TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Tanel Tomson

Võrgutopoloogia visualiseerimine

Bakalaureusetöö (9 EAP)

Juhendaja: Meelis Roos, MSc

Tartu 2019

Võrgukommutaatorite visualiseerimine

Lühikokkuvõte:

Võrguadministraatori hallatavas kohtvõrgus võib arvutite ja neid kohtvõrku ühendatavate võrgukommutaatorite arv olla suur. Kohtvõrgu topoloogiast ülevaate saamine võib muutuda tülikaks. Bakalaureusetöö eesmärgiks on luua lihtsasti kasutatav, eraldiseisev ja graafilist väljundit pakkuv töövahend võrguadministraatorile kohtvõrgu topoloogiast ülevaate saamiseks.

Töö raames kirjutati programm, mis küsib seadistatud võrguseadmetelt üle SNMP protokolli andmed tema naabrite kohta ja koostab seejärel veebilehe, kus kuvatakse seadmete topoloogia graafina. Rakenduse paigaldamine ja käivitamine on lihtne, vajalik ei ole mahuka võrguhaldustarkvara paigaldamine. Valminud rakendust testiti Cybernetica AS kohtvõrgus.

Võtmesõnad:
Kohtvõrgu topoloogia, visualiseerimine, SNMP, CDP, LLDP

CERCS: P175 (Informaatika, süsteemiteooria)

Visualizing network topology

Abstract:

The number of computers and network switches that connect the computers to a local area network can grow large. Maintaining an overview of the topology of those networks can get cumbersome to network administrators. The aim of this thesis is to write a standalone tool that is simple to use and produces a graphical overview of the topology of a local area network for network administrators to use.

As a result a program was written. The program asks configured network devices for information about their neighbours using the SNMP protocool and proceeds to create a website displaying the network topology as a graph. The installation and running of the program is simple, there is no need to install a large network monitoring system. Finished application was tested in the local area network of Cybernetica AS.

Keywords:
Local area network topology, visualization, SNMP, CDP, LLDP

CERCS: P175 (Informatics, systems theory)

Sisukord

1 Sissejuhatus

Võrgukommutaator (inglise keeles *switch*, edaspidi ka lihtsalt kommutaator) on võrguseade, mis ühendab seadmeid kohtvõrku. Näiteks kontorites, kus on reeglina hulgaliselt võrguseadmeid (arvutid, printerid jne), kasutatakse kommutaatoreid, et suur arv seadmeid saaksid kohtvõrguga ühenduda.

Võrgukommutaatoritest võib mõelda kui sorteerimisekeskustest postiteenuses. Igal sorteerimiskeskusesse jõudval kirjal on sihtkoht, selleks on ümbrikul addressaat ja aadress. Sorteerimiskeskuses otsustatakse, kuhu iga kiri edasi tuleb saata. Sama ülesanne on ka võrgukommutaatoril – otsustada, millisele ühendatud võrguseadmele iga pakett (kiri) edasi saata.

Kommutaatoreid – mida reeglina haldavad süsteemiadministraatorid – võib arvutivõrgus olla mitu. Kohtvõrgu topoloogiast (kuidas kommutaatorid üksteisega ühendatud on) ülevaate saamine võib muutuda tülikaks ja aeganõudvaks. Kommutaatoritelt saab küll naabrite kohta infot küsida, ent liidesed selleks on disainitud eelkõige masinloetavaks, mis teeb nende otse kasutamise inimesele ebamugavaks.

Antud töö eesmärk on luua graafilist väljundit pakkuv eraldiseisev töövahend kohtvõrgu topoloogiast ülevaate saamiseks. Fookus on just kommutaatorite üksteise suhtes paiknemise visualiseerimisel, jättes lisainformatsiooni (muud ühendatud seadmed) tagaplaanile.

Järnevalt on toodud töö struktuur. Peatükk 2 annab tehnilise ülevaate protokollidest, mida rakendus kasutab. Peatükis 3 kirjeldatakse rakenduse jooksutamiseks vajalikud tingimused ning enne rakenduse arendamist paika pandud nõuded. Peatükis 4 tuuakse ülevaade sarnast funktsionaalsust pakkuvatest rakendustest. 5 kirjeldab tehnoloogiaid ja teeke, mida programm kasutab või mida kasutati arendusprotsessis. Peatükis 6 tehakse detailne ja tehniline ülevaade rakenduse arhitektuurist. Samuti põhjendatakse tehtud otsuseid ning näidatakse rakenduse nõuete täitmist ja tuuakse puudused.

2 Tehniline taust

Järgnevalt teeme tehnilise sissejuhatuse võrguprotokollidele, mille abil saab võrguseadmetelt haldusinfot küsida. Antud protokollid on loodud haldusandmete jagamiseks ja loovad vundamendi meie rakenduse kirjutamiseks.

2.1 CDP ja LLDP

CDP (*Cisco Discovery Protocol*) ja LLDP (*Link Layer Discovery Protocol*) on mõlemad võrguprotokollid ühendatud võrguseadmetele haldusinformatsiooni jagamiseks. LLDP ja CDP on funktsionaalsuse poolest küllaltki sarnased. Peamine erinevus kahe protokolli vahel on, et CDP on Cisco (võrguseadmete tootja) loodud ja hallatav omandprotokoll [Bha15a]. LLDP seevastu on avatud ja seadmetootjate ülene protokoll, LLDP standardit haldab IEEE [Bha15b].

