5.2 zobrazuje tieto informácie.



Obr. 5.2: Zachytený paket na uzle N1 (sieť n1-n2).

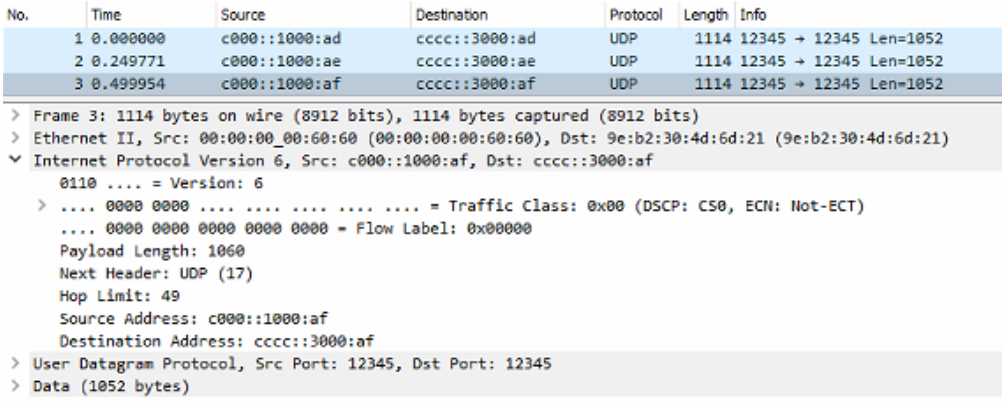
Zo zachyteného paketu je možné vidieť, že medzi medzi IPv6 hlavičku a hlavičku transportného protokolu UDP sa vložila rozširujúca hlavička SRH (Segment Routing

Header). V nej sa nachádza zoznam segmentov, ktoré sú definované v politike P3 na uzle N1. Keďže táto politika definuje svoj model správania ako H.Insert, vkladá sa len SRH hlavička s SR informáciami a nepridáva sa komplet zapuzdrenie novou IPv6 hlavičkou. Z tohto dôvodu je nutné uchovať informáciu o pôvodnej cieľovej adrese, ktorou je v zobrazenom prípade IPv6 adresa `cccc::3000:9`. Táto adresa sa vložila ako SID na nultý index zoznamu segmentov v SRH. Aktuálna hodnota poľa `Segments Left` je 4 a z tohto dôvodu je aktívny segment na indexe 4, čiže SID `fd22::100`. Aktívny segment sa taktiež vložil do poľa cieľovej adresy IPv6 hlavičky a zamenil tým pôvodnú cieľovú adresu. Takto sformovaný segment sa smeruje na daný segment implementovaný na uzle N2.

V tejto politike sa využívajú iba segmenty s modelom správania End. Na každom uzle, ktorý implementuje jeden z týchto segmentov sa znižuje hodnota poľa `Segments left`, čím sa postupne stávajú aktívnymi všetky segmenty v zozname segmentov. Posledným segmentom v politike je segment definovaný na uzle N6 (SID fd66::100) a jeho vykonaním sa do cieľovej adresy IPv6 hlavičky vkladá pôvodná cieľová adresa. Tým sa kompletne rekonštruuje pôvodný paket bez rozširujúcej hlavičky SRH, ktorá sa odstraňuje, keďže sa jedná o posledný, dodatočný segment pridaný k tomuto paketu. Zachytením paketov vo vstupnom smere rozhrania `vpp` v systéme Ubuntu kontajnera N6 je možné vidieť, že z platformy VPP tohto kontajnera odchádza pôvodný paket.

```
tcpdump -nni vpp -Q in dst net cccc::/16 -w /n6-vpp.pcap
```

Obrázok Obr. 5.4 zobrazuje pakety zachytené uvedeným príkazom na rozhraní medzi platformou VPP a systémom Ubuntu kontajnera N6.



Obr. 5.3: Zachytený paket na uzle N6 (sieť trex-2).

Tieto pakety následne systém Ubuntu odosiela na základe definovaných statických trias na rozhranie `eth3` smerom ku kontajneru Trex. Čo sa týka štatistík, tak

jednak je možné vidieť štatistiky počtu generovaných paketov v TRex konzole pomocou príkazu `stats`, prípadne cez interaktívne zobrazovanie príkazom `tui`, alebo je možné si zobraziť aj počet paketov spracovaných lokálnymi segmentami v platforme VPP v jednotlivých kontajneroch. Nasledujúci príkaz zobrazuje segment na uzle N6 aj s jeho základnými štatistikami o počte spracovaných paketov.

```
vpp# show sr localsids
SRv6 - My LocalSID Table:
=========================
        Address:        fd66::100
        Behavior:       End
        PSP:    True
        Good traffic:   [35436 packets : 38981424 bytes]
        Bad traffic:    [0 packets : 0 bytes]
--------------------
...
```

Nasledujúca sekcia 5.2 sa venuje scenáru s politikou využívajúcou H.Encaps model správania.

5.2 Scenár s politikou modelu H.Encaps

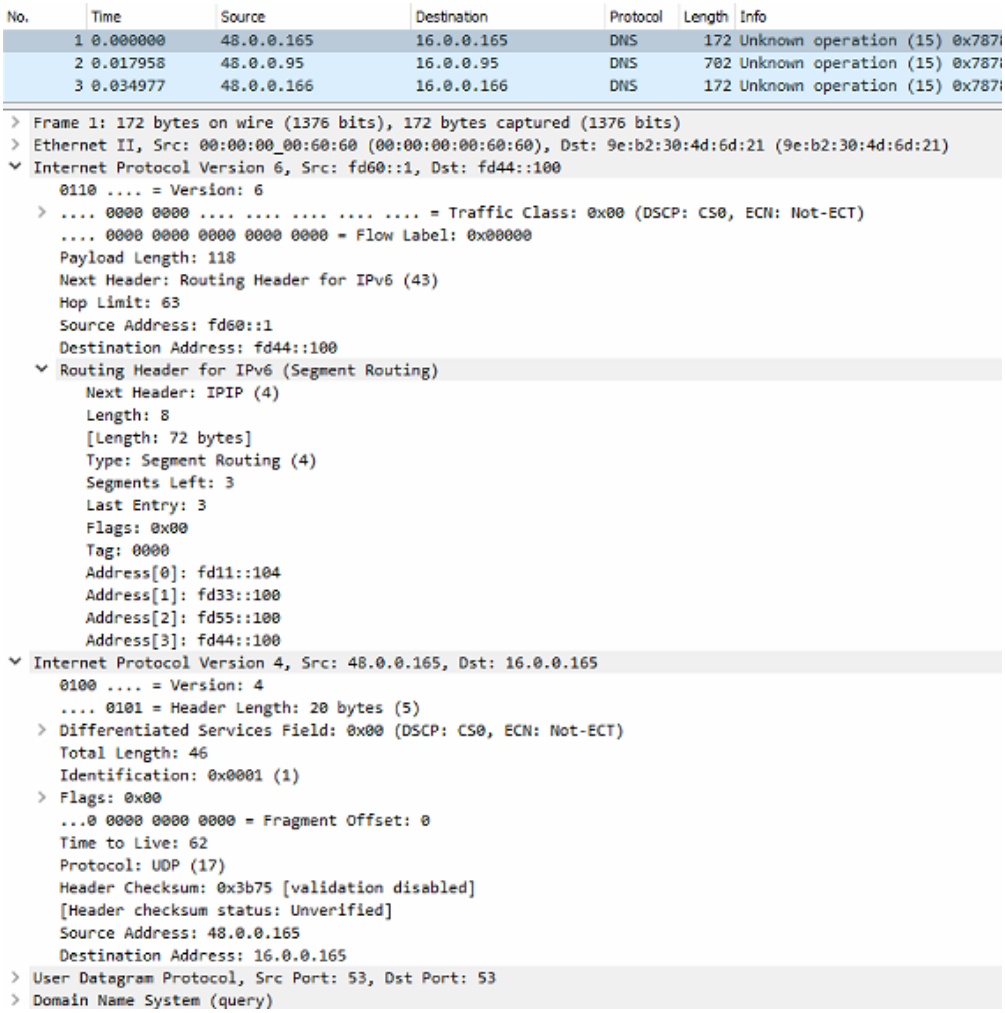
Vybranou politikou pre tento model správania je politika P4 na uzle N6. Táto politika je graficky znázornená na obrázku Obr. 3.6 a jej parametre sú definované v tabuľke Tab. 3.3. Príkazy implementácie tejto politiky aj s jej príslušným pravidlom pre vkladanie paketov do nej je zobrazená v nasledujúcom výpise.

```
sr policy add bsid fd66:6041::4 next fd44::100 next fd55::100 next fd33::100 next fd11::104 encap
sr steer l3 16.0.0.0/24 via bsid fd66:6041::4
```

Touto politikou je smerovaný IPv4 sieťový tok (profilom sieťovej prevádzky definovanej v súbore `ipv4_stl_profile.py`) a z tohto dôvodu je nutné aby politika využívala model správania H.Encaps, keďže nie je možné do generovaných paketov vložiť SRH hlavičku priamo. Pakety sa v tomto scenári generujú z TRex generátora smerom ku kontajneru N6. Následne ich systém Ubuntu odosiela do platformy VPP kde sa k nim pridáva IPv6 hlavička s potrebnými SRv6 informáciami. Upravené pakety sa už ako pakety protokolu IPv6 odosielajú späť do systému Ubuntu, ktorý ich podľa smerovacej tabuľky odosiela naprieč SR doménou. K zachyteniu upravených paketov platformou VPP sa využil nasledujúci príkaz v systéme Ubuntu.

```
tcpdump -nni vpp -Q in -w /n6-vpp-p4.pcap
```

Zachytená sieťová prevádzka v súbore `/n6-vpp-p4.pcap` sa skopíruje na hostiteľský systém platformy Docker, systém Windows. Následne je možné túto prevádzky analyzovať s použitím programu Wireshark. Nasledujúci obrázok Obr. 5.4 zobrazuje upravené platformou VPP pakety, ktoré obsahujú potrebné SRv6 informácie.



Obr. 5.4: Zachytený paket na uzle N6 (rozhranie vpp).

V obrázku Obr. 5.4 je možné vidieť, že sa k prijatým paketom platformou VPP, pridáva IPv6 hlavička spolu s rozširujúcou hlavičkou SRH. V hlavičke SRH je uvedený zoznam segmentov tejto politiky, ale keďže sa jedná o H.Encaps model správania, zoznam segmentov bez dodatočného segmentu s pôvodnou cieľovou adresou ako tomu bolo v predošlom scenári 5.1. Cieľovou IPv6 adresou je aktuálne aktívny segment, v tomto prípade segment s identifikátorom `fd44::100`. Zdrojová IPv6 adresa je tá, ktorá bola definovaná nasledujúcim príkazom.

```
set sr encaps source addr fd60::1
```

Takto sformovaný paket sa odosiela naprieč SR doménou k jednotlivým segmentom. Posledný segment tejto politiky je segment implementovaný na uzle N1 (SID fd11::104) s modelom správania End.DX4. Vykonaním tohto segmentu sa odstraňuje doplnená IPv6 hlavička a pôvodný IPv4 paket sa uzlom N1 odosiela smerom ku kontajneru TRex. Zobrazenie štatistík spracovávaných paketov jednotlivými lokálnymi

segmentami na uzle N1 je možné nasledovným VPP príkazom v termináli vppctl.

```
vpp# show sr localsids
SRv6 - My LocalSID Table:
=========================
    Address:        fd11::104
    Behavior:       DX4 (Endpoint with decapsulation and IPv4 cross-connect)
    Iface:          tapcli-0
    Next hop:       192.168.10.1
    PSP:    True
    Good traffic:   [3255 packets : 777250 bytes]
    Bad traffic:    [0 packets : 0 bytes]
--------------------
...
```

Takýmto spôsobom je možné vytvárať nad implementovanou topológiou nové politiky spolu s lokálnymi segmentami a následne testovať ich funkcionalitu generovaním sieťovej prevádzky cez generátor TRex.

Záver

Cieľom tejto diplomovej práce bol teoretický rozbor konceptu Segment Routing, jeho inštancie Segment Routing for IPv6 (SRv6) a platformy FD.io VPP. Praktická časť práca sa zameriavala na vytvorenie topológie s funkčnými simulačnými scenármi SRv6 pomocou platformy VPP.

