TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Oliver Tiit 134401IAPB

ANDMETE SÜNKRONISEERIMINE EBAKINDLATES VÕRKUDES

Bakalaureusetöö

Juhendaja: Juhan-Peep Ernits
Doktorikraad

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Oliver Tiit

20.05.2019

Annotatsioon

Töös kirjeldatakse andmete sünkroniseerimise protokolli ideed, mis ei vaja paketikao tõttu kaotatud pakettide kohest uuesti saatmist ega pakettide kohalejõudmist kontrollivat mehhanismi. Kaduma läinud pakettide uuesti saatmise probleem on lahendatud Bloomi filtriga, mille saadab välja andmeid küsiv seade. Kui võrgu topoloogia ei ole fikseeritud (nt. raadiovõrk), siis saame osa informatsiooni ka teiste võrgusõlmede omavahelisest suhtlusest salvestada. Protokolli ükski andmepakett ei sõltu eelnevatest, seetõttu ei ole paketikadu, ükskõik kui suur (v.a täielik paketikadu), liiga suureks probleemiks. Suurema paketikao juures on täieliku sünkroniseerimise aeg pikem, et säästa andmeside hulka. Andmed jõuavad kohale igal juhul eeldusel, et uusi andmeid ei toodeta rohkem, kui on võrgu efektiivne läbilaskevõime, mis oleks sama suureks probleemiks iga sünkroniseerimisprotokolli jaoks.

Protokoll peab töötama hästi hajusvõrkudes, kus info delegeeritakse seadmetele edasi nii, et lõpuks on igal seadmel kõik andmed olemas ja seejuures on kasutatud võimalikult vähe võrguliiklust.

Bakalaureusetöö on kirjutatud eesti keeles ning sisaldab teksti 17 leheküljel, 6 peatükki, 4 joonist, 4 tabelit ja 2 koodinäidet.

Abstract

Data Synchronization in Unreliable Networks

This work describes an idea of protocol that is efficient in synchronizing data over unstable networks and does not require immediate packet acknowledgement. Problem of finding missing packets to send to receiver is solved by bloom filter that is broadcasted by devices time to time. If network is capable of point-to-multipoint (radio networks), then protocol is capable of receiving and processing information not meant for that node specifically. As no packet is dependent of the previous, high packet loss is not so much of a problem. In case of higher packet loss, length of time for full synchronization is sacrificed for lower data usage. In the end, all data will be synchronized, unless packets produced is higher than network capability to handle them.

Protocol works well in point-to-multipoint distributed networks, for example radio networks while using bandwidth as efficiently as possible.

The thesis is in Estonian and contains 17 pages of text, 6 chapters, 4 figures, 4 tables and 2 code examples.

Lühendite ja mõistete sõnastik

IP	Interneti Protokoll (Internet Protocol)
TCP	Edastusohje protokoll (Transmission Control Protocol)
UDP	Kasutajadiagrammi protokoll (User Datagram Protocol)
RX	Infot vastuvõttev juhe (Receive)
TX	Infot saatev juhe (Transmit)
USB	Universaalne järjestiksiin (Universal Serial Bus)
ACK	Kinnitav vastus (Acknowledgement)

Sisukord

Jooniste loetelu

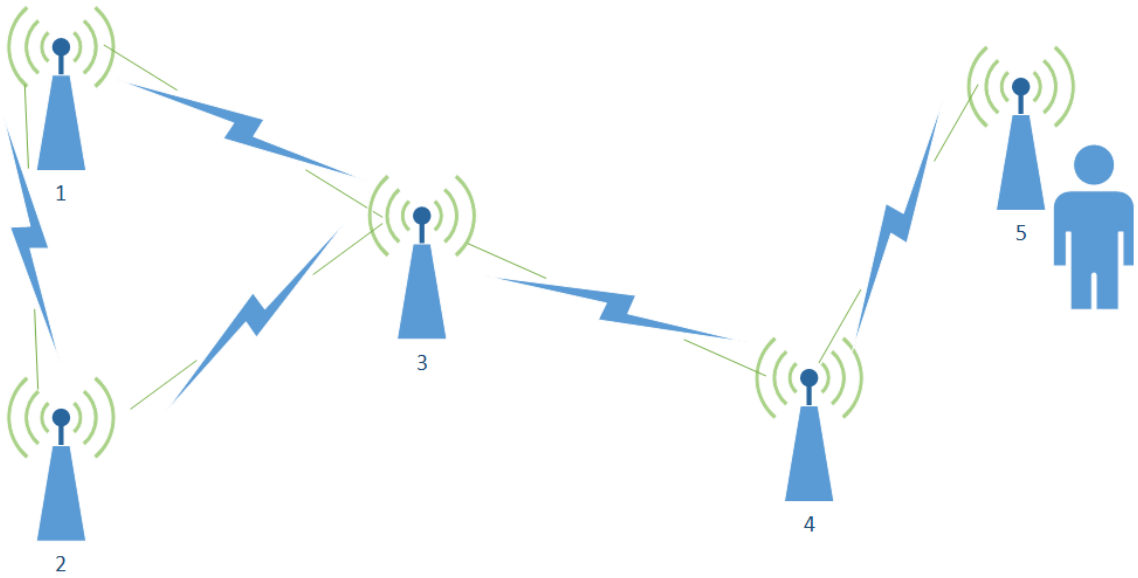
Tabelite loetelu

Koodinäidete loetelu

1 Sissejuhatus

Andmeid seadmete vahel sünkroniseerimiseks tuleb võrrelda, millised andmed on seadmetel juba olemas ja siis puuduolevad vahetada. Kui ühendus seadmete vahel on kehv, tekib probleem, kus paketid võivad kaduma minna ja sünkroniseerimisprotsess võib kasutada ebapraktiliselt palju andmemahtu või hoopis katki minna. Selline olukord tekib juhul, kui andmeside paketikadu kontrollivad protokollid (nt. TCP) ei ole praktilised (nt. raadiovõrgus).

