SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Petri Koivulahti

Aikasarjadatan käsittely reunalaskennan avulla

Opinnäytetyö
Kevät 2023
Insinööri (ylempi AMK), Automaatiotekniikka

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Insinööri (ylempi AMK), Automaatiotekniikka

Tekijä: Petri Koivulahti

Työn nimi: Aikasarjadatan käsittely reunalaskennan avulla

Ohjaajat: Petteri Mäkelä (SeAMK), Juhamatti Kuusisaari (Wapice)

Vuosi: 2023 Sivumäärä: 83 Liitteiden lukumäärä: 2

Tämä työ on kehittämistyö, jonka tavoitteena on teollisen internetin järjestelmän suunnittelu. Järjestelmän tehtävä on kerätä paikallisesti aikasarjamuotoista tietoa eri lähteistä ja käsitellä kerättyä tietoa käyttäen reunalaskentaa. Työn taustalla on tarve tuottaa järjestelmätoteutus liikkuvien ajoneuvojen IoT-reunalaskentaan. Tässä käyttökohteessa laitteet toimivat useasti vaihtuvien verkkoyhteyksien alueella ja tiedonsiirtokaista ja saatavuus on ajoittain rajoittunut. Käsitelty data lähetetään laitteelta telemetriana pilvipalvelun käyttöön.

Reunalaskennan toteutusvaihtoehdoista tutkittiin stream processing -mallia ja aikasarjatiedon käsittelyn periaatteita. Kirjallisten lähteiden pohjalta tutustuttiin seuraavien työkalujen käyttämiseen reunalaskennassa: Microsoft Stream Processing for Edge, InfluxDB, Redis TimeSeries ja eKuiper.

Rajoitetun tiedonsiirtokaistan asettamat vaatimukset on huomioitu vertailemalla erilaisten tietomuotojen ja pakkaustyökalujen tehokkuutta. Vertailussa pyrittiin selvittämään, millaisessa muodossa aikasarjadata kannattaa tallentaa ja millaista työkalua käyttää tiedon pakkaamiseen. Pakkaustyökaluista haettiin kompromissia pakkaustehokkuuden ja pakkausnopeuden välillä.

Vertailu suoritettiin käyttämällä avointa liikkuvasta ajoneuvosta kerättyä tietoa. Lähdetieto sisälsi sijaintitiedon, kiihtyvyystiedot (lineaarikiihtyvyys ja kiertonopeus), sekä erilaisia OBD-mittausarvoja. Sijainti- ja kiihtyvyystietojen näytteenottotaajuus on 10-kertainen OBD-mittauksiin nähden. Lähdetieto muutettiin binääri-, BSON-, CBOR-, CSV-, JSON-, SQLite- ja XML-muotoiseksi telemetriamuodoksi. Lisäksi tutkittiin Apache Avro -tietomuodon käyttämistä. Tietomuodot pakattiin GZip-, BZip2-, Zstandard-, XZ-, LZ4- ja Brotli-työkaluilla käyttäen kahta esiasetusta: nopea pakkaus ja tehokas pakkaus. Pakatuista tietomuodoista mitattiin pakatun tiedoston koko ja pakkausnopeus. Mittaukset suoritettiin reunalaskentalaitteen referenssialustalla, joka oli Raspberry Pi 3B+.

Tutkimuksen lopputuloksena telemetriatieto kannattaa kirjoittaa Apache Avro -tietomuotoon tiedoston koon näkökulmasta. Pakkaustyökaluista paras kompromissi pakkaustehokkuuden ja nopeuden välillä on käyttää joko Gzip-, Zstd- tai Brotli-työkalua telemetrian pakkaukseen.

[1] Asiasanat: reunalaskenta, internet of things, aikasarjadata, telemetria, stream processing, tiedon pakkaus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Master‘s Degree Programme in Automation Engineering

Author: Petri Koivulahti

Title of thesis: Timeseries data processing with edge computing

Supervisors: Petteri Mäkelä (SeAMK), Juhamatti Kuusisaari (Wapice)

Year: 2023 Number of pages: 83 Number of appendices: 2

The thesis investigated technologies and practices to build an Industrial Internet system for moving vehicles. The system ingests locally produced time series data and processes the data with edge computing. Moving machines operate in environments where connectivity availability and bandwidth are sometimes limited. Therefore, the machine-produced telemetry will utilize the available bandwidth with high efficiency.

The stream processing principle is intended to be utilized in edge computing implementation. The time series data is filtered, normalized and aggregated locally before being transmitted to the cloud services. The following tools considered for this purpose were Microsoft Azure Stream Processing for Edge, InfluxDB, Redis TimeSeries, and eKuiper.

Telemetry data formats are ranked by the file size when a test data set is written to the tested telemetry formats. A test dataset is an open database consisting of sampled values from a moving vehicle. The test dataset values are location information, acceleration information, and OBD-values. The location and acceleration values are sampled ten times faster than OBD-values. The telemetry formats tested were Apache Avro, Binary, BSON, CBOR, CSV, JSON, SQLite and XML.

Telemetry data formats are compressed with a selection of compression tools. The tools were tested by using “fast” and “best” settings. The compressed file size and the compression speed were measured to determine the compression factor and compression rate. The compression tools tested were GZip, Bzip2, ZStandard, XZ, LZ4 and Brotli. The compression speed was measured on a reference edge computing platform that was a Raspberry Pi 3B+ device. A balance between the compression factor and compression speed was desired in this benchmark.

The best telemetry format by file size is Apache Avro format. A good bargain between the compression factor and compression speed is either GZip, ZStd or Brotli.

[1] Keywords: edge computing, internet of things, time series data, telemetry, stream processing, data compression

SISÄLTÖ

Kuvaluettelo

Kuvioluettelo

Taulukkoluettelo

Käytetyt termit ja lyhenteet

Aikasarjadata	Kolme komponenttia käsittävä tiedon tallennusmuoto. Komponentit ovat aikaleima, tiedon tunniste ja arvo.
Avain–arvopari	Tietorakenne joka koostuu kahdesta komponentista: tiedon avaimesta ja tiedon arvosta. Tiedon avain on parina myös uudelle avain–arvoparille.
Internet of Things	Asioiden tai esineiden internet. Verkottuneiden ja älykkäiden laitteiden ekosysteemi. Käytetään useasta sateenvarjoterminä asioiden internettiin liittyvistä hierarkioista ja teknologioista.
Kenttälaite	Fyysinen laite, joka suorittaa erityisesti sille määriteltyjä tehtäviä esimerkiksi ohjaa muita laitteita tai kerää tietoa reaalimaailmasta.
Ohjelmistokontti	Tapa paketoida ja jaella ohjelmia niin, että paketissa on mukana myös kaikki ohjelman tarvitsemat riippuvuudet.
Pilvipalvelu	Maailmanlaajuisessa mittakaavassa tarjottava tuote, jonka avulla voidaan rakentaa skaalautuvia digitaalisia palveluita. Palvelun hinnoittelu perustuu yleensä käytettyyn määrään.
Skeema	Tietomallin määrämuotoinen esitys.
Stream processing	Tiedon käsittely jatkuvana tietovirtana.
Telemetria	Laitteen ulos tuottama tietovirta. Esimerkiksi IoT-laitteen tuottama tietovirta pilvipalveluun.
Tietomalli	Kuvaus tiedon rakenteesta ja tiedon välisistä suhteista.
Tietomuoto	Miten tieto kirjoitetaan teksti- tai binäärimuodossa ottaen huomioon käytetty tietomalli.

1 JOHDANTO

1.1 Tausta

Internet of Things -laitteiden kanssa työskenneltäessä törmätään usein samaan ongelmaan: miten tietoa pystytään keräämään silloin, kun laite ei ole yhteydessä internetiin? Langattoman yhteyden ei voida olettaa toimivan koko ajan luotettavasti, ja yhteydet ovat monessa tapauksessa hitaita, alttiita häiriöille tai kalliita käyttää. Tästä syystä laitteen täytyy kyetä puskuroimaan tietoa laitteen flash-muistille ja purkamaan puskurin sisältö, kun laite on saanut muodostettua uuden yhteyden. Tiedonsiirron luotettavan toimivuuden varmistamiseksi, laitteen täytyy käsitellä tämän tyyppiset tilanteet niin, että tietoa ei katoa.

Usein tieto täytyy myös jalostaa lähdelaitteella sellaiseen muotoon, että säästetään tiedonsiirtokaistaa. Jalostettu muoto voi olla myös sellainen, miten se esitetään myös tiedon käyttäjälle. Laite tuottaa eri lähteistä kerättyä telemetriatietoa, jota lähetetään eteenpäin. Käytössä oleva tiedonsiirtokaista voidaan hyödyntää tehokkaasti, kun laitteen lähettämään tietomuotoon kiinnitetään huomiota. Tiedon pakkaaminen häviöttömällä pakkaustyökalulla on järkevää, kun halutaan hyödyntää käytössä oleva tiedonsiirtokaista vielä tehokkaammin.

Laite tai kone saattaa operoida tietoliikenneverkkojen peittoalueen ulkopuolella pitkiäkin aikoja, ja tiedonsiirtoyhteys voi olla vain lyhyen aikaan käytettävissä. Suomessa matkapuhelinverkkojen kuuluvuus kattaa vuonna 2022 melkeinpä kaikki asutut alueet ja sen ulkopuolella on lähinnä erämaa-alueita ja matkapuhelinverkon saatavuus on vain harvoin ongelma. Monissa muissa maissa matkapuhelinverkkojen saatavuus saattaa kuitenkin vaihdella suuresti. Joissain kehittyvissä maissa saattaa tiedonsiirtoverkkoja olla saatavilla vain suurimpien valtaväylien ja asutuskeskusten läheisyydessä.

Haastavissa käyttöympäristöissä voisi harkita sovellettavan LPWAN-verkkoa (Low-Power Wide-Area Network) tai satelliittipohjaista tiedonsiirtoa. LPWAN-verkot ovat tiedonsiirtoverkkoja, jotka tarjoavat hyvän kattavuuden, mutta vaatimattoman tiedonsiirtokaistan. Tästä syystä tieto pitäisi pakata mahdollisimman tehokkaasti ennen lähetystä. Tässä ratkaisussa koko käytössä oleva tiedonsiirtokaista olisi suositeltavaa käyttää hyötydatan siirtämiseen.

Laite tai kone saattaa operoida myös maanpinnan alapuolella, jolloin toimintaympäristöön on mahdollisesti rakennettu paikallinen tiedonsiirtoverkko. Verkon tukiasemat eivät välttämättä

kata koko toimintaympäristöä, ja tiedonsiirtoyhteys on saatavilla vain rajatuilla alueilla. Tästä syystä on tarpeen tallentaa kerätty tieto laitteelle paikallisesti, jotta sitä voidaan hyödyntää muissa paikallisissa palveluissa.

1.2 Tavoite

Tämä työ on kehittämistyö, ja sen tavoitteena on kehittää järjestelmä sulautetulla laitteella tapahtuvan aikasarjadatan tallentamiseen ja käsittelyyn paikallisesti. Tavoitteena on hyödyntää saatavilla olevia alustoja ja valmiita työkaluja järjestelmän rakentamiseen. Reunalaskennan lisäksi tavoitteena on selvittää, miten laitteelta lähetettävä telemetriatietoa kannattaa siirtää, että käytössä oleva tiedonsiirtokaista voitaisiin hyödyntää mahdollisimman tehokkaasti.

Tutkittavat työkalut ja menetelmät on rajattu sellaisiin työkaluihin, joita voidaan soveltaa Single Board Computer (SBC) -laitteisiin, kuten esimerkiksi Raspberry Pi. Kehitettävän ratkaisun täytyy siis olla käytettävissä laitteessa, joka koostuu mikroprosessorista sekä RAM- ja flash-muistista. Tyypillisesti nämä laitteet on koottu yhdelle piirilevylle ilman mahdollisuutta käyttäjän päivittää tai vaihtaa laitteen osia.

Työ ei ota kantaa käytössä olevaan tiedonsiirtoverkkoon tai -tapaan. Oletus on, että tietoa voidaan siirtää tavuista koostettuina binääripaketteina.

1.3 Rakenne

Luku 2 antaa lyhyen katsauksen teollisen internetin IoT-järjestelmään ja esittelee mahdollisia pilvipalvelualustoja sellaisen toteuttamiseen. Luvussa 3 esitellään reunalaskentaa ja yleisimpiä stream processing -menetelmiä. Stream processing -menetelmällä voidaan käsitellä sisään tulevaa tietoa tietovirtana. Luvussa 4 kartoitetaan datamalleja telemetriatiedon paketoimiseksi. Luvussa 5 tutkitaan erilaisten telemetriamuotojen eroavaisuuksia simuloidussa toimintaympäristössä. Luvussa 6 tarkastellaan simuloinnissa saatuja tuloksia ja pohditaan erilaisten mallien ja metodien hyviä ja huonoja puolia. Luku 7 sisältää yhteenvedon ja pohdintaa saaduista testaustuloksista ja ajatuksia teknologiapinon kehittymisestä tulevaisuudessa.

1.4 Tutkimusmenetelmät

Reunalaskentaan, stream processingiin ja telemetriatiedon paukkaamiseen perehdyttiin kirjallisuuden avulla. Kirjallisuuden avulla selvitettiin, miten reunalaskentajärjestelmät yleisesti rakentuvat ja miten järjestelmä suunnitellaan yhdessä pilvipalveluiden kanssa. Stream processingin käyttämisestä reunalaskentalaitteella selvitettiin järjestelmän toimintaperiaate ja tehtiin katsaus tarjolla oleviin valmiisiin ohjelmistoratkaisuihin. Selvitystuloksilla vahvistettiin valmiuksia lähteä suunnittelemaan ja toteuttamaan omaa reunalaskentajärjestelmää aikasarjadatan käsittelyyn.

Telemetriatiedon pakkauksesta suoritettiin vertailu eri tietomallien ja pakkaustapojen suorituskyvystä. Vertailulla selvitettiin tehokkain tapa pitää telemetriatiedon tiedostokoko mahdollisimman pienenä. Tiedostokoko mitattiin tallentamalla sama aikasarjamuotoinen tieto käyttäen vertailussa mukana olevia tietomuotoja. Pakkaamattomien tietomuotojen tulokset asetettiin vertailuun, jossa selvitettiin paras tapa kirjoittaa tietoa, jos tieto lähetetään pakkaamattomana. Vertailtavat tietomuodot pakattiin tämän jälkeen valituilla työkaluilla käyttäen kahta asetusta: nopea ja tehokas pakkaus. Pakatun tietomuodon tiedostokoko (tavuina) mitattiin, ja tuloksista saatiin paras pakkaustyökalu, joka tuottaa pienimmän tiedostokoon.

Pakkaustyökaluista mitattiin vielä tämän lisäksi pakkausnopeus. Pakkausnopeus määritettiin mittaamalla pakkaustyökalun suoritusaika eri tiedostomuodoilla ja pakkausasetuksilla. Lähdetiedoston (eli pakkaamattoman tiedon) tiedostokoko jaettiin ohjelman suoritusajalla, jolloin saatiin pakkausnopeus muodossa MB/s. Pakkausnopeus mitattiin Raspberry Pi 3B+ -laitteella, jota käytettiin tässä työssä referenssialustana reunalaskentalaitteesta. Pakattujen tiedostomuotojen keskimääräisestä pakkauskertoimesta ja keskimääräisestä pakkausnopeudesta muodostettiin kuvaaja, josta määritettiin paras kompromissi pakkausnopeuden ja -tehokkuuden välillä.

Kaikki vertailun testit ajettiin automatisoidusti Python-ohjelman avulla. Ohjelma tallensi lähdedatan eri tiedostomuodoissa käyttäen Pythonin standardikirjaston komponentteja ja kolmannen osapuolen kirjastoja. Pakkausohjelman suoritusaika mitattiin käyttämällä järjestelmän monotonista kelloa, joka mittasi suoritusajan nanosekunnin tarkkuudella. Kerätyt tiedostokoot ja pakkausajat tulostettiin tiedostoon, josta tulokset vietiin Google Sheets -palveluun. Sheets-palvelussa tiedot jäsenneltiin kuvaajiksi ja taulukoiksi, joihin lisättiin värikorostus tulkitsemisen helpottamiseksi.

1.5 Toimeksiantaja

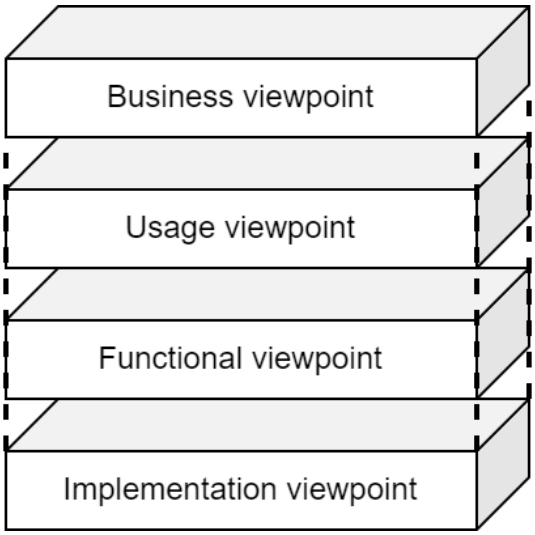
Työn toimeksiantaja on Wapice Oy, joka on digitaalisia palveluita ja ratkaisuja tarjoava suomalainen yritys. Yritys on perustettu vuonna 1999 Vaasassa, ja se on laajentunut voimakkaasti Suomessa monilla eri paikkakunnilla. Yrityksen pääkonttori sijaitsee Vaasassa. Wapice Oy tarjoaa palveluita monilla eri digitalisaation osa-alueilla, joista yksi on asioiden internet (Internet of Things, IoT). Yrityksen IoT-liiketoiminta on kasvanut voimakkaasti viime vuosina.

Wapice Oy tarjoaa ja kehittää omaa SaaS-pohjaista palvelua: IoT-Tickettiä. IoT-Ticketissä asiakas kytkee laitteensa palveluun käyttämällä tarjolla olevia ohjelmointirajapintoja. IoT-Ticket-palvelussa voidaan käsitellä laitteiden tuottamaa tietoa graafisilla ohjelmointityökaluilla. Tietoa voidaan esittää reaaliaikaisesti käyttäjän itse rakentamissa näkymissä, ja tiedolle voidaan tehdä pilvipohjaista analysointia ja muodostaa automaattisia raportteja. Järjestelmää voidaan käyttää myös osana isompaa pilvipalvelujärjestelmää, jolloin IoT-Ticketillä kerätty tieto voidaan kytkeä suoraan asiakkaan muihin käytössä oleviin järjestelmiin.

