Vilniaus universiteto
Fizikos fakulteto
Cheminės fizikos institutas

Karolis Beliašas

AUTOMATIZUOTAS TRADICINIŲ ĮRENGINIŲ KONFIGŪRAVIMAS HIBRIDINIUOSE TINKLUOSE

Bakalauro studijų baigiamasis darbas

kompiuterinė fizikos ir modeliavimo
studijų programa

Studentas	Karolis Beliašas
Leista ginti	2019-05-22
Darbo vadovas	Doc. Feliksas Kuliešius
Recenzentas	Doc. Justinas Čeponkus
Instituto direktorius	Prof. Valdas Šablinskas

Vilnius 2019

Turinys

Darbe naudojamų trumpinių ir sąvokų žodynas 3

Įvadas 4

Tradicinių įrenginių konfigūravimo problemų apžvalga hibridiniame tinkle 6

Programine įranga apibrėžtas tinklas 7

OpenFlow 8

Loginė OpenFlow komutatoriaus architektūra 8

Srauto lentelės komponentai 9

Srauto lentelių komandų grandinė 11

OpenFlow protokolas 13

Topologijos nustatymas tradiciniuose tinkluose 15

LLDP 15

SNMP 17

Tradicinių įrenginių nuotolinis konfigūravimas 19

Netconf 19

Topologijos nustatymas OpenFlow tinkluose 21

OpenDayLight kontroleris 22

Tradicinių įrenginių automatizuoto konfigūravimo realizacija 23

Autonet algoritmas 24

Autonet programa 26

Perspektyviniai programos patobulinimai 29

Pagrindiniai rezultatai ir išvados 30

Literatūra 31

Santrauka 32

Summary 33

1 priedas. Autonet programos kodas 34

Darbe naudojamų trumpinių ir sąvokų žodynas

API (angl. *application programming interface*) – programuojama aplikacijos sąsaja;

ARP (angl. *addrezz resolution protocol*) – adreso rezoliucijos protokolas;

BDDP (angl. *broadcast domain discovery protocol*) – transliacijos domeno aptikimo protokolas;

BGP (angl. *border gateway protocol*) – kraštinio tinklo sietuvo protokolas;

BoS (angl. *bottom of stack*) – dėklo apačia;

CDP (angl. *Cisco discovey protocol*) – Cisco aptikimo protokolas;

ICMP (angl. *internet control message protocol*) – interneto kontrolės žinučių protokolas;

IP (angl. *internet protocol*) – interneto protokolas;

IS-IS (angl. *intermediate system to intermediate system*) – viena tarpinė sistema kitai tarpinei sistemai;

JSON (angl. *JavaScript object notation*) – JavaScript objekto notacija;

LAN (angl. *local area network*) – vietinis tinklas;

LLDP (angl. *link layer discovery protocol*) – jungties lygmens aptikimo protokolas;

LLDPDU (angl. *LLDP data unit*) – LLDP duomenų vienetas;

MAC (angl. *media access control*) – medijos pasiekiamumo kontrolė;

MIB (angl. *management information base*) – valdymo informacijos duomenų bazė;

MPLS (angl. *multiprotocol label switching*) – daugiaprotokolis žymių komutavimas;

Netconf (angl. *network configuration protocol*) – tinklo konfigūravimo protokolas;

OID (angl. *object identifier*) – objekto identifikatorius;

OpenFlow – protokolas aprašantis SDN kontrolerio bendravimą su įrenginiu;

OSPF (angl. *open shortest path first*) – pirmiausia atidaryk trumpiausia kelia;

SCTP (angl. *stream control transmission protocol*) – srauto kontrolės siuntimo protokolas;

SDN (angl. *software defined networks*) – programine įranga apibrėžtas tinklas;

SNMP (angl. *simple network management protocol*) – paprastas tinklo valdymo prtokolas;

SSH (angl. *secure shell*) – saugi aplinka;

SSL (angl. *secure socket layer*) – saugiųjų jungimų lygmuo;

TCP (angl. *transmission control protocol*) – siuntimo valdymo protokolas;

TLV (angl. *type-length-value*) – tipas-ilgis-reikšmė;

TSL (angl. *transport layer security*) – transporto lygmens apsauga;

TTL (angl. *time-to-live*) – gyvavimo trukmė;

UDP (angl. *user datagram protocol*) - vartotojo datagarnu protokolas;

VLAN (angl. *virtual LAN*) – virtualus LAN;

XML (angl. *extensible markup language*) – plečiama žymėjimo kalba.

Įvadas

Kompiuterių tinklai šiais laikais yra neatsiejama verslo ir individualių vartotojų kasdienio gyvenimo dalis. Augant poreikiui - didėja ir reikalavimai keliami tinklui tokie kaip saugumas, našumas, dinamiškumas. Tradiciniai sprendimai tampa per sudėtingi, juos sunku valdyti, konfigūruoti ypač tuomet, kai konfigūravimas yra neautomatizuotas. Taip pat tradiciniam tinklui augant, pridedant vis naujų įrenginių, tinkle gali susidaryti kilpos, dėl to tinklas gali tapti nenuspėjamas ir sunku žinoti, kur koks paketų srautas juda. Sugedus vienam iš įrenginių gali būti sunku nustatyti gedimo vietą ir taip dalis vartotojų gali likti be prieigos prie tinklo. Su tokiomis bėdomis susiduria ir Vilniaus universiteto fizikos fakulteto tinklas.

Šioms ir panašioms problemoms spręsti vis dažniau naudojamas SDN (angl. *software defined networks* – programine įranga apibrėžtas tinklas). Šių tinklų išskirtinis bruožas yra tas, kad jų valdymas yra centralizuotas. Tai ir yra vienas iš pranašumų, lyginant SDN su tradiciniu tinklu. Kai valdymas yra centralizuotas - patys įrenginiai gali būti paprastesni ir pigesni. SDN tinklo vartotojams daug lengviau optimizuoti lėšų investavimą į kompiuterinius tinklus. Pagrindinis SDN privalumas – papildomas funkcionalumas tinkle. Dėl to, jog SDN kontroleris turi centralizuotą informaciją apie tinklą, tai gali priimti sprendimus ir pakeitimus remiantis viso tinklo apkrova. Pavyzdžiui, įdiegus SDN technologiją į Vilniaus universiteto fizikos fakulteto tinklą, galima būtų išspręsti anksčiau minėtas tinklo problemas.

Naujų technologijų įdiegimas į jau esamą infrastruktūrą dažniausiai vyksta palaipsniui. Seni įrenginiai vienas po kito yra keičiami naujais ir taip pereinama iš vienos technologijos naudojimo į kitą. Toks tinklas, kai jame yra tiek tradicinių įrenginių (angl. *legacy equipment*), tiek SDN įrenginių, vadinamas hibridiniu tinklu (angl. *hybrid network*). Perėjimas nuo vienų technologijų naudojimo prie kitų yra praktiškai neišvengiamas, vartotojams vienaip ar kitaip tenka susidurti su hibridiniais tinklais. Verta pabrėžti, jog hibridiniai tinklai yra ne tik žingsnis link naujų technologijų, tačiau vartotojams sukelia ir papildomus iššūkius. Hibridiniam tinklui valdyti reikia imtis specialių priemonių, padedančių SDN kontroleriui (angl. *controller*) perduoti informaciją apie esamus tradicinius įrenginius bei juos centralizuotai valdyti.

Baigiamojo darbo tikslas: sukurti sistemą, gebančią surinkti bendrą hibridinio tinklo topologiją bei automatizuotai konfigūruoti tradicinius įrenginius.

Pagrindinės darbo užduotys:

- išnagrinėti SNMP, LLDP, OpenFlow ir Netconf protokolų veikimą ir sintaksę;
- sukurti hibridinio tinklo topologijos surinkimo ir automatizuoto tradicinių įrenginių konfigūravimo algoritmą;
- parašyti algoritmą realizuojančią programą;

- ištestuoti programą realioje aplinkoje ir nustatyti algoritmo topologijos įvertinimo parametrus.

Tradicinių įrenginių konfigūravimo problemų apžvalga hibridiniame tinkle

Norint automatizuotai konfigūruoti tradicinius įrenginius hibridiniame tinkle susiduriama su įvairiomis problemomis. Viena jų - SDN ir tradicinio tinklo skirtumas, nes tradicinis tinklas yra decentralizuotas, o SDN - centralizuotas. Norint valdyti hibridinį tinklą reikia surinkti (sužinoti) tradicinio tinklo elementų informaciją bei susieti ją su SDN kontrolerio turima tinklo informacija.

Tradicinio tinklo informacijai surinkti nėra vieno universalaus būdo. Tam yra naudojami turimi tinklo informacijos surinkimo protokolai. Tai galima įgyvendinti naudojant maršrutizavimo protokolus: BGP (išorinės sąsajos maršrutizavimo protokolus), OSPF arba IS-IS (jungties būsenos maršrutizavimo protokolus), tačiau renkant topologiją maršrutizavimo protokolų pagalba surenkama tik trečio lygmens (L3) informacija. Šios informacijos pagalba nustatoma, kaip tarpusavyje sujungti trečio lygmens įrenginiai, bet negalima sužinoti, kokie antro lygmens (L2) įrenginiai yra tinkle. Kitas būdas yra naudoti jungties lygmens (angl. *link layer*) protokolus, tokius kaip LLDP (angl. *link layer discovery protocol* – jungties lygmens aptikimo protokolas) arba CDP (angl. *Cisco discovery protocol* - Cisco aptikimo protokolas). Šiame darbe buvo pasirinktas LLDP protokolas, kadangi jį palaiko didžioji dalis tinklo įrangos – tuo tarpu CDP protokolą palaiko tik Cisco tinklo įranga.

Tačiau vien LLDP neužtenka, kadangi jis veikia antrojo lygmens (L2) ribose, surenka informaciją tik apie savo kaimynus ir savo surinktą informaciją laiko lokaliai. Kiekvieno įrenginio apie kaimynus surinktos informacijos apjungimui į vieną duomenų bazę reikalingas pagalbinis instrumentas. Šiai užduočiai atlikti naudojamas SNMP (angl. *simple network management protocol* – paprastas tinklo valdymo protokolas) protokolas. SNMP pagalba galima surinkti įrenginių sukauptą informaciją į vieną centralizuotą serverį. Papildomai, SNMP galima centralizuoti ir trečio lygmens informaciją

Reikia pripažinti, jog nustatant hibridinio tinklo topologiją iškyla dar viena problema. SDN ir SNMP surinkta informacija gali skirtis. SDN surinkta topologija gali būti nepilna (tarp SDN įrenginių gali stovėti tradicinis įrenginys, tačiau SDN kontroleris matys tik šiuos du SDN įrenginius, kai tuo tarpu tradicinio įrenginio stovinčio tarp jų nematys). Taigi, norint automatizuotai konfigūruoti tradicinius įrenginius tinkle, reikia naujo instrumento, kuris surinktų SDN kontrolerio informaciją ir ją perteiktų tradiciniams įrenginiams.

