TARTU ÜLIKOOL

Arvutiteaduse instituut

Informaatika (2476) õppekava

Kaire Koljal

Apache MiNiFi efektiivsus servandmetöötluse raamistikuna

Bakalaureusetöö (9 EAP)

Juhendaja: Pelle Jakovits

Tartu 2020

Apache MiNiFi efektiivsus servandmetöötluse raamistikuna

Lühikokkuvõte:

IoT võrgustiku laienemise ning seadmete poolt kogutavate andmemahtude suurenemise tagajärjel ei ole andmete pilve saatmine alati kõige mõistlikum. Paraku pole kõik IoT seadmetel piisavalt ressursse, et andmeid ise töödelda. Selle bakalaureusetöö eesmärk on hinnata, kas piiratud ressurssidega seadmetes, nagu Raspberry Pi, on Apache MiNiFi'l eeliseid võrreldes kõikide andmete edasi saatmisega pilve ning kui mõistlik on selliste raamistike kasutamise servaseadmes üldiselt. Käesolevas töös võrreldakse MiNiFi'ga saadetud andmete ja analüüsimata andmete edastamisel kulunud Raspberry ressursside hulka ning töödeldavate andmete mahtu. Samuti võrreldakse kogu andmevoo ahela latentsusaega Raspberry Pi'st Apache Sparkini jõudmiseks.

Võtmesõnad:

Andmevoo töötlus, Apache MiNiFi, Raspberry Pi, Asjade Internet (IoT)

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Efficiency of Apache MiNiFi as a framework for edge computing

Abstract:

With the development of IoT and the growth of data collected by IoT devices, sending all the collected data straight to cloud is not always the best option. Unfortunately, not all IoT devices have enough resources to process all the data. The goal of this thesis is to evaluate if using Apache MiNiFi on resource constrained devices like Raspberry Pi has any benefits compared to sending all the data to cloud and if using such frameworks on edge devices is useful. The goal of the comparison is to assess the amount of Raspberry's resources used and the amount of data processed in both scenarios. In addition, the difference in latency from edge device to Apache Spark will be compared.

Keywords:

Stream data processing, Apache MiNiFi, Raspberry Pi, Internet of Things (IoT)

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

1. Sissejuhatus

Asjade internet (inglise keeles *Internet of Things*, lühidalt IoT) on elektrooniliste seadmete võrgustik, kus seadmed on ühendatud internetti ning saavad üksteisega suhelda ja andmeid vahetada. Nende seadmete poolt kogutud andmed kogutakse üldjuhul pilves asuvasse IoT serverisse või andmebaasi. Kogutud andmete maht on suur ning IoT võrgustiku laienemise tõttu suureneb ka pidevalt kogutavate andmete maht, millega kaasneb suurenev latentsusaeg ja ummikud võrguühenduses [1] ning kõigi andmete pilve saatmine ei ole alati parim lahendus. Seetõttu on muutunud populaarsemaks andmete lokaalne töötlemine seadmes või seadme lähedal enne pilve saatmist. Samas on andmeid koguvad seadmed tihti jõudluse poolest ressursivaesed ning nendes ei ole võimalik teha efektiivselt mahukaid arvutusi. Neid seadmeid kutsutakse servaseadmeteks (inglise keeles *edge devices*) [2].

Servaseadmeid kasutatakse näiteks keskkonnaseires, kus andmete kogumispunktides on servaseadme külge ühendatud erinevad sensorid, mis koguvad infot näiteks temperatuuri, õhuniiskuse ja saastatuse taseme kohta [3]. Nendega kogutakse pidevalt suures koguses andmeid, mida on nendega ühendatud seadmel keeruline efektiivselt analüüsida. Üheks selliseks seadmeks on Raspberry Pi. Raspberry Pi paistab silma selle poolest, et pole väga kallis, töötab efektiivselt servaseadmena ning on ka sobilik platvorm uurimustööde läbiviimiseks [4].

Selleks, et lihtsustada andmete töötlust servaseadmetes või nende lähedal, on loodud erinevaid lahendusi. Üheks selliseks raamistikuks on Apache MiNiFi [5], mis keskendub andmevoo haldamisele ja kogumisele nende loomise kohas.

Selleks, et teada saada, kas raamistiku kasutamisel servaseadmes on eeliseid, tehti selles töös otsus võrrelda andmete edastamist raamistikuga ja ilma, et tuua välja raamistikuga seotud ressursside kulu. Uuritavaks raamitikuks valiti MiNiFi kuna see on üks levinumatest servaandmetöötluse raamistikest.

Käesolevas töös uuritakse Apache MiNiFi raamistiku kasutamise mõistlikkust servaseadmes läbi praktilise katsetuse, valides riistvaraliseks lahenduseks Raspberry Pi. Katsetuse käigus võrreldakse Raspberry Pi ressursside kulu ning töödeldavate andmete mahtu ilma raamistikuta ning MiNiFi'd kasutades. Vaadeldakse Raspberry Pi protsessori tuumade hõivatust ja temperatuuri. Samuti kogutakse andmeid mälu kasutuse, töödeldavate andmete mahu ning andmevootöötluse protsessi pikkuse kohta.

Servaseadmes oleva raamistiku poolt kogutud infot on vaja edasi saata. Selleks on antud töös kasutatud Apache Kafkat [6]. Kafka on kõrge läbilaskevõimega ja madala latentsusajaga

andmevoo käsitsemise platvorm, mille eesmärk on edastada andmeid erinevate andmetöötluse raamistike vahel, kuid võimaldab ka andmeid töödelda Kafka Streaming liidese kaudu.

Vajalik on ka edastavate andmete kogumine ja salvestamine. Selleks on antud töös kasutatud Apache Sparki [7]. Apache Spark on kiire ja üldine raamistik suurte andmehulkade töötlemiseks, mida kasutatakse selle töö raames Kafkast andmete kätte saamiseks ja salvestamiseks.

Käesoleva töö raames antakse ka ülevaade testides kasutatavate keskkondade ülesseadmisest, võimalikest probleemidest ja nende lahendustest. Antud tööga tutvumine oleks kasulik nendele, kes soovivad teada saada, kui otstarbekas on kasutada servaarvutusraamistikke ja NiFi't.

Töö koosneb viiest peatükist. Peatükis 2 antakse ülevaade kasutatud tehnoloogiatest ning varasematest uuringutest. Peatükis 3 kirjeldatakse testide ülesehitust. Võrdluse tulemused ning analüüs on kirjeldatud peatükis 4. Kogu töö kokkuvõte esitatakse peatükis 5.

2. Taust

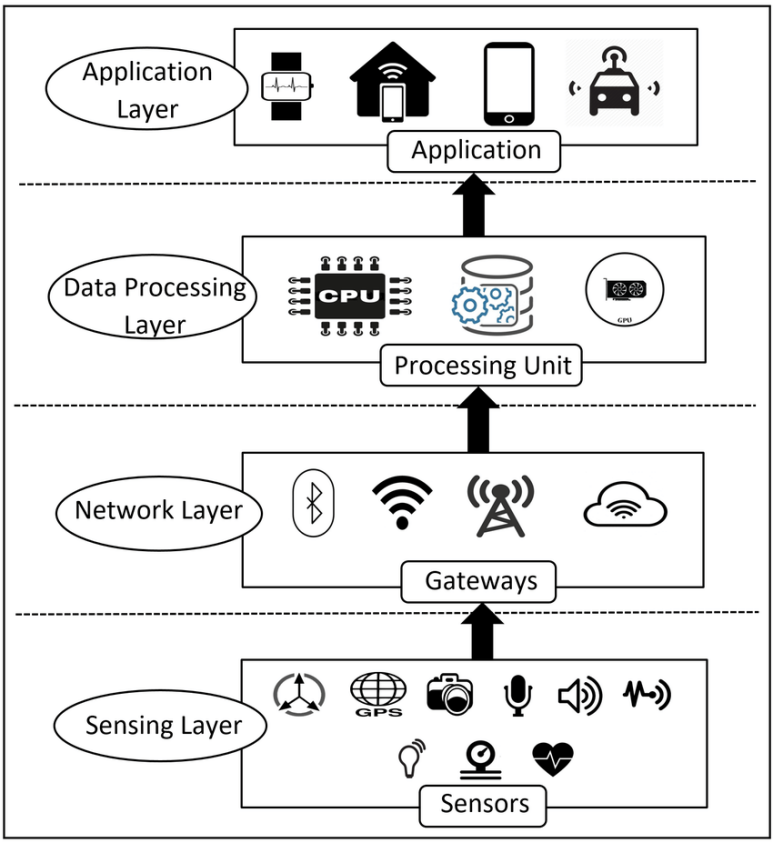
Selles peatükis antakse ülevaade selles töös kasutatavatest andmetöötluse raamistikest, seotud terminitest ning varem läbi viidud uurimustest.

Alampeatükis 2.1 antakse ülevaade asjade internetist ja selle seotusest pilveandmetöötlusega. Peatükis 2.2 kirjeldatakse lühidalt servaarvutuse ning udutöötluse tööpõhimõtet, arhitektuuri ning põhjendatakse nende vajalikkust. Peatükk 2.3 annab lühiülevaate enamlevinumatest andmetöötluse raamistikest IoT ja uduarvutuse keskkonnas ning peatükis 2.4 kirjeldatakse varem läbi viidud uurimusi ning teemakohaseid töid.

2.1 Asjade internet

Asjade internet ehk IoT on erinevatest seadmetest koosnev võrgustik. Seadmed koguvad ja vahetavad infot iseseisvalt ning üldjuhul saadavad kogutud andmed edasi pilve. Jürisoo [8] toob oma artiklis välja, et sellisteks seadmeteks on näiteks nutitelefonid aga ka mõned külmikud, pesumasinad, nutikellad ja meditsiiniseadmed. Tema hinnangul moodustavad kõige suurema osa IoT seadmetest kodumasinad. Jürisoo kirjutab, et IoT laienemisega on suurenemas koduväliste IoT seadmete osakaal, mida kasutatakse transpordisüsteemides, linnaplaneerimises ja tervisepoliitikas. Üldjuhul on sellisteks seadmeteks erinevad sensorid tänavatel või tervist jälgivates seadmetes. Samuti on välja toodud, et asjade internet edendab ärisid, suurendades tootmise, energiahalduse, transpordi ja põllumajanduse tõhusust ning vähendades tegevuskulusid.

Joonisel 1 on toodud IoT kihid ja komponendid ning nende sõltuvus üksteisest. Esimese kihi moodustavad seadmed ja sensorid, mis andmeid koguvad. Teise kihi moodustavad Wi-Fi ning muud andmevõrgud, mille kaudu saadetakse andmed edasi kolmandasse kihti: andmeid analüüsivatesse seadmetesse. Sealt edasi saadetakse andmed kasutajakesksetesse seadmetesse.



Joonis 1. IoT kihid ja komponendid [9]

Asjade interneti laienemisega on suurenenud kogutavate andmete maht, mida ei ole alati mõistlik pilve saata. Samuti on andmevahetusel oluline minimeerida latentsusaega. Rouse [1] kirjutab, et servaseadmete poolt kogutud andmete hulk on suur ning selle analüüsimiseks pilve saatmine ning sealt vastuse ootamine võtab liiga kaua aega. Selleks on vaja võimalikult palju andmeid võimalikult efektiivselt seadme lähedal läbi töödelda. IoT seadmed pole loodud suurte andmemahtude läbi töötamiseks ja seega pole selliseks protsessiks piisavalt ressursse. Selleks on loodud servaarvutus (inglise keeles *Edge computing*), mis teeb ära ressursinõudlikuma töötluse, vähendades sellega latentsusaega ning pilve koormust.

2.2 Servaarvutus ja uduarvutus

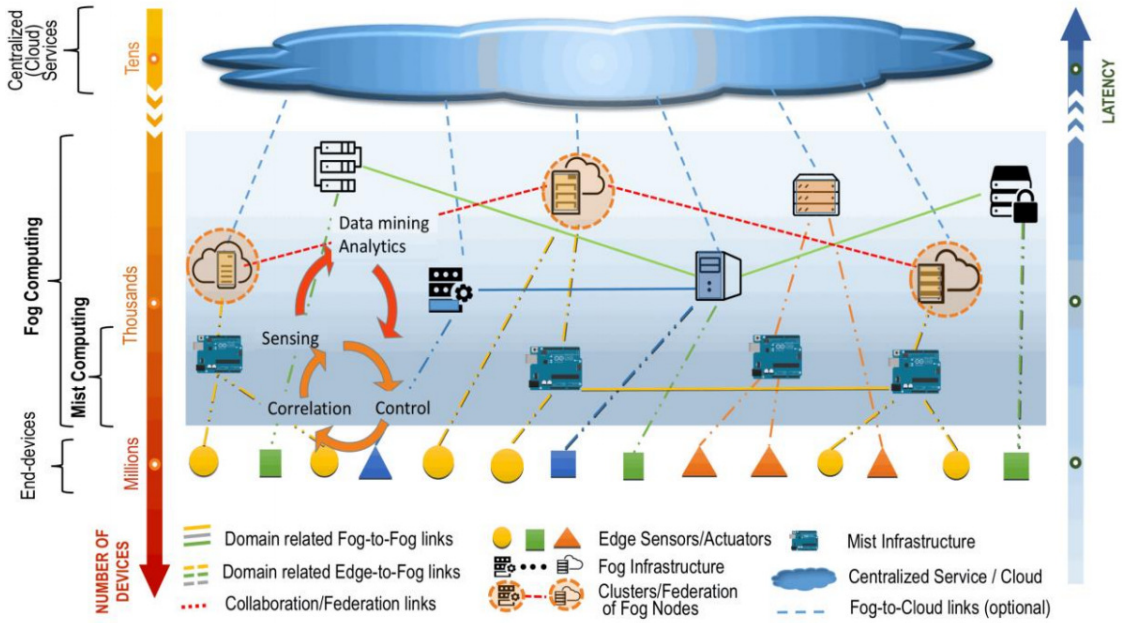
Servaarvutus (inglise keeles *Edge computing*) on võrgu arhitektuur, mille eesmärgiks oli laiendada pilveandmetöötlust lõppseadmeni [10]. Servaarvutus võimaldab teha osa andmetöötlust servaseadmes, saates pilve ainult olulisema info. Seega väheneb pilve saadetavast suurest andmemahust tulenev kõrge latentsusaeg. Servaarvutuse kasutamine on leidnud rakendust kohtades, kus kogutud andmete töötluse kiirusest sõltub olukorrale

reageerimine. Sellisteks kohtadeks on näiteks pangad, mis kasutavad reaalajas andmetöötlust pettuste avastamiseks, ning ka isesõitvad autod, mis peavad ümbritsevas toimuvale kiiresti reageerima [11]. Mõnikord kasutatakse servaarvutuse asemel sõna uduarvutus (inglise keeles *Fog computing*).

Uduarvutus keskendub sarnaselt servaarvutusele andmete töötlusele seadmete läheduses, kuid ei kasuta servaseadme ressursse. Uduarvutus on pilve laiendus, mis pakub lisaressursse andmete töötlemiseks ja nende salvestamiseks[12]. Põhiline erinevus servaarvutuse ja uduarvutuse vahel on andmete analüüsimise koht. Servaarvutuse puhul tehakse andmetöötlus servaseadmes endas, kus andmeid kogutakse. Uduarvutuse puhul viiakse andmete töötlus seadmest väljapoole, kuid mitte LAN võrgust välja [12].

Uduarvutusele lisaks on kasutusele võetud termin *Mist computing*, mis oli alternatiivne nimi uduarvutusele, kuid on nüüdseks muutunud uduarvutuse alamosaks.

Preden jt [13] toovad oma töös välja, et sarnaselt uduarvutusele teeb *Mist computing* andmetöötluse seadme lähedal, kuid erineb selle poolest, et kaasab seadmetega ühendatud sensoreid ja mikro-kontrollereid ning saadab andmed edasi udusse. Nad väidavad, et *Mist computing* paneb põhirõhu seadmete vahelisele suhtlusele ning seega on vähem suhtlust udu ja pilvega. Sellega kaasnevalt väheneb latentsusaeg, pikeneb servaseadme aku eluiga ning suureneb alamsüsteemi autonoomsus. Samuti toovad Preden jt välja, et uduarvutussüsteemide rakendamisega kaasnevad väljakutsed seisnevad võrgu keerukuses ja interaktsioonis, mida peavad seadmed ise haldama, kuna selliste süsteemide tsentraalne haldamine on raskesti teostatav.



Joonis 2. Uduarvutuse osakaal pilvearvutuses [14]

Joonisel 2 on toodud uduarvutuse osalus pilvearvutuse protsessis. Uduarvutust ei kirjeldata kohustusliku pilvetöötluse osana ning samuti ei ole pilv kohustuslik element seadmete omavaheliseks suhtluseks udus [14]. Joonisel 2 on *Mist computing* toodud välja uduarvutuse alamosana.

2.3 Andmetöötluse raamistikud uduarvutuse ja IoT keskkonnas

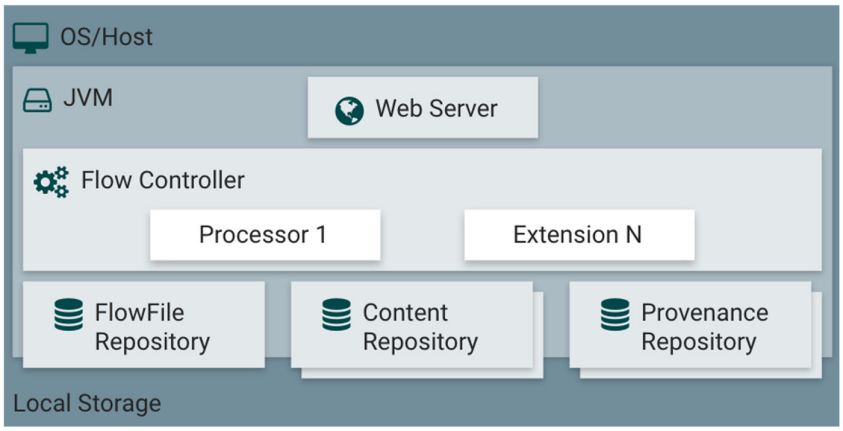
Selles peatükis tuuakse välja enamlevinumad andmetöötluse raamistikud uduarvutuses ja IoT keskkonnas. Iga raamistiku puhul kirjeldatakse lühidalt nende arhitektuuri ning tööpõhimõtet.

2.3.1 Apache NiFi ja MiNiFi

Apache NiFi on Java virtuaalmasinal töötav graafilise kasutajaliidesega platvorm [15]. See automatiseerib andmete liikumist erinevate andmeallikate ja süsteemide vahel, muutes andmete kogumise, transportimise ja vahepealse töötluse kiireks ja turvaliseks [16].