2 TEOLLISEN INTERNETIN ARKKITEHTUURI

Teollisen internetin järjestelmäarkkitehtuurin määrittäminen on iso tehtävä ja sen vaikutukset näkyvät järjestelmän jokaisella tasolla (Lea, 2020, s. 41). Tämän vuoksi sen määrittämiseen on syytä käyttää aikaa ja harkintaa, jolloin päästään lopputulokseen, joka palvelee järjestelmän käyttäjien tarpeita ja kattaa myös tulevat tarpeet. Kokonaisvaltainen IIoT-järjestelmäarkkitehtuuri tarvitsee asiantuntijoita pilvipalveluiden, tietoliikenteen, sulautettujen järjestelmien, ohjelmistotuotannon, tietokantojen, data-analytiikan ja web-sovellusten osa-alueilta. Lisäksi tarvitaan myös muiden ammattialojen asiantuntijoita.

IIoT-järjestelmää voidaan tarkastella monesta eri näkökulmasta, jotka asettavat omat tarpeet ja vaatimukset järjestelmälle. Industrial Internet Consortium (IIC) on määritellyt referenssiarkkitehtuurin teollisen internetin järjestelmälle, ja siinä on määritelty 4 eri näkökulmaa teollisen internetin järjestelmälle (Lin, ym., 2019, s. 14). Kuvio 1 esittelee nämä neljä näkökulmakerrosta. Ylempien kerrosten on tarkoitus ohjata alempien kerrosten suunnittelua. Alempien kerroksien suunnittelulla voidaan toteuttaa ylempien kerrosten vaatimukset ja tarvittaessa uudelleen arvioida niiden tarpeellisuutta.



Kuvio 1. Neljä näkökulmaa teollisen internetin järjestelmään (Lin, ym., 2019, s. 14).

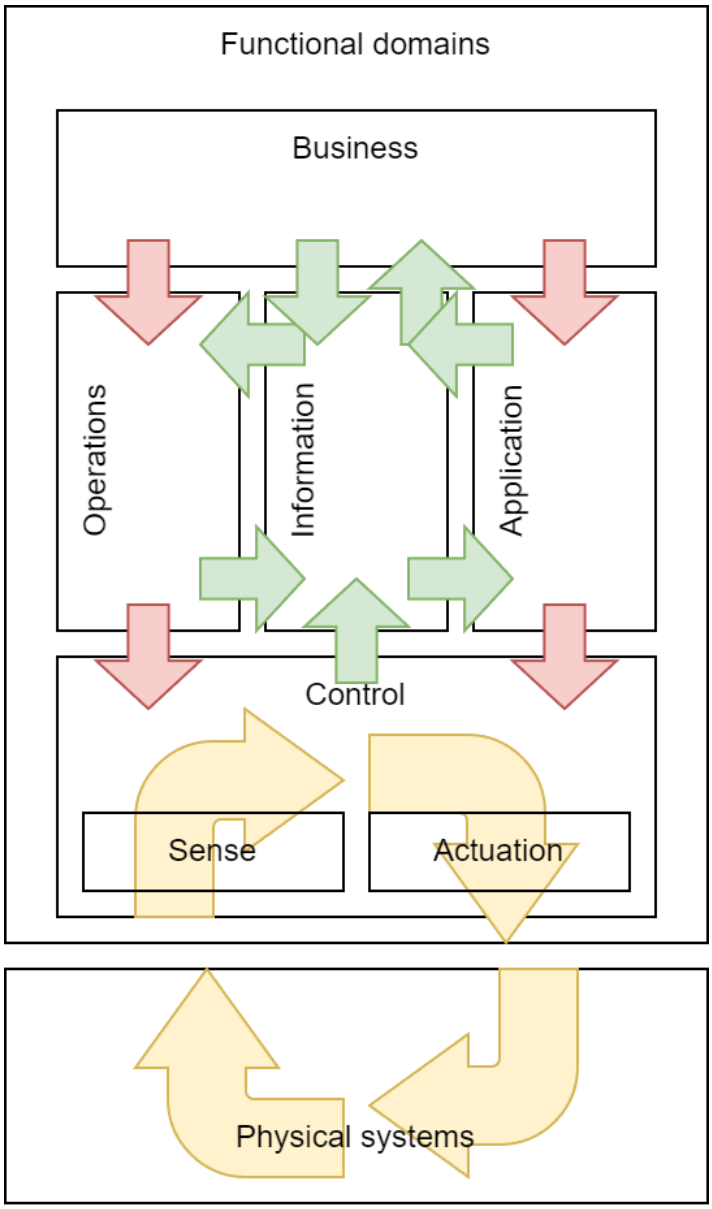
Ensimmäinen näkökulma on liiketoiminnan näkökulma. Liiketoiminnan näkökulma tuo järjestelmään vision ja kertoo, millä tavalla järjestelmä muutetaan lisäarvoksi yhtiön liiketoiminnassa (Lin, ym., 2019, s. 19). Järjestelmän tuomat velvoitteet yritykselle ja lainsäädännölliset rajoitteet ovat kiinnostavia tästä näkökulmasta. Vision lisäksi voidaan määritellä myös

mitattavissa olevat keskeiset tavoitteet ja perustoiminnallisuudet, jotka järjestelmän täytyy täyttää, että järjestelmä on liiketaloudellisesti kannattava investointi.

Käytön näkökulma tarkastelee järjestelmää käyttäjien ja tehtävien näkökulmasta. Käyttötapaukset jaetaan tehtäviin, jotka määrittelevät käyttäjän roolin järjestelmässä, sekä toiminnallisuuksiin ja toteutettaviin komponentteihin, jotka tehtävän suorittaminen vaatii (Lin, ym., 2019, s. 22). Esimerkkejä tehtävistä ovat uuden laitteen kytkeminen järjestelmään tai viikkoraportin hakeminen. Tehtävät voidaan koota vielä toiminnoiksi, jotka määrittelevät mitkä tehtävät täytyy suorittaa ennalta määritellyn lopputuloksen saavuttamiseksi. Käyttötapauskuvauksia voidaan käyttää apuna, kun määritellään järjestelmän toimintoja.

Toiminallinen näkökulma voidaan jakaa viiteen alueeseen (domain) (Lin, ym., 2019, s. 26). Nämä alueet ovat: hallinta (control), toiminto (operation), tieto (infomation), sovellus (application) ja liiketoiminta (business). Kuvio 2 esittää näiden alueiden väliset vuorovaikutukset. Kuvassa tieto liikkuu vihreiden nuolien osoittamaan suuntaan ja komennot punaisten nuolten osoittamaan suuntaan. Järjestelmän toiminnalliset vaatimukset määritellään osa-aluekohtaisesti.

Fyysiset laitteet tuottavat tietoa, jota voidaan ottaa vastaan (sense) (Lin, ym., 2019, s. 26). Vastaavasti fyysisille laitteille voidaan syöttää ohjauskäskyjä (actuation). Tiedon varastointi on järjestelmän ytimessä, josta sitä tarjotaan toiminnoille, sovelluksille ja liiketoimintalogiikalle. Toiminnot ovat kokoelma järjestelmään liittyviä tehtäviä, jotka voidaan pyynnöstä aktivoida. Sovellukset tarjoavat palveluita ja käyttöliittymiä järjestelmän käyttäjille tiedon tarkasteluun ja hallintaan. Liiketoimintalogiikka jalostaa operaatioiden ja sovellusten tuottaman tiedon, jota voidaan hyödyntää yrityksen prosessien tarkastelussa. Esimerkki tällaisesta tiedosta voisi olla tilannekuva tuotantolaitoksen eri osien käyttöasteista.



Kuvio 2. Toiminnallisten alueiden vuorovaikutus (Lin, ym., 2019, s. 26).

Lisäksi toiminnallisten alueiden yhteydessä on huomioitava järjestelmän luonteeseen liittyviä vaatimuksia, kuten turvallisuus, tietoturvallisuus, vikasietoisuus, luotettavuus, yksityisyys ja skaalautuvuus (Lin, ym., 2019, s. 35). Näitä vaatimuksia kutsutaan myös ei-toiminnallisiksi vaatimuksiksi.

Toteutuksen näkökulma määrittelee käytettävät teknologiat, ohjelmistokomponentit ja järjestelmän toteutustavan (Lin, ym., 2019, s. 39). Toteutus määrittelee arkkitehtuurin, miten valitut ohjelmistot linkittyvät toisiinsa ja millaisia rajapintoja niiden välillä on. Teknologiavalinnat määrittelevät, millaisia laitteita järjestelmään tarvitaan ja millaisia tiedonsiirtoratkaisuja eri

järjestelmän tasojen välillä tarvitaan. Toteutusmalli vastaa myös järjestelmän luonteelle asetettuihin vaatimuksiin, kuten tietoturvan huomioimiseen kokonaisarkkitehtuurissa.

Teollisen internetin järjestelmän rakentaminen on iso projekti, ja monet toteutukset käyttävät jotain pilvipalveluun toteutettua IoT-ratkaisua (Collin & Saarelainen, 2016, s. 228). Esimerkiksi Microsoft Azure tai Amazon Web Services pilvipalveluissa on valmiina IoT-sovellusten tekemistä tukevia palveluita. Pilvipalveluiden tarjoamat palvelut voidaan luokitella IaaS (Infrastructure as a service), PaaS (Platform as a service) tai SaaS (Software as a service) -palveluiksi. Monet Microsoft Azuren palveluista ovat esimerkiksi PaaS-palveluita. Markkinoilla on myös useita SaaS-tyyppisiä IoT-palveluita, jotka tarjoavat valmiin ohjelmiston asiakkaan käyttöön.

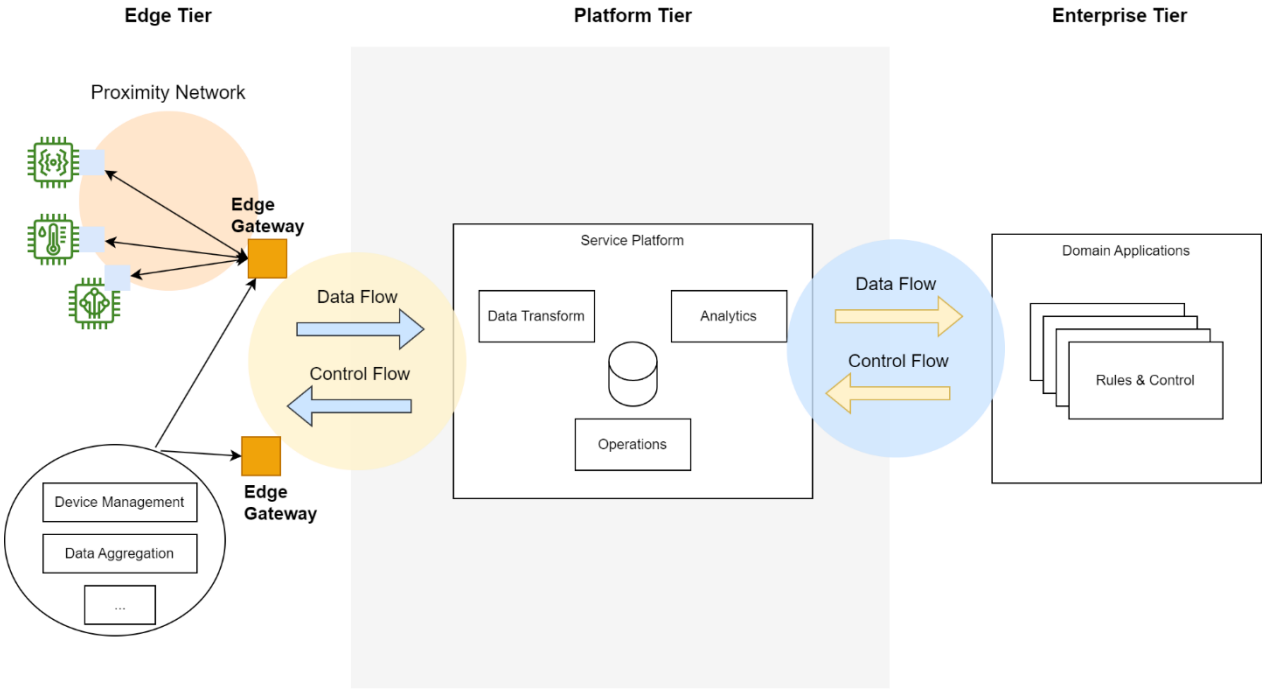
Hyvä IoT-alusta täyttää sille asetetut keskeiset vaatimukset (Collin & Saarelainen, 2016, s. 230). Laitteiden liitettävyys ja tiedon harmonisointi ovat ensimmäinen vaatimus. Laitteet kytkeytyvät palveluun samankaltaisella tavalla riippumatta yhteyden teknisestä ratkaisusta. Hyvä IoT-alusta sisältää myös toiminnot laitteiden hallintaan. Palvelun tulisi mahdollista laitteiden etähallinta ja etäpäivitykset.

Tiedon tallentaminen on alustan keskeinen tehtävä, ja tietokannan täytyy soveltua suurien määrien tallentamiseen (Collin & Saarelainen, 2016, s. 231). Tieto itsessään voi olla hyvin erityyppistä ja puutteellista, joten tietoa täytyy pystyä käsittelemään myös lähes reaaliaikaisesti. Pilvipalvelussa tapahtuva tiedon analysointi on myös järjestelmän keskeinen tehtävä. Sääntöperusteisella ja automatisoidulla tiedon käsittelyllä ja prosessoinnilla tuotetaan jalostettua dataa järjestelmän muille palveluille.

Käsitellyn tiedon visualisointi ja raportointi ovat loppukäyttäjän näkökulmasta tärkeitä ominaisuuksia IoT-järjestelmässä (Collin & Saarelainen, 2016, s. 232). Tiedon hallinta- ja kehitystyökalujen halutaan olevan helposti lähestyttäviä ja monipuolisia. Järjestelmä voi tarjota valmiita työkaluja esimerkiksi datan esittämiseen web-sivulla tai mobiilisovelluksessa. Ulkoiset rajapinnat, jotka mahdollistavat järjestelmän integroinnin muihin järjestelmiin, ovat myös keskeinen vaatimus alustan toiminnalle.

2.1 Kolmitasoinen IIoT-arkkitehtuuri

Lin ym. (2019, s. 40) esittelevät kolmitasoisen mallin yhtenä mahdollisena tapana toteuttaa teollisen internetin järjestelmä. Siinä kokonaisuus on jaettu kolmeen osa-alueeseen: reuna (edge), alusta (platform) ja enterprise (kuvio 3). Osa-alueet on sijoitettu vierekkäin, niin että reuna on vasemmalla, alusta keskellä ja enterprise-taso oikealla. Tieto virtaa vasemmalta oikealle ja ohjauskomennot oikealta vasemmalle.

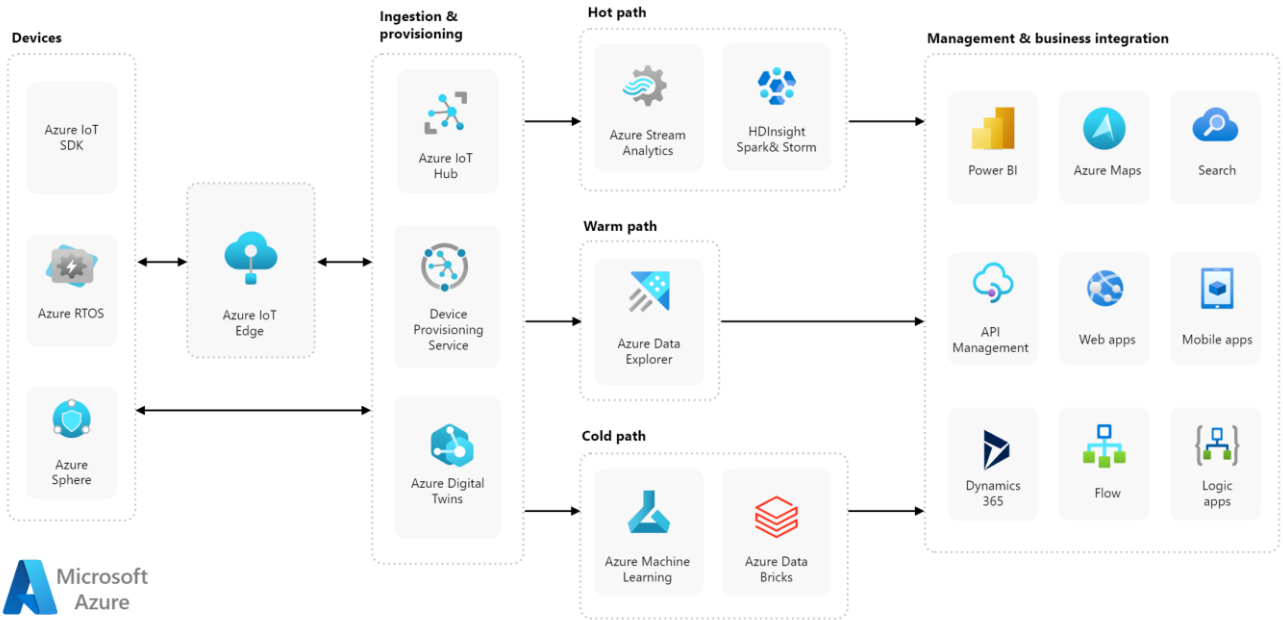


Kuvio 3. Kolmitasoinen teollisen internetin arkkitehtuuri (Lin, ym., 2019, s. 41).

Reunatasolla toimivat kenttälaitteet, paikalliset verkot ja verkkoyhdyskäytävät (Lin, ym., 2019, s. 41). Reunataso tuottaa tietoa alustalle, joka voi vastaavasti antaa toimintakäskyjä kenttälaitteille. Alustatasolla tietoa käsitellään, analysoidaan ja tehdään kenttälaitteiden hallintaa. Alusta muuttaa enterprise-tasolta tulevat ohjauskäskyt niin, että ne ohjautuvat oikeille laitteille. Alustan tehtävä on myös tuottaa jalostettua tietoa enterprise-tasolle. Loppukäyttäjän sovellukset toimivat enterprise-tasolla, missä myös hallitaan laitteiden ohjauksesta vastaavaa toimintalogiikkaa.

2.2 Microsoft Azure

Microsoftin tarjoama IoT-järjestelmän referenssiarkkitehtuuri noudattelee vahvasti kolmitasoista arkkitehtuurimallia (Microsoft, i. a.). Kuvio 4 esittelee erilaisia Azuren palveluita, joita voidaan käyttää järjestelmän rakentamiseen.



