VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Matti Rita-Kasari

# ÄLYKKÄÄN SÄHKÖVERKON MITTAUSDATAN TALLENNUS-JÄRJESTELMÄ

Tekniikan yksikkö
2015

# TIIVISTELMÄ


| | |
|---|---|
| Tekijä | Matti Rita-Kasari |
| Opinnäytetyön nimi | Älykkään sähköverkon mittausdatan tallennusjärjestelmä |
| Vuosi | 2015 |
| Kieli | suomi |
| Sivumäärä | 35 + 0 liitettä |
| Ohjaaja | Kalevi Ylinen |



Opinnäytetyön aiheena on tallennusjärjestelmän suunnittelu ja toteutus Vaasan Sundomin Smart Grid -tutkimusprojektiin. Järjestelmän tehtävänä on tallentaa sähköverkon laitteilta broadcast-streamina lähetettävä mittausdata kahdennetusti siten, ettei yksikään yksittäinen verkko-, sovellus- tai laitevirhe keskeytä tallennusta. Tallennetusta datasta suodatetaan tiettyjen suojausfunktioiden viestit, jotka indeksoidaan tietokantaan. Tallennettu data pystytään hakemaan aikakriteereiden mukaan järjestelmään toteutetun REST-rajapinnan kautta.

Toteutettava työ jakautuu järjestelmä- ja ohjelmisto-osaan. Järjestelmäosa sisältää palvelintietokoneiden käyttöönoton ja konfiguroinnin. Ohjelmisto-osa taas sisältää tallennusohjelmiston toteutuksen sekä käyttöönoton. Järjestelmä käyttää Debian Linux -käyttöjärjestelmiä jaettuna LXC-containereihin. Ohjelmisto toteutetaan käyttäen Python-ohjelmointikieltä.



Avainsanat Smart Grid, broadcast, REST, Debian, Linux, LXC, Python

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikan koulutusohjelma

# ABSTRACT

| | |
|---|---|
| Author | Matti Rita-Kasari |
| Title | A Storage System for Smart Grid Measurement Data |
| Year | 2015 |
| Language | Finnish |
| Pages | 35 + 0 Appendices |
| Name of Supervisor | Kalevi Ylinen |

The subject of this thesis is the design and implementation of a storage system for Sundom Smart Grid research project. The system stores measurement data from the power grid sent using broadcast packets over an ethernet redundantly so there will not be a single point of failure. Certain protection function messages will be filtered from the stream and stored into a database. The stored data can be fetched using REST API by time criteria.

The work is divided into a system and software sections. The system section consists of server configuration and deployment. The software part consists of storage software implementation and deployment. The system uses Debian Linux operating system in LXC-containers. The software is implemented using Python programming language.

Keywords SmartGrid, broadcast, REST, Debian, Linux, LXC, Python

# SISÄLLYS

**KÄSITELUETTELO**

| | |
|---|---|
| Smart Grid | Älykäs sähköverkko. |
| INKA | Innovatiiviset kaupungit. |
| REST | Representational State Transfer |
| CRUD | Create Read Update Delete |
| LXC | Käyttöjärjestelmätason virtualisointiteknologia. |
| KVM | Kernel-based Virtual Machine |
| Python | Tulkattava olio-ohjelmointikieli. |
| Pyro | TCP-tasolla toimiva etäsuorituskirjasto Python -ohjelmointikielelle |
| IP | Internet Protocol, OSI-mallin Internet -kerroksen protokolla |
| TCP | yhteydellinen tietoliikenneprotokolla |
| SMV | Sampled Measured Values, mitatut näytteistetyt arvot |
| GOOSE | Generic Object Oriented Substation Event |
| Elasticsearch | NoSQL-tyyppinen tietokanta ja hakumoottori |
| L2 | Layer 2, OSI-mallin toinen kerros |
| XML | Extensible Markup Language |
| BIOS | Basic Input Output System |
| MAD | Mean Absolute Deviation |
| deduplikointi | Datan kompressiometodi, jossa poistetaan duplikaatit. |
| serialisointi | Datamallin muuntaminen eri muotoon tallennusta tai siirtoa varten. |

## KUVIO- JA TAULUKKOLUETTELO

# 1 JOHDANTO

Tämä työ perustuu Sundom Smart Grid INKA -ohjelman tutkimushankkeeseen. Projektin toteuttavat yritykset Anvia, ABB, Vaasan Sähkö sekä Vaasan yliopisto, joista tämä työ toteutetaan Anvia Oy:lle. Työn osuus projektista on sähköverkosta broadcast-streamina lähetettävän mittausdatan tallentaminen varmennetusti Anvian konesaleissa sijaitsevaan tallennusjärjestelmään siten, että lopullinen tallennettu data on kuitenkin deduplikoitu ja pakattu kapasiteetin säästämiseksi. Tallennettu data tulee pystyä myös hakemaan tietyillä kriteereillä tallennusjärjestelmästä. Työhön kuuluu tallennusjärjestelmän suunnittelu ohjelmistotasolla, tallennus- ja tiedonhakuohjelmiston suunnittelu sekä ohjelmistokokonaisuuden toteutus ja dokumentaatio.

Yksi projektin tärkeimmistä tavoitteista on saada kaikki mahdollinen sähköverkosta mitattu data varmennetusti tallennettua jatkoanalyysia varten. Sähköverkosta mitattu data on pakattu SMV-tyyppiseen streamiin, joka sisältää releiltä mitattuja jännite-, virta- ja taajuusarvoja joista muutoksia etsimällä pyritään havainnoimaan uusia sähköverkossa tapahtuvia ilmiöitä. Näiden analyysien perusteella tullaan jatkossa kehittämään esimerkiksi uusia suojausfunktioita, joita hyödynnetään sähköverkon laitteissa.

Suurimmaksi haasteeksi tässä työssä muodostui datan broadcast-tyypistä johtuva analysoinnin vaikeus. Koska käytössä ei ole mitään IP-kerroksen teknologioita, jouduttiin datan tallennuksen varmentamiseen kehittämään uusia menetelmiä.

# 2 SUNDOM SMART GRID

## 2.1 Projektin tavoitteet

Sundom Smart Grid on INKA -ohjelman projekti jonka pääpainona on tutkia älykkäiden sähköverkkojen mahdollisuuksia. INKA-ohjelma on yleisellä tasolla tarkoitettu luomaan uutta liiketoimintaa.

”INKA – innovatiiviset kaupungit -ohjelman tavoitteena on synnyttää korkeaan osaamiseen perustuvia kilpailukykyisiä yrityksiä ja siten vauhdittaa innovaatiokeskittymien syntymistä Suomeen.” /1/

Tämän kyseisen INKA-projektin tavoitteina on kehittää älykkäiden sähköverkkojen teknologiaa ja luoda uutta osaamista energiasektorilla Suomeen. Projektin aikana kerättyä dataa analysoidaan ja saatujen tietojen avulla pyritään kehittämään täysin uudenlaisia metodeja sähköverkon toiminnan parantamiseksi. Analysoinnin kohteena on myös kuituverkon hyödyntäminen sähköverkon laitteiden tiedonsiirtomenetelmänä.

## 2.2 Älykäs sähköverkko

Älykäs sähköverkko eli Smart Grid yhdistää useita teknologioita ja pyrkii ratkaisemaan monia loppukäyttäjän ongelmia. Sillä ei ole yhtä yksiselitteistä kuvausta joten alle on koottu joidenkin auktoriteettien näkemykset:

**European Technology Platform:**
Smart Grid on sähköverkko, joka integroi älykkäästi kaikkien siihen liittyneiden käyttäjien toiminnot – generaattorit, kuluttajat ja ne jotka ovat molempia - mahdollistaakseen kestävän, taloudellisen ja turvallisen energian saatavuuden. /2/

**US Department of Energy:**
Älykäs sähköverkko käyttää digitaaliteknologiaa parantaakseen sähköjärjestelmän luotettavuutta, turvallisuutta ja tehokkuutta hyödyntäen kasvavaa määrää hajautettuja sähkögeneraattoreita ja tallennusjärjestelmiä. /3/

## 2.3 IEC61850-standardi

IEC61850 on sähköasema-automaatiostandardi, joka on kehitetty kansainvälisen sähköalan standardiorganisaation valvonnassa yhtenäistämään sähköverkkojen laitteiden kommunikaatiometodeja. Standardi sisältää referenssin koko sähköaseman datamallista edeten hierarkkisesti yksittäisen laitteen konfiguraatioon asti. Tässä työssä käytetään IEC61850-standardin määrittämiä GOOSE- sekä SMV-malleja tiedon tallentamiseen ja analysointiin. /4/

## 2.4 Anvia

”Anvia on kasvava, tieto-, viestintä- ja turvateknologian palveluita ja tuotteita tarjoava konserni ja Suomen neljänneksi suurin tietoliikenneoperaattori. Anvia tarjoaa nykyaikaiset ja korkealaatuiset ratkaisut kuluttajien ja yritysten viestintään, tietohallintoon ja turvallisuuteen.” /11/

Anvian osuus Sundom Smart Grid -projektista on toteuttaa moderni kuituverkko sähköasemille ja niiltä konesaleihin sekä tuottaa edellytykset datan pitkäaikaiseen tallentamiseen. Kuituverkon käyttäminen sähköasema-automaatiossa on nykyään vielä harvinaista ja osa tämän projektin tavoitteista onkin tutkia sen toimivuutta ja uusia mahdollisuuksia. Tämä työ keskittyy tallennusjärjestelmän sekä -ohjelmiston suunnitteluun ja toteutukseen.