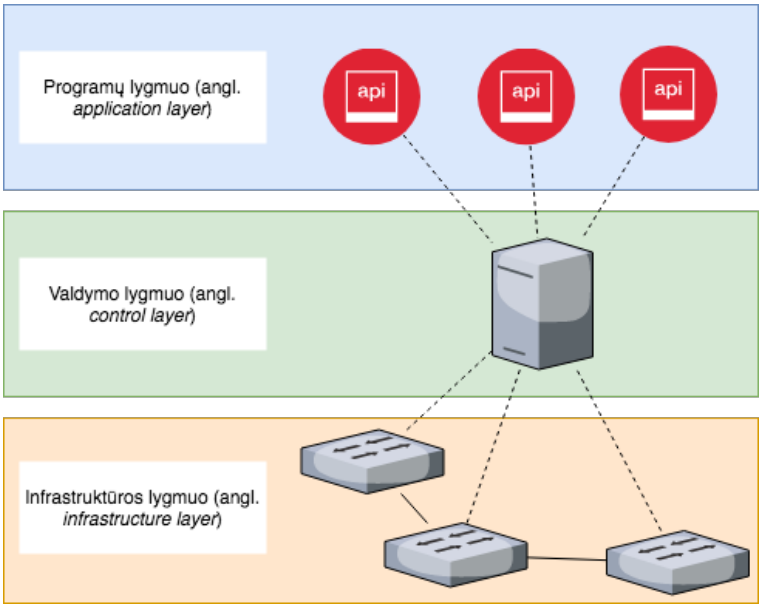
Siekiant išspręsti minėtas problemas žemiau pateikta topologijos nustatymo SDN ir tradiciniuose tinkluose, topologijos informacijos koncentravimo bei nuotolinio įrenginių konfigūravimo galimybių analizė.

Programine įranga apibrėžtas tinklas

SDN (angl. *software defined network* – programine įranga apibrėžtas tinklas) yra tinklas, kurio visų sprendimų priėmimas perkeliamas iš lokalaus įrenginio į centralizuotą kontrolerį. Tuo tarpu tradiciniuose tinkluose kiekvienas įrenginys sprendimus priima individualiai. Centralizacija SDN tinklui suteikia galimybę matyti visą tinklą ir pridėti papildoma funkcionalumą. Galima srautą valdyti ne tik pagal tikslo adresą, bet ir pagal šaltinio adresą arba pagal gautą prievadą ir kitus parametrus.

SDN tinklas yra skirstomas į tris lygmenis (1 pav.):

- programų lygmenį;
- valdymo lygmenį;
- infrastruktūros lygmenį.



1 pav. Pagrindiniai SDN lygmenys.

Aplikacijų lygmenyje galimos naujos programos, kurioms reikėtų naujų tinklo savybių arba reikalautų tam tikromis valandomis optimizuoti tam tikras programas. Valdymo lygmuo yra pagrindinis SDN lygmuo. Jame priimami visi sprendimai susiję su tinklu ir jis, reaguodamas į programų poreikius, pagal tai konfigūruoja įrenginius. Infrastruktūros lygmenyje sujungiami visi kontrolerio konfigūruojami fiziniai įrenginiai.

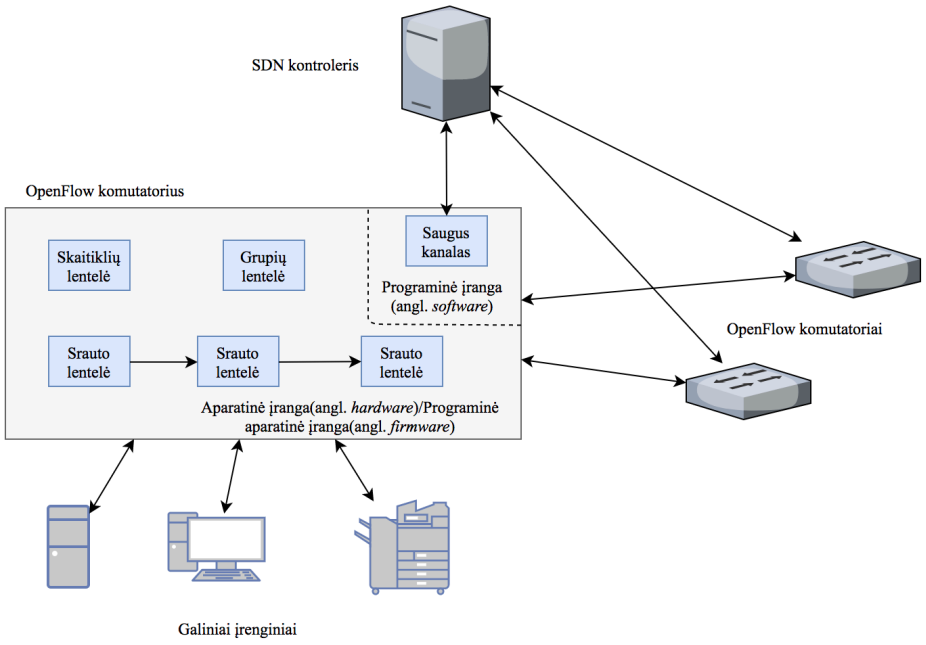
OpenFlow

Viena populiariausių SDN technologijų yra OpenFlow (OF). Vienas iš pagrindinių jos protokolų aprašo, kaip komutatorius bendrauja su kontroleriu. OpenFlow komutatoriai yra skirstomi į tris grupes:

- tradiciniai komutatoriai, kurie palaiko OpenFlow protokolą. Šie įrenginiai gali būti konfigūruojami tiek kaip tradiciniai įrenginiai, tiek kaip OpenFlow komutatoriai;
- OpenFlow komutatoriai. Šie įrenginiai gali būti konfigūruojami tik kaip OpenFlow komutatoriai;
- virtualūs įrenginiai. Šie gali būti konfigūruojami kaip tradicinis įrenginys, kuris palaiko OpenFlow arba tik kaip OpenFlow komutatorius.

Loginė OpenFlow komutatoriaus architektūra

SDN kontroleris bendrauja su OpenFlow komutatoriumi per saugų kanalą (juo perduodama valdymo informacija), naudojant SSL protokolą (angl. *secure socket layer protocol* - saugiųjų jungimų lygmens protokolas) [1]. Be to, siekiant užtikrinti klientų informacijos srautą, kiekvienas komutatorius sujungiamas su kitais OpenFlow komutatoriais arba su galiniais įrenginiais. Kiekvienas OpenFlow komutatorius turi dinaminių lentelių rinkinį. Jos yra atsakingos už paketų valdymą įrenginyje. Šios lentelės dažniausiai yra įdiegtos į komutatoriaus aparatinę arba programinę aparatinę įrangą. OpenFlow paketų valdymo struktūra pavaizduota 2 pav.



2 pav. OpenFlow paketų valdymo struktūra.

Į OpenFlow komutatorių yra įrašomos trys lentelių rūšys. Srauto lentelėje saugomi įrašai, kuriose yra instrukcijos ką daryti su įeinančiais paketais. Gali būti daug srauto lentelių, kurios

tuomet yra apdorojamos komandų grandinės (angl. *pipeline*) būdu. Srauto lentelė gali nurodyti grupių lentelę, kuri gali atlikti tam tikras funkcijas paketams, priklausantiems vienam ar keliems srautams. Skaitiklių lentelė gali aktyvuoti įvairius su efektyvumu susijusius procesus.

Srauto lentelės komponentai

Pagrindinis loginio OF komutatoriaus komponentas yra srauto lentelė. Kiekvienas paketas patekęs į komutatorių patikrinamas vienoje ar daugiau srauto lentelių. Kiekviena srauto lentelė yra sudaryta iš šešių komponentų [2]:

- tikrinimo laukai (angl. *match fields*) – naudojami siekiant pasirinkti paketus, kurie atitinka laukuose esančias reikšmes;
- prioritetas (angl. *priority*) – santykinis prioritetas lentelių įrašams;
- skaitliukai (angl. *counters*) – naudojami sekti veiklai. Pavyzdžiui, gaunamų baitų bei paketų kiekis vienam prievadui, srauto lentelei, bei srauto lentelės įrašui. Įtraukiama yra ir atmestų paketų skaičius, srautų trukmė;
- instrukcijos – jos apibrėžia, kokie veiksmai turi būti atlikti tuo atveju, jei randamas paketą atitinkantis įrašas;
- laiko limitai (angl. *timeouts*) – maksimalus laiko tarpas, po kurio srautas bus ištrintas;
- slapukai (angl. *cookie*) – kontrolerio pasirinktos neaiškios duomenų vertės. Gali būti naudojami kontrolerio, siekiant išfiltruoti srautų statistiką, srautų modifikaciją bei srautų panaikinimą. Kita vertus, slapukai nėra naudojami apdorojant srautus.

Į srautų lentelę galima įtraukti ir nepataikytos lentelės (angl. *table-miss*) įrašą, kuris galioja visiems paketams ir turi žemiausio lygio - nulinį prioritetą. Tikrinimo laukai yra sudaryti iš šių privalomų elementų:

- įėjimo prievadas – prievado identifikatorius, kuris nurodo, per kurį prievadą buvo gautas paketas. Prievadas gali būti tiek fizinis, tiek virtualus;
- Eterneto šaltinio ir paskirties adresai (angl. *Ethernet source and destination addresses*) – kiekvienas įrašas gali būti konkretus adresas, potinklis (angl. *subnet*), arba pakaitos simbolis (angl. *wildcard*) (atitinkantis visas reikšmes);
- IPv4 arba IPv6 protokolų identifikatorius – nurodoma sekančios antraštės reikšmė;
- IPv4 arba IPv6 šaltinio adresas bei paskirties adresas – kiekvienas įrašas gali būti konkretus adresas, potinklis, arba pakaitos simbolis (atitinkantis visas reikšmes);
- TCP šaltinio ir paskirties prievadai – konkretus atitikmuo arba pakaitos simbolis;
- UDP šaltinio ir paskirties prievadai – konkretus atitikmuo arba pakaitos simbolis.

OpenFlow komutatorius privalo palaikyti aukščiau paminėtus tikrinimo laukus. Žemiau pateikiami galimi laukai, kurie OpenFlow komutatoriuose nėra privalomi:

- Fizinis prievadas – naudojamas fizinio prievado identifikavimui;
- Metaduomenys (angl. *metadata*) – papildoma informacija, kuri paketo apdorojimo metu gali būti perduota iš vienos lentelės į kitą;
- Eterneto tipas – eterneto tipo laukas;
- VLAN ID ir VLAN vartotojo prioritetas (angl. *user priority*) – IEEE 802.1Q virtualių LAN antraščių laukai;
- IPv4 arba IPv6 DS ir ECN – diferencijuotos paslaugos (angl. *differentiated services*) bei neabejotinos perpildos pranešimo (angl. *explicit congestion notification*) laukai;
- SCTP (angl. *stream control transmission protocol* – srauto kontrolės siuntimo protokolas) šaltiniai ir paskirties prievadai – konkretus atitikmuo arba pakaitos simbolis;
- ICMP (angl. *internet control message protocol* – interneto kontrolės žinučių protokolas) tipas ir kodo laukai – konkretus atitikmuo arba pakaitos simbolis;
- ARP (angl. *address resolution protocol* – adreso rezoliucijos protokolas) operacijos kodas – konkretus atitikmuo eterneto tipo lauke;
- Šaltinio ir paskirties IPv4 adresai ARP turinyje (angl. *payload*) – gali būti konkretus adresas, reikšmė, potinklis arba pakaitos simbolis, atitinkantis visas reikšmes;
- IPv6 srauto žymė – konkretus atitikmuo arba pakaitos simbolis;
- ICMPv6 tipas ir kodo laukai – konkretus atitikmuo arba pakaitos simbolis;
- IPv6 kaimyno aptikimo paskirties adresas (angl. *neighbor discovery target address*) – IPv6 kaimyno aptikimo žinutė (angl. *neighbor discovey message*);
- IPv6 kaimyno aptikimo šaltinio ir paskirties adresas (angl. *neighbor dicovery source and target address*) – jungties lygmens pasirinkimai IPv6 kaimyno aptikimo žinutėje;
- MPLS (angl. *multiprotocol label switching* – daugiaprotokolis žymių komutavimas) žymės reikšmė, srauto klasė bei BoS (angl. *bottom of stack* – dėklo apačia) – laukai MPLS dėkle.