2.2 SNMP

SNMP (*Simple network management protocol*) on laialdaselt kasutatav võrguhaldusprotokoll, mis kuulub OSI mudelis rakenduskihti [Laa08, 151]. Protokoll lihtsustab ja ühtlustab võrguseadmetelt haldusinfo küsimist [Laa08, 151].

Hallatavad seadmed hoiavad haldusinfot MIB (*Management Information Base*) struktuuris, mis on hierarhiline andmebaas [Laa08, 152]. Igal objektil on oma unikaalne identifikaator (OID, *Object Identifier*) [Laa08, 152]. Valminud rakendus kasutab konkreetseid OID-sid, et kommutaatoritelt vajalikke andmeid küsida.

SNMP protokoll defineerib päringute päises ka parameetri kogukonnasõne (*community string*). Kogukonnasõne võimaldab teha päringutele triviaalset autentimist (nt lubada andmete pärimist vaid kindlate kogukonnasõnedega) [Laa08, 154-155].

SNMP-st on 3 põhilist versiooni: 1, 2 ja 3 [Laa08, 151]. Valminud rakendus kasutab versiooni 2 (täpsemalt 2c).

Valminud rakendus kasutab SNMP protokolli, et võrguseadmetelt CDP ja LLDP andmeid küsida.

3 Rakenduse nõuded

Enne rakenduse programmeerimist pandi paika visioon sellest, kuidas ja mida rakendus tegema peab. Vajadus oli programmi järele, mis teeks ühte konkreetset asja hästi, mitte ei prooviks kompromissidega pakkuda suurt funktsionaalsust. Rakenduse paigaldamine ja käivitamine peab olema lihtne, ei tohi eeldada ühegi suurema võrguhaldustarkvara olemasolu. Võrguadministraator peaks saama seda lihtsasti käsitsi käivitada, vastavalt enda töövoole. Samuti peab võimalik olema programmmi automaatne perioodiline käivitamine ja tulemuse (genereeritud veebilehe) serveerimine vabalt valitud veebiserveri kaudu.

Et kõik rakenduse eesmärkidest ühtmoodi aru saaksid, fikseeriti töö algusjärgus rakenduse nõuded, mis järnevalt välja tuuakse.

3.1 Kasutamise eeldused

Rakenduse kasutamiseks on tehtud eeldus, et kasutajal on kontroll kaardistatavate võrgukommutaatorite üle. Võrgukommutaatorid peavad olema programmi jaoks kättesaadavad ja vajalikku informatsiooni edastama. See tähendab, et võrgukommutaatoritel peab töötama SNMP teenus ja see peab olema seadistatud jagama CDP või LLDP andmeid.

Rakenduse jooksutamiseks on peamine riistvaraline nõue, et masinal peab olema ligipääs seadmetele, millelt andmeid küsitakse. Rakendus arendati peamiselt operatsioonisüsteemi Linux silmas pidades ja selle jaoks koostati ka käivitamisjuhend ja teostati rakenduse testimine. On tõenäoline, et rakendus toimib ka teistel operatsioonisüsteemidel, ent seda ei testitud.

3.2 Rakenduse nõuded

1. Rakendus peab toimima vähemalt operatsioonisüsteemil Linux.
2. Programmeerimmiskeeleks sobib Python.
3. Rakenduse käivitamine toimub käsurealt.
4. Võimalusel kasutame mõnda olemasolevat SNMP klienditeeki.
5. Rakenduse sisendiks on ühe või mitme võrgukommutaatori aadress ja seadmetes seadistatud SNMP kommuuni nimi.
6. Kui antakse ette mitu seadet, on võimalus määrata erinevatele seadmetele erinevad SNMP kommuunide nimed.

7. Rakendus küsib igalt sisendiks saadud seadmelt LLDP/CDP andmed tema enese kohtha (nimi, IP-aadress, ühenduste kirjeldused) ja andmed iga tema naabri kohta (nimi, IP-aadress, ühenduse kirjeldus).
8. Rakendus töötleb ja vormindab saadud andmed.
9. Tekkinud vead kuvatakse kasutajale mõistlikult (mis juhtus, mida teha).
10. Rakendus loob antud andmete põhjal staatilise veebilehe (.html), kus kuvatakse graaf võrgukommutaatorite kohta.
11. Staatilise veebilehe vaatamiseks pole vaja internetiühendust (CSS ja JS on lähtekoodiga kaasas, mitte ei laeta üle võrgu).
12. Veebilehte saab vaadata ka ilma veebiserverita (üle `file://` URI-skeemi).
13. Graafi tipud on võrgukommutaatorid.
14. Graafi servad tippude vahel on võrgukommutaatorite ühendused, erinevad ühenduste kiirused on tähistatud erinevalt.
15. Graafis on vaikimisi paigutus mõistlik, see tähendab, et tipud ei kattu ja kui tekib mitu graafi, kuvatakse need üksteisest pisut eemal.
16. Graafis on võimalik seadmeid enda suva järgi lohistada.
17. Graafis kuvatakse tippudes seadme nimi (aadress), tipule peale vajutades avaneb hüpikvihje.
18. Tipule peale vajutades on võimalik näha nimekirja kommutaatoriga ühendatud teistest seadmetest.
19. Kasutaja graafile tehtud muudatused salvestatakse kasutaja brauseriga lokaalselt (küpsised või localstorage) ja taastatakse järgmisel avamisel.

4 Sarnased lahendused

Käesolevas peatüki eesmärk on tutvustada ja anda ülevaade sarnastest lahendustest, tuues välja nende tugevused ja puudused.