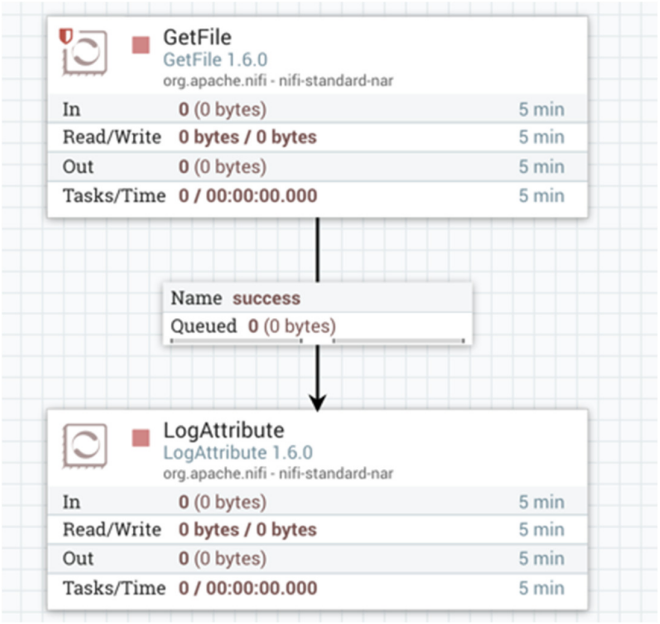
NiFi käivitab kõiki oma operatsioone Java virtuaalmasinas, mis töötab kasutaja operatsioonisüsteemil. NiFi arhitektuuri põhikomponentideks on veebiserver, voo kontroller (*flow controller*) ning kolm repositooriumi. Voo kontroller juhib operatsioone ja tagab lõimed, millel lisadel (*extensions*) töötavad. Repositooriume kasutatakse, et hoida meeles infot iga

voofaili kohta, salvestada voofaili sisu ja hoida meeles voofaili kogu teekond [17]. Eelmainitud NiFi arhitektuuri komponendid on toodud Joonisel 3.



Joonis 3. NiFi arhitektuur [17]

NiFi põhineb voo põhisel programmeerimisel (*flow-based programming*). NiFi voo põhikomponentideks on protsessorid ning voo failid (*FlowFile*), mis saadavad andmeid protsessorite vahel [17]. Protsessorid teevad erinevaid toiminguid nagu andmete edastamine ja analüüs erinevate atribuutide alusel. Üksteisega ühendatud protsessorid loovad reaalajas töötava andmevoo. Voole ei ole seatud protsessorite ega nende vaheliste seoste arvu piirangut. Üks näide protsessorite ahelast on toodud Joonisel 4, kus on näha protsessorid ning nende vahelised seosed.



Joonis 4. NiFi voo näidis [18]

Kasutajal on võimalus valida 200[1] erineva protsessori vahel, millest igaühel on üldjuhul üks kindel ülesanne. Lihtsuse mõttes võib protsessoreid võrrelda programmeerimises kasutatavate funktsioonidega. Protsessorite ahela alguses on üldjuhul port või protsessor, mis andmeid loob või sisse võtab. NiFi'l on olemas ka protsessorid, mis suudavad koguda andmeid erinevatest allikatest nagu veebilehed, lokaalsed failisüsteemid, andmebaasid ja servaseadmed. Protsessorid ühendatakse omavahel seostega (*relationships*), mille tüübist olenevalt saadetakse andmed protsessorist edasi järgmisse või jäetakse kõrvale. Seoste kaudu suunatakse andmed, mida ahelas liikumise ajal kutsutakse voo failideks, edasi teistesse protsessoritesse. Protsessorid võimaldavad andmeid erinevalt töödelda ning lõpuks edasi saata järgmisse andmetöötluse platvormi või salvestada vastavalt kasutaja vajadustele faili või andmebaasi. NiFi pakub ka lahendusi erinevatele IoT'ga seotud tehnilistele probleemidele, pakkudes andmete edastamise turvalisust krüpteerimise, nagu SSL, SSH ja HTTPS näol [19].

Apache MiNiFi on NiFi alamprojekt, mis täiendab NiFi põhiprintsiipe andmevoo haldamisel ja keskendub andmete kogumisele nende loomise kohas [5]. MiNiFi viib NiFi põhielemendid andmete kogumise punktile lähemale ja tagab kahesuunalise suhtluskanali. Kuigi MiNiFi'l puudub graafiline liides, on see servaseadmetes eelistatum kui NiFi, kuna see on väiksem ja sellel on võrreldes NiFi'ga väiksem latentsusaeg [20, 21]. Seetõttu on see leidnud laialdast kasutust servaseadmetes. MiNiFi protsessorite skeem tehakse valmis NiFi graafilise liidesega. Valmis skeem salvestatakse *template*'ina, mis konverteeritakse MiNiFi jaoks sobivasse formaati. MiNiFi'l on kaks erinevat versiooni. Üks versioon töötab Java virtuaalmasina põhiselt ning teine C++ keelel. Lisaks graafilise liidese puudumisele on MiNiFi'l vähem protsessoreid kui NiFi'l. Java põhine MiNiFi toetab ligikaudu 70 protsessorit. MiNiFi C++ versioon toetab veel vähem protsessoreid, keskendudes rohkem andmeid koguvatele ja edasi saatvatele protsessoritele.

2.3.2 Apache Edgent

Apache Edgent on raamistik, mida saab lisada servaseadmetesse, võimaldades lokaalset, reaalajas, erinevatest seadmetest ja anduritest pärinevate pidevate andmevoogude analüüsi [22]. Edgenti kasutamisel servaseadmetes väheneb serversisse saadetavate ja salvestatud andmete hulk. Edgent analüüsib, millal tuleb andmed saata edasiseks analüüsiks, teha andmetega toiming või salvestada need serverisse. Edgenti töö põhirõhk on pandud süsteemis probleemide

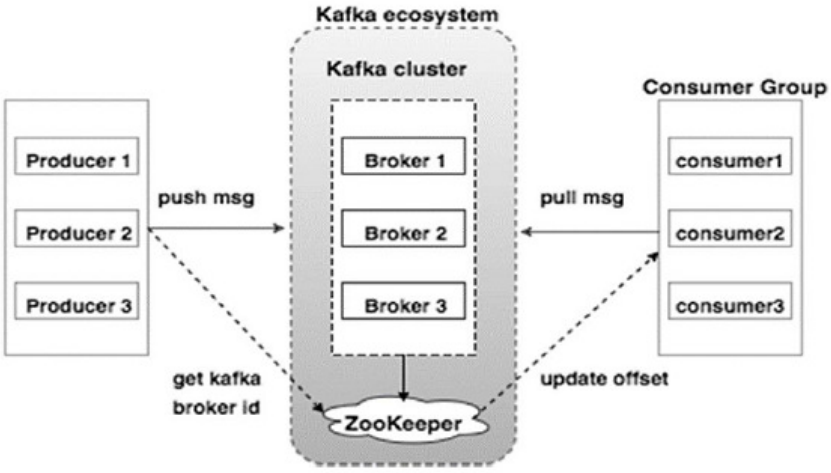
[1] http://nifi.apache.org/docs.html

avastamisele. Näiteks saab Edgenti abil kindlaks teha, kas süsteem töötab väljaspool tavalisi parameetreid. Kui süsteem töötab normaalselt, ei pea andmeid serverisse saatma, kuna see oleks süsteemile lisakoormus. Kui aga Edgent tuvastab probleemi, saab need andmed edasiseks analüüsiks serverile edastada. Põhilisteks rakenduskohtadeks on IoT servaseadmed ning serverid [23].

Edgentiga töödeldakse andmed lokaalselt andmeid koguvas seadmes. Edgenti põhikomponendiks on voog (*stream*), mis on esitatud pideva andmete järjendina (*tuple*) [24]. Servaseadmetest on Edgentit testitud Raspberry Pi'l, Androidil ning Java7 ja Java8 keskkondades. Toetatud kanalid Edgentist andmete edasi saatmiseks on MQTT, IBM Watson IoT Platform ning Apache Kafka [23].

2.3.3 Apache Kafka

Kafka on kõrge läbilaskevõimega ja madala latentsusajaga andmevoo käsitsemise platvorm [6]. Seda kasutatakse sõnumite puhverdamiseks andmevooge tootvate süsteemide vahel. Kafka on disainitud edastama miljoneid sõnumeid sekundis. Sõnumid on jaotatud teemadesse (*topics*), mis indekseeritakse, et tagada nende õige järjestus ning esitatakse võtmeväärtuse paarina [25, 26]. Kafka põhikomponentideks on vahendajad (*brokers*) ehk serverid, tootjad (*producers*), tarbijad (*consumers*) ning koordineerimisteenus nagu Zookeeper. Need arhitektuuri komponendid ja nende seos on näha ka Joonisel 5.



Joonis 5. Kafka architecture diagramm [27]

Kafka klastrid koosnevad vahendajatest, milles igal vahendajal on unikaalne ID. Iga vahendaja sisaldab teema logisid. Tootjad edastavad sõnumeid vahendajatele, ootamata kinnitust sõnumi

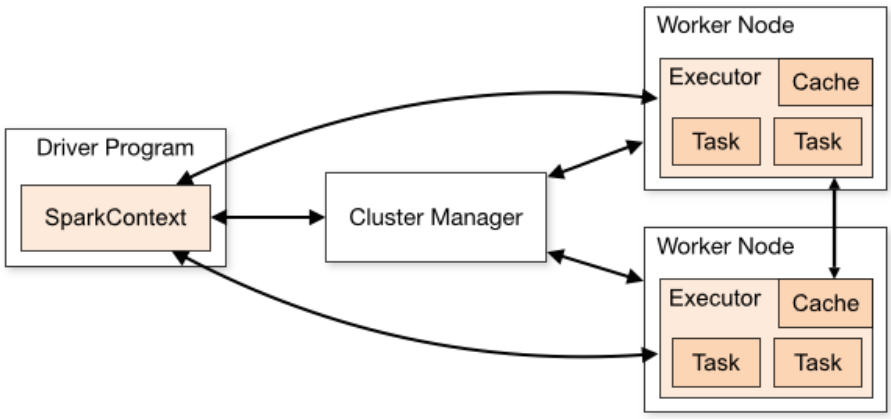
kohale jõudmise kohta. Kuna Kafka vahendajad on olekuta (*stateless*), siis on nende haldamiseks vaja koordineerimisteenust nagu Zookeeper. Samuti peavad tarbijad ise järge pidama juba loetud sõnumite üle. Selleks on igal tarbijal ofset (*offset*), mille järgi tehakse vahet, millised sõnumid on juba loetud [27].

Kafka toetab ühilduvust rakendustega mitmetel platvormidel nagu Java, .NET, PHP, Ruby, ja Python. Samuti on Kafka võimeline saama üle tavapärastest reaalajas andmete töötlemisega seotud probleemidest nagu vähene ribalaius ja andmete edastamise kiirus. Kafka üheks kasutusvaldkonnaks on andmevoo töötlus, kus töötlus tehakse mitmes etapis [26].

2.3.4 Apache Spark

Apache Spark on populaarne avatud lähtekoodiga andmete töötlemise raamistik, mis aitab suurendada suurandmeid analüüsivate rakenduste jõudlust. Rakenduse eesmärk on hallata andmeid, mis on traditsiooniliste andmebaaside jaoks liiga suured või keerulised. Samuti on välja toodud Sparki kiirust võrreldes Apache Hadoopiga, mis on muutnud Sparki eelistatumaks suuremates ettevõtetes nagu Netflix, Yahoo ja Tencent. [7]

Apache Spark koosneb kahest põhikomponendist: draiverist (*driver*) ja täidesaatjatest (*executors*). Draiver teisendab kasutaja koodi ülesanneteks, mis edastatakse töösõlmedesse. Täidesaatjad töötavad töösõlmedes ja täidavad neile määratud ülesandeid. Nagu on Joonisel 6 toodud, on nende kahe vaheliseks suhtluseks vaja mõnda klastri haldurit (*cluster manager*) [7, 28]. Klastri halduritest on toetatud Standalone, Apache Mesos, Hadoop YARN ja Kubernetes [28].



Joonis 6. Apache Spark klastrid [28]

Apache Spark koondab kasutaja andmetöötluskäsklused DAG-isse (*Directed Acyclic Graph*). DAG on Apache Sparki planeerimis kiht, mis määrab, milliseid ülesandeid igas sõlmes ja millises järjekorras täidetakse. Spark võib töötada ka eraldiseisva klastri režiimis, mille puhul on vaja Sparki raamistikku ja Java virtuaalmasinat igale klastris olevale masinale [7].

Apache Sparki keskmes on RDD (*Resilient Distributed Dataset*). See on muutumatu objektide kogum, mida saab jagada arvutiklastri vahel. RDD-dega seotud toiminguid saab jagada klastri vahel ja teostada paralleelse protsessina, mis tagab kiirema ja skaleeritava paralleeltöötlemise. RDD-sid saab luua näiteks tekstifailidest või SQL-andmebaasidest. Suur osa Sparki API-st (*Application Programming Interface*) on üles ehitatud RDD kontseptsioonile, pakkudes sisse ehitatud tuge andmekogumite ühendamiseks, filtreerimiseks ja koondamiseks [29]. Sparkil on olemas API-d Javale, Scalale, Pythonile, R-le ja SQL-le [7].

2.4 Uduarvutuse raamistike jõudlus

Varasemalt on läbi viidud mitmeid uurimustöid, et uurida uduarvutuseraamistike rakendamise mõistlikkust ning võrrelda nende jõudlust. Ühe sellise uurimustöö on avaldanud Yassinea jt [30], kes pakuvad oma töös välja uue platvormi andmete analüüsimiseks. Nende ideeks on kasutada seadme ja pilve vahel olevat udu kihti, mis lubaks keerukamate ja ressursirohkemate andmeprotsesside läbiviimist. Oma töös toovad nad välja selliseks süsteemiks vajalikud eeldused, komponendid ning kihi disaini. Tulemuste saamiseks testisid autorid oma ideed reaalsete andmetega ühest Kanada nutika kodu seadmetest. Tulemused näitasid, et udus andmete töötlemine suurendab platvormi võimekust IoT andmevoogude haldamiseks. Samuti parandab uduandmetöötlus seadmete jõudlust.

Asjade interneti pilvandmetöötlusega (ingl *cloud computing*) tekkinud probleemidega on tegelenud teisedki uurimisrühmad. Singh jt [31] sõnul on pilvandmetöötlus siiani mänginud olulist rolli suurte andmehulkade töötlusel ning asjade interneti tekkimisega on andmehulgad muutunud massiivseks ning on tekkinud vajadus andmete töötlemiseks seadmele lähemal, mis on viinud uduandmetöötluseni. Nende väitel vähendaks udus andmete töötlemine võrguliikluse koormust ning pilvandmetöötlusega kaasnevaid tagasilööke. Nende töö järelduses oli välja toodud, et udus andmete töötlemine aitab üle saada ressursside ja latentsusajaga seotud probleemidest, mis tekiksid kui nende andmete töötlus teha pilves.

Uduarvutusest kirjutasid oma töös Preden jt [13], kelle arvates viib uduarvutus andmetöötluse seadmele veelgi lähemale kui udus andmete töötlus ning muutub olulisemaks seadmete

omavaheline suhtlus. Nende töö eesmärgiks oli hinnata süsteemide eneseteadvuse (*self-awareness*), olukorrateadlikkuse (*situation awareness*) ja tähelepanu (*attention*) olulisust efektiivseks uduandmetöötluseks ja uduarvutuseks.

Kuigi pilvandmetöötlusel on omad eelised siis Yassinea jt [30] sõnul on udus andmete töötlemine kriitilise tähtsusega seadmetele, mis vajavad reaalajas tagasisidet. Selliseid seadmeid leidub nende sõnul nii tervishoiu, korrakaitse ning energiatöötluse valdkonnas.

Nagu Chiang ja Zhang [32] oma töös välja toovad, on paljud seadmed nagu turvakaamerad ja haigete kehas asuvad meditsiinilised seadmed piiratud ressurssidega. Kuna sellised seadmed moodustavad ülekaaluka osa asjade internetist, siis nende hinnangul on nende seadmete jaoks oluline suurem andmehulk töödelda seadme lähedal ja vajadusel saata need edasi pilve. Pisani jt [33] on oma töös välja toonud kaks mudelit, mis aitavad seadmel otsustada, kas andmetöötlus on mõistlikum udus või pilves. Kuigi udutöötlus on kasulik, ei ole see nende arvates alati parim lahendus igale olukorrale, näiteks siis, kui udutöötlus võtaks kauem aega kui andmete pilve saatmine ja seal analüüsimine. Chiang ja Zhang leiavad, et probleemidest olenemata võimaldab udutöötlus rohkematel seadmetel ühel platvormil töötada ilma üksteist segamata. Samuti leidsid nad, et udu on võimeline läbi viima ressursinõudlikke protsesse, mis ei jõua mõnel põhjusel pilve, pikendades sel moel seadme eluiga ning vähendades energiakulu.

Kuigi eelnevalt mainitud töödes kiidetakse udutöötluse kasulikkust ning latentsusaja vähenemist, on Dastjerdi jt [34] ühe udutöötluse probleemina välja toodud andmekaitse. Süsteemid, mis kasutavad udu peavad nende sõnul ka arvesse võtma seadme poolt saadetavate andmete delikaatsust ning erinevates riikides kehtestatud andmekaitseseadusi. Samuti toodi ühe probleemina välja praeguste voogandmete töötlusega tegelevate süsteemide puudused. Enamlevinumad süsteemid nagu Apache Storm ja S4 ei ole nende arvates uduandmetöötluseks piisavalt paindlikud näiteks oma staatilise konfiguratsiooni tõttu. Leiti, et uduandmetöötlus eeldab seadmete dünaamilist lisamist ja eemaldamist, kuna erinevate asukohtade vahel liikuvad seadmed, nagu näiteks nutitelefonid, sisenevad ja lahkuvad võrgust sagedasti.

Karimov jt [35] on oma töös kirjeldanud erinevate Apache süsteemidega tehtud uuringut. Nende uuringu kasutajalood keskendusid ainult veebimängude stsenaariumitele ning ei kasutanud piiratud ressurssidega seadmeid. Oma katseid viisid nad läbi kolme süsteemiga: Apache Flink, Apache Storm ja Apache Spark. Tulemustes kirjeldasid nad, et Apache Spark andis kõige paremaid tulemusi vigaste andmete puhul. Samuti oli sellel madal kuni keskmine latentsusaeg ning sai hakkama ülekoormuse puhul suurest andmehulgast tekkinud ummikute

kõrvaldamisega. Apache Storm andis kehvemaid tulemusi. Ülekoormuse korral katkestas Storm ühenduse andmevooga, mis reaalses situatsioon ei oleks aktsepteeritav. Katsetulemuste kokkuvõttes oli kirjeldatud, et nii latentsusaja kui ka läbilaskvuse poolest on Flink neist kolmest parim.

Samade Apache süsteemidega tehtud katsete tulemusi kirjeldavad on töös Lee jt [36]. Nende töö aluseks oli aastal 2016 teise uuringu käigus läbi viidud katsete tulemuste võrdlus. Nagu ka Karimovi jt [35] töös kasutasid Lee jt sama kolme Apache süsteemi. Need installiti Raspberry Pi 3 arvutisse ning võrreldi nende latentsusaega ja läbilaskvust. Erinevalt Karimovi jt tööst oli Lee jt eesmärgiks testida Apache süsteemide tööd ressursivaestel seadmetel. Nad kasutasid lineaarregressiooni ja naiivset Bayes'i klassifikaatorit masinate ebatavalise käitumise avastamiseks ning kasutuses olevate seadmete eluea ennustamiseks. Testimiseks loodi kaks stsenaariumit, mõlema sihtgrupiks CNC (inglise keeles *Computerized Numerical Control*) põhised masinad. CNC ehk arvprogrammjuhtimine on masinate juhtimine programmiga, mis koosneb tähtedest ja numbritest. Oma katsete tulemustest järeldasid Lee jt, et Apache Storm ei ole sobilik süsteem andmevoo töötluseks piiratud ressurssidega seadmetele. Võrreldes Apache Sparkiga võtab Apache Storm enda alla liiga palju mälu ning tegelikuks töö protsessiks ei jagu piisavalt mälu. Tööst tuli ka välja, et Apache Spark kasutab paralleelset töötlust ning kasutas lineaarregressiooni kiiremini kui Flink. Kokkuvõtteks leiti, et nii Apache Spark kui ka Apache Flink oleksid sobilikud süsteemid andmevoo töötluseks piiratud ressurssidega seadmetele.