Näiteks võib tuua metsas puude külge kinnitatavad, väikese võimsusega mikrokontrolleriga ja raadioga andurid, mille tööks on erinevate mõõtmiste tegemine ja sündmuse logimine (näiteks hõiked, lindude/loomade hääled, mootorsae heli), kus lõpuks on oluline info transportida kuhugi kesksesse kohta. Ühe raadio ulatus ei pruugi andmelattu ulatuda, samas võiksid raadiod infot edasi anda üle mitme raadio (joonis 1). Kuna raadiod peavad töötama võimalikult väikese energiaga, ei ole praktiline neid marsruutima panna, ega kasutada muud keerulist lahendust.



Joonis 1 Näidisraadiovõrk

Mikko Eronen on lahendanud sarnast probleemi kaevanduses [1]. Tema probleem seisneb selles, et kavandusmasinad toodavad infot kasutuse kohta ja need andmed tuleb edastada

tootjale. Kuna traadita internet maa all ei levi, oli tema lahenduseks tuua maa alt info USB andmekandjal maa peale ja siis laadida võrguga ühendatud masinasse, mis saadab andmed tootjale. Tegemist on lahendusega, kus inimene peab vaeva nägema, et tootjale tagasisidet anda. Oleks parem, kui maa-alused masinad suudaksid näiteks sünkroniseerida omavahel, kui ka tööliste kasutuses olevate akudel töötavate raadioseadmetega siis, kui nad juhuslikult masinate lähedusse satuvad. Tööpäeva lõppedes lahkub töötaja ja viib passiivselt andmed maa peale.

Probleemi lahenduseks saab luua andmevahetusprotokolli, kus andmete saatmisel ja vastuvõtmisel esinev paketikadu ei mõjuta järgnevate ega eelnevate kohalejõudnud pakettide käsitlemist. Töö eesmärk on optimeerida mitme seadme vaheliste andmete sünkroniseerimiseks kuluvat infovahetust. Üks seade saadab mingi aja tagant välja oma andmete seisu, millega teised seadmed oma andmeid võrrelda saavad ja siis puuduolevad välja saata. Seejuures ei pea üks raadio olema teadlik leviulatusest väljas olevate raadiote olemasolust. Piisab, et mingi raadio on levialas, kellega aeg-ajalt infot vahetada.

Protokoll peab sisaldama võimekust oma andmete edasisaatmiseks ning mingisuguse andmestruktuuri edastamiseks, mis võimaldab võrrelda, millised andmed erinevates seadmetes olemas on.

Töös kirjeldatakse andmete sünkroniseerimise protokoll, luuakse prototüüp ja sooritatakse tulemuste analüüs erinevate eksperimentide kaudu prototüüpvõrgus. Võrdlusbaasiks on rsync protokolliga andmete sünkroniseerimine.

2 Raadiovõrgu probleemid

2.1 Leviulatus

Nagu eelnevalt mainitud, on eesmärgiks luua raadiovõrk, kus kõigi võrgusõlmede kogutud andmeid sünkroniseeritakse ja sünkroniseerimiseks kuluvat andmemahtu optimeeritakse, et sedasi leviulatust suurendada. Kuna saatevõimsus ei saa väga suur olla seadmetel, mille toiteallikaks on näiteks päikesepaneelid ja/või akud, ei ole ka ühe raadio leviulatus väga suur. Suureks segavaks faktoriks on ka maastik. Kui info peab levima kaugemale, kui ühe raadio leviulatus, on praktiline kasutada võrgus mitut raadiot nii, et info ka ümber/üle muidu raadiosignaali jaoks läbimatute takistuste (nt mägede) jõuaks.

2.2 Madal ribalaius

Teiseks suuremaks probleemiks on litsentseerimata sagedusalades [2] toimivate raadiote ribalaius. Näitena võib tuua Digi Xbee ~868MHz raadiod, mis võimaldavad kuni 10Kbps üle mõningate kilomeetrite. Et raadiovõrku mitte üleliigselt koormata, ei tasu selle peal kõrgema taseme TCP/IP ja sarnaseid protokolle kasutada. Sobiv lahendus peaks vältima üleliigsete andmete saatmist. Üleliigsete andmete osas ei ole paketipäised kõige suuremaks probleemiks andmevahetuse juures. Nimelt tuleb andmete sünkroniseerimisel kindlaks teha, kas andmed on jõudnud kohale. Ühendusele orienteeritud protokollide, nagu näiteks TCP/IP korral saadetakse andmete kättesaamisel tagasi ACK pakett, millega kinnitatakse andmete kättesaamist. Raadiovõrgus ACK pakkettide kasutamine on äärmiselt ebaefektiivne. Ideaalne oleks saata välja üks pakett, et kindlaks teha, mis andmed teises sõlmes olemas on.

2.3 Paketikadu

Raadiovõrkkude suur probleem on ka paketikadu. Kui paljud paketid kaduma lähevad, peab olema võimekus neid uuesti saata ilma liigset andmesidet kasutamata. Seega pärast andmete väidetavat sünkroniseerimist tuleks uuesti kontrollida, mis paketid kohale ei jõudnud. Kuna TCP protokolli rakendamine suurte kadudega raadiovõrgus on ebapraktiline, sest uuringus [3] näidatakse, et 2-3%-line paketikadu põhjustab TCP

ühenduse läbilaskevõime vähenemist poole võrra. Seepärast tuleb efektiivselt lahendada kaduma läinud andmete uuesti saatmine.

