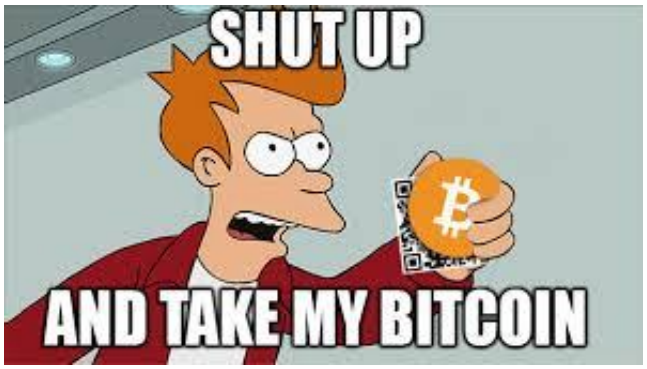
Vorgurakendused 2 prax 1 2019 kevad

Allikas: Lambda



Sisukord

Ülesanne

Sul on võimalik valida kahe erineva ülesande vahel:

- Lihtsa P2P võrgu realiseerimine ja sellele distributed ledgeri mõne aluskomponendi (mitte veel terve ledgeri) realiseerimine.
- Lihtsa P2P võrgu realiseerimine ja sellel anonümiseeriva võrgu (mikro-TOR) realiseerimine.

Kummalgi juhul on sul alustuseks vaja implementeerida lihtne P2P protokoll üle HTTP, nii et:

- protokoll oleks realiseeritud HTTP protokolli peale (vaja implementeerida ka HTTP server ja klient)

- mitu sinu rakenduse klooni suudavad suhelda nii samas masinas, töötades eri portide peal, kui mitmes erinevas masinas eri IP-de peal.
- eesmärk on kõigi rakendused omavahel suhtlema ja ühise võrguna tegutsema panna

Programmeerimiskeel on vabalt valitav. Masin(ad), kus su süsteem töötab, on vabalt valitavad: võid kasutada oma laptoppi, arvutiklassi arvutit, dijkstrat või mõnda muud serverit.

NB! Siin praksis ei või kasutada nö suurt veebiserverit a la Apache, Nginx vms. Lihtsad http teegid on samas OK. Ei soovitaks kasutada ka veebiraamistikke a la Spring vms: need ei ole esimese praksi jaoks mõttekad ning nende jaoks on eraldi 2. praksi variant.

Praktikumi lõpus pead tegema juhendajale oma grupiga väikese demo - et kuidas töötab - ja seletama natuke oma koodi ehitust.

Jooksvalt praktikumide käigus täienev nimekiri soovitustest kuidas rakendust samm-sammult ehitada asub siin: https://github.com/martinve/pyp2p Näidisrakendus on kirjutatud Pythonis aga tegevuste nimekiri on universaale sõltumata platvormist.

Kuidas alustada

Sul on kõigepealt vaja teha väike P2P rakendus, ehk proge, mis suudaks oma kloonidega suhelda. Konkreetsemalt:

- sinu proge on käsurea rakendus ilma kasutajaliideseta
- ta kuulab, kas temaga tahetakse võtta ühendust ja võtab ise teistega ühendust
- tahame saavutada, et meil käib korraga N koopiat, eri masinates, ja nad süngivad omavahel datat (ledgeri puhul) või saadavad päringuid teistele laiali (tori puhul).
- "teiste kuulamine tähendab", et ta on server
- eeldame, et ühendus toimub üle http

Kõigepealt tee nii, et sinu proge reageeriks http päringutele ja vastaks neile mingi testvastuse, et sa saaks ise aru, et ta on värgi kätte saanud.

Järgmine asi on panna ta ise teistega ühendust võtma: siis ta avab urli nagu brauser.