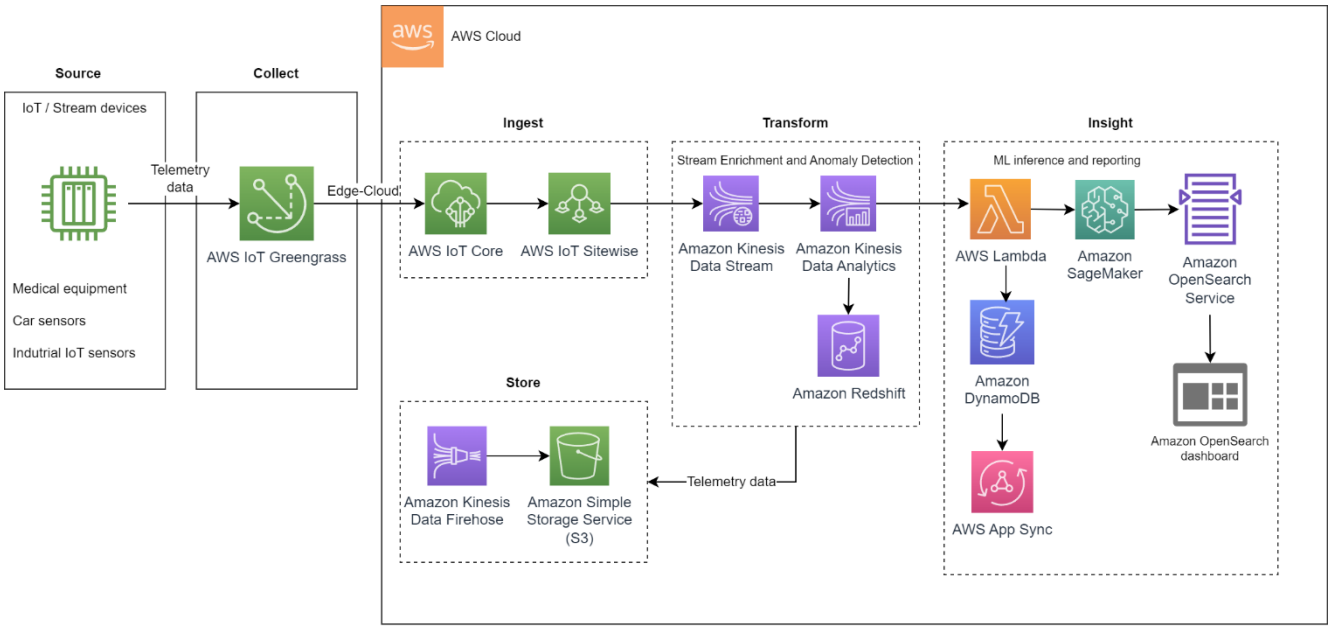
Kuvio 4. Microsoft Azure IoT-järjestelmän referenssiarkkitehtuuri (Microsoft, i. a.).

Kenttälaitteissa voidaan käyttää joitain Microsoftin tarjoamia työkaluja, kuten Azure IoT SDK -kirjastoja, Azure RTOS -käyttöjärjestelmää tai Azure IoT Edge -järjestelmää (Microsoft, i. a.). Microsoft Azure on kuitenkin pääasiassa pilvipalvelu, joten tarjolla olevien palveluiden painopiste on enemmän pilvipalvelussa tapahtuvassa tiedonkäsittelyssä. Azure IoT Hub on solmupiste kenttälaitteiden yhteyksille muihin Azuren palveluihin. Tässä mallissa tietovirrat on jaettu kolmeen erityyppiseen kanavaan eli hot, warm ja cold. Hot-polussa tieto käsitellään hyvin nopeasti Azure Stream Analytics -palvelussa ja sillä voidaan reagoida nopeasti datassa esiintyviin muutoksiin. Warm-polku on käytännössä kanava, missä tietoa voidaan seurata kehittäjän toimesta ilman, että sille tehdään mitään toimenpiteitä. Cold-polku tarkoittaa tietoa, jota kerätään mahdollisesti suuria määriä, mutta sille ei tehdä nopeaa tiedon käsittelyä. Tämäntyyppistä tietoa voidaan hyödyntää esimerkiksi koneoppimisen opetustietona tai käsitellä Azure Data Bricks -palvelussa.

Enterprise-tasolle siirryttäessä tarjolla on erilaisia työkaluja tiedon esittämiseen loppukäyttäjille (Microsoft, i. a.). Power BI antaa työkalut automaattisten raporttien luomiseen järjestelmän tuottamasta tiedosta. Pilvipalvelussa voidaan ylläpitää web-palveluita ja mobiilisovelluksien taustaohjelmia, joiden kautta tietoa voidaan viedä loppukäyttäjien saataville. Pilvipalvelussa on myös mahdollista ylläpitää esimääriteltyjä tapahtumaketjuja (flow ja logic apps), jotka voidaan käynnistää dataohjautuvalla säännöillä.

2.3 Amazon Web Services (AWS)

Amazon tarjoaa useita erilaisia malleja arkkitehtuurin rakentamiseksi erilaisissa käyttötapauksissa. Kuviossa Kuvio 5 on esitetty arkkitehtuuri, joka on tarkoitettu aikasarjadatan käsittelyyn pilvipalvelussa. Tämä arkkitehtuuri noudattelee myös tässä tapauksessa kolmitasoista teollisen internetin mallia.



Kuvio 5. AWS-pilvipalvelun arkkitehtuuri tapahtumapohjaisen IoT-datan käsittelyyn. (Soveltaen Sodabathina ym., 2022).

Amazon Web Services tarjoaa monia komponentteja ja palveluita, joiden avulla voidaan rakentaa kattava IoT-järjestelmä (Sodabathina ym., 2022). IoT-laitteisiin voidaan asentaa Amazonin FreeRTOS-pohjainen käyttöjärjestelmä tai käyttää AWS:n tarjoamia valmiita kirjastoja, joilla laitteen saa kytkettyä AWS IoT Core -palveluun. AWS IoT Greengrass on reunalaskenta-alusta, jossa voidaan ajaa ohjelmia lamba-funktioiden muodossa. AWS IoT core on

yhdistämispiste, jossa laitteiden tuottama tieto saadaan sisään AWS:n muille pilvipalvelussa toimiville palveluille.

AWS:n pilvipalvelun käytössä olevista palveluista keskeisimmät ovat tiedon analysointiin tarkoitetut työkalut ja tiedon tallentamiseen soveltuvat palvelut (Sodabathina ym., 2022). Tietoa voidaan käsitellä Amazon Kinesis -palvelussa lähes reaaliaikaisesti. Kehittäjä voi luoda myös omia lambda-funktioita, jotka ovat serverless-tyyppisiä skaalautuvia sovelluksia. Tiedon tallentamiseen voidaan käyttää Amazon Timestream -tietokantaa, joka on optimoitu aikasarjanmuotoisen tiedon tallentamiseen. Muita tiedon tallennusratkaisuja ovat Amazon S3, joka soveltuu suurten tietomäärien tallentamiseen, sekä NoSQL-tietokanta DynamoDB. Amazon Sagemaker on koneoppimisjärjestelmä ja Amazon OpenSearch on dashboardeihin perustuva visualisointipalvelu. AWS App sync mahdollistaa API-kutsujen tekemisen järjestelmän tuottamaan tietoon.

2.4 Yksityinen pilvipalvelu ja hybridi

Yritys voi rakentaa myös yksityisen pilvipalveluratkaisun (Lea, 2020, s. 435). Siinä IT-infrastruktuurin resursseja, kuten palvelinkapasiteettia, ei jaeta muiden pilvipalvelun tarjoajan asiakkaiden kanssa. Yritys saa kaikki resurssit yksinomaan omaan käyttöönsä. Yksityinen pilvipalvelu voi olla myös yrityksen itse ylläpitämä ja sijaita omissa konesaleissa. Hybridimallissa yritys käyttää yhdistelmää, jossa osa toiminnoista sijaitsee ulkoisen toimijan pilvipalvelussa ja osa omassa, yksityisessä pilvipalvelussa. Palveluita voidaan sijoittaa yksityiseen pilvipalveluun, jos niiden toimintaan liittyy sensitiivisen tiedon käsittelyä ja tietoturvauhkia (mts. 436). Julkista pilvipalvelua voidaan käyttää myös järjestelmän skaalaukseen tilanteissa, joissa yrityksen oma pilvipalvelu ei pysty tarjoamaan riittävästi resursseja esimerkiksi hetkellisessä kuormituspiikissä. Järjestelmät voidaan rakentaa myös siten, että käyttöliittymä toimii julkisessa pilvipalvelussa ja taustasovellukset omassa yksityisessä pilvipalvelussa.

Suosittu yksityinen pilvipalvelualusta on OpenStack (Silverman & Solberg, 2018, s. 8). Se koostuu avoimen lähdekoodin ohjelmistoista, jotka tarjoavat vastaavia palveluita kuin Microsoft Azure ja Amazon Web Services. Pilvipalvelun keskeisiä rakennuspalikoita ovat virtuaalitietokoneet, tietoliikenneverkkojen hallinta, kuormantasaus, objektitietokanta ja näiden palveluiden hallinta eli orkestrointi. OpenStack tarjoaa joistain palveluista rajapinnan, joka on yhteensopiva AWS:n vastaavan palvelun rajapinnan kanssa. Tällainen palvelu on esimerkiksi

Nova, joka tarjoaa virtuaalikoneympäristön, jonka hallintarajapinta on yhteensopiva AWS:n EC2-palvelun kanssa.

OpenStack tarjoaa infrastruktuurin, minkä päälle voidaan rakentaa oman IoT-järjestelmän sovellukset (Lea, 2020, s. 436). Koska OpenStack on alusta, siinä ei ole valmiina IoT-ympäristössä tarvittavia palveluita, toisin kuin Microsoft Azuressa tai AWS:ssä. Tarvittavat palvelut täytyy valita ja ottaa käyttöön luomalla niistä virtuaalisia palvelinkoneita, joissa tarvittavia sovelluksia ajetaan. Käytettävät sovellukset voidaan valita melko vapaasti, ja mahdollisuus on käyttää esimerkiksi yhdistelmää avoimen lähdekoodin tuotteita sekä kaupallisia palveluita.

3 REUNALASKENTA JA STREAM PROCESSING

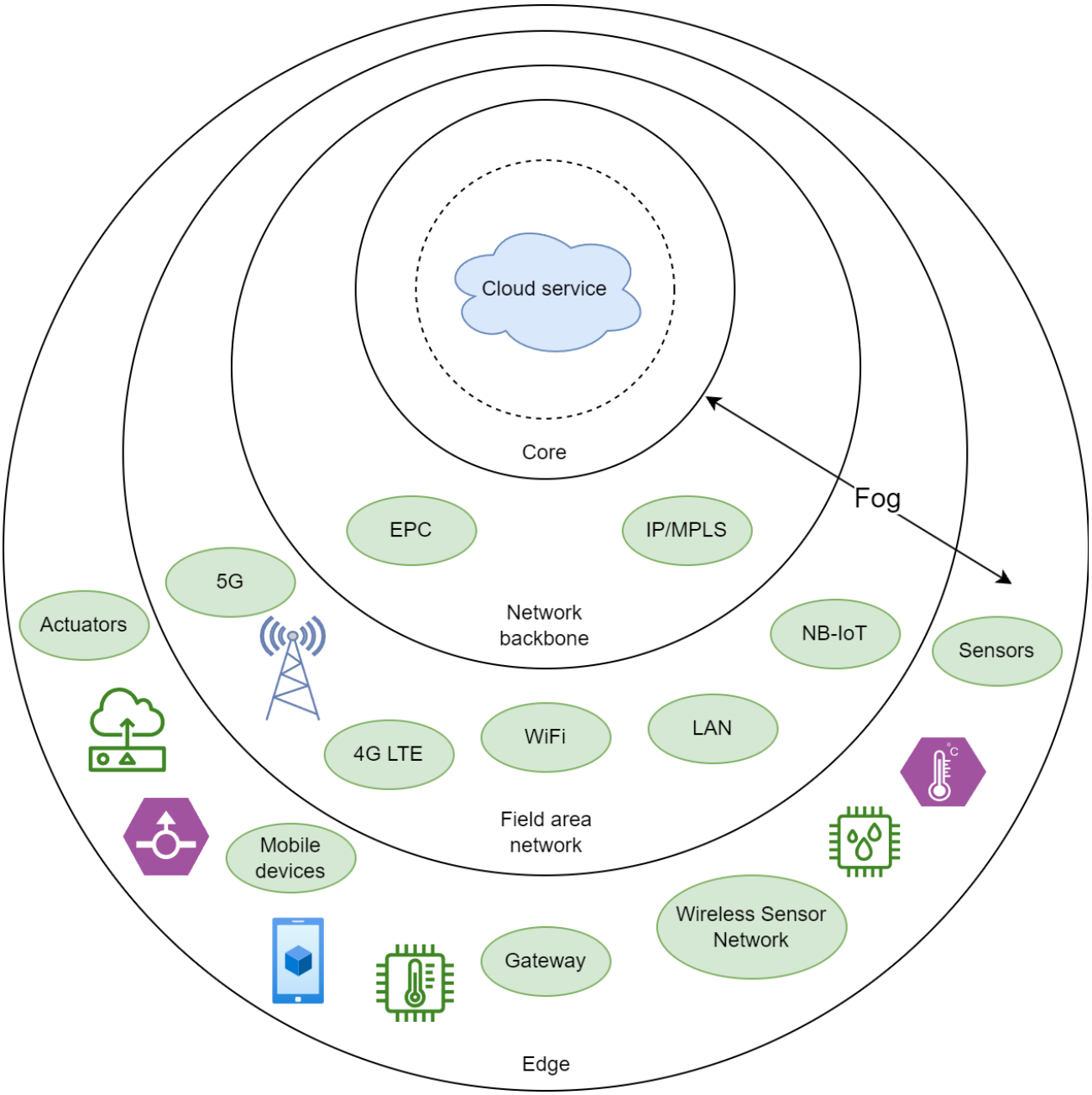
Tässä luvussa käsitellään reunalaskentaa osana IoT-järjestelmää. Reunalaskenta voi tarkoittaa monia eri asioita, ja tässä osassa keskitytään pääpainona aikasarjadatan käsittelyyn reunalaskennassa.

IoT-järjestelmiä voidaan toteuttaa monilla erilaisilla rakenteilla. Teollisessa internetissä käytetään useasti kolmiosaista ratkaisua järjestelmäarkkitehtuurissa (Gilchrist, 2016, s. 76). Ensimmäisessä osassa on järjestelmän reuna eli edge, missä varsinainen tiedon tuottaminen ja komentojen toimeenpano tapahtuu. Reuna voi koostua suuresta määrästä erilaisia kenttälaitteita, sensoreita ja paikallisia sensoriverkkoja. Laitteet eivät välttämättä käytä IP-pohjaista protokollaa kommunikointiin, jolloin tarvitaan yhdyskäytävä (gateway), joka yhdistää laitteet internetverkkoon.

Toisessa osassa järjestelmäarkkitehtuuria toimii alusta eli platform (Gilchrist, 2016, s. 76). Alusta tarjoaa ympäristön ja työkalut datan analysointiin, käsittelyyn ja varastointiin. Alusta voi toimia pilvipalvelussa, tai se voidaan yhdistää edge-laitteeseen, joka toimii myös samalla yhdyskäytävänä. Tällöin puhutaan sumulaskennasta (sumu on pilven alapuolella), mutta hyvin yleisesti puhutaan kuitenkin vain reunalaskennasta (Bonomi ym., 2012, s. 13).

Kolmas osa on enterprise-taso (Gilchrist, 2016, s. 76). Tässä osassa tapahtuu tiedon hyödyntäminen ja muuttaminen liiketoiminnaksi. Tieto esitetään loppukäyttäjille sopivilla käyttöliittymillä, jotka voivat tuottaa myös ohjauskäskyjä ensimmäisen tason laitteille. Järjestelmä toimii kaksisuuntaisesti, jossa tietoa virtaa toiseen suuntaan ja komennot taas vastakkaiseen suuntaan.

Tasojen lisäksi Internet of Things -järjestelmän rakennetta voidaan tarkastella myös hierarkiana, minkä ytimessä on pilvipalveluita tarjoava alusta. Kuvio 6 havainnollistaa, miten laitteet sijoittuvat eri tasoille, joka voidaan mallintaa pilvipalveluiden ympärille. Ytimen ulkopuolella on yleensä verkkoinfrastruktuurin ydinosat, kuten IP/MPLS, jossa operaattori tarjoaa yritykselle OSI-mallin tason 2 mukaisen salatun yhteyden usean maantieteellisesti hajautetun verkkosolmupisteen välille (Bonomi ym., 2012, s. 14). EPC on 4G LTE -arkkitehtuurissa taustajärjestelmä, joka hallinnoi puhe- ja tiedonsiirtopalveluiden tuottamista mobiiliverkon tukiasemassa (Lea, 2020, s. 259).



Kuvio 6. Internet of Things -järjestelmä laitehierarkia. (Soveltaen Bonomi ym., 2012, s. 14 ja Lea, 2020, s. 35).

Ennen reunatasoa on yleensä mobiiliverkkoja, kuten 4G LTE, 5G, NB-IoT ja Wi-Fi, tai fyysisiä yhteyksiä kenttälaitteisiin (Lea, 2020, s. 41). Hierarkian uloimmalla kerroksella toimivat varsinaiset reunalaitteet. Reunalaitteita voi olla hyvinkin paljon ja monen tyyppisiä. Sensori ja toimilaite voivat olla suoraan yhteydessä IP-verkkoon tai ne voivat käyttää protokollaa, joka vaatii toisen laitteen toimimaan protokollamuuntimena. Laite voi toimia myös osana sensoriverkostoa, joka on paikallinen lyhyen kantaman verkko, jossa laiteet voivat

kommunikoida toistensa välityksellä (Gilchrist, 2016, s. 89). Sensorinverkoissa on yleensä oma reunalaite, joka yhdistää paikallisen verkon IoT-järjestelmään.

Reunalaskennassa puhutaan useasti myös sumulaskennasta (fog computing) (Lea, 2020, s. 448). Sumulaskennassa on kyse pilvialustalla toimivien palveluiden tuomisesta lähemmäksi loppukäyttäjiä. Edge-laite saattaa olla osa sumulaskentaa, mutta tyypillisesti edge-laite ei tarjoa pilvialustan palveluita, vaan käyttää niitä itse. Sumulaskentaa voi kuitenkin esiintyä laitehierarkian muilla tasoilla.

Laitehierarkian alimmilla kerroksilla saattaa esiintyä vielä jossain tapauksissa yksi kerros eli ”usva” (mist) (Lea, 2020, s. 448). Yleensä tämän kerroksen laitteet ovat melko tehottomia ja toimivat yhdyspisteenä varsinaisiin kenttälaitteisiin ja sensoreihin, jotka eivät pysty yhdistymään suoraan edge-laitteeseen. Yksi esimerkki voisi olla langattoman sensoriverkon (Wireless Sensor Network, WSN) solmupiste, joka välittää sensoriverkon lähettämiä tietoja eteenpäin laitehierarkiassa.