Prvá kapitola sa venuje popisu konceptu Segment Routing a jeho inštancie využívajúcu protokol IPv6 – SRv6. V rámci tejto kapitoly je popísaná platforma FD.io VPP, ktorá slúžila na vytvorenie simulačných scenárov SRv6.

Teoretická časť práce pokračuje aj druhou kapitolou, ktorá sa venuje platformám a programom, ktoré sa využívajú pre vytvorenie funkčnej topológie. Platforma Docker, popisovaná v prvej sekcii tejto kapitoly, slúži k vytvoreniu základu topológie. Ďalšia sekcia sa venuje generátoru TRex, ktorým sa testuje topológia a konfigurované simulačné scenáre. V poslednej sekcií je popisovaný program Ansible, s ktorého pomocou sa vo vytvorenej topológii konfiguruje VPP cez protokol NETCONF.

Tretia kapitola sa venuje návrhu topológie a stručnému prehľadu jej vytvárania. Topológia pozostáva zo šiestich smerovačov, ktoré sú súčasťou SR domény a z emulovaných externých sietí, pripojených do dvoch hraničných uzlov. V druhej sekcií je podrobný popis ôsmich vytváraných simulačných scenárov, resp. SRv6 politík. Tieto politiky sú implementované na dvoch hraničných uzloch a reprezentujú smerovacie trasy naprieč SR doménou pomocou zoznamu segmentov.

Nasledujúca štvrtá kapitola sa zameriava na implementáciu navrhnutej topológie aj so simulačnými scenármi. Topológia bola vytváraná pomocou platformy Docker, ktorá umožňuje vytváranie jednotlivých uzlov siete prostredníctvom Docker kontajnerov. Koncept SRv6 bol zavedený do topológie pomocou platformy VPP. Pre vzdialenú konfiguráciu VPP prostredníctvom protokolu NETCONF sa využil program Ansible. Testovanie funkčnosti celej topológie ako aj jednotlivých simulačných scenárov bolo možné za pomoci generátoru TRex.

V piatej kapitole sú popisované dva typy vytváraných scenárov. Prvý scenár využíva SRv6 politiku s modelom správania H.Insert. Pri tomto modeli správania sa potrebné SR informácie vkladajú prostredníctvom rozširujúcej hlavičky SRH priamo do originálnych IPv6 paketov. V prípade druhého scenára sa jedná o model správania SRv6 politiky H.Encaps. Táto politika zapuzdruje originálne pakety do novej IPv6 hlavičky spolu s rozširujúcou hlavičkou SRH. Popis týchto scenárov sa zameriava na demonštrovanie prechodu jednotlivých paketov naprieč topológiou prostredníctvom SRv6. K tomu sa využíva zachytávanie paketov v rôznych častiach siete a ich následná analýza v prostredí Wireshark.

Koncept Segment Routing sa javí ako perspektívny spôsob zavádzania sieťového programovania nad tradičnou architektúrou sietí bez nutnosti kompletnej zmeny ich

dizajnu. Vďaka vytvorenej topológie je možné pokračovať v skúmaní možností tohto konceptu. Topológia zahŕňa spojazdnenie platformy VPP, jej konfiguráciu prostredníctvom protokolu NETCONF, ako aj funkčné zavedenie programu FRRouting pre spojazdnenie dynamických protokolov nad ňou.

Literatúra

[1] Abdelsalam, A., Ventre, P. L., Scarpitta, C., Mayer, A., Salsano, S. et al. SRPerf: A Performance Evaluation Framework for IPv6 Segment Routing. *IEEE Transactions on Network and Service Management*. 2021, zv. 18, č. 2, s. 2320–2333. DOI: 10.1109/TNSM.2020.3048328.

[2] Arun, S. *Container Namespaces – Deep Dive into Container Networking @ONLINE* [https://platform9.com/blog/container-namespaces-deep-dive-container-networking/]. Máj 2022.

[3] Bashandy, A., Filsfils, C., Previdi, S., Decraene, B., Litkowski, S. et al. *Segment Routing with the MPLS Data Plane* [RFC 8660]. RFC Editor, december 2019. DOI: 10.17487/RFC8660. Dostupné z: https://rfc-editor.org/rfc/rfc8660.txt.

[4] Björklund, M. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)* [RFC 6020]. RFC Editor, október 2010. DOI: 10.17487/RFC6020. Dostupné z: https://www.rfc-editor.org/info/rfc6020.

[5] Björklund, M. *The YANG 1.1 Data Modeling Language* [RFC 7950]. RFC Editor, august 2016. DOI: 10.17487/RFC7950. Dostupné z: https://www.rfc-editor.org/info/rfc7950.

[6] Björklund, M. *A YANG Data Model for Interface Management* [RFC 8343]. RFC Editor, marec 2018. DOI: 10.17487/RFC8343. Dostupné z: https://www.rfc-editor.org/info/rfc8343.

[7] Cisco. *Segment Routing Overview @ONLINE* [https://www.cisco.com/c/en/us/td/docs/routers/ncs4200/configuration/guide/segment-routing/17-1-1/b-segment-routing-17-1-ncs4200/b-segment-routing-17-1-ncs4200_chapter_00.pdf]. December 2021.

[8] Cisco. *TRex Advance stateful support @ONLINE* [https://trex-tgn.cisco.com/trex/doc/trex_astf.html]. Máj 2022.

[9] Cisco. *TRex @ONLINE* [https://trex-tgn.cisco.com/trex/doc/trex_manual.html]. Máj 2022.

[10] Cisco. *TRex Stateless support @ONLINE* [https://trex-tgn.cisco.com/trex/doc/trex_stateless.html]. Máj 2022.

[11] Clarence, F. *What is SRv6 network programming? @ONLINE* [https://blog.apnic.net/2020/05/01/what-is-srv6-network-programming]. Máj 2020.

[12] Cmarada, M. *Honeycomb for VPP @ONLINE* [https://gerrit.fd.io/r/gitweb?p=hc2vpp.git;a=tree;f=examples/docker;h=5cf4ec2517f81e9de415ad057fec8bbe6e2088e7;hb=refs/changes/71/13371/6]. 2022.

[13] Docker. *Docker overview @ONLINE* [https://docs.docker.com/get-started/overview/]. Máj 2022.

[14] Docker. *Dockerfile reference @ONLINE* [https://docs.docker.com/engine/reference/builder/]. Máj 2022.

[15] Docker. *Overview of Docker Compose @ONLINE* [https://docs.docker.com/compose/]. Máj 2022.

[16] Dugeon, O., Guedrez, R., Lahoud, S. a Texier, G. Demonstration of Segment Routing with SDN based label stack optimization. In: *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. 2017, s. 143–145. DOI: 10.1109/ICIN.2017.7899404.

[17] Enns, R., Björklund, M., Bierman, A. a Schönwälder, J. *Network Configuration Protocol (NETCONF)* [RFC 6241]. RFC Editor, jún 2011. DOI: 10.17487/RFC6241. Dostupné z: https://www.rfc-editor.org/info/rfc6241.

[18] Fabricplane. *SRv6 Overview @ONLINE* [http://www.fabricplane.com/srv6-overview]. Január 2020.

[19] FD.io. *VPP/What is VPP? @ONLINE* [https://wiki.fd.io/view/VPP/What_is_VPP%3F]. Máj 2017.

[20] FD.io. *Hc2vpp @ONLINE* [https://wiki.fd.io/view/Hc2vpp]. Február 2019.

[21] FD.io. *VPP Technology @ONLINE* [https://fd.io/gettingstarted/technology/]. December 2021.

[22] 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://rfc-editor.org/rfc/rfc8986.txt.

[23] 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://rfc-editor.org/rfc/rfc8754.txt.

[24] 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://rfc-editor.org/rfc/rfc8402.txt.

[25] Hanoch, H. *Test Your Limits With TRex Traffic Generator @ONLINE* [https://legacy.netdevconf.info/0x14/pub/papers/9/0x14-paper9-talk-paper.pdf]. Máj 2022.

[26] Hinden, B. a Deering, D. S. E. *Internet Protocol, Version 6 (IPv6) Specification* [RFC 2460]. RFC Editor, december 1998. DOI: 10.17487/RFC2460. Dostupné z: https://rfc-editor.org/rfc/rfc2460.txt.

[27] Huaweii. *Segment Routing @ONLINE* [https://support.huawei.com/enterprise/en/doc/EDOC1100092117]. September 2019.

[28] Iana. *IPv6 Extension Header Types @ONLINE* [https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml]. August 2021.

[29] Lee, C., Mori, N., Ohara, Y., Murakami, T., Asaba, S. et al. The Latency Characteristics of GTP-U and SRv6 Stateless Translation on VPP Software Router. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2021, s. 1429–1436. DOI: 10.1109/COMPSAC51774.2021.00212.

[30] Leonardo, R. *High-speed Traffic Generation @ONLINE* [https://nsg.ee.ethz.ch/fileadmin/user_upload/theses/sa_2020_06.pdf]. Máj 2022.

[31] Li, Z. Comparison between common virtualization solutions: VMware Workstation, Hyper-V and Docker. In: *2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC)*. 2021, s. 701–707. DOI: 10.1109/ICFTIC54370.2021.9647226.

[32] Linux, F. *What is the Vector Packet Processor (VPP) @ONLINE* [https://s3-docs.fd.io/vpp/22.02/index.html]. December 2021.

[33] RedHat. *Intro to playbooks @ONLINE* [https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html]. Máj 2022.

[34] RedHat. *Netconf_config - netconf device configuration @ONLINE* [https://docs.ansible.com/ansible/2.3/netconf_config_module.html]. Máj 2022.

[35] Spraggs, S. a Hagarty, D. *Cisco Converged 5G xHaul Transport @ONLINE* [https://www.cisco.com/c/en/us/solutions/service-provider/mobile-internet/5g-transport/converged-5g-xhaul-transport.html]. December 2021.

[36] Steven, B. *VPP lab with Honeycomb@ONLINE* [https://github.com/cisco-ie/vpp-lab]. 2022.

[37] Steven, V. *Segment Routing (SR) – What You Need To Know @ONLINE* [https://blogs.juniper.net/en-us/industry-solutions-and-trends/segment-routing-sr-what-you-need-to-know]. November 2019.

[38] Tamilselvan Gowtham, R. Y. *Getting Started: Network Automation with Ansible @ONLINE* [https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2019/pdf/TECDEV-1500.pdf]. Máj 2022.

[39] TRAN, D. *Prostředí pro testování zařízení umožňujících ochranu před DoS útoky.* Brno, 2020. Diplomová práca. Vysoké učení technické v Brně, Fakulta informačních technologií.

[40] TRex. *TRex Stateless @ONLINE* [https://gerrit.fd.io/r/gitweb?p=trex.git;hb=5c3e179aef55392a0cb69035bb05b4878bf81d20;f=doc%2Ftrex_stateless.asciidoc]. Máj 2022.

[41] Ventre, P. L., Salsano, S., Polverini, M., Cianfrani, A., Abdelsalam, A. et al. Segment Routing: A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results. *IEEE Communications Surveys Tutorials.* 2021, zv. 23, č. 1, s. 182–221. DOI: 10.1109/COMST.2020.3036826.

[42] VICAN, P. *Syntaktická analýza a validace datových modelů popsaných jazykem YANG.* Brno, 2016. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií.

[43] Wikipedia. *Ansible (software) @ONLINE* [https://cs.wikipedia.org/wiki/Ansible_(software)]. Máj 2022.

[44] Yan, J., Tang, L., Li, T., Lv, G., Quan, W. et al. PPB: a Path-based Packet Batcher to Accelerate Vector Packet Processor. In: *2020 15th International Conference on Computer Science Education (ICCSE).* 2020, s. 681–686. DOI: 10.1109/ICCSE49874.2020.9201881.

[45] Yan, Z. *What Is SRv6? @ONLINE* [https://info.support.huawei.com/info-finder/encyclopedia/en/SRv6.html]. September 2021.