Küsimus, et kuidas päringuid kodeerida.

võtame GET näite urliga

```
http://1.2.3.4:4500/request?param1=val1&param2=val2
```

siin http://1.2.3.4:4500/ ütleb, mis masinaga (1.2.3.4) ühendust võtta mis pordilt (4500) ja see

```
request?param1=val1&param2=val2
```

on string, mis rakendus kätte saab ja ise parsima peab, ja mida võib kodeerida ehk ka näiteks nii

```
/val1/val2/
```

POST puhul see osa

```
request?param1=val1&param2=val2
```

on mittevajalik ja parem oleks teha ainult

```
http://1.2.3.4:4500/
```

ja saata sisu jsoni tekstina a la

```
{"param1":val1, "param2":val2}
```

kuigi saab ka saata postiga lihtsalt

```
param1=val1&param2=val2
```

mis on veidi vähem mainstream.

Mis teeb käsurea rakendus? Panen ta käima, ta jääbki käima ja trükib vajadusel debuginfot. Tal peab olema pordikuulamise tükk sees ja talle peab kuskil ütlema, et mis porti kuulata. Tal võiks olla default port a la 5134 ja saad käivitades öelda pordi a la ühes aknas

```
prog.py 6345
```

teises aknas aga

```
prog.py 7890
```

NB! Iga kloon samas masinas peaks olema eri pordil.

Näidisrakendus

Näidisrakenduse implementatsioon Pythonis on leitav Githubist: https://github.com/martinve/pyp2p

HTTP server/klient

HTTP baasil töötav protokoll näeb välja umbes järgmine:

```
GET http://1.2.3.4/request?param1=val1&param2=val
```

```
POST http://1.2.3.5/response
...
param1=val1&param3=val3
```

jne. Seega tuleb meil teha:

1. HTTP server, mis suudaks vastu võtta GET ja POST päringut ning parseerida pathi ja parameetrid (query stringi). Meid huvitab ka IP, kust päring tuli, see tuleb edaspidi kasutamiseks meelde jätta. POST päringu puhul peab lugema päringu "keha" (body).
2. Selle peale peaks serveris käivituma mingi funktsioon. Selle võime jätta hetkel tühjaks
3. Lõpuks saadame vastuse samuti HTTP päringuga. Seega vaja selleks HTTP klienti.

Server tuleb realiseerida ise, kasutades võrgust leitavaid näiteid mikro-http-serveri jaoks (tüüpiliselt paar lehekülge koodi). Selliseid näitekoode võib täiesti otse kasutada.

C jaoks on sobiv mikroserver näiteks tiny (http://tinyhttpd.cvs.sourceforge.net/viewvc/tinyhttpd/tinyhttpd/httpd.c?revision=1.4&view=markup), java jaoks sobib alustuseks sisseehitatud HTTP server (http://stackoverflow.com/questions/3732109/simple-http-server-in-java-using-only-java-se-api). Socketi tasemel alternatiividest on see server (http://www.java2s.com/Code/Java/Tiny-Application/HttpServer.htm) väga hea väike näide, aga selgitava tekstita, ning selgitavate tekstidega servereid leiad mh siit (http://onjava.com/pub/a/onjava/2003/04/23/java_webserver.html) ja siit (http://fragments.turtlemeat.com/javawebserver.php).Loe java socketitest põhjalikumalt siit (http://download.oracle.com/javase/tutorial/networking/sockets/). Pythoni serverinäite saad siit (http://muharem.wordpress.com/2007/05/29/roll-your-own-server-in-50-lines-of-code/); lühem ja uuem (https://daanlenaerts.com/blog/2015/06/03/create-a-simple-http-server-with-python-3/). Http kohta võib lugeda näiteks siit (http://www.jmarshall.com/easy/http/).

Rakenduse variant 1: distributed ledgeri komponent

Sul on vaja kirjutada lihtne bitcoini-tüüpi rakendus, mis suudab leida oma kloone ehk sõlmi võrgus, küsida neilt olemasolevaid ledgeri blokke ja saata neile endale teadaolevaid blokke ja uusi transaktsioone (transaktsioonid on ülekanded ehk mistahes info, millest blokk kokku pannakse).