3 Tehnoloogia

3.1 Bloomi Filter

1970. aastal välja antud artiklis “Space/time trade-offs in hash coding with allowable errors” [4] toob Burton Howard Bloom välja meetodi, mida tänapäeval kutsutakse Bloomi filtriks.

Tegemist on tõenäosusliku andmestruktuuriga, mis koostatakse lähteandmete põhjal ja vastuseks saadakse bitijada, kus iga elementi tähistab mingi arv bitte (tavaliselt vähem kui 10). Bloomi filtrist ei saa kätte andmeid, küll aga saab arvutada, kas mingid andmed on sinna sisestatud või mitte. Siiski ei ole tegemist täielikult kindla vastusega. Saab teada, kas elementi ei ole kindlasti või et suure tõenäosusega on. Filtrist ei saa hiljem andmeid eemaldada, sest mitme elementi bitid võivad viidata samale kohale. Kuna iga element viitab k arvu bittidele filtris, võivad tekkida kollisioonid ja vahepeal võib tulla tulemus valepositiivne. Õnneks on võimalik vältida teisel korral samasuguse kollisiooni tekkimist, näiteks vähesel määral filtri suuruse varieerimisega. Nii ei pruugi järgmises sünkroniseerimistsüklis sama element enam kollisiooni tekitada.

Tabel 1 Bloomi filtri näide

Hüpoteetiline bloom filter mis on 16 bitti suur ja iga elemendi kohta 3 bitti.															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lisame filtrisse elemendi A, mille räsifunktsioon viitab kohtadele 0,3,6															
1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
Lisame filtrisse elemendi B, mille räsifunktsioon viitab kohtadele 2,6,10															
1	0	1	1	0	0	1	0	0	0	1	0	0	0	0	0
Lisame filtrisse elemendi C, mille räsifunktsioon viitab kohtadele 8,11,12															
1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	0

Et saada teada, kas filtris võib olla element B (tabel 1), saame räsifunktsiooniga 3 bitti (k=3), mis viitavad kohtadele 2, 6, 10. Kuna need bitid on räsis kõik väärtusega 1, on element B suure tõenäosusega filtris. Vaadates filtrit, proovime leida, kas element D (kohad 1, 11, 15) on seal sees, leiame, et bitid 1, 15 on nullid ja 11 on 1, seega vähemalt üks bitt on null ja element D ei asu kindlasti filtris. Kollisioon tekib siis, kui element E, mida küll filtris ei ole aga tema kohad vastavad filtris kohtadele, mis on kõik juhuslikult ühed. Selle pärast ei saagi kunagi täielikult teada, kas element ka tegelikult filtris asub.

Bloomi filtri loomiseks on vaja kahte parameetrit: bittide arvu filtris ja mitu bitti elemendi kohta täidetakse filtris. Need parameetrid saab ka arvutada nii, et valepositiivse tõenäosus oleks võimalikult väike.

Meie raadiovõrgus peaks olema filtri suurus suhteliselt kindlalt määratud suurus, sest ei ole praktiline saata liiga suurt filtrit. Olgu filtri suurus 200 baiti (1600 bitti), siis teades, mitu elementi me soovime filtrisse panna, saame arvutada välja parima bitiarvu elemendi kohta ja valepositiivse tõenäosuse.

Tähised:

k – tõeseid bitte filtrisse lisatava elemendi kohta

m – filtri suurus bittides

n – lisatavate elementide arv

p – valepositiivse tõenäosus

Tähtsamad valemid [5]:

$$k = \frac{m}{n} * ln2$$

$\ln(p) = -\frac{m}{n} * (\ln(2))^2 \Rightarrow p = e^{m*\frac{\log^2(2)}{n}}$, kus $\ln(p)$ on naturaallogaritm p'st ja $\ln(2)$ on naturaallogaritm 2'st

Tabel 2 Bloomi filtri valepositiivse tõenäosus sõltuvalt filtri suurusest, elementide arvust optimaalse k juures

filtri suurus (m)	1600		3200	
elementide arv (n)	bitti elemendi kohta (k)	valepositiivse tõenäosus (p)	(k)	(p)
100	11	0,000458711	22	2,10416E-07
200	6	0,021577141	11	0,000458711
400	3	0,146891598	6	0,021577141
800	1	0,39346934	3	0,146891598
1600	1	0,632120559	1	0,39346934

Tabelist 2 võib välja lugeda, et kui tahta säilitada valepositiivse tõenäosust tuleb säilitada elementide arvu ja filtri suuruse suhet, seejuures jälgida kõige paremat biti arvu elemendi

kohta. Näiteks, et säilitada tõenäosust 0.02, peaks filtri suurus olema 8 bitti lisatava elemendi kohta ja tõeste bittide arv elemendi kohta 6.

Nagu paljude tehnoloogiatega, on ka Bloomi filtril palju erinevaid rakendusi. Artiklis „Theory and Practice of Bloom Filters for Distributed Systems“ [6] on välja toodud pikk nimekiri erinevatest Bloomi filtri rakendustest. Näiteks Bloomi filtrid, millest saab elemente kustutada, küsida, mitu korda on mingit elementi sisestatud, kas mingid elemendid on lähestikku ja palju muud.