4.1 Netcrawl

Netcrawl on lihtne programm, mille sisendiks on ühe võrguseadme aadress ja SNMP kogukonnasõne. Rakendus küsib seejärel seadmelt naabrite info ja kordab tegevust naabrite puhul [Ytt]. Väljundfail on DOT-keeles[1] [Ytt], mida kasutab visualiseerimiseks Graphviz[2] – graafide visualiseerimise tarkvara. Tulemuseks on pildifail graafiga, kus on leitud võrgukommutaatorid ja nende naabrid [Ytt].

Netcrawl on üsna sarnane töö käigus loodava rakendusega. Rakenduse kasutamine on lihtne ja mõne üksiku mööndusega rahuldab programm meie nõuded. Küll aga on mõned olulised puudused. Suurim erinevus on programmi väljundi formaat - Netcrawli kasutamisel tekib pildifail. Nõue 16 täpsustab aga võimaluse kasutajal graafi tippe interaktiivselt ümber paigutada. Samuti paneb programm kõik leitud seadmed graafi (sh kommutaatoriga ühendatud teised seadmed) – see tähendab, et kui seadmeid on palju, muutub graaf segaseks. Loodud programmis tehti otsus, et kommutaatoritega ühendatud seadmeid graafis ei kuvata, vaid need kuvatakse graafi tippudele (kommutaatoritele) vajutades hüpikvihjetega (nõue 18). Lisaks ei saa Netcrawliga anda erinevatele kommutaatoritele erinevaid kogukonnasõnesid (nõue 6).

4.2 Netdisco

Netdisco on laia funktsionaalsusega võrguhaldustarkvara, mis on kirjutatud Perlis [Gor]. Rakendus koosneb mitmest komponendist: taustal jooksev deemon, mis arvutivõrgu kohta andmeid pärib, käsurealiides ja veebiserver kasutajaliidese jaoks [Gor]. Lisaks on rakenduse paigaldamiseks vajalik PostgreSQL andmebaas [Gor].

Rakendus pakub võrguseireks hulgaliselt erinevaid vaateid, sealhulgas oskab Netdisco ka seadme naabreid kaardistada. Selleks pakub Netdisco vaadet *Network Map*[3], mis koostab võrguseadmetest graafi. Antud funktsionaalsus täidab sama eesmärki, mis antud töö raames loodud rakendus. Näiteks saab kasutaja graafis tippe ümber paigutada, asetus salvestatakse ja taastatakse lehelt lahkudes. Lisaks pakub Netdisco lisafunktsionaalsust, näiteks saab reaalajas kuvatavaid seadmeid filtreerida, muuta graafi välimust ja liikuda erinevate seadmete vahel (kuvada järgmise seadme naabrid).

[1] `https://graphviz.gitlab.io/_pages/doc/info/lang.html`

[2] urlhttps://www.graphviz.org/

[3] `https://github.com/netdisco/netdisco/wiki/Network-Map`

Peamine erinevus Netdisco ja loodud rakenduse vahel on keerukus. Netdisco paigaldamine ja seadistamine on ajamahukam ja vajab rohkem ressursse. Netdisco eesmärk on aidata laiemalt võrguhaldusuega, loodud rakenduse eesmärk on teha vaid üht osa.

4.3 Switchmap

Switchmap[4] on tööriist võrgukommutaatorite haldusinfo kogumiseks ja vaatamiseks. Rakendus kogub seadmetelt üle SNMP andmed ja koostab saadud informatsiooniga veebilehe.

Erinevalt teistest kirjeldatud sarnastest rakendustest ei tee Switchmap andmete visualiseerimist, väljund on vormistatud hoopis tabelitena. Seetõttu sobib rakendus hästi kommutaatori ühendustest ülevaate saamiseks, aga kommutaatorite topoloogia uurimine – kuigi kõik selleks vajalikud andmed kuvatakse – on visuaalse ülevaate puudumise tõttu raskendatud.

[4] https://sourceforge.net/projects/switchmap/

5 Kasutatud tehnoloogiad

Antud peatükis tutvustatakse prgorammeerimiskeeli, teeke ning muid vahendeid, mida valminud rakendus kasutab.

5.1 Keeled

Rakenduse tagasüsteem on kirjutatud programmeerimiskeeles Python, kasutati versiooni 3. Peamine põhjus oli, et autoril oli varasem kogemus Pythoniga juba olemas. Samuti kontrolliti enne valiku tegemist, et leiduks sobiv SNMP teek, mis on rakenduse loomiseks vajalik.

Kasutajaliides on loodud kasutades HTML-i ja CSS-i. Lisaks on graafi kuvamiseks kasutatud programmeerimiskeelt Javascript.

5.2 easySNMP

Kommutaatoritelt üle SNMP protokolli andmete küsimiseks kasutab rakendus Pythoni teeki easySNMP. Töö algfaasis katsetati alternatiivina ka teeki PySNMP[5]. Valik tehti aga easySNMP kasuks, kuna PySNMP on kommutaatoritega suhtlemisel tuntavalt aeglasem [Eas].

5.3 Cytoscape.js

Järgmiseks oli vaja Javascripti teeki graafide brauseris esitamiseks. Andmeid visualiseerivaid (ja muuhulgas graafe toetavaid) teeke leidub arvukalt, ent autor kitsendas otsingu lihtsuse huvides vaid graafidele spetsialiseerunud teekidele. Samuti pidi teek olema avatud lähtekoodiga ning toetama graafi tippudele hüpikvihjete lisamist.