Zoznam symbolov a skratiek

DPDK Data Plane Development Kit
DUT Device Under Test
FIB Forwarding Information Base
IETF Internet Engineering Task Force
IGP Interior Gateway Protocols
MPLS Multiprotocol Label Switching
MTU Maximum Transmission Unit
NETCONF Network Configuration Protocol
NFV Network Function Virtualization
NTP Network Time Protocol
OSPF Open Shortest Path First
OSPFv3 Open Shortest Path First routing protocol for IPv6
RH Routing Header
RPC Remote Procedure Call
SDN Software-defined networking
SID Segment Identifier
SNMP Simple Network Management Protocol
SR Segment Routing
SR-MPLS Segment Routing over MPLS
SRGB Segment Routing Global Block
SRH Segment Routing Header
SRv6 Segment Routing over IPv6
SSH Secure Shell Protocol
URI Uniform Resource Identifier
VPN Virtual private network
VPP Vector Packet Processing
YANG Yet Another Next Generation

Zoznam príloh

A Vytvorenie topológie

A.1 Súbor docker-compose.yaml

```
version: '2.1'
services:
  ansible:
    container_name: ansible
    depends_on: ["node1", "node2", "node3", "node4", "node5", "node6"]
    build: ansible
  trex:
    container_name: trex
    image: datmanslo/cisco-trex:2.88
    build: trex
    privileged: true
    sysctls:
      net.ipv6.conf.all.disable_ipv6: 0
    stdin_open: true
    tty: true
    ports:
      - "4500:4500"
      - "4501:4501"
      - "4507:4507"
      - "4500:4500/udp"
      - "4501:4501/udp"
      - "4507:4507/udp"
    networks:
      default:
        priority: 1000
      trex-1:
        ipv4_address: 192.168.91.99
        ipv6_address: fd91::99
        priority: 900
      trex-2:
        ipv4_address: 192.168.92.99
        ipv6_address: fd92::99
        priority: 800
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure
    command: ["./t-rex-64", "-i", "--cfg", "/etc/trex_cfg.yaml"]
  node1:
    container_name: node1
    build: .
    ports: ["8301:2831", "4431:8445"]
    privileged: true
    sysctls:
      net.ipv6.conf.all.disable_ipv6: 0
      net.ipv4.ip_forward: 1
      net.ipv6.conf.all.forwarding: 1
    networks:
      default:
        priority: 1000
      node1_node2:
        ipv4_address: 192.168.12.101
        ipv6_address: fd12::101
```

```
      priority: 200
    node1_node3:
      ipv4_address: 192.168.13.101
      ipv6_address: fd13::101
      priority: 100
    trex-1:
      ipv4_address: 192.168.91.101
      ipv6_address: fd91::101
      priority: 900
node2:
  container_name: node2
  build: .
  ports: ["8302:2831", "4432:8445"]
  privileged: true
  sysctls:
    net.ipv6.conf.all.disable_ipv6: 0
    net.ipv4.ip_forward: 1
    net.ipv6.conf.all.forwarding: 1
  networks:
    default:
      priority: 1000
    node1_node2:
      ipv4_address: 192.168.12.102
      ipv6_address: fd12::102
      priority: 200
    node2_node3:
      ipv4_address: 192.168.23.102
      ipv6_address: fd23::102
      priority: 100
    node2_node4:
      ipv4_address: 192.168.24.102
      ipv6_address: fd24::102
      priority: 100
    node2_node5:
      ipv4_address: 192.168.25.102
      ipv6_address: fd25::102
      priority: 100
node3:
  container_name: node3
  build: .
  ports: ["8303:2831", "4433:8445"]
  privileged: true
  sysctls:
    net.ipv6.conf.all.disable_ipv6: 0
    net.ipv4.ip_forward: 1
    net.ipv6.conf.all.forwarding: 1
  networks:
    default:
      priority: 1000
    node1_node3:
      ipv4_address: 192.168.13.103
      ipv6_address: fd13::103
      priority: 200
    node2_node3:
      ipv4_address: 192.168.23.103
      ipv6_address: fd23::103
      priority: 100
    node3_node4:
      ipv4_address: 192.168.34.103
```

```
      ipv6_address: fd34::103
      priority: 100
    node3_node5:
      ipv4_address: 192.168.35.103
      ipv6_address: fd35::103
      priority: 100
node4:
  container_name: node4
  build: .
  ports: ["8304:2831", "4434:8445"]
  privileged: true
  sysctls:
    net.ipv6.conf.all.disable_ipv6: 0
    net.ipv4.ip_forward: 1
    net.ipv6.conf.all.forwarding: 1
  networks:
    default:
      priority: 1000
    node2_node4:
      ipv4_address: 192.168.24.104
      ipv6_address: fd24::104
      priority: 200
    node3_node4:
      ipv4_address: 192.168.34.104
      ipv6_address: fd34::104
      priority: 100
    node4_node5:
      ipv4_address: 192.168.45.104
      ipv6_address: fd45::104
      priority: 100
    node4_node6:
      ipv4_address: 192.168.46.104
      ipv6_address: fd46::104
      priority: 100
node5:
  container_name: node5
  build: .
  ports: ["8305:2831", "4435:8445"]
  privileged: true
  sysctls:
    net.ipv6.conf.all.disable_ipv6: 0
    net.ipv4.ip_forward: 1
    net.ipv6.conf.all.forwarding: 1
  networks:
    default:
      priority: 1000
    node2_node5:
      ipv4_address: 192.168.25.105
      ipv6_address: fd25::105
      priority: 200
    node3_node5:
      ipv4_address: 192.168.35.105
      ipv6_address: fd35::105
      priority: 100
    node4_node5:
      ipv4_address: 192.168.45.105
      ipv6_address: fd45::105
      priority: 100
    node5_node6:
```

```
        ipv4_address: 192.168.56.105
        ipv6_address: fd56::105
        priority: 100
  node6:
    container_name: node6
    build: .
    ports: ["8306:2831", "4436:8445"]
    privileged: true
    sysctls:
      net.ipv6.conf.all.disable_ipv6: 0
      net.ipv4.ip_forward: 1
      net.ipv6.conf.all.forwarding: 1
    networks:
      default:
        priority: 1000
      node4_node6:
        ipv4_address: 192.168.46.106
        ipv6_address: fd46::106
        priority: 200
      node5_node6:
        ipv4_address: 192.168.56.106
        ipv6_address: fd56::106
        priority: 100
      trex-2:
        ipv4_address: 192.168.92.106
        ipv6_address: fd92::106
        priority: 800
networks:
  default:
    driver: bridge
    driver_opts:
      com.docker.network.driver.mtu: 9000
  node1_node2:
    driver_opts:
      com.docker.network.driver.mtu: 9000
    enable_ipv6: true
    ipam:
      config:
        - subnet: 192.168.12.0/24
        - subnet: fd12::/64
  node1_node3:
    driver_opts:
      com.docker.network.driver.mtu: 9000
    enable_ipv6: true
    ipam:
      config:
        - subnet: 192.168.13.0/24
        - subnet: fd13::/64
  node2_node3:
    driver_opts:
      com.docker.network.driver.mtu: 9000
    enable_ipv6: true
    ipam:
      config:
        - subnet: 192.168.23.0/24
        - subnet: fd23::/64
  node2_node4:
    driver_opts:
      com.docker.network.driver.mtu: 9000
```

```
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.24.0/24
      - subnet: fd24::/64
node2_node5:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.25.0/24
      - subnet: fd25::/64
node3_node4:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.34.0/24
      - subnet: fd34::/64
node3_node5:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.35.0/24
      - subnet: fd35::/64
node4_node5:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.45.0/24
      - subnet: fd45::/64
node4_node6:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.46.0/24
      - subnet: fd46::/64
node5_node6:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  enable_ipv6: true
  ipam:
    config:
      - subnet: 192.168.56.0/24
      - subnet: fd56::/64
trex-1:
  driver_opts:
    com.docker.network.driver.mtu: 9000
  ipam:
    config:
      - subnet: 192.168.91.0/24
```

```
        - subnet: fd91::/64
  trex-2:
    driver_opts:
      com.docker.network.driver.mtu: 9000
    ipam:
      config:
        - subnet: 192.168.92.0/24
        - subnet: fd92::/64
```

A.2 Skript topology-create.sh

```
#!/bin/bash
echo Bringing up containers...
docker-compose up -d

echo Waiting for Ansible to complete...
docker-compose logs -f ansible

echo Configuring TREX container networking...
docker-compose exec trex ip addr add fd91::99/64 dev eth1
docker-compose exec trex ip addr add fd92::99/64 dev eth2
docker cp ./trex/ipv6_stl_profile.py trex:/ipv6_stl_profile.py
docker cp ./trex/ipv4_stl_profile.py trex:/ipv4_stl_profile.py

echo Configuring NODE1 Ubuntu networking...
docker-compose exec node1 ip addr add 192.168.10.1/255.255.255.0 dev vpp
docker-compose exec node1 ip addr add fd10::1/64 dev vpp
docker-compose exec node1 ip link set dev vpp mtu 9000
docker-compose exec node1 ip addr add fd91::101/64 dev eth3
docker-compose exec node1 ip route add 48.0.0.0/24 via 192.168.10.10 dev vpp
docker-compose exec node1 ip route add 16.0.0.0/24 via 192.168.91.99 dev eth3
docker-compose exec node1 ip route add aaaa::/16 via fd10::10 dev vpp
docker-compose exec node1 ip route add a000::/16 via fd91::99 dev eth3
docker-compose exec node1 ip route add bbbb::/16 via fd10::10 dev vpp
docker-compose exec node1 ip route add b000::/16 via fd91::99 dev eth3
docker-compose exec node1 ip route add cccc::/16 via fd10::10 dev vpp
docker-compose exec node1 ip route add c000::/16 via fd91::99 dev eth3
docker-compose exec node1 ip route add fd11::/64 via fd10::10 dev vpp
docker cp ./frr/daemons node1:/etc/frr/daemons
docker cp ./frr/frr1.conf node1:/etc/frr/frr.conf
docker-compose exec node1 service frr start

echo Configuring NODE2 Ubuntu networking...
docker-compose exec node2 ip addr add 192.168.20.1/255.255.255.0 dev vpp
docker-compose exec node2 ip addr add fd20::1/64 dev vpp
docker-compose exec node2 ip link set dev vpp mtu 9000
docker-compose exec node2 ip route add fd22::/64 via fd20::20 dev vpp
docker cp ./frr/daemons node2:/etc/frr/daemons
docker cp ./frr/frr2.conf node2:/etc/frr/frr.conf
docker-compose exec node2 service frr start

echo Configuring NODE3 Ubuntu networking...
docker-compose exec node3 ip addr add 192.168.30.1/255.255.255.0 dev vpp
docker-compose exec node3 ip addr add fd30::1/64 dev vpp
docker-compose exec node3 ip link set dev vpp mtu 9000
docker-compose exec node3 ip route add fd33::/64 via fd30::30 dev vpp
docker cp ./frr/daemons node3:/etc/frr/daemons
```

```
docker cp ./frr/frr3.conf node3:/etc/frr/frr.conf
docker-compose exec node3 service frr start

echo Configuring NODE4 Ubuntu networking...
docker-compose exec node4 ip addr add 192.168.40.1/255.255.255.0 dev vpp
docker-compose exec node4 ip addr add fd40::1/64 dev vpp
docker-compose exec node4 ip link set dev vpp mtu 9000
docker-compose exec node4 ip route add fd44::/64 via fd40::40 dev vpp
docker cp ./frr/daemons node4:/etc/frr/daemons
docker cp ./frr/frr4.conf node4:/etc/frr/frr.conf
docker-compose exec node4 service frr start

echo Configuring NODE5 Ubuntu networking...
docker-compose exec node5 ip addr add 192.168.50.1/255.255.255.0 dev vpp
docker-compose exec node5 ip addr add fd50::1/64 dev vpp
docker-compose exec node5 ip link set dev vpp mtu 9000
docker-compose exec node5 ip route add fd55::/64 via fd50::50 dev vpp
docker cp ./frr/daemons node5:/etc/frr/daemons
docker cp ./frr/frr5.conf node5:/etc/frr/frr.conf
docker-compose exec node5 service frr start

echo Configuring NODE6 Ubuntu networking...
docker-compose exec node6 ip addr add 192.168.60.1/255.255.255.0 dev vpp
docker-compose exec node6 ip addr add fd60::1/64 dev vpp
docker-compose exec node6 ip link set dev vpp mtu 9000
docker-compose exec node6 ip addr add fd92::106/64 dev eth3
docker-compose exec node6 ip route add 16.0.0.0/24 via 192.168.60.60 dev vpp
docker-compose exec node6 ip route add 48.0.0.0/24 via 192.168.92.99 dev eth3
docker-compose exec node6 ip route add a000::/16 via fd60::60 dev vpp
docker-compose exec node6 ip route add aaaa::/16 via fd92::99 dev eth3
docker-compose exec node6 ip route add b000::/16 via fd60::60 dev vpp
docker-compose exec node6 ip route add bbbb::/16 via fd92::99 dev eth3
docker-compose exec node6 ip route add c000::/16 via fd60::60 dev vpp
docker-compose exec node6 ip route add cccc::/16 via fd92::99 dev eth3
docker-compose exec node6 ip route add fd66::/64 via fd60::60 dev vpp
docker cp ./frr/daemons node6:/etc/frr/daemons
docker cp ./frr/frr6.conf node6:/etc/frr/frr.conf
docker-compose exec node6 service frr start

docker-compose exec node1 vppctl sr steer l3 48.0.0.0/24 via bsid fd11:1046::4
docker-compose exec node6 vppctl sr steer l3 16.0.0.0/24 via bsid fd66:6041::4

cat <<EOF
All configurations done!
==================================== README =======================================
Open a shell to a Ubuntu container:     docker-compose exec node1 bash
Open the VPP CLI (vppctl) of a node:    docker-compose exec node1 vppctl
Teardown whole topology:                docker-compose down
Open TRex console:                      docker-compose exec trex bash ./trex-console
 - start IPv6 traffic (in console):         start -f /ipv6_stl_profile.py
 - start IPv4 traffic (in console):         start -f /ipv4_stl_profile.py
TRex CONSOLE COMMANDS:
    start -f [traffic profile file]     Start generating of traffic.
    update -m 100mbps                   Update generating speed.
    pause                               Pause generating of traffic.
    resume                              Resume generating of trarric.
    stats                               Show generated traffic statistics.
    tui                                 Open real-time statistics.
    stop                                Stop generating of traffic.
```

```
    clear                           Clear statistics.

EOF
```