Bloomi filtrile leiab väga palju kasutusi. Andrei Broder ja Michael Mitzenmacher on maininud oma töös „Network Applications of Bloom Filters: A Survey“ [7] mitmeid võimalikke kasutusi võrgus. Hajus-vahemälud, P2P võrgus andmete leidmine, otsingumootori optimeerimisel, marsruutimisprotokollides ja paljudes muudes kohtades. Google kasutab „Bigtable“ [8] süsteemis Bloomi filtrit, et vähendada kõvaketaste pöördumiste arvu, leides Bloomi filtri abil, kas otsitavad read või veerud andmebaasides üldse võivad eksisteerida.

Hea näide on andmebaasi koormuse leevendamiseks ka elektrooniliste postiaadresside lisamine Bloomi filtrisse. Kui kasutaja loob uut aadressi, on väga kiiresti filtrist võimalik teada saada, kas selline aadress on vaba. Kui ei ole, tasub alles andmebaasilt üle küsida, sest on väike võimalus valepositiivsele. Kui on vaba, siis teame filtri põhjal seda kindlalt.

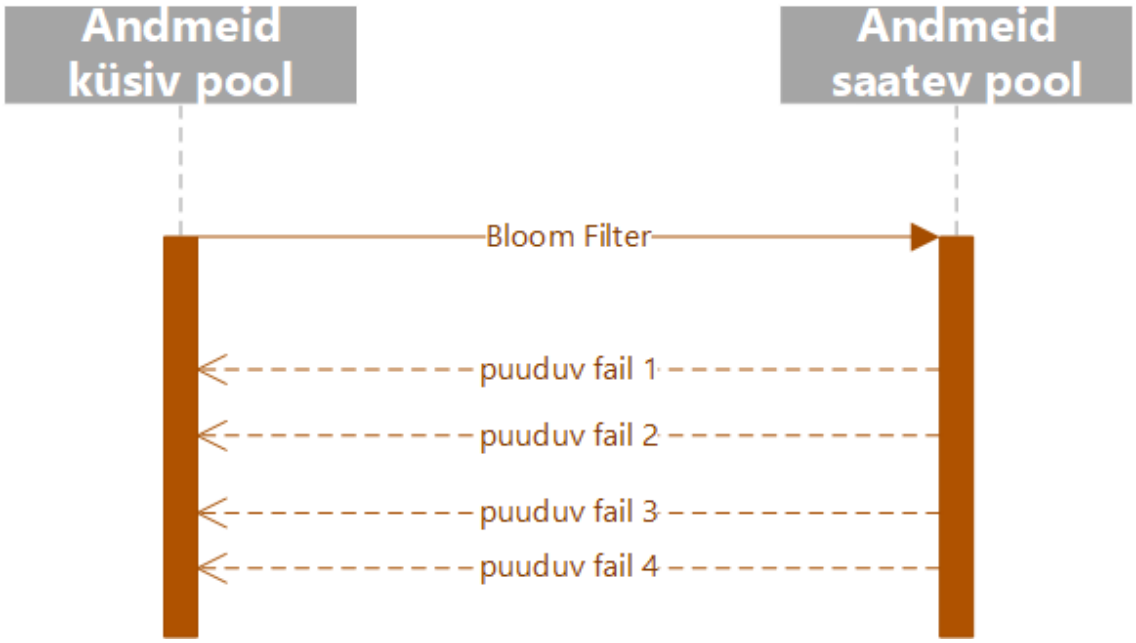
3.2 Rsync

Rsync on kõrgema taseme protokoll andmete sünkroniseerimiseks, mis tugineb kadudeta andmevahetusele, tüüpiliselt TCP/IP ühendusele. Rsync jagab faili sisud blokkideks (1024,2048 baiti blokk) ja teeb nendest räsid. Seega saadab rsync päris palju andmeid juba räside kujul. Rsync saadab üle võrgu alustuseks terve failipuu ja iga faili kohta vähemalt ühe 16 baidise räsi. See on juba väga palju andmeid, mida saata enne, kui andmeid saatma tuleb kahata. Lisaks kontrollib TCP ühendus ka pakettide kohalejõudmist, mis veelgi suurendab andmemahtu.

4 Andmete sünkroniseerimine Bloomi filtri põhise protokolliga

4.1 Sünkroniseerimiseks kuluva andmemahu optimeerimine

Et andmemahtu hoida tagasihoidlikuna, olen valinud võrgusõlmedes oleva informatsiooni võrdlemiseks välja Bloomi filtri, mille sisse lisame elemendid (failid, andmebaasiread, sõnumid jms.) ja siis saadame selle välja seadmetele, kelle vahel soovime sünkronisatsiooni saavutada. Teine (andmeid saatev) pool, saades kätte esimese (andmeid küsiva) poole filtri, saab võrrelda, kas küsival poolel on olemas elemendid, mis saatval poolel on. Kui elemendile viitavatest bittidest on vähemalt üks null, siis pole küsival poolel kõnealust elementi. Võrreldes kogu informatsioonihulka, saame teada peaaegu kõik sõnumid, mida on vaja küsijale saata (joonis 2).



Joonis 2 Kommunikatsiooniskeem

Bloomi filter on valitud vahend andmemahu optimeerimiseks, sest see annab väga vähese andmehulgaga suhteliselt täpset infot, milliseid andmeid saata on vaja.

Bloomi filtriga loodud protokolli eelis TCP-l põhinevate ees on see, et me ei vaja mehhanismi, mis saadab tagasi andmed, millised paketid kohale jõudsid või mitte. Järgmise sünkroniseerimistsükli ajal saadakse kaduma läinud paketid teada ja saadetakse

uuesti. See põhjustab küll pikemad viited andmete kohalejõudmises, kuid hoiab kokku andmevahetust, mis on põhiline probleem kitsaribalistes võrkudes. Kui kasutaja rakendus ei vaja andmeid kohe, peaks selline lähenemine andmete sünkroniseerimisele olema väga efektiivne.