Reunalaskentaa käytetään usein, kun halutaan lyhentää vasteaikaa, säästää tiedonsiirtokaistaa, varmistaa palveluiden saatavuus tai parantaa tietoturvaa ja yksityisyyttä (Lea, 2020, s. 313). Kun palvelut tuodaan lähemmäksi käyttäjää, niiden vasteaika tyypillisesti paranee, koska pyyntöjen ei tarvitse kulkea internetverkossa niin monen solmupisteen kautta. Tiedonsiirtokaistaa voidaan säästää tekemällä datalle esikäsittelyä edge-laitteella tai data voidaan yhdistää muihin paikallisten laitteiden tuottamaan dataan. Tiedon tallentaminen voidaan varmistaa edge-laitteen avulla tilanteessa, jossa internet-yhteys ei ole saatavilla. Edge-laite voi puskuroida sisään tulevaa tietoa omaan muistiinsa ja lähettää tiedon eteenpäin, kun yhteys on saatu palautettua. Edge-laite toimii myös tietoturvakerroksena kenttälaitteille. Kun kenttälaitteet muodostavat yhteyden edge-laitteen välityksellä, ne eivät ole suoraan kytkettynä internet-verkkoon. Joissain tapauksissa tietoon voi liittyä yksityisyysvaatimuksia, jolloin tietoa ei välttämättä voida siirtää pois paikallisilta laitteilta.

3.1 Verkkoyhteydet

Edge-laitteet voidaan kytkeä internet-verkkoon monella eri tavalla, käyttäen paikallisverkkoja tai laajan alueen verkkoja (Lea, 2020, s. 87). Yhteystavat voidaan luokitella IP-pohjaisiin, ei-IP-pohjaisiin sekä langallisiin ja langattomiin tekniikoihin. Langattomista laajan alueen yhteyksistä matkapuhelinverkko on suosituin valinta (Marshall, 2021, s. 42). Se tarjoaa yleisesti

melko hyvän maantieteellisen kattavuuden, hyvät siirtonopeudet ja se toimii myös liikkuvissa kohteissa. Joitain vuosia sitten 4G LTE -tekniikka toi mukanaan myös useita erityisesti IoT-käyttöön suunniteltuja standardeja. LTE-M ja NB-IoT ovat erityisesti vähävirtaisiin sovelluksiin tarkoitettuja tekniikoita, jotka tarjoavat paremman kattavuuden, mutta hitaammat tiedonsiirtonopeudet kuin matkapuhelimiin suunnatut 4G LTE -yhteydet (Lea, 2020, s. 307). Näiden yhteyksien saatavuus vaihtelee operaattoreiden välillä. Edge-laitteessa voidaan käyttää myös tavallista 4G LTE-yhteyttä, jos virrankulutus tai käyttökustannukset eivät ole este tai tiedonsiirtotarve on suuri.

5G-teknologia on itsessään myös reunalaskennan käyttötapaus (Marshall, 2021, s. 42). Virtualisoidussa verkkoympäristössä tukiasemille voidaan sijoittaa monenlaista ohjelmistoa. Tukiasemien yhteydessä on pieniä datakeskuksia, missä voidaan ajaa 5G-verkkosovellusten lisäksi myös esimerkiksi pilvialustan palveluita (mts. 43). 5G mahdollistaa myös verkkojen viipaloinnin (slicing), jolloin voidaan perustaa yksityisiä tiettyyn käyttötarkoitukseen tehtyjä verkkoja (Lea, 2020, s. 291). IoT-laitteille voidaan esimerkiksi lohkaista oma verkko, jonka toimivuuteen ei vaikuta muiden paikallisten käyttäjien kuormitus.

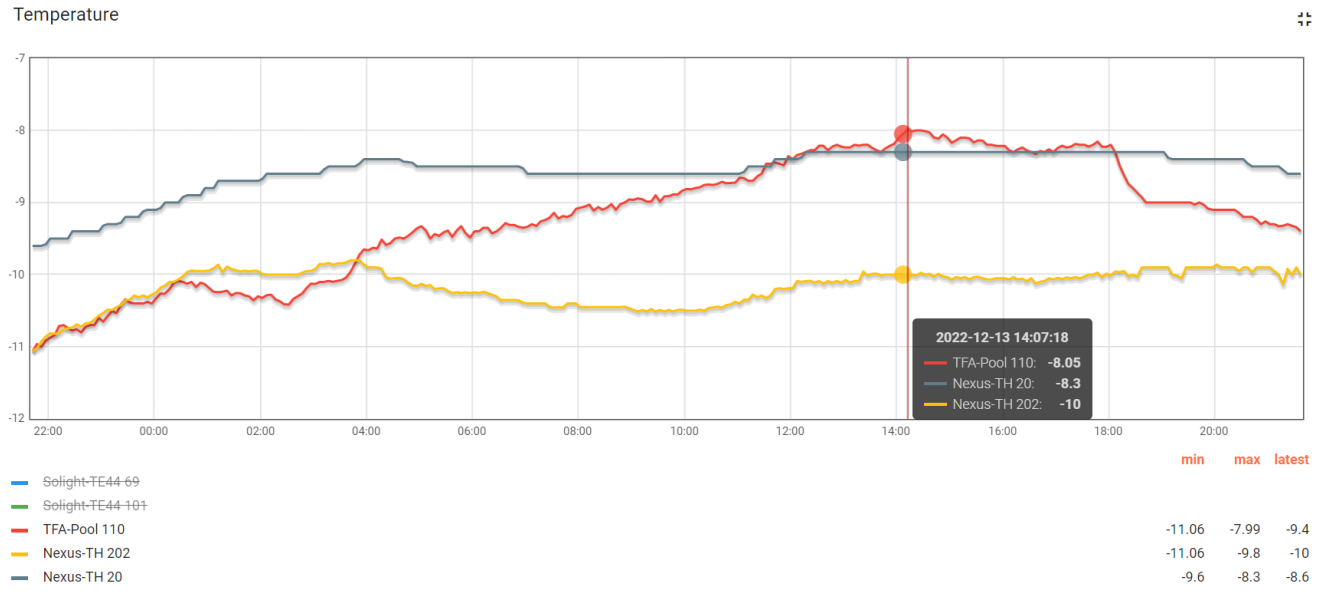
Laitteet tarvitsevat myös yhteisen protokollan, eli tavan miten laitteet kommunikoivat keskenään. Yleisesti käytetään protokollia, jotka toimivat joko pyyntö–vastaus-periaatteella tai julkaisija–tilaaja-mallisesti (Gilchrist, 2016, s. 134). Ensimmäisessä mallissa toinen järjestelmä toimii palvelimena ja toinen asiakkaana. Asiakas lähettää pyynnön palvelimelle ja palvelin palauttaa vastauksen pyyntöön. Tietoliikenne on kaksisuuntaista, mutta asiakaan täytyy tehdä aloite tiedon siirtämisestä. HTTP ja CoAP ovat tämän toimintamallin mukaisia protokollia (mts. 128). Julkaisija–tilaaja-mallissa laitteet voivat kommunikoida ulkoisen välittäjäpalvelun (broker) välityksellä tai niin, että jokin laite toimii välittäjänä (mts. 133). Laite tekee tilauksen tiettyyn aiheeseen liittyvästä viestistä. Julkaisija lähettää viestin tähän aiheeseen, ja tilaajat saavat lähetetystä viestistä ilmoituksen. Laitteilla ei ole erityisiä rooleja, ja kumpikin voi toimia joko julkaisijana ja tilaajana. Välittäjäpalvelu voi tehdä myös viestien puskurointia, eli jos tilaajalaite ei ole viestin lähetyksen hetkellä yhteydessä yhdistettynä välittäjäpalveluun, se saa viestin viivästetysti seuraavalla yhdistämiskerralla. Julkaisija–tilaaja-mallisia protokollia ovat AMQP, DDS, MQTT ja XMPP.

Protokollan lisäksi täytyy määritellä, minkä muotoista tietoa siirretään protokollan hyötykuormassa (payload). Näitä mahdollisuuksia on käsitelty tarkemmin luvussa 4.

3.2 Aikasarjadata

Aikasarja on ajan suhteen järjestetty havaintoarvojen joukko (Nummenmaa ym., 2014, s. 269). Aikasarja voi olla jatkuva tai diskreetti. Diskreetin aikasarjan arvot on otettu vain tiettyinä, yleensä tasavälisinä ajankohtina. Jatkuvassa aikasarjassa arvoja mitataan ja tallennetaan jatkuvasti. Käytännössä jatkuvia aikasarjoja mitataan diskreetisti digitaalisella mittalaitteistolla hyvin nopealla taajuudella, jolloin aikasarjaa voidaan pitää käytännön kannalta jatkuvana (mts. 270).

Paras tapa aikasarjojen tutkimiseen on niiden kuvaaminen graafisesti (Nummenmaa ym., 2014, s. 270). Yleensä graafinen kuvaus tehdään siten, että aika kuvataan x-akselilla ja aikasarjan arvot y-akselilla. Aikasarjan pisteet yhdistettään toisiinsa suoralla viivalla, jolloin kuvaaja on yhtenäinen ja mahdollisimman helppo hahmottaa. Kuva 1 antaa esimerkin aikasarjan kuvaamisesta. Kuvaajassa on mitattu vuorokauden 5 minuutin liukuva keskilämpötila eri mittausantureilta.



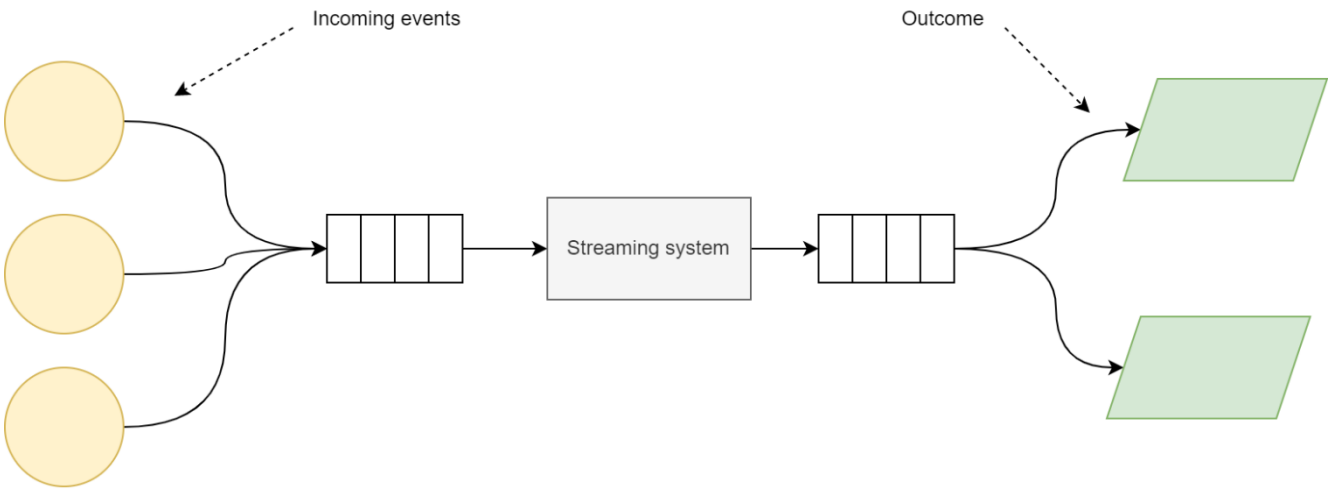
Kuva 1. Ulkolämpötilan mittauksia vuorokauden ajalta ThingsBoard-palvelussa aikasarjadatan kuvaajana.

Aikasarjadataa voidaan kerätä säännöllisesti kiinteällä aikavälillä tai epäsäännöllisesti, jolloin mittauspisteiden välinen aika vaihtelee suuresti (InfluxData, i. a.). Säännöllisessä datassa laite joko lähettää arvot säännöllisellä aikavälillä, esimerkiksi 1 sekunti, tai sovellus tallentaa mittauksen tutkimalla kohdetta ulkoisesti. Epäsäännöllisestä aikasarjadatasta

voidaan käyttää myös nimitystä tapahtumat (events), ja ne tapahtuvat yleensä jonkin ulkoisen tekijän vaikutuksesta. Tiedon vastaanottaja ei voi vaikuttaa milloin näitä tapahtumia syntyy toimintaympäristössä.

3.3 Stream processing

Big data -ympäristössä käytetään useasti termiä ”stream processing” kun käsitellään reaaliaikaisesti tai lähes reaaliaikaisesti tapahtumia (Fischer & Ning, 2022, s. 4). Tapahtumia voi olla monia erilaisia, mutta aikasarjadatan kanssa työskenneltäessä kysymyksessä on useimmiten jokin mittausarvo. Stream processing -järjestelmässä voidaan mittausarvoja käsitellä jatkuvana, ja se on myös samaan aikaan rajaton, eli sillä ei ole alkamis-, eikä loppumisajankohtaa. Kuvio 7 esittää yleisen kuvauksen stream processing -järjestelmästä. Sisään tulevat arvot puskuroidaan, ja käsittelijä purkaa niistä tietoa. Käsittelijöitä voi olla useampia, ja niiden välinen tiedonsiirto voidaan myös puskuroida. Ulos tulevat uudet arvot menevät muiden järjestelmien käyttöön.

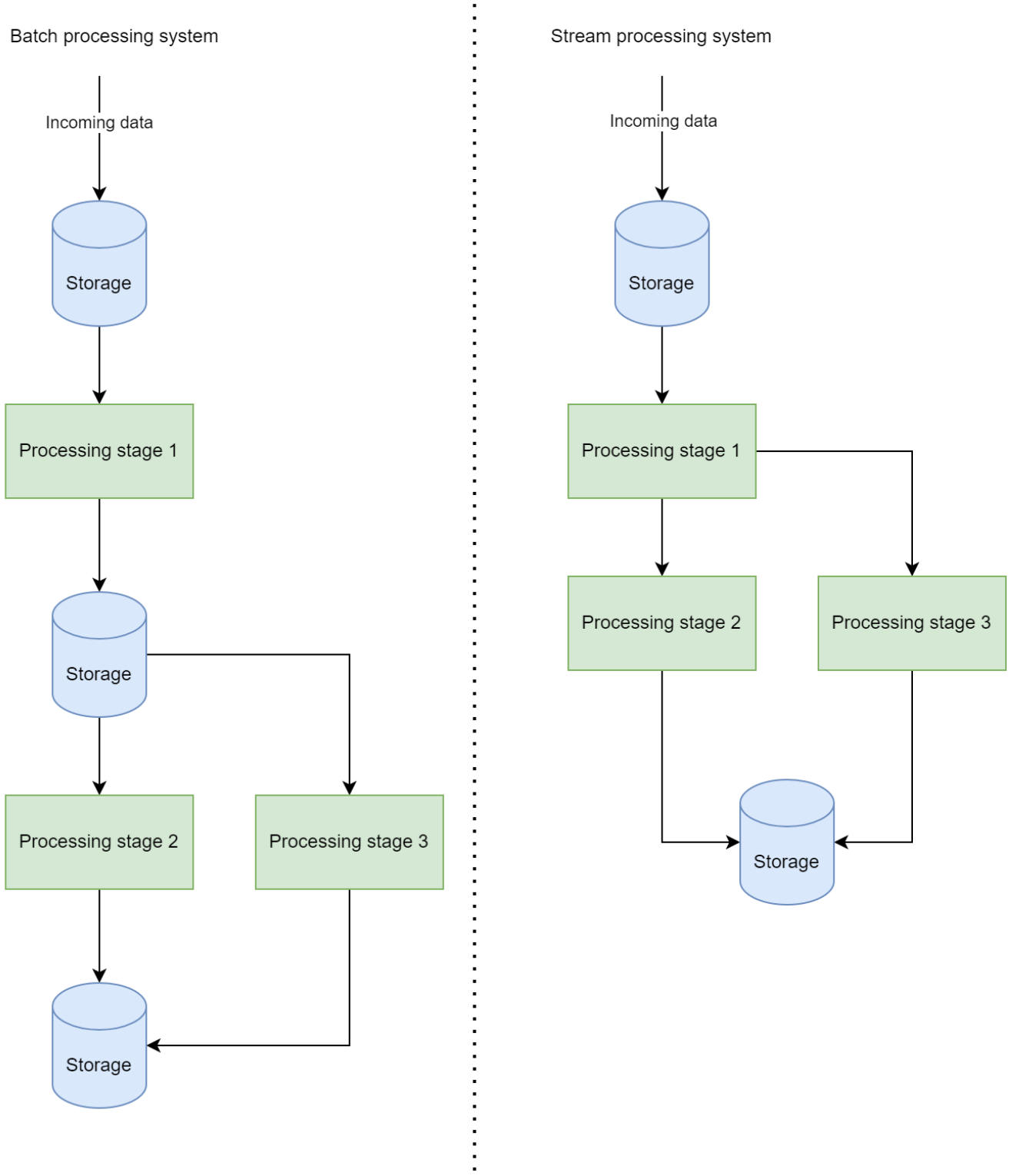


Kuvio 7. Stream processing -järjestelmän yleinen toimintaperiaate (Fischer & Ning, 2022, s. 6).

Sisään tulevaa tietoa voidaan käsitellä erissä tai jatkuvana virtana (Fischer & Ning, 2022, s. 11). Kuvio 8 havainnollistaa näiden kahden menetelmän käsittelijöiden rakenteellista eroa. Kun tieto viedään järjestelmän eri osille käsiteltäväksi suurempina kokonaisuuksina, on tieto yleensä tallennettuna massamedialaitteelle odottamassa seuraavaa käsittelyvaihetta. Eräkäsittelyssä pyritään maksimoimaan sen tiedon määrä, joka voidaan viedä tietoa

käsittelevästä järjestelmästä läpi. Uusi tieto ei ole heti käytettävissä, koska eräkäsittely suoritetaan vain etukäteen määriteltyinä ajankohtina.

Jatkuvana virtana käsitelty tieto kulkee koko putken läpi lähes reaaliaikaisesti ja tietoa ei talleteta massamedialaitteille käsittelyjen välissä (Fischer & Ning, 2022, s. 13). Tieto pidetään useimmiten tallessa järjestelmän käyttömuistissa. Tämän tyyppisellä järjestelmällä haetaan nopeaa vasteaikaa ja käsitellyn tiedon nopeaa saatavuutta jatkokäyttöä varten.