Kasutusele võeti Cytoscape.js[6], mis vastas kirjeldatud nõuetele [Cyt]. Pärast esmaseid katsetusi demodega tõusis esile hea dokumentatsioon, mis tegi teegi kasutamise lihtsaks, lai valik erinevaid kujundusi ning graafide kuvamisel mõistlik ekraanipinna kasutamine.

Kuigi teek vaikimisi hüpikvihjeid kuvada ei oska, on selleks loodud laiendeid. Hüpikvihjete kuvamiseks kasutusele võetud teegid on kirjeldatud peatükis 6.2.1.

[5] `http://snmplabs.com/pysnmp`
[6] `http://js.cytoscape.org/`

6 Valminud rakenduse kirjeldus

Rakenduse võib funktsionaalsuse järgi jagada kaheks: tagasüsteem ja kasutajaliides. Tagasüsteemi ülesanne on küsida ja töödelda sisendiks saadud võrguseadmetelt info ühendatud seadmete kohta. Kasutajaliides koostab tagasüsteemi poolt saadud andmetest veebilehe kommutaatorite graafiga. Peatükis kirjeldatakse detailselt mõlema osa arhitektuuri.

6.1 Tagasüsteem

Tagasüsteem on kirjutatud programmeerimiskeeles Python. Välistest teekidest on kasutusel vaid easySNMP (5.2). Tagasüsteemi lähtekood asub `src/` kaustas (va `src/web/`, mis on kasutajaliides).

6.1.1 Rakenduse seadistamine

Rakenduse sätted loeb programm failist `config.ini`. Rakenduse lähtekoodis on näidisfail `config.ini.sample`, mille rakenduse käivitaja saab kopeerida ja vastavalt soovidele muuta.

```
[Application Config]

# Default community string is used when switch has no specific
    community string set
defaultCommunityString = public

# Switches to be looked up.
switches = with-specific-community.example.com specificcommunity
           with-default-community.example.com
           another-with-default-community.example.com

# Enable or disable application logging (true or false)
debug = false
```

Joonis 1. Näidis sättefail config.ini.sample

Kasutaja saab määrata kommutaatorid, millelt naabrite info küsitakse. Seejuures saab igale kommutaatorile vajadusel määrata eraldi kogukonnasõne (vt 2.2 ja samuti nõue 6). Lisaks on võimalik seadistada rakenduse logimist silumise tasemele (täpsemalt vt 6.1.6).

Rakendus kasutab sätetefaili parsimiseks Pythoni moodulit configparser[7] ja sellest lähtuvalt on sätetefail INI-struktuuriga. Sätetefaili puudumisel või kui sätete lugemine

[7] `https://docs.python.org/3.6/library/configparser.html`

ebaõnnestub, kuvatakse kasutajale vastav veateade ja programm lõpetab töö. Sätete lugemise implementatsioon on failis `src/config_helper.py`.

6.1.2 Rakenduse põhivoog

Kui sätetefailist on seadistatud kommutaatorid loetud, asutakse neilt andmeid küsima. Seda tehakse ükshaaval, üle kommutaatorite itereerides. Vastuseks saadud andmed hoiustab programm mälus (sõnastikuna), seejuures jagatakse iga kommutaatori käest saadud andmed kolmeks:

1. kommutaatori enda andmed (graafis tipud)
2. kommutaatoriga ühendatud teiste kommutaatorite andmed (graafis servad)
3. ülejäänud kommutaatoriga ühendatud seadmete info (graafi tippude hüpikvihjetes)

Iga saadud naabri korral kontrollitakse, kas see on seadistatud kommutaator (ehk graafis teine tipp). Kui jah, siis talletatakse andmed, mis on vajalikud graafi serva kuvamiseks (nt portide nimed, ühenduse kiirus). Samas sel juhul ei kasutata naabri kohta saadud andmeid. Teisisõnu – eelistatakse kommutaatori andmeid, mida kommutaator ise enda kohta andis, mitte andmeid, mida temaga ühenduses olev kommutaator tema kohta andis. Kui naaber aga ei ole teine sisendiks saadud kommutaator (vaid kommutaatoriga ühendatud muu seade), siis on tema kohta saadud info meile oluline ja naabri info salvestatakse.

Oluline on, et rakendus oskab kommutaatoritelt graafi koostamiseks andmeid küsida mõlema protokolli (nii CDP kui ka LLDP) kaudu. Kui kommutaator toetab mõlemat, siis lõpuks kuvatakse LLDP andmeid (küsitakse mõlemad, meelde jäetakse LLDP). Kui kommutaator kasutab vaid üht, kasutatakse seda.

6.1.3 Päritavad andmed

Kommutaatorite käest küsitavad andmed võib jagada kaheks: info seadme enese kohta ja info kommutaatoriga ühenduses olevate seadmete (naabrite) kohta. Tabelis 1 tuuakse välja kõik andmed, mida kommutaatoritelt päritakse, sealhulgas ka SNMP identifikaatorid (OID-d) ja MIB-struktuuride nimed (vt 2.2).

Seadmetelt andmete küsimiseks kasutatakse easySNMP (vt 5.2) teegi funktsiooni session.walk()[8]. Funktsioon teeb sisemiselt SNMP GetNextRequest päringu.