Young jt [37] kirjeldavad oma töös andmevoo töötluse plaani IoT ääreseadmetes. Nende plaan kasutab *Hortonworks Data Flow* (HDF) platvormi ning Apache NiFi't ja MiNiFi'd. Põhiliste töövahenditena kasutati veel Apache Kafkat ning Apache Stormi. Apache NiFi tagas graafilise kasutajaliidese ja üle 200 protsessori. See andis võimaluse suunata ja töödelda andmevoost tulevat infot erinevate atribuutide alusel. Toodi välja, et NiFi pakub lahendust paljudele asjade internetiga kaasnevatele probleemidele. Samuti oli välja toodud andmevoo turvatus sisseehitatud SSL, SSH, HTTPS ning kasutajapõhise autentimisega. Young jt testisid oma süsteemi veoautode kiiruse analüüsimisega erinevates ilmastikutingimustes. Nende süsteem jagas NiFi andmed kahte voogu. MiNiFi analüüsis andmeid ning seejärel edastas andmed NiFi serverisse ning sealt edasi Apache Kafkasse ja Apache Stormi. Ääreseadmetena oli kasutatud Raspberry Pi'sid, millesse MiNiFi oli installitud. Tulemuste analüüsis oli välja toodud, et andmete töötlemine ääreseadmetes vähendas edastatava andmevoo suurust ilmastikutingimusi arvestamata üle 90% ning ilmastikutingimusi arvesse võttes üle 80%.

Suurte andmemahtude analüüsimiseks on servaseadmetes kasutatud ka Apache Edgentit. Li jt [38] kasutasid oma 2020 aasta uurimuses Apache Edgentit närvivõrkudel põhinevate mobiilirakenduste kasutamisel tekkiva latentsusaja vähendamiseks. Edgenti kasutati nii staatilises kui ka dünaamilises keskkonnas. Töös toodi välja, et Edgenti kasutamisel Raspberry Pi'l dünaamilises keskkonnas ei tekkinud probleeme ribalaiuse muutumisel. Tulemustes kirjeldati, et staatilises keskkonnas vähendas Edgenti kasutamine latentsusaja pudelikaela efekti.

Varasemate uuringute põhjal selgub, et uduandmetöötlus vähendab ressursivaeste seadmete koormust ning annab paremaid tulemusi kui suurte andmehulkade töötlus pilves.

Kuigi varasemates töödes pole selgelt välja toodud ühte parimat raamistikku, on ühe populaarsema raamistikuna mainitud MiNiFi'd. Sellepärast valiti see raamistik ka käesoleva töö jaoks, et selgitada välja, mis on selle raamistikuga seotud eelised, võimalikud puudused ja kasutamisega seotud probleemid.

Young jt [37] kasutasid oma töös koos MiNiFi'ga Apache Kafkat andmete edastamiseks. Kuna nende töös oli samuti kasutatud Raspberry Pi'sid, siis sobib Kafka ka käesoleva töö raames MiNiFi'st andmete edastamiseks.

Dastjerdi jt [34] tõid välja Apache Stormi staatilise konfiguratsiooni probleemi ning nõrga ülekoormuse taluvuse. Karimov jt [35] testidel andis Spark häid tulemusi vigaste andmete ja suure ülekoormuse puhul. Samuti oli sellel madal kuni keskmine latentsusaeg. Stormi ebasobivust ressursivaestel seadmetel nagu Raspberry Pi kinnitas ka Lee jt [36] töö, tuues välja, et Spark on kiirem kui Storm ja Apache Flink. Seega otsustati, et selle töö raames oleks sobilikum kasutada Apache Sparki servaseadmest saadetud andmete analüüsimiseks.

3. Võrdlustestide disain

See peatükk annab ülevaate käesoleva töö kasutusloost, andmete valikust ning nende seotusest kasutuslooga. Samuti kirjeldatakse testide tegemiseks loodud keskkonna riistvara ja konfiguratsiooni. Lõpuks kirjeldatakse testides kasutatud andmevoo torusid ning testide disaini.

3.1 Kasutuslugu

Igapäevaselt koguvad mitmed servaseadmed nagu Raspberry Pi suurel hulgal andmeid. Kõiki kogutud andmeid pole kas mõistlik või ribalaiuse piiratuse tõttu võimalik edasi saata. Samuti võib andmete edastamisel olla probleemiks privaatsus. Selleks oleks vaja teha osa andmete töötlusest seadmes. Üldjuhul on servaseadmeid palju ning igasse seadmesse eraldi analüüsivat programmi lisada ning hiljem hallata on väga ajamahukas. Selleks on kasutusele võetud raamistik nagu Apache MiNiFi, mis võimaldab hallata mitmeid seadmeid ning lubab läbi viia muudatusi kõigis seadmetes korraga.

Üheks valdkonnaks, kus selliste raamistike kasutamisest võiks suur kasu olla on keskkonnaseire. Erinevad servaseadmed, sealhulgas Raspberry Pi'd, on ühendatud sensoritega, mis andmekogumispunktides koguvad erinevate parameetrite, nagu temperatuur ja õhuniiskus, infot. Käesoleva töö raames on edasisaadetavateks andmeteks valitud temperatuur, CO_2 tase ning keskkonnast salvestatud kaamera kaadrid. Samuti lisatakse nendele andmetel erinevad parameetrid Raspberry protsessori tuumade, temperatuuri ja mälukasutuse kohta. Need on andmed, mida ka tavaliselt servaseadmetes kogutakse ja pilve saadetakse. Kõigi andmete kohene edasi saatmine pilve võib kulutada liiga palju servaseadme ressurssi ning suuremate andmehulkade korral põhjustada seadme ülekuumenemise. Selle ennetamiseks on otstarbekas kasutada mõnda raamistikku, mis võtab võimalikult vähe servaseadme ressursse ning analüüsib, milliseid andmeid on vaja edasi saata.

3.2 Andmete valik

Selleks, et oleks võimalik läbi viia erinevate profiilidega jõudlusteste otsustati eksperimentide jaoks genereerida sünteetilised andmed. See võimaldab muuta andmete saatmise sagedust, andmete mahtu ning läbi viia suurema varieeruvusega eksperimente. Selleks on jõudlustestides kasutatavad temperatuuri- ja CO_2 näidud kirjas tekstifailis ning kaamerapilt salvestatud pildina failisüsteemi. Temperatuuri ja CO2 taseme näidud on üsna tavapärased andmed, mida

keskkonnaseires võiks edastada. Kaamera kaadrit imiteeriv pilt lisati, et näha kuidas NiFi ja MiNiFi saavad hakkama piltide saatmisega.

Tekstifailis on igal real kahe komakohaga temperatuurinäit ning CO_2 taseme näit. Kuna Eestis kasutatakse temperatuuri mõõtmiseks Celsiuse skaalat, siis on ka kõik testandmetes kasutatavad temperatuurid selles skaalas. Temperatuurinäidud on genereeritud vahemikus -20 kuni 20 kraadi nii, et iga järgnev erineb eelmisest 0.01 kuni 0.1 kraadi võrra. CO2 näidud on genereeritud vahemikus 250 kuni 400 ppm nii, et iga järgnev erineb eelmisest 1 kuni 10 ppm võrra.

Pildifailina kasutatakse ühte pilti, mis saadetakse edasi base64 sõne kujul. Selleks, et muuta kogu töötlemiseks edasi saadetava sõnumi suurust, on pildifailist loodud neli erinevat versiooni. Muudetud on pildi resolutsiooni, et edasi saadetava pildi suurus oleks erinev ja saaks hinnata kuivõrd ressursside kulu sõltub pildi mahust.

Kuna andmete töötlus peaks olema võimalikult efektiivne, siis on vaja testimisel hinnata ka süsteemide jõudlust ning seadme ressursside kulu. Sellisteks parameetriteks on sobilikud näiteks CPU- ja mälukasutus. Selleks saadetakse andmevoos edasi ka info Raspberry protsessori tuumade, temperatuuri ning mälukasutuse kohta.

3.3 Riistvara

Ressursivaese servaseadmena kasutatakse 4 tuumaga ja 4 GB mäluga Raspberry Pi 4. Raspberry kasutab U3 kiirusklassi 16 GB SD-kaarti. Raspberry's on Raspian GNU/Linux 10 operatsioonisüsteem.

Edasiseks andmetöötluse raamistike serverina kasutatakse Tartu Ülikooli poolt kasutatava OpenStack[2] pilveteenuse keskkonnas loodud virtuaalmasinat (edaspidi virtuaalmasin). Virtuaalmasinaks valiti m2.small, 8 tuumaga, 32 GB RAM ja 20 GB kettaruumiga Ubuntu 18.04. Virtuaalmasinasse on installeeritud Dockeri kaudu NiFi, mis võimaldab luua andmevoo toru graafilise liidese abil. Samuti kasutatakse virtuaalmasinat Kafka üles seadmiseks ja käivitamiseks.

Testides kasutatakse veel Tartu Ülikoolilt õppetööks renditud HP EliteBook sülearvutit, milles on 64-bitine Windows 10 Enterprise ja 16 GB RAM-i. Arvuti kogu SSD maht on 250 GB. Sülearvutit kasutati testide jaoks Sparki raamistiku üles seadmiseks ning testide tulemuste

[2] https://www.openstack.org/

analüüsimiseks. Sparki raamistikku kasutatakse käesoleva töö raames Kafkaga ühenduse loomiseks ning Kafkast uute sõnumite sisse lugemiseks ja salvestamiseks.

3.4 Seadmete konfiguratsioon

Testide jaoks sobiva keskkonna üles seadmiseks installiti virtuaalmasinasse vabavaralise virtualiseerimisprogrammi Dockeri versioon 19.03.8. NiFi graafilise liidese kasutamiseks installiti Dockerisse NiFi *image*'i versioon 1.11.4. Virtuaalmasinasse installiti Apache Kafka versioon 2.12-2.5.0. Kafka teenuse käivitamiseks on vajalik eelnevalt käima panna hajussüsteeme toetav haldamisteenus Apache Zookeeper. Zookeeperi ja Kafka käivitamiseks on Kafka ametlikul lehel ka juhend[3].

Kafka käivitamiseks on vajalik käivitada Zookeperi server, mille järel saab käivitada Kafka serveri. Raspberry'st edastatud andmete nägemiseks loodi Kafkasse teema (*topic*) nimega 'RaspToSpark'. Eelnevalt saadetud ning reaalajas saadetavate andmete nägemiseks tuleb käivitada ka Kafka consumer, mis näitab kõiki Raspberry'st edasi saadetud andmeridu. Kafka konfiguratsioonifalis tuleb samuti määrata, et Kafka kuulaks ka serverist väljapoolt tulevaid andmepakette.

Kuna Raspberry'st ühenduti VPN tunneliga ülikooli võrku ning Putty SSH ühendusega virtuaalmasinasse, siis oli vajalik, et Kafka port 9092 oleks virtuaalmasinas avatud. Vaikimisi kuulab Kafka ainult lokaalseid ühendusi. Seetõttu tuli Kafka konfiguratsioonifailis määrata, et see port kuulaks väljastpoolt tulevaid ühendusi. Selleks tuli Kafka `server.properties` failis lisada või muuta kahte rida.

`config/server.properties` juures muuta rida:

```
advertised.listeners=PLAINTEXT://<virtuaalmasina_ip_aadress>:9092
```

`config/producer.properties` juures muuta rida:

```
bootstrap.servers= <virtuaalmasina_ip_aadress>:9092
```

Raspberry Pi'le installiti Pythoni versioon 2.7.16. MiNiFi Java versiooni kasutamiseks installiti versioon 0.5.0 ja sellele vastav MiNiFi-toolkit 0.5.0. NiFi's tehtud *template*'i konverteerimiseks YML-failiks on vaja Javat. Kuna MiNiFi toetab ainult Java 8 versiooni, siis tuli Raspberry'sse installida Java versioon 1.8.0_212. Käesoleva töö raames tekkis Raspberry's probleem MiNiFi failide õigustega, mille tõttu tuli kõikidele MiNiFi failidele määrata protsessi käivitava kasutaja

[3] https://kafka.apache.org/quickstart

õigused. Selleks kasutati käsku `sudo chown -R -L kasutaja:kasutaja alamkaust/kuhu/installiti/minifi`. Selle käsuga muudetud failide õigused ei säilinud peale Raspberry taaskäivitamist ning seega tuli käsk anda korduvalt.

Raspberry Pi `/etc/hosts` faili tuli lisada ka rida virtuaalmasinas oleva Dockeri NiFi *image*'i nimelahendamiseks. RemoteGroup tegemisel suunatakse andmed MiNiFi'st edasi mitte ip aadressi, vaid NiFi *image*'i nime alusel. Seetõttu tuli teha `hosts` failis ümbersuunamine.

Samuti tuli Raspberry Pi'sse installida sysstat pakett, et oleks võimalik `mpstat` käsuga Raspberry protsessori tuumade seisu koht päringuid teha.

Sülearvutisse installeeriti PyCharm community versioon. Sparki raamistiku üles seadmiseks kasutati PyCharm community versiooni, milles käivitati testides kasutatavat Kafkaga ühenduses olevat Pythoni koodi. PyCharmi seadistamisel võeti aluseks Tartu Ülikooli Arvutiteaduse instituudi Pilvetehnoloogia kursuse praktikumi juhend[4].

Et vältida WiFi ühendusest tulenevaid anomaaliaid, on nii Raspberry kui sülearvuti on ühendatud internetivõrku kaabliga.

[4] https://courses.cs.ut.ee/2020/cloud/spring/Main/Practice9

3.5 Võrdlustestide andmevood

Selles peatükis antakse ülevaade võrdlustestides kasutatud andmevoo torude ülesehitusest ning täpsemast konfiguratsioonist. Iga andmevoo toru puhul on välja toodud iga elemendi kirjeldus ning tööülesanne selles torus.

3.5.1 Pythoni kood Raspberry Pi's – Kafka – Spark – Dataframe

Selles andmevoo torus käivitatakse Raspberry käsurealt Pythoni kood, mis loeb andmeid teatud intervalliga ning saadab koos Raspberry süsteemi parameetritega edasi Kafkasse. Vastavast Kafka teemast kuulab uusi sõnumeid Spark, mis salvestab need CSV-faili edasiseks analüüsimiseks.

3.5.1.1 Pythoni kood Raspberry Pi's

Raspberry's käivitatakse Pythoni kood, mis loeb failist CO_2 ja temperatuurinäite ning edastab need Kafka teemasse 'RaspToSpark'. Kafkasse andmete saatmiseks on vajalik installida kafka-python, et koodis saaks luua KafkaProduceri, mis toimib Kafkale andmete saatjana. KafkaProducer luuakse koodis uue isendina, millel on määratud sihtserveri ip aadress ja port, mida serveris käivitatud Kafka kuulab. Samuti tuleb andmed enne Kafkasse saatmist viia JSON-i formaati.

Koos temperatuurinäitude, CO_2 taseme näidu ja pildiga saadetakse ka Raspberry süsteeminäidud nagu mälukasutus, protsessori temperatuur, protsessori tuumade kasutus keskmiselt ning iga tuuma kohta eraldi. Nende parameetrite kättesaamiseks on Pythoni koodis kasutatud `mpstat -P ALL 1` käsku. Lisaks pannakse andmevoosse ka protsessori temperatuur ning mälukasutus. Nendeks on Pythoni koodis ja skriptis kasutatud vastavalt `vcgencmd measure_temp` ja `free` käske. Et koguda infot andmete edastamise kiiruse kohta, lisatakse Pythoni koodis ja skriptis andmete saatmist tähistav ajatempel. Andmete saabumist andmevoo ahela lõppu tähistav ajatempel lisatakse Sparkis.

3.5.1.2 Kafka

Kuna Pytoni koodiga ei ole võimalik andmeid saata Raspberry'st otse Sparki, siis selleks on vajalik nende vaheline andmetoru. Selleks kasutatakse käesolevas töös Kafkat. Kafkas ei töödelda selle töö raames andmeid, kuid see võimaldab näha reaalajas andmete saatmist.

3.5.1.3 Spark

Selle töö raames kasutatakse Sparki käivitamiseks Pythoni skripti, mis peale Kafka teemaga ühenduse loomist jääb kuulama teemasse saadetavaid sõnumeid. Spark kogub need sõnumid puhvrisse ning iga kasutaja poolt määratud sekundi tagant töötleb puhvrisse kogunenud andmed. Spark jätab meelde, kust viimati andmeid võeti ja jätkab sealt. Kui Kafkasse andmeid ei saadeta, siis on Spark ooteolekus ning jätkab andmete kuulamist Kafka teemast.

Töödeldud andmed salvestatakse CSV-faili, et neid oleks võimalik lihtsasti visualiseerida ning kasutada edasiseks analüüsiks.

3.5.1.4 Dataframe

Sparkis kasutatava Pythoni koodiga loetakse andmed Kafkast reaalajas sisse ning muudetakse JSON formaadist kasutajale paremini loetavale tabeli kujule. Sparkis lisatakse sisse loetud andmeridadele ajatempel andmete töötlemise ajal. Kogu Sparki poolt sisse loetud info salvestatakse CSV-failidesse. Kuna andmeid loetakse sisse readStream käsuga ning salvestatakse kohe faili, siis on tulemuseks mitu andmefaili.

3.5.2 MiNiFi (Java) – NiFi - Kafka - Spark - Dataframe

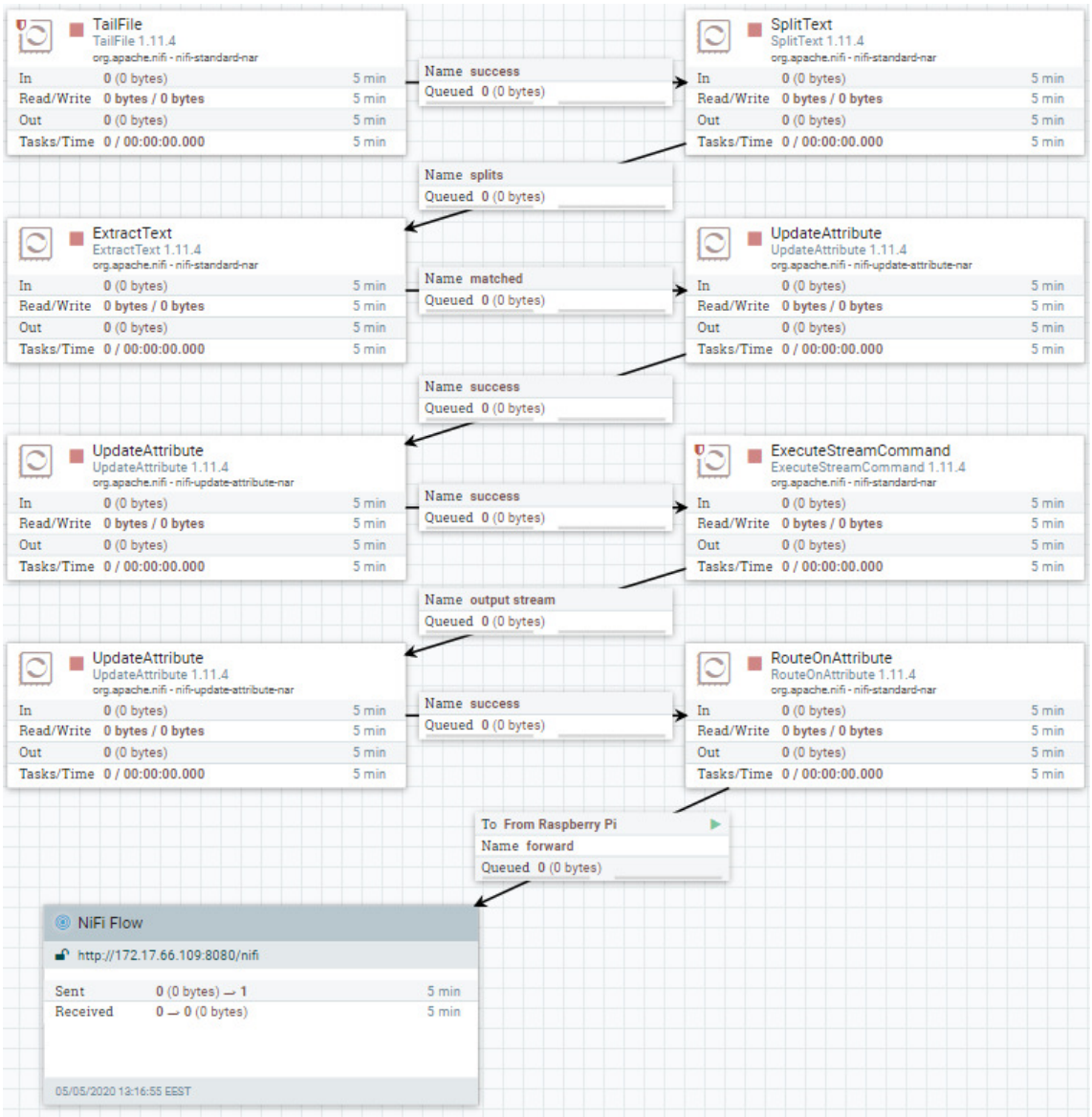
Selles andmevoo torus kasutatakse MiNiFi Java versiooni, mis saadab andmed edasi NiFi'sse. NiFi kaudu suunatakse andmed ilma töötluseta edasi Kafka teemasse, millest Spark uusi andmeridu oma puhvrisse kogub ning töötleb.