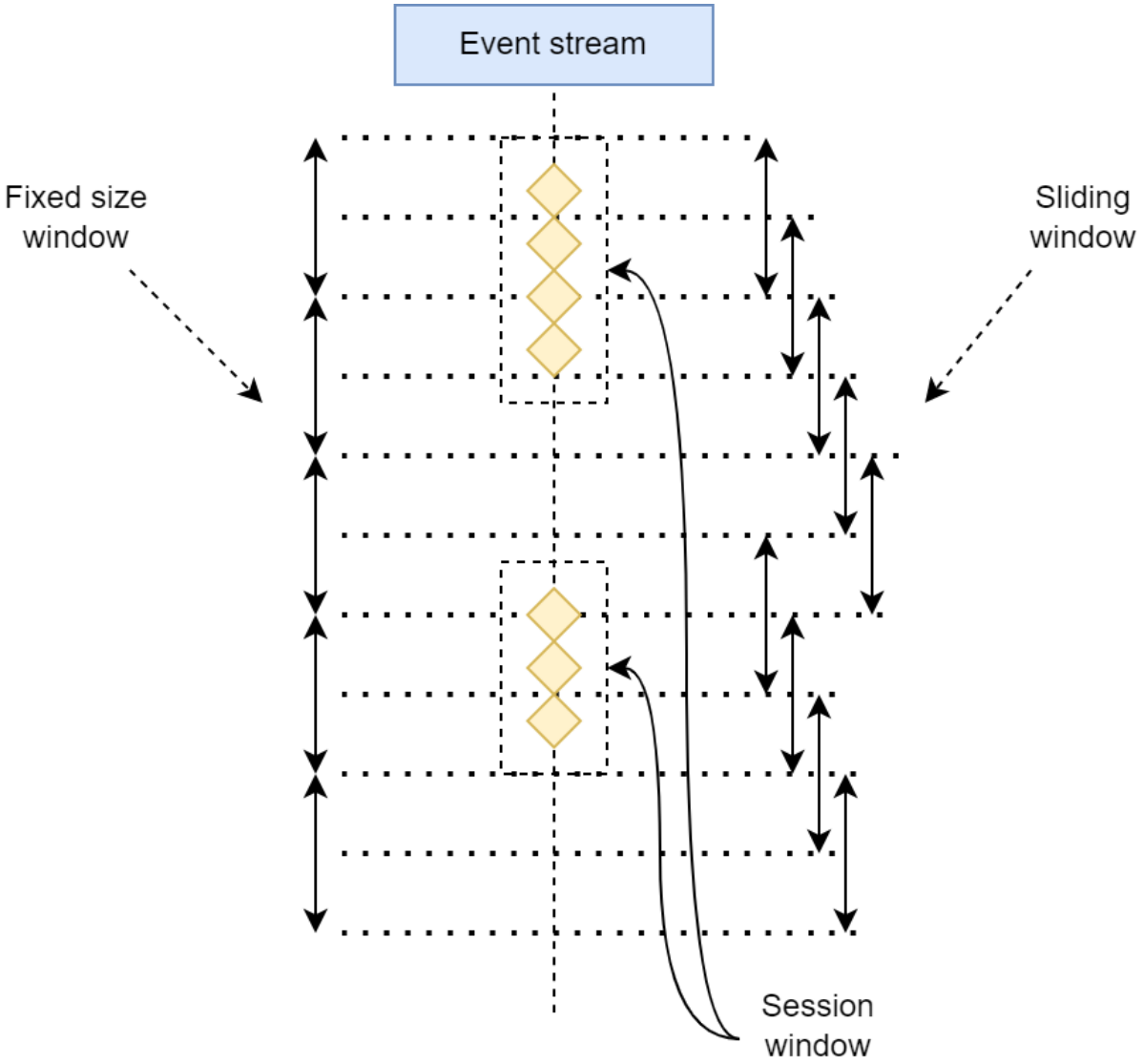


Kuvio 8. Tiedon rypäskäsittelyn ja jatkuvan virran käsittely (Fischer & Ning, 2022, s. 11–14).

3.4 Ikkunointi ja koostaminen

Datan ikkunoinnilla tarkoitetaan, että tietovirrasta otetaan määritellyn mittainen jana, joka kattaa ikkunan osuneet tapahtumat (Fischer & Ning, 2022, s. 163). Ikkunaan osuville

tapahtumille voidaan suorittaa tiedonkäsittelyä ja koostaa arvoista uusia tapahtumia. Ikkunoita on pääsääntöisesti kolmea erilaista tyyppiä: kiinteän kokoinen ikkuna, liukuva ikkuna sekä istuntoikkuna. Erilaiset ikkunointityypit on esitetty kuviossa Kuvio 9.



Kuvio 9. Kiinteä ikkuna, liukuva ikkuna ja sessioikkuna samassa tietovirrassa (Fischer & Ning, 2022, s. 174).

Kiinteä ikkuna on vakiomittainen, yleensä aikaan sidottu otanta tietovirrasta (Fischer & Ning, 2022, s. 164). Tietovirrasta voidaan poimia esimerkiksi minuutin mittainen otanta tietoa. Seuraava otanta alkaa heti vanhan ikkunan jälkeen. Koska ikkunan pituus on kiinteä, ikkunaan osuvien tapahtumien määrä saattaa vaihdella ikkunoiden välillä.

Liukuva ikkuna poikkeaa kiinteästä ikkunasta siten, että ikkunat voivat kohdistua samoihin tapahtumiin tapahtumavirrassa (Fischer & Ning, 2022, s. 168). Ikkuna voidaan sitoa joko aikaan tai tapahtumien määrään. Esimerkiksi datavirrasta voidaan poimia 10 tapahtumaa ensimmäiseen ikkunan ja seuraavaan ikkunaan otetaan yksi uusi arvo datavirrasta, joten ikkunat jakavat keskenään samat 9 tapahtumaa.

Istuntoikkuna pyrkii etsimään tietovirrasta tapahtumia, jotka ovat tapahtuneet suhteellisen lähellä toisiaan (Fischer & Ning, 2022, s. 171). Esimerkiksi tietovirrassa voi olla useampi hälytys, jotka ovat tapahtuneet ajallisesti lähellä toisiaan ja näitä hälytyksiä ei muuten havaita tietovirrassa. Ikkunalle täytyy määritellä, miten iso aikapoikkeama kahden tapahtuman välillä on, että ne tulkitaan eri istuntoihin kuuluviksi.

Ikkunoidulle datalle voidaan tehdä tämän jälkeen erilaista suodattamista, muuttamista ja koostamista. Näissä operaatiossa käytetään tilastollisia menetelmiä, jolla tietoa pyritään järjestelemän uudelleen.

Perustekniikat datan muuttamiseen ovat (Vogiatzis, 2022):

- Datan tasoittaminen (Smoothing)
- Datan yleistäminen (Generalization)
- Datan koostaminen (Aggregation)
- Datan luokittelu (Discretization)
- Datan normalisointi (Normalization)

Datan tasoittamisella raakadatasta pyritään suodattamaan mittausvirheitä pois (Vogiatzis, 2022). Arvot voivat olla esimerkiksi selkeästi tutkittavan alueen ulkopuolella, joten arvot voidaan luokitella poikkeaviksi. Datan klusteroinnissa tieto ryhmitellään sen mukaan, miten lähellä arvot ovat tosiaan. Klusterin ulkopuolelle jäävät arvot voidaan tulkita poikkeaviksi ja suodattaa pois lähdedatasta. Lokeroinnilla (binning) tieto voidaan luokitella samankokoisiin lokeroihin. Lokerosta johdetaan uusi suure, esimerkiksi keskiarvo, joka kuvaa kaikkia lokeron sisältämiä arvoja. Regressiossa datan liikkumista voidaan ennakoida seuraamalla jotain toista suuretta. Jos tiedetään, että arvon tulisi muuttua tietyllä tapaa suhteessa toiseen mittavaan arvoon, voidaan tilanteissa, joissa arvot käyttäytyvät säännönmukaisuudesta poikkeavalla tavalla todeta, että arvot voidaan jättää huomioimatta.

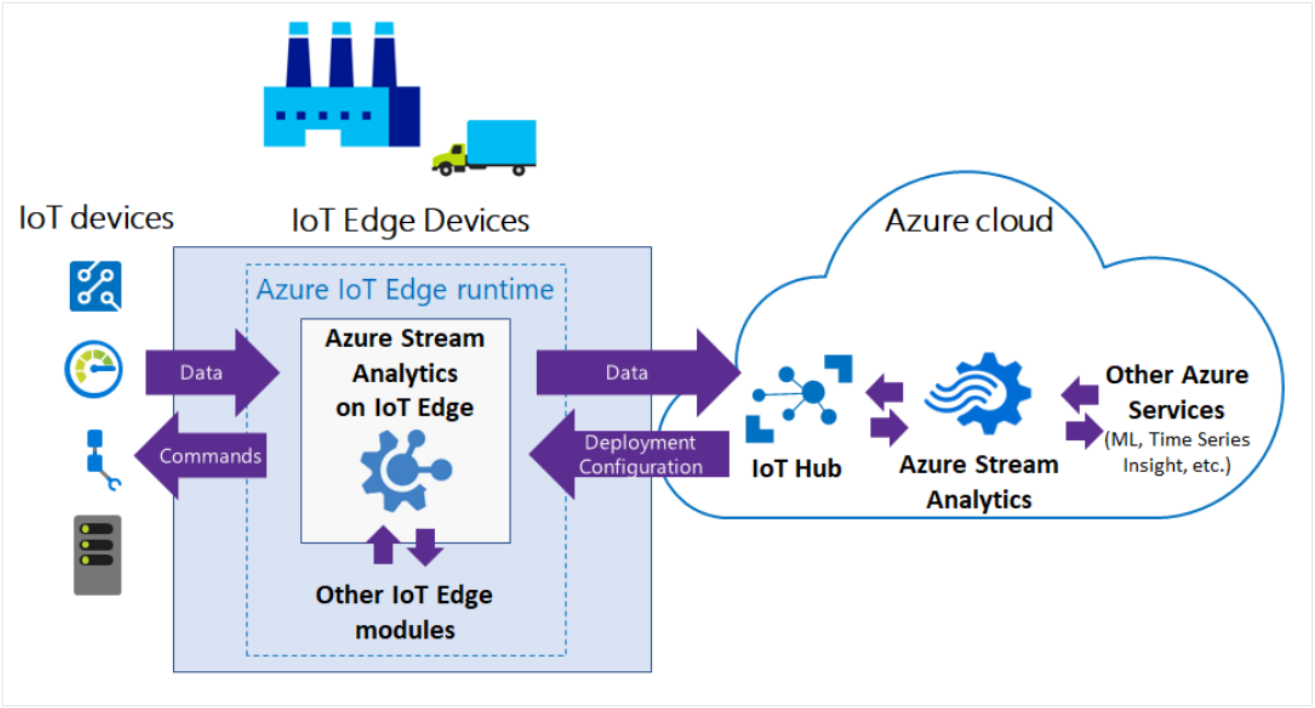
Datan yleistämisessä (generalization) data pyritään muuttamaan matalamman tason tiedosta korkeamman tason tietoon (Vogiatzis, 2022). Esimerkiksi kadun nimet voidaan niputtaa kuuluvaksi tiettyyn kaupunginosaan ja kaupunginosa kuuluvaksi tiettyyn kaupunkiin. Tästä syntyy puumainen rakenne, jossa siirryttäessä korkeammalta tasolta alemmas, alkaa tiedon yksityiskohtaisuus kasvaa. Datan koostamisessa (aggregation) tutkittavasta datasarjasta muodostetaan ikkunoinnilla uusia arvoja, joista johdetaan uusi yleistetympi datasarja. Yleisimpiä koostamisfunktioita ovat keskiarvo, summaus, minimi, maksimi, arvojen määrä ja keskihajonta.

Datan luokittelussa datasarjaa tutkitaan muodostamalla ryhmiä, joissa arvoja voidaan käsitellä ryhmän kautta (Vogiatzis, 2022). Esimerkiksi vuorokauden pituisista mittaussarjoista voidaan valita vain tiettyyn vuorokaudenaikaan kerätyt arvot, jotka esimerkiksi kohdentuvat aamulle. Normalisoinnissa lähdedata pyritään muuttamaan helpommin esitettävään muotoon ilman, että tieto muuttuu tai menetetään lähdedatan tarkkuutta. Yksi toimenpide on datan skaalaaminen. Skaalauksessa lähdearvot muutetaan käyttämällä sopivaa kerrointa, jolloin data sopii paremmin minimi- ja maksiarvojen välille.

3.5 Azure Stream Analytics on IoT Edge

Microsoft Azure Stream Analytics on datavirtojen reaaliaikaiseen käsittelyyn kehitetty työkalu Microsoft Azure -pilvipalveluun (Microsoft, 2022a). Työkalu pystyy vastaanottamaan tietoa monipuolisesti muista Azuren palveluista kuten event hubista, blob storagesta ja IoT hubista. Reunalaskennassa käsitelty tieto voidaan lähettää Azuren sisällä taas muille Azuren palveluille, kuten Power BI -palvelulle. Azure Stream Analytics toimii SQL-pohjaisella kielellä, jota käytetään järjestelmän toiminnan ohjaukseen.

Microsoft Azure Stream Analytics palvelun yksi käyttötapaus on myös tiedonkäsittelyn vieminen Azure IoT Edge -laitteelle (Microsoft, 2021). Tässä tapauksessa tiedon käsittely ei toimi Azuren pilvipalvelussa vaan laitteella, jossa on käytössä Azuren IoT Edge -ohjelmistoalusta. Laitteelle asennetaan Azure Stream Analytics -ohjelmisto, joka toimitetaan ohjelmistokontteina. Laite liitetään Azuren pilvipalveluun Azure IoT Hub -palvelun avulla. Stream Analytics -palvelun hallinta tapahtuu Azuren Stream Analytics -palvelusta ja se toimii itsenäisesti IoT Edge -laitteella, eikä vaadi jatkuvaa yhteyttä Azure IoT Hub -palveluun. Kuva 2 havainnollistaa Azure Stream Analyticsin käyttämistä edge-laitteella. Edge-laitteen ulkopuolelta tuotetaan tietoa, joka käsitellään paikallisesti ja siirretään Azuren pilvipalveluun käsittelyn jälkeen.

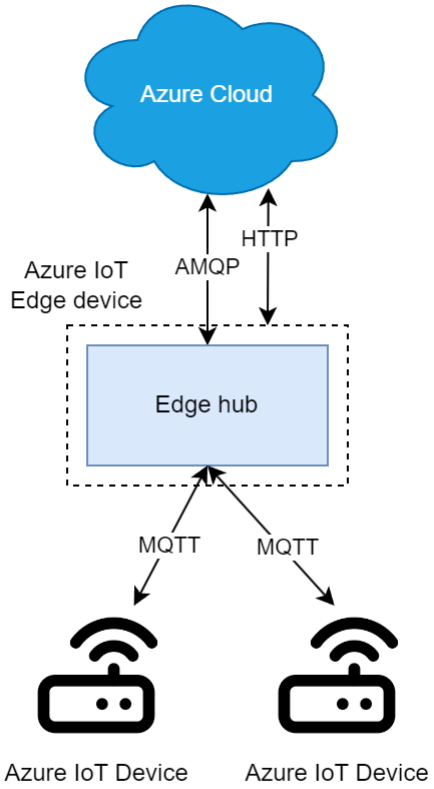


Kuva 2. Azure Stream Analytics toiminta Azure IoT Edge -laitteessa (Microsoft, 2021).

Azure IoT Edge on ohjelmistokokonaisuus, joka asennetaan edge-laitteelle (Lea, 2020, s. 351). Se tarjoaa työkalut ohjelmistokontteihin (containers) perustuvien ohjelmien asentamiseen ja käyttämiseen reunalaskentalaitteissa. Laitteiden hallinta tapahtuu Azuren IoT Hub -palvelun avulla, missä määritellään sovellukset, joita Azure IoT Edge -laitteessa aiotaan käyttää.

Azure IoT Edge -ohjelmisto koostuu neljästä komponentista: IoT edge security daemon, IoT edge agent, IoT edge hub ja container engine (Microsoft, 2022b). Security daemon on ensimmäinen sovellus, joka alustalla käynnistetään. Se yhdistää laitteen Azure IoT hub -palveluun ja hakee tiedon siitä, mikä versio IoT edge agent -ohjelmistosta sen täytyy käynnistää. Security daemon komentaa container engine -palvelun hakemaan ja käynnistämään annetun IoT edge agent -ohjelmistokontin. IoT edge agent hoitaa järjestelmässä muiden sovelluksien käynnistämisen, sekä valvoo niiden toimintaa. Kun käyttäjä on Azure IoT Hub -palvelussa asettanut deployment-kuvauksen, eli mitä sovelluksia laitteella halutaan ajaa, hoitaa IoT agent näiden sovelluksien lataamisen laitteelle ja käynnistämisen. Jokaisessa toimivassa IoT edge -kokoonpanossa täytyy olla mukana myös IoT hub -ohjelmisto, joka hoitaa viestiliikenteen välittämistä IoT edge -alustalla toimivien sovelluksien ja Azuren IoT Hub -palvelun välillä.

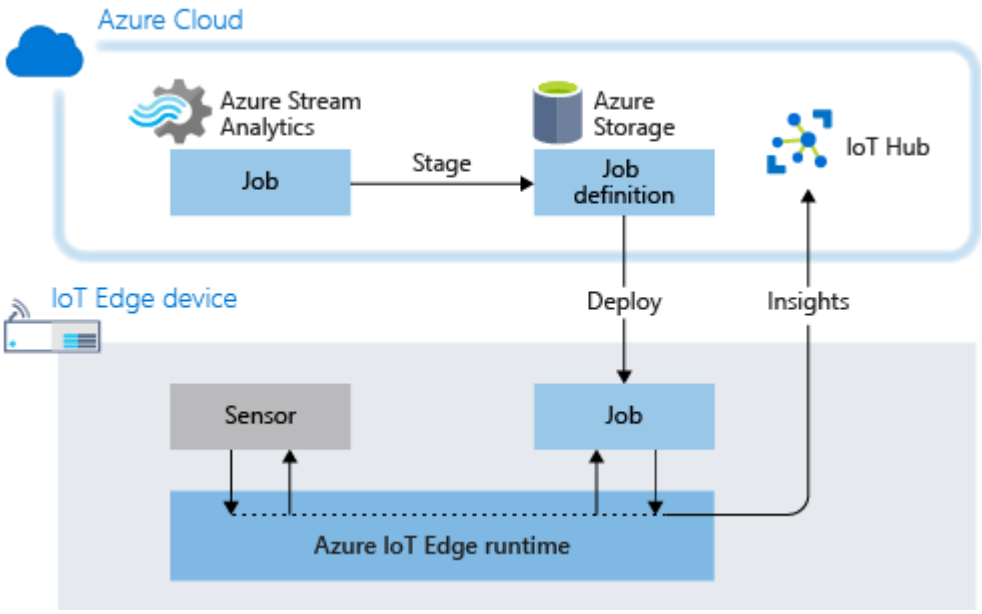
Azure IoT Edge -laite toimii tässä käyttötapauksessa niin sanottuna yhdyskäytävänlaitteena (Microsoft, 2022c). IoT edge hub reitittää edge-laitteeseen yhdistävät muut IoT-laitteet niin, että sisään tuleva tieto voidaan ohjata edge-laitteen sisäisille palveluille tai Azuren palveluille. IoT-laitteet yhdistyvät edge-laitteeseen käyttäen MQTT-protokollaa, ja IoT edge hub yhdistää sisään tulevat yhteydet yhdeksi AMQP-protokollaa käyttäväksi yhteydeksi. Kuva 3 hahmottaa laitteiden välisiä yhteyksiä ja hierarkiaa. MQTT-protokollan toteutus ei ole tavanomainen, vaan IoT-laitteiden täytyy tunnistautua ja muodostaa salattu yhteys edge-laitteeseen käyttäen Microsoftin määrittelemää kommunikointitapaa. Microsoft suositteleekin käytettäväksi heidän toimittamiaan SDK-paketteja IoT-laitteiden ohjelmiston kehittämiseen (Microsoft, 2022d).