"Ledgeri blokk" võib selles praksis olla lihtsalt suvaline string või json struktuur: neid ei pea kontrollima, sünkima ega kaevandama. Piisab lihtsalt nende kogumisest ja soovijatele laialisaatmisest.

Sa võid ise valida, kas

- realiseerid lihtsustatud protokolli, millega saad ise distributed ledgeri ehitada, ei saa aga suhelda päris-bitcoini sõlmedega.
- või realiseerid alamhulga tegelikust bitcoini protokollist: see on tehniliselt päris keeruline (protokoll on bititasemel ja seal on palju detaile), samas saad siis ise leida päris-bitcoini võrgusõlmi ja katsetada, mis juhtub, kui neile näiteks ise ülekandeid vms saadad. Kui seda kaalud, siis loe enne bitcoini materjale kursuse põhilehel ja tutvu hardcore dokumentatsiooniga algul siit https://bitcoin.org/en/developer-guide#p2p-network ja siis siit https://en.bitcoin.it/wiki/Protocol_documentation

Nüüd detailsemalt meie oma lihtsustatud protokollist. Edaspidises eeldame, et http päringud on GET päringud, kui mõne puhul pole spetsiaalselt öeldud, et tegu on POST päringuga. Pane tähele, et sul on vaja realiseerida nii kloonide/blokkide küsimine kui nende küsimisele vastamine!

Kloonide ehk sõlmede leidmine võrgus

Klooni leidmine tähendab, et saad teada ip aadressi ja pordi, kus võiks olla töötav kloon. Sinu rakendusel peaks kõigepealt olema väike konfifail, kuhu saad kirjutada järjest mõned ip aadressid ja pordid: soovitav on see esitada teha jsoni listina. Käivitades hakkab sinu rakendus neid läbi käima ja küsima neilt, milliseid teisi kloone nad teavad. Nii kogud kokku pikema listi kloonidest. Samuti pead suutma vastata endale teadaolevate kloonide päringule.

Soovitav on realiseerida päring http protokolliga, näiteks nii:

```
http://xx.xx.xx.xx:yyyy/addr
```

millele vastatakse kloonide json listiga, samas formaadis, kui sinu konfifailis.

Reaalse töö käigus ei ole sul vaja leida kõiki töötavaid kloone, piisab mingist "mõistlikust" hulgast.

Hea mõte on taustaks läbi lugeda need lühikesed jutud kursuse põhilehelt (arusaadavalt ei pea sa neid meetodeid realiseerima):

- https://bitcoin.stackexchange.com/questions/3536/how-do-bitcoin-clients-find-each-other
- https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery

Ledgeri blokkide küsimine

Sul on vaja regulaarselt küsida ja kettal hoida nö ametliku ledgeri kõiki blokke. Ledger on lihtsalt blokkide list. Igal blokil on oma tekstiline id (bitcoini puhul bloki hash) ja sisu. Bloki hash tuleb arvutada bloki tekstilise esituse pealt sha-256 hashifunktsiooniga, ning konvertida hex-kujule. Hashe saadame siis alati hex-kujul.

Nagu öeldud, ei pea me esimeses praksis blokkide sisuga ja nende kontrollimisega tegelema, selleks on teine praks.

Kuna sul võib juba olla mingi hulk ledgeri blokke salvestatud, siis on kasulik saada küsida ka blokke alates mingist teadaolevast blokist (ehk tema hashist).

Blokkide järjestuse võid teha esialgu lihtsalt nii, et hoiad blokke saabumise järjekorras. Lõplikus ledgeri variandis peaks blokid olema hashchaini järjestuses (blokk N+1 hash moodustatakse bloki sisust+eelmise bloki hashist, vaata seda pilti (https://blog.ethereum.org/wp-content/uploads/2015/11/mining.jpg), kus on näha bloki sees olev hashide merkle puu ja blokkide järjestus lihtsa hashiahelana), aga seda hashchaini ja kontrolli ei ole esialgu vaja teha.