3.5.2.1 MiNiFi ja NiFi

MiNiFi's andmetöötluseks kasutatud protsessorite ahel loodi Apache NiFi graafilise kasutajaliidese abil. Vajalikud protsessorid salvestati *template* formaadis ja hiljem konverteeriti MiNiFi jaoks YML-failiks. NiFi's loodud *template* salvestati arvuti kettale XML-failina, mis seejärel Raspberry SD-kaardile salvestati. Raspberry's kasutati MiNiFi config.sh faili, et *template* konverteerida MiNiFi jaoks sobivale kujule. Õige vorminguga *template*'i fail tuli liigutada MiNiFi ./conf alamkataloogi ning nimetada ümber config.yml-ks. Kuna MiNiFi kasutab ainult konkreetse nimega faili, siis teistsuguse nimega faile MiNiFi ei aktsepteeri.

Sellest tulenevalt saab conf kaustas olla ainult üks *template* korraga, kuid *template*'i sees võib olla mitu andmetöötlusvoogu.

Template tehti valmis kasutades NiFi graafilist veebiliidest. Protsessorite ahelas kasutati TailFile, SplitText, ExtractText, UpdateAttribute ExecuteStreamCommand, ja RouteOnAttribute protsessoreid. Ahela lõppu lisati Remote Process Group, mis edastab andmeid NiFi'sse. Loodud MiNiFi andmevoog on näha järgneval Joonisel 7.



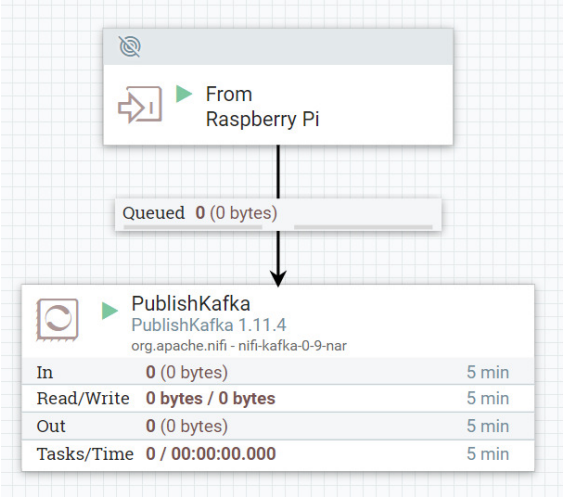
Joonis 7. NiFi's loodud protsessorite ahel, mida MiNiFi kasutab

TailFile protsessor loeb failist uusi andmeridu ning juhul kui loeti sisse rohkem kui üks rida, siis **SplitText** jagab need üksikuteks ridadeks. Sisse loetud andmeread on voo failina, aga andmete analüüsimiseks on vaja need NiFi atribuutidesse. **ExtractText** protsessoris luuakse

uus atribuut *data*, millele antakse iga sisse loetud failirea väärtus. Esimeses **UpdateAttribute** protsessoris jagatakse muutuja *data* väärtus kaheks: 'temperatuur, CO_2 näit' ning 'reanumber'. Kuna samas protsessoris ei ole võimalik teha edasisi muutujateks jagamisi, siis selleks on järgmine **UpdateAttribute**, mis jagab 'temperatuur, CO_2 näit' väärtuse kahte muutujasse. **ExecuteStreamCommand** käivitab Pythoni skripti, mis saab käsurea argumentidena sisse temperatuuri ja CO_2 taseme näidu ning reanumbri. Skriptis käivitatakse käsud, millega saadakse protsessori info ning lisatakse kõik vajalikud andmed sõnastikku. Järgnevas **UpdateAttribute** protsessoris arvutatakse iga rea temperatuuri ja CO_2 väärtuste vahe eelneva Kafkasse saadetud väärtusega, mille põhjal **RouteOnAttribute**'sse kirjutatud reegel otsustab, kas andmerida saadetakse edasi või jäetakse kõrvale. Kõik edasi saadetavad andmeread edastatakse **Remote Process Group**'i, mis saadab andmed MiNiFi'st NiFi'sse.

MiNiFi Java versioon ei toeta mõnda protsessorit, mida oleks olnud loogilisem kasutada. Näiteks ei saanud MiNiFi'st otse PublishKafka protsessoriga andmeid saata, vaid seda pidi läbi NiFi tegema. Samuti tekkis olukord, kus ahelas oli kasutatud ExecuteScript protsessorit kuni selgus, et MiNiFi ei toeta seda ja see asendati ExecuteStreamCommand protsessoriga. Antud protsessorite ahelas sai PublishKafka protsessori asemel kasutati Remote Process Group'i.

Selleks, et NiFi saaks MiNiFi poolt saadetud andmed kätte, on vaja lisada InputPort, kuhu Remote Process Group andmeid saadab. Port suunab andmed edasi PublishKafka protsessorisse, mis andmed edasi Kafka teemasse saadab. NiFi's olev InputPort ja PublishKafka protsessor on näha Joonisel 8.



Joonis 8. NiFi InputPort ja PublishKafka protsessor

MiNiFi's olev ExecuteStreamCommand käivitab Pythoni skripti, milles küsitakse protsessori parameetreid. Samuti loetakse skriptis sisse pildifail ning lisatakse ajatempel. Skript saab käsurea argumentidena ette ExecuteStreamCommand protsessoris määratud muutujad, mille väärtusteks on temperatuuri ja CO_2 taseme näit ning reanumber. Koodis lisatakse kõik muutujad Pythoni sõnastikku ning saadetakse edasi voofailina `print` käsu abil.

Andmed, mida TailFile protsessor sisse loeb genereeritakse reaalajas teise Pythoni skriptiga. Selles skriptis määratakse kui palju ridu ning millise ajalise intervalliga need read andmefaili lisatakse. Selle skripti käivitamiseks pole kasutatud eraldi protsessorit, vaid kood käivitatakse käsurealt. Selleks, et andmeread kirjutataks õigesse faili tuleb failinimi anda ette käsurea argumendina. Selle andmevoo toru jaoks andmete genereerimisel kasutati Pythoni skripti käivitamiseks järgnevat käsku:

```
python –u genereeri_andmed.py >> tee/sihtfailini/sihtfail.txt
```

3.5.2.2 Kafka, Spark ja Dataframe

Kafka ja Sparki konfiguratsioonis ei muudetud midagi, kui hakati MiNiFi poolt saadetud andmeid koguma. Andmed saadeti samasse Kafka teemasse ning loeti Sparkis sisse. Samuti salvestatakse andmed samal viisil CSV-faili ning analüüsitakse eraldiseisva Pythoni koodi abil.

3.6 Testide disain

Selle töö raames disainitud testid koosnevad mitmest alameksperimendist, mille parameetreid muudetakse.

Testide läbiviimisel kasutatakse põhiliste andmetena temperatuurinäitu, CO2 taseme näitu ning pildifaili, mis imiteerib kaamera kaadrit. Pildifail viiakse base64 vormingusse ning lisatakse koos pildi suuruse parameetriga andmetele. Põhiandmetel lisaks saadetakse edasi ka Raspberry Pi protsessori tuumade näidud. Nendeks valiti kõigi tuumade keskmise ja iga nelja tuuma usr, iowait ning idle veerud, et saada parem ülevaade kuidas erinevad eksperimendid mõjutavad Raspberry jõudlust.

Alameksperimentides muudetavateks parameetriteks valiti saadetavate andmeridade arv, andmete sisselugemise intervall ning andmevoole listava pildifaili suurus.

Andmeridade arv varieerub, et intervallist sõltuvalt ei läheks ühe eksperimendi kestus liiga pikaks. Andmete sisselugemise intervalli muudetakse, et võrrelda, kas andmete edastamisel

pikema ajavahe jooksul on näha muutust Raspberry ressursside kasutuses. Andmetega koos edastatava pildifaili suurust muudetakse, et saadetava andmesõnumi mahtu suurendada ning võrrelda, kuidas see mõjutab andmete edastamise kiirust.

4. Testide kirjeldused ja tulemused

Selles peatükis antakse ülevaade iga testi raames tehtud eksperimentidest ning nende tulemustest.

Selleks, et varieeruvus oleks piisavalt suur valiti nelja erineva suurusega pildid. Suurima pildi maht oli kettal ligikaudu 700 kB. Suuremaid pilte ei kasutata sellepärast, et 1MB failiga testides ei jõudnud nii Pythoni koodis olev KafkaProduceri ega ka MiNiFi poolt saadetud andmed Kafkasse. Põhjuseks on Kafka sõnumite suuruse piirang. Sellest andis märku ka NiFi's olev PublishKafka protsessor, mis andmeid edasi ei saatnud ning protsessori ülemises paremas nurgas veateadet kuvas. Pythoni koodiga suurt pildifaili saates veateadet ega muud märguannet ei tekkinud. Testimisel on suurimaks pildiks valitud 735 kB pildifail, kuna peale selle viimist base64 sõne kujule on pildi suuruseks 980 kB.

Latentsusaja arvutamiseks tuli nii Raspberry kui sülearvuti panna aega sünkroniseerima samast kellaserverist. Selleks valiti 0.debian.pool.ntp.org server. Kuigi Raspberry Pi ja sülearvuti kasutavad sama kellaserverit, tekkis probleem sülearvuti kellaga, mis oli Raspberry kellaga võrreldes rohkem kellaserveri ajast mööda. Raspberry Pi kell oli testide tegemise jooksul piisavalt täpne: kellaserverist 0.001 sekundit taga. Sülearvuti kell erines testide ajal kellaserverist 0.4 kuni 1.3 sekundit. See oli liiga suur vahe ning seetõttu on testide tulemuste ja eksperimentide analüüsimisel võetud seda vahet arvesse. Selleks on Sparkis pandud ajatemplile see erinevus lisatud, et vältida negatiivseid latentsusaegu ja saada võimalikult täpne testi tulemus. Kellade erinevust kontrolliti perioodiliselt iga testi puhul. Raspberry's kasutati selleks `ntpdate -qv 0.debian.pool.ntp.org` käsku. Sülearvutis kasutati administraatorina käivitatud käsuviibas käsku `w32tm /stripchart /Computer:0.debian.pool.ntp.org /sampless:5`. Edaspidi viidatakse sülearvuti kella ja kellaserveri erinevusele terminiga ajanihe.

Käsuga `mpstat` päringu tegemine võtab nii Pythoni koodis kui MiNiFi poolt käivitatavas skriptis aega ligikaudu 1 sekund. Lisaks võtavad aega ka käsud `free` ja `vcgencmd measure_temp`. Kokku kulub päringute tegemiseks ligikaudu 1.1 sekundit.

Protsessori temperatuur ja mälukasutus testide käigus oluliselt ei muutunud. Temperatuur muutus vahemikus 39 kuni 44 kraadi. Mälukasutus muutus vahemikus 10 kuni 12%. Seetõttu pole nende parameetrite väärtusi testides eraldi välja toodud.

Testides kasutatud koodid on kättesaadavad Lisa 1. Repositoorium alt.

4.1 Test 1 andmete sisselugemine ilma ajalise viiteta

Test 1 konstrueeriti selleks, et võrrelda kahe test andmevoo jõudlus üldiselt.

Pythoniga eksperimente tehes oli sülearvuti ajanihe kellaserverist 0.42 sekundit. MiNiFi'ga testimisel oli ajanihe kellaserverist 1.3 sekundit. Latentsusaja korrektseks arvutamiseks on ajanihe lisatud Sparki ajatemplile.

Eksperimentides kasutati 100-realist andmefaili, kus olid temperatuurinäidud 0.3 kraadise vahega. See tagas, et MiNiFi saadab igal juhul iga andmerea edasi. Testiks valiti pildid, mille suurusteks on 21 kB, 204 kB, 352 kB ja 753 kB.

Testi tulemusteks on ajad, mis kuluvad Pythonil andmete sisselugemiseks ja saatmiseks ning Sparkil andmete vastuvõtmiseks kulunud aeg. Nende aegade põhjal saab arvutada, kui kiiresti suudab Python keskmiselt andmeid sisse lugeda ja saata ning kui kiiresti suudab Spark neid vastu võtta. Samad andmed on esitatud ka MiNiFi'ga tehtud testide kohta Tabelis 1.

Tabel 1. Pythoni ja MiNiFi jõudluse hindamine

	Python				MiNiFi			
	Kokku kulunud aeg sekundites		1 rea saatmise aeg sekundites		Kokku kulunud aeg sekundites		1 rea saatmise aeg sekundites	
Pildi-fail (kB)	Raspberry	Spark	Raspberry	Spark	MiNiFi	Spark	MiNiFi	Spark
21	7.8	8.5	0.08	0.09	10.6	10.4	0.11	0.11
204	49.9	50.6	0.5	0.51	0:9.5	26.4	0.1	0.27
352	84.00	85.0	0.85	0.86	11.0	48.8	0.11	0.49
735	172.00	172.0	1.74	1.74	26.1	83.3	0.26	0.84

Alameksperimentides saadeti 100 pilti ilma sisendi lugemise ajalise viiteta.

Pythoni puhul on Raspberry's ühe rea sisselugemise ja saatmise aeg peaaegu sama kui Sparki ühe andmerea vastu võtmise aeg. Põhjus on selles, et Pythoni koodis ei loeta järgmist pildifaili sisse enne, kui eelmine on ära saadetud.

MiNiFi puhul loetakse sisse kõik pildifailid nii kiiresti kui jõutakse ning igale pildile pannakse ajatempel. Seetõttu on Tabelis 1 MiNiFi veerus ühe pildi sisselugemiseks kulunud aeg, mitte pildi saatmise aega. Reaalset saatmisele kulunud aega saab hinnata Sparki vastuvõtmise aja järgi. MiNiFi sisselugemise ajad ja Sparki vastuvõtmise ajad erinevad oluliselt, kuna MiNiFi's ei ole pildi sisselugemine ja pildi Sparki saatmine omavahel seotud.

21 kB pildifaili puhul oli ühe faili saatmisele kulunud aeg Pythoni ja MiNiFi puhul sarnane, vastavalt 0.09 ja 0.11 sekundit. Suuremate pildifailide puhul saatis MiNiFi faile kiiremini.

See test võeti aluseks, et järgnevates testides valida andmete sisselugemise intervallideks 1.5 ja 3 sekundit. 1.5 sekundilise intervalli kasutamise eesmärgiks oli testida nende pildifailide saatmist, mille saatmiseks kulus Test 1 raames alla 0.5 sekundi. 3 sekundilise intervalli sisse peaksid mahtuma kõikide Test 1 raames kasutatud pildifailide saatmised.

4.2 Test 2 pika ajalise viite põhjuste uurimine

Selle testi jaoks kasutati MiNiFi Pythoni skripti. Testi raames plaaniti teha neli alameksperimenti, kasutades 200 realist andmefaili, kust loetakse sisse andmeid 1.5 sekundi tagant. Eksperimentide käigus oli aeg-ajalt näha Kafka *consumer*is sõnumite vahelise aja olulist suurenemist. Sõnumite sisselugemisel failist ei oleks tohtinud tekkida suuremat viidet, kui see, mis oli määratud. Enamasti oli sõnumite vaheline aeg õige, kuid vahepeal tekkis 4-5 sekundiline paus kuigi ei oleks tohtinud nii pikka vahet olla.

Anomaalia põhjuse uurimiseks tehti alameksperimente nii, et samal ajal oli käsureal näha reaalajas `mpstat` käsk. Väljundist oli näha, et iowait veeru väärtused olid väga suured ja idle veeru väärtused olid mitu sekundit nullis. Seda on näha ka Joonisel 9. Sel ajal kui idle veeru väärtused olid nullid, ei toimunud ka andmete saatmist.

11:06:05 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
11:06:06 AM	all	3.31	0.00	1.78	67.43	0.00	0.00	0.00	0.00	0.00	27.48
11:06:06 AM	0	8.08	0.00	2.02	30.30	0.00	0.00	0.00	0.00	0.00	59.60
11:06:06 AM	1	2.06	0.00	1.03	72.16	0.00	0.00	0.00	0.00	0.00	24.74
11:06:06 AM	2	3.06	0.00	3.06	93.88	0.00	0.00	0.00	0.00	0.00	0.00
11:06:06 AM	3	0.00	0.00	1.01	73.74	0.00	0.00	0.00	0.00	0.00	25.25
11:06:06 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
11:06:07 AM	all	2.51	0.00	1.25	96.24	0.00	0.00	0.00	0.00	0.00	0.00
11:06:07 AM	0	1.00	0.00	2.00	97.00	0.00	0.00	0.00	0.00	0.00	0.00
11:06:07 AM	1	5.05	0.00	1.01	93.94	0.00	0.00	0.00	0.00	0.00	0.00
11:06:07 AM	2	4.00	0.00	1.00	95.00	0.00	0.00	0.00	0.00	0.00	0.00
11:06:07 AM	3	0.00	0.00	1.00	99.00	0.00	0.00	0.00	0.00	0.00	0.00
11:06:07 AM	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
11:06:08 AM	all	1.52	0.00	0.51	97.98	0.00	0.00	0.00	0.00	0.00	0.00
11:06:08 AM	0	2.04	0.00	1.02	96.94	0.00	0.00	0.00	0.00	0.00	0.00
11:06:08 AM	1	1.03	0.00	0.00	98.97	0.00	0.00	0.00	0.00	0.00	0.00
11:06:08 AM	2	2.00	0.00	1.00	97.00	0.00	0.00	0.00	0.00	0.00	0.00
11:06:08 AM	3	0.99	0.00	0.00	99.01	0.00	0.00	0.00	0.00	0.00	0.00

Joonis 9. Anomaaliat illustreeriv lõik `mpstat` käsu väljundist

Kahtlustati, et pildifaili sisselugemine võtab liiga palju ressurssi. Selle testimiseks tehti uued eksperimendid, kus kasutati 6kB pildifaili. Samuti tehti samad eksperimendid ilma pildifailita.