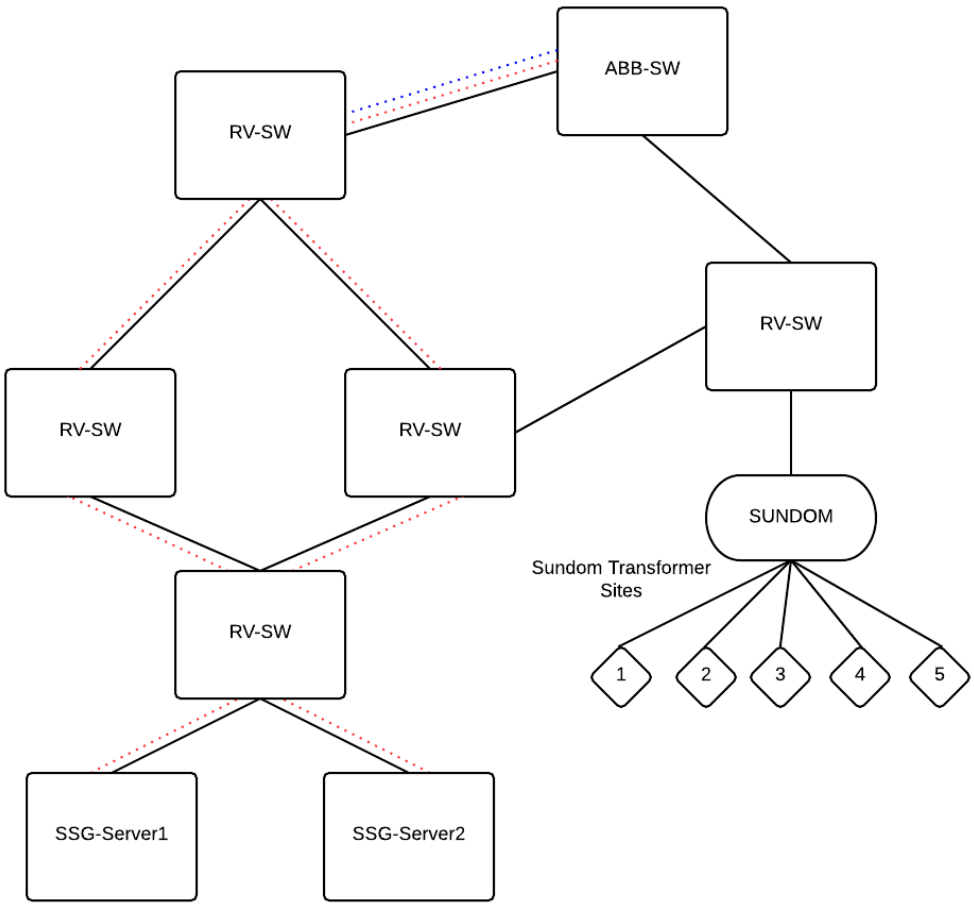
Tutkimusprojekteissa kerättävän tiedon yhtenäisyys tulee olla taattua tulosten luotettavuuden takaamiseksi. Tästä syystä verkkojen sekä datan tallennuksen luotettavuus ovat erittäin tärkeässä roolissa tämän projektin aikana.

# 3 ARKKITEHTUURIKUVAUS

Projektikokonaisuus koostuu Anvian osalta kolmesta suuremmasta kokonaisuudesta: verkosta, järjestelmästä sekä ohjelmistosta. Tämä työ rajautuu järjestelmän sekä ohjelmiston suunnitteluun ja toteutukseen.

## 3.1 Verkkoarkkitehtuuri

Kuvassa 1 on esitettynä runkoverkon topologia Sundomin sähköasema-alueen sekä Anvian konesalien osalta. Kuvasta huomataan siirtoteiden kahdennukset katkoviivoilla kytkinten, SW, välillä. Tällä pystytään eliminoimaan palvelukatkokset yksittäisen laitteen tai siirtotien vikatilanteessa.



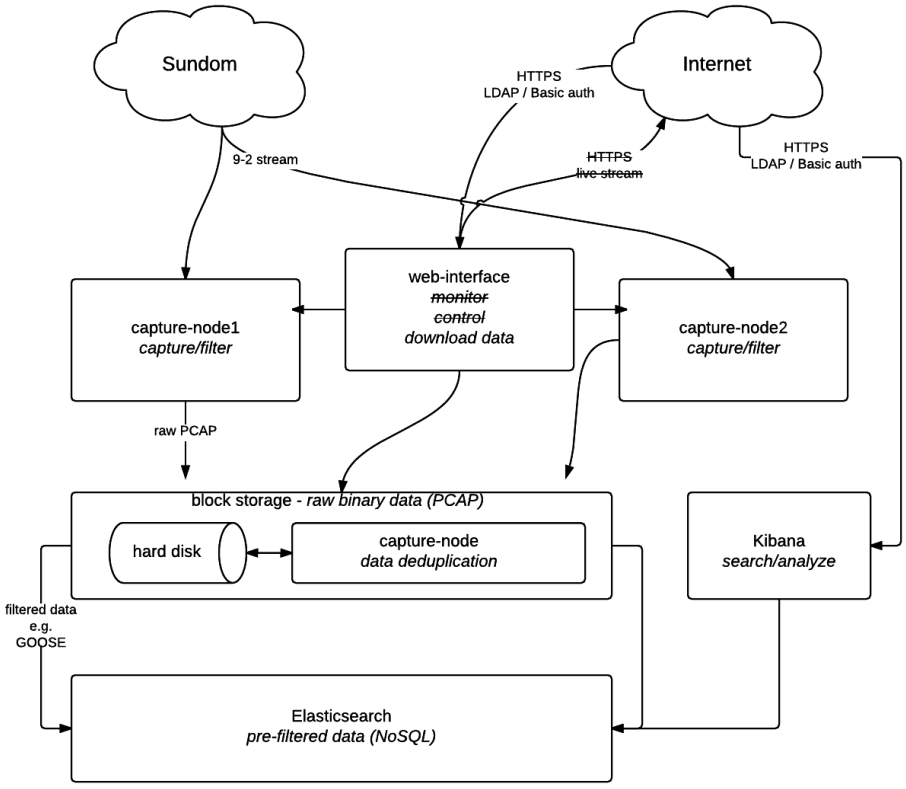
**Kuvio 1.** Verkkoarkkitehtuuri.

### 3.2 Järjestelmäarkkitehtuuri

Järjestelmä koostuu kahdesta fyysisestä palvelintietokoneesta, levyjärjestelmästä ja tietokantaklusterista. Data-stream tuodaan L2-tasolla capture-nodeille, jotka pitävät tallenteita puskurissa maksimissaan vuorokauden ajan. Puskurissa oleva data siirretään sitten storage-nodelle, jossa se deduplikoidaan ja tallennetaan levyjärjestelmään. Ennen tallentamista datasta suodatetaan GOOSE-viestit, jotka serialisoidaan XML-muotoon ja tallennetaan Elasticsearch-klusteriin.