Blokkide küsimine peaks välja nägema nii, et saad

- küsida korraga blokkide nimistut ehk hashide listi (kas tervet või ainult uuemaid)

- küsida konkreetseid blokke vastavalt nende hashidele.

Kõigi blokkide nimistu küsimine võiks välja näha nii:

```
http://xx.xx.xx.xx:yyyy/getblocks
```

ja alates-blokist-Z nii:

```
http://xx.xx.xx.xx:yyyy/getblocks/Z
```

kus Z on bloki id ehk hash.

Vastuseks võiksid saada bloki hashide listi.

Ühe konkreetse bloki küsimine võiks välja näha nii:

```
http://xx.xx.xx.xx:yyyy/getdata/H
```

kus H on bloki hash, millele vastavat blokki tahad.

Uute transaktsioonide ja uute ledgeri blokkide laialisaatmine

Siin toimub tehingute ja blokkide laviinitaoline laialisaatmine kõigile kloonidele: iga kloon salvestab talle saadetud info ja saadab selle siis teistele edasi.

Sul olema võimalik omal initsiatiivil laiali saata transaktsioone ehk ülekandeid (a la Jaan kannab 2018-02-15 Antsule 0.0001 bitcoini).

See päring võiks olla POST päring, kus url on

```
http://xx.xx.xx.xx:yyyy/inv
```

ning postitatud väärtus (st tekst peale http päiste järel olevat tühja rida) on on transaktsiooni hash ja tema json-esitus. Transaktsiooni sisu võiks esialgu olla suvaline (json) string. Sellise postitatud päringu vastuseks võiks anda kas arvu 1 kui transaktsioon aktsepteeriti või veateate (näiteks {"errcode": ..., "errmsg": ...} kui teda ei aktsepteeritud.

Kättesaadud transaktsioon tuleks salvestada ja omakorda teistele kloonidele laiali saata. Pane tähele, et kui sa oled mingi hashiga transaktsiooni saad, siis peaksid vaatama, kas ta sul juba on, ja kui on, siis mitte edasi saatma.

Samuti peab sul olema võimalik saata teistele kloonidele omal initsiatiivil uusi blokke, mis nemad peaks oma ahela lõppu lisama, kontrolle pole esialgu vaja (teises praksis hakkame tegelema blokkide kontrolli ja "ametliku" ahela sünkimisega. Blokk koosneb üldjuhul hulgast kontrollitud transaktsioonidest.

See päring võiks olla POST päring, kus url on

```
http://xx.xx.xx.xx:yyyy/block
```

ning postitatud väärtus (st tekst peale http päiste järel olevat tühja rida) on bloki hash ja tema tekstiline sisu (vali ise, kuidas neid kodeerid). Sellise postitatud päringu vastuseks võiks anda kas arvu 1 kui blokk aktsepteeriti või veateate (näiteks {"errcode": ..., "errmsg": ...} kui teda ei aktsepteeritud.

Jällegi, kättesaadud blokk tuleks salvestada ja omakorda teistele kloonidele laiali saata. Pane tähele, et kui sa oled mingi hashiga bloki saad, siis peaksid vaatama, kas ta sul juba on, ja kui on, siis mitte edasi saatma.

Rakenduse variant 2: anonümiseeriv võrk

Eesmärgiks on ehitada anonümiseeriv võrk nagu Tor (https://www.torproject.org/). Täpsemalt on eeskujuks praeguseks mitteaktiivne anonüümne failijagamise võrk MUTE Network (http://mute-net.sourceforge.net/). Sealt võetud ideid kasutades teeme võrgu, kus saab anonüümselt veebist faile (html vms) alla laadida.