Ka nende eksperimentide ajal oli näha `mpstat` käsu väljund, mis näitas, et iowait veeru väärtused ei vähenenud.

Eelnevate eksperimentide jooksul oli tekkinud vajadus uurida logifailide sisu, millest oli näha, et logifaili kirjutatakse ka info iga saadetud sõnumi kohta. MiNiFi `boostrap.conf` failis oli näha, et logimise parameetri `nifi.minifi.status.reporter.log.level` väärtus oli seatud 'INFO'-ks aga välja kommenteeritud. Sellest tekkis mõte, et võib-olla kulutab MiNiFi palju aega logifailidesse kirjutamisele, kui vaikimisi on logimise tase 'INFO'.

Sellest tulenevalt konstrueeriti Test 3, mille alameksperimentides kasutati samu parameetreid, mis Test 2-s, kuid mille jooksul on MiNiFi logimine seatud 'WARN' tasemele.

4.3 Test 3 andmete sisselugemise intervallide mõõtmine 'WARN' tasemega

Esialgu oli testides edastatavateks protsessori parameetriteks valitud ainult kõikide tuumade keskmine ja ka iga nelja tuuma kasutus eraldi. Selle parameetri väärtus saadi `mpstat` käsu väljundi idle veeru väärtuse sajast lahutamisega. Seetõttu polnud näha, kuhu täpselt ressursid kuluvad. Edasiste testide paremaks analüüsimiseks peeti tarvilikuks lisada andmereaga edastatavasse infosse parameetrid iga protsessori tuuma usr, iowait ja idle veeru väärtustega.

Alameksperimentides kasutati MiNiFi'd ning 200-realist andmefaili, millest loeti andmeid iga 1.5 sekundi tagant. Pildifailina kasutati sama 6 kB pildifaili. Samuti tehti eksperiment ilma pildifailita. Sülearvuti kõikide alameksperimentide jooksul oli sülearvuti kella ajanihe kellaserverist 0.42 sekundit.

Tulemustest oli näha, et logimise tasemega 'INFO' oli tegelik andmete sisselugemise intervall 1.59 kuni 4.91 sekundit. Tasemega 'WARN' oli intervall 1.59 kuni 4.09 sekundit. Vastavad väärtused on välja toodud Tabelis 2.

Tabel 2. INFO ja WARN tasemega tehtud võrdlustestide andmed

	INFO tase		WARN tase	
	iowait (%)	Sisselugemise intervall (s)	iowait (%)	Sisselugemise intervall (s)
min	0	00:01.59	0	00:01.59
max	43.26	00:04.91	45.18	00:04.09
keskmine	1.91	00:01.82	1.66	00:01.77

Seega muutusid pausid andmete saatmise vahel natukene lühemaks. Samuti vähenesid iowait veeru väärtused. Esialgne keskmine iowait väärtus oli 1.91%, kuid 'WARN' tasemega langes keskmine väärtus 1.66% peale. Erinevus pole suur, aga tulemus oli siiski veidi parem.

NiFi kasutab failisüsteemi oma repositooriumide hoidmiseks [39]. See võib olla põhjuseks, miks iowait'i väärtus muutub aeg-ajalt kõrgeks sõltumata edastatavate andmete mahust.

Kuna tulemused siiski paranesid, siis on ka edasised testid tehtud nii, et MiNiFi logimise tase on 'WARN'.

4.4 Test 4 MiNiFi 1.5 sekundilise intervalliga andmete sisselugemine

Test 4 jaoks kasutati MiNiFi'd. Kõigis alameksperimentides salvestati andmefaili iga 1.5 sekundi järel uus andmerida. Kokku lisati faili 200 andmerida, kust oli temperatuuri ja CO_2 taseme näidud. Iga alameksperimendi jaoks kasutati erineva suurusega pildifaile. Pildifailide suurused oli 21 kB, 204 kB, 352 kB ja 735 kB. Sülearvuti ajanihe võrreldes kellaserveriga oli -0.97 sekundit.

Tabelis 3 ja Tabelis 4 kirjeldatud väärtused mõõdeti 21 kB pildifaili saatmisel. Testi tulemuste illustreerimiseks valiti 21 kB pildifailiga tehtud eksperiment, sest Test 1 jooksul mõõdetud keskmine ühe pildifaili saatmise aeg oli 0.11 sekundit. Seega oleks pidanud pildifailide saatmise aeg koos süsteemi parameetrite päringu ajaga mahtuma 1.5 sekundi sisse.

Tabel 3. Andmete sisselugemise ajad 21 kB pildifaili puhul

	Andmete sisselugemise intervall (s)
min	00:01.10
max	00:08.08
keskmine	00:01.75

Tabelist 3 on näha, et andmete sisselugemise intervall muutus vahemikus 1.1 kuni 8.08 sekundit. Seega kulus mõne andmerea sisselugemiseks mitu korda rohkem aega kui eksperimendis määratud 1.5 sekundit. Andmete sisselugemise intervalli arvutamisel kasutati ka iga andmereaga kaasas olevat reanumbrit. Reanumbrid andsid ülevaate, millised andmeread edasi saadeti ning millised välja filtreeriti MiNiFi's rakendatud reeglite abil.

Selle eksperimendi andmeid lähemalt uurides on näha, et 8.08 sekundi jooksul ei lugenud MiNiFi sisse ühtegi andmerida. Need andmeread on välja toodud ka Tabelis 4, kus on näha, et sisse loetud ridade numbrid on järjestikused. Seega ei olnud antud juhul pika ajalise viite põhjuseks ridade välja filtreerimine.

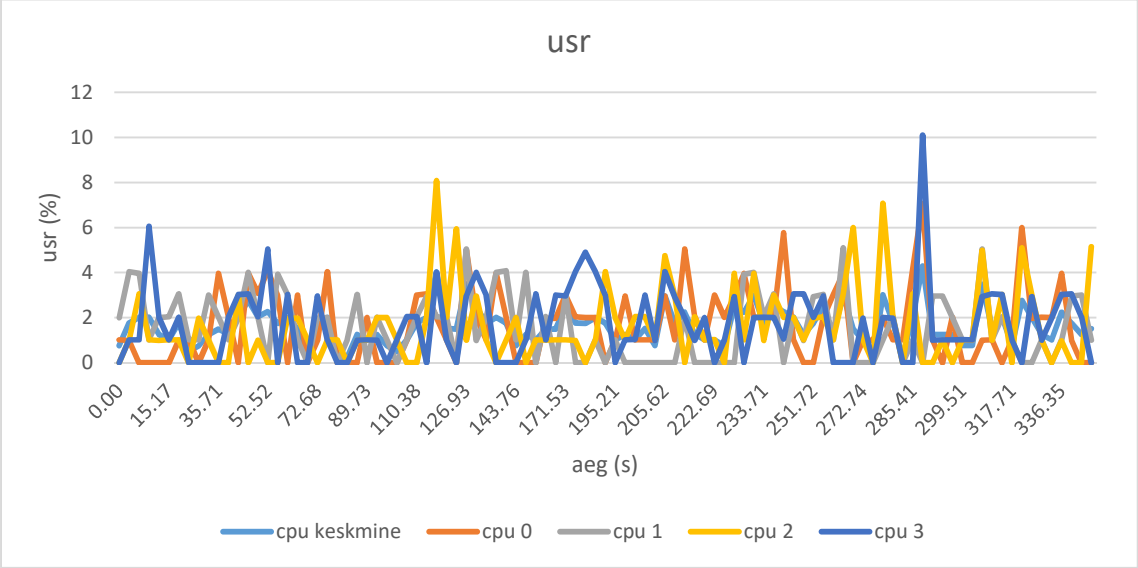
Tabel 4. Näide andmete sisselugemisel tekkinud pausist

MiNiFi ajatempel	Rida	Sparki ajatempel ajanihkega	Latentsusaeg	Sisselugemise intervall (s)
22:21:05.02	66	22:21:05.28	00:00.253	00:01.34
22:21:13.11	67	22:21:13.35	00:00.242	00:08.08

Eksperimendis kirjutati andmefaili uusi ridu iga 1.5 sekundi tagant. Kui MiNiFi oli liiga hõivatud, et uusi andmeid sisse lugeda, siis järgmisel korral andmefailist uusi ridu lugedes võis TailFile protsessor lugeda korraga sisse rohkem kui ühe rea. Sellisel juhul töötles ja edastas MiNiFi sisse loetud andmeridu nii kiiresti kui võimalik. Andmeridade töötlemisel tehakse päring ka protsessori parameetrite teada saamiseks. See päring võtab aega ligikaudu 1.1 sekundit. Juhul kui TailFile luges sisse mitu rida korraga, siis minimaalne andmete töötlemise ja edasi saatmise intervall on 1.1 sekundit.

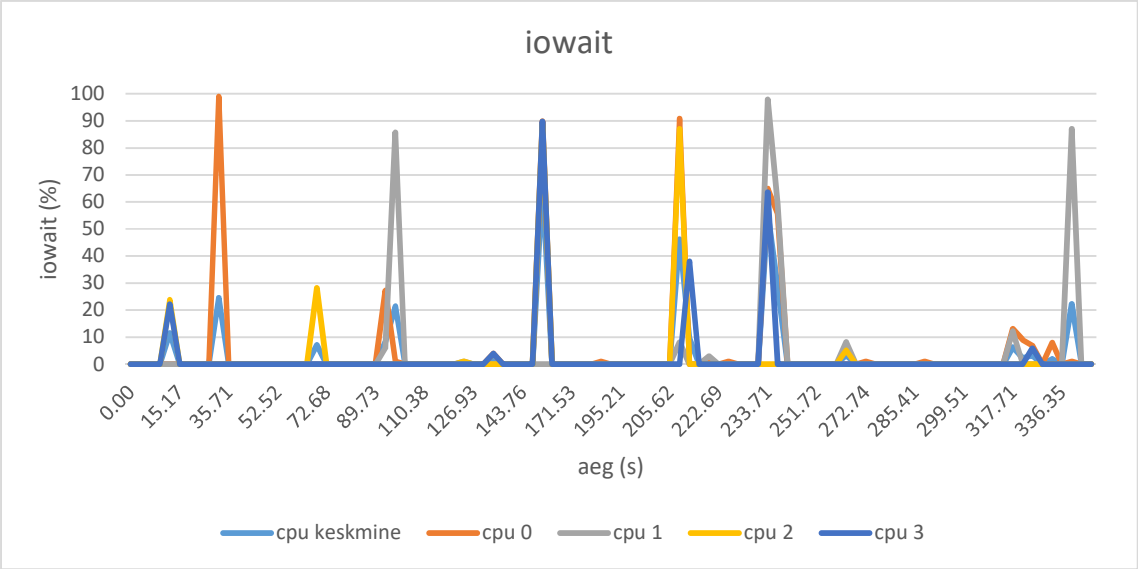
Järgneval kolmel joonisel on graafikud, mis näitavad alameksperimendi jooksul mõõdetud protsessori tuumade usr, iowait ja idle väärtusi `mpstat` käsust.

Joonisel 10 olevalt graafikult on näha, et protsessori tuumade usr väärtused ei tõuse üle 10%. See näitab, et protsessorite tuumade koormus on suhteliselt madal.



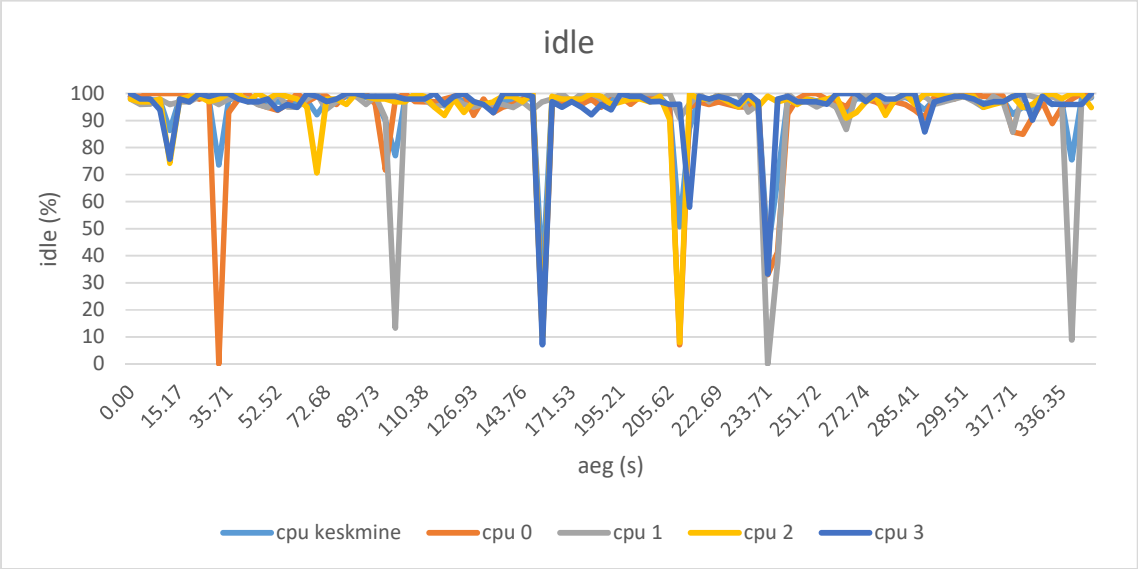
Joonis 10. Graafik usr veergude väärtustega mpstat käsust

Samas läheneb mõne protsessori tuuma iowait väärtus aeg-ajalt sajale protsendile, nagu on näha Joonisel 11 oleval graafikul. See tähendab, et süsteemil on täitmata ketta I/O päring.



Joonis 11. Graafik iowait veergude väärtustega mpstat käsust

Joonisel 12 oleval graafikul on näha, et samal ajal langeb protsessori mõne tuuma idle väärtus peaaegu nullini. See võib olla põhjuseks, miks andmete sisselugemisel tekivad pikad pausid.



Joonis 12. Graafik idle veergude väärtustega mpstat käsust

Selle testi raames tehti neli alameksperimenti, mille olulisemad parameetrid on välja toodud Tabelis 5.

Tabel 5. Kõikide alameksperimentide olulisemad parameetrid

	21 kB			204 kB			352 kB			735 kB		
	min	max	keskm.	min	max	keskm.	min	max	keskm.	min	max	keskm.
cpu all usr	0.3	4.3	1.6	0.5	3.8	1.8	0.3	13.0	1.8	0.3	9.7	1.8
cpu all iowait	0.0	67.0	3.3	0.0	26.8	3.0	0.0	52.8	3.2	0.0	29.8	1.8
cpu all idle	31.2	99.5	94.4	72.2	99.2	94.3	45.3	99.2	94.2	68.0	99.2	95.5
cpu 0 usr	0.0	7.1	1.7	0.0	5.9	1.8	0.0	16.7	1.8	0.0	8.3	2.1
cpu 0 iowait	0.0	99.0	4.8	0.0	0.0	0.0	0.0	94.9	3.3	0.0	100.0	3.7
cpu 0 idle	0.0	100.0	92.8	91.4	100.0	97.2	3.1	100.0	94.1	0.0	100.0	93.1
cpu 1 usr	0.0	5.1	1.6	0.0	5.9	1.9	0.0	24.7	1.9	0.0	7.8	1.7
cpu 1 iowait	0.0	97.9	3.7	0.0	99.0	5.4	0.0	97.0	3.8	0.0	36.0	0.7
cpu 1 idle	0.0	100.0	93.9	0.0	100.0	91.8	3.0	100.0	93.6	61.0	100.0	96.8
cpu 2 usr	0.0	8.1	1.6	0.0	7.8	1.6	0.0	18.2	2.2	0.0	13.1	1.6
cpu 2 iowait	0.0	88.8	2.4	0.0	100.0	3.5	0.0	71.7	1.8	0.0	97.0	1.8
cpu 2 idle	7.9	100.0	95.3	0.0	100.0	94.1	24.2	100.0	95.1	0.0	100.0	95.8
cpu 3 usr	0.0	10.1	1.7	0.0	8.0	1.9	0.0	8.8	1.4	0.0	24.7	1.8
cpu 3 iowait	0.0	89.8	2.3	0.0	95.0	3.2	0.0	99.0	3.8	0.0	33.3	1.0
cpu 3 idle	7.1	100.0	95.4	1.0	100.0	94.2	0.0	100.0	94.0	60.6	100.0	96.2
Latentsus	0.2	9.2	0.8	0.3	4.3	0.4	0.3	4.8	0.6	0.6	8.4	0.9
Intervall*	1.1	8.1	1.8	1.0	3.6	1.6	0.6	9.3	1.7	0.8	11.0	1.8

* Andmete sisselugemise intervall

Kõigis neljas alameksperimendis genereeriti andmefaili ridu iga 1.5 sekundi tagant. Tabelist 5 on näha, et tegelik andmete sisselugemise intervall on kuni 11 sekundit. Eksperimentides kasutatava pildifaili suuruse ja maksimaalse intervalli vahel ei ole otsest seost.

Eksperimentide viimistlemise käigus oli vaja teha korduvalt katseid samade parameetritega. Iga kord oli MiNiFi puhul eksperimendi tulemus märgatavalt erinev. Maksimaalne sisselugemise

intervall sõltus sellest, kui pikalt oli MiNiFi muude protsessidega hõivatud. Iga eksperiment võttis aega üle viie minuti, mille jooksul tekkis korduvalt 2-10 sekundilisi pause.

Tabelist 5 on näha, et protsessori usr väärtused ei tõuse eksperimentide käigus kõrgemale kui 25%. Samas ulatuvad protsessori tuumade iowait väärtused kuni 100 protsendini. Seetõttu langevad tuumade idle väärtused nullini, mis omakorda põhjustab MiNiFi töös pause.

Selliseid pikki pause andmete sisselugemise ajatemplite vahel, mille näide on toodud Tabelis 4, tekkis ka ülejäänud eksperimentides.

Andmete analüüsist selgub, et lisaks andmete sisselugemise intervalli suurenemisele põhjustasid pausid MiNiFi töös ka pikki latentsusaegu. Kui paus tekkis peale MiNiFi ajatempli lisamist, aga enne andmerea edasi saatmist, siis oli ka latentsusaeg pikk.

Selle testi tulemuste analüüsimisel tekkis kahtlus, kas pikkade viidete teke võis olla seotud MiNiFi filtreerimisega ning reanumbrite valesti omistamisega. Selle kontrollimiseks konstrueeriti selle testi raames veel üks alameksperiment. Eksperimendis genereeriti iga 1.5 sekundi tagant andmefaili uus andmerida. Kokku genereeriti 200 andmerida. Andmete genereerimisel määrati temperatuuride muutuse intervall selliselt, et MiNiFi saadaks edasi kõik read. Kasutati 21 kB suurust pildifaili. Andmete analüüsimisel oli näha, et kõik 200 andmerida saadeti edasi, kuid sellegipoolest tekkis mitu 5 kuni 9-sekundilist pausi andmete sisselugemisel. Ka Raspberry protsessori tuumade iowait väärtused lähenesid sajale protsendile.

Selleks, et saada ülevaade Raspberry ressursside kasutusest Pythoni koodiga, konstrueeriti Test 5 samade parameetritega nagu Test 4.

4.5 Test 5 Pythoni koodiga 1.5 sekundilise intervalliga andmete sisselugemine

Test 5 jaoks kasutati Pythoni koodi, mis luges 200-realisest andmefailist sisse temperatuuri ja CO2 taseme näite iga 1.5 sekundi tagant. Iga alameksperimendi jaoks kasutati erineva suurusega pildifaile. Pildifailide suurused oli 21 kB, 204 kB, 352 kB ja 735 kB. Sülearvuti ajanihe võrreldes kellaserveriga oli 0.4 sekundit.