6.1.4 Andmete faili salvestamine

Et andmed jõuaksid tagasüsteemilt kasutajaliidesesse, kirjutatakse need Javascripti lähtekoodi faili. Põhjus, miks kasutatakse Javascripti faili (laiendiga `.js`) ja mitte näiteks

[8] `https://easysnmp.readthedocs.io/en/latest/session_api.html`

Tabel 1. Muutujad, mis üle SNMP küsitakse

OID	Muutuja nimi	MIB	Kirjeldus
1.3.6.1.4.1.9.9.23.1.3.4.0	cdpGlobalDeviceId	CISCO-CDP-MIB	Seadme id (nimi). Kasutatakse sisemiselt unikaalse identifikaatorina teiste CDP päringute jaoks
1.3.6.1.4.1.9.9.23.1.2.1.1.6	cdpCacheDeviceId	CISCO-CDP-MIB	Ühendatud seadme nimi
1.3.6.1.4.1.9.9.23.1.2.1.1.4	cdpCacheAddress	CISCO-CDP-MIB	Ühendatud seadme IP-aadress
1.3.6.1.4.1.9.9.23.1.2.1.1.7	cdpCacheDevicePort	CISCO-CDP-MIB	Ühendatud seadme pordi kirjeldus (ühendatud seadme poolel)
1.3.6.1.2.1.2.2.1.2	ifDesc	IF-MIB	Ühenduse kirjeldus (kommutaatori poolel)
1.0.8802.1.1.2.1.3.3.0	lldpLocSysName	LLDP-MIB	Seadme nimi. Kasutatakse sisemiselt unikaalse identifikaatorina teiste LLDP päringute jaoks
1.0.8802.1.1.2.1.3.7.1.3	lldpLocPortId	LLDP-MIB	Ühendatud pordi kirjeldus (kommutaatori poolel)
1.0.8802.1.1.2.1.4.1.1.9	lldpRemSysName	LLDP-MIB	Ühendatud seadme nimi
1.0.8802.1.1.2.1.4.1.1.7	lldpRemPortId	LLDP-MIB	Ühendatud pordi id (ühendatud seadme poolel)
1.0.8802.1.1.2.1.4.1.1.8	lldpRemPortDesc	LLDP-MIB	Ühendatud pordi kirjeldus (ühendatud seadme poolel)
1.0.8802.1.1.2.1.4.2.1.4	lldpRemManAddrIfId	LLDP-MIB	Ühendatud seadme IP-aadress

JSON-formaati, on kirjeldatud peatükis 6.2.3. Andmed itereeritakse ja vormindatakse vastavalt Cytoscape.js teegi sisendformaadile. Andmete faili kirjutamise implementatsioon on failis `src/output_helper.py`.

6.1.5 Veahaldus

Vigadele kõige altim on kommutaatoritelt andmete küsimine – juhul kui kommutaatoriga ei saada ühendust (vale seadistus, võrguprobleem vms). Veahalduse roll on sel juhul tagada, et programmi töö jätkuks teistelt kommutaatoritelt andmete küsimisega. Sel juhul logitakse kasutajale olemasolevate andmetega võimalikult informatiivne viga ja minnakse programmi vooga edasi.

6.1.6 Logimine

Logimiseks kasutatakse Pythoni moodulit logging[9]. Rakenduse sättefailis on võimalik määrata logimise taset (`debug`, kui `debug = true` või `info`, kui `debug = false`). `Debug` tasemel logikirjed on mõeldud vaid arendamiseks ja vigade silumiseks.

6.2 Kasutajaliides

Kasutajaliides on staatiline veebileht. Kasutajaliidese töö on kuvada tagasüsteemi poolt loodud andmed graafina kasutaja brauseris. Kasutajaliidese lähtekood asub `src/web/` kataloogis.

6.2.1 Välised teegid

Graafide kuvamiseks kasutatakse Javascripti teeki cytoscape.js (5.3). Lisaks kasutatakse laiendit cytoscape-popper[10], mis lisab cytoscape.js-le Popper.js[11] (teek muuhulgas hüpikvihjete kuvamiseks) toe. Lisaks võeti kasutusele Tippy.js[12], mis teeb hüpikvihjete loomise ja kohandamise Popper.js-iga lihtsamaks.

Välised teegid asuvad kataloogis `src/web/lib/`. Kõik kasutatud välised failid on lähtekoodiga kaasas (ei laeta internetist). Põhjuseks turvalisus ja võimalus rakendust ilma võrguühenduseta kasutada.

6.2.2 Lokaalse faili laadimine brauseris

Lokaalsel veebilehel lokaalse faili Javascriptiga lugemine on keerulisem kui võiks arvata. Brauserid kaitsevad kasutajaid pahatahtliku koodi laadimise ja jooksutamise eest, aga

[9] `https://docs.python.org/3.6/library/logging.html`
[10] `https://github.com/cytoscape/cytoscape.js-popper`
[11] `https://popper.js.org`
[12] `https://atomiks.github.io/tippyjs/`

samamoodi piiratakse ka pääsu kasutaja lokaalsetele failidele.

Brauserid rakendavad reeglina doomenisisese ressursikasutuse (*same-origin policy*) printsiipi. See tähendab, et veebileht ei saa jooksutada koodi ning ei pääse ligi failidele, mis asuvad teistes domeenides. Et võimaldada ka doomenivälist ressursikaasutust on W3C loonud spetsifikatsiooni CORS (*Cross-origin resource sharing*) [W3C14].

Antud spetsifikatsiooni implementeerivad brauserid ise ning sellele on erinevad arendajad lähenenud erinevalt [Bar08]. Loodud programmi kontekstis tuli esile probleem, et kasutades `file://` URI-skeemi, on faili päritolu (*origin*) definitsioon CORS-i spetsifikatsioonis lahtine [Bar11, peatükk 4, punkt 4]. See tähendab, et brauserid käituvad antud olukorras erinevalt ja ei ole garantiid, et igas brauseris tagasüsteemi poolt loodud faili lugemine õnnestub. Chromium (ja Google Chrome) ei lubanud veebilehel `file://` URI-skeemiga lokaalset faili üldse laadida. Firefox lubaks faili laadida, aga (alates küljendusmootori Gecko versioonist 1.9) seab ette nõuded faili asukohale [FFC].