Kuvakkeet: CC-BY 4.0, Tabler Icons

Kuva 3. Azure IoT Edge -laite yhdyskäytävänä. (soveltaen Microsoft, 2022c).

Azure Stream Analytics on IoT edge palvelu, jota ajetaan Azure IoT Edge -laitteessa, mutta tiedonkäsittelytehtävien määrittely tapahtuu Azuressa pyörivän Stream Analytics -palvelun avulla (Microsoft, 2022e). Kuva 4 havainnollistaa, miten eri järjestelmät ja palvelut kytkeytyvät toisiinsa.



Kuva 4. Kuvaus Azure Stream Analytics on IoT Edge -palvelun käyttämisestä (Microsoft, 2022e).

Palvelun käyttäjä kirjoittaa ohjelman joko Stream Analytics hallintatyökalulla Azure Portal -palvelussa tai kehittäjätyökaluilla, jotka toimivat Microsoft Visual Studio Code -työkalussa. Ohjelma on SQL-kielinen kuvaus siitä, miten dataa halutaan käsitellä laitteen sovelluksessa. Azure Stream Analytics käyttää T-SQL nimistä muunnosta SQL-kielestä, mistä vain osa toiminnoista on käytössä reunalaskennassa. Kuva 5 sisältää esimerkkiohjelman, jossa lähetetään hälytys, kun mitattava lämpötila on ollut yli 30 sekuntia yli asetetun raja-arvon. Ohjelmaan viitataan myös nimikkeellä ”tehtävä”. Tehtäväkuvaus talletetaan Azure Storage -palveluun. Laitteella toimiva Stream Analytics sovellus käy itsenäisesti hakemassa uuden tehtäväkuvauksen ja ottaa sen käyttöön laitteella. Tehtäväkuvauksen päivitys voidaan tehdä myös ryhmänä, eli kohdistaa moneen eri laitteiseen, jolloin sama tehtäväkuvaus tulee heti käyttöön koko laitekantaan.

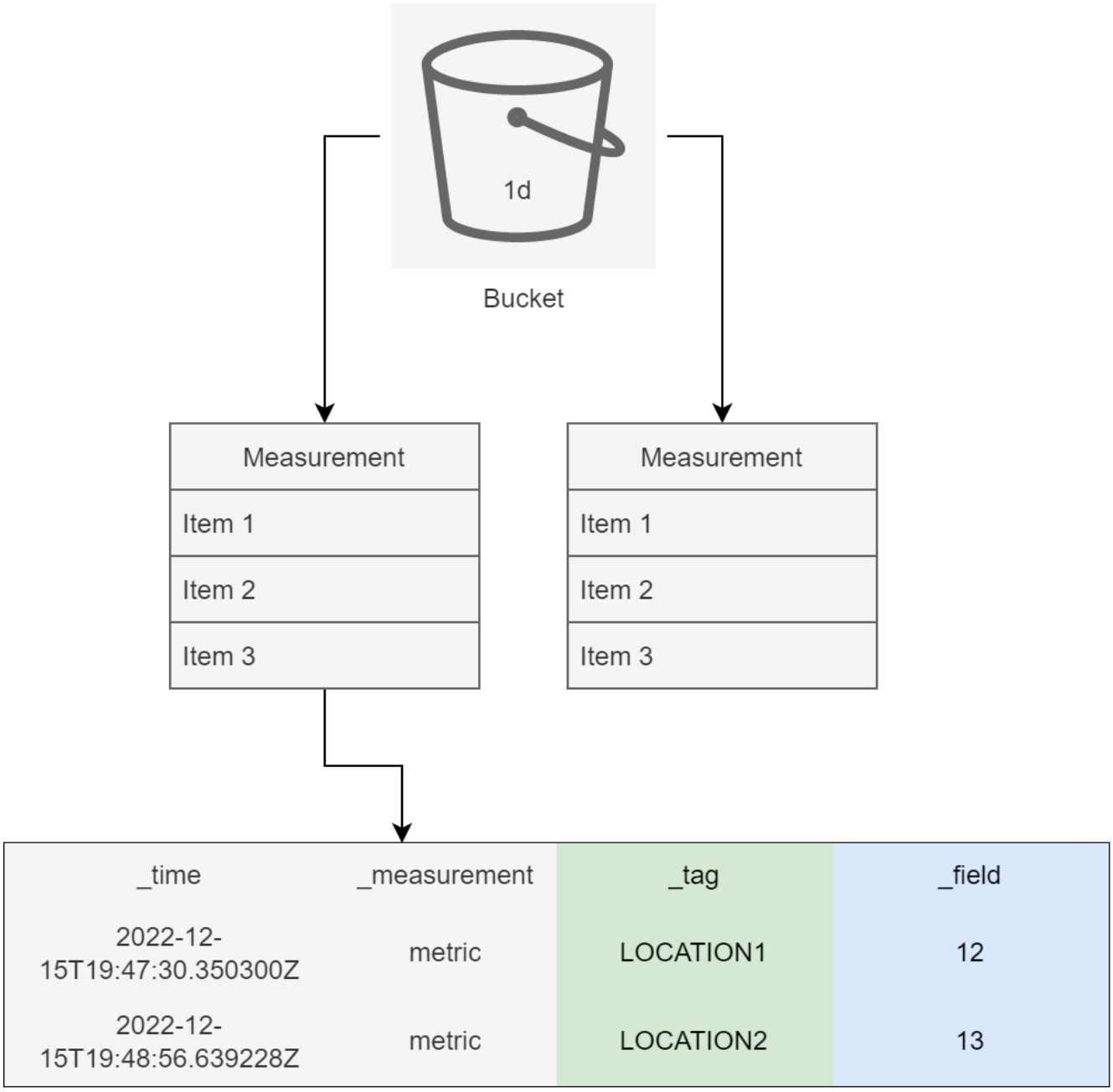
```sql
SELECT
    'reset' AS command
INTO
   alert
FROM
   temperature TIMESTAMP BY timeCreated
GROUP BY TumblingWindow(second,30)
HAVING Avg(machine.temperature) > 70
```

Kuva 5. Esimerkki Azure Stream Analytics SQL-kielisestä ohjelmasta (Microsoft, 2022e).

3.6 InfluxDB

InfluxDB on hiukan poikkeava työkalu tässä ryhmässä. InfluxDB on NoSQL-tietokanta, joka on erikoistunut aikasarjamuotoisen tiedon varastointiin ja käsittelyyn (Influxdata, 2022b). Tietokanta on kuitenkin toteutettu siten, että se on helppo asentaa toimimaan myös edge-laitteelle. Ohjelmointi tapahtuu työkalun omalla flux-ohjelmointikielellä. Tieto tuodaan tietokantaan RESTful Web Service -rajapinnan avulla tai tietokantaan sisäänrakennetulla telegraf-työkalulla. Tietokanta on optimoitu suurien datamäärien käsittelyyn, ja sillä voidaan tehdä automaattista tiedon koostamista. Tietokanta osaa myös poistaa vanhaa tietoa automaattisesti kehittäjän tekemien sääntöjen perusteella.

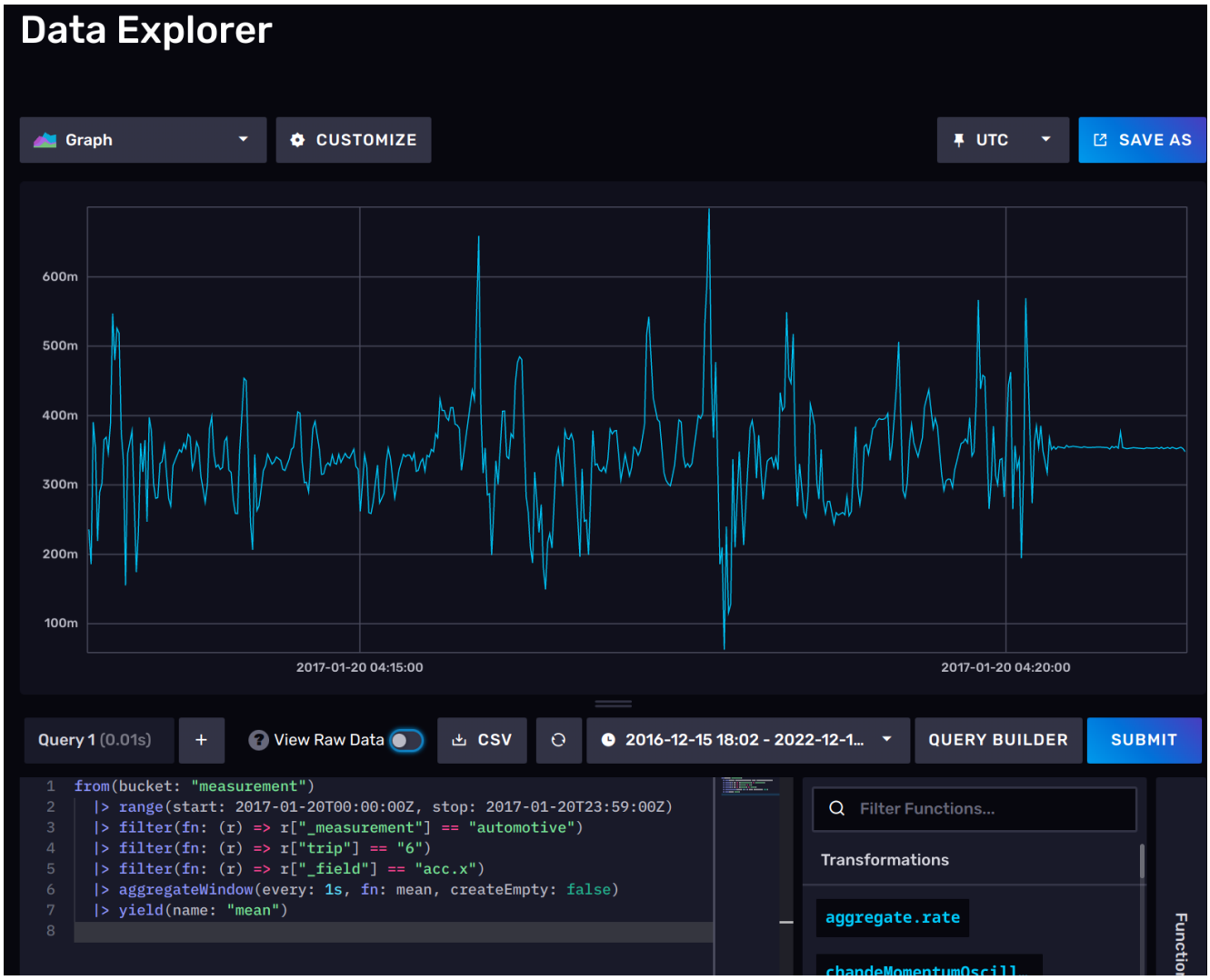
Tietokannan datamalli perustuu sarjoihin, joita kutsutaan nimellä ”measurements” eli mittaukset (Marble, 2022). Mittaukset varastoidaan lokeroissa (bucket) ja lokerolla on asetus, miten pitkään tietoa varastoidaan. Kuva 6 hahmottaa tietokannan rakennetta. Mittauksessa on aina mukana kenttä ”time” eli aikaleima. Mittaukseen voidaan liittää myös kenttiä (fields) ja tageja. Kentät sisältävät varsinaisia mittausarvoja, jotka liittyvät aikaleiman osoittamana ajan hetkeen. Kentällä on tunniste, jolla voidaan nähdä mikä mittaussarja on kyseessä. Tageilla voidaan mittaukseen liittää metatietoa, joka ei ole varsinaista mittaustietoa, mutta auttaa erottamaan eri mittaukset toisistaan. Kentät ja tagit ovat molemmat avain–arvopareja ja niiden merkittävän ero on siinä, että tagit ovat indeksoituja ja kentät eivät. Tageihin tulisikin sijoittaa sellaista tietoa, mitä useimmiten haetaan mittauksesta, jolloin tietokannan toiminta nopeutuu, kun mittaussarjaa eri tarvitse tutkia kokonaan tietokantakyselyjen yhteydessä.



Kuva 6. Esimerkki influxDB-tietokannan rakenteesta ja mittauksien sisällöstä. (soveltaen Marble, 2022).

Flux-ohjelmointikieli on funktionaalinen ohjelmointikieli, joka on kehitetty tiedon käsittelyyn InfluxDB-tietokannassa (Influxdata, 2022a). Sillä voidaan tehdä kyselyjä, tiedonkäsittelyä ja analysointia tietokannassa olevaan tietoon. Kysely alkaa from-komennolla, jolla kerrotaan tiedon lähde (source) eli mistä sarjasta tietoa lähetään keräämään. Kyselyn alle voidaan lisätä useita ”pipe-forward operator” (|>) komentoja, joilla tietovirta ohjataan käsittelijälle. Käsittelijöitä on useita erilaisia tiedon suodattamiseen, rajaamiseen ja koostamiseen. Kerätystä tiedosta voidaan laskea myös uusia arvoja, esimerkiksi keskiarvo.

Kuva 7 havainnollistaa esimerkkikyselyn luomisen influxDB Data Explorer -työkalulla. Kyselyssä valitaan lähteeksi ”measurement” lokero, josta suodatetaan arvot valitulle aikajaksolle. Tämän jälkeen valitaan mitkä mittaussarja (”automotive”), matka (”6”) ja kenttä (”acc.x”) halutaan kyselyyn. Lopuksi kyselyn vastauksena tuleviin arvoihin tehdään yhden sekunnin kiinteä ikkunointi, josta lasketaan keskiarvo. Viivakuvaaja esittää kyselyn tuloksena tulevien arvojen muuttumisen valitulla aikavälillä.



Kuva 7. Esimerkki Flux-kyselyn käyttämisestä ja tuloksen visualisoinnista InfluxDB-käyttöliittymässä.

3.7 Redis Timeseries

Redis on avoimeen lähdekoodiin perustuva muistinvarainen tietokantajärjestelmä (Redis, 2022a). Tarkemmin määriteltynä se on tietorakennevarasto, koska ohjelmisto sisältää useita muitakin ominaisuuksia. Tietomalli käyttää avain–arvopareja tiedon tallentamiseen. Avaimet ovat indeksoituja, ja niitä käytetään tiedon hakemiseen tietokannasta. Tietorakennevaraston lisäksi järjestelmä tarjoaa julkaisija–tilaaja-mallisen viestintäjärjestelmän, ohjelman toimintojen laajentamisen Lua-ohjelmointikielellä, viestinvälityspalvelun ja järjestelmän hajauttamisen useampaan klusterin.

Tieto varastoidaan käyttömuistissa, mistä sen käyttäminen on erittäin nopeaa (Redis, 2022a). Muistinvarainen tietokanta kopioidaan säännöllisesti myös pysyväismuistiin, jolloin tieto säilyy myös vikatilanteissa ja järjestelmän uudelleenkäynnistysten välissä. Redis on suosittu työkalu esimerkiksi web-palvelinsovellusten välimuistina.

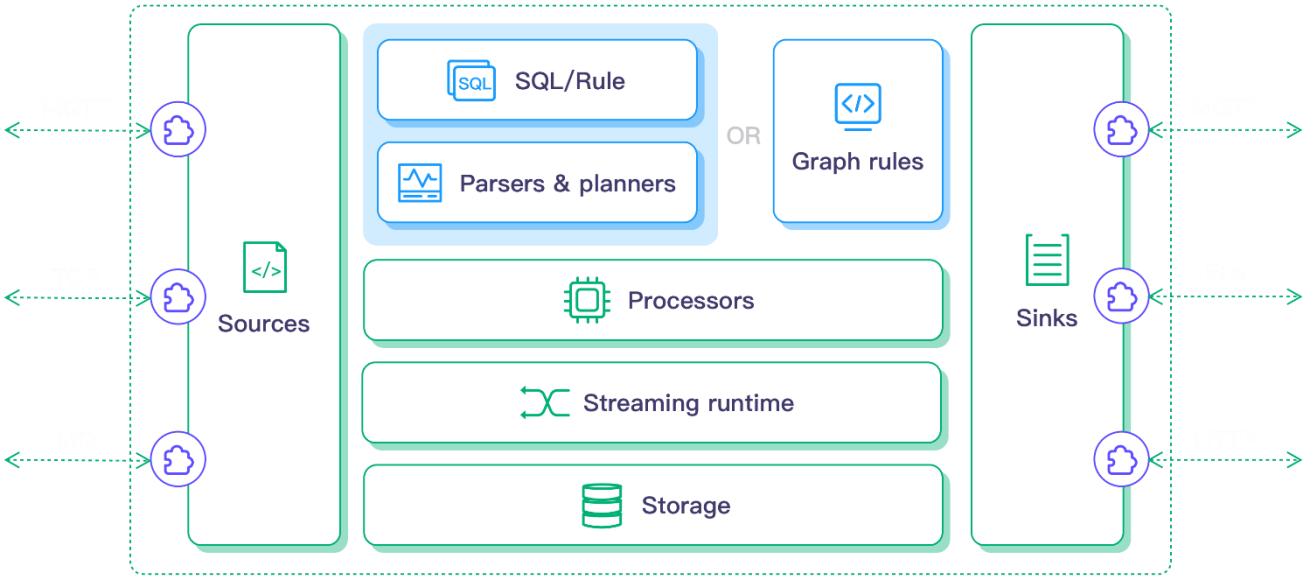
Redis Timeseries on laajennus ja liitännäinen Redis-tietokantaan (Redis, 2022c). Se täytyy asentaa erikseen Redisin asennuksen jälkeen tai käyttää valmiiksi paketoitua ohjelmistokonttia. Laajennus lisää redisiin aikasarjadataan liittyviä ominaisuuksia. Tietokantaan voidaan luoda aikasarja-tyyppinen tietorakenne, johon käyttäjä kirjoittaa arvoja aikaleimasta ja arvosta koostuvina pareina.