OpenFlow protokolas palaiko šias pagrindines instrukcijas – veiksmo (angl. *action*) ir veiksmo nustatymo (angl. *action set*). Veiksmo instrukcijos susideda iš:

- išvestis (angl. *output*) – išsiųsti paketą per nurodytą prievadą;
- eilės nustatymas (angl. *set-queue*) – nustato eilės identifikatorių paketui. Naudojamas, kai paketas išsiunčiamas naudojant išvesties veiksmą, kad pavyktų

nustatyti, kurį paketą pirmiau išsiųsti. Suteikia sistemai QoS (angl. quality of service – paslaugos kokybė) palaikymą.

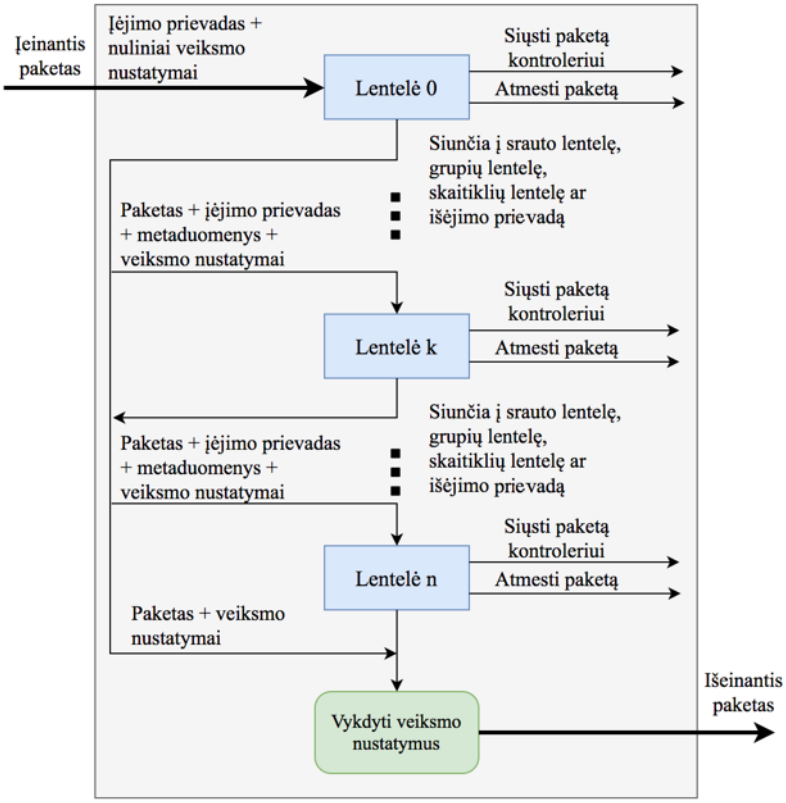
- grupė (angl. *group*) – paketo apdorojimas nurodytai grupei;
- stūmimo žymė (angl. *push-tag*) / sprogimo žymė (angl. *pop-tag*) – laukai VLAN arba MPLS paketams;
- nustatymo laukas (angl. *set-field*) – nustato reikšmes, esančias paketo antraštės (angl. *header*) laukuose;
- TTL pakeitimas (angl. *set-TTL*) – pakeičia IPv4 TTL, IPv6 žingsnių ribos (angl. *hop limit*) ar MPLS TTL reikšmes paketuose.

Veiksmo nustatymo instrukcijos yra naudojamos, kai paketas yra apdorojamas komandų grandinėje. Veiksmo nustatymai gali būti tokie:

- nuorodos paketui komandų grandinėje (angl. *direct packet through pipeline*) – nurodo kuri lentelė bus sekanti komandų grandinėje;
- veiksmo atlikimas paketui (angl. *perform action on packet*) – veiksmai, kuriuos reikia atlikti, kai randamas paketą atitinkantis įrašas lentelėje;
- veiksmo nustatymo atnaujinimas (angl. *update action set*) – naudojamas sujungiant nurodytus veiksmus į veiksmų nustatymus arba ištrinant visus veiksmus iš veiksmų nustatymo.
- metaduomenų atnaujinimas (angl. *update metadata*) – atnaujina metaduomenis, kurie yra susieti su paketu. Metaduomenys yra naudojami perduodant informaciją iš vienos lentelės į kitą.

Srauto lentelių komandų grandinė

Komutatorius gali turėti vieną arba daugiau srauto lentelių. Jeigu komutatorius turi daugiau nei vieną srauto lentelę, tai jos yra apdorojamos komandų grandinių būdu ir sunumeruotos nuo 0 didėjimo tvarka (3 pav.).



3 pav. Paketų apdorojimas komandų grandinių būdu.

Kai paketas pristatomas lentelei tikrinimui, pateikiamas paketas, įėjimo prievadas, metaduomenys ir veiksmo nustatymai. Nulinei lentelei metaduomenys ir veiksmo nustatymai pateikiami kaip tušti (angl. *null*). Kiekvienoje srauto lentelėje atliekama tokia veiksmų seka:

1. Ieškomas aukščiausią prioritetą turintis įrašas, kuris atitinka paketą. Jei neranda atitikmens ir nėra nepataikytos lentelės įrašo, tai paketas atmetamas. Jei paketas atitinka tik nepataikytos lentelės įrašą, vykdomas vienas iš šių veiksmų, kuris nurodytas įraše:
    a. paketas siunčiamas kontroleriui. Kontroleris nusprendžia ar sukurti naują įrašą, kuris bus pritaikytas šiam bei panašiems paketams, ar paketą atmesti;
    b. paketas siunčiamas sekančiai komandų grandinės lentelei;
    c. paketas atmetamas.
2. Jeigu paketas atitinka vieną arba daugiau lentelės įrašų, kurie nėra nepataikytos lentelės įrašai, tai pritaikomas tas įrašas, kurio prioritetas aukščiausias. Įrašas gali nurodyti šiuos veiksmus:
    a. atnaujinti skaitiklius, susietus su šiuo įrašu;
    b. įvykdyti instrukcijas, susietas su įrašu. Šios instrukcijos gali susidaryti iš veiksmo nustatymo reikšmės atnaujinimo, metaduomenų atnaujinimo ir veiksmų atlikimo;

c. Tada paketas gali būti nusiunčiamas toliau komandų grandine, į grupių lentelę, į skaitiklių lentelę ar į dedikuotą išėjimo prievadą.

Paskutinėje komandų grandinės lentelėje nėra galimas siuntimas į sekančią lentelę. Kai paketas yra nusiunčiamas į išėjimo prievadą, yra įvykdomi veiksmo nustatymai ir tuomet paketas pastatomas į eilę išsiuntimui.

OpenFlow protokolas

OpenFlow protokolas nustato bendravimą tarp OpenFlow komutatoriaus ir kontrolerio. Dažniausiai protokolas yra realizuojamas kartu su SSL arba TSL (angl. *transport layer security* – transporto lygmens apsauga), siekiant užtikrinti protokolo saugumą. OpenFlow protokolo pagalba kontroleris sukuria (angl. *add*), atnaujina (angl. *update*) ir ištrina (angl. *delete*) srautų įrašus. OpenFlow palaiko šias žinučių grupes (2 lentelė):

- Iš kontrolerio į komutatorių (angl. *controller-to-switch*) – šios žinutės yra inicializuojamos kontrolerio ir kartais reikalauja komutatoriaus atsako. Ši žinučių grupė įgalina valdyti komutatorių konfigūracijas, gauti informaciją apie srauto ir grupių lentelių įrašus. Šiai žinučių grupei priklauso ir išeinančio paketo (angl. *packet-out*) žinutė. Ši žinutė yra naudojama tada, kai komutatorius užklausia kontrolerio ką daryti su paketu, o šis nurodo, per kurį prievadą paketą išsiųsti;
- Asinchroninės (angl. *asynchronous*) – šios žinutės yra siunčiamos be kontrolerio prašymo. Ši žinučių grupė yra sudaryta iš įvairių statuso žinučių kontroleriui, tarp kurių yra ir įeinančio paketo (angl. *packet-in*) žinutė. Ši žinutė yra siunčiama komutatoriaus, siekiant išsiųsti paketą kontroleriui, kai paketas neatitinka srauto lentelės įrašų;
- Simetrinės (angl. *symmetric*) – šios žinutės yra siunčiamos be kontrolerio prašymo. Ši žinučių grupė atlieka paprastą tikrinimo paskirtį. Pasveikinimo (angl. *hello*) žinutės yra siuntinėjamos pirmyn atgal, kai sukuriamas ryšys tarp komutatoriaus ir kontrolerio. Aido užklausos (angl. *echo request*) ir aido atsakymo (angl. *echo reply*) žinutės gali būti naudojamos tiek kontrolerio tiek komutatoriaus norint išmatuoti delsą (angl. *latency*) ar pralaidumą (angl. *bandwidth*) tarp kontrolerio ir komutatoriaus. Šios žinutės gali būti naudojamos siekiant patikrinti ar įrenginys vis dar funkcionuoja.

1 lentelė. OpenFlow žinutės.

Žinutė	Aprašymas
Iš kontrolerio į komutatorių	
Funkcijos (angl. *features*)	Prašymas nurodyti komutatoriaus funkcionalumą.

	Komutatorius atsako su funkcijų atsakymo (angl. *feature reply*) žinute, kurioje nurodo savo galimybes.
Konfigūracijos (angl. *configuration*)	Nustato ir prašo konfigūracijos parametrus. Komutatorius atsako su parametrų nustatymais.
Būsenos pakeitimo (angl. *modify-state*)	Prideda, ištrina, pakeičia srauto/grupių lentelių įrašus ir nustato komutatoriaus prievadų nustatymus.
Būsenos nuskaitymo (angl. *read-state*)	Surenka iš komutatoriaus esamą konfigūraciją, statistiką ir galimybes.
Išeinančio paketo	Nurodo išėjimo prievadą paketui.
Barjero (angl. *barrier*)	Naudojamos kontrolerio norint gauti pranešimus apie įvykdytas operacijas.
Rolių prašymo (angl. *role-request*)	Nustato arba užklausia OpenFlow kanalo rolę. Naudojamos, kai komutatorius yra sujungtas su keliais kontroleriais.
Asinchroninės konfigūracijos (angl. *asynchronous-configuration*)	Nustato arba užklausia asinchroninių žinučių filtrą. Naudojamos, kai komutatorius yra sujungtas su keliais kontroleriais.
Asinchroninės	
Įeinančio paketo	Perduoda paketą kontroleriui.
Srauto ištrynimo (angl. *flow-removed*)	Informuoja kontrolerį apie įrašo ištrynimą iš srauto lentelės.
Prievado būsenos (angl. *port-status*)	Informuoja kontrolerį apie prievado būsenos pokytį.
Klaidos (angl. *error*)	Informuoja kontrolerį apie klaidą.
Simetrinės	
Pasisveikinimo	Ryšio pradžioje pasikeičiama žinutėmis tarp komutatoriaus ir kontrolerio.
Aido	Aido užklausos/atsakymo žinutes gali išsiųsti tiek kontroleris, tiek komutatorius. Į užklausos žinutę yra atsakoma atsakymo žinute.
Eksperimentinės (angl. *experimenter*)	Papildomoms funkcijoms.

OpenFlow protokolo pagalba kontroleris gali valdyti loginę komutatoriaus struktūrą, neatsižvelgiant, kaip komutatorius realizuoja OpenFlow loginę architektūrą.

Topologijos nustatymas tradiciniuose tinkluose

Tradiciniuose tinkluose nėra vieno tinklo topologijos nustatymo būdo. Yra pasiūlyta daug įvairių būdų tam pasiekti, pavyzdžiui, BGP, OSPF ar SNMP. Maršrutizavimo protokolų pagalba galima surinkti trečio lygmens (L3), tačiau naudojant SNMP ir LLDP ar CDP galima ne tik surinkti visą tinklo topologiją, bet ir tradicinius įrenginius konfigūruoti centralizuotai. Šiame darbe buvo pasirinktas LLDP protokolas, kadangi jis yra standartizuotas ir jį taikant galima pasiekti ne tik Cisco įrenginius kaip CDP protokolo pagalba.