A.3 Konfigurácia FRRouting

A.3.1 Súbor daemons

```
# This file tells the frr package which daemons to start.
#
# Sample configurations for these daemons can be found in
# /usr/share/doc/frr/examples/.
#
# ATTENTION:
#
# When activating a daemon for the first time, a config file, even if it is
# empty, has to be present *and* be owned by the user and group "frr", else
# the daemon will not be started by /etc/init.d/frr. The permissions should
# be u=rw,g=r,o=.
# When using "vtysh" such a config file is also needed. It should be owned by
# group "frrvty" and set to ug=rw,o= though. Check /etc/pam.d/frr, too.
#
# The watchfrr, zebra and staticd daemons are always started.
#
bgpd=no
ospfd=no
ospf6d=yes
ripd=no
ripngd=no
isisd=no
pimd=no
ldpd=no
nhrpd=no
eigrpd=no
babeld=no
sharpd=no
pbrd=no
bfdd=no
fabricd=no
vrrpd=no
pathd=no

#
# If this option is set the /etc/init.d/frr script automatically loads
# the config via "vtysh -b" when the servers are started.
# Check /etc/pam.d/frr if you intend to use "vtysh"!
#
vtysh_enable=yes
zebra_options="  -A 127.0.0.1 -s 90000000"
bgpd_options="   -A 127.0.0.1"
ospfd_options="  -A 127.0.0.1"
ospf6d_options=" -A ::1"
ripd_options="   -A 127.0.0.1"
ripngd_options=" -A ::1"
isisd_options="  -A 127.0.0.1"
pimd_options="   -A 127.0.0.1"
```

```
ldpd_options="   -A 127.0.0.1"
nhrpd_options="  -A 127.0.0.1"
eigrpd_options=" -A 127.0.0.1"
babeld_options=" -A 127.0.0.1"
sharpd_options=" -A 127.0.0.1"
pbrd_options="   -A 127.0.0.1"
staticd_options="-A 127.0.0.1"
bfdd_options="   -A 127.0.0.1"
fabricd_options="-A 127.0.0.1"
vrrpd_options="  -A 127.0.0.1"
pathd_options="  -A 127.0.0.1"

# configuration profile
#
#frr_profile="traditional"
#frr_profile="datacenter"

#
# This is the maximum number of FD's that will be available.
# Upon startup this is read by the control files and ulimit
# is called.  Uncomment and use a reasonable value for your
# setup if you are expecting a large number of peers in
# say BGP.
#MAX_FDS=1024

# The list of daemons to watch is automatically generated by the init script.
#watchfrr_options=""

# To make watchfrr create/join the specified netns, use the following option:
#watchfrr_options="--netns"
# This only has an effect in /etc/frr/<somename>/daemons, and you need to
# start FRR with "/usr/lib/frr/frrinit.sh start <somename>".

# for debugging purposes, you can specify a "wrap" command to start instead
# of starting the daemon directly, e.g. to use valgrind on ospfd:
#   ospfd_wrap="/usr/bin/valgrind"
# or you can use "all_wrap" for all daemons, e.g. to use perf record:
#   all_wrap="/usr/bin/perf record --call-graph -"
# the normal daemon command is added to this at the end.
```

A.3.2 Konfigurácia uzlu N1: frr1.conf

```
frr version 8.2.2
frr defaults traditional
hostname node1
domainname localdomain
log syslog informational
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
```

```
router ospf6
 ospf6 router-id 1.1.1.1
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node1_SIDs seq 5 permit fd11::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node1_SIDs
exit
!
```

A.3.3 Konfigurácia uzlu N2: frr2.conf

```
frr version 8.2.2
frr defaults traditional
hostname node2
domainname localdomain
log syslog informational
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
interface eth2
 ipv6 ospf6 area 0
exit
!
interface eth3
 ipv6 ospf6 area 0
exit
!
router ospf6
 ospf6 router-id 2.2.2.2
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node2_SIDs seq 5 permit fd22::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node2_SIDs
exit
!
```

A.3.4 Konfigurácia uzlu N3: frr3.conf

```
frr version 8.2.2
frr defaults traditional
hostname node3
domainname localdomain
```

```
log syslog informational
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
interface eth2
 ipv6 ospf6 area 0
exit
!
interface eth3
 ipv6 ospf6 area 0
exit
!
router ospf6
 ospf6 router-id 3.3.3.3
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node3_SIDs seq 5 permit fd33::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node3_SIDs
exit
!
```

A.3.5 Konfigurácia uzlu N4: frr4.conf

```
frr version 8.2.2
frr defaults traditional
hostname node4
domainname localdomain
log syslog informational
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
interface eth2
 ipv6 ospf6 area 0
exit
!
interface eth3
 ipv6 ospf6 area 0
exit
!
router ospf6
```

```
 ospf6 router-id 4.4.4.4
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node4_SIDs seq 5 permit fd44::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node4_SIDs
exit
!
```

A.3.6 Konfigurácia uzlu N5: frr5.conf

```
frr version 8.2.2
frr defaults traditional
hostname node5
domainname localdomain
log syslog informational
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
interface eth2
 ipv6 ospf6 area 0
exit
!
interface eth3
 ipv6 ospf6 area 0
exit
!
router ospf6
 ospf6 router-id 5.5.5.5
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node5_SIDs seq 5 permit fd55::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node5_SIDs
exit
!
```

A.3.7 Konfigurácia uzlu N6: frr6.conf

```
frr version 8.2.2
frr defaults traditional
hostname node6
domainname localdomain
log syslog informational
```

```
service integrated-vtysh-config
!
interface eth0
 ipv6 ospf6 area 0
exit
!
interface eth1
 ipv6 ospf6 area 0
exit
!
router ospf6
 ospf6 router-id 6.6.6.6
 redistribute kernel route-map VPP_SIDs
exit
!
ipv6 prefix-list node6_SIDs seq 5 permit fd66::/64
!
route-map VPP_SIDs permit 1
 match ipv6 address prefix-list node6_SIDs
exit
!
```

A.4 Manuálna konfigurácia VPP

```
# NODE1
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:10:10 host-ip6-addr fd10::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd10::10/64
set interface ip address tapcli-0 192.168.10.10/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd10::1

# SRv6 Local segments definition
sr localsid address fd11::100 behavior end
sr localsid address fd11::104 behavior end.dx4 tapcli-0 192.168.10.1
sr localsid address fd11::106 behavior end.dx6 tapcli-0 fd10::1

# SRv6 Policices definition
sr policy add bsid fd11:1066::1 next fd22::100 next fd55::100 next fd66::106 encap
sr policy add bsid fd11:1066::2 next fd33::100 next fd44::100 next fd66::106 encap
sr policy add bsid fd11:1166::3 next fd22::100 next fd44::100 next fd55::100 next fd66::100 insert
sr policy add bsid fd11:1046::4 next fd33::100 next fd55::100 next fd44::100 next fd66::104 encap

# SRv6 Source address definition for T.Encaps behavior
set sr encaps source addr fd10::1

# SRv6 Steering traffic rules definition
sr steer l3 AAAA::/16 via bsid fd11:1066::1
sr steer l3 BBBB::/16 via bsid fd11:1066::2
sr steer l3 CCCC::/16 via bsid fd11:1166::3
sr steer l3 48.0.0.0/24 via bsid fd11:1046::4

#############################################################################################################
# NODE2
```

```
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:20:20 host-ip6-addr fd20::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd20::20/64
set interface ip address tapcli-0 192.168.20.20/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd20::1

# SRv6 Local segments definition
sr localsid address fd22::100 behavior end

#############################################################################################################
# NODE3
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:30:30 host-ip6-addr fd30::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd30::30/64
set interface ip address tapcli-0 192.168.30.30/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd30::1

# SRv6 Local segments definition
sr localsid address fd33::100 behavior end

#############################################################################################################
# NODE4
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:40:40 host-ip6-addr fd40::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd40::40/64
set interface ip address tapcli-0 192.168.40.40/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd40::1

# SRv6 Local segments definition
sr localsid address fd44::100 behavior end

#############################################################################################################
# NODE5
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:50:50 host-ip6-addr fd50::1/64 host-if-name vpp
set interface state tapcli-0 up
set interface ip address tapcli-0 fd50::40/64
set interface ip address tapcli-0 192.168.50.50/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd50::1

# SRv6 Local segments definition
sr localsid address fd55::100 behavior end

#############################################################################################################
# NODE6
# Creating tap interface to the host Ubuntu system
create tap id 0 hw-addr 00:00:00:00:60:60 host-ip6-addr fd60::1/64 host-if-name vpp
set interface state tapcli-0 up
```

```
set interface ip address tapcli-0 fd60::60/64
set interface ip address tapcli-0 192.168.60.60/24

# Default route towards the host Ubuntu system
ip route ::/0 via fd60::1

# SRv6 Local segments definition
sr localsid address fd66::100 behavior end
sr localsid address fd66::104 behavior end.dx4 tapcli-0 192.168.60.1
sr localsid address fd66::106 behavior end.dx6 tapcli-0 fd60::1

# SRv6 Policices definition
sr policy add bsid fd66:6061::1 next fd55::100 next fd22::100 next fd11::106 encap
sr policy add bsid fd66:6061::2 next fd44::100 next fd33::100 next fd11::106 encap
sr policy add bsid fd66:6161::3 next fd55::100 next fd44::100 next fd22::100 next fd11::100 insert
sr policy add bsid fd66:6041::4 next fd44::100 next fd55::100 next fd33::100 next fd11::104 encap

# SRv6 Source address definition for T.Encaps behavior
set sr encaps source addr fd60::1

# SRv6 Steering traffic rules definition
sr steer l3 A000::/16 via bsid fd66:6061::1
sr steer l3 B000::/16 via bsid fd66:6061::2
sr steer l3 C000::/16 via bsid fd66:6161::3
sr steer l3 16.0.0.0/24 via bsid fd66:6041::4
```