Meie võrgus teab iga võrgusõlm (node) näiteks kolme naabrit. Kui sõlm A tahab alla laadida URL-i http://google.ee, saadab ta päringu kõigile oma naabritele. Need omakorda saadavad päringu oma naabritele jne. Päringul on kaasas salajane identifikaator - ainult A teab, et see identifikaator seostub temaga. Lõpuks jõutakse võrgusõlmeni B, kes otsustab, et laadib ise URL-i sisu alla. B ei tea kes URL-i sisu soovis, aga saadab vastuse suunas, kust ta päringu sai. Samal viisil liigub vastus mööda võrku tagasi, kuni jõuab A-ni, kes tunneb ära oma salajase identifikaatori.

Võrku liituda ja sealt lahkuda võib suvalisel ajal. Seetõttu tuleb liitudes ja edasi perioodiliselt värsket naabrite nimekirja küsida, selleks paigutatakse samasse võrgu eraldi kataloogiserver.

Protokoll

Port. Vaikimisi on eeldatud, et võrgusõlmed kasutavad sõnumite vastuvõtmiseks porti 1215. Kasuta seda, kui suhtled teiste üliõpilaste programmidega ja kataloogiserveriga virtuaallaboris. Iseseisvaks või kohalikuks testimiseks võib kasutada ka muid porte.

Päringu saamise kinnitus. Peale iga HTTP päringu saamist vastab võrgusõlm tekstiga `"OK"` juhul kui ei olnud viga ning tekstiga `"Error: xxxxxxxxxxx"` kui oli viga (*xxxxxxxxxxx* tähistab vabas vormis vea kirjeldust ning selle võib ka ära jätta). Vastuse saatmine on kohustuslik, isegi kui võrgusõlm päringuga tulnud sõnumit ignoreerib. Korrektse sõnumi puhul peab vastuse HTTP staatus olema 200, muul juhul on lubatud ka HTTP veakoodide esinemine.

Naabrite nimekiri. Iga minuti tagant tuleb naabrite nimekirja kataloogiserverist uuendada (vt allpool).

URL-i (faili) alla laadimise päring

nimi (path)	/download
parameetrid	?id=*xxxxxxxx*&url=*yyyyyyy*
tüüp	HTTP GET
body	

xxxxxxxx on selle sõnumi jaoks genereeritud juhuslik number. Kui sama võrgusõlm näeb hiljem sama id-ga `file` sõnumit, siis see sisaldab soovitud vastust. *yyyyyyy* on täispikk URL, mis peab olema turvaliselt kodeeritud (https://en.wikipedia.org/wiki/Percent-encoding), Pythoni lahendus (https://docs.python.org/3.5/library/urllib.parse.html#url-quoting).

Näide: http://1.2.3.4:1215/download?id=7652354352&url=http%3A//google.ee/%3Fs%3Dbla%26lang%3Den

Käsitlemise reeglid:

1. Kui sõlm algatab ise selle päringu, peab ta meelde jätma `id`, et hiljem vastust ära tunda
2. Päringu algatav võrgusõlm saadab sõnumi kõigile hetkel teadaolevatele naabritele
3. Esimest korda selle sõnumi vastu võtmisel otsustab võrgusõlm, kas faili alla laadida (tõenäosus *x%*, konfitav)
    1. Kui faili alla ei laadita, saadab võrgusõlm sõnumi edasi kõigile oma naabritele, v.a juhul kui naabri IP langeb kokku sõnumi edastanud sõlme IP-ga
    2. Kui fail alla laaditakse, siis algatatakse `file` tüüpi sõnum (vt. allpool)
    3. Mõlemal juhul tuleb meelde jätta sõnumi `id` ja edastanud võrgusõlme IP
4. Teist korda sama (või enda algatatud) `id`-ga sõnumi vastu võtmise korral seda ignoreeritakse
5. Päringu algatanud sõlm peab arvestama võimalusega, et võrgu ebastabiilse iseloomu tõttu ei ole vastuse saabumine garanteeritud