LLDP

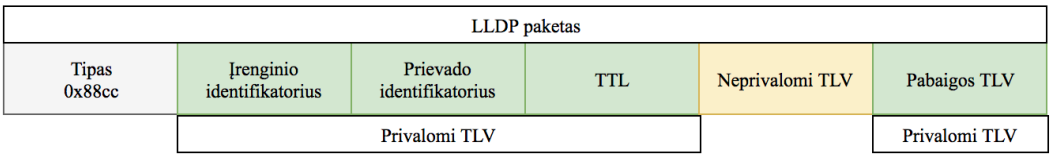
LLDP yra nepriklausomas nuo gamintojų protokolas, kuris yra skirtas tinklo įrangai dalintis informacija su kaimynais apie savo tapatybę (angl. *identity*), galimybėmis (angl. *capabilities*) LAN (angl. *local area network* – vietinis tinklas) segmente. LLDP yra apibrėžtas IEEE 802.1AB. LLDP periodiškai (standartiškai kas 30 sekundžių) retransliuoja savo paketus per kiekvieną prievadą. Šis protokolas yra vienakryptis, tai yra jis nelaukia patvirtinimo, jog paketas yra gautas.

LLDP paketo struktūra

LLDP paketai yra siunčiami fiksuotu intervalu per kiekvieną prievadą ir turi eterneto formą. Kiekvienas LLDP paketas yra sudarytas iš vieno LLDPDU (angl. *LLDP data unit* – LLDP duomenų vienetas). Seka TLV (angl. *type-length-value* – tipas-ilgis-reikšmė) sudaro LLDPDU. LLDPDU paketo struktūra (4 pav.) yra tokia:

- Paketo tipas yra 0x88cc;
- Kiekvienas LLDP paketas prasideda su privalomais TLV:
    - Įrenginio (angl. *chassis*) identifikatorius;
    - Prievado identifikatorius;
    - TTL (angl. *time-to-live* – gyvavimo trukmė) trukmė, kuri nurodo kiek laiko informaciją, gautą iš LLDP paketo, laikyti galiojančia;
- Po privalomų TVL gali būti neprivalomi TLV;
- Paketo pabaigoje yra specialus TLV, kuris žymį paketo pabaigą. Jo tipo ir ilgio reikšmės lygios nuliui.

LLDP sukurtas taip, kad organizacijos galėtų kurti savo TLV struktūras, kurių tipo reikšmė lygi 127 [3]. TLV tipų reikšmės gali kisti nuo 0 iki 127 (2 lentelė). Visa surinkta informacija yra saugoma MIB duomenų bazėje.



4 pav. LLDP paketo struktūra.

2 lentelė. TLV tipai.

TLV tipai		
TLV tipo reikšmė	TLV pavadinimas	Panaudojimas
0	Pabaigos	Privalomas
1	Įrenginio identifikatorius	Privalomas
2	Prievado identifikatorius	Privalomas
3	TTL	Privalomas
4	Prievado apibrėžimas(angl. *port description*)	Neprivalomas
5	Sistemos vardas	Neprivalomas
6	Sistemos apibrėžimas	Neprivalomas
7	Sistemos galimybės	Neprivalomas
8	Valdymo adresas	Neprivalomas
9-126	Rezervuoti	Neapibrėžtas
127	Organizacijų	Neprivalomas

LLDPDU tipai

LLDPDU paketai yra dviejų rūšių:

- Įprasti LLDPDU, kurie perduoda informaciją kaimynams apie save. Šie paketai perduoda informaciją tiek apie privalomus, tiek apie neprivalomus TLV.
- Išjungimo LLDPDU, kurie perduoda informaciją kaimynui, jog visa kaimyno turima informacija apie įrenginį turėtų būti pašalinta. Šis specialus LLDPDU išsiunčiamas tada, kai įrenginys žino, jog prievadas tuoj pakeis būseną. Tai gali nutikti, kai prievadas pereina į išjungto prievado būseną arba kai yra išjungiamas LLDP paketų siuntimas įrenginyje. Šio įvykio metu įrenginys bando išsiųsti paskutinį LLDP paketą, kurio visos privalomų TLV tipų reikšmės lygios nuliui. Išjungimo LLDPDU neturi neprivalomų TLV ir yra bandomas išsiųsti iki momento, kai prievadas tampa neveiksnus.

LLDP veiksenos (angl. *operating modes*)

LLDP palaiko tris pagrindines veiksenas:

- Išsiuntimo (angl. *transmit only*) – siunčia LLDP paketus, tačiau gautus LLDP paketus atmeta ir neapdoroja;
- Gavimo (angl. *receive only*) – gautus LLDP paketus apdoroja, tačiau nesiunčia LLDP paketų apie save;
- Išsiuntimo ir gavimo (angl. *both transmit and receive*) – tiek siunčia tiek apdoroja LLDP paketus.

LLDP skaitliukai

- Išlaikymo trukmė (angl. *hold time*) – trukmė, per kurią negavus iš kaimyno LLDP paketų laikyti, jog ryšys su kaimynu nutrūko (standartinė trukmė 120 sekundžių);
- LLDP paketų dažnis (angl. *LLDP packet frequency timer*) – nurodo kokiu dažniu siųsti LLDP paketus (standartinė trukmė 30 sekundžių);
- Inicializavimo laikas (angl. *reinit time*) – trukmė, per kurią LLDP inicializuojasi (standartinė trukmė 2 sekundės).

SNMP

SNMP protokolas buvo sukurtas tam, kad būtų galima centralizuotai surinkti informaciją apie įrenginius. Dėl to, jog yra daug skirtingų gamintojų, SNMP protokolas standartizuoja informaciją. SNMP turi du pagrindinius komponentus: SNMP agentus (angl. *SNMP agents*) ir SNMP serverius (angl. *SNMP manager*) [4]. SNMP agentas yra programa, kuri surenka informaciją apie įrangą iš MIB duomenų bazės bei atsako į SNMP užklausas. SNMP serveris siunčia SNMP užklausas ir surenka informaciją apie įrenginius.

MIB duomenų bazės struktūra

MIB duomenų bazė yra globaliai standartizuota hierarchinė struktūra, kuri yra lanksti ir palaiko nestandartinius gamintojų įskiepius. MIB struktūrą galima suprasti kaip iš viršaus į apačią besišakojantį medį, kurio kiekvienas išsišakojimas turi tiek skaitinę, tiek žodinę vertę. Norint pasiekti tam tikrą vertę reikia pereiti per visus medžio atsišakojimus. Kiekvienas atsišakojimas turi savo vertę, kiekviena vertė yra atskiriama tašku. Taip sudaromas identifikatorius, kuris vadinamas OID (angl. *object identifier* – objekto identifikatorius). Gamintojai įrenginiuose dažnai prideda savo MIB atšakų su unikaliais duomenimis apie įrangą, tačiau yra standartinės MIB šakos, kurios yra atviros ir visi gali jas naudoti.

Standartinė MIB šaka, kuri ir yra dažniausiai naudojama - 1.3.6.1.2.1 (arba iso.org.dod.internet.mgmt.mib-2). Tokiu hierarchiniu pavidalu yra sustruktūrizuota visa MIB duomenų bazė.

SNMP protokolo žinučių tipai

SNMP protokolas turi nedaug žinučių tipų, tačiau tos žinutės yra lanksčios ir to pakanka, kad galima būtų tiek surinkti informaciją apie įrenginį, tiek jį konfigūruoti. SNMP sudaro šie septyni žinučių tipai:

- Gavimo (angl. *get*) – šią žinutę siunčia serveris, siekdamas gauti tam tikrus duomenis, kuriuos atitinka nurodytas OID. Agentas atsako su atsakymo (angl. *response*) žinute;
- Kito gavimo (angl. *getnext*) – šią žinutę siunčia serveris, siekdamas gauti sekančią MIB reikšmę. Taip galima gauti visus įrenginio duomenis, nežinant OID reikšmių;
- Nustatymo (angl. *set*) – šią žinutę siunčia serveris, tam, kad įrenginyje pavyktų pakeisti objekto, nusakyto OID reikšmę. Šios žinutės pagalba galima konfigūruoti įrenginius nuotoliniu būdu;
- Gauti masiškai (angl. *getbulk*) – šią žinutę siunčia serveris, imituodamas daug kito gavimo žinučių vienoje žinutėje. Taip galima gauti daugiau duomenų viena žinute.
- Atsakymo – šią žinutę siunčia agentas ir naudoja siekdamas perduoti informaciją serveriui, patvirtinti nustatymo žinutę, ar, jei įvyksta klaida, pateikti klaidos pranešimus;
- Pertraukčių (angl. *trap*) – šią žinutę siunčia agentas; jos tikslas yra perspėti serverį apie pasikeitusią įrenginio būseną;
- Informavimo (angl. *inform*) – serveris siunčia šią žinutę, tam kad informuotų agentą, jog gavo pertraukties žinutę. Jei agentas negauna informavimo žinutės, pertrauktis siunčiama pakartotinai.

SNMP versijos

SNMP protokolas pirmą kartą buvo aprašytas 1988 metais RFC 1065, 1066 ir 1067, tačiau ši pirmoji versija dar dažnai palaikoma įrenginių. Ši versija turi didelių saugumo spragų, tokių kaip autentifikavimas atviru tekstu (angl. *plain text*). Antroji SNMP versija pasirodė 1993 metais. Joje protokolas yra atnaujintas, tačiau ji nebuvo naudojama, nes buvo sudėtinga ją įdiegti. Bėgant laikui atsirado šios versijos patobulinimai. SNMPv2c pristatyta bendruomene pagrįstas (angl. *community-based*) autentifikavimas. 1998 metais buvo išleista trečioji (naujausia) protokolo versija. Šioje versijoje išpildyta vartotojų autentifikavimu pagrįsta (angl. *user-based*) apsauga. Trečioji versija autentifikavimą skirsto į tris modelius:

- NoAuthNoPriv – šis modelis nesuteikia jokio autentifikavimo ir duomenys nešifruojami;
- AuthNoPriv – šiame modelyje vykdomas autentifikavimas, tačiau nėra šifravimo;

- AuthPriv – šiame modelyje vykdomas tiek autentifikavimas, tiek šifravimas.

Tradicinių įrenginių nuotolinis konfigūravimas

Netconf

Netconf (angl. *Network configuration protocol* – tinklo konfigūravimo protokolas) yra protokolas, naudojamas tinklo įrenginiams konfigūruoti nuotoliniu būdu. Netconf yra sudarytas iš dviejų dalių: kliento (angl. *client*) ir serverio (angl. *server*). Serveris siunčia komandas klientams ir juos konfigūruoja, tuo tarpu klientai atsako ar operacija įvykdyta sėkmingai. Sesijos užmezgimo metu klientai tai pat suteikia informaciją ir apie save (pvz. Netconf versija, kokias papildomas operacijas palaiko ir t.t.).

Netconf lygmenys

Netconf yra suskirstytas į keturis lygmenis [5]:

- Turinio (angl. *content*);
- Operacijų (angl. *operations*);
- Žinučių (angl. *messages*);
- Saugaus transporto (angl. *secure transport*).