Palvelintietokoneet on sijoitettu fyysisesti eri sijainteihin Vaasan alueella. Tällä pystytään varmistamaan tallennetun datan yhtenäisyys verkko- ja laitevirheiden aikana. Järjestelmän klusterityyppinen luonne asettaa tiettyjä edellytyksiä myös toteutettavalle ohjelmistolle. Ohjelmiston tulee sopeutua kaikkiin mahdollisiin järjestelmän virhetilanteisiin ja palautua niistä automaattisesti.

Kuvassa 2 nähdään järjestelmän arkkitehtuurikuvaus korkealla tasolla. Järjestelmäkuvaus kattaa loogiset komponentit ottamatta kantaa verkon tai laitteiston fyysisiin ominaisuuksiin. Kuvassa on esitetty yhteydet järjestelmään Internetistä ja Sundomin sähköaseman alueelta sekä järjestelmän sisäiset loogiset yhteydet. Nodet voidaan ajatella erillisinä järjestelmämoduuleina jotka sisältävät käyttöjärjestelmätasolla isoloidun ympäristön. Tallennusjärjestelmät block storage ja Elasticsearch ovat itsenäisiä järjestelmiä, joihin tämä järjestelmä liitetään.



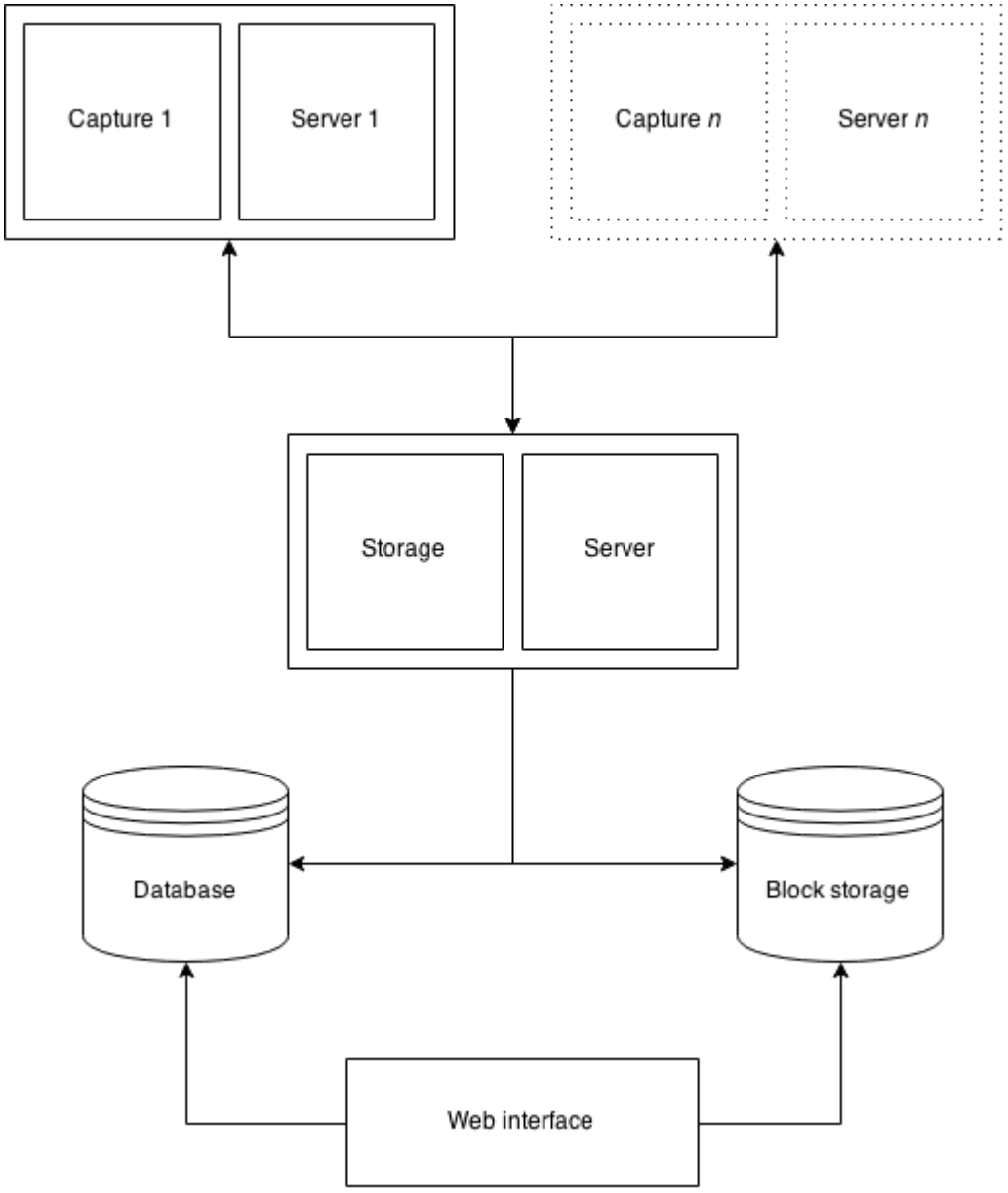

**Kuvio 2.** Järjestelmäarkkihtehtuuri.

### 3.3 Ohjelmistoarkkitehtuuri

Ohjelmisto koostuu kolmesta pääkomponentista: capture-, storage- ja server-nodesta. Jokainen näistä komponenteista toimii itsenäisenä sovelluksenaan ja kommunikaatio niiden välillä toteutetaan TCP-protokollalla käyttäen Pyro-kirjastoa. Ohjelmisto on toteutettu käyttäen Python-ohjelmointikieltä sen korkean abstraktiotason ja kevyen implementaation vuoksi.

Pääkomponenttien lisäksi ohjelmistokokonaisuuteen toteutetaan REST-rajapinta, jonka kautta tallenteita pystytään hakemaan tietyillä kriteereillä. Tätä rajapintaa

hyödynnetään myöhemmin toteutettavassa käyttöliittymässä, jonka kautta loppukäyttäjä pystyy hakemaan tallenteet omalle tietokoneelleen analysointia varten.




**Kuvio 3.** Ohjelmistoarkkitehtuuri.

# 4 JÄRJESTELMÄN SUUNNITTELU

## 4.1 Käyttöjärjestelmä

Järjestelmä toteutettiin käyttäen käyttöjärjestelmänä Debian Wheezy Linuxin versiota 7.8.0. Debian on Linux-kerneliin pohjautuva avoimen lähdekoodin käyttöjärjestelmä, jota ylläpitää tuhansien kehittäjien yhteisö Debian Projectin alaisuudessa. Suurin osa käyttöjärjestelmän työkaluista tulee GNU Projectista joka, on myös kokoelma avoimen lähdekoodin ohjelmistoja. Debian tuo Linux-käyttöjärjestelmään mukanaan hyvin ylläpidetyn paketinhallintajärjestelmän, joka helpottaa huomattavasti erilaisten ohjelmistokokonaisuuksien toteuttamisen käyttäen Linux-ydintä. /5/

## 4.2 Virtualisointi ja containerit

Virtuaalinen tietokone on looginen vastine fyysisestä tietokoneesta. Käytännössä tämä mahdollistaa useamman käyttöjärjestelmän tai käyttäjäympäristön suorittamisen käyttäen yhtä fyysistä instanssia tietokoneesta. Tämä mahdollistaa koko tietokoneen resurssien hyödyntämisen tehokkaammin sekä helpottaa eri palveluiden eristämistä toisistaan. Tässä projektissa hyödynnetään käyttöjärjestelmätason virtualisointia ja otetaan vertailukohteeksi myös täysimittainen virtualisointi. /6/

Täysimittaisessa virtualisoinnissa emuloidaan koko tietokone käyttöjärjestelmän ytimestä ja BIOS:sta lähtien. Tämä mahdollistaa parhaan mahdollisen kontrollin virtuaalikoneesta antaen käyttäjälle mahdollisuuden käsitellä sitä kuin fyysistä laitetta. Huonona puolena täysimittaisessa virtualisoinnissa on tiettyjen resurssien moninkertainen kuluttaminen. Esimerkiksi ajettaessa useaa rinnakkaista käyttöjärjestelmää, joudutaan lataamaan sama ydin ja perustyökalut uudestaan jokaista instanssia kohden.