Aikasarjalle voidaan määritellä säilytysaika, ja tietokanta osaa poistaa automaattisesti arvoja aikasarjasta, kun asetettu vanhenemisaika ylittyy (Redis, 2022b). Kyselyissä voidaan antaa aikajana, miten pitkältä ajalta halutaan arvoja kerätä ja tehdäänkö arvoille jokin koostamisoperaatio. Koostamisoperaatiosta voidaan määritellä myös sääntö, joka tuottaa jatkuvasti koostettua aikasarjaa, kun lähdetietoa päivitetään.

3.8 eKuiper

LF Edge eKuiper on EMQ yrityksen kehittämä avoimen lähdekoodin tuote IoT data-analytiikkaan ja datavirojen käsittelyyn (eKuiper, i.a.). Tavoitteena on tuottaa kevyt ohjelmisto, joka toimii myös niukasti resursseja tarjoavassa reunalaskentalaitteessa. Ohjelmiston toiminta perustuu datalähteisiin (sources), SQL-kielellä määriteltävään sääntöpohjaiseen tiedonkäsittelyyn ja kohteisiin (sinks), mihin käsitelty tieto syötetään (kuva 8). Hallinta tapahtuu REST-rajapinnan välityksellä tai erillisellä web-pohjaisella management-työkalulla. Ohjelmistoa

voidaan ajaa itsenäisenä palveluna tai se voidaan integroida osaksi EdgeX Foundry -ohjelmistoalustalla.



Kuva 8. eKuiper ohjelmiston komponentit (eKuiper, i.a.).

Datan lähde (source) voi olla jatkuva tietovirta (stream), otantataulu (scan table) tai hakutaulu (lookup table) (eKuiper, i.a.). Jatkuva tietovirta saadaan kytkeytymällä eKuiperin tukemaan tietolähteeseen esimerkiksi MQTT-viestinvälityspalveluun. Tietovirta soljuu jatkuvalla syötöllä järjestelmään, ja sisään tulevat tiedot käsitellään sääntömoottorissa lähes reaaliaikaisesti. eKuiper tukee oletuksena binääri-, erotinmerkki-, JSON- tai protobuf-muotoon jäsenneltyä tietoa. Tukea pystyy laajentamaan omilla laajennuskomponenteilla.

Tietovirrasta voidaan koota myös muistinvaraiseen puskuri, jossa on otanta viimeisimmistä data-arvoista (eKuiper, i.a.). Otantataulu (scan table) ja hakutaulu (lookup table) ovat rakenteeltaan hyvin samanlaisia. Ratkaiseva ero on, että otantataulu voi sisältää useamman tietueen samasta datasta, kun taas hakutaulussa säilytetään vain viimeisintä. Otantataulu voidaan sitoa myös vain yhteen sääntöön, kun taas hakutaulu voidaan jakaa monen säännön kesken. Molemmat voidaan sitoa myös tietovirran sijasta esimerkiksi tiedostoon. Otantataulun ja hakutaulun käyttökohde on esimerkiksi sisään tulevan tiedon rikastaminen. Sisään tulevassa tieto ei välttämättä sisällä yksityiskohtaista tietoa lähettäjästä (esim. laite), joten tämä tieto voidaan liittää toisesta lähteessä haetusta tiedosta. Tieto yhdistetään yksilöivällä tunnistekentällä (vrt. SQL-kyselyn JOIN-operaatio).

Tiedon käsittely sääntömoottorissa tapahtuu SQL-tyyppisellä ohjelmointikielellä (eKuiper, i.a.). Tiedonkäsittelysääntöön määritellään mistä tieto haetaan, liitetäänkö siihen lisätietoa muista lähteistä (otantataulu tai hakutaulu), käytetäänkö tiedon hakemiseen ikkunointia ja millaisilla ehdoilla tietoa kerätään. Kyselyjä voidaan ketjuttaa useampia yhteen kirjoittamalla kyselyn tulos väliaikaiseen muistinvaraiseen puskuriin. Kyseleistä voidaan muodostaa myös kaaviorakenteita (Direct Acyclic Graph, DAG), jossa voidaan yksinkertaistetuilla säännöillä muodostaa monimutkaisempia kokonaisuuksia. eKuiper tukee aikasidonnaista kiinteää ikkunaa, liukuvaa ikkunaa ja sessioikkunaa. Lisäksi tuettuna on tapahtumien määrään perustuva ikkunointi.

Esimerkissä (kuva 9) tehdään kaikille mittapisteille ryhmittelyoperaatio (GROUP BY) tiedon tunnisteen ja arvosta muodostetun kiinteän ikkunan perusteella. Kiinteä ikkuna on 10 sekunnin mittainen, ja sitä siirretään 5 sekuntia eteenpäin kerrallaan. Lopputuloksesta annetaan ulos tiedon tunniste, arvojen keskiarvo ikkunan sisällöstä, sekä ikkunan aloitus ja lopetusajankohtien aikaleima.

```
SELECT key, avg(value) as average, max(t) as end_ts, min(t) as begin_ts
FROM telemetry
GROUP BY key, HOPPINGWINDOW(ss, 10, 5);
```

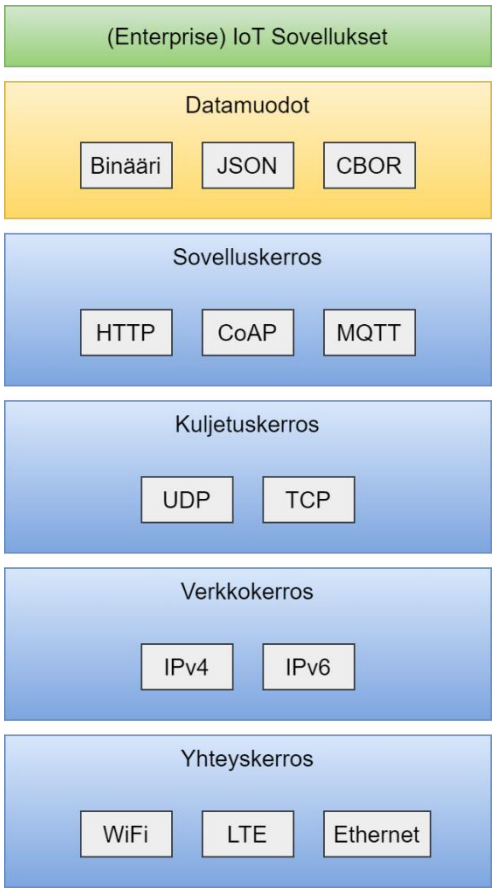
Kuva 9. Esimerkki eKuiperin SQL-säännöstä.

Kun tieto on käsitelty, se ohjataan yhteen tai useampaan kohteeseen (sink), jotka voivat olla MQTT-viestin julkaisu tai HTTP-kutsun tekeminen (eKuiper, i.a.). Kohteena voi toimia myös InfluxDB-tietokanta tai Redis-tietokanta. Näin siis eKuiperia voi käyttää myös yhdessä aikaisemmin esiteltyjen työkalujen kanssa. Jos kohde on muistinvarainen puskuri, voidaan julkaistua tietoa käyttää järjestelmän sisällä muissa säännöissä.

4 TELEMETRIATIEDON PAKETOINTI

Telemetriatiedon paketoinnissa ja lähettämisessä on kysymys tiedon vaihtamisessa usean eri ohjelman ja järjestelmän välillä. Tiedon vaihtamisessa voidaan tunnistaa kolme elementtiä: arkkitehtuurimalli (architectural pattern), datamuoto (data format) ja kommunikointitapa (communication protocol) (Charest & Rogers, 2020). Kun halutaan valita sopiva datamuoto järjestelmän käyttöön, täytyy myös ottaa huomioon käytettävä arkkitehtuurimalli ja kommunikointitapa. Esimerkiksi jos käytössä on viestinvälitykseen perustuva kommunikointimalli, niin silloin on parempi käyttää siihen sopivia datamuotoja kuin sellaisia, jotka on tarkoitettu tiedostojen siirtämiseen perustuvaan malliin.

Korkean tason arkkitehtuurissa IoT-järjestelmän tiedonsiirto rakentuu protokollapinosta (McCain, 2023, s. 20). Pinon alin kerros on linkkitaso, joka määrittää käytettävän median tiedonsiirtokanavalle. Esimerkkejä yhteyskerroksen teknologioista ovat WiFi, LTE tai Ethernet. Seuraava taso on verkkotaso, jossa toimivat esimerkiksi IP-protokollan versiot 4 ja 6. Kuljetuskerros on tämän yläpuolella oleva kerros, ja sen protokollia ovat UDP ja TCP. Sovellustasolla toimivat varsinaiset sovellusprotokollat, kuten CoAP, HTTP ja MQTT. Datamuoto löytyy sovellustason yläpuolelta protokollapinossa, ja pinon huipulla ovat sitten varsinaiset IoT-ohjelmistot. Kuvio 10 esittää tämän protokollapinon rakenteen.



Kuvio 10. IoT-järjestelmän protokollapino (soveltaen McCain, 2023, s. 21).

Datamuodon (ja tietomallin) valintaa ohjaavat erilaiset näkökulmat itse lähetettävään tietoon (Charest & Rogers, 2020). Tiedon monimutkaisuus asettaa vaatimuksia, esimerkiksi siihen suuntaan, miten erilaisia rakenteita voidaan datamuodolla esittää. Tasainen (flat) datamuoto, kuten esimerkiksi CSV on hankala käyttää, jos on tarvetta esittää hierarkkisia rakenteita. JSON ja XML sopivat paremmin tämän tyyppisen tiedon esittämiseen. Useasti päivittyvä tieto saattaa lisätä ylimääräisen tiedon lähettämistä, jos tietoa tarvitsee synkronoida eri järjestelmien välillä. Lisäksi tiedon koko vaikuttaa suoraan siihen, miten tietoa olisi tehokkainta käsitellä. Isoista tietomääristä halutaan useasti hakea vain osia, tekemällä tietoon kyselyjä.

Arkkitehtuurimallin, datamuodon ja kommunikointitavan yhdistelmä tuottaa myös ympäristöön liittyviä näkökulmia (Charest & Rogers, 2020). Tiedon virtaaminen eri järjestelmien välillä, tiedon päivitystiheys, tiedon versiointi, tietoturva, tiedonsiirron monimutkaisuus ja tiedonsiirtokanavan pysyvyys vaikuttavat datamuodon valintaan.

Datamuoto voidaan luokitella strukturoiduksi tai strukturoimattomaksi tiedoksi (Smallcombe, 2023). Strukturoitu tieto on standardoitua, tarkasti määriteltyä ja helposti haettavaa tietoa. Se on saatavilla ennakkoon määritellyssä datamuodossa, ja sen sisältö on useimmiten määrällistä tietoa. SQL-tietokanta on yksi esimerkki strukturoidusta tiedosta. Siihen voidaan tallentaa vain tietokannan rakenteen määrittelevän mallin mukaisesti, ja siihen voidaan tehdä helposti hakuja ja tiedonkäsittelyä.

Strukturoimaton tieto on vastaavasti rakenteeltaan hyvin vaihtelevaa (Smallcombe, 2023). Sitä on hankala käsitellä ja siitä on vaikea hakea tietoa monesta eri kohteesta. Strukturoimatonta tietoa varastoidaan yleensä alkuperäisessä muodossa, ja sen hyödyntäminen vaatii tiedon esikäsittelyä, jolloin tiedosta saadaan kerättyä normalisoitua arvoja. Esimerkkejä strukturoimattomasta tiedosta ovat esimerkiksi kuvat, videot ja useimmat tekstit. Tieto on useimmiten laadullista ja etukäteen tuntematonta.

Näiden kahden välissä on myös kolmas luokka: osittain strukturoitu tieto (Smallcombe, 2023). Useimmat tässä työssä käsitellyt datamuodot ovat tätä luokkaa, kuten esimerkiksi JSON, CSV ja XML. Tieto voidaan kirjoittaa strukturoidusti myös näissä datamuodoissa, mutta koska tietorakenne on vapaamuotoinen, mikään ei kuitenkaan pakota käyttämään tiettyä mallia.

Strukturoituun tietoon liittyy seuraavia ominaisuuksia (TIBCO, i.a.):

- Tieto sisältää tunnistettavia rakenteita ja noudattaa rakenteen tietomallia.
- Tieto on jäsennelty rivi- ja sarakemuotoon.
- Data on jäsennelty ja muotoiltu siten, että sen voi tulkita vain yhdellä tavalla.
- Tieto on sijoitettu kiinteisiin kenttiin tiedostoissa tai muissa rakenteissa.
- Samantyyppinen tieto on ryhmitelty yhteen.
- Tietopisteillä samassa ryhmässä on samat ominaisuudet ja kentät.
- Ihminen tai ohjelma voi lukea tietoa helposti.
- Rakenteen elementteihin voidaan tehdä suoria viittauksia.

Yksi mahdollisuus on käyttää esimerkiksi Eclipse Sparkplug -standardia, joka on MQTT-protokollaan sidottu määritelmä siitä, miten järjestelmien tulisi vaihtaa tietoa keskenään (Eclipse, 2022). Sparkplug-määritelmä osoittaa käytettävät MQTT-otsikot, joihin tietoa täytyy lähettää

ja joista sitä voidaan ottaa vastaan. Viestien sisältö on Google Protocol Buffers -kirjaston avulla kirjoitettua binäärimuotoista dataa. Protocol Buffers tarjoaa työkalut, millä datan skeema täytyy määritellä, jolloin lähettäjän ja vastaanottajan täytyy lukea ja kirjoittaa tietoa tämän skeeman mukaisesti.

Tässä työssä on annettu esimerkkejä muutaman valikoidun tiedostomuodon käyttämisestä telemetriatiedon paketointia varten. Tiedostomuotojen suunnittelussa on haettu rakennetta, jossa tieto voidaan esittää avain–arvopareina. Tyhjiä kenttiä ei kirjoiteta tietorakenteeseen mukaan.

4.1 JSON

JavaScript Object Notation (JSON) on tekstipohjainen muoto osittain strukturoidun tekstin käsittelyyn missä tahansa ohjelmointikielessä (Ecma International, 2017, s. 5). JSON tukee monia samankaltaisia rakenteita, jotka löytyvät monista ohjelmointikielistä, mutta useimmiten hiukan eri tavalla kirjoitettuna. JSON määrittelee yleisen rakenteen, jota voidaan käyttää tiedon siirtämiseen kahden ohjelman välillä käyttäen yhdenmukaista muotoa.

JSON-dokumentin perustyyppejä ovat: objekti, lista, teksti, numero, boolean-arvo tai määrittelemätön (Ecma International, 2017, s. 10). Objekti on keskeisin elementti, jossa tieto määritellään avain–arvopareina. Objekti alkaa aaltosulun aloitusmerkillä ({) ja päättyy aaltosulun lopetusmerkkiin (}). Merkkien välissä on kaksoispisteellä (:) erotettuna avain–arvoparit. Avaimet ovat aina tekstimuotoisia, mutta arvo voi olla mikä tahansa perustietotyyppi. Lista voi koostua vain perustietotyypeistä, mutta kaikkien listan alkioiden ei tarvitse olla samaa tyyppiä. Lista alkaa hakasulun aloitusmerkillä ([) ja päättyy hakasulun lopetusmerkkiin (]). Listan sisällön ei tarvitse olla järjestetty (mts. 11). Kuva 10 antaa esimerkin JSON-rakenteesta, josta on käytetty kaikkia tuettuja perustietotyyppejä.

```
{
  "key": "text",
  "number": 0,
  "boolean": true,
  "undefined": null,
  "array": [
    "text",
    0,
    true
  ],
  "object": {
    "key": "value"
  }
}
```

Kuva 10. Esimerkki JSON-datamuodosta.

4.2 XML

Extensible Markup Language (XML) on hierarkkinen dokumenttimalli osittain strukturoidun tekstimuotoisen tiedon tallentamiseen, esittämiseen ja tiedon vaihtamiseen (W3C, 2016). XML-dokumenttia käytetään useasti tiedon siirtämiseen internetissä ja esimerkiksi HTML-sivut ovat rakenteeltaan XML-dokumentin kaltaisia, mutta eivät kuitenkaan aivan sama asia (Mozilla, 2022). HTML-dokumentissa on mahdollista käyttää vain ennalta määriteltyjä tietorakenteita, kun taas XML-dokumentissa tietorakenteet täytyy määritellä itse.

XML-dokumentti on puurakenne, joka rakentuu loogisista elementeistä, joissa yksi elementti muodostuu alkavasta tagista ja päättävästä tagista (Mozilla, 2022). Tagi on vapaavalintainen tekstitunniste kentälle, joka on muotoiltu pienempi kuin (<) ja suurempi kuin (>) merkkien väliin alkavassa tagissa. Lopettava tagi alkaa merkkien pienempi kuin ja kauttaviiva (</) yhdistelmällä ja päättyy suurempi kuin (>) merkkiin. Aloittavan ja lopettavan tagin välissä voi olla lapsielementtejä tai vapaamuotoista tekstiä. Tagiin vodaan liittää attribuutteja avain–arvopareina. Lisäksi dokumenteissa voi olla sen käsittelyä ohjaavia metatietoja. Kuva 11 antaa esimerkin yksinkertaisesta XML-rakenteesta. Dokumentti alkaa ohjaustiedolla ja sisältää kaksi tagia, joilla molemmilla on yksi jäsentagi.

```
<?xml version="1.0" encoding="UTF-8"?>
<tag attribute="text">
    <child>Text data</child>
</tag>
<anothertag>
  <child attribute=0 />
</anothertag>
```

Kuva 11. Esimerkki XML-dokumentista.

4.3 BSON

BSON on terminä muunnos sanoista ”binary JSON” eli binäärimuotoinen JSON dokumentti (BSON, i.a.). BSON-rakenne on saman kaltainen JSON-dokumentin kanssa, mutta se on kirjoitettu binäärimuotoisena tekstimuotoisen datan sijaan. BSON-dokumenttia ei voi siis lukea suoraan, vaan siihen tarvitaan oma työkalunsa. Binäärimuotoisena kirjoitetulla dokumentilla haetaan tehokkuutta datan säilyttämiseen tiedostomuodossa. JSON dokumenttiin verrattuna BSON tukee samoja rakenteita, mutta tukee laajempaa valikoimaa tietotyyppejä esimerkiksi päivämäärän ja binäärimuotoisten alidokumenttien käyttämiseen.