6.2.3 Tagasüsteemi loodud sisendi lugemine

Töö autor lootis algselt kommutaatorite andmed tagasüsteemilt kasutajaliidesele edastada JSON-formaadis, seejuures kasutada Pythoni ja Javascripti standardteeke. Katsetuste käigus tuli aga välja eelnevalt kirjeldatud brauserite erinev käitumine.

Kasutaja brauserite kohta eeldusi teha ei tahetud – programm peaks töötama sõltumata brauserist. Üks lahendus oleks veebirakenduse kuvamiseks kasutada lokaalset veebiserverit ja küsida andmefail üle `http://` URI-skeemi [13] [14] [15]. See on aga töö raames loodava rakenduse kontekstis üleliigne keerukus ja probleemi lahenduseks ei sobinud.

Selle asemel otsustati andmed tagasüsteemi poolt vormindada Javascripti koodifailiks. Fail defineerib ühe muutuja, milles on kommutaatorite andmed sõnastikus. Seejuures vormindatakse andmed sõnastikus nõnda, et selle saab otse cytoscape.js (vt 5.3) teegile ette anda. See tähendab, et sõnastikus on elemendid `nodes` ja `edges`, vastavalt graafi tipud ja servad. Antud fail laetakse kasutajaliideses Javascripti lähtekoodina ja seejärel on kommutaatorite andmed kasutajaliideses olemas ja neid saab kasutajale kuvada.

6.2.4 Graafi kuvamine

Kuna sisendandmed on juba tagasüsteemi poolt teegile cytoscape.js loetavaks vormindatud, on graafi kuvamiseks vajalik vaid cytoscape.js seadistada. Peamine otsus, mis tuli teha, oli küljenduse valimine[16]. Hea küljendus kasutab kogu ekraanipinna ära (jaotab

[13] `https://stackoverflow.com/questions/46258449/cors-error-requests-are-only-supported-for-protocol-schemes-http-etc`

[14] `https://stackoverflow.com/questions/20041656/xmlhttprequest-cannot-load-file-cross-origin-requests-are-only-supported-for-ht`

[15] `https://stackoverflow.com/questions/35335047/how-to-get-rid-of-cross-origin-request-block-in-chrome`

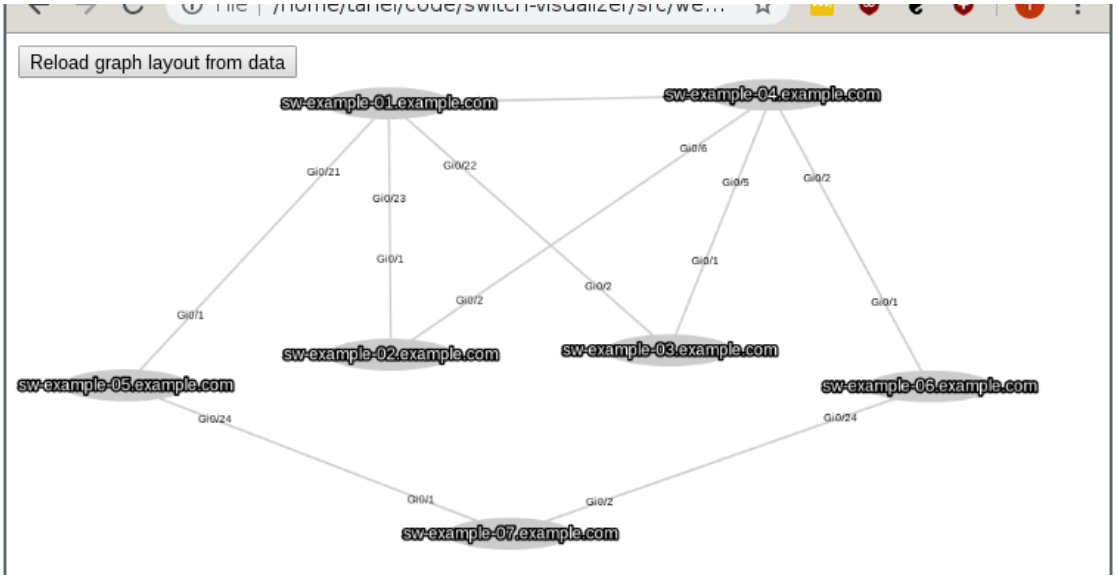
[16] `http://js.cytoscape.org/#layouts`

tipud laiali) ja samuti asetab tipud nõnda, et servad ei kattuks (vt nõue 15). Autori katsetustest tuli välja, et küljendus `breadthfirst` kasutas ekraanipinda kõige mõistlikumalt – see võeti ka kasutusele. Välja tasub tuua ka `concentric` küljendus, mis toimis samuti hästi, ent mitme graafi kuvamisel jäi viimane hätta kogu ekraanipinna ära kasutamisega.

Graafi kujunduses võeti aluseks cytoscape.js-i demo *linkout-example* [17]. Samas lisati tippudes tekstile kontrasti (must taust), et lugemist lihtsustada. Graafi kuvamine on implementeeritud failis `src/web/code.js`.