Tabelis 6 kirjeldatud väärtused mõõdeti 21 kB pildifaili saatmisel. Testi tulemuste illustreerimiseks valiti 21 kB pildifailiga tehtud eksperiment, et võrrelda MiNiFi ja Pythoni tööd samade parameetrite puhul. Test 1 jooksul mõõdetud keskmine ühe pildifaili saatmise aeg oli 0.08 sekundit. Seega mahub pildifailide saatmise aeg koos süsteemi parameetrite päringu ajaga 1.5 sekundi sisse.

Tabel 6. Andmete sisselugemise ajad 21 kB pildifaili puhul

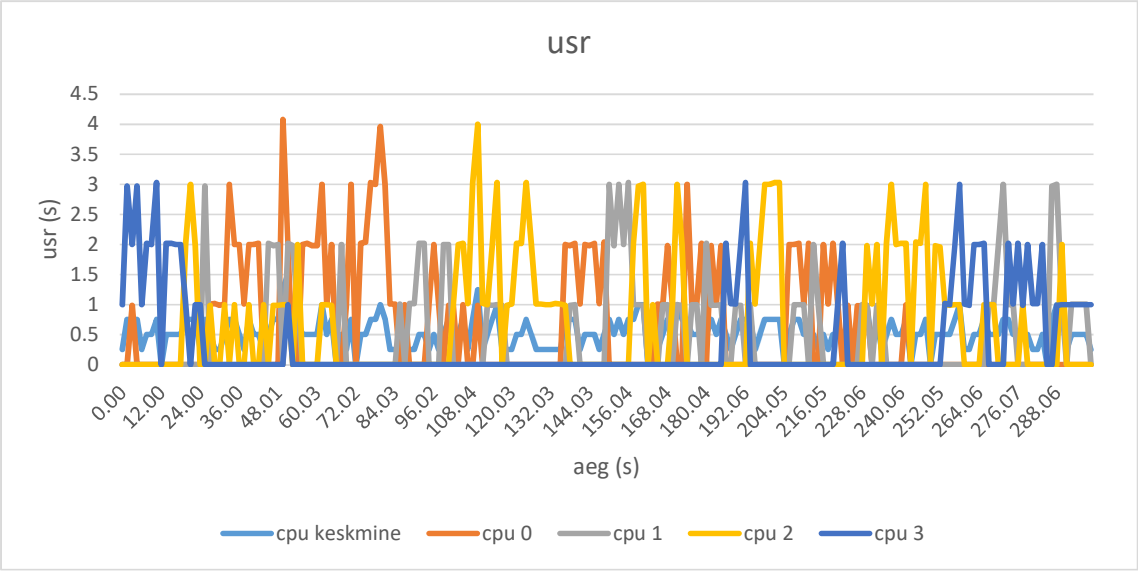
	Andmete sisselugemise intervall (s)
min	00:01.47
max	00:01.52
keskmine	00:01.50

Tabelist 6 on näha, et andmete sisselugemise intervall muutus vahemikus 1.47 kuni 1.52 sekundit. Seega on andmete sisselugemise intervall võrreldes MiNiFi'ga palju stabiilsem.

Pythoni koodis ei kasutatud andmete töötlemiseks ühtegi reeglit ega filtrit ning seega saadeti kõik andmeread edasi.

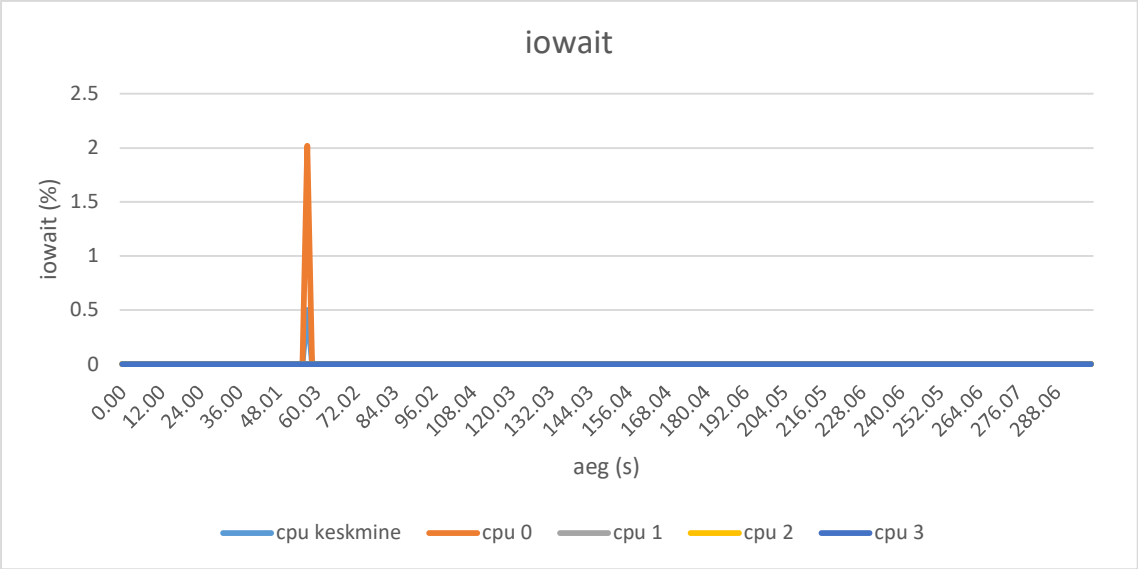
Järgneval kolmel joonisel on graafikud, mis näitavad alameksperimendi jooksul mõõdetud protsessori tuumade usr, iowait ja idle väärtusi `mpstat` käsust.

Joonisel 13 olevalt graafikult on näha, et protsessori tuumade usr väärtused ei tõuse üle 4%.



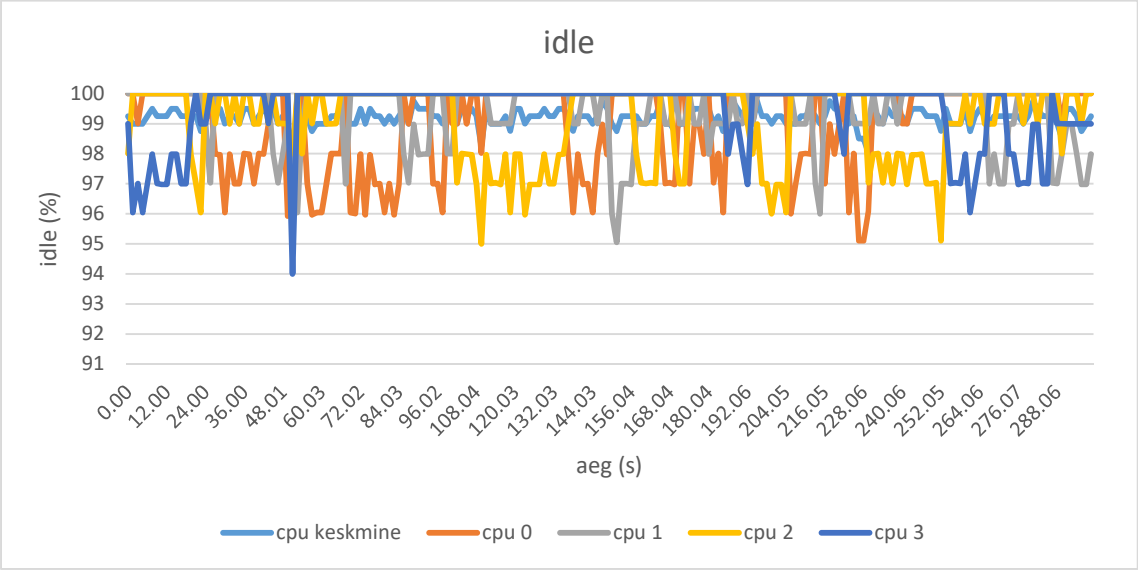
Joonis 13. Graafik usr veergude väärtustega mpstat käsust

Joonisel 14 on näha, et iowait väärtused on kogu eksperimendi vältel nullilähedased. Käesolev joonis on töösse lisatud selleks, et välja tuua oluline erinevus, kuidas MiNiFi ja Pythoni kasutavad Raspberry ressursse.



Joonis 14. Graafik iowait veergude väärtustega mpstat käsust

Joonisel 15 on näha, et kogu eksperimendi ajal on protsessori kõigi tuumade idle väärtused saja protsendi lähedased, mis näitab, et protsessorite tuumad ei olnud hõivatud.



Joonis 15. Graafik idle veergude väärtustega mpstat käsust

Võrreldes MiNiFi'ga ja Pythoni koodiga 21 kB pildifaili saatmist iga 1.5 sekundi tagant tuleb välja oluline erinevus. MiNiFi tekitab Raspberry protsessorile sellise koormuse, mille puhul tekivad kuni 8-sekundilised pausid raamistiku töös. Pythoni kood saadab andmeid ilma Raspberry protsessorile olulist koormust tekitamata ja seetõttu on kogu programmi töö stabiilne.

Selle testi raames tehti neli alameksperimenti, mille olulisemad parameetrid on välja toodud Tabelis 7.

Tabel 7. Kõikide alameksperimentide olulisemad parameetrid

	21 kB			204 kB			352 kB			735 kB		
	min	max	keskm.	min	max	keskm.	min	max	keskm.	min	max	keskm.
cpu all usr	0.0	1.5	0.5	0.0	1.0	0.5	0.0	1.0	0.4	0.0	1.5	0.5
cpu all iowait	0.0	0.5	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.3	0.0
cpu all idle	96.5	99.8	99.2	98.3	99.8	99.3	98.8	100.0	99.4	96.3	99.8	99.3
cpu 0 usr	0.0	4.1	0.6	0.0	3.0	0.6	0.0	2.0	0.2	0.0	3.0	0.4
cpu 0 iowait	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu 0 idle	95.1	100.0	99.0	96.0	100.0	99.1	97.0	100.0	99.6	96.0	100.0	99.3
cpu 1 usr	0.0	3.0	0.5	0.0	3.0	0.5	0.0	3.0	0.3	0.0	3.0	0.5
cpu 1 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
cpu 1 idle	95.1	100.0	99.3	95.1	100.0	99.2	96.0	100.0	99.4	95.1	100.0	99.2
cpu 2 usr	0.0	4.0	0.6	0.0	3.0	0.3	0.0	3.0	0.4	0.0	3.0	0.4
cpu 2 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu 2 idle	95.0	100.0	99.0	95.1	100.0	99.6	95.2	100.0	99.2	95.1	100.0	99.4
cpu 3 usr	0.0	3.0	0.4	0.0	3.0	0.4	0.0	3.0	0.4	0.0	3.0	0.5
cpu 3 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu 3 idle	94.0	100.0	99.4	93.9	100.0	99.3	97.0	100.0	99.5	95.9	100.0	99.2
Latentsus	0.1	0.1	0.1	0.5	1.0	0.5	0.8	1.5	0.9	1.7	2.2	1.8
Intervall*	1.5	1.5	1.5	1.5	2.0	1.6	1.9	2.5	1.9	2.8	3.3	2.9

* Andmete sisselugemise ja saatmise intervall

Kõigis neljas alameksperimendis loeti andmefailist ridu sisse iga 1.5 sekundi tagant. Tabelist 7 on näha, et tegelikult muutus andmete sisselugemise ja saatmise intervall vastavalt pildi suurusele. Test 1 tulemustest oli näha, kui palju võtab iga pildifaili saatmine keskmiselt aega. 21 kB suuruse pildifaili sisselugemine ja saatmine võttis aega 0.08 sekundit. Iga andmerea

korral tehti päring protsessori parameetrite saamiseks, mis võttis aega 1.1 sekundit. 21 kB pildifaili puhul mahtus nii päringu tegemine kui pildifaili saatmine 1.5 sekundi sisse.

Suuremate pildifailide puhul ei mahtunud parameetrite päringu aeg ja pildifaili saatmise aeg 1.5 sekundi sisse. Eksperimendid tehti ka suuremate pildifailidega, et teha kindlaks, kuidas programm saab hakkama suuremate pildifailide edastamisega, kui andmete saatmise aeg ei mahu 1.5 sekundi sisse. Tabelist on näha, et andmete saatmise intervall oli ka suurte pildifailide puhul stabiilne. Suurim erinevus minimaalse ja maksimaalse intervalli vahel oli 0.6 sekundit.

Tabelist on näha, et mõõtmiste ajal ei tõuse protsessori usr väärtused kõrgemale kui 4%. See on tingitud asjaolust, et `mpstat` käsuga tehtud parameetrite päring toimus kahe pildifaili saatmise vahel. Testide disainimise käigus katsetati ka sellist varianti, et käivitati pildifaili saatmine ning alustati kohe järgmise andmerea sisselugemist ja protsessori parameetrite päringut.

```
10:44:36 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
10:44:37 AM  all   26.52    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   73.48
10:44:37 AM    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
10:44:37 AM    1    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   99.00
10:44:37 AM    2    4.12    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   95.88
10:44:37 AM    3  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00

10:44:37 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
10:44:38 AM  all   14.75    0.00    4.50    0.00    0.00    0.00    0.00    0.00    0.00   80.75
10:44:38 AM    0   36.00    0.00    3.00    0.00    0.00    0.00    0.00    0.00    0.00   61.00
10:44:38 AM    1    4.04    0.00    4.04    0.00    0.00    0.00    0.00    0.00    0.00   91.92
10:44:38 AM    2    3.92    0.00    2.94    0.00    0.00    0.00    0.00    0.00    0.00   93.14
10:44:38 AM    3   15.15    0.00    8.08    0.00    0.00    0.00    0.00    0.00    0.00   76.77

10:44:38 AM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
10:44:39 AM  all   25.19    0.00    1.48    0.00    0.00    0.00    0.00    0.00    0.00   73.33
10:44:39 AM    0   87.00    0.00    3.00    0.00    0.00    0.00    0.00    0.00    0.00   10.00
10:44:39 AM    1    1.98    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00   97.03
10:44:39 AM    2    4.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00   95.00
10:44:39 AM    3    8.65    0.00    0.96    0.00    0.00    0.00    0.00    0.00    0.00   90.38
```

Joonis 16. Kõrged usr väärtused mpstat käsu väljundis

Nagu on näha Joonisel 16, siis oli `mpstat` käsuga näha protsessori tuumade usr kõrgeid väärtusi, aga osa sisse loetud andmeridadest võis jääda saatmata, sest Pythoni kood lõpetas töö kui kõik andmeread olid sisse loetud. Kui pildifaili saatmise aeg oli pikem kui intervalliks määratud 1.5 sekundit, siis jäid viimased andmeread saatmata. Seetõttu muudeti Pythoni koodi ning uue andmerea sisselugemist ei alustatud enne, kui pildifaili saatmine oli lõpetatud.

4.6 Test 6 MiNiFi 3 sekundilise intervalliga andmete sisselugemine

Test 6 jaoks kasutati MiNiFi'd. Kõigis alameksperimentides salvestati andmefaili iga 3 sekundi järel uus andmerida. Kokku lisati faili 100 andmerida, kust oli temperatuuri ja CO_2 taseme näidud. Iga alameksperimendi jaoks kasutati erineva suurusega pildifaile. Pildifailide suurused oli 21 kB, 204 kB, 352 kB ja 735 kB. Sülearvuti ajanihe võrreldes kellaserveriga oli -0.97 sekundit.

Tabelis 8 kirjeldatud väärtused mõõdeti 735 kB pildifaili saatmisel. Testi tulemuste illustreerimiseks valiti 735 kB pildifailiga tehtud eksperiment, sest Test 1 jooksul mõõdetud keskmine ühe pildifaili saatmise aeg oli 0.84 sekundit. Seega oleks pidanud pildifailide saatmise aeg koos süsteemi parameetrite päringu ajaga mahtuma 3 sekundi sisse.

Tabel 8. Andmete sisselugemise ajad 735 kB pildifaili puhul

	Andmete sisselugemise intervall (s)
min	00:01.17
max	00:09.66
keskmine	00:03.31

Tabelist 8 on näha, et andmete sisselugemise intervall muutus vahemikus 1.17 kuni 9.66 sekundit. Seega kulus mõne andmerea sisselugemiseks mitu korda rohkem aega kui eksperimendis määratud 3 sekundit. Andmete sisselugemise intervalli arvutamisel kasutati ka iga andmereaga kaasas olevat reanumbrit. Reanumbrid andsid ülevaate, millised andmeread edasi saadeti ning millised välja filtreeriti MiNiFi's rakendatud reeglite abil.

Uurides selle eksperimendi andmeid lähemalt on näha, et 9.66 sekundi jooksul ei lugenud MiNiFi sisse ühtegi andmerida. Need andmeread on välja toodud ka Tabelis 9, milles on ka näha, et sisse loetud ridade numbrid on järjestikused. Seega ei saanud olla antud juhul pika ajalise viite põhjuseks ridade välja filtreerimine.

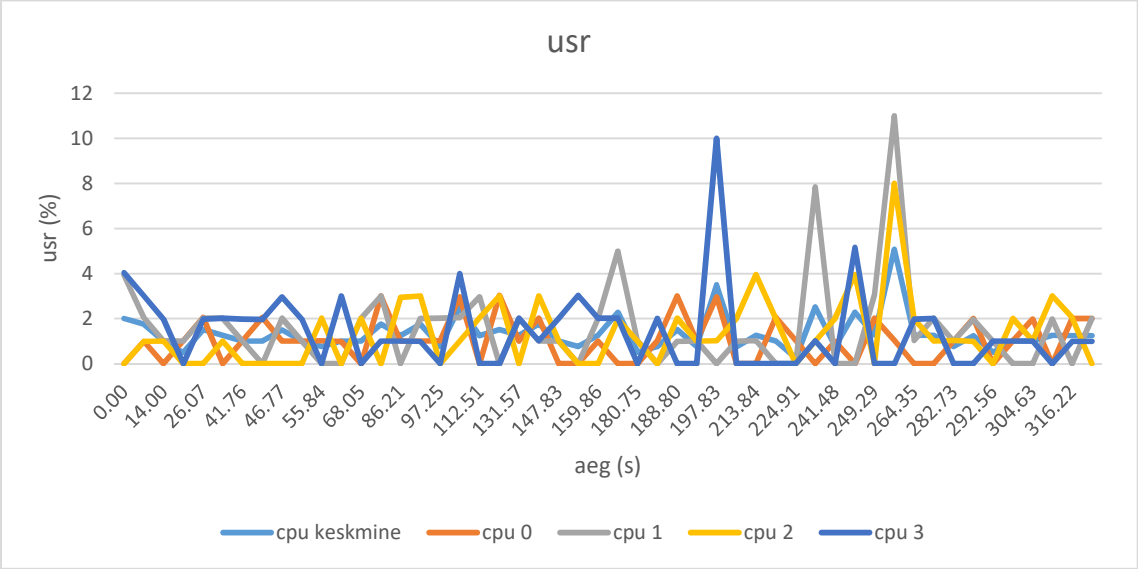
Tabel 9. Näide andmete sisselugemisel tekkinud pausist

MiNiFi ajatempel	Rida	Sparki ajatempel ajanihkega	Latentsusaeg	Sisselugemise intervall (s)
23:52:19.902881	11	23:52:20.517	00:00.614	00:03.01
23:52:29.565796	12	23:52:30.199	00:00.633	00:09.66

Selles eksperimendis kirjutati andmefaili uusi ridu iga 3 sekundi tagant. Sarnaselt Test 4 toodud näitele, on ka selle eksperimendi puhul tõenäoliselt tekkinud olukord, kus TailFile luges korraga sisse mitu andmerida. Sellisel juhul saadeti sisse loetud andmeread edasi nii kiiresti kui suudeti. Seega võis tekkida mõnikord andmete töötlemise ja edasi saatmise intervalliks 1.17 sekundit.