Protokoll võib kasutada korraga mitut võrguliidest, mis lubab andmeid liigutada näiteks raadiovõrgust IP võrku ja vastupidi, lubades luua kiirteid teatud seadmete vahel. Näiteks kaks kõige kaugemat seadet omavad 4G internetiühendust omavahel ja paketid jõuavad võrgu kaugematest osadest sissepoole delegeeruda samaaegselt.

Bloomi filtri saab asendada ka näiteks cuckoo filtriga, mida saab rakendada samal viisil, kuid mille realisatsioon ja omadused erinevad mõningal määral. Artikkel “Cuckoo Filter: Practically Better Than Bloom” [9] toob välja ka cuckoo filtri eripärasuse kustutada andmeid filtrist. Kuna meie kustutamist ei vaja, pole vahet, kumba tehnoloogiat oma rakenduses kasutame. Küll aga võimaldab cuckoo filter võrku paremini analüüsida, kustutades teiste filtritest enda sõnumid. Nii saame hinnata, kui palju meil endal andmeid puudu on filtri omanikuga võrreldes. Lisaks saaksime võrrelda ka kahe teise seadme cuckoo filtreid ja arvutada nende omavahelised erinevused.

4.2 Protokolli realisatsioon

Protokoll on iseenesest suhteliselt primitiivne. Andmeid küsiv seade saadab välja enda andmebaasiseisu teatud ajavahemikus loodud andmete põhjal loodud Bloomi filtri. Filtri paketti lisatakse veel ajavahemik, Bloomi filtri parameetrid k (mitu tõest bitti elemendi kohta) ja m (filtri suurus bittides) ja kontrollsumma. Ajavahemik on filtrite tükeldamiseks tähtis, sest andmed kulmineeruvad ja mõistliku suurusega filter muutub suuremate andmekoguste juures ebaefektiivseks. Parameetrid (k, m) on tähtsad, et Bloomi filtri andmeid saatev pool saaks taastada samasuguse filtri. Seepeale leiab andmeid saatev pool oma andmebaasi põhjal, millised andmed on Bloomi filtri omanikul puudu ja saadab need välja.

Kui leviulatuses on rohkem kui 1 seade, siis oleks kasulik enne saatmist kuulata juhusliku pikkusega aja jooksul, ega keegi teine juba vastama ei hakanud.

Sünkroniseerimistsüklit korratakse mingi aja tagant, et mitte kohale jõudnud andmed uuesti saata ja ka uued sõnumid sünkroniseerida.

Selle töö tarbeks ei ole vaja realiseerida ajavahemikke, sest ilma selleta saab piisavalt analüüsida protokolli jõudlust ilma andmemahtu liiga suureks ajamata, et neid ei peaks jaotama vahemikeks. Et võrrelda Bloomi filtril põhinevat protokolli rsync protokolliga, kohandame loodava protokolli sünkroniseerima faile kaustas.

Kuna protokoll sisaldab kõiki tööks vajalikke päiseid, siis ei ole lisaks vaja TCP [10] või UDP [11] ja IP [12] päiseid, mis lisaksid omakorda veel vähemalt 28 baiti andmeid paketi kohta. See annab võimaluse protokollil töötada võrkudes, mis ei toeta TCP/IP või UDP/IP protokolle.

4.2.1 Andmeid küsiva seadme pseudokood

File[] files;

Serial serial; // raadio

```
async function sendBloomFilter()
    while true:
        short m = 1600 //filtri suurus bittides
        byte k = 11 //tõest bitti elemendi kohta
        BloomFilter filter = new BloomFilter(m, k)
        for file in files:
            filter.add(file.getName() + file.getData())
        Byte[] data = new Byte[]
        serial.write(data.joinAll(m.toBytes(),
                                  k.toBytes(),
                                  checksum(m.toBytes(),
                                           k.toBytes(),
                                           filter.toBytes()
                                  filter.toBytes())))
        wait(1 minute)

async function receiveData(Byte[] bytes)
    if !isChecksumOK(bytes.getInt(), bytes):
        return
    String name = getFileName(bytes)
    String contents = getFileContents(bytes)
    if (!files.containsFile(name)):
        files.add(new File(name, contents))
```

Koodinäide 1 Andmeid küsiva seadme pseudokood

4.2.2 Andmeid saatva seadme pseudokood

```
async function receiveBloomFilter(Byte[] bytes)
    short m = bytes.getShort()
    byte k = bytes.getByte()
    int checksum = bytes.getInt()
    filterData = bytes.getRemainingBytes()
    BloomFilter filter = new BloomFilter(m, k, bytes)
    for file in files:
        if !filter.contains(file.getName() + file.getData()):
            Byte[] data = new Byte[]
            serial.write(data.joinAll(checksum(file.getData()),
                                                file.getData()))
```

Koodinäide 2 Andmeid saatva seadme pseudokood

4.3 Probleemid

Protokolli juures tuleb meeles pidada kindlasti seda, et kõik raadiod võivad proovida vastata. Selleks tuleb rakendada „kuula enne kui saadad“ loogikat, et aru saada, ega keegi juba enne ei saada samu andmeid.

Teiseks, kui filter väga ei muutu, võivad samad elemendid kollisiooni tekitada ja mingi element ei pruugi kunagi sünkroniseerituks saada. Seda võib lahendada filtri suuruse vähese varieerimisega. Isegi kui filter varieerub 1 baidi võrra, hakkavad igale elemendile mõeldud kohad drastiliselt muutuma ja seega on korduv kollisioon juba suurusjärgu võrra väiksema võimalusega eeldusel, et filter ei ole üle täidetud ja kollisiooni võimalus seega liiga suur.