Käyttöjärjestelmätason virtualisoinnissa käyttöjärjestelmän ydin ladataan vain kerran, mutta kaikki käyttäjätason ohjelmistot eristetään toisistaan luoden vaikutelman itsenäisestä järjestelmästä. Tämä lähestymistapa kuluttaa resursseja vähemmän ja

on hieman tehokkaampaa kuin täysimittainen virtualisointi. Haittapuolena käyttöjärjestelmätason virtualisoinnissa on käyttäjän kontrolli käyttöjärjestelmästä. Vaikka käyttäjälle luodaan täysin dedikoitu ympäristö, mahdollisuudet muokata käyttöjärjestelmän ominaisuuksia ovat rajoitetut. Käyttöjärjestelmätason virtualisointi ei myöskään mahdollista eri käyttöjärjestelmäytimien kuin isäntätietokoneessa virtualisointia.

Tämän järjestelmän toteutuksen aikana testattiin yleisellä tasolla virtualisoinnin vaikutusta käyttöjärjestelmän vasteaikaan. Virtualisointi on lähes välttämätöntä järjestelmän osiin jakamisen kannalta ja tavoitteena oli etsiä projektin kannalta tehokkain vaihtoehto. Erityisen tärkeiksi tekijöiksi nousivat verkon suorituskyky sekä aikasynkronoinnin tarkkuus käyttöjärjestelmän tasolla. Verkon suorityskyvyn tulee luonnollisesti vastata järjestelmän vaatimuksia datan tallennuksen mahdollistamiseksi. Aikasynkronointi on tärkeää, koska jokainen tallennettu paketti tulee tunnistaa mahdollisimman tarkasti tilanteiden jatkoanalyysin mahdollistamiseksi.

### 4.2.1 KVM

KVM, Kernel-based Virtual Machine, on Linuxin ytimeen integroitu moduuli, joka mahdollistaa täysimittaisen virtualisoinnin rautavirtalisointia tukevilla laitteistoilla. Se on ollut osa Linuxin kerneliä Tammikuusta 2007 lähtien ja on saavuttanut täysin vakaan kehitystason. KVM on täysin avoin ohjelmisto tarjoten silti vakaan toiminnan, mikä onkin syy sen kasvavaan suosioon. /7/

### 4.2.2 LXC

LXC, Linux Containers, on käyttöjärjestelmätason virtualisointiteknologia, jolla pystytään ajamaan useaa Linux-instanssia yhdessä tietokoneessa. Yhdistäen käyttöjärjestelmäytimen eristetyt nimiavaruudet sekä cgroups-ohjelmiston, LXC luo täysin itsenäisen käyttöjärjestelmäympäristön jokaista instanssia varten. /8/

### 4.2.3 Vasteaikavertailu

Verkon suorituskyvyn ollessa erittäin tärkeää suurien datamäärien tallentamisessa aikakriittisesti, testattiin kahden eri virtualisointiteknologian vaikutuksia verkkolaitteiden viiveisiin. Testin kohteeksi otettiin kaksi yllä mainittua teknologiaa: LXC ja KVM. LXC-järjestelmässä isäntätietokoneen verkkokortti on paljastettu suoraan virtuaalikoneelle ja KVM-järjestelmässä Linux virtual bridgen, eli sillatun virtuaalisen verkkokortin kautta.

Testissä mitattiin viivettä virtualisoidulle käyttöjärjestelmälle paljastetulta verkkokortilta lähimmälle fyysiselle laitteelle. Testi toteutettiin käyttäen Linux-käyttöjärjestelmään kuuluvia verkkotyökaluja. Toteutettujen iteraatioiden määrä oli 10 kpl ja vasteaika mitattiin millisekunteina.

Testin tulokset:

**Taulukko 1.** Vasteaikavertailu.

| Iteraatio | LXC (ms) | KVM (ms) |
|---|---|---|
| 1 | 0.025 | 0.146 |
| 2 | 0.026 | 0.130 |
| 3 | 0.027 | 0.123 |
| 4 | 0.026 | 0.138 |
| 5 | 0.025 | 0.118 |
| 6 | 0.026 | 0.137 |
| 7 | 0.026 | 0.158 |
| 8 | 0.026 | 0.137 |
| 9 | 0.025 | 0.161 |
| 10 | 0.024 | 0.165 |
| *MAD* | *0.005* | *0.0115* |

Testin tuloksista voidaan arvioida, että LXC-virtualisointi suoralla yhteydellä isäntäjärjestelmän verkkokorttiin on noin 5.5x nopeampi vasteajaltaan kuin KVM-virtualisointi käyttäen Linux virtual bridgeä. Testissä on otettava huomioon, että käytössä oli kaksi hieman eri spesifikaatiolla olevaa palvelintietokonetta, joiden suorituskyky vaikutta testituloksiin.

Testin tulos oli täysin odotettavissa, koska virtuaalisten laitteiden käyttäminen tuottaa poikkeuksetta viivettä. Vasteaikaeron vuoksi järjestelmässä päädyttiin käyttämään LXC-virtualisointia.

### 4.3 Block storage

Järjestelmän pääasiallisena tallennustyyppinä käytetään block storagea eli käyttöjärjestelmän raakaa levypintaa. Tähän ratkaisuun päädyttiin tallennettavan datan harvinaisen muodon ja suuren määrän vuoksi. Tallennettavan datan ollessa binäärimuotoista, sen tallentamisessa tietokantaan ei ole juuri merkittävää etua. Datan serialisointi tekstimuotoon ennen tietokantaan tallentamista taas kasvattaisi sen kokoa niin huomattavasti, ettei normaalien tallennusmedioiden kirjoitusnopeus enää riittäisi sen tallentamiseen.

### 4.4 Tietokanta

Yksilöidyn datan tallentamiseen käytetään ELK-stackia, joka sisältää kolme komponenttia: Elasticsearch tietokannan ja hakumoottorin, Logstash-lokinhallintaympäristön sekä Kibana-käyttöliittymän. Näistä komponenteista käytetään tämän työn aikana pääosin Elasticsearchia ja myöhemmässä analysointivaiheessa Kibanaa.

#### 4.4.1 Elasticsearch

Elasticsearch on avoimen lähdekoodin NoSQL-tyyppinen tietokanta sekä hakumoottori, joka on nostanut suosiotaan tehokkuutensa ja skaalautuvuutensa vuoksi.

Sitä kehitetään avoimen lähdekoodin yhteisön lisäksi Elasticsearch BV -yhtiön toimesta. Elasticsearchin tärkeimpiin ominaisuuksiin kuuluvat datan reaaliaikaisuus, mahdollisuus korkean saatavuuden ympäristöihin sekä helppokäyttöinen REST-rajapinta. Kaikki nämä ominaisuudet mahdollistaa taustalla käytettävä Apache Lucene. Se on Apachen kehittämä avoimen lähdekoodin tiedonhakukirjasto joka mahdollistaa tiedostoformaatista riippumattoman tekstipohjaisen haun ja indexoinnin. /9/

### 4.4.2 API

Elasticsearch sisältää REST-rajapinnan, joka noudattaa täysin CRUD-periaatetta. Rajapinnat toteutuvat CRUD:n mukaisesti samassa järjestyksessä: Index API, Get API, Update API ja Delete API. Saatavilla on myös useampien dokumenttien käsittelyyn tarkoitettuja rajapintoja, mutta niiden toiminnallisuuksia ei tässä sovelluksessa käytetä. /10/

# 5 OHJELMISTON SUUNNITTELU

Ohjelmmistokokonaisuus koostuu kolmesta moduulista: capture-node, storage-node ja server-node. Nämä kaikki moduulit toimivat itsenäisinä prosesseina mutta niiden toiminnallisuutta voidaan yhdistellä riippuen järjestelmäarkkitehtuurista. Tämän työn lopputoteutuksessa ohjelmistot on jaettu siten, että capture-node on kahdennettu ja tallennuksesta huolehtii yksi storage-node. Jokaisella moduulilla on myös oma server-node, joka huolehtii moduulien välisestä kommunikaatiosta ja tiedon siirrosta.

## 5.1 Capture-node

Datan kaappaamisesta sekä ensivaiheen tallentamisesta huolehtii capture-node. Se koostuu Python-skriptistä, joka tallentaa kaiken verkkolaitteella tapahtuvan liikenteen.