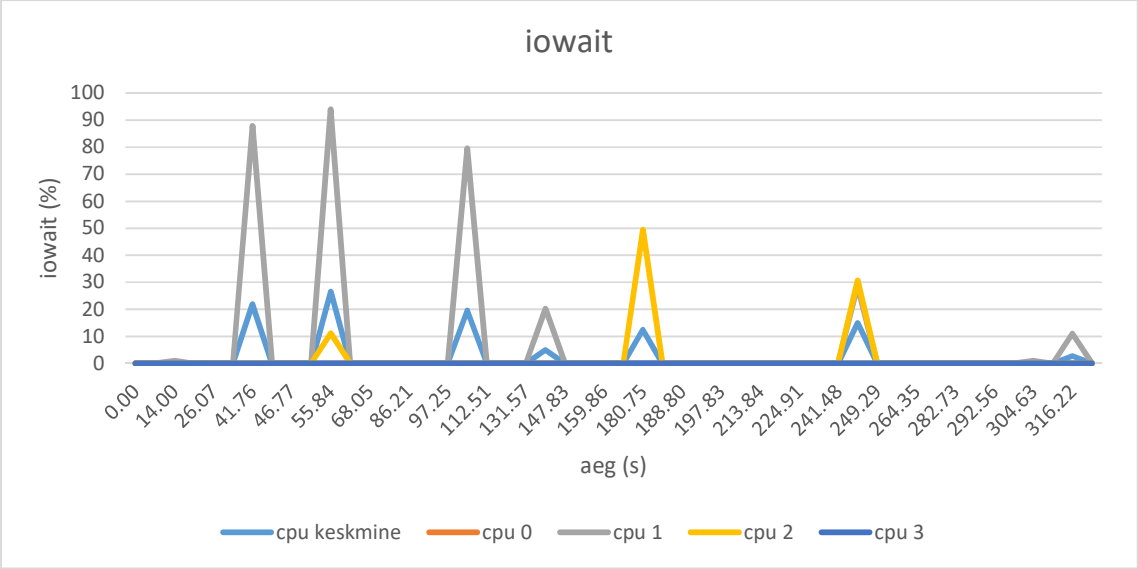
Järgneval kolmel joonisel on graafikud, mis näitavad selle alameksperimendi jooksul mõõdetud protsessori tuumade usr, iowait ja idle väärtusi `mpstat` käsust.

Joonisel 17 olevalt graafikult on näha, et protsessori tuumade usr väärtused ei tõuse üle 12%.



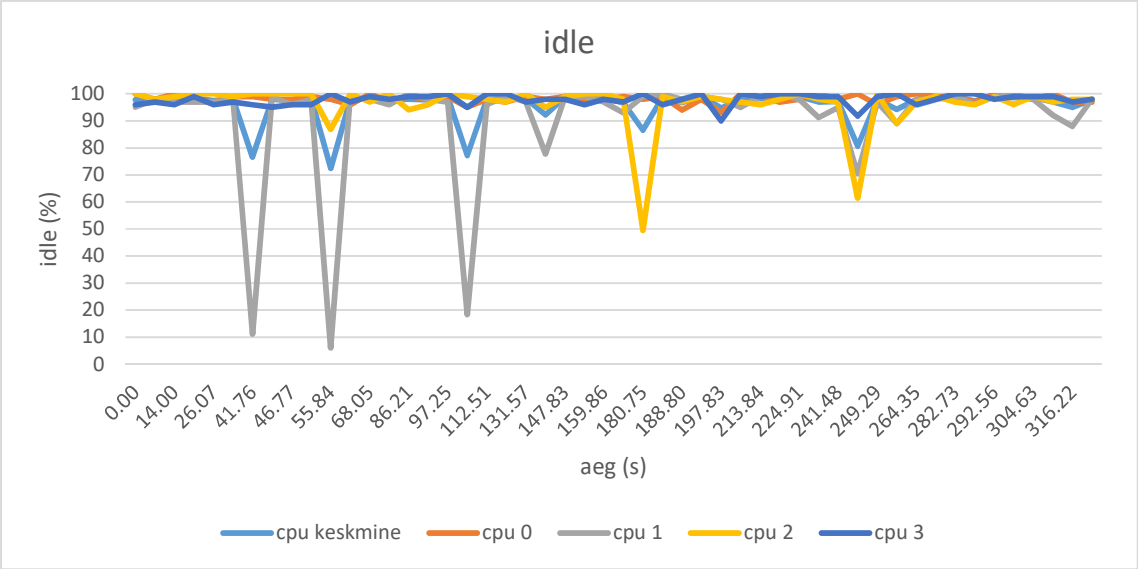
Joonis 17. Graafik usr veergude väärtustega mpstat käsust

Samas on mõne protsessori tuuma iowait üle 90 protsendi, nagu on näha Joonisel 18 oleval graafikul.



Joonis 18. Graafik iowait veergude väärtustega mpstat käsust

Joonisel 19 oleval graafikul on näha, et samal ajal langeb protsessori mõne tuuma idle väärtus alla 10 protsendi. See võib olla põhjuseks, miks andmete sisselugemisel tekivad pikad pausid.



Joonis 19. Graafik idle veergude väärtustega mpstat käsust

Selle testi raames tehti neli alameksperimenti, mille olulisemad parameetrid on välja toodud Tabelis 10.

Tabel 10. Kõikide alameksperimentide olulisemad parameetrid

	21 kB			204 kB			352 kB			735 kB		
	min	max	keskm.	min	max	keskm.	min	max	keskm.	min	max	keskm.
cpu all usr	0.3	4.2	1.4	0.5	3.5	1.2	0.0	4.3	1.5	0.3	5.1	1.3
cpu all iowait	0.0	21.7	1.2	0.0	42.5	2.3	0.0	24.6	1.4	0.0	26.6	2.1
cpu all idle	75.3	99.2	96.6	55.3	99.2	95.8	73.7	98.7	96.4	72.4	99.0	95.8
cpu 0 usr	0.0	8.7	1.5	0.0	5.8	1.3	0.0	5.0	1.2	0.0	3.0	1.0
cpu 0 iowait	0.0	1.0	0.0	0.0	13.1	0.5	0.0	0.0	0.0	0.0	0.0	0.0
cpu 0 idle	88.4	100.0	97.6	85.7	100.0	97.3	94.0	100.0	98.2	93.1	100.0	98.2
cpu 1 usr	0.0	6.9	1.5	0.0	5.1	0.8	0.0	9.2	1.9	0.0	11.0	1.6
cpu 1 iowait	0.0	11.2	0.4	0.0	80.0	3.3	0.0	98.0	2.3	0.0	94.0	6.5
cpu 1 idle	88.8	100.0	97.0	19.0	100.0	95.3	2.0	100.0	95.0	6.0	100.0	91.0
cpu 2 usr	0.0	5.8	1.2	0.0	4.0	1.4	0.0	5.0	1.4	0.0	8.0	1.3
cpu 2 iowait	0.0	12.6	0.3	0.0	76.2	5.0	0.0	83.7	2.3	0.0	49.5	1.8
cpu 2 idle	79.6	100.0	97.8	17.8	100.0	92.7	14.3	100.0	95.4	49.5	100.0	96.2

cpu 3 usr	0.0	7.1	1.4	0.0	4.1	1.3	0.0	5.0	1.3	0.0	10.0	1.4
cpu 3 iowait	0.0	86.0	4.0	0.0	10.0	0.2	0.0	29.0	1.0	0.0	0.0	0.0
cpu 3 idle	11.0	100.0	93.9	87.0	100.0	98.0	70.0	100.0	97.0	90.0	100.0	97.8
Latentsus	0.2	2.2	0.4	0.3	2.2	0.4	0.3	3.3	0.5	0.6	3.5	0.8
Intervall*	1.9	5.4	3.1	1.5	4.8	3.1	1.5	5.4	3.2	1.2	9.7	3.3

* Andmete sisselugemise intervall

Kõigis neljas alameksperimendis genereeriti andmefaili ridu iga 3 sekundi tagant. Tabelist 10 on näha, et tegelik andmete sisselugemise intervall on kuni 9.7 sekundit.

Tabelist on näha, et protsessori usr väärtused ei tõuse eksperimentide käigus kõrgemale kui 11%. Samas tõusevad protsessori tuumade iowait väärtused üle 90 protsendi. Seetõttu langevad tuumade idle väärtused alla 10 protsendi, mis võib põhjustada MiNiFi töös pause.

Ülevaate saamiseks Raspberry ressursside kasutusest Pythoni koodiga, konstrueeriti Test 7 samade parameetritega.

4.7 Test 7 Pythoni koodiga 3 sekundilise intervalliga andmete sisselugemine

Test 7 jaoks kasutati Pythoni koodi, mis luges 100-realisest andmefailist sisse temperatuuri ja CO2 taseme näite iga 3 sekundi tagant. Iga alameksperimendi jaoks kasutati erineva suurusega pildifaile. Pildifailide suurused oli 21 kB, 204 kB, 352 kB ja 735 kB. Sülearvuti ajanihe võrreldes kellaserveriga oli 1.3 sekundit.

Tabelis 11 kirjeldatud väärtused mõõdeti 735 kB pildifaili saatmisel. Testi tulemuste illustreerimiseks valiti 735 kB pildifailiga tehtud eksperiment, et võrrelda MiNiFi ja Pythoni tööd samade parameetrite puhul. Test 1 jooksul mõõdetud keskmine ühe pildifaili saatmise aeg oli 1.74 sekundit. Seega mahub pildifailide saatmise aeg koos süsteemi parameetrite päringu ajaga 3 sekundi sisse.

Tabel 11. Andmete sisselugemise ajad 735 kB pildifaili puhul

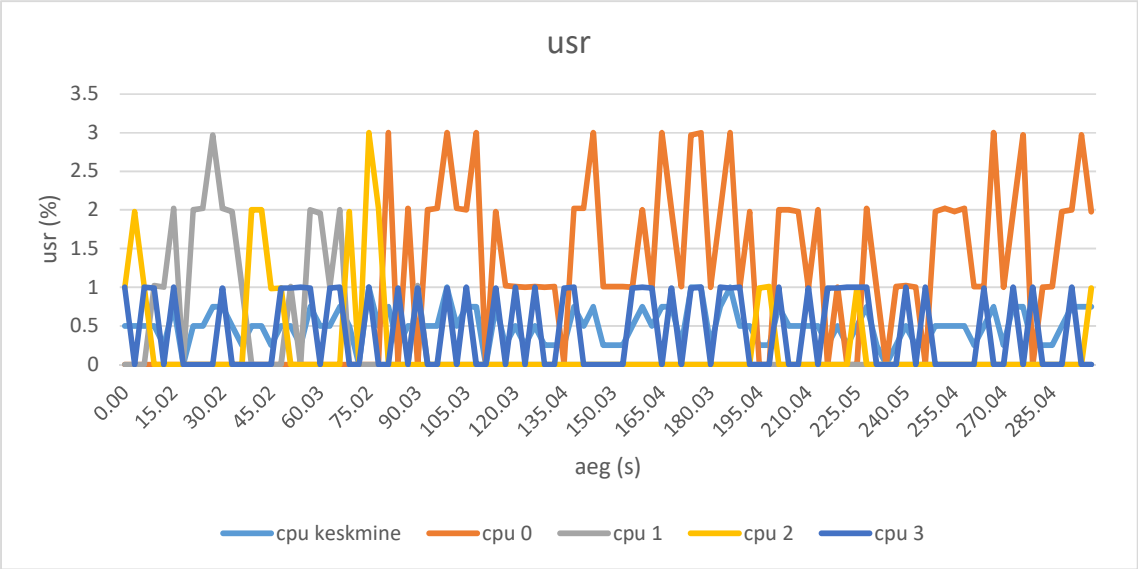
	Andmete sisselugemise intervall (s)
min	00:02.98
max	00:03.02
keskmine	00:03.00

Tabelist 11 on näha, et andmete sisselugemise intervall muutus vahemikus 2.98 kuni 3.02 sekundit. Andmete sisselugemise intervall võrreldes MiNiFi'ga on palju stabiilsem.

Pythoni koodis ei kasutatud andmete töötlemiseks ühtegi reeglit ega filtrit. Seega saadeti kõik andmeread edasi.

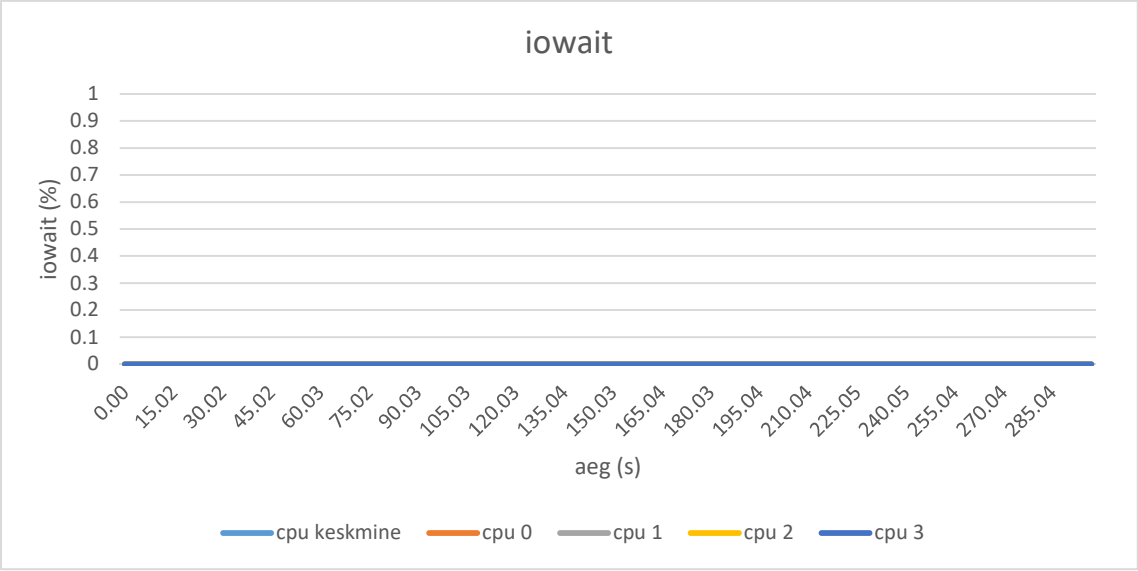
Järgneval kolmel joonisel on graafikud, mis näitavad alameksperimendi jooksul mõõdetud protsessori tuumade usr, iowait ja idle väärtusi `mpstat` käsust.

Joonisel 20 oleval graafikul ei tõuse usr väärtused üle 3%.



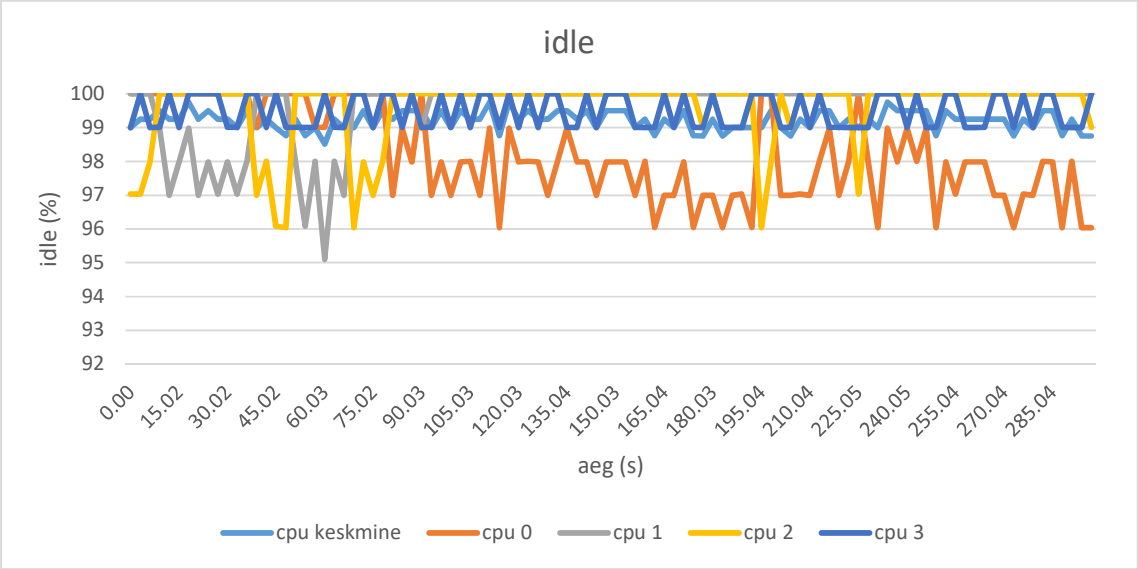
Joonis 20. Graafik usr veergude väärtustega mpstat käsust

Eksperimendi käigus ei tõusnud iowait väärtused nullist kõrgemale nagu on näha ka Joonisel 21.



Joonis 21. Graafik iowait veergude väärtustega mpstat käsust

Sarnaselt Test 5-le on idle väärtused kõrged ning ei lange alla 95%. Seda on näha Joonisel 22.



Joonis 22. Graafik idle veergude väärtustega mpstat käsust

Kui võrrelda MiNiFi ja Pythoni poolt tekitatud koormust Raspberry protsessorile 735 kB pildifaili sisselugemisel ja saatmisel iga 3 sekundi tagant, siis selgub, et Pythoni kood ei tekita Raspberry protsessorile nii suurt koormust nagu MiNiFi.

Selle testi raames tehtud nelja alameksperimendi olulisemad parameetrid on välja toodud Tabelis 12.