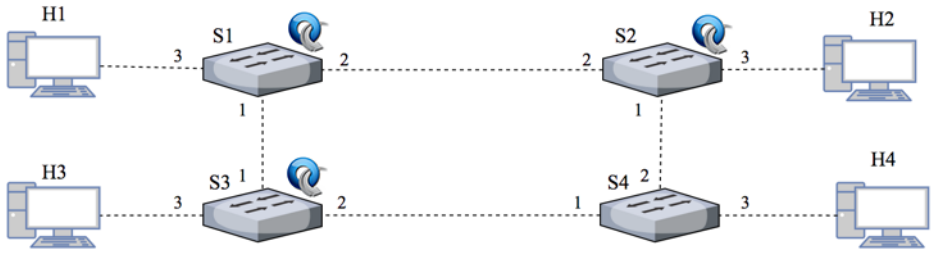
Turinio lygmenyje yra saugomas XML failas, kuriame nurodoma kokia komanda bus persiunčiama į įrenginį. Operacijų lygmenyje yra nurodoma, ką įrenginiui reikės daryti. Operacijų lygmenyje yra šios pagrindinės operacijos:

- Gavimo (angl. *get*) – pasiima įrenginio vykdomąją konfigūraciją ir įrenginio būsenos informaciją;
- Konfigūracijos gavimo (angl. *get-config*) – iš nurodytos duomenų bazės pasiima visą konfigūraciją arba nurodytą jos dalį;
- Konfigūracijos pakeitimo (angl. *edit-config*) – pakeičia konfigūraciją sukuriant, ištrinant, sujungiant ar pakeičiant turinį;
- Konfigūracijos kopijavimo (angl. *copy-config*) – nukopijuoja visą konfigūracijos duomenų bazę į naują duomenų bazę;
- Konfigūracijos ištrynimo (angl. *delete-config*) – ištrina konfigūracijos duomenų bazę;
- Užrakinimo (angl. *lock*) – užrakina konfigūracijos duomenų bazę;
- Atrakinimo (angl. *unlock*) – atrakina konfigūracijos duomenų bazę;
- Sesijos uždarymo (angl. *close-session*) – prašymas užbaigti Netconf sesiją;
- Sesijos nutraukimo (angl. *kill-session*) – priverstinis Netconf sesijos užbaigimas.

Šių funkcijų gali būti ir daugiau, tačiau aukščiau išvardintos yra privalomos. Žinučių lygmenyje klientai bendrauja su serveriu, kad praneštų apie sėkmingai įvykdytas operacijas ir suteiktų kitą reikiamą informaciją. Saugaus transporto lygmenyje nusakoma, kokiu būdu bus bendraujama tarp kliento ir serverio (SSH, TSL ir t.t.).

Topologijos nustatymas OpenFlow tinkluose

OpenFlow tinkle topologija yra nustatoma kontrolerio pagalba. Iš pradžių komutatoriai prisijungia prie kontrolerio ir taip kontroleris sužino, kokie OpenFlow komutatoriai yra tinkle. Tada, siekiant nustatyti kokie yra ryšiai tarp šių įrenginių, yra naudojami LLDP bei BDDP (angl. *broadcast domain discovery protocol* – transliacijos domeno aptikimo protokolas). Kontroleris periodiškai liepia kiekvienam komutatoriui išsiųsti LLDP paketus per visus prievadus. Jei du OpenFlow komutatoriai yra sujungti tiesiogiai - jie gauna vienas kito LLDP paketus ir praneša kontroleriui, jog gavo LLDP paketą. Taip kontroleris sužino apie tiesiogiai sujungtus OpenFlow kontrolerius. Tam, kad būtų galima rasti netiesiogiai sujungtus OpenFlow komutatorius, naudojamas BDDP. Tradicinis komutatorius nepersiunčia LLDP paketų, todėl tam yra naudojamas BDDP.



5 pav. OpenFlow topologija.

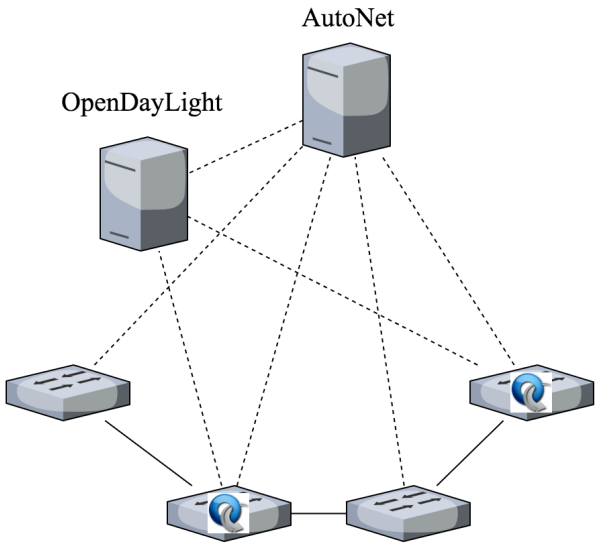
Tarkime turime paprastą topologiją (5 pav.), kurioje S1, S2 ir S3 yra OpenFlow komutatoriai, o S4 yra tradicinis komutatorius. Taigi, po to kai OpenFlow komutatoriai prisijungia prie kontrolerio, šis žino, kad yra 3 OpenFlow komutatoriai. Kai komutatoriai išsiunčia LLDP paketus yra sužinoma, kad S1 sujungtas per pirmą prievadą su S3 komutatoriaus pirmu prievadu ir kad S1 antras prievadas sujungtas su S2 antru prievadu. Tada, tarkime, S2 išsiunčia BDDP paketą ir jį gauna S4 komutatorius. Dėl to, jog jis nėra OpenFlow, kontroleris apie tai nieko nesužino, tačiau S4 komutatorius persiunčia BDDP paketą per visus savo prievadus ir S3 gauna S2 siųstą BDDP per antrą prievadą ir apie tai praneša kontroleriui. Taip kontroleris sužino, kad tarp S2 ir S3 yra tradicinių įrenginių, t. y. S2 ir S3 yra sujungti netiesiogiai. Galinių įrenginių vietos topologijoje gavimas paremtas paketų srautais. Tarkime H1 išsiunčia paketą H3. Kai S1 gauna paketą, jis persiunčia jį kontroleriui, kadangi nežino kur jį siųsti. Taip kontroleris sužino, kad H1 yra S1 trečiame prievade. Tada kontroleris liepia S1 išsiųsti tą paketą per visus savo prievadus. Tą patį daro ir S3, kuris gauna atsaką iš H3 per savo trečią prievadą ir apie tai praneša kontroleriui. Taip kontroleris sužino H3 buvimo vietą bei suformuoja srauto įrašą. Tokiu mechanizmu atrandami visi galiniai įrenginiai ir galimi keliai, kaip juos pasiekti.

OpenDayLight kontroleris

Šiame darbe buvo pasirinktas OpenDayLight kontroleris [6], nes jis yra populiariausias nemokamas SDN kontroleris. Šis kontroleris yra vienas labiausiai paplitusių kontrolerių, kadangi nors ir atviro kodo, tačiau yra palaikomas daugelio gamintojų (tokių kaip Cisco, Redhat, Huawei, Juniper ir kt.). OpenDayLight kontroleris palaiko OpenFlow protokolą, taip pat turi API funkcionalumą, dėl kurio galima lengvai rašyti papildomas programas, kurios bendrautų su kontroleriu. Pats kontroleris yra parašytas Java programavimo kalba.

Tradicinių įrenginių automatizuoto konfigūravimo realizacija

Pagrindinis šio darbo tikslas buvo suprojektuoti sistemą ir algoritmus bei parašyti programą, kuri automatizuotai konfigūruotų tradicinius įrenginius įdiegtus hibridiniame tinkle bei suteiktų informaciją apie juos OF kontroleriui. Parašyta programa buvo pavadinta Autonet.



6 pav. Darbe naudota supaprastinta topologija.

Autonet programa automatizuotai konfigūruoja tradicinius įrenginius, esančius šalia OpenFlow komutatoriaus. Pirmiausia, programos pagalba surenkama informacija apie visus tinkle esančius įrenginius. Tradicinių įrenginių informacija gaunama SNMP protokolu, o apie įrenginio kaimynus sužinoma iš LLDP informacijos. Iš OpenDayLight kontrolerio yra sužinoma informacija tiek apie OpenFlow įrenginius, tiek apie jų turimus srauto įrašus. Surinkta informacija yra panaudojama tradicinių įrenginių MAC lentoms, kurios nurodo, kur siųsti paketus, konfigūruoti. Nauji MAC lentos įrašai į tradicinius įrenginius yra siunčiami Netconf protokolu. Šių įrašų pagalba yra nukreipiami srautai pagal OpenFlow komutatorių konfigūraciją. Programa naudoja vieną valdymo Vlan, darbinių Vlan skaičius nėra ribojamas. Supaprastinta darbe naudotos topologijos schema pateikta 6 paveiksle. Vadovaujantis tokia logika buvo parašyta programa, konfigūruojanti tradicinius įrenginius hibridiniame tinkle.

Autonet programa yra HNT [7] programos tęsinys. Autonet programoje buvo ištaisyti HNT programos minusai. HNT programoje nustatant įrenginio administravimo adresą buvo naudojamas LLDP pateikiamas adresas, tačiau šis adresas nebūtinai yra administravimo adresas kadangi LLDP pateikia mažiausią VLAN numerį atitinkantį adresą, o administravimo VLAN gali būti ir aukštą numerį turintis adresas. Autonet programoje, tai buvo ištaisyta iš pradžių kreipiantis į įrenginį

LLDP pateiktu adresu, o po to, pagal administravimo VLAN numerį buvo gaunamas administravimo adresas ir toliau visi veiksmai su įrenginiu atliekami naudojant administravimo adresą. Be to, HNT programa tinklo įrenginių nekonfigūruoja, o tik surenka informaciją apie tinkle esančius įrenginius.

Autonet algoritmas

Autonet programos algoritmo pagrindinė užduotis – konfigūruoti tradicinius įrenginius hibridiniame tinkle.

Iš pradžių Autonet programa kreipiasi į OpenDayLight kontrolerio API. Jis grąžina prijungtų prie kontrolerio OpenFlow įrenginių informaciją JSON formatu. Tada programa konvertuoja JSON formatą į masyvą. Vėliau yra išsaugomi OpenFlow įrenginių MAC adresai. Pagal MAC adresą yra atskiriamas tradicinis įrenginys nuo OpenFlow įrenginio. Tada Autonet SNMP protokolu kreipiasi į pirmąjį įrenginį. Iš jo gauna administravimo adresą pagal VLAN numerį. Tuomet patikrinama ar naudojamas adresas yra administravimo. Jeigu ne, tai toliau jungiantis prie įrenginio, bus naudojamas administravimo adresas ir šis adresas yra patalpinamas į administruojamų įrenginių sąrašą; jei taip, tai ir toliau bus naudojamas šis adresas ir jis taip pat yra pridedamas prie administruojamų įrenginių sąrašo. Be to, yra gaunama informacija ir apie įrenginio kaimynus. Šią informaciją įrenginys surenka naudodamas LLDP protokolą. Įrenginys yra pridedamas į patikrintų įrenginių sąrašą, o jo kaimynai yra pridedami į bendrą kaimynų sąrašą. Po to yra parenkamas įrenginys iš bendro kaimynų sąrašo. Tada tikrinama ar įrenginys nėra patikrintų įrenginių sąraše. Jeigu yra, tai parenkamas sekantis įrenginys iš bendro kaimynų sąrašo. Jeigu nėra, tai vėl kreipiamasi į įrenginį SNMP protokolu ir kartojama tokia pat procedūra kaip su pirmuoju įrenginiu. Tai tęsiasi tol, kol nebelieka nepatikrintų įrenginių bendrame kaimynų sąraše. Tada programa praeina administruojamų įrenginių sąrašą ir užklausia MAC (angl. *media access control* – medijos pasiekiamumo kontrolė) adresų. Šie MAC adresai yra sutikrinami su OpenDayLight kontrolerio gautais MAC adresais ir jiems sutapus, įrenginys yra pridedamas prie OpenFlow įrenginių sąrašo. Tada yra nubraižoma nustatyta (išaiškinta) topologija su visai žinomais įrenginiais. Tuomet iš tradicinių įrenginių yra pašalinami programos sugeneruoti MAC lentų įrašai (pirmą kartą sudarant topologiją tokių įrašų nebūna, tačiau kartojant veiksmus yra ištrinami prieš tai sugeneruoti įrašai), jei tokie buvo pridėti. Iš OpenDayLight kontrolerio API gauto JSON failo yra surenkami OpenFlow įrenginių įrašai. Po to yra sugeneruojamas įrašas kiekvienam OpenFlow įrenginio kaimynui ir išsiunčiamas Netconf protokolu. Kai visi įrašai išsiunčiami, įrenginiai yra tikrinami iš naujo ir viskas kartojama ta pačia seka.