Faili sisu tagastamine

nimi (path)	/file
parameetrid	?id=*xxxxxxxx*
tüüp	HTTP POST
body	{"status":yyy, "mime-type":"zzzzzzzz", "content":"..."}

xxxxxxxx on sama, mis `download` sõnumis, mille saamise tõttu fail alla laaditi. HTTP päringu kehas on JSON formaadis objekt, mis sisaldab *base64* kodeeringus faili ja selle metadatat. *yyy* on faili allalaadimisel saadud HTTP staatus. *zzzzzzzz* on faili sisu MIME tüüp (peale dekodeerimist). "content" väljal on RFC-3548 ühilduvas *base64* kodeeringus (Pythoni tugi (https://docs.python.org/3/library/base64.html)) faili sisu.

URI näide: http://5.6.7.8:1215/file?id=7652354352

Keha näide 1:

```
{"status":200, "mime-type":"text/html", "content":
"PCFET0NUWVBFIGh0bWwgUFVCTElDICItLy9XM0MvL0RURCBYSFRNTC..."}
```

Keha näide 2 (viga):

```
{"status":404}
```

Käsitlemise reeglid:

1. Päringu algatav võrgusõlm saadab sõnumi naabrile, kellelt sama `id`-ga `download` sõnumi vastu võttis
2. Kui eelmine samm ebaõnnestus, saadetakse sõnum kõigile naabritele
3. Esimest korda selle sõnumi vastu võtmisel otsustab võrgusõlm, kas fail oli tema enda soovitud (`id` parameeter)
    1. Kui ei olnud, saadab võrgusõlm sõnumi edasi naabrile, kellelt saabus sama `id`-ga `download` päring
    2. Kui eelmine samm ebaõnnestus (või sellist naabrit ei teata), saadetakse sõnum edasi kõigile naabritele, v.a. juhul kui naabri IP langeb kokku käsitletava `file` sõnumi saatja IP-ga
    3. Kui fail oli sõlme enda soovitud, parseeritakse päringu kehast staatus ja olemasolul faili sisu.
4. Teist korda sama (või enda algatatud) `id`-ga sõnumi vastu võtmise korral seda ignoreeritakse

Millal faili alla laadida?

Oletame, et võrgus on 10 arvutit. Kui allalaadimispäring laiali saadetakse, peaks üks neist hakkama faili alla laadima, ülejäänud 9-l pole seda vaja teha. Kuidas aga otsustada, kelle töö on fail alla laadida? Korrektne viis seda teha oleks DHT (https://en.wikipedia.org/wiki/Distributed_hash_table) abil, mis aga mitmekordistaks protokolli keerukust. Kui me aga loobume korrektsusest lihtsuse huvides, saab asja teha ka täiesti suhtlusvabalt.

Igal võrgusõlmel on oma "laiskuse mõõt", näiteks antud juhul number 0.9. Kui võrgusõlm näeb päringut, mille peale tuleks algatada faili alla laadimine, leiab ta juhusliku numbri vahemikus 0 kuni 1. Kui see on suurem kui tema laiskus, siis tuleb hakata faili alla laadima. Kui aga laiskus on suurem, siis saadetakse sõnum edasi teistele. Antud näite puhul on kõige tõenäolisem, et üks võrgusõlm hakkab faili alla laadima, ehkki võib ka juhtuda, et neid on mitu või mitte ühtegi.

Kataloogiserver

Kataloogiserveriga suhtlemine toimub lihtsa klient-serveri mudeli abil, seal pole P2P protokolli tarvis. Naabrite küsimine:

```
http://192.168.3.11:1215/getpeers
```

Vastus:

```
[ "1.2.3.4:1215", "5.6.7.8:1215", "1.2.3.6:1215" ]
```

Vastus on JSON formaadis. Naabreid võib olla ka rohkem või vähem kui 3 (näiteks juhul, kui keegi teine pole võrguga liitunud).