Graafi tippudeks on kommutaatorid, millelt andmeid küsiti (nõue 13). Servadeks on ühendused kommutaatorite vahel (nõue 14). Seejuures kuvatakse servadel ka mõlema kommutaatori port, mille kaudu nad ühendatud on. Algselt oli plaanis servadel kuvada ühenduste kiirused, kuid otsustati pordi nimede kasuks, sest nimi sisaldab lisaks pordi numbrile infot ka kiiruse kohta (Gi on 1000Mbps, Fa on 100Mbps jne).

Joonisel 2 on toodud kuvatõmmis programmi poolt loodud graafist.



Joonis 2. Näidis rakenduse poolt loodud graafist (nimed muudetud).

6.2.5 Hüpikvihjed

Vastavalt nõuetele 17 ja 18, peavad tippudele vajutades avanema hüpikvihjed. Nagu peatükis 6.2.1 juba kirjeldati, implementeeriti hüpikvihjed teegiga Tippy.js, mis sisemiselt kasutab Popper.js teeki. Popper.js teegi cytoscape.js teegiga integreerimiseks kasutati cytoscape-popper laiendit.

[17] `http://js.cytoscape.org/demos/linkout-example`

Autori roll hüpikvihjete tööle saamiseks oli Tippy.js implementatsioon seadistada. Tippy pakub mitmeid kujundusi[18]. Autor valis läbipaistva kujunduse, et hüpikaknad ei kataks ära hüpikakna all olevaid graafi servasid. Hüpikvihjete avamine ja sulgemine (mõlemad tipule vajutades) implementeeeriti Tippy.js teegi väliselt eraldi, eesmärgiga, et korraga oleks võimalik avada mitu vihjet.

Vihjete sisu koostatakse Javascriptiga, andmed saadakse samast tagasüsteemi poolt loodud sisendfailist, mille põhjal ka graaf tehakse. Hüpikvihjes kuvatakse kõik kommutaatoriga ühendatud seadmed, välja arvatud sätetes määratud kommutaatorid. Hüpikvihjete implementatsioon on samuti failis `src/web/code.js`.

Joonisel 3 on toodud kuvatõmmis programmi poolt loodud graafist.

Joonis 3. Näidis graafi hüpikvihjest (nimed ja IP-aadressid muudetud).

6.2.6 Kasutaja muudatuste salvestamine

Vastavalt nõudele 19 pidi kasutajaliides salvestama kasutaja muudatused graafile (tippude asukohtade muudatused). Antud funktsionaalsust teek cytoscape.js otse ei paku, küll aga defineeritakse funktsioon `cy.json()`[19], mis võimaldab graafi paigutust JSON-formaadis importida ja eksportida.

[18] `https://atomiks.github.io/tippyjs/themes`
[19] `http://js.cytoscape.org/#cy.json`

Implementeeriti graafi asetuse salvestamine, kui kasutaja veebilehelt lahkub või seda värskendab. JSON andmed salvestatakse sõnena brauseri lokaalsesse andmesalvestisse (localStorage[20]). Kui kasutaja veebilehe avab, laetakse ja taastatakse graafi paigutus andmesalvestist. Kui seda varasemalt salvestatud ei ole (esimene külastus), siis laetakse graaf andmefailist. Brauseri lokaalne andmesalvesti on domeenipõhine, seega kui antud rakendut on erinevate domeenide taga, siis need üksteist ei sega. Samuti lisati veebilehele nupp graafi paigutuse taastamiseks – graafi andmefailist uuesti laadimiseks.

6.3 Lähtekood ja litsents

Rakenduse lähtekood on üleval Github keskkonnas: `https://github.com/taneltomson/switch-visualizer`. Autor valis lähtekoodile MIT-litsentsi.

6.4 Rakenduse puudused

Järnevalt toob töö autor valminud rakenduse puudused ja võimalikud edasiarendused.

6.4.1 Tekst graafi servadel

Nagu peatükis 6.2.4 mainiti, kuvatakse graafi servadel mõlema seadme (tipu) lähedal lipikud ühenduse pordi infoga. Seda tehakse teegi cytoscape.js poolt pakutava funktsionaalsusega Lipikud[21]. Teek võimaldab määrata nihke (kauguse tipust), kus lipikut kuvada. Töö autor määras selleks kauguseks 90 pikslit.

Üldjuhul töötavad lipikud nagu eeldada võiks – lipik, mis on tipule lähemal, kirjeldab just selle tipu ühendust. Probleem ilmneb, kui programmi kasutaja veab tipud üksteisele väga lähedale. Kuna lipiku kaugus tipust on fikseeritud (90 pikslit), siis kui tipud on üksteisele lähemal kui ca 180 pikslit, vahetavad lipikud kohad ja jääb mulje, et lipikud lähevad vahetusse (annavad infot teise tipu kohta).

Töö autor proovis probleemi leevendamiseks tuua lipikud tippudele lähemale. Siis tekib aga probleem, et lipikud võivad jääda tipu alla, sest tipus kuvatakse seadme nimi (tipp on ristkülikukujuline, horisontaalselt lai). Lõpuks kasutusele võetud 90 pikslit oli kompromiss, mis tehtud katsetuste puhul kõige paremini töötas.

6.4.2 Andmete uuenemise märkamine

Rakenduse kasutajaliides salvestab muudatused, mida kasutaja graafile teeb (vt 6.2.6). Tegelikult tähendab see ka seda, et salvestatakse tagasüsteemi poolt küsitud andmed ja need laetakse järgmisel külastusel. Seega on võimalik olukord, kus taustal on tagasüsteem kommutaatoritelt uued andmed küsinud, aga kasutajaliides kuvab veel vanu andmeid.