Supaprastintas Autonet algoritmas pateiktas 1 algoritme:

1 algoritmas. Hibridinio tinklo konfigūravimo algoritmas.

```
1  openFlowInformacija = openFlowKontrolerioUžklausaREST()
2
3  KIEKVIENAM openFlowInformacija[„mazgų informacija“] KAIP mazgoInformacija
4     openFlowMacAdresai -> mazgoInfroamcija[„mac adresai“]
5  PABAIGA
6
7  ip = pirmasisĮrenginys()
8  ištestuotiĮrenginiai -> ip
9  JEI įrenginioIpNėraAdministravimo(ip)
10    ip  = administravimoIp(ip)
11    ištestuotiĮrenginiai(ip)
12
13 KIEKVIENAM ĮrenginioKaimynai(ip) KAIP įrenginys
14    įrenginiųPrievadai -> įrenginioPrievadai(įrenginys)
15    administruojamiĮrenginiai -> administracijosAdresas(įrenginys)
16    JEI įrenginioAdresasAdministravimo(įrenginys)
17       visiĮrenginiai -> įrenginys
18    JEI įrenginysNeturiAdministravimoAdreso(įrenginys)
19       visiĮrenginiai -> įrenginys
20    JEI įrenginysNepasiekiamas(įrenginys)
21       ištestuotiĮrenginiai -> įrenginys
22 PABAIGA
23
24 KOL yraNeištestuotųĮrenginiųSąraše(visiĮrenginiai) KAIP įrenginys
25    įrenginiųPrievadai -> įrenginioPrievadai(įrenginys)
26    administruojamiĮrenginiai -> administracijosAdresas(įrenginys)
27    JEI įrenginioAdresasAdministravimo(įrenginys)
28       visiĮrenginiai -> įrenginys
29    JEI įrenginysNeturiAdministravimoAdreso(įrenginys)
30       visiĮrenginiai -> įrenginys
31    JEI įrenginysNepasiekiamas(įrenginys)
32       ištestuotiĮrenginiai -> įrenginys
33 PABAIGA
34
35 KIEKVIENAM openFlowMacAdresai KAIP openFlowMacAdresas
36    KIEKVIENAM administruojamiĮrenginiai KAIP įrenginys
37       įrenginioMacAdresai = gautiĮrenginioMacAdresus(įrenginys)
38       KIEKVIENAM įrenginioMacAdresai KAIP įrenginioMacAdresas
39          JEI macAdresasSutampa(openFlowMacAdresas, įrenginioMacAdresas)
40             openFlowĮrenginiai -> įrenginys
41       PABAIGA
42    PABAIGA
43 PABAIGA
44
45 nubraižytiTopologiją(administruojamiĮrenginiai, įrenginiųPrievadai)
46
47 ištrintiPridėtusMacĮrašus(pridėtiMacĮrašai)
48
49 KIEKVIENAM openFlowInformacija[„OpenFlow įrašai“] KAIP openFlowĮrašas
50    openFlowĮrenginys = openFlowĮrenginys(openFlowĮrašas)
51    openFlowĮrenginioKaimynai = openFlowĮrenginioKaimynai(openFlowĮrenginys)
52    KIEKVIENAM openFlowĮrenginioKaimynai KAIP įrenginys
53       pridėtiMacĮrašą(įrenginys, openFlowĮrašas)
54       pridėtiMacĮrašai -> įrenginys, openFlowĮrašas
55    PABAIGA
56 PABAIGA
```

Autonet programa

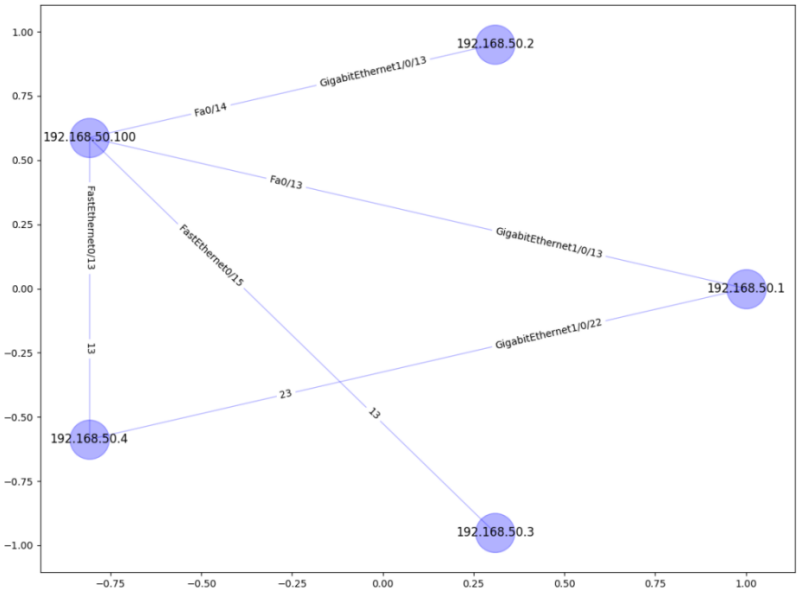
Programa parašyta Python [8] programavimo kalba. **Autonet** programos tekstas pateikiamas 1 priede, o programa patalpinta į github tinklalapį [9] ir yra viešai prieinama vartotojams.

Programoje buvo naudojama easysnmp biblioteka. Šios bibliotekos pagalba Python programos gali siųsti SNMP paketus bei gauti įrenginio atsakymus. Iš įrenginių buvo reikalinga informacija apie MAC adresus ir jų kaimynų adresus. Šiam tikslui programoje buvo naudojami šie OID:

- 1.0.8802.1.1.2.1.4.2.1.3 – šis OID yra iš LLDP MIB [10] duomenų bazės ir suteikia informaciją apie įrenginio kaimynų adresus.
- 1.3.6.1.2.1.2.2.1.6 – šis OID yra iš MIB-2 [11] (ISO standartinė duomenų bazė) duomenų bazės ir suteikia informaciją apie įrenginio prievadų MAC adresus. MAC adresai naudojami tam, kad pavyktų sutapatinti OpenFlow kontrolerio ir SNMP protokolo pateiktą informaciją.

OpenDayLight kontrolerio API buvo pasiekiamas httplib2 bibliotekos pagalba. Iš kontrolerio gautas JSON failas naudojantis json biblioteka buvo konvertuojamas į masyvą.

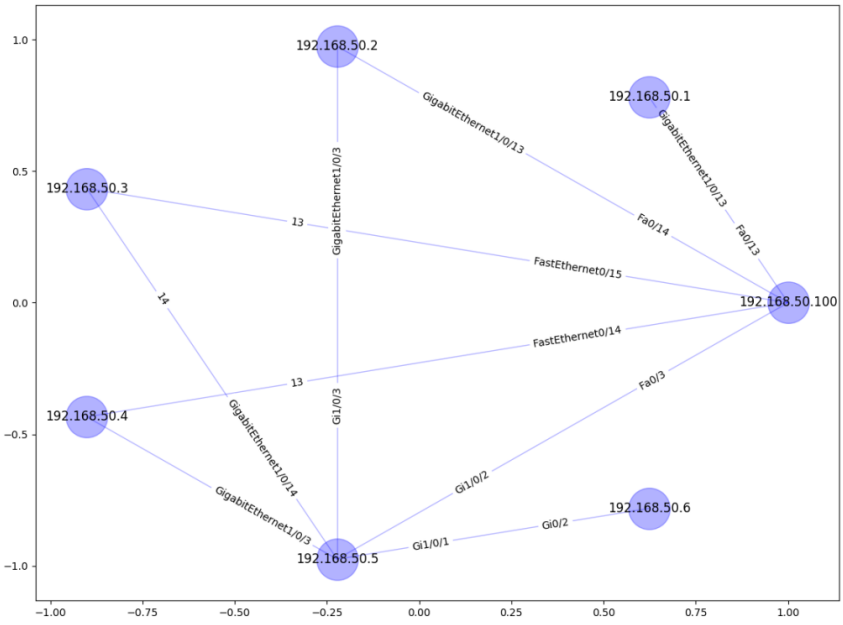
Netconf protokolo realizacijai buvo naudojama ncclient Python biblioteka. Pasitelkiant ncclient buvo sukuriama Netconf sesija su įrenginiu, nusiunčiama nauja konfigūracija ir ištrinama sena.



7 pav. Pirmoji topologija, kurioje 192.168.50.3 ir 192.168.50.4 yra OpenFlow įrenginiai.

Programa buvo testuojama sujungus tinklą pagal kelias topologijas. Pirmoje topologijoje (7 pav.) įrenginys adresu 192.168.50.100 buvo pasirinktas kaip pirmasis, nes jis šioje topologijoje yra tiesiogiai prijungtas prie kiekvieno kito įrenginio. Šioje ir kitose topologijose 192.168.50.3 ir 192.168.50.4 adresais yra įvardinti OpenFlow įrenginiai. Šios topologijos nustatymo trukmė yra 6

sekundės. Laikas tiesiogiai priklauso nuo įrenginių skaičiaus, kadangi su kiekvienu įrenginiu programa bando užmegzti SNMP sesiją ir gauti informaciją apie įrenginio kaimynus. Taigi, kuo daugiau įrenginių, tuo daugiau laiko užtrunka surinkti informacijai. Kaip ir čia, taip ir realiose sąlygose - geriausia pirmąjį įrenginį pasirinkti centrinį, nes jeigu pirmasis įrenginys būtų nepasiekiamas, programa negalėtų surasti nei vieno įrenginio.



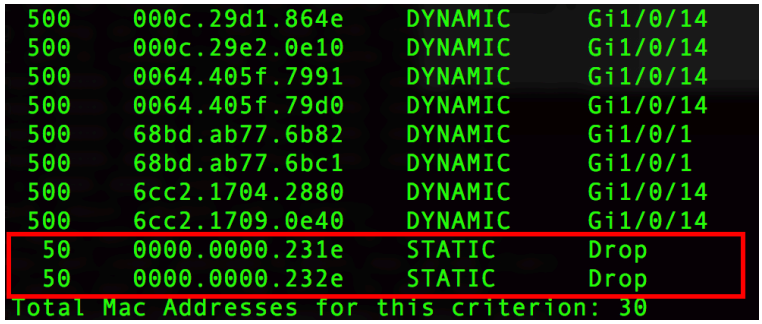
8 pav. Antroji topologija, kurioje 192.168.50.3 ir 192.168.50.4 yra OpenFlow įrenginiai.