### 5.1.1 python-libpcap

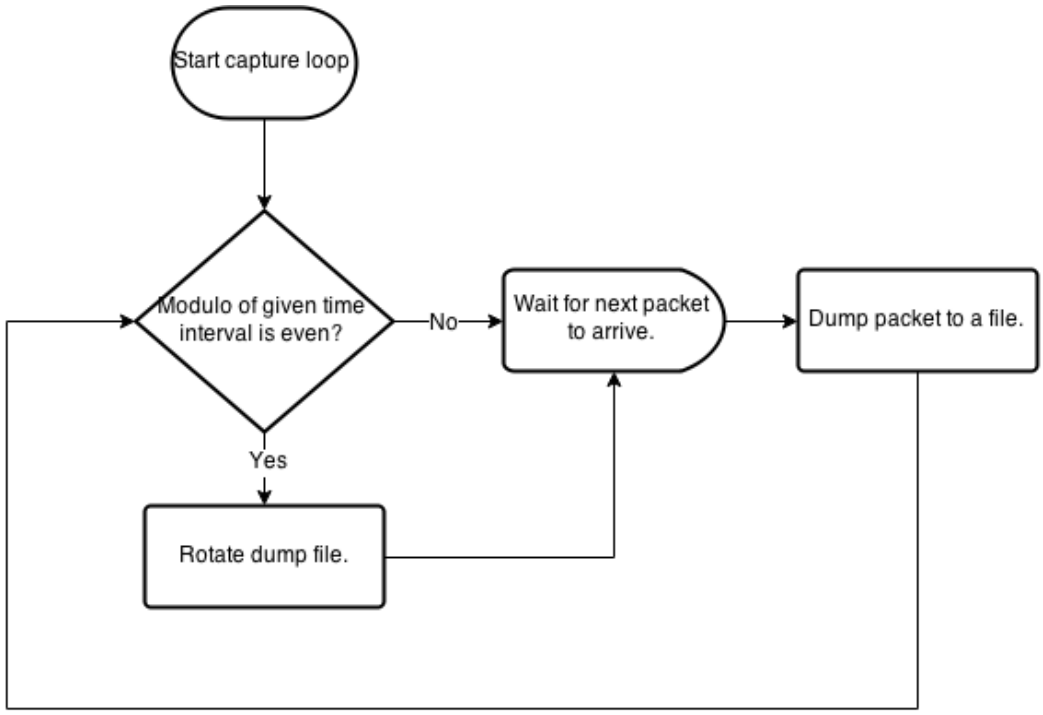
Datan ollessa broadcast-liikennettä, täytyy verkkokortti asettaa vastaanottamaan kaiken verkossa liikkuvan datan riippumatta sen tyypistä. Tämän helpottamiseksi hyödynnettiin valmista paketinkaappauskirjastoa nimeltä libpcap. Koska ohjelmisto toteutettiin käyttäen Python-ohjelmointikieltä, tarvittiin kyseisestä kirjastosta tälle ohjelmointikielelle toteutettu implementaatio python-libpcap.

### 5.1.2 Moduulikuvaus

Capture-node on sovelluksen kolmesta osasta kaikista yksinkertaisin. Se koostuu yhdestä Python-skriptistä, joka tallentaa verkkokortilla tapahtuvan liikenteen PCAP-tiedostomuotoon tietokoneen kovalevylle. Tässä järjestelmässä capture-node on konfiguroitu pitämään minuutin pituisia tallenteita puskurissa maksimissaan vuorokauden, jonka jälkeen ne siirretään storage-nodelle jälkikäsittelyä varten.

Kuvassa 4 on esitetty capture-noden toiminta vuokaaviona. Kaaviosta nähdään moduulin toiminta pääpiirteittäin. Joka paketin välissä tarkistetaan, onko tämän hetken

aikaleiman jakojäännös annetun aikaintervallin suhteen nolla. Jos on, vaihdetaan tallennettavan tiedoston nimeä. Tällä metodilla saadaan toteutettua yksinkertainen tiedostonimen kierrätys kuluttamatta laskentatehoa juurikaan.




**Kuvio 4.** Paketinkaappausmoduulin vuokaavio.

Alla olevassa esimerkkikoodissa on esitetty paketinkaappausalgoritmin toteutus Python-ohjelmointikielellä.

```
# Main routine for capture
while self.is_running:
    # Rotate path on defined interval
    if time.localtime().tm_min % int(self.config['interval']) == 0:
        self.rotate_file()
    # Try/catch for empty frames
    try:
        # Extract data from capture and move to next frame
        (header, packet) = self.capt.next()

        # Dump capture into .pcap -file
        self.dumper.dump(header, packet)
    except pcapy.PcapError:
        continue
```

## 5.2 Storage-node

Tiedostojen siirtäminen capture-nodeilta lopulliseen arkistoon, sekä datan deduplikointi ja suodattaminen toteutetaan storage-node -moduulissa. Moduuli koostuu tallennusosasta sekä Pyro4-asiakasosasta, joka on abstraktoitu erillisen luokan sisään.

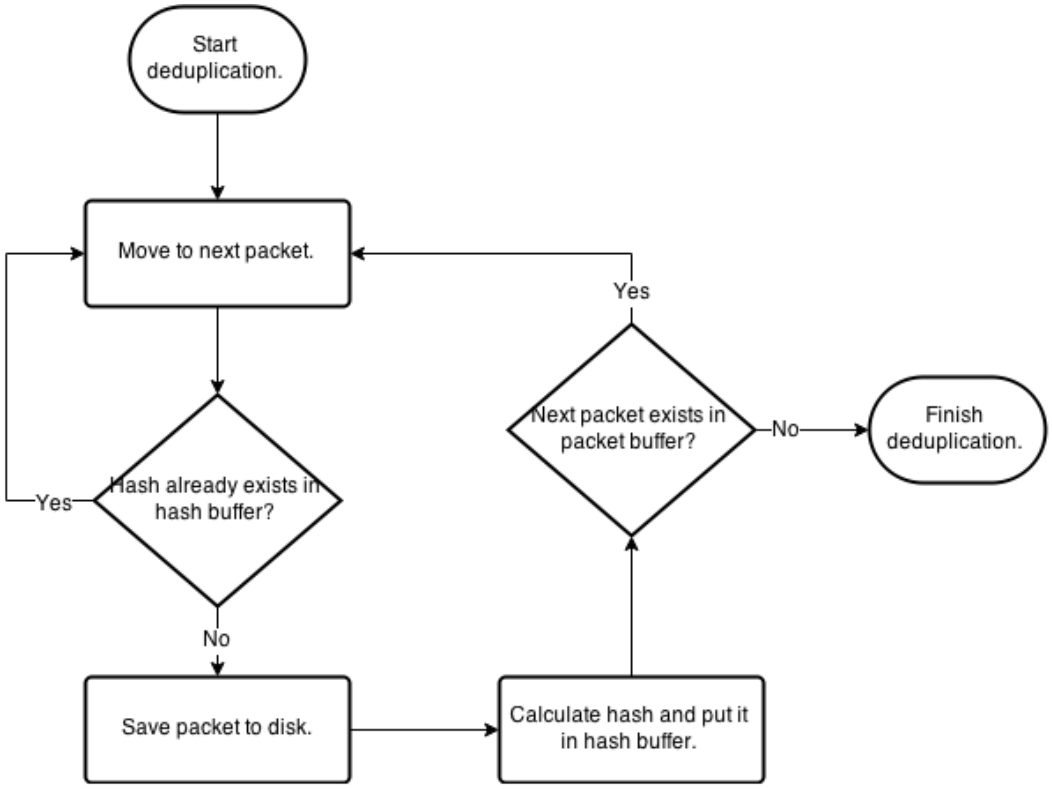
### 5.2.1 Datan noutaminen

Data noudetaan capture-nodeilta käyttäen Pyro4-etäsuorituskirjastoa. Tämän kirjaston avulla pystytään suorittamaan funktioita Python ohjelmointikielellä TCP-yhteyden yli toisella palvelimella. Tähän tarkoitukseen toteutettiin erillinen luokka, joka sisältää funktiot tiedostojen listaamiseen ja noutamiseen capture-nodeilta. Pyro4-kirjaston korkean abstraktiotason ansiosta kommunikaation toteuttaminen moduulien välille oli todella helppoa. Kaikki tarvittavat ominaisuudet, kuten siirrettävän datan serialisointi, kuuluvat kirjaston sisäänrakennettuihin ominaisuuksiin, joten tehtäväksi jäi ainoastaan tarvittavien funktioiden toteuttaminen.