Kui elementide arv filtris läheb suureks, tekib uus probleem. Nimelt efektiivne filtri suurus on lineaarne elementide arvuga ja ei ole praktiline luua liiga suuri filtreid võrgus saatmiseks. Kui kasutada ajavahemikke, siis tekib iga ajavahemiku jaoks filter. Kui neid on palju, näiteks üle 100, siis ei ole praktiline iga ajavahemikku kogu aeg sünkroniseerida. Lahenduseks oleks hilisemaid ajavahemikke järjest harvemini küsida ja kui leitakse erinevused, siis tihendada selle ajavahemiku küsimist. Teiseks võib filtreid kokku pakkida. Michael Mitzenmacher on toonud artiklis „Compressed Bloom Filters“ [13] välja, kuidas vähendada filtri suurust ilma valepositiivse tõenäosuse suurenemiseta. Lisaks saab ka ajavahemikest filtri koostada, kus iga element on iga ajavahemiku kogu andmehulga räsi. Nii saab teada, kas mingis ajavahemikus on muudatusi toimunud või mitte.

Kui kellaajad ei ole seadmetel samad, võib juhtuda, et sõnum luuakse tulevikus oleva kellaajaga. Nii võib uus pakett ajavahemikust välja jääda. Kui usaldada alati andmepaketiga saadetud andmete loomisaega, ei lähe lõpuks midagi kaduma, aga info võib halvimal juhul lihtsalt hiljem kohale jõuda või kui kasutaja rakendus kasutab loomisaega analüüsiks, võib see tulemusi segada.

5 Katse

5.1 Katse keskkond

Testimiseks koostasin Java rakenduse, mis simuleerib võrgusõlmede vahelist suhtlust läbi jadaliidese või UDP pakettide. UDP lahendust protokoll ei nõua, see on lihtsalt vahend protokolli simuleerimiseks IP võrkudes. Üle jadaliidese töötab protokoll sama hästi puhtalt enda päistega. Programmeerisin koodi otse sisse ka andmekao tõenäosuse. Kaustas „files“ asusid 100 200 baidist faili, mida tuli sünkroniseerida teisele samasugusele rakendusele. Rakendus loeb sisse failid ja saadab iga teatud aja tagant välja Bloomi filtri, mis on koostatud failide andmete (faili nimi ja sisu) põhjal. Ühel poolel on kõik failid, teisel pole ühtegi. Bloomi filtriga protokolliga uurin, kui palju andmevahetust toimub hea kvaliteediga võrgu kui ka väga suure kaoga võrgu puhul. Kaks seadet ühendasin omavahel serial null moodem kaabliga, mille RX ja TX olid risti ühendatud.

Kuna rsync eeldab tüüpiliselt TCP protokolli, on raske testida paketikadudega andmevahetust, sest kadunud paketid saadab uuesti võrgukaart ja selline info ei jõua tarkvarani tagasi. Samas on võimalik testida, kui palju kasutab andmeid rsync, et võrrelda, mis andmed puudu on. Rsynci jaoks kasutasin samu faile, mida ülalnimetatud katses.

5.2 Katse tulemused

Esimeses katses (tabel 3) on iga faili suurus saatmisel koos päistega 219 baiti. Bloomi filter koos päistega on 418 baiti. Katse jaoks kasutame konstantse suurusega Bloomi filtrit. Katseid kordasin 1000 korda andmekao tõenäosuste 0, 0.1, 0,25 juures ja 100 korda andmekao tõenäosuste 0.5 ja 0.75 juures, leidsin ka keskmise tulemuse ja standardhälbe.

Tabel 3 Bloomi filtriga protokolli andmekasutus erinevate andmekadude juures

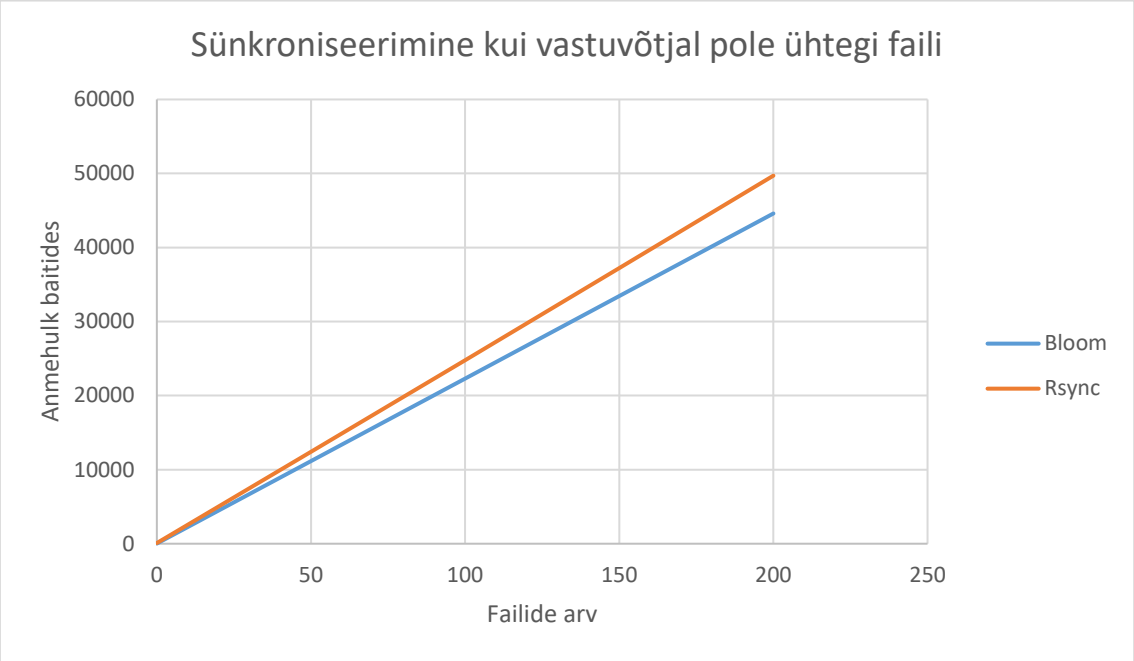
	0 tõenäosusega kadu	0.1 tõenäosusega kadu	0.25 tõenäosusega kadu	0.5 tõenäosusega kadu	0.75 tõenäosusega kadu
Bloom filtriga protokolliga kohale jõudnud sünkroniseerimis-pakettide arv	1 σ=0	2 σ=0.99	4 σ=1.02	7 σ=2.08	18 σ=4.49
Bloom filtriga protokolliga andmepaketikadu täielikul (100 faili) sünkroniseerimisel	0 σ=0	11 σ=3.5	33 σ=6.8	100 σ=14.2	300 σ=34.26
Saadetud andmemaht kokku	22,3 KB	25.1 KB	30.8 KB	46.7 KB	95.1 KB