Antroji topologijoja (8 pav.) yra labai panaši į pirmąją, tačiau joje yra daugiau įrenginių. Šios topologijos nustatymo trukmė - 7 sekundės. Automatizuotas įrenginių konfigūravimas buvo stebimas programos veikimo metu, pridedant įrašus tradiciniams įrenginiams. Prieš aktyvuojant programą, įrenginyje nebuvo jokių statinių įrašų (9 pav.).

```
 500    000c.29d1.864e    DYNAMIC     Gi1/0/14
 500    000c.29e2.0e10    DYNAMIC     Gi1/0/14
 500    0064.405f.7991    DYNAMIC     Gi1/0/14
 500    0064.405f.79d0    DYNAMIC     Gi1/0/14
 500    68bd.ab77.6b82    DYNAMIC     Gi1/0/1
 500    6cc2.1704.2880    DYNAMIC     Gi1/0/14
 500    6cc2.1709.0e40    DYNAMIC     Gi1/0/14
Total Mac Addresses for this criterion: 27
```

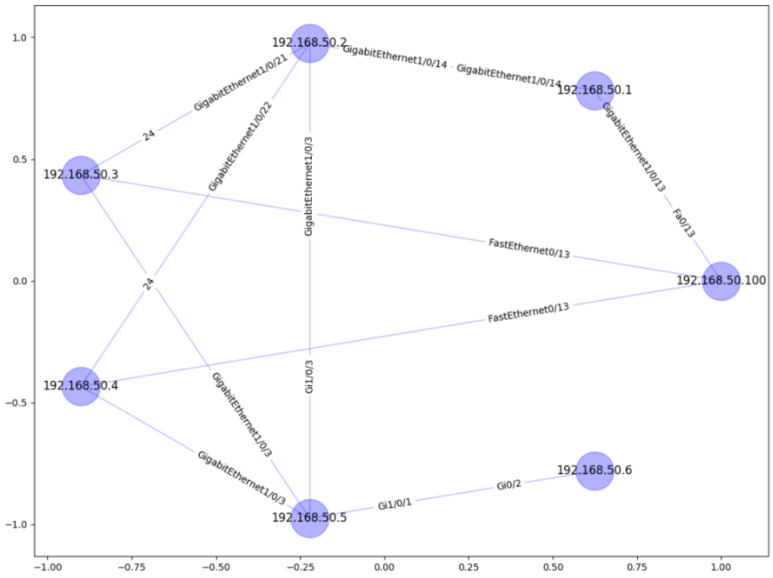
9 pav. MAC adresų lenta prieš konfigūravimą.

Aktyvavus programą buvo pridėti statiniai (angl. *Static*) įrašai (10 pav.), tuo tarpu seni nebereikalingi įrašai buvo ištrinami.



10 pav. Tradiciniame įrenginyje pridėti įrašai apibraukti raudonai.

Trečioji topologija (11 pav.) yra sudėtingesnė, kadangi norint nustatyti visus įrenginius reikia atlikti daugiau žingsnių. Pirmasis įrenginys mato tik kelis komutatorius iš visos topologijos, o šie taip pat nemato visų įrenginių. Tačiau topologijai surinkti užteko 7 sekundžių. Taigi, nesunku pastebėti, jog topologijos nustatymo trukmė priklauso tik nuo įrenginių skaičiaus ir nepriklauso nuo to, kaip jie yra išsidėstę. Įrenginiai buvo konfigūruojami automatizuotai, analogiškai, kaip ir antroje topologijoje.



11 pav. Trečioji topologija, kurioje 192.168.50.3 ir 192.168.50.4 yra OpenFlow įrenginiai.

Panašiu principu hibridinio tinklo topologiją surenka HybNet [12] programa, kurią patentavo NEC korporacija. Programa topologijos informacijai surinkti taip pat naudoja LLDP protokolą, o komunikacijai su tradiciniais įrenginiais gali naudoti SNMP arba NETCONF protokolus. Programa su SDN kontroleriu komunikuoja per API. HybNet šiuo metu turi daugiau funkcionalumo (automatizuotai konfigūruoja tradicinių įrenginių VLAN, surinktoje topologijoje saugo ir galinius įrenginius ir kt.) konfigūruojant tradicinius įrenginius, tačiau Autonet privalumas yra tas, jog ši programa yra atviro kodo, todėl kiekvienas gali paprastai papildyti programą sau reikalingu funkcionalumu.

Perspektyviniai programos patobulinimai

Pilnam Autonet programos išbaigimui ir paruošimui realiam tinklui reikalingi keli patobulinimai. Visų pirma, programą reikėtų perrašyti naudojant karkasą (angl. *framework*). Tai leistų paprasčiau valdyti programoje įvykusias klaidas bei leistų patogiau integruoti API. Tokiu būdu realiomis sąlygomis galima būtų pasiimti iš programos duomenis ar papildomai konfigūruoti įrenginius naudojant kitas sistemas. Be to, karkaso pagalba galima paprasčiau konfigūruoti pasikartojančius darbus pagal grafiką. Tarkime, galima būtų sukonfigūruoti, jog pilnai surenkama tinklo informacija būtų kiekvieną naktį, o dieną - ieškomi tik pakeitimai. Priedo, reikėtų pridėti funkcionalumą, jog pirmasis įrenginys būtų ne vienas, o keli, užtikrinant programos veikimo patikimumą, jei įrenginys taptų nepasiekiamas. Taip pat, reikėtų programai pridėti duomenų bazės palaikymą. Šiuo metu programa saugo duomenis apie įrenginius operatyviojoje atmintyje. Taigi, jeigu įrenginys, kuriame veikia programa netikėtai netektų elektros tiekimo (nors turėtų būti aprūpintas nepertraukiamos energijos šaltiniu), dingtų visa informacija apie įrenginius. To galima išvengti duomenis įrašant į duomenų bazę. Kitas vartotojui svarbus komponentas yra grafinė vartotojo sąsaja. Reikėtų parašyti programai grafinę sąsają, kurioje vartotojas galėtų patogiau matyti esamą topologiją, joje esančius įrenginius. Administratorius galėtų konfigūruoti programos nustatymus, tokius kaip įrenginių prisijungimo slaptažodžiai. Norint padidinti programos veikimo spartą, reikėtų perrašyti programą naudojant asinchroninius algoritmus. Taip galima būtų kreiptis į daug įrenginių iš karto, o ne po vieną iš eilės. Be to, reikėtų pridėti įvairesnių OpenFlow įrašų palaikymą, kadangi dabar programa pasiima iš OpenFlow komutatorių tik tuos įrašus, kuriuose yra paketo atmetimo komanda. Taigi, pilnai naudotis programa realiame tinkle reikėtų kelių papildomų patobulinimų, tačiau pagrindas sėkmingam hibridinio tinklo valdymui yra parašytas.

Pagrindiniai rezultatai ir išvados

Darbo metu buvo išanalizuoti topologijos nustatymo metodai tradiciniame, OpenFlow ir hibridiniame tinkle. Buvo išnagrinėti SNMP, OpenFlow, LLDP ir Netconf protokolai ir jų panaudojimo galimybės siekiant realizuoti darbo tikslą.

Sukurtas algoritmas ir programa įgalinantys nustatyti hibridinio tinklo topologiją bei automatiškai ir centralizuotai sukonfigūruoti tradicinius įrenginius, esančius jame.

Nustatyta, kad parašyta programa veikia stabiliai, realizuoja numatytas funkcijas ir išsprendžia hibridinio tinklo keliamas valdymo problemas, palengvindama perėjimą iš tradicinio tinklo į SDN tinklą.

Testų metu nustatyta tinklo topologijos įvertinimo trukmė, priklausomai nuo topologijos sudėtingumo, neviršijo 6 – 7 sekundžių ir yra priimtina realiam organizacijos tinklui.

Literatūra

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communication Review, 38 (2), p. 69-74 (2008), https://www.net.t-labs.tu-berlin.de/teaching/ss09/IR_seminar/papers/openflow-wp-latest.pdf (peržiūrėta 2019-05-16)

[2] W. Stallings, Software-Defined Networks and OpenFlow, The Internet Protocol Journal, 16 (1), p. 2-14 (2013), https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html (peržiūrėta 2019-05-16)

[3] P. Congdon, Link Layer Discovery Protocol and MIB v2.0, IEEE 802.1 (2002), http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-02.pdf (peržiūrėta 2019-05-16)

[4] D. Harrington, R. Presuhn and B. Wijnen, An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC3411 (2002), https://tools.ietf.org/html/rfc3411.html (peržiūrėta 2019-05-16)

[5] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network Configuration Protocol (NETCONF), RFC6241 (2011) https://tools.ietf.org/html/rfc6241 (peržiūrėta 2019-05-16)

[6] OpenDayLight, https://www.opendaylight.org/ (peržiūrėta 2019-05-16)

[7] K. Beliašas, Informacijos apie topologiją hibridiniuose tinkluose surinkimas pritaikant SNMP, kursinis darbas (2018)

[8] Python, https://www.python.org/ (peržiūrėta 2019-05-16)

[9] https://github.com/kbeliasas/Autonet (peržiūrėta 2019-05-16)

[10] LLDP MIB OID, http://www.mibdepot.com/cgi-bin/getmib3.cgi?win=mib_a&r=cisco&f=LLDP-MIB-V1SMI.my&v=v1&t=tree (peržiūrėta 2019-05-16)

[11] OID information http://www.oid-info.com/ (peržiūrėta 2019-05-16)

[12] N. Arora, H. Zhang, C. Lumezanu, J. Rhee, G. Jiang, and H. Lu, Hybrid Network Management, Patentas US 9.450,823 B2, https://patentimages.storage.googleapis.com/0a/6f/2c/4f60eac1978ef1/US9450823.pdf (peržiūrėta 2019-05-16)

Santrauka

Automatizuotas tradicinių įrenginių konfigūravimas hibridiniuose tinkluose

Karolis Beliašas

Darbo tikslas – išnagrinėti centralizuoto tradicinių įrenginių konfigūravimo metodus, topologijos surinkimo būdus, taip pat parašyti programą, kuri automatizuotai konfigūruotų tradicinius įrenginius hibridiniame tinkle. Pagrindinės darbo užduotys:

- išnagrinėti SNMP, LLDP, OpenFlow ir Netconf protokolų veikimą ir sintaksę;
- sukurti hibridinio tinklo topologijos surinkimo ir automatizuoto tradicinių įrenginių konfigūravimo algoritmą;
- parašyti algoritmą realizuojančią programą;
- ištestuoti programą realioje aplinkoje ir nustatyti algoritmo topologijos įvertinimo parametrus.

Darbo rezultatas – parašyta programa, kuri tiksliai nustato hibridinio tinklo topologiją bei konfigūruoja įrenginius.

Summary

Automated device configuration in hybrid network

Karolis Beliašas

Aim of the thesis is to examine centralized traditional device configuration methods, topology collection methods in legacy infrastructure and therefore write the program enabling automated traditional device configuration in hybrid network. The main objectives of this thesis were as follows:

- examine how SNMP, LLDP, OpenFlow, Netconf works and analyze their syntax;
- create algorithm seeking to collect hybrid network topology and automated traditional device configuration;
- write a program for algorithm realization;
- test program in real environment and find topology evaluation parameters for algorithm.

The main results of this work: program which precisely collects hybrid network topology and configure traditional devices is written.