Tabel 12. Kõikide alameksperimentide olulisemad parameetrid

	21 kB			204 kB			352 kB			735 kB		
	min	max	keskm.	min	max	keskm.	min	max	keskm.	min	max	keskm.
cpu all usr	0.0	1.0	0.5	0.0	1.0	0.5	0.0	2.0	0.6	0.0	1.0	0.5
cpu all iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	22.8	0.2	0.0	0.0	0.0
cpu all idle	98.5	99.5	99.1	98.5	99.8	99.1	76.2	99.5	98.8	98.5	99.8	99.2
cpu 0 usr	0.0	3.0	0.3	0.0	3.0	0.2	0.0	4.0	0.2	0.0	3.0	1.1
cpu 0 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	91.9	0.9	0.0	0.0	0.0
cpu 0 idle	96.0	100.0	99.5	96.0	100.0	99.6	8.1	100.0	98.7	96.0	100.0	98.2
cpu 1 usr	0.0	3.1	0.7	0.0	3.0	0.9	0.0	5.0	0.8	0.0	3.0	0.3
cpu 1 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

cpu 1 idle	96.0	100.0	98.9	95.1	100.0	98.6	95.0	100.0	98.8	95.1	100.0	99.6
cpu 2 usr	0.0	3.0	0.8	0.0	3.0	0.4	0.0	4.0	0.7	0.0	3.0	0.2
cpu 2 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu 2 idle	95.1	100.0	98.7	95.1	100.0	99.3	95.1	100.0	98.9	96.0	100.0	99.6
cpu 3 usr	0.0	3.0	0.4	0.0	4.0	0.7	0.0	3.0	0.7	0.0	1.0	0.4
cpu 3 iowait	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
cpu 3 idle	95.1	100.0	99.4	95.1	100.0	99.0	96.0	100.0	98.9	99.0	100.0	99.5
Latentsus	0.1	0.1	0.1	0.5	0.8	0.5	0.8	1.0	0.8	1.6	1.9	1.7
Intervall*	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0

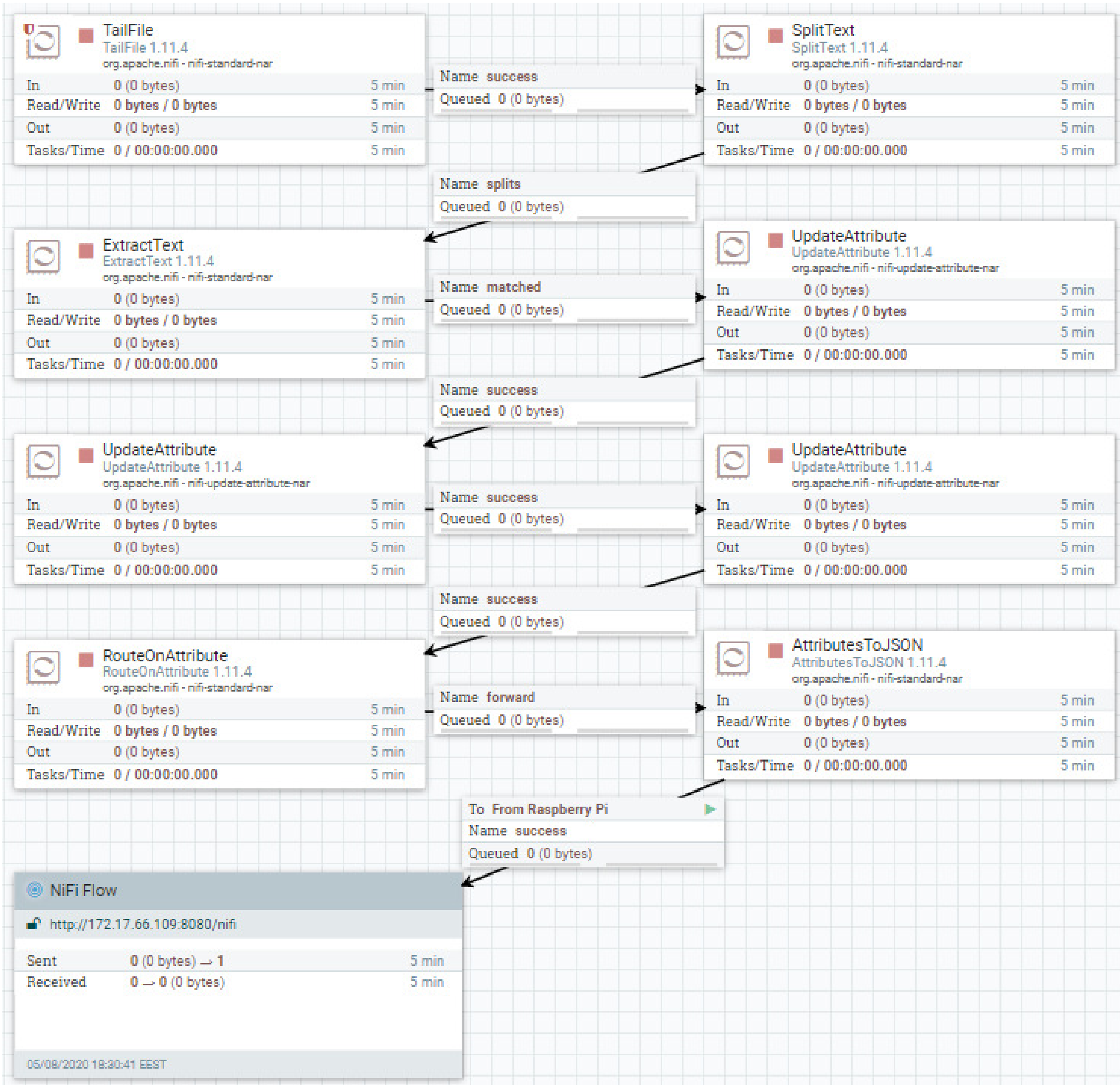
* Andmete sisselugemise ja saatmise intervall

Neljas alameksperimendis loeti andmefailist ridu sisse iga 3 sekundi tagant. Tabelist 12 on näha, et andmete sisse lugemise intervall oli 3.0 sekundit ja ei sõltu pildifaili suurusest, sest kõikide pildifailide saatmise aeg mahub 3 sekundi sisse.

4.8 Test 8 MiNiFi lisatest ilma Linuxi käskudeta

Eelnevate testide analüüsist selgus, et MiNiFi puhul tekivad pikad pausid andmete sisselugemise ja saatmise vahel. Tekkis kahtlus, et põhjuseks võivad olla ExecuteStreamCommand protsessoris käivitatavas Pythoni koodis olevad Linuxi käsud, millega tehti päringuid protsessori parameetrite info saamiseks.

Selle kontrollimiseks konstrueeriti lisatest, mille käigus eemaldati ExecuteStreamCommand protsessor MiNiFi protsessorite ahelast. Uus protsessorite ahel on näha Joonisel 23. Protsessorite parameetrites olevates väärtused tuleb enne Kafkasse saatmist viia JSON formaati. Eelnevalt viidi andmed JSON formaati ExecuteStreamCommand'iga käivitatud Pythoni skriptis. Selles testis on protsessorite ahelasse lisatud AttributesToJSON protsessor.



Joonis 23. Protsessorite ahel ilma Linuxi käske kasutava protsessorita

Testis salvestati andmefaili iga 1.5 sekundi tagant uus andmerida, mis koosnes neljast parameetrist: temperatuuri näit, CO_2 taseme näit, reanumber ja ajatempel. Kokku genereeriti 200 andmerida. Andmete genereerimisel määrati temperatuuride muutuse intervall selliselt, et MiNiFi saadaks edasi kõik read. Andmetega ei saadetud pildifaili, et testida MiNiFi tööd võimalikult väikse andmemahuga.

Kõik 200 andmerida jõudsid Sparkini. Andmete analüüsist selgus, et maksimaalne andmete sisselugemise intervall oli 1.5 sekundi asemel 6.16 sekundit. Need andmeread on välja toodud ka Tabelis 13, kus on näha, et sisse loetud ridade numbrid on järjestikused.

Tabel 13. Näide andmete sisselugemisel tekkinud pausist

MiNiFi ajatempel	Rida	Sparki ajatempel ajanihkega	Latentsusaeg	Sisselugemise intervall (s)
19:11:15.731	191	19:11:21.756	6.024	1.502
19:11:21.891	192	19:11:22.874	0.982	6.160

Tabelis 13 on näha ka pikk latentsusaeg, mis on samuti põhjustatud pausist MiNiFi töös. Antud testi andmete analüüsist selgus, et üle 3 sekundi pikkuseid viiteid tekkis 200 andmerea edastamise jooksul 6 korda.

Selle testi põhjal võib järeldada, et pausid MiNiFi töös ei ole tingitud pildifaili saatmisest ega ka MiNiFi ahela keskel käivitatud Linuxi käskudest.

4.9 Tulemuste kokkuvõte

Test 4 ja Test 5 puhul loeti andmeid sisse 1.5-sekundilise intervalliga. Pythoni koodiga andmete saatmisel oli sisselugemise intervall stabiilne. MiNiFi Java versiooni puhul tekkisid andmete sisselugemisel ja saatmisel kuni 10-sekundilised pausid. Tõenäoliselt on viidete põhjuseks Raspberry protsessori tuumade kõrged iowait väärtused. Selgitati ka välja, et andmete sisselugemise pikad intervallid ei ole seotud MiNiFi andmete filtreerimisega. Samuti ei sõltu viited saadetava pildifaili suurusest. Ka ilma pildifailita andmete saatmisel tekkisid seisakud MiNiFi töös. See näitab, et seisakud ei ole seotud edastatavate andmete mahuga. Tekkinud pauside kordumise intervall ning kestus olid juhuslikud.

Test 6 ja Test 7 puhul oli andmete sisselugemise intervall 3 sekundit. Hoolimata andmete sisselugemise intervalli suurendamisest, tekkisid MiNiFi puhul andmete sisselugemisel sarnased pikad pausid. See näitab, et seisakute põhjuseks ei olnud ka liiga lühike andmete sisselugemise intervall.

MiNiFi Java versiooni ja Pythoni koodiga andmete saatmise võrdlemisel töötas Pythoni kood stabiilsemalt. Kui hallatavate servaseadmete arv pole suur ning andmete edastamise stabiilsus on tähtis, siis võiks eelistada Pythoni koodi kasutamist.

5. Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli uurida Apache MiNiFi raamistiku kasutamise mõistlikkust servaseadmes läbi praktilise katsetuse, valides riistvaraliseks lahenduseks Raspberry Pi. Testide eesmärgiks oli hinnata Raspberry Pi ressursside kulu ning töödeldavate andmete mahtu ilma raamistikuta ning MiNiFi'd kasutades.

Testide teostamiseks seati üles keskkonnad, mille abil oleks võimalik saata andmeid Raspberry Pi'st Apache Sparki. Raspberry'l töötav MiNiFi saatis NiFi kaudu andmeid Kafkasse, kust andmed liikusid edasi Sparki. Töö raames konstrueeriti erinevaid teste, mille käigus koguti andmeid Raspberry Pi protsessori parameetrite kohta. Vaadeldavateks parameetriteks olid näiteks protsessori tuumade usr, iowait ja idle väärtused. Testide tulemuste analüüsis võrreldi Raspberry Pi tööd kogutud parameetrite alusel.

Praktilise võrdluse käigus selgus, et töös kirjeldatud tingimustel ei andnud MiNiFi Java raamistiku kasutamine paremaid tulemusi kui Raspberry'st Pythoni koodiga andmete saatmine. Testide põhjal selgub, et MiNiFi Java raamistiku töös tekivad pausid ka väikeste andmemahtude korral. Probleemi põhjuseks võib olla SD-kaardile kirjutamise ja lugemise kiirus. Sellele viitavad Raspberry protsessori tuumade kõrged iowait väärtused.

MiNiFi ja Pythoni testide käigus ei tulnud välja olulisi erinevusi Raspberry mälu kasutamisel. Kõikide testide jooksul oli mälukasutuse tase 10 kuni 12%. Samuti ei muutunud testide käigus oluliselt protsessori temperatuur, jäädes 39 ja 44 kraadi vahele.

Kuna katsetes tekkisid MiNiFi töös pausid, siis on soovituslik ajakriitilistes rakendustes Raspberry Pi'l MiNiFi Java versiooni mitte kasutada. Kui aga on soov seda siiski teha, siis on otstarbekas uurida rohkem MiNiFi repositooriumide ja salvestusseadmele kirjutamise kohta.

Kui kasutada Raspberry Pi's MiNiFi Java versiooni, siis võiks kasutada võimalikult kiire klassi SD-kaarti või välist SSD-d. Samuti oleks soovituslik seadistada MiNiFi's logimise tase võimalikult minimaalseks.

Tehtud teste saaks täiendada veel sarnastes tingimustes MiNiFi C++ versiooniga katseid tehes. Lisaks võiks uurida rohkem MiNiFi seadistuste kohta, et välja selgitada, millistel tingimustel on võimalik vähendada SD-kaardi kasutamist. Samuti võiks teha võrdlusi teiste raamistikega nagu Apache Edgent.

6. Viidatud kirjandus

[1] Rouse, Margaret. internet of things (IoT). Updated in: 2020, IOT Agenda.

https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (2020.03.04)

[2] Ken Hosac. Three Reasons Why You need Edge Computing Now. Published in: 2017 in Dataversity. https://www.dataversity.net/three-reasons-need-edge-computing-now/ (2020.03.07)

[3] Bharat S. Chaudhari, Marco Zennaro, Suresh Borkar. LPWAN Technologies: Emerging Application Characteristics, Requirements, and Design Considerations. Published in: 2020. https://doi.org/10.3390/fi12030046

[4] Mahmut Taha Yazici, Shadi Basurra, Mohamed Medhat Gaber. Edge Machine Learning: Enabling Smart Internet of Things Applications. Published in: 2018. https://doi.org/10.3390/bdcc2030026

[5] Apache MiNiFi. https://nifi.apache.org/minifi/index.html (2020.03.06)

[6] Apache Kafka. https://kafka.apache.org/intro (2020.03.04)

[7] Denny Lee, Jules Damji. Apache Spark Key Terms, Explained. Published in: 2016. https://databricks.com/blog/2016/06/22/apache-spark-key-terms-explained.html (2020.03.31)

[8] Jürisoo, Lauri 2016. Ülevaade: asjade internet ja mis sellest kasu on. https://forte.delfi.ee/news/tarkvara/ulevaade-asjade-internet-ja-mis-sellest-kasu-on?id=73717233 (29.03.2020)

[9] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, A. Selcuk Uluagac. A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. Published in 2018. https://www.researchgate.net/figure/IoT-Architecture-Layers-and-Components_fig1_322975901 (2020.03.31)

[10] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu. Edge computing: Vision and challenges. Published in: 2016, IEEE internet of things journal, 3(5), 637-646. https://doi.org/10.1109/JIOT.2016.2579198

[11] Radiocrafts. Cloud vs Fog vs Mist Computing, Which One Should You Use? Published in: 2019. https://radiocrafts.com/cloud-vs-fog-vs-mist-computing-which-one-should-you-use/ (2020.03.31)

[12] Shanhe Yi, Zijiang Hao, Zhengrui Qin, Qun Li. Fog Computing: Platform and Applications. Published in: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies. https://doi.org/10.1109/HotWeb.2015.22

[13] Jürgo S. Preden, Kalle Tammemäe, Axel Jantsch, Mairo Leier, Andri Riid, Emine Calls. The Benefits of Self-Awareness and Attention in Fog and Mist Computing. Published in: Computer, Volume: 48 , Issue: 7 , July 2015, pp 37-45. https://doi.org/10.1109/MC.2015.207

[14] Michaela Iorga, Larry Feldman, Robert Barton, Michael J. Martin, Nedim Goren, Charif Mahmoudi. Fog Computing Conceptual Model. Published in: 2018. https://doi.org/10.6028/NIST.SP.500-325

[15] Apache NiFi: Developer Guide. http://nifi.apache.org/developer-guide.html (2020.03.29)

[16] Apache NiFi. https://www.cloudera.com/products/open-source/apache-hadoop/apache-nifi.html (2020.03.05)

[17] Apache NiFi Documentation. https://nifi.apache.org/docs.html (2020.03.05)

[18] Apache NiFi. https://nifi.apache.org/docs/nifi-docs/html/user-guide.html (2020.05.07)

[19] Apache NiFi: Administration Guide. https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html (2020.03.28)

[20] Apache MiNiFi. https://cwiki.apache.org/confluence/display/MINIFI (2020.03.05)

[21] Tim Spann. Intelligently Collecting Data at the Edge – Intro to Apache MiNiFi. Published in: 2019. https://www.slideshare.net/Hadoop_Summit/intelligently-collecting-data-at-the-edge-intro-to-apache-minifi-141236290 (2020.03.18)

[22] Apache Edgent. https://edgent.incubator.apache.org/ (2020.03.06)

[23] Apache Edgent Documentation. http://edgent.incubator.apache.org/docs/overview (2020.03.30)

[24] Apache Edgent: Getting Started. http://edgent.incubator.apache.org/docs/edgent-getting-started.html (2020.03.28)

[25] Kuidas kasutada Apache Kafka sõnumit .Net-is. https://est.small-business-tracker.com/how-use-apache-kafka-messaging-net-410220 (19.03.2020)

[26] Apache Kafka – Your Event Stream Processing Solution. Published in: HTC global services. https://www.htcinc.com/wp-content/uploads/2017/05/apache_kafka_event_stream_processing_solution.pdf (2020.03.29)

[27] Apache Kafka – Cluster Architecture.
https://www.tutorialspoint.com/apache_kafka/apache_kafka_cluster_architecture.htm
(2020.05.06)

[28] Apache Spark Cluster Mode Overview. https://spark.apache.org/docs/latest/cluster-overview.html (2020.04.08)

[29] Ian Pointer. What is Apache Spark? The big data platform that crushed Hadoop. Published in: 2020. https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html (2020.04.03)

[30] Abdulsalam Yassinea, Shailendra Singhb, M. Shamim Hossain, Ghulam Muhammadd. IoT big data analytics for smart homes with fog and cloud computing. Published in: 2019 Future Generation Computer Systems, Volume 91, February 2019, pp 563-573. https://doi.org/10.1016/j.future.2018.08.040

[31] Simar Preet Singh, Anand Nayyar, Rajesh Kumar, Anju Sharma. Fog computing: from architecture to edge computing and big data processing. Published in: The Journal of Supercomputing, April 2019, Volume 75, Issue 4, pp 2070–2105. https://doi.org/10.1007/s11227-018-2701-2

[32] Mung Chiang, Tao Zhang. Fog and IoT: An Overview of Research Opportunities. Published in: 2016 IEEE Internet of Things Journal, Volume 3, Issue 6, pp 854-864. https://doi.org/10.1109/JIOT.2016.2584538

[33] Flávia Pisani, Vanderson Martins do Rosario, Edson Borin. Fog vs. Cloud Computing: Should I Stay or Should I Go? Published in: 2019 Proceedings of the INTelligent Embedded Systems Architectures and Applications, pp 27-32. https://doi.org/10.3390/fi11020034

[34] Amir Vahid Dastjerdi, Rajkumar Buyya. Fog Computing: Helping the Internet of Things Realize Its Potential. Published in: 2016 IEEE, Computer Volume 49, Issue 8, pp 112-116. https://doi.org/10.1109/MC.2016.245

[35] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, Volker Markl. Benchmarking Distributed Stream Data Processing Systems. Published in: 2018 IEEE 34th International Conference on Data Engineering (ICDE). https://doi.org/10.1109/ICDE.2018.00169

[36] Hwejoo Lee, Junghyun Oh, Kyungrae Kim, Hunje Yeon. A data streaming performance evaluation using resource constrained edge device. Published in: 2017 IEEE 2017 International

Conference on Information and Communication Technology Convergence (ICTC) in South Korea. https://doi.org/10.1109/ICTC.2017.8191055

[37] Roger Young, Sheila Fallon,Paul Jacob. An Architecture for Intelligent Data Processing on IoT Edge Devices. Published in: 2017 UKSim-AMSS 19th International Conference on Computer Modelling & Simulation (UKSim). https://doi.org/10.1109/UKSim.2017.19

[38] En Li, Liekang Zeng, Zhi Zhou, Xu Chen. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. Published in: 2020 IEEE Transactions on Wireless Communications Volume 19, Issue 1, January 2020, Article number 8876870, pp 447-457. DOI: 10.1109/TWC.2019.2946140

[39] Apache NiFi In Depth. https://nifi.apache.org/docs/nifi-docs/html/nifi-in-depth.html (2020.05.03)

Lisad

Lisa 1. Repositoorium

Käesoleva töö raames konstrueeritud testides kasutatud koodid ning *template* on kättesaadavad Bitbucketi repositooriumis, mis asub veebiaadressil https://bitbucket.org/kaire/apache_minifi_raspberrypis/.

Kaustas “Koodid“ on olemas Pythoni koodid testandmete genereerimiseks ning analüüsimiseks. Samuti on olemas MiNiFi poolt käivitatav Pythoni skript. Kaustas “Nifi_templates“ asuvad MiNiFi ja NiFi voo *template*’id.

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kaire Koljal**,

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

Apache MiNiFi efektiivsus servandmetöötluse raamistikuna,

mille juhendaja on Pelle Jakovits,

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, alates **08.05.2020** 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.

Kaire Koljal

08.05.2020