B Kontajner node

B.1 Súbor Dockerfile

```
FROM ubuntu:18.04

ARG VPP=18.10-release
ARG HONEYCOMB=1.18.10-RC2~16
ARG FRRVER=frr-stable

ARG VPP_BASE=https://nexus.fd.io/content/repositories/\
fd.io.stable.1810.ubuntu.bionic.main/io/fd/vpp/
ARG HONEYCOMB_BASE=https://nexus.fd.io/content/repositories/\
fd.io.stable.1810.ubuntu.xenial.main/io/fd/hc2vpp/honeycomb/

ARG PKG_VPP=${VPP_BASE}/vpp/${VPP}_amd64/vpp-${VPP}_amd64-deb.deb
ARG PKG_VPP_LIB=${VPP_BASE}/vpp-lib/${VPP}_amd64/vpp-lib-${VPP}_amd64-deb.deb
ARG PKG_VPP_PLUGINS=${VPP_BASE}/vpp-plugins/${VPP}_amd64/vpp-plugins-${VPP}_amd64-deb.deb
ARG PKG_HONEYCOMB=${HONEYCOMB_BASE}/${HONEYCOMB}_all/honeycomb-${HONEYCOMB}_all-deb.deb

RUN apt-get update && apt-get install -y \
    iproute2 iputils-ping net-tools vim-tiny jshon telnet curl wget ethtool vim tcpdump\
    libnuma1 libssl1.0.0 libmbedcrypto1 libmbedtls10 libmbedx509-0 \
    openjdk-8-jre-headless \
&& mkdir /tmp/deb && cd /tmp/deb \
&& echo $PKG_VPP \\n $PKG_VPP_LIB \\n $PKG_VPP_PLUGINS \\n $PKG_HONEYCOMB > urls \
&& wget -i urls && dpkg --ignore-depends=vpp --ignore-depends=vpp-plugins -i *.deb \
&& cd / && rm -rf /var/lib/apt/lists/* /tmp/deb \
&& apt-get update && apt-get install -y gnupg2 \
&& curl -s https://deb.frrouting.org/frr/keys.asc | apt-key add - \
&& apt-get update && apt-get install -y lsb-release && apt-get clean all \
&& echo deb https://deb.frrouting.org/frr $(lsb_release -s -c) ${FRRVER} | \
tee -a /etc/apt/sources.list.d/frr.list \
&& apt-get update && apt-get install -y frr frr-pythontools

COPY entrypoint.sh /
CMD ["/entrypoint.sh"]
```

B.2 Skript entrypoint.sh

```
#!/bin/bash

mknod /dev/vhost-net c 10 238

grep -q statseg /etc/vpp/startup.conf \
|| echo -e "statseg {\ndefault\n}
plugins {\nplugin dpdk_plugin.so { disable }\n}\n" \
>> /etc/vpp/startup.conf
ulimit -c unlimited

vpp -c /etc/vpp/startup.conf &
while [ ! -S "/run/vpp/stats.sock" -o ! -S "/run/vpp-api.sock" ]; do
    sleep 1;
done
```

```
/opt/honeycomb/honeycomb-start

while true; do
    vpp_prometheus_export /net /if /err >>/var/log/vpp_prometheus_export 2>&1
    sleep 5
done
```

C Kontajner trex

C.1 Súbor Dockerfile

```
ARG UBUNTU_VERSION=focal-20210119
FROM ubuntu:${UBUNTU_VERSION}

ARG TREX_VERSION=v2.88

RUN apt-get update \
    && apt-get -y install --no-install-recommends \
    iproute2 iputils-ping nano net-tools netbase pciutils \
    python3 python3-distutils strace wget vim tcpdump \
    && apt-get autoclean \
    && apt-get autoremove -y \
    && rm -rf /var/lib/apt/lists/*

RUN wget --no-check-certificate https://trex-tgn.cisco.com/trex/release/\
${TREX_VERSION}.tar.gz && \
    tar -zxvf ${TREX_VERSION}.tar.gz -C / && \
    chown root:root /${TREX_VERSION}  && \
    rm ${TREX_VERSION}.tar.gz

COPY trex_cfg.yaml /etc/trex_cfg.yaml
WORKDIR /${TREX_VERSION}

CMD tail -f /dev/null
```

C.2 Súbor trex_cfg.yaml

```
- port_limit    : 2
  version       : 2
  interfaces    : ["eth1", "eth2"]
  port_info     :
                 - ip         : 192.168.91.99
                   default_gw : 192.168.91.101
                 - ip         : 192.168.92.99
                   default_gw : 192.168.92.106
```

C.3 Profily sieťovej prevádzky

C.3.1 Profil ipv6_stl_profile.py

```
from trex_stl_lib.api import *
import argparse


# IMIX profile - involves 3 streams of UDP packets
# 1 - 60 bytes
# 2 - 590 bytes
# 3 - 1514 bytes
```

```
class STLImix(object):

    def __init__ (self):
        # default IP range
        self.ip_range = {'src': {'start': "16.0.0.1", 'end': "16.0.0.254"},
                         'dst': {'start': "48.0.0.1",  'end': "48.0.0.254"}}

        # default IMIX properties
        self.imix_table = [ {'src': 'a000::1', 'dst':'aaaa::2',
                             'size': 60,   'pps': 28,  'isg':0 },
                            {'src': 'b000::1', 'dst':'bbbb::2',
                             'size': 590,  'pps': 16,  'isg':0.1 },
                            {'src': 'c000::1', 'dst':'cccc::2',
                             'size': 1514, 'pps': 4, 'isg':0.2 } ]


    def create_stream (self, src, dst, size, pps, isg, direction, vm ):
        # Create base packet and pad it to size
        base_pkt  = Ether()
        if direction == 0:
            base_pkt /= IPv6(src=src, dst=dst)
        else:
            base_pkt /= IPv6(src=dst, dst=src)
        base_pkt /= UDP(sport=12345, dport=12345)
        pad = max(0, size - len(base_pkt)) * 'x'

        pkt = STLPktBuilder(pkt = base_pkt/pad,
                            vm = vm)

        return STLStream(isg = isg,
                         packet = pkt,
                         mode = STLTXCont(pps = pps))


    def get_streams (self, direction, tunables, **kwargs):
        parser = argparse.ArgumentParser(
            description='Argparser for {}'.format(os.path.basename(__file__)),
            formatter_class=argparse.ArgumentDefaultsHelpFormatter)
        args = parser.parse_args(tunables)

        if direction == 0:
            src = self.ip_range['src']
            dst = self.ip_range['dst']
        else:
            src = self.ip_range['dst']
            dst = self.ip_range['src']

        # construct the base packet for the profile
        vm = STLVM()

        # define two vars (src and dst)
        vm.var(name="src",min_value=src['start'],
               max_value=src['end'],size=4,op="inc")
        vm.var(name="dst",min_value=dst['start'],
               max_value=dst['end'],size=4,op="inc")

        # write them
        vm.write(fv_name="src",pkt_offset= "IPv6.src", offset_fixup=12)
        vm.write(fv_name="dst",pkt_offset= "IPv6.dst", offset_fixup=12)
```

```
        # fix UDP checksum in HW
        vm.fix_chksum_hw(l3_offset='IPv6', l4_offset = 'UDP',
            l4_type=CTRexVmInsFixHwCs.L4_TYPE_UDP)

        # create imix streams
        return [self.create_stream(x['src'], x['dst'], x['size'],
            x['pps'],x['isg'], direction, vm) for x in self.imix_table]


# dynamic load - used for trex console or simulator
def register():
    return STLImix()
```

C.3.2 Profil ipv4_stl_profile.py

```
from trex_stl_lib.api import *
import argparse


# IMIX profile - involves 3 streams of UDP packets
# 1 - 60 bytes
# 2 - 590 bytes
# 3 - 1514 bytes
class STLImix(object):

    def __init__ (self):
        # default IP range
        self.ip_range = {'src': {'start': "16.0.0.1", 'end': "16.0.0.254"},
                         'dst': {'start': "48.0.0.1",  'end': "48.0.0.254"}}

        # default IMIX properties
        self.imix_table = [ {'size': 60,   'pps': 28,  'isg':0 },
                            {'size': 590,  'pps': 16,  'isg':0.1 },
                            {'size': 1514, 'pps': 4,   'isg':0.2 } ]


    def create_stream (self, size, pps, isg, vm ):
        # Create base packet and pad it to size
        base_pkt = Ether()/IP()/UDP()
        pad = max(0, size - len(base_pkt)) * 'x'

        pkt = STLPktBuilder(pkt = base_pkt/pad,
                            vm = vm)

        return STLStream(isg = isg,
                         packet = pkt,
                         mode = STLTXCont(pps = pps))


    def get_streams (self, direction, tunables, **kwargs):
        parser = argparse.ArgumentParser(
            description='Argparser for {}'.format(os.path.basename(__file__)),
            formatter_class=argparse.ArgumentDefaultsHelpFormatter)
        args = parser.parse_args(tunables)

        if direction == 0:
```

```
            src = self.ip_range['src']
            dst = self.ip_range['dst']
        else:
            src = self.ip_range['dst']
            dst = self.ip_range['src']

        # construct the base packet for the profile
        vm = STLVM()

        # define two vars (src and dst)
        vm.var(name="src",min_value=src['start'],max_value=src['end'],
               size=4,op="inc")
        vm.var(name="dst",min_value=dst['start'],max_value=dst['end'],
               size=4,op="inc")

        # write them
        vm.write(fv_name="src",pkt_offset= "IP.src")
        vm.write(fv_name="dst",pkt_offset= "IP.dst")

        # fix checksum
        vm.fix_chksum()

        # create imix streams
        return [self.create_stream(x['size'],
            x['pps'],x['isg'] , vm) for x in self.imix_table]



# dynamic load - used for trex console or simulator
def register():
    return STLImix()
```

D Kontajner ansible

D.1 Súbor Dockerfile

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y ansible python-ncclient wait-for-it && \
        rm -rf /var/lib/apt/lists/* && \
        echo "[local]\nlocalhost ansible_connection=local" > /etc/ansible/hosts
COPY entrypoint.sh *.yaml /
ENTRYPOINT ["/entrypoint.sh"]
```

D.2 Skript entrypoint.sh

```
#!/bin/bash

echo Waiting for Honeycomb agents to start...
wait-for-it node1:2831 -t 300
wait-for-it node2:2831 -t 300
wait-for-it node3:2831 -t 300
wait-for-it node4:2831 -t 300
wait-for-it node5:2831 -t 300
wait-for-it node6:2831 -t 300

if [ "$#" = 0 ]; then
    for playbook in /*.yaml; do
        ansible-playbook $playbook
    done
else
    for playbook in "$@"; do
        ansible-playbook $playbook
    done
fi
```

D.3 Konfiguračné playbooky

D.3.1 Playbook playbook-1-interfaces.yaml

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node1 TAP Interface to host Ubuntu system
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
```

```
          <interface>
            <name>tap0</name>
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:10:10</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.10.10</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>fd10::10</ip>
                <prefix-length>64</prefix-length>
              </address>
            </ipv6>
          </interface>
        </interfaces>
      </config>

- name: Configure Node2 TAP Interface to host Ubuntu system
  netconf_config:
    host: node2
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
            <name>tap0</name>
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:20:20</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.20.20</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>fd20::20</ip>
                <prefix-length>64</prefix-length>
              </address>
            </ipv6>
          </interface>
        </interfaces>
      </config>
```

```
- name: Configure Node3 TAP Interface to host Ubuntu system
  netconf_config:
    host: node3
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
            <name>tap0</name>
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:30:30</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.30.30</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>fd30::30</ip>
                <prefix-length>64</prefix-length>
              </address>
            </ipv6>
          </interface>
        </interfaces>
      </config>

- name: Configure Node4 TAP Interface to host Ubuntu system
  netconf_config:
    host: node4
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
            <name>tap0</name>
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:40:40</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.40.40</ip>
                <prefix-length>24</prefix-length>
```

```
            </address>
          </ipv4>
          <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
            <address>
              <ip>fd40::40</ip>
              <prefix-length>64</prefix-length>
            </address>
          </ipv6>
        </interface>
      </interfaces>
    </config>

- name: Configure Node5 TAP Interface to host Ubuntu system
  netconf_config:
    host: node5
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
            <name>tap0</name>
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:50:50</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.50.50</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>fd50::50</ip>
                <prefix-length>64</prefix-length>
              </address>
            </ipv6>
          </interface>
        </interfaces>
      </config>

- name: Configure Node6 TAP Interface to host Ubuntu system
  netconf_config:
    host: node6
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
            <name>tap0</name>
```

```
            <type xmlns:v3po="urn:opendaylight:params:xml:ns:yang:v3po">
              v3po:tap
            </type>
            <tap xmlns="urn:opendaylight:params:xml:ns:yang:v3po">
              <tap-name>vpp</tap-name>
              <mac>00:00:00:00:60:60</mac>
            </tap>
            <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>192.168.60.60</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
            <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
              <address>
                <ip>fd60::60</ip>
                <prefix-length>64</prefix-length>
              </address>
            </ipv6>
          </interface>
        </interfaces>
      </config>
```