### 5.2.2 Datan deduplikointi

Koska identtinen data tallennetaan kahdennetusti, on tallennuskapasiteetin säästämiseksi syytä poistaa datasta duplikaatit. Tämä toteutetaan storage-nodella ennen datan lopullista tallentamista. Deduplikointi toteutetaan yhdistämällä paketit ensin aikajärjestykseen, jonka jälkeen käymällä läpi jokainen paketti poistaen duplikaatit. Paketeista laskettuja tarkastussummia pidetään rengaspuskurissa muistissa tietty määrä, jota vasten pystytään vertaamaan, onko paketti tallennettu jo kertaalleen.

Alla olevasta vuokaaviosta nähdään datan deduplikointialgoritmin toiminta. Algoritmi käy jokaisen tallennetun paketin läpi ja tarkistaa, onko paketin MD5-tarkastussumma puskurissa. Jos summa täsmää, on paketti jo tallennettu ja siirrytään seuraavaan pakettiin. Jos taas summaa ei löydy puskurista, lisätään se sinne ja tallennetaan paketin sisältö levylle.




**Kuvio 5.** Deduplikointialgoritmin vuokaavio.

Alla olevassa esimerkkikoodissa on esitetty deduplikointialgoritmin toteutus Python-ohjelmointikielellä.

```
while True:
        try:
            # Read next packet from data stream
            (header, packet) = reader.next()
            # Calculate MD5-sum
            hash = hashlib.md5(packet).digest()
            # Check if hash is in memory
            if hash in dumphash:
                # Ignore packet
                continue
            else:
                # Dump packet and save hash to buffer
                dumper.dump(header, packet)
                dumphash.append(hash)
        except pcapy.PcapError:
            print pcapy.PcapError
            break
```

### 5.2.3 Datan suodattaminen

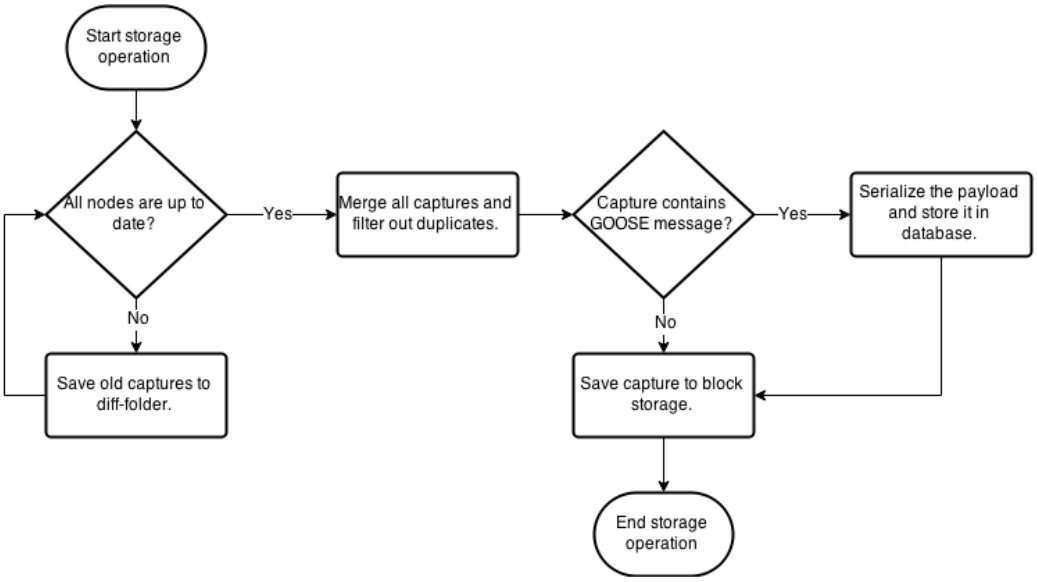
Deduplikoinnin jälkeen datasta suodatetaan GOOSE-paketit, jotka tallennetaan block storagen lisäksi myös Elasticsearch-tietokantaan. Suodattaminen toteutetaan etsimällä paketin Ethernet-kehyksestä GOOSE-viestille ominaisia merkkijonoja joiden perusteella pystytään erottelemaan paketit toisistaan. Suodatetut GOOSE-paketit tallennetaan ensin paikallisesti levylle jonka jälkeen ne serialisoidaan ja tallennetaan ajastetusti Elasticsearch-tietokantaan. Tallentaminen tapahtuu Elasticsearchin CRUD API:n avulla.

### 5.2.4 Datan tallentaminen

Deduplikoinnin ja suodattamisen jälkeen data tallennetaan lopulliseen arkistoon PCAP-muotoon. Tiedostot tallennetaan minuutin pituisissa osissa kansioihin kuukausittain ja nimetään päivämäärän mukaan hakuoperaatioiden helpottamiseksi. Tämän lisäksi data pakataan ajastetusti päivittäin.

### 5.2.5 Moduulikuvaus

Alla olevassa vuokaaviossa on kuvattu koko storage-noden toiminta. Ennen jokaista tallennusoperaatiota tarkistetaan, ovatko kaikki capture-nodet saatavilla ja niiden data yhtenäistä. Poikkeustilanteissa eroava data siirretään erilliseen kansioon. Jos kaikki capture-nodet ovat ajan tasalla, siirretään niiltä uusimmat paketit storage-nodelle. Tämän jälkeen data deduplikoidaan, siitä suodatetaan GOOSE-viestit ja se tallennetaan lopulliseen arkistoon. Tätä tallennusalgoritmia toistetaan moduulin pääsilmukassa konfiguraation määrittämällä aikavälillä.




**Kuvio 6.** Tallennusmoduulin vuokaavio.

## 5.3 Server-node

Sovelluksen kaikkien osien etäsuoritusominaisuudet on paketoitu server-node-moduuliin. Moduuli koostuu Pyro4-kirjastolla toteutetusta palvelinsovelluksesta jonka kautta pystytään etäsuorittamaan Python-funktioita moduulien välillä.

### 5.3.1 Pyro4-kirjasto

Pyro4, Python Remote Objects, on Python-kirjasto, joka sallii sovellusten keskustelun toistensa kesken TCP-yhteyden välityksellä. Pyro4 toteuttaa hyvin korkean tason abstraktion kommunikaatioon jättäen ohjelmoijan tehtäväksi käytännössä vain suoritettavien funktioiden suunnittelun. Tässä sovelluskokonaisuudessa kirjastoa käytetään tiedostojen listaamiseen ja siirtämiseen moduulien välillä. Tiedostojen siirtämiseen käytetään Pyro4:n erillistä ominaisuutta nimeltä Flame, joka sallii binääritiedostojen siirtämisen. /12/

### 5.3.2 Tiedostojen listaus

Tiedostojen listaaminen etätietokoneella toteutetaan käyttäen yksinkertaista funktiota, joka hyödyntää Pythonin sisäänrakennettua os.walk-ominaisuutta. Sen avulla

saadaan listattua tietyssä polussa sijaitsevat tiedostot ja järjestettyä ne esimerkiksi aikajärjestykseen. Tähän sovellukseen toteutettiin funktio, joka hakee tallennettujen tiedostojen nimen ja polun käyttöjärjestelmässä. Funktio palauttaa sitä kutsuvalle moduulille listan, jossa on esitettynä kyseisen tietokoneen nimi, sekä tiedostot aikajärjestyksessä.

### 5.3.3 Tiedostojen siirto

Tiedostojen siirto toteutetaan käyttäen Pyro4-kirjaston moduulia nimeltä Flame. Tämä moduuli sisältää valmiin funktion binääritiedostojen siirtämiseen Pyro4:lla muodostetun yhteyden yli. Tiedostojen siirto tapahtuu serialisoimalla ne ensin tekstimuotoon Pythonin pickle-serialisointitekniikalla ja siirtämällä tekstimuotoinen data verkon yli. Siirron jälkeen serialisoitu data muunnetaan takaisin binäärimuotoon. Tällä tavoin datan siirtäminen on hyvin tehokasta, koska siirto tapahtuu TCP-yhteyden päällä ilman erillistä HTTP-kerrosta.

### 5.3.4 Tiedostojen lataus

Server-node sisältää vaihtoehtoisen lisämoduulin, joka mahdollistaa tallennettujen tiedostojen lataamisen käyttäen REST-rajapintaa. Rajapinta toteutettiin käyttäen Python-ohjelmointikielen sisäänrakennettua HTTP-palvelinta, jolla pyynnöt reititetään oikein. Rajapinnasta pystytään hakemaan tiedostoja päivämäärän ja ajan perusteella sekunnin tarkkuudella.