Teises katses (tabel 4) saadan Bloomi filtrit optimaalse suurusega, hoides kollisiooni tõenäosust 2% peal. Katse näitab, kui palju lisaandmeid läheb sünkroniseerimiseks mõlema protokolli juures, kui võrgu paketikadu mitte arvestada (tabel 4).

Tabel 4 Rsynci ja Bloomi filtriga loodud protokolli võrdlus kadudeta võrgu juures

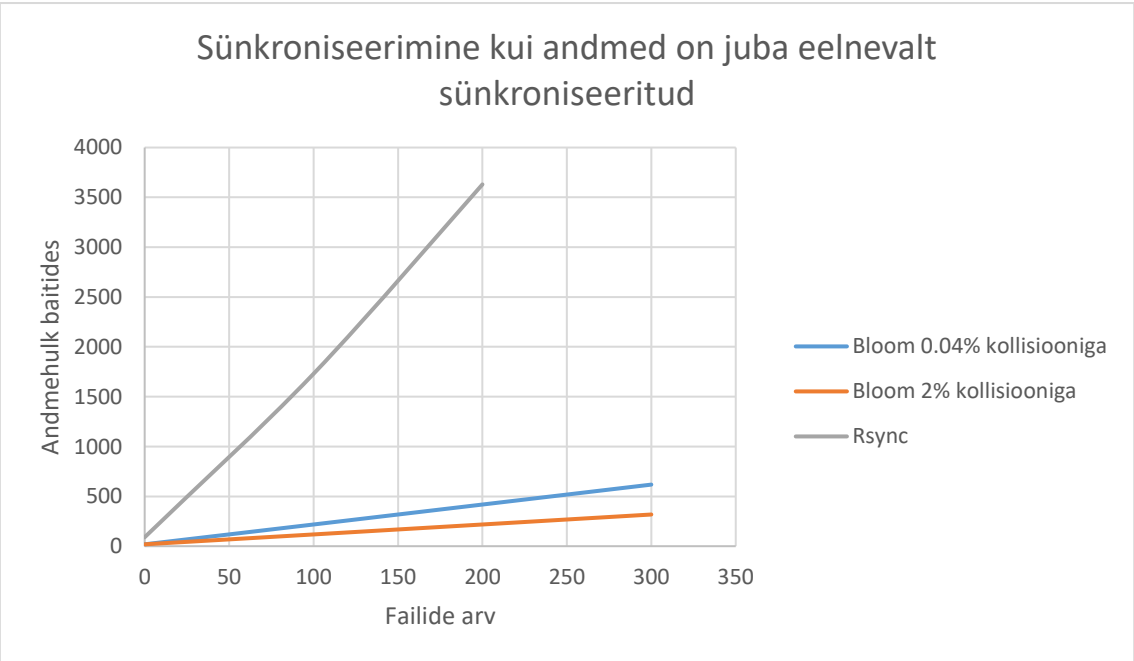
		0 faili	100 faili	200 faili
Rsync tühi-sünkroniatsioon	failidega	90 B	24.8 KB	49.7 KB
	ilma failideta	90 B	1.7 KB	3.63 KB
Bloom filtriga sünkronisatsioon	failidega	19 B	22 KB	44 KB
	ilma failideta	19 B	118 B	218 B

5.3 Tulemuste analüüs



Joonis 3 Täieliku sünkronisatsiooni andmevahetuse hulga võrdlus Bloomi filtriga protokolli ja rsynci vahel

Kui mitte kaasa arvata rsync TCP liiklusega kaasnevat andmehulka, siis puhtaks andmevahetuseks kulus rsync protokollil märksa rohkem andmeid, juba 200 faili juures oli vaheks 1.1 KB (joonis 3).



Joonis 4 Täielikult sünkroniseeritud seadmete vahel uuesti sünkroniseerimise andmeside hulk Bloomi filtriga loodud protokolli ja rsynci vahel

Kuna meie süsteemis tuleb sünkroniseerida iga mingi aja tagant, võivad andmed juba enne sünkroonis olla. Siit võib välja lugeda, et Bloomi filter on väga efektiivne ka siis, kui andmed on juba sünkroniseeritud, sest puudub vajadus reaalse nimekirja edastamiseks. Kui koostada filtri suurus elementide arvu järgi, siis 2% kollisiooni võimalusega filtri suurus on 1 bait iga lisatava elemendi korral ja 0.04% kollisiooniga 2 baiti elemendi kohta. See tuleb eriti kasuks suurte andmekoguste juures, sest Bloomi filtri suurus kasvab küll lineaarselt, kuid aeglasemalt kui rsync protokolli lineaarne kasv (joonis 4).