D.3.2 Playbook playbook-2-sid-definition.yaml

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node1 static route to host system Ubuntu and SRv6 SIDs
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
            <control-plane-protocols>
              <control-plane-protocol>
                <type>static</type>
                <name>learned-protocol-0</name>
                <vpp-protocol-attributes
                    xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                  <primary-vrf>0</primary-vrf>
                </vpp-protocol-attributes>

                <static-routes>
                  <ipv6 xmlns="urn:ietf:params:
                               xml:ns:yang:ietf-ipv6-unicast-routing">
                    <route>
                      <destination-prefix>::/0</destination-prefix>
                      <description>Default route</description>
                      <next-hop>
                        <outgoing-interface>tap0</outgoing-interface>
                        <next-hop-address>fd10::1</next-hop-address>
```

```
          </next-hop>
        </route>
      </ipv6>
    </static-routes>
  </control-plane-protocol>
</control-plane-protocols>
<srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
  <encapsulation>
    <source-address>fd10::1</source-address>
    <ip-ttl-propagation>false</ip-ttl-propagation>
  </encapsulation>
  <locators>
    <locator>
      <name>fd11::</name>
      <enable>true</enable>
      <is-default>false</is-default>
      <prefix>
        <address>fd11::</address>
        <length>64</length>-
      </prefix>
      <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
        <local-sids>
          <sid>
            <opcode>256</opcode> <!-- ::100 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
                                       xml:ns:yang:ietf-srv6-types">
              ietf-srv6-types:End
            </end-behavior-type>
            <end/>
          </sid>
          <sid>
            <opcode>260</opcode> <!-- ::104 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
                                       xml:ns:yang:ietf-srv6-types">
              ietf-srv6-types:End.DX4
            </end-behavior-type>
            <end-dx4>
              <paths>
                <path>
                  <path-index>1</path-index>
                  <interface>tap0</interface>
                  <next-hop>192.168.10.1</next-hop>
                  <weight>1</weight>
                  <role>PRIMARY</role>
                </path>
              </paths>
            </end-dx4>
          </sid>
          <sid>
            <opcode>262</opcode> <!-- ::106 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
                                       xml:ns:yang:ietf-srv6-types">
              ietf-srv6-types:End.DX6
            </end-behavior-type>
            <end-dx6>
              <paths>
```

```
                        <path>
                          <path-index>1</path-index>
                          <interface>tap0</interface>
                          <next-hop>fd10::1</next-hop>
                          <weight>1</weight>
                          <role>PRIMARY</role>
                        </path>
                      </paths>
                    </end-dx6>
                  </sid>
                </local-sids>
              </static>
              <fib-table
                  xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
                <table-id>0</table-id>
                <address-family
                    xmlns:vpp-fib-table-management="urn:opendaylight:params:
                    xml:ns:yang:vpp-fib-table-management">
                  vpp-fib-table-management:ipv6
                </address-family>
              </fib-table>
            </locator>
          </locators>
        </srv6>
      </routing>
    </config>

- name: Configure Node2 static route to host system Ubuntu and SRv6 SIDs
  netconf_config:
    host: node2
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
          <control-plane-protocols>
            <control-plane-protocol>
              <type>static</type>
              <name>learned-protocol-0</name>
              <vpp-protocol-attributes
                  xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                <primary-vrf>0</primary-vrf>
              </vpp-protocol-attributes>

              <static-routes>
                <ipv6 xmlns="urn:ietf:params:
                             xml:ns:yang:ietf-ipv6-unicast-routing">
                  <route>
                    <destination-prefix>::/0</destination-prefix>
                    <description>Default route</description>
                    <next-hop>
                      <outgoing-interface>tap0</outgoing-interface>
                      <next-hop-address>fd20::1</next-hop-address>
                    </next-hop>
                  </route>
                </ipv6>
              </static-routes>
```

```
            </control-plane-protocol>
          </control-plane-protocols>
          <srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
            <encapsulation>
              <source-address>fd20::1</source-address>
              <ip-ttl-propagation>false</ip-ttl-propagation>
            </encapsulation>
            <locators>
              <locator>
                <name>fd22::</name>
                <enable>true</enable>
                <is-default>false</is-default>
                <prefix>
                  <address>fd22::</address>
                  <length>64</length>-
                </prefix>
                <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
                  <local-sids>
                    <sid>
                      <opcode>256</opcode> <!-- ::100 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:
                                                 xml:ns:yang:ietf-srv6-types">
                        ietf-srv6-types:End
                      </end-behavior-type>
                      <end/>
                    </sid>
                  </local-sids>
                </static>
                <fib-table
                    xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
                  <table-id>0</table-id>
                  <address-family
                      xmlns:vpp-fib-table-management="urn:opendaylight:params:
                      xml:ns:yang:vpp-fib-table-management">
                    vpp-fib-table-management:ipv6
                  </address-family>
                </fib-table>
              </locator>
            </locators>
          </srv6>
        </routing>
      </config>

- name: Configure Node3 static route to host system Ubuntu and SRv6 SIDs
  netconf_config:
    host: node3
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
          <control-plane-protocols>
            <control-plane-protocol>
              <type>static</type>
              <name>learned-protocol-0</name>
              <vpp-protocol-attributes
```

```
          xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
        <primary-vrf>0</primary-vrf>
      </vpp-protocol-attributes>

      <static-routes>
        <ipv6 xmlns="urn:ietf:params:
                    xml:ns:yang:ietf-ipv6-unicast-routing">
          <route>
            <destination-prefix>::/0</destination-prefix>
            <description>Default route</description>
            <next-hop>
              <outgoing-interface>tap0</outgoing-interface>
              <next-hop-address>fd30::1</next-hop-address>
            </next-hop>
          </route>
        </ipv6>
      </static-routes>
    </control-plane-protocol>
  </control-plane-protocols>
  <srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
    <encapsulation>
      <source-address>fd30::1</source-address>
      <ip-ttl-propagation>false</ip-ttl-propagation>
    </encapsulation>
    <locators>
      <locator>
        <name>fd33::</name>
        <enable>true</enable>
        <is-default>false</is-default>
        <prefix>
          <address>fd33::</address>
          <length>64</length>-
        </prefix>
        <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
          <local-sids>
            <sid>
              <opcode>256</opcode> <!-- ::100 -->
              <end-behavior-type
                  xmlns:ietf-srv6-types="urn:ietf:params:
                                         xml:ns:yang:ietf-srv6-types">
                ietf-srv6-types:End
              </end-behavior-type>
              <end/>
            </sid>
          </local-sids>
        </static>
        <fib-table
            xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
          <table-id>0</table-id>
          <address-family
              xmlns:vpp-fib-table-management="urn:opendaylight:params:
              xml:ns:yang:vpp-fib-table-management">
            vpp-fib-table-management:ipv6
          </address-family>
        </fib-table>
      </locator>
    </locators>
  </srv6>
</routing>
```

```
      </config>

- name: Configure Node4 static route to host system Ubuntu and SRv6 SIDs
  netconf_config:
    host: node4
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
          <control-plane-protocols>
            <control-plane-protocol>
              <type>static</type>
              <name>learned-protocol-0</name>
              <vpp-protocol-attributes
                  xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                <primary-vrf>0</primary-vrf>
              </vpp-protocol-attributes>

              <static-routes>
                <ipv6 xmlns="urn:ietf:params:
                             xml:ns:yang:ietf-ipv6-unicast-routing">
                  <route>
                    <destination-prefix>::/0</destination-prefix>
                    <description>Default route</description>
                    <next-hop>
                      <outgoing-interface>tap0</outgoing-interface>
                      <next-hop-address>fd40::1</next-hop-address>
                    </next-hop>
                  </route>
                </ipv6>
              </static-routes>
            </control-plane-protocol>
          </control-plane-protocols>
          <srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
            <encapsulation>
              <source-address>fd40::1</source-address>
              <ip-ttl-propagation>false</ip-ttl-propagation>
            </encapsulation>
            <locators>
              <locator>
                <name>fd44::</name>
                <enable>true</enable>
                <is-default>false</is-default>
                <prefix>
                  <address>fd44::</address>
                  <length>64</length>-
                </prefix>
                <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
                  <local-sids>
                    <sid>
                      <opcode>256</opcode> <!-- ::100 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:
                                                 xml:ns:yang:ietf-srv6-types">
                        ietf-srv6-types:End
                      </end-behavior-type>
```

```
                    <end/>
                  </sid>
                </local-sids>
              </static>
              <fib-table
                  xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
                <table-id>0</table-id>
                <address-family
                    xmlns:vpp-fib-table-management="urn:opendaylight:params:
                    xml:ns:yang:vpp-fib-table-management">
                  vpp-fib-table-management:ipv6
                </address-family>
              </fib-table>
            </locator>
          </locators>
        </srv6>
      </routing>
    </config>

- name: Configure Node5 static route to host system Ubuntu and SRv6 SIDs
  netconf_config:
    host: node5
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
          <control-plane-protocols>
            <control-plane-protocol>
              <type>static</type>
              <name>learned-protocol-0</name>
              <vpp-protocol-attributes
                  xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                <primary-vrf>0</primary-vrf>
              </vpp-protocol-attributes>

              <static-routes>
                <ipv6 xmlns="urn:ietf:params:
                             xml:ns:yang:ietf-ipv6-unicast-routing">
                  <route>
                    <destination-prefix>::/0</destination-prefix>
                    <description>Default route</description>
                    <next-hop>
                      <outgoing-interface>tap0</outgoing-interface>
                      <next-hop-address>fd50::1</next-hop-address>
                    </next-hop>
                  </route>
                </ipv6>
              </static-routes>
            </control-plane-protocol>
          </control-plane-protocols>
          <srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
            <encapsulation>
              <source-address>fd50::1</source-address>
              <ip-ttl-propagation>false</ip-ttl-propagation>
            </encapsulation>
            <locators>
```

```
              <locator>
                <name>fd55::</name>
                <enable>true</enable>
                <is-default>false</is-default>
                <prefix>
                  <address>fd55::</address>
                  <length>64</length>-
                </prefix>
                <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
                  <local-sids>
                    <sid>
                      <opcode>256</opcode> <!-- ::100 -->
                      <end-behavior-type
                          xmlns:ietf-srv6-types="urn:ietf:params:
                                                 xml:ns:yang:ietf-srv6-types">
                        ietf-srv6-types:End
                      </end-behavior-type>
                      <end/>
                    </sid>
                  </local-sids>
                </static>
                <fib-table
                    xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
                  <table-id>0</table-id>
                  <address-family
                      xmlns:vpp-fib-table-management="urn:opendaylight:params:
                      xml:ns:yang:vpp-fib-table-management">
                    vpp-fib-table-management:ipv6
                  </address-family>
                </fib-table>
              </locator>
            </locators>
          </srv6>
        </routing>
      </config>

- name: Configure Node6 static route to host system Ubuntu and SRv6 SIDs
  netconf_config:
    host: node6
    port: 2831
    hostkey_verify: no
    username: admin
    password: admin
    xml: |
      <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
        <routing xmlns="urn:ietf:params:xml:ns:yang:ietf-routing">
          <control-plane-protocols>
            <control-plane-protocol>
              <type>static</type>
              <name>learned-protocol-0</name>
              <vpp-protocol-attributes
                  xmlns="urn:ietf:params:xml:ns:yang:vpp-routing">
                <primary-vrf>0</primary-vrf>
              </vpp-protocol-attributes>