Rajapinta toimii siten, että HTTP-pyynnön saapuessa se tarkistaa kutsusta päivämäärän sekä halutun aikavälin. Tämän jälkeen sovellus käy kansiorakenteen läpi ja etsii annettuja kriteerejä vastaavan tiedoston tiedostonimen mukaan. Jos haluttu aikaintervalli on pienempi kuin tiedoston koko, puretaan tiedosto ja siitä generoidaan tarkempaa aikaväliä vastaava tallenne. Tämä lopullinen tiedosto palautetaan binäärimuodossa HTTP-yhteyden yli pyynnön lähettäneelle asiakassovellukselle.

Tähän ohjelmistokokonaisuuteen toteutettu web-rajapinta ei noudata tavanomaista REST-rajapinnan määritelmään. Normaalitilanteessa REST-kutsun palautusarvo

on joko JSON- tai XML-muotoista merkkidataa, mutta tässä sovelluksessa palautetaan raakaa binääridataa. Alla on esitetty esimerkkikutsut ja niiden palautusarvot.

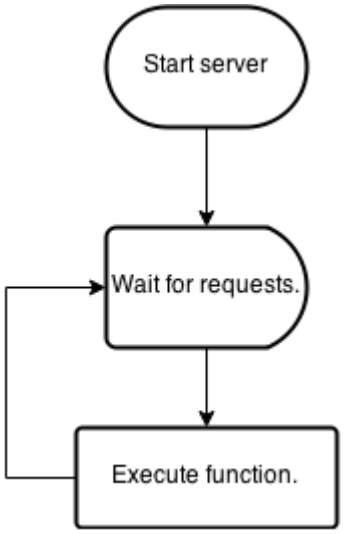
Kutsu 1:
**https://esimerkkipalvelin.fi/api/2015/01/01?length=10**
Palautusarvo:
PCAP-tiedosto joka sisältää kaiken mittausdatan ajanhetkestä 01.01.2015 kymmenen sekuntia eteenpäin.

Kutsu 1:
**https://esimerkkipalvelin.fi/api/2015/05/20?length=5**
Palautusarvo:
PCAP-tiedosto joka sisältää kaiken mittausdatan ajanhetkestä 20.05.2015 viisi sekuntia eteenpäin.

Sovelluksen ensimmäisessä versiossa tallenteita pystyy lataamaan vain sekunteina määritetyllä aikavälillä rajapinnan kautta. Tämä johtuu asiakkaan näkemyksestä, että kerralla ladattavien tallenteiden pituus ei tule olemaan useaa minuuttia.

### 5.3.5 Moduulikuvaus

Alla olevassa kuvassa on kuvattuna server-noden toiminta sekvenssikaaviona. Server-noden looginen toiminta on hyvin yksinkertainen, se odottaa etäpyyntöjä toisilta moduuleilta ja kun pyyntö saapuu, se suorittaa pyydetyn funktion. Suoritettuaan funktion loppuun, ohjelma palaa takaisin normaalitilaansa odottamaan pyyntöjä.




**Kuvio 7.** Palvelinmoduulin vuokaavio.

# 6 KÄYTTÖÖNOTTO

## 6.1 Järjestelmän käyttöönotto

Järjestelmän käyttöönotto koostuu käyttöjärjestelmän asentamisesta ja konfiguroinnista storage-nodeihin, sekä tarvittavien yhteyksien ja verkkojen käyttöönotosta. Tässä kappaleessa käydään läpi tämän työn kannalta oleelliset osat järjestelmän käyttöönotosta.

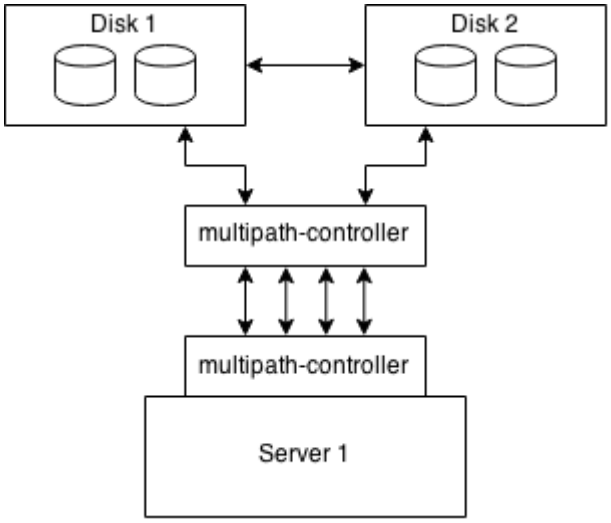
### 6.1.1 Debian Linux

Debian Linux -käyttöjärjestelmän asentaminen tapahtuu pääosin automatisoidun asennusohjelman avulla. Tässä järjestelmässä on kuitenkin käytössä normaalista poikkeavaa laitteistoa, joten myös asennusprosessi poikkeaa hieman normaalista. Debianin asennusohjelma onneksi antaa mahdollisuuden muokata asennettavan käyttöjärjestelmän ominaisuuksia asennusprosessin aikana, joten spesifimmätkin konfiguraatiot pystyttiin toteuttamaan.

### 6.1.2 Multipath

Storage-nodejen fyysiset palvelintietokoneet käyttävät multipath-tyyppistä tallennusjärjestelmää. Kuvassa 8 kuvassa on esitetty multipath-levyjärjestelmän toiminta. Multipath mahdollistaa vikasietoisen yhteyden tallennusjärjestelmään käyttämällä useampaa fyysistä yhteyspistettä. Tämä vaatii kuitenkin erityisen konfiguraation käyttöjärjestelmätasolla.

Linux-käyttöjärjestelmä käsittelee normaalitilanteessa multipath-laitteen fyysisiä yhteyspisteitä jokaista erillisenä levynään. Täyden hyödyn saamiseksi tulee käyttöjärjestelmä konfiguroida käyttämään yhteyspisteitä oikein. Tähän tarkoitukseen hyödynnettiin DM-Multipath-ohjelmistoa, joka pystytään konfiguroimaan siten, että virhetilanteessa käyttöjärjestelmä siirtyy vikasietoisesti käyttämään seuraavaa toimivaa yhteyspistettä.




**Kuvio 8.** Multipath-levyjärjestelmä.

### 6.1.3 LXC

LXC-ympäristön käyttöönotto Debian Linuxissa tapahtuu sisäänrakennetun pakетinhallinnan avulla. Debian-distribuution pakettikirjastot sisältävät vakaan version LXC:stä riippuvuuksineen, joten binäärien asentaminen tapahtuu automaattisesti. Tämän lisäksi tulee ottaa käyttöön cgroups-sovellus, jonka avulla pystytään hallinnoimaan vierasjärjestelmien käyttämiä resursseja, kuten muistia, prosessoritehoa ja levykapasiteettia.

## 6.2 Ohjelmiston käyttöönotto

### 6.2.1 Moduulit

Ohjelmiston eri moduulien käyttöönotto suoritettiin käyttäen GIT-versionhallintasovellusta. GIT mahdollistaa ohjelmiston lähdekoodin hallinnan keskitetyllä palvelimella sekä sisältää monia sovelluskehitystä helpottavia työkaluja.

Ohjelmiston moduulit ladattiin jokaiselle järjestelmän palvelimelle GIT-arkistosta, joka sisältää kaikki moduulit sekä Python-implementaation binäärit ja moduulien vaatimat riippuvuudet. Koska ohjelmisto on toteutettu Python-ohjelmointikielellä, ei käyttöönotto vaadi erillisiä käännösoperaatioita.

### 6.2.2 Upstart

Upstart on Linux-käyttöjärjestelmän sovellus, jonka avulla pystytään automatisoimaaan sovellusten suorittaminen ja vikatilanteista palautuminen. Upstart ei ota kantaa itse sovelluksen toimintaan vaan suorittaa kaikki operaatiot ulkoisilla komennoilla mahdollistaen yksinkertaisemman ohjelmistoarkkitehtuurin. Kaikki tämän ohjelmistokokonaisuuden moduulit automatisoitiin upstart:lla siten, että ne käynnistyvät aina käyttöjärjestelmän mukana ja suoritetaan vikatilanteessa uudelleen.