Soovitused

Konfiguratsioon. Formaat pole oluline, aga mõtekas on järgmised parameetrid konfigureeritavaks teha. Pordi võib teha ka käsureaparameetriks:

```
; lokaalse testimise korral vähenda, suures võrgus suurenda
laziness = 0.5
; kataloogiteenus, siia võid panna ka staatilise tekstifaili URL-i
```

```
directory = http://192.168.3.249:1215/getpeers
; lokaalseks mitme instantsiga testimiseks
port = 1215
```

Naabrite tabel. Täidetakse kataloogiserverit kasutades.

nr	ip	port	elus
1	1.2.3.4	1215	Y
2	5.6.7.8	1215	Y
3	1.2.3.6	1215	N
4			

Väli `elus` võimaldab meelde jätta, kui naaber võrgust ära kaob. Seda saab asendada ka ajatempliga, aga jah/ei lipu käsitlemine on lihtsam.

Minu päringute tabel. Kuna igal päringul on uus `id`, tuleb mitme päringu tegemisel ajalugu meelde jätta.

nr	id	url	ajatempel
1	7652354352	http://google.ee/?s=bla&lang=en	Sat Sep 17 02:16:57 2016
2			

Väli `url` on kasulik näiteks veateadete edastamiseks. `ajatempel` abil saab tuvastada, kui kaua päring on vastust oodanud ning vajadusel päringut korrata - sel juhul tuleb kindlasti uus identifikaator genereerida - või anda veateade.

Marsruutimistabel. Kui ülejäänud tabelid on "maitse küsimus", siis protokollis ette nähtud sõnumite käsitlusreeglite järgmiseks on järgnev marsruutimistabeli formaat oluliseks lihtsustavaks abivahendiks.

nr	id	download ip	file ip
1	7652354352	1.2.3.4	
2	7657856788		5.6.7.8
3	9845345233	1.2.3.6	5.6.7.8
4			

`download ip` on naabri IP, kes saatis meile vastava identifikaatoriga `download` sõnumi. Juhul kui näeme sama identifikaatoriga `file` sõnumit, tuleks meil just `download ip` aadressile see edastada. Mõeldav on ka juhtum, kus meil tuleb sisse `file` sõnum, millele vastavat `download` sõnumit me ei ole näinud. See tuleb aga ikkagi marsruutimistabelisse registreerida, et korduvat sõnumite laialisaatmist vältida (rida 2). `file` sõnumi saatja IP läheb väljale `file ip`. Juhul kui mõlemad sõnumid on nähtud, siis on mõlemad IP-d täidetud (rida 3) ning igasugusel edasisel kokkupuutel antud identifikaatoriga pole enam tarvis reageerida.

Seda tabelit saab täiendada ka ajatempliga, et vanemaid kirjeid kustutama hakata. Väikeste testimismahtude juures pole see veel vajalik.

Juhtimine Et testida, on minimaalselt vaja rakendusele ka alla laaditavaid URL-e ette sööta. Selleks on mõeldavad järgmised viisid:

- lähtekoodi sisse kirjutatud, lihtne teha aga raske hallata
- loetakse tekstifailist rakenduse käivitamisel
- antakse rakendusele käsurea argumendiks
- reaalajas üle HTTP (võimaldab näiteks AJAX-iga veebiliides lisada)
- reaalajas käsurealiidese (CLI) abil
- ...

Testimiseks on soovitatav esialgu lihtsam variant võtta ning hiljem kui rakendus juba suuremalt osalt töötab, leida viis, mis võimaldab paindlikumalt ja massilisemalt testida.

Praktikumi kirjeldus täieneb jooksvalt semestri käigus

Pärit leheküljelt "http://lambda.ee/w/index.php?title=Vorgurakendused_2_prax_1_2019_kevad&oldid=15315"