[20] `https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage`
[21] `http://js.cytoscape.org/#style/labels`

Ideaalne lahendus oleks siin see, kui programm oskaks teha tagasüsteemi poolt loodud andmetele sisulist võrdlust. See tähendab, et saaks aru, kui on toimunud sisuline muutus, kus näiteks mõni seade on lisandunud või eemaldatud. Seejuures ignoreerides mittesisulisi muutusi, näiteks kui mõne seadme järjekord loetelus on vahetunud.

Veidi lihtsam lahendus oleks, kui kasutajaliides salvestaks koos kasutaja muudatustega ka andmefailist (mille põhjal graaf loodi) arvutatud räsi. Antud räsi saaks igal veebilehe avamisel (kui laetakse kasutaja poolt muudetud graaf) võrrelda andmefaili räsiga ning kui need on muutunud, kasutajale teade kuvada.

6.4.3 Ilma veebiserverita mitme rakenduse instanssi kasutamine

Andmete brauseri lokaalsesse andmesalvestisse salvestamisega on ka teine probleem. Nagu peatükis 6.2.6 juba mainiti, on brauseri andmesalvesti domeenipõhine.

Samas vähemalt Chromium arvestab kõik failid, mis on avatud üle `file://` URI-skeemi ühte domeeni kuuluvaks. See tähendab, et ilma veebiserverita saab korraga kasutada vaid üht rakenduse instanssi, või täpsemalt vaadata vaid üht tagasüsteemi poolt loodud graafi korraga.

Probleemi saaks lahendada, kui lisada (juhul, kui kasutajaliidest vaadatakse üle `file://` URI-skeemi) ka märge faili asukoha kohta ja kasutajaliidese avamisel kontrollida, et varasemalt salvestatud andmed on salvestatud sama rakenduse instanssi poolt.

7 Kokkuvõte

Antud bakalaureusetöö raames kirjutas autor rakenduse, mis abistab võrgutopoloogia visualiseerimisel. Rakendus on ennekõike mõeldud võrguadministraatoritele oma kontrolli all olevate seadmete (peamiselt võrgukommutaatorite) topoloogia vaatamimseks ja probleemide avastamiseks. Selleks küsib rakendus sisendiks antud seadmetelt üle SNMP andmed seadme enese ja tema naabrite kohta, töötleb vastused ja koostab tulemustest veebilehe graafiga.

Kuigi rakendus loodi vaid kommutaatorite topoloogiat silmas pidades, ei pea programmi kasutamisel piirduma rangelt 2. kihi kohtvõrguga. Ka ruuterid ja tulemüürid oskavad üle SNMP andmeid jagada, nii et teoreetiliselt on võimalik programmiga luua graafe ka üle mitme kohtvõrgu.

Rakendust testiti Cybernetica AS kohtvõrgus. Arenduse käigus kasutas töö autor programmi testimiseks piiratud arvu võrguseadmeid. Arendusprotsessi lõpus testis programmi ka võrguadministraator suurema hulga võrguseadmetega ja rakendus täitis seatud eesmärgi.

Rakenduse edasiseks täiendamiseks on mitmeid võimalusi. Kasutusmugavus paraneks, kui rakendus saaks ise aru, kui taustal on andmed muutunud (seadmete topoloogia muutus võrreldes viimase seisuga) ja annaks sellest kasuatajaliideses märku. Samuti saaks lisada rohkem andmeid, mida programm seadmete kohta küsib ja kuvab.

Viidatud kirjandus

[Bar08] Adam Barth. Security in depth: Local web pages. Chromium Blog, https://blog.chromium.org/2008/12/security-in-depth-local-web-pages.html, 2008.

[Bar11] Adam Barth. The web origin concept. https://tools.ietf.org/html/rfc6454, 2011.

[Bha15a] Deben Bhattarai. Cisco discovery protocol (cdp). The Cisco Learning Network, 2015. https://learningnetwork.cisco.com/docs/DOC-26872.

[Bha15b] Deben Bhattarai. Link layer discovery protocol (lldp). The Cisco Learning Network, 2015. https://learningnetwork.cisco.com/docs/DOC-26851.

[Cyt] Cytoscape.js. http://js.cytoscape.org/#introduction. Vaadatud: 25.03.2019.

[Eas] Easy snmp documentation. https://easysnmp.readthedocs.io. Vaadatud: 25.03.2019.

[FFC] Same-origin policy for file: Uris. https://developer.mozilla.org/en-US/docs/Archive/Misc_top_level/Same-origin_policy_for_file:_URIs. Vaadatud: 26.03.2019.

[Gor] Oliver Gorwits. App::netdisco - an open source web-based network management tool. https://metacpan.org/pod/App::Netdisco. Vaadatud: 09.05.2019.

[Laa08] Erkki Laaneoks. *Sissejuhatus võrgutehnoloogiasse*. Tartu Ülikooli Kirjastus, 2008.

[W3C14] Cross-origin resource sharing. W3C Recommendation, https://www.w3.org/TR/cors/, 2014.

[Ytt] Github Kasutaja Ytti. Netcrawl - lldp/cdp crawler. https://github.com/ytti/netcrawl. Vaadatud: 08.04.2019.

Lisad

I. Lähtekood

Programmi lähtekood ja kasutusjuhend on saadaval aadressil: `https://github.com/taneltomson/switch-visualizer`

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Tanel Tomson**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Võrgutopoloogia visualiseerimine,

   mille juhendaja on Meelis Roos,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Tanel Tomson
14.05.2019