Protokolli suur eelis on tema laiskus. Kui sünkroniseerimistsükkel on läbi, ei kontrollita, kas kõik jõudis kohale või mitte. Järgmine sünkroniseerimistsükli jooksul saadakse teada puuduvad andmed ja edastatakse nad uuesti. See annab aega teistele seadmetele oma sünkroniseerimised teha ja kui mingi seade kuuleb teiste pakette, saab ta need endale igal juhul, kui tal neid ei peaks eksisteerima. Lõpuks jõuavad andmed ikka kohale.

Kokkuvõte

Töös välja töötatud ja analüüsitud Bloomi filtriga andmete sünkroniseerimise protokoll ei vaja paketikao juures kaotatud pakettide kohest uuesti saatmist ega kontrollivat mehhanismi, kas pakett jõudis kohale. Kaduma läinud pakettide probleem on lahendatud Bloomi filtriga, mille saadab välja andmeid küsiv seade. Kui võrgu topoloogia ei ole fikseeritud (nt raadiovõrk), siis saame osa informatsiooni ka teiste võrgusõlmede omavahelisest suhtlusest salvestada. Kuna ükski pakett ei sõltu eelnevatest, ei ole paketikadu, ükskõik kui suur (v.a täielik paketikadu) see ka oleks, liiga suureks probleemiks. Suurema paketikao juures on täieliku sünkroniseerimise aeg pikem, aga andmed jõuavad kohale igal juhul eeldusel, et uusi andmeid ei toodeta rohkem, kui on efektiivne võrgu läbilaskevõime, mis oleks sama suureks probleemiks iga sünkroniseerimisprotokolli jaoks.

Katsetest tuli välja, et loodud lahendus toimib võrreldes rsync protokolliga eriti säästlikult olukorras, kus enamus andmeid on juba sünkroniseeritud.

Bloomi filter osutus heaks andmestruktuuriks üle võrgu andmete seisu saatmiseks. Kuigi väikese tõenäosusega võivad mõned andmed saatmata jääda, ei ole see probleemiks, kui varieerida filtri suurust. Ohverdades andmete sünkroniseerimise kiiruse, saame palju efektiivsemalt võrrelda puuduolevaid andmeid ja saata ainult puuduolevad, kasutades võimalikult vähe andmesidet.

Protokoll töötab hästi hajusvõrkudes, kus info propageeritakse seadmetele edasi sama raadiokanali või võrgu peal, vajamata selleks IP ja sellega kaasnevaid UDP või TCP protokolle, mis põhjustavad juba ainuüksi paketipäiste tõttu suuremat andmekasutust.

Kasutatud kirjandus

[1] M. Eronen, „Implementation of Data Synchronization Over a Challenged Network,“ 2015 8 2015. [Võrgumaterjal]. Available: https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/23292/eronen.pdf. [Kasutatud 23 5 2018].

[2] „Eesti Raadiosageduste Plaan,“ 4 4 2003. [Võrgumaterjal]. Available: https://www.riigiteataja.ee/akt/563189. [Kasutatud 23 5 2018].

[3] S. Chowdhury ja K. Fatema, „Analysing TCP Performance When Link Experiencing Packet Loss,“ 10 2014. [Võrgumaterjal]. Available: http://publications.lib.chalmers.se/records/fulltext/193786/193786.pdf. [Kasutatud 23 5 2018].

[4] H. B. Bloom, „Space/Time Trade-offs in Hash Coding With Allowable Errors,“ *Commun. ACM,* kd. 1, nr 7, pp. 422-426, 1970.

[5] N. Siddikov, „Scalable Cooperative Caching Algorithm Based on Bloom Filters,“ 8 2011. [Võrgumaterjal]. Available: http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1471&context=theses. [Kasutatud 18 5 2018].

[6] S. Tarkoma, C. E. Rothenberg ja E. Lagerspetz, „Theory and Practice of Bloom Filters for Distributed Systems,“ *IEEE Communications Surveys & Tutorials,* kd. 14, nr 1, pp. 131-155, 2011.

[7] A. Brodery ja M. Mitzenmacherz, „Network Applications of Bloom Filters: A Survey,“ *Internet Mathematics,* kd. 1, nr 4, pp. 485-509, 2004.

[8] F. Chang, J. Dean, S. Ghemawat, C. W. Hsieh, A. D. Wallach, M. Burrows, T. Chandra, A. Fikes ja E. R. Gruber, „Bigtable: A Distributed Storage System for Structured Data,“ %1 *7th {USENIX} Symposium on Operating Systems Design and Implementation (OSDI),* 2006, pp. 205-218.

[9] B. Fan, G. D. Andersen, M. Kaminsky ja D. M. Mitzenmacher, „Cuckoo Filter: Practically Better Than Bloom,“ *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies,* pp. 75-88, 2014.

[10] „Transmission Control Protocol,“ 9 1981. [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc793. [Kasutatud 21 5 2018].

[11] „User Datagram Protocol,“ 28 8 1980. [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc768. [Kasutatud 21 5 2018].

[12] „Internet Protocol,“ 9 1981. [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc791. [Kasutatud 23 5 2018].

[13] M. Mitzenmacher, „Compressed Bloom filters,“ *IEEE/ACM Transactions on Networking,* kd. 10, nr 5, pp. 604-612, 2002.