# 7 TESTAUS

Tämän järjestelmä- ja ohjelmistokokonaisuuden tärkeimpänä vaatimuksena on vikasietoisuus eri tilanteissa. Järjestelmän toimivuuden kannalta on myös tärkeää, että laitteiston ja verkon suorituskyky vastaa tarvittavia vaatimuksia. Järjestelmän vikasietoisuus ja suorituskyky testattiin todellista tuotantotilannetta vastaavassa testiympäristössä käyttäen tuotantoon siirtyviä palvelimia ja testilaitteita ABB:n laboratorioympäristössä.

## 7.1 Vikasietoisuus

Järjestelmän vikasietoisuus testattiin aiheuttamalla manuaalisesti mahdollisia verkon ja laitteiston häiriötilanteita. Alla on listattuna erilaiset häiriötilanteet ja niistä palautuminen:

- yhteys storage-noden ja yhden capture-noden välillä katkeaa
- yhteys storage-noden ja kaikkien capture-nodejen välillä katkeaa
- jollekin capture-nodeista saapuu virheellistä mittaustietoa
- capture- tai storage-node sammuu tai käynnistyy uudelleen.

Kaikista testatuista virhetilanteista palautuminen onnistui oletetusti. Minkä tahansa yksittäisen capture-noden virhetilanteessa järjestelmä tallentaa mahdollisesti virheellisen datan duplikaattina, mikä varmistaa kaiken datan vastaanottamisen. Storage-noden virhetilanteessa hash-puskuri tyhjenee, mikä aiheuttaa myös duplikaattidatan tallentamisen, mutta mitään dataa ei menetetä. Virheellistä tietoa ei tällä hetkellä pystytä suodattamaan pois, koska autenttista tietoa ei ole määritelty tarpeeksi tarkasti tämän projektin aikana. Kaikkien capture-nodejen sammuminen tai yhteyden sähköasemilta katkeaminen samanaikaisesti on ainoa tilanne, jossa dataa menee hukkaan.

## 7.2 Suorituskyky

Järjestelmän suorituskykyä testattiin ABB:n laboratorioympäristöä vasten tallentamalla ympäristöstä lähetettävää dataa testipalvelimille. Laboratorioympäristö vastaa lopullista tuotantoympäristöä mittalaitteiden ja yhteyksien osalta, joten testitallennuksista nähdään riittääkö järjestelmän kapasiteetti. Alla on listattuna mitatut arvot sekä niiden perusteella lasketut suorituskykyarviot:

- capture-noden verkkoliikenne sisään: 48,5 Mt/s
- storage-noden verkkoliikenne sisään: 97,0 Mt/s
- capture-noden prosessorikuorma (1 prosessoriydin): 39 – 41%.

Mittalaitteilta tallennettava datamäärä pysyy lähes samana riippumatta mittausten arvoista. Sähköasemalla tapahtuvat virhetilanteet lisäävät dataliikennettä, mutta kuitenkin niin vähän ettei se tässä mittakaavassa vaikuta suorituskykyyn. Järjestelmässä on käytössä 10Gt/s verkko, jonka ylittää näiden testien perusteella järjestelmän kapasiteettivaatimukset n. 10-kertaisesti. Prosessorin kuorma ei muutu datamäärän kasvaessa, koska suurin osa kuormasta kohdistuu verkkolaitteistoon.

Järjestelmän levykapasiteettivaatimukset testattiin tallentamalla yksi täysi minuutin pituinen mittaussarja josta pystytään laskemaan vuoden tallennusten kokonaiskapasiteetin tarve. Yksi minuutin sarja kaikilta mittalaitteilta tallennettua dataa kuluttaa 211Mt levykapasiteettia, josta pystytään laskemaan vuosittainen tallennuskapasiteetin tarve.

$$211Mt * 60 * 24 * 365 = 110Tb$$

Tallennettu raakadata pystytään deduplikoinnin lisäksi kuitenkin myös kompressoimaan. Kompressointimetodiksi valittiin tehokkuuden perusteella XZ Utils, joka mahdollistaa raa'an binääridatan 85% kompressiosuhteella. Kompressoinnin jälkeen vuosittainen kapasiteetin tarve laskee siis huomattavasti.

$$110Tb * 0{,}15 = 16{,}64Tb$$

# 8 YHTEENVETO

Tämän työn tuloksena saatiin tuotantokäyttöön soveltuva kahdennettu datan tallennusjärjestelmä, joka ei ota kantaa tallennettavan tiedon rakenteeseen. Järjestelmä otetaan käyttöön Sundom SmartGrid INKA -projektiin kesän 2015 aikana. Työn vaatimuksiin kuuluivat järjestelmän sekä ohjelmiston suunnittelu ja toteutus sekä yleinen tuotekehitys liittyen projektiin.

Kaikki työn määrittelyvaiheessa ehdotetut valmiit ratkaisut osoittautuivat ominaisuuksiltaan riittämättömiksi projektin vaatimuksiin, joten projektiin toteutettiin täysin uusi tallennusohjelmisto. Ohjelmiston tärkeimpiin vaatimuksiin kuuluvat kahdennettu datan tallentaminen, suodattaminen sekä mahdollisuus ladata tallenteita arkistosta rajapinnan kautta. Kaikki ohjelmiston vaatimukset saatiin toteutettua määräaikaan mennessä ja tärkeimmät ominaisuudet saatiin testattua.

Suurimmaksi haasteeksi ohjelmiston toteutuksessa osoittautui tallennusjärjestelmän kahdentaminen. Datan ollessa broadcast-liikennettä, ei mikään saatavilla oleva valmis tallennusohjelmisto pysty sen kahdennettuun tallennukseen. Tallennusohjelmiston kahdennus toteutettiin lopuksi tallentamalla kaikki data kahteen kertaan ja poistamalla duplikaatit. Tämän kaltainen ohjelmisto ei itsessään ole uusi asia, mutta duplikaattien poistaminen reaaliaikaisesti datavirrasta on suhteellisen ainutlaatuinen ratkaisu.

Järjestelmätasolla työn haasteiksi muodostui eri komponenttien suuri määrä. Järjestelmässä tarvittiin tässä työssä toteutetun ohjelmiston lisäksi hyvin paljon kolmannen osapuolen ohjelmistokokonaisuuksia, joiden käyttöönotto vaati suhteellisen suuren määrän perehtymistä. Näistä komponenteista suuri osa on kuitenkin pitkälle kehitettyjä, joten käyttöönottoprosessi saatiin automatisoitua hyvin.

Vaikka kaikki työn vaatimukset saatiin täytettyä, jäi tallennusjärjestelmä keskeneräiseksi projektitasolla. Järjestelmän vieminen tuotantoon vaatii vielä lisätestejä sekä ohjelmiston muokkaamista vastaamaan lopullisia vaatimuksia.

# LÄHTEET

/1/ TEKES. 2015. Verkkosivut. Viitattu 21.4.2015.
http://www.tekes.fi/ohjelmat-ja-palvelut/ohjelmat-ja-verkostot/inka/

/2/ Ter-Gazarian, A. 1994. Energy Storage And Power Systems, IEE Energy Series 6. The Institution of Engineering and Technology.

/3/ Arulampalam, A., Ekanayake, J.B ja Jenkins, N. 2003. Application study of a STAT-COM with energy storage. IEE Proceedings: Generation, Transmission and Distribution.

/4/ 2003. IEC61850 Standard Documentation Ed.2. International Electrotechnical Commission.

/5/ Debian.org. 2015. Verkkosivut. Viitattu 24.4.2015.
https://www.debian.org/intro/about

/6/ 2007. IBM Global Education, Virtualization in Education. The Greaves Group.

/7/ KVM. 2015. Verkkosivut. Viitattu 27.4.2015.
http://www.linux-kvm.org/

/8/LXC. 2015. Verkkosivut. Viitattu 27.4.2015.
https://linuxcontainers.org/

/9/ Elastic. 2015. Verkkosivut. Viitattu 5.5.2015.
https://www.elastic.co/products/elasticsearch

/10/ Elasticsearch APIs. 2015. Verkkosivut. Viitattu 5.5.2015.
http://www.elastic.co/guide/en/elasticsearch/reference/1.4/docs.html

/11/ Anvia. 2015. Verkkosivut. Viitattu 13.5.2015.
http://www.anvia.fi/anvia/tietoa-anviasta/perustietoa-anviasta/anvia-lyhyesti

/12/ Pyro4. 2015. Verkkosivut. Viitattu 19.5.2015.
https://pypi.python.org/pypi/Pyro4