              <static-routes>
                <ipv6 xmlns="urn:ietf:params:
                             xml:ns:yang:ietf-ipv6-unicast-routing">
                  <route>
```

```
          <destination-prefix>::/0</destination-prefix>
          <description>Default route</description>
          <next-hop>
            <outgoing-interface>tap0</outgoing-interface>
            <next-hop-address>fd60::1</next-hop-address>
          </next-hop>
        </route>
      </ipv6>
    </static-routes>
  </control-plane-protocol>
</control-plane-protocols>
<srv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-base">
  <encapsulation>
    <source-address>fd60::1</source-address>
    <ip-ttl-propagation>false</ip-ttl-propagation>
  </encapsulation>
  <locators>
    <locator>
      <name>fd66::</name>
      <enable>true</enable>
      <is-default>false</is-default>
      <prefix>
        <address>fd66::</address>
        <length>64</length>-
      </prefix>
      <static xmlns="urn:ietf:params:xml:ns:yang:ietf-srv6-static">
        <local-sids>
          <sid>
            <opcode>256</opcode> <!-- ::100 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
                                       xml:ns:yang:ietf-srv6-types">
              ietf-srv6-types:End
            </end-behavior-type>
            <end/>
          </sid>
          <sid>
            <opcode>260</opcode> <!-- ::104 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
                                       xml:ns:yang:ietf-srv6-types">
              ietf-srv6-types:End.DX4
            </end-behavior-type>
            <end-dx4>
              <paths>
                <path>
                  <path-index>1</path-index>
                  <interface>tap0</interface>
                  <next-hop>192.168.60.1</next-hop>
                  <weight>1</weight>
                  <role>PRIMARY</role>
                </path>
              </paths>
            </end-dx4>
          </sid>
          <sid>
            <opcode>262</opcode> <!-- ::106 -->
            <end-behavior-type
                xmlns:ietf-srv6-types="urn:ietf:params:
```

```
                                        xml:ns:yang:ietf-srv6-types">
                    ietf-srv6-types:End.DX6
                  </end-behavior-type>
                  <end-dx6>
                    <paths>
                      <path>
                        <path-index>1</path-index>
                        <interface>tap0</interface>
                        <next-hop>fd60::1</next-hop>
                        <weight>1</weight>
                        <role>PRIMARY</role>
                      </path>
                    </paths>
                  </end-dx6>
                </sid>
              </local-sids>
            </static>
            <fib-table
                xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-ietf-srv6-base">
              <table-id>0</table-id>
              <address-family
                  xmlns:vpp-fib-table-management="urn:opendaylight:params:
                  xml:ns:yang:vpp-fib-table-management">
                vpp-fib-table-management:ipv6
              </address-family>
            </fib-table>
          </locator>
        </locators>
      </srv6>
    </routing>
  </config>
```

D.3.3 Playbook playbook-3-srv6-policies.yaml

```
- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node1 SRv6 Policies and traffic steering
    netconf_config:
      host: node1
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <segment-routing xmlns="http://cisco.com/ns/yang/oc-srte-policy">
            <traffic-engineering>
              <named-segment-lists>
                <named-segment-list>
                  <name>fd11:1066::1-1</name>
                  <config>
                    <name>fd11:1066::1-1</name>
                  </config>
                  <segments>
                    <segment>
                      <index>1</index>
```

```
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd22::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd55::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>3</index>
      <config>
        <index>3</index>
        <type>type-2</type>
        <sid-value>fd66::106</sid-value>
      </config>
    </segment>
  </segments>
</named-segment-list>
<named-segment-list>
  <name>fd11:1066::2-1</name>
  <config>
    <name>fd11:1066::2-1</name>
  </config>
  <segments>
    <segment>
      <index>1</index>
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd33::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd44::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>3</index>
      <config>
        <index>3</index>
        <type>type-2</type>
        <sid-value>fd66::106</sid-value>
      </config>
    </segment>
  </segments>
</named-segment-list>
<named-segment-list>
  <name>fd11:1166::3-1</name>
  <config>
```

```
    <name>fd11:1166::3-1</name>
  </config>
  <segments>
    <segment>
      <index>1</index>
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd22::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd44::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>3</index>
      <config>
        <index>3</index>
        <type>type-2</type>
        <sid-value>fd55::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>4</index>
      <config>
        <index>4</index>
        <type>type-2</type>
        <sid-value>fd66::100</sid-value>
      </config>
    </segment>
  </segments>
</named-segment-list>
<named-segment-list>
  <name>fd11:1046::4-1</name>
  <config>
    <name>fd11:1046::4-1</name>
  </config>
  <segments>
    <segment>
      <index>1</index>
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd33::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd55::100</sid-value>
      </config>
    </segment>
```

```
      <segment>
        <index>3</index>
        <config>
          <index>3</index>
          <type>type-2</type>
          <sid-value>fd44::100</sid-value>
        </config>
      </segment>
      <segment>
        <index>4</index>
        <config>
          <index>4</index>
          <type>type-2</type>
          <sid-value>fd66::104</sid-value>
        </config>
      </segment>
    </segments>
  </named-segment-list>
</named-segment-lists>
<policies>
  <policy>
    <config>
      <name>fd11:1066::1</name>
      <color>1</color>
      <endpoint>fd60::60</endpoint>
    </config>
    <color>1</color>
    <endpoint>fd60::60</endpoint>
    <candidate-paths>
      <candidate-path>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <preference>1</preference>
        <distinguisher>0</distinguisher>
        <config>
          <name>candidatePath1</name>
          <provisioning-method>
            provisioning-method-config
          </provisioning-method>
          <computation-method>
            path-explicitly-defined
          </computation-method>
          <preference>1</preference>
          <distinguisher>0</distinguisher>
        </config>
        <binding-sid>
          <config>
            <alloc-mode>explicit</alloc-mode>
            <type>srv6</type>
            <value>fd11:1066::1</value>
          </config>
        </binding-sid>
        <segment-lists>
          <segment-list>
            <name>fd11:1066::1-1</name>
            <config>
              <name>fd11:1066::1-1</name>
```

```
            <weight>1</weight>
          </config>
        </segment-list>
      </segment-lists>
    </candidate-path>
  </candidate-paths>
  <autoroute-include>
    <config>
      <metric-type>constant</metric-type>
      <metric-constant>0</metric-constant>
    </config>
    <prefixes>
      <config>
        <prefixes-all>false</prefixes-all>
      </config>
      <prefix>
        <ip-prefix>aaaa::/16</ip-prefix>
        <config>
          <ip-prefix>aaaa::/16</ip-prefix>
        </config>
      </prefix>
    </prefixes>
  </autoroute-include>
  <binding-sid>
    <config>
      <alloc-mode>explicit</alloc-mode>
      <type>srv6</type>
      <value>fd11:1066::1</value>
    </config>
  </binding-sid>
  <vpp-sr-policy
      xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
    <config>
      <policy-type>Default</policy-type>
      <policy-behavior>Encapsulation</policy-behavior>
      <table-id>0</table-id>
      <address-family
          xmlns:fib="urn:opendaylight:params:
                     xml:ns:yang:vpp-fib-table-management">
        fib:ipv6
      </address-family>
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd11:1066::2</name>
    <color>2</color>
    <endpoint>fd60::60</endpoint>
  </config>
  <color>2</color>
  <endpoint>fd60::60</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
```

```
    <distinguisher>0</distinguisher>
    <config>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <computation-method>
        path-explicitly-defined
      </computation-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
    </config>
    <binding-sid>
      <config>
        <alloc-mode>explicit</alloc-mode>
        <type>srv6</type>
        <value>fd11:1066::2</value>
      </config>
    </binding-sid>
    <segment-lists>
      <segment-list>
        <name>fd11:1066::2-1</name>
        <config>
          <name>fd11:1066::2-1</name>
          <weight>1</weight>
        </config>
      </segment-list>
    </segment-lists>
  </candidate-path>
</candidate-paths>
<autoroute-include>
  <config>
    <metric-type>constant</metric-type>
    <metric-constant>0</metric-constant>
  </config>
  <prefixes>
    <config>
      <prefixes-all>false</prefixes-all>
    </config>
    <prefix>
      <ip-prefix>bbbb::/16</ip-prefix>
      <config>
        <ip-prefix>bbbb::/16</ip-prefix>
      </config>
    </prefix>
  </prefixes>
</autoroute-include>
<binding-sid>
  <config>
    <alloc-mode>explicit</alloc-mode>
    <type>srv6</type>
    <value>fd11:1066::2</value>
  </config>
</binding-sid>
<vpp-sr-policy
    xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
  <config>
    <policy-type>Default</policy-type>
    <policy-behavior>Encapsulation</policy-behavior>
```

```
      <table-id>0</table-id>
      <address-family
          xmlns:fib="urn:opendaylight:params:
                     xml:ns:yang:vpp-fib-table-management">
        fib:ipv6
      </address-family>
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd11:1166::3</name>
    <color>3</color>
    <endpoint>fd60::60</endpoint>
  </config>
  <color>3</color>
  <endpoint>fd60::60</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
      <config>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <computation-method>
          path-explicitly-defined
        </computation-method>
        <preference>1</preference>
        <distinguisher>0</distinguisher>
      </config>
      <binding-sid>
        <config>
          <alloc-mode>explicit</alloc-mode>
          <type>srv6</type>
          <value>fd11:1166::3</value>
        </config>
      </binding-sid>
      <segment-lists>
        <segment-list>
          <name>fd11:1166::3-1</name>
          <config>
            <name>fd11:1166::3-1</name>
            <weight>1</weight>
          </config>
        </segment-list>
      </segment-lists>
    </candidate-path>
  </candidate-paths>
  <autoroute-include>
    <config>
      <metric-type>constant</metric-type>
      <metric-constant>0</metric-constant>
    </config>
```

```
    <prefixes>
      <config>
        <prefixes-all>false</prefixes-all>
      </config>
      <prefix>
        <ip-prefix>cccc::/16</ip-prefix>
        <config>
          <ip-prefix>cccc::/16</ip-prefix>
        </config>
      </prefix>
    </prefixes>
  </autoroute-include>
  <binding-sid>
    <config>
      <alloc-mode>explicit</alloc-mode>
      <type>srv6</type>
      <value>fd11:1166::3</value>
    </config>
  </binding-sid>
  <vpp-sr-policy
      xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
    <config>
      <policy-type>Default</policy-type>
      <policy-behavior>SegmentRoutingHeaderInsert</policy-behavior>
      <table-id>0</table-id>
      <address-family
          xmlns:fib="urn:opendaylight:params:
                     xml:ns:yang:vpp-fib-table-management">
        fib:ipv6
      </address-family>
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd11:1046::4</name>
    <color>4</color>
    <endpoint>192.168.60.60</endpoint>
  </config>
  <color>4</color>
  <endpoint>192.168.60.60</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
      <config>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <computation-method>
          path-explicitly-defined
        </computation-method>
        <preference>1</preference>
        <distinguisher>0</distinguisher>
```

```
          </config>
          <binding-sid>
            <config>
              <alloc-mode>explicit</alloc-mode>
              <type>srv6</type>
              <value>fd11:1046::4</value>
            </config>
          </binding-sid>
          <segment-lists>
            <segment-list>
              <name>fd11:1046::4-1</name>
              <config>
                <name>fd11:1046::4-1</name>
                <weight>1</weight>
              </config>
            </segment-list>
          </segment-lists>
        </candidate-path>
      </candidate-paths>
      <autoroute-include>
        <config>
          <metric-type>constant</metric-type>
          <metric-constant>0</metric-constant>
        </config>
        <prefixes>
          <config>
            <prefixes-all>false</prefixes-all>
          </config>
          <prefix>
            <ip-prefix>48.0.0.0/24</ip-prefix>
            <config>
              <ip-prefix>48.0.0.0/24</ip-prefix>
            </config>
          </prefix>
        </prefixes>
      </autoroute-include>
      <binding-sid>
        <config>
          <alloc-mode>explicit</alloc-mode>
          <type>srv6</type>
          <value>fd11:1046::4</value>
        </config>
      </binding-sid>
      <vpp-sr-policy
          xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
        <config>
          <policy-type>Default</policy-type>
          <policy-behavior>Encapsulation</policy-behavior>
          <table-id>0</table-id>
          <address-family
              xmlns:fib="urn:opendaylight:params:
                         xml:ns:yang:vpp-fib-table-management">
            fib:ipv4
          </address-family>
        </config>
      </vpp-sr-policy>
    </policy>
  </policies>
</traffic-engineering>
```