1 priedas. Autonet programos kodas

```
#!/usr/bin/env python

import easysnmp
import httplib2
import networkx as nx
import matplotlib.pyplot as plt
import json
import sys
import logging
from ncclient import manager
import time

log = logging.getLogger(__name__)

ip = '192.168.50.100'
com = 'public'
man_vlan = '500'
prod_vlan = "50"
user = "admin"
password = "cisco"
added_macs = []

ADD_MAC_ADDRESS = """
<config>
      <cli-config-data>
         <cmd>mac address-table static %s vlan %s drop</cmd>
      </cli-config-data>
</config>
"""

REMOVE_MAC_ADDRESS = """
<config>
      <cli-config-data>
         <cmd>no mac address-table static %s vlan %s drop</cmd>
      </cli-config-data>
</config>
"""

def get_ip_add(ip):
   session = easysnmp.Session(hostname=ip, version=2, community=com, retries=1)
```

```
    try:
        res = session.walk('.1.0.8802.1.1.2.1.4.2.1.3')
        ans = []
        temp1 = []
        for item in res:
            temp1.append(item.oid)
        c = 0
        for item in temp1:
            ans.append('')
            a = 0
            b = 1
            while a < 4:
                if (item[len(item)-b] == '.'):
                    ans[c] = item[len(item)-b] + ans[c]
                    a = a + 1
                    b = b + 1
                else:
                    ans[c] = item[len(item)-b] + ans[c]
                    b = b + 1
            ans[c] = ans[c].lstrip('.')
            c = c + 1
        return ans
    except Exception as e:
        print 'Something wrong on ip = ' + ip
        print(e)
        ans = failed
        return ans


def get_man_ip_add(ip):
    try:
        session = easysnmp.Session(hostname=ip, version=2, community=com)
        res = session.walk('.1.3.6.1.2.1.4.20.1.2')
        res1 = None
        for item in res:
            if item.value == man_vlan:
                res1 = item.oid_index
                break
        if res1 == None:
            for item in res:
                if item.value == str(int(float(man_vlan)) + 329):
                    res1 = item.oid_index
                    break
```

```
        if res1 == None:
            for item in res:
                if item.value == "34":
                    res1 = item.oid_index
                    break
        return res1
    except Exception as e:
        print 'Something wrong on ip = ' + ip
        print(e)
        ans = 'failed'
        return ans

def get_id_port(ip): # Pasiima kaimynu sasaju numerius
    session = easysnmp.Session(hostname=ip, version=2, community=com)
    try:
        res = session.walk('.1.0.8802.1.1.2.1.4.1.1.7')
        ans = []
        for item in res:
            ans.append(item.value)
        return ans
    except Exception as e:
        ans = []
        ans.append('')
        return ans

def get_port_macs(ip):
    session = easysnmp.Session(hostname=ip, version=2, community=com, use_sprint_value=True)
    res = session.walk('.1.3.6.1.2.1.2.2.1.6')
    ans = []
    for item in res:
        ans.append(item.value)
    return ans

def mac_corr(string):
    a = 0
    x = 0
    while (x > -1):
        if (x < len(string)):
            if (string[x] != ':'):
                a = a + 1
            elif(string[x] == ':'):
                if (a < 2):
```

```
                    a = 0
                    string = string[:x-1] + '0' + string[x-1:]
                    x = x + 1
                elif (a >= 2):
                    a = 0
            x = x + 1
        else:
            if (a < 2):
                string = string[:x-1] + '0' + string[x-1:]
            x = -2
            break
    return string

def mac_corr1(mac):
    mac = mac.replace(":","")
    mac = mac[:4] + "." + mac[4:]
    mac = mac[:9] + "." + mac[9:]
    return mac

def _check_response(rpc_obj, snippet_name):
    log.debug("RPCReply for %s is %s" % (snippet_name, rpc_obj.xml))
    xml_str = rpc_obj.xml
    if "<ok />" in xml_str:
        log.info("%s successful" % snippet_name)
    else:
        log.error("Cannot successfully execute: %s" % snippet_name)

def add_mac_second(host, user, password, mac, vlan):
    with manager.connect(host=host, port=22, username=user, password=password,
hostkey_verify=False, allow_agent=False, look_for_keys=False) as m:
        try:
            confstr = ADD_MAC_ADDRESS % (mac, vlan)
            rpc_obj = m.edit_config(target='running', config=confstr)
            _check_response(rpc_obj, 'ADD_MAC_ADDRESS')
        except Exception:
            log.exception("Exception in adding %s mac %s vlan" % (mac, vlan))

def remove_mac_second(host, user, password, mac, vlan):
    with manager.connect(host=host, port=22, username=user, password=password,
hostkey_verify=False, allow_agent=False, look_for_keys=False) as m:
        try:
            confstr = REMOVE_MAC_ADDRESS % (mac, vlan)
```

```
                rpc_obj = m.edit_config(target='running', config=confstr)
                _check_response(rpc_obj, 'REMOVE_MAC_ADDRESS')
            except Exception:
                log.exception("Exception in removing %s mac %s vlan" % (mac, vlan))

        def add_mac(host,user,password,mac,vlan):
            if __name__ == '__main__':
                logging.basicConfig(level=logging.ERROR)
                add_mac_second(host,user,password, mac, vlan)

        def remove_mac(host,user,password,mac,vlan):
            if __name__ == '__main__':
                logging.basicConfig(level=logging.ERROR)
                remove_mac_second(host,user,password, mac, vlan)

        def draw_topology(graph, labels, graph_layout='shell',
                          node_size=1600, node_color='blue', node_alpha=0.3,
                          node_text_size=12,
                          edge_color='blue', edge_alpha=0.3, edge_tickness=1,
                          edge_text_pos=0.3,
                          text_font='sans-serif'):
            G=nx.Graph()

            for edge in graph:
                G.add_edge(edge[0], edge[1])

            if graph_layout == 'spring':
                graph_pos=nx.spring_layout(G)
            elif graph_layout == 'spectral':
                graph_pos=nx.spectral_layout(G)
            elif graph_layout == 'random':
                graph_pos=nx.random_layout(G)
            else:
                graph_pos=nx.shell_layout(G)


            nx.draw_networkx_nodes(G, graph_pos, node_size=node_size, alpha=node_alpha,
node_color=node_color)
            nx.draw_networkx_edges(G, graph_pos, width=edge_tickness, alpha=edge_alpha,
edge_color=edge_color)
            nx.draw_networkx_labels(G, graph_pos, font_size=node_text_size, font_family=text_font)
```

```
            if labels is None:
                labels = range(len(graph))

            edge_labels = dict(zip(graph, labels))
            nx.draw_networkx_edge_labels(G, graph_pos, edge_labels=edge_labels,
label_pos=edge_text_pos)

            plt.show()




        while True:
            start_time = time.time()
            h = httplib2.Http(".cache")
            h.add_credentials('admin', 'admin')
            resp, content = h.request('http://192.168.50.254:8181/restconf/operational/opendaylight-
inventory:nodes', "GET")
            OF_swi = []
            all_OF_node_ports = []
            all_OF_node_macs = []

            allOFnodes = json.loads(content)

            for node in allOFnodes['nodes']['node']:
                OF_node_macs = []
                for node_connector in node['node-connector']:
                    OF_node_macs.append(node_connector['flow-node-inventory:hardware-address'])
                all_OF_node_macs.append(OF_node_macs)

            tested = []
            tested.append(ip)
            manage = []
            manage.append(get_man_ip_add(ip))
            if ip != get_man_ip_add(ip):
                ip = get_man_ip_add(ip)
                tested.append(ip)
            all_list = []
            temp_list = []
            all_ports_topology = []
            port_list = []
            device_nei_ports = []
```

```
for item in get_ip_add(ip):
    if get_man_ip_add(item) == 'failed':
        temp_list.append(item)
        tested.append(item)
    elif get_man_ip_add(item) == None:
        print "%s Don't have man VLAN" % item
        temp_list.append(item)
        tested.append(item)
    else:
        temp_list.append(get_man_ip_add(item))
all_list.append(temp_list)

for item in get_id_port(ip):
    all_ports_topology.append(item)
    port_list.append(item)

device_nei_ports = [port_list]



x = 0
y = 0
z = 0
while (x > -1):
    while (y > -1):
        while (z > -1):
            if (all_list[x][y] == tested[z]):
                if (len(all_list[x]) > y + 1):
                    y = y + 1
                    z = 0
                elif (len(all_list[x]) <= y + 1):
                    if (len(all_list) > x + 1):
                        x = x + 1
                        y = 0
                        z = 0
                    elif (len(all_list) <= x + 1):
                        x = -2
                        y = -2
                        z = -2
                        break
            elif (all_list[x][y] != tested[z]):
                if (len(tested) > z + 1):
                    z = z + 1
```

```
elif (len(tested) <= z + 1):
    ip = all_list[x][y]
    if ip != get_man_ip_add(ip):
        b = 0
        while (b > -1):
            if manage[b] == get_man_ip_add(ip):
                tested.append(ip)
                b = -2
                break
            elif manage[b] != get_man_ip_add(ip):
                if (len(manage) > b + 1 ):
                    b = b + 1
                elif (len(manage) <= b + 1):
                    tested.append(ip)
                    tested.append(get_man_ip_add(ip))
                    manage.append(get_man_ip_add(ip))
                    temp_list = []
                    for item in get_ip_add(ip):
                        if get_man_ip_add(item) == 'failed':
                            temp_list.append(item)
                            tested.append(item)
                        elif get_man_ip_add(item) == None:
                            temp_list.append(item)
                            tested.append(item)
                            print "%s Don't have man VLAN" % item
                        else:
                            temp_list.append(get_man_ip_add(item))
                    all_list.append(temp_list)

                    port_list = []

                    for item in get_id_port(ip):
                        all_ports_topology.append(item)
                        port_list.append(item)

                    device_nei_ports = [port_list]

                    b = -2
                    break
    elif ip == get_man_ip_add(ip):
        tested.append(ip)
        manage.append(ip)
```

```
                temp_list = []
                for item in get_ip_add(ip):
                    if get_man_ip_add(item) == 'failed':
                        temp_list.append(item)
                        tested.append(item)
                    elif get_man_ip_add(item) == None:
                        temp_list.append(item)
                        tested.append(item)
                        print "%s Don't have man VLAN" % item
                    else:
                        temp_list.append(get_man_ip_add(item))
                all_list.append(temp_list)

                port_list = []

                for item in get_id_port(ip):
                    all_ports_topology.append(item)
                    port_list.append(item)

                device_nei_ports = [port_list]

            if (len(all_list[x]) > y + 1):
                y = y + 1
                z = 0
            elif (len(all_list[x]) <= y + 1):
                if (len(all_list) > x + 1):
                    x = x + 1
                    y = 0
                    z = 0
                elif (len(all_list) <= x + 1):
                    x = -2
                    y = -2
                    z = -2
                    break


for x in range(0, len(all_OF_node_macs)):
    for y in range(0, len(manage)):
        try:
            macs = get_port_macs(manage[y])
            for mac in macs:
                mac = mac_corr(mac)
```

```
            if (mac == all_OF_node_macs[x][0]):
                OF_swi.append(manage[y])
                break
        except Exception as e:
            print "miss"

realations = []

for x in range(0, len(manage)):
    for y in range(0, len(all_list[x])):
        temp = (manage[x], all_list[x][y])
        realations.append(temp)



elapsed_time = time.time() - start_time
print ("Topology gathered in %s s." % elapsed_time )
draw_topology(realations,all_ports_topology)



for list1 in added_macs:
    try:
        remove_mac(list1[0],user,password,list1[1],prod_vlan)
        print "removed mac %s form %s" % (list1[1], list1[0])
    except Exception:
        print "Nothing to remove"

added_macs = []
dest_mac = []
node1 = []

for node in allOFnodes['nodes']['node']:
    for OF in OF_swi:
        try:
            macs = get_port_macs(OF)
            for mac in macs:
                mac = mac_corr(mac)
                for node_connector in node['node-connector']:
                    if mac == node_connector['flow-node-inventory:hardware-address']:
                        node1 = OF
                    break
        except Exception:
            print "miss"
```

```
                for node_table in node['flow-node-inventory:table']:
                    if node_table["id"] == 0:
                        try:
                            for node_table_flow in node_table["flow"]:
                                try:
                                    dest_mac = node_table_flow["match"]["ethernet-match"]["ethernet-
destination"]["address"]
                                    dest_mac = mac_corr1(dest_mac)
                                    print "dest mac = %s" % dest_mac
                                    for x in range(0, len(manage)):
                                        if node1 == manage[x]:
                                            for swi in all_list[x]:
                                                try:
                                                    add_mac(swi,user,password,dest_mac,prod_vlan)
                                                    added_macs_temp = [swi,dest_mac]
                                                    added_macs.append(added_macs_temp)
                                                    print "Added %s successfully to %s" % (dest_mac, swi)
                                                except Exception:
                                                    print "Device %s can't use netconf" % swi
                                except Exception:
                                    err = 0
                        except Exception:
                            print "No flow entries in %s" % node["id"]



            print 'OpenFlow = %s' % OF_swi
            print "added_macs =  %s" % added_macs

            print 'Manage = %s' % manage
            print 'all_list %s= ' % all_list
            print 'tested = %s' % tested
            #time.sleep(5)
```