```
          </segment-routing>
        </config>

- hosts: localhost
  gather_facts: no
  connection: local
  tasks:
  - name: Configure Node6 SRv6 Policies and traffic steering
    netconf_config:
      host: node6
      port: 2831
      hostkey_verify: no
      username: admin
      password: admin
      xml: |
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <segment-routing xmlns="http://cisco.com/ns/yang/oc-srte-policy">
            <traffic-engineering>
              <named-segment-lists>
                <named-segment-list>
                  <name>fd66:6061::1-1</name>
                  <config>
                    <name>fd66:6061::1-1</name>
                  </config>
                  <segments>
                    <segment>
                      <index>1</index>
                      <config>
                        <index>1</index>
                        <type>type-2</type>
                        <sid-value>fd55::100</sid-value>
                      </config>
                    </segment>
                    <segment>
                      <index>2</index>
                      <config>
                        <index>2</index>
                        <type>type-2</type>
                        <sid-value>fd22::100</sid-value>
                      </config>
                    </segment>
                    <segment>
                      <index>3</index>
                      <config>
                        <index>3</index>
                        <type>type-2</type>
                        <sid-value>fd11::106</sid-value>
                      </config>
                    </segment>
                  </segments>
                </named-segment-list>
                <named-segment-list>
                  <name>fd66:6061::2-1</name>
                  <config>
                    <name>fd66:6061::2-1</name>
                  </config>
                  <segments>
                    <segment>
                      <index>1</index>
```

```
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd44::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd33::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>3</index>
      <config>
        <index>3</index>
        <type>type-2</type>
        <sid-value>fd11::106</sid-value>
      </config>
    </segment>
  </segments>
</named-segment-list>
<named-segment-list>
  <name>fd66:6161::3-1</name>
  <config>
    <name>fd66:6161::3-1</name>
  </config>
  <segments>
    <segment>
      <index>1</index>
      <config>
        <index>1</index>
        <type>type-2</type>
        <sid-value>fd55::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>2</index>
      <config>
        <index>2</index>
        <type>type-2</type>
        <sid-value>fd44::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>3</index>
      <config>
        <index>3</index>
        <type>type-2</type>
        <sid-value>fd22::100</sid-value>
      </config>
    </segment>
    <segment>
      <index>4</index>
      <config>
        <index>4</index>
        <type>type-2</type>
```

```
          <sid-value>fd11::100</sid-value>
        </config>
      </segment>
    </segments>
  </named-segment-list>
  <named-segment-list>
    <name>fd66:6041::4-1</name>
    <config>
      <name>fd66:6041::4-1</name>
    </config>
    <segments>
      <segment>
        <index>1</index>
        <config>
          <index>1</index>
          <type>type-2</type>
          <sid-value>fd44::100</sid-value>
        </config>
      </segment>
      <segment>
        <index>2</index>
        <config>
          <index>2</index>
          <type>type-2</type>
          <sid-value>fd55::100</sid-value>
        </config>
      </segment>
      <segment>
        <index>3</index>
        <config>
          <index>3</index>
          <type>type-2</type>
          <sid-value>fd33::100</sid-value>
        </config>
      </segment>
      <segment>
        <index>4</index>
        <config>
          <index>4</index>
          <type>type-2</type>
          <sid-value>fd11::104</sid-value>
        </config>
      </segment>
    </segments>
  </named-segment-list>
</named-segment-lists>
<policies>
  <policy>
    <config>
      <name>fd66:6061::1</name>
      <color>1</color>
      <endpoint>fd10::10</endpoint>
    </config>
    <color>1</color>
    <endpoint>fd10::10</endpoint>
    <candidate-paths>
      <candidate-path>
        <name>candidatePath1</name>
        <provisioning-method>
```

```
      provisioning-method-config
    </provisioning-method>
    <preference>1</preference>
    <distinguisher>0</distinguisher>
    <config>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <computation-method>
        path-explicitly-defined
      </computation-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
    </config>
    <binding-sid>
      <config>
        <alloc-mode>explicit</alloc-mode>
        <type>srv6</type>
        <value>fd66:6061::1</value>
      </config>
    </binding-sid>
    <segment-lists>
      <segment-list>
        <name>fd66:6061::1-1</name>
        <config>
          <name>fd66:6061::1-1</name>
          <weight>1</weight>
        </config>
      </segment-list>
    </segment-lists>
  </candidate-path>
</candidate-paths>
<autoroute-include>
  <config>
    <metric-type>constant</metric-type>
    <metric-constant>0</metric-constant>
  </config>
  <prefixes>
    <config>
      <prefixes-all>false</prefixes-all>
    </config>
    <prefix>
      <ip-prefix>a000::/16</ip-prefix>
      <config>
        <ip-prefix>a000::/16</ip-prefix>
      </config>
    </prefix>
  </prefixes>
</autoroute-include>
<binding-sid>
  <config>
    <alloc-mode>explicit</alloc-mode>
    <type>srv6</type>
    <value>fd66:6061::1</value>
  </config>
</binding-sid>
<vpp-sr-policy
    xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
```

```
    <config>
      <policy-type>Default</policy-type>
      <policy-behavior>Encapsulation</policy-behavior>
      <table-id>0</table-id>
      <address-family
          xmlns:fib="urn:opendaylight:params:
                     xml:ns:yang:vpp-fib-table-management">
        fib:ipv6
      </address-family>
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd66:6061::2</name>
    <color>2</color>
    <endpoint>fd10::10</endpoint>
  </config>
  <color>2</color>
  <endpoint>fd10::10</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
      <config>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <computation-method>
          path-explicitly-defined
        </computation-method>
        <preference>1</preference>
        <distinguisher>0</distinguisher>
      </config>
      <binding-sid>
        <config>
          <alloc-mode>explicit</alloc-mode>
          <type>srv6</type>
          <value>fd66:6061::2</value>
        </config>
      </binding-sid>
      <segment-lists>
        <segment-list>
          <name>fd66:6061::2-1</name>
          <config>
            <name>fd66:6061::2-1</name>
            <weight>1</weight>
          </config>
        </segment-list>
      </segment-lists>
    </candidate-path>
  </candidate-paths>
  <autoroute-include>
    <config>
```

```
      <metric-type>constant</metric-type>
      <metric-constant>0</metric-constant>
    </config>
    <prefixes>
      <config>
        <prefixes-all>false</prefixes-all>
      </config>
      <prefix>
        <ip-prefix>b000::/16</ip-prefix>
        <config>
          <ip-prefix>b000::/16</ip-prefix>
        </config>
      </prefix>
    </prefixes>
  </autoroute-include>
  <binding-sid>
    <config>
      <alloc-mode>explicit</alloc-mode>
      <type>srv6</type>
      <value>fd66:6061::2</value>
    </config>
  </binding-sid>
  <vpp-sr-policy
      xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
    <config>
      <policy-type>Default</policy-type>
      <policy-behavior>Encapsulation</policy-behavior>
      <table-id>0</table-id>
      <address-family
          xmlns:fib="urn:opendaylight:params:
                     xml:ns:yang:vpp-fib-table-management">
        fib:ipv6
      </address-family>
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd66:6161::3</name>
    <color>3</color>
    <endpoint>fd10::10</endpoint>
  </config>
  <color>3</color>
  <endpoint>fd10::10</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
      <config>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <computation-method>
          path-explicitly-defined
```

```
      </computation-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
    </config>
    <binding-sid>
      <config>
        <alloc-mode>explicit</alloc-mode>
        <type>srv6</type>
        <value>fd66:6161::3</value>
      </config>
    </binding-sid>
    <segment-lists>
      <segment-list>
        <name>fd66:6161::3-1</name>
        <config>
          <name>fd66:6161::3-1</name>
          <weight>1</weight>
        </config>
      </segment-list>
    </segment-lists>
  </candidate-path>
</candidate-paths>
<autoroute-include>
  <config>
    <metric-type>constant</metric-type>
    <metric-constant>0</metric-constant>
  </config>
  <prefixes>
    <config>
      <prefixes-all>false</prefixes-all>
    </config>
    <prefix>
      <ip-prefix>c000::/16</ip-prefix>
      <config>
        <ip-prefix>c000::/16</ip-prefix>
      </config>
    </prefix>
  </prefixes>
</autoroute-include>
<binding-sid>
  <config>
    <alloc-mode>explicit</alloc-mode>
    <type>srv6</type>
    <value>fd66:6161::3</value>
  </config>
</binding-sid>
<vpp-sr-policy
    xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
  <config>
    <policy-type>Default</policy-type>
    <policy-behavior>
      SegmentRoutingHeaderInsert
    </policy-behavior>
    <table-id>0</table-id>
    <address-family
        xmlns:fib="urn:opendaylight:params:
                   xml:ns:yang:vpp-fib-table-management">
      fib:ipv6
    </address-family>
```

```
    </config>
  </vpp-sr-policy>
</policy>
<policy>
  <config>
    <name>fd66:6041::4</name>
    <color>4</color>
    <endpoint>fd10::10</endpoint>
  </config>
  <color>4</color>
  <endpoint>fd10::10</endpoint>
  <candidate-paths>
    <candidate-path>
      <name>candidatePath1</name>
      <provisioning-method>
        provisioning-method-config
      </provisioning-method>
      <preference>1</preference>
      <distinguisher>0</distinguisher>
      <config>
        <name>candidatePath1</name>
        <provisioning-method>
          provisioning-method-config
        </provisioning-method>
        <computation-method>
          path-explicitly-defined
        </computation-method>
        <preference>1</preference>
        <distinguisher>0</distinguisher>
      </config>
      <binding-sid>
        <config>
          <alloc-mode>explicit</alloc-mode>
          <type>srv6</type>
          <value>fd66:6041::4</value>
        </config>
      </binding-sid>
      <segment-lists>
        <segment-list>
          <name>fd66:6041::4-1</name>
          <config>
            <name>fd66:6041::4-1</name>
            <weight>1</weight>
          </config>
        </segment-list>
      </segment-lists>
    </candidate-path>
  </candidate-paths>
  <autoroute-include>
    <config>
      <metric-type>constant</metric-type>
      <metric-constant>0</metric-constant>
    </config>
    <prefixes>
      <config>
        <prefixes-all>false</prefixes-all>
      </config>
      <prefix>
        <ip-prefix>16.0.0.0/24</ip-prefix>
```

```
            <config>
              <ip-prefix>16.0.0.0/24</ip-prefix>
            </config>
          </prefix>
        </prefixes>
      </autoroute-include>
      <binding-sid>
        <config>
          <alloc-mode>explicit</alloc-mode>
          <type>srv6</type>
          <value>fd66:6041::4</value>
        </config>
      </binding-sid>
      <vpp-sr-policy
          xmlns="urn:hc2vpp:params:xml:ns:yang:vpp-oc-srte-policy">
        <config>
          <policy-type>Default</policy-type>
          <policy-behavior>Encapsulation</policy-behavior>
          <table-id>0</table-id>
          <address-family
              xmlns:fib="urn:opendaylight:params:
                         xml:ns:yang:vpp-fib-table-management">
            fib:ipv6
          </address-family>
        </config>
      </vpp-sr-policy>
    </policy>
  </policies>
 </traffic-engineering>
</segment-routing>
</config>
```

E Obsah elektronickej prílohy

```
/
├── ansible/
│   ├── Dockerfile
│   ├── entrypoint.sh
│   ├── playbook-1-interfaces.yaml
│   ├── palybook-2-sid-definition.yaml
│   └── playbook-3-srv6-policices.yaml
├── frr/
│   ├── daemons
│   ├── frr1.conf
│   ├── frr2.conf
│   ├── frr3.conf
│   ├── frr4.conf
│   ├── frr5.conf
│   └── frr6.conf
├── trex/
│   ├── Dockerfile
│   ├── ipv4_stl_profile.py
│   ├── ipv6_stl_profile.py
│   └── trex_cfg.yaml
├── docker-compose.yaml
├── Dockerfile
├── entrypoint.sh
└── topology-create.sh
```