4.4 CBOR

Concise Binary Object Representation (CBOR) on binäärimuotoinen osittain strukturoidun tiedon talletusmuoto (Bormann & Hoffman, 2020). Rakenne on JSON-dokumentin kaltainen ja binäärimuoto poikkeaa BSON-muotoisesta dokumentista. CBOR pohjautuu vanhempaan MessagePack formaattiin, ja sen tavoitteena on kirjoittaa data tehokkaasti tiedostoon tai viestiin. Formaatti onkin suunniteltu helpoksi käsitellä myös laitteissa, joissa on niukasti resursseja tiedostojen käsittelyyn. CBOR on osa CoAP (Constrained Application Protocol) protokollaa, joka on tarkoitettu todella pieniin laitteisiin, joissa vaaditaan kevyttä ja tahokasta tapaa siirtää tietoa.

4.5 Apache AVRO

Apache AVRO on binäärimuotoinen datan tallennusformaatti, ja se on laajalti käytössä big data -sovelluksissa (Estrada, 2018, s. 95). Se tukee monia erilaisia rakenteita tiedon

tallentamiseen. Tiedoston sisältö kuvataan tiedoston liitteenä olevassa skeemassa, joka on JSON muotoinen kuvaus tiedon rakenteesta. Skeema voidaan hakea myös ulkoisesta rekisteristä, jossa ylläpidetään listaa skeemojen eri versioista. JSON-muotoisessa skeemassa voidaan kuvata myös monimutkaisempia rakenteita kuin yksittäisten kenttien tietotyyppi (mts. 96).

AVRO tukee seuraavia yksinkertaisia tietotyyppejä (Apache, i.a.):

- null, ei arvoa
- boolean, binääriarvo
- int, 32-bittinen etumerkillinen kokonaislukutyyppi
- long, 64-bittinen etumerkillinen kokonaislukutyyppi
- float, 32-bittinen liukuluku
- double, 64-bittinen liukuluku
- bytes, sarja 8-bittisiä tavuja
- string, unicode koodattu merkkijono

Skeemaan on mahdollista määritellä myös monimutkaisempia tyyppejä, kuten lista arvoista, avain–arvopari tai arvojoukko (enumeration) (Apache, i.a.). Koska tiedoston sisältö perustuu skeemaan, on sen tilan käyttö hyvin tiivis ja sisältää vain hyötydataa. Skeemaa vasten voidaan myös tarkistaa tiedoston rakenteen eheys, ja tiedosto voidaan lukea ilman, että sen sisäistä rakennetta tunnetaan etukäteen. Kuva 12 antaa esimerkin AVRO-skeemasta JSON-muodossa.

```
{
 "name": "HealthCheck",
 "namespace": "kioto.avro",
 "type": "record",
 "fields": [
 { "name": "event", "type": "string" },
 { "name": "factory", "type": "string" },
 { "name": "serialNumber", "type": "string" },
 { "name": "type", "type": "string" },
 { "name": "status", "type": "string"},
 { "name": "lastStartedAt", "type": "long", "logicalType": "timestamp-
    millis"},
 { "name": "temperature", "type": "float" },
 { "name": "ipAddress", "type": "string" }
 ]
}
```

Kuva 12. Esimerkki Apache AVRO -tiedoston skeemasta (Estrada, 2018).

4.6 CSV

Comma-Separated Values (myös CSV) on hyvin yksinkertainen tapa kirjoittaa riveistä ja sarakkeista muodostuvaa strukturoitua tietoa (Shafranovich, 2005). Nimi kääntyy vapaasti suomeksi ”pilkulla erotetut arvot”. Nimensä mukaisesti arvot kirjoitetaan tekstimuotoisena riville, jossa sarakkeet erotetaan käyttämällä pilkkua (,) erotinmerkkinä. Myös muita merkkejä voidaan käyttää erotinmerkkinä. Yksi rivi loppuu Windows-muotoiseen rivin vaihdon merkkiin (CR+LF), jonka jälkeen alkaa uusi rivi. CSV ei erottele kenttien tietotyyppejä, ja kaikki kentät ovat tekstiä. Tietotyypin tarkistus ja käsittely tehdään tietoa lukevan sovelluksen toimesta.

4.7 SQLite

SQLite on maailman käytetyin SQL-kielellä (Structured Query Language) toimiva tietokanta (Donahoo & Speegle, 2005, s. 3). Tietokannan käyttö on sulautettu osaksi oman ohjelman toimintaa ja se ei vaadi ulkoista prosessia tietokannan käyttöä varten. Tietokantatiedosto voidaan siirtää ja käsitellä missä tahansa muussa järjestelmässä, joka tukee SQLite-tietokantoja (SQLite, 2021).

SQL-tietokanta rakentuu tauluista (table), joihin tieto kirjataan (Donahoo & Speegle, 2005, s. 11). Taulussa data on jaettu riveihin ja sarakkeisiin. Sarakkeille voidaan määritellä tietotyyppi,

joka määrittelee, minkä muotoista tietoa kenttään on mahdollista sijoittaa. Tietokannan taulut voidaan liittää toisiinsa käyttäen viiteavaimia. Tauluun, jossa on pääavaimeksi määritelty sarake, voidaan tehdä viittaus toisesta taulusta viiteavaimella. Taulun välisiä viittaustapoja voi olla useita erilaisia: yhden suhde yhteen, yhden suhde moneen tai monen suhde moneen. Tietokannan sisältämistä tauluista ja taulujen välisistä viittauksista voidaan kuvata tietokannan skeema, eli miten tietokanta on rakennettu ja miten tieto on siihen sijoitettu.

4.8 Telemetriatiedon pakkaaminen

Tiedon pakkaamisessa pyritään esittämään sama tieto pienemmällä määrällä bittejä (Pu, 2006, s. 3). Tiedosta pyritään etsimään toistuvuuksia, jotka voidaan kirjoittaa uudestaan käyttämällä matemaattista mallia, millä tieto voidaan esittää niin, että se vaatii vähemmän levytilaa. Lähdetiedon käsittelijää kutsutaan pakkaajaksi ja vastaavasti käsittelijää, joka muuntaa pakatun tiedon takaisin alkuperäiseen muotoon, tiedon purkajaksi. Pakkausmenetelmiä on häviöttömiä ja häviöllisiä (mts. 5). Häviöttömässä tavassa tieto säilyy muuttumattomana ja se vastaa purkamisen jälkeen täysin alkuperäistä, ennen pakkausta ollutta tietoa. Häviöllisessä pakkaustavassa lähdetiedosta pyritään poistamaan tietoa, niin ettei se vaikuta kovin merkittävästi pakattuun tietoon. Pakkauksessa tehdään peruuttamattomia muutoksia, joita ei voi palauttaa enää purkamisvaiheessa. Häviöllinen pakkausmenetelmä soveltuu lähinnä multimedian, kuten kuvan, äänen ja videon pakkaamiseen. Telemetriatiedon pakkauksessa halutaan säilyttää kaikki lähetettävä tieto, joten ainoa vaihtoehto on käyttää häviötöntä pakkausmenetelmää.

Pakkausmenetelmän laatua voidaan mitata häviöttömässä pakkauksessa tarkastelemalla tiedostokokoa ennen ja jälkeen pakkauksen (Pu, 2006, s. 11). Pakkauksen jälkeinen tiedostokoko jaettuna alkuperäisellä tiedostokoolla antaa pakkaussuhteen, jossa pienempi arvo kertoo paremmasta pakkaussuhteesta. Pakkauskerroin on käänteinen arvo pakkaussuhteesta, ja siinä tiedoston koko ennen pakkausta jaetaan tiedostokoolla pakkauksen jälkeen. Isompi pakkauskerroin on parempi. Pakkauskerroin kertoo, miten paljon enemmän tietoa mahtuu pakatussa muodossa samaan tilaan kuin alkuperäisen tiedon vaatima tiedostokoko. Jossain erikoistapauksissa myös pakattu tieto voi olla tiedostokooltaan suurempi kuin lähdetieto.

Pakkausalgoritmeja on erilaisia ja erilaisiin sovelluksiin kehitettyjä (Pu, 2006, s. 12). Pakkauksen tehokkuus voi vaihdella eri algoritmeilla riippuen lähdedatan rakenteesta. Lisäksi

algoritmin soveltuvuuteen vaikuttaa sen monimutkaisuus ja sitä kautta pakkauksen ja purkamisen vaatima suoritinaika. Joissain sovelluksissa pakkausajalla ei ole merkitystä, mutta purkamisnopeuden täytyy olla mahdollisimman hyvä.

GNU Zip (gzip) on Lempel-Ziv (LZ77) algoritmiin perustuva pakkaustyökalu (Gailly, i. a.). Sitä käytetään monissa Linux- ja BSD-pohjaisissa käyttöjärjestelmissä ja se on julkaistu GPL-lisenssin alaisena avoimen lähdekoodin ohjelmana.

bzip2 on pakkaustyökalu, joka käyttää Burrows-Wheelerin järjestettyjen lohkojen tekstinpakkausalgoritmia ja Huffman-koodausta (Seward, i. a.). Pakkaustehokkuutta pidetään yleisesti parempana kuin LZ77/LZ78-pohjaisissa työkaluissa (esimerkiksi gzip). Pakkaustehokkuus riippuu käytettävästä lohkon koosta. Suuremmalla lohkon koolla saavutetaan parempi pakkaustehokkuus, mutta tämä vaatii enemmän RAM-muistia pakkaus- ja purkamisvaiheissa. Pakkauksen ja purkamisen nopeus pysyy kuitenkin lähes samana riippumatta käytettävästä lohkon koosta. Suurempaan lohkokokoon siirryttäessä hyödyt muuttuvat pienemmiksi, joten optimaalinen arvo täytyy määrittää tapauskohtaisesti riippuen käytettävistä resursseista.

Yann Collet on luonut kaksi hyvin samansukuista pakkausalgoritmia: LZ4 ja Zstandard (zstd). LZ4 on vanhempi algoritmi, jonka tavoitteena on ollut nopea tiedon pakkaaminen ja erittäin nopea tiedon purkaminen (Collet, 2022). Se kuuluu Lempel-Ziv (LZ77) sukuisiin algoritmeihin. Facebook julkaisi vuonna 2016 vapaaseen levitykseen Colletin toisen pakkaustyökalun nimeltä Zstandard (Simone, 2016). Siinä on samankaltaiset ominaisuudet kuin LZ4:ssa, mutta pidemmälle kehitettyinä. Pakkausalgoritmi pohjautuu ”finite state entropy” -malliin ja optimoituun huffman-koodaukseen. Toisin kuin LZ4, Zstd algoritmi ja tiedostorakenne on standaroitu IETF RFC 8878 -standardissa (Collet, 2021).

Brotli on Googlen kehittämä yleiskäyttöinen algoritmi erityisesti web-sisällön pakkaamiseen (Alakuijala & Szabadka, 2016). Se käyttää LZ77-sukuista pakkausalgoritmia ja huffman-koodausta. Tavoitteena on ollut gzip-pakkausta parempi pakkaustehokkuus nopeammalla tiedon käsittelyllä. Pakkausalgoritmi ja tiedostomuoto on standardoitu IETF RFC 7932 -standardissa.

XZ on Lempel-Ziv-Markov (LZMA) algoritmia käyttävä pakkaustyökalu (xz: Man Page, 2022). Sillä päästään hyvään pakkaussuhteeseen, mutta pakkaustoimenpide on hidas ja purkaminenkin melko hidasta. Se soveltuu parhaiten käyttötapauksiin, missä pakattua tietoa on

tarvetta purkaa useasti esimerkiksi silloin, kun jaetaan isoja tiedostoja usealle käyttäjälle internetin välityksellä.

5 TELEMETRIALÄHETYKSEN SIMULOINTI

Tässä luvussa tutkitaan, miten aikasarjamuotoinen strukturoitu tieto kannattaa paketoida ja siirtää laitteelta eteenpäin. Simuloinnissa käytettään AEGIS Big Data Project -hankkeessa ajoneuvoista kerättyä aikasarjadataa (Stocker ym., 2017). Data sisältää aikaleimaan sidottuja mittausarvoja kiihtyvyysanturilta, paikkatietolaitteelta ja auton moottorinohjauslaitteistolta OBD-liittimen kautta kerättynä. Alkuperäinen data on kerätty useasta ajokerrasta ja se on jäsennelty omiin lähdetiedostoihin. Tieto on kerätty siten, että yhdestä lähteestä otetut näytteet on sidottu samaan aikaleimaan pois lukien OBD-mittauspisteet, joilla on oma aikaleima. Datalähteiden talletusväli vaihtelee lähteen mukaan. Esimerkiksi kiihtyvyystieto on kerätty 10 kertaa sekunnissa ja paikkatieto sekä OBD-tieto kerran sekunnissa.

Tietoa on käsitelty siten, että kaikki kerätyt datapisteet voidaan esittää samassa taulukossa. Simulointitestit on ajettu käyttäen mittaustietoa, joka on kerätty lähdedatan mukaan mittauskerrasta numero 6. Tämä otanta koostuu 38085 aikaleimasta (rivistä) ja 356290 mittauspisteestä. Laajennettu otanta käytetystä esimerkkidatasta on liitteessä 1 ja typistetty osaotanta taulukossa 1.

Taulukko 1. Karsittu otanta (vain kiihtyvyystiedot) käytetystä esimerkkidatasta.

Aikaleima	acc.x	acc.y	acc.z	gyro.x	gyro.y	gyro.z
1484885573.80952				-3.47826	-1.66957	-8.69565
1484885573.8112	0.22266	0.24609	1.07812			
1484885573.8513				-2.92174	0.90435	-8.69565
1484885573.8531	0.23828	0.20313	1.05859			
1484885573.89317				-2.22609	0.97391	-8.69565
1484885573.89487	0.24219	0.20313	1.03906			

5.1 Telemetriakehysten määrittely

Telemetrian simulointia varten määriteltiin mittaussarjan paketointimuotoja. Paketointimuoto on määritelmä siitä, miten aikasarjadata täytyy kirjoittaa tiedostoon tai viestiin, jonka laite lähettää telemetriana eteenpäin. Esimerkiksi miten JSON-dokumentti täytyy muotoilla, kun siihen kirjoitetaan valitut mittapisteet.

Termillä signaali viitataan yksittäiseen mittauspisteeseen. Se toimii tunnisteena tiedolle, joka on kerätty mittauspisteestä. Esimerkiksi lämpötila-anturin arvon kerääminen kerran sekunnissa. Lämpötila-anturi on tässä signaalin tunniste arvoille, jotka datalähteestä on kerätty.

Paketointimuodot on suunniteltu siten, että vain hyödyllinen tieto voidaan kirjoittaa mukaan. Esimerkiksi jos paketissa on useampi signaali (mittaussarja) ja monta aikaleimaa, mutta kaikkiin signaaleihin ei ole arvoa kyseisellä ajankohdalla, signaalin arvoa ei silloin kirjoiteta mukaan aikaleiman yhteyteen.

Tiedon aikaleimana käytettään 64-bittistä liukulukua, joka lasketaan sekunteina ajankohdasta 1.1.1970 00:00.00. Tämä tunnetaan myös UNIX-aikaleimana, jossa on mukana myös millisekunnit. Aikaleiman tarkkuus on yksi millisekunti. Mittauspisteiden muoto on 64-bittinen liukuluku, 32-bittinen kokonaisluku tai 64-bittinen kokonaisluku. Simuloinnin lähdedatassa kiihtyvyysanturien tiedot ja paikkatieto ovat liukulukuja ja OBD-datapisteet kokonaislukuja.

Tässä työssä esitellyt telemetriatiedon lähetysmuodot eivät vastaa reaalimaailman käyttötapausta, koska niissä on vain hyötydataa. Todellisessa käyttötapauksessa mukana kulkee enemmän metatietoa, jota tarvitaan esimerkiksi laitteen tunnistamiseen. Metatietoa voidaan kuljettaa myös hyötydatan ulkopuolella, eikä sitä ole aina välttämättä pakko liittää telemetriatiedon mukaan.

5.2 JSON-muotoinen telemetria

JSON-muodosta määriteltiin 2 erilaista mallia. Ensimmäisessä mallissa käytössä on kiinteät avaimet, jotka on lyhennetty yhden tai kahden kirjaimen mittaisiksi tilan säästämiseksi.

Ylätasolla on listatyyppinen rakenne, jonka sisällä on objekteja. Objektin avaimina on aikaleima (”ts”) ja lista datapisteistä (”d”). Yksittäinen datapiste on kuvattu objektina, jossa on 2 avainta: signaalin tunniste (”k”) ja arvo (”v”). Kuva 13 havainnollistaa testidatan kirjoittamista JSON-muotoon.

```
[
  {
    "ts": 1484885573.056958,
    "d": [
      {
        "k": "gyro.x",
        "v": -1.8087
      },
      {
        "k": "gyro.y",
        "v": 1.04348
      },
      {
        "k": "gyro.z",
        "v": -9.6
      }
    ]
  },
  {
    "ts": 1484885573.057703,
    "d": [
      {
        "k": "acc.x",
        "v": 0.0
      },
      {
        "k": "acc.y",
        "v": 0.0
      },
      {
        "k": "acc.z",
        "v": 0.0
      }
    ]
  }
]
```


Kuva 13. Esimerkki aikasarjadatan kirjoittamisesta JSON-rakenteeksi.

Tieto voidaan organisoida myös siten, että ylätason objektina toimii signaalin tunniste (”k”) ja sen alle on kerättynä mittapisteiden aikaleimat ja arvot. Näin aikaleima voi toistua useaan kertaan sen sijaan, että signaalin tunniste toistuu useasti.

Toisessa mallissa käytössä ei ole kiinteitä avaimia objekteissa. Objektien avaimet on korvattu tekstikentillä, jotka sisältävät myös sisältöön liittyvää tietoa. Tietomuotoa lukevan sovelluksen täytyy olla siis tietoinen siitä, miten dokumentti on rakennettu, että se kykenee lukemaan sen oikein. Kuva 14 havainnollistaa tämän rakenteen käyttöä.

```
{
  "1484885573.056958": {
    "gyro.x": -1.8087,
    "gyro.y": 1.04348,
    "gyro.z": -9.6
  },
  "1484885573.057703": {
    "acc.x": 0.0,
    "acc.y": 0.0,
    "acc.z": 0.0
  },
  "1484885573.07438": {
    "obd.05": 16
  },
  "1484885573.099399": {
    "gyro.x": -1.87826,
    "gyro.y": -0.76522,
    "gyro.z": -9.94783
  },
  "1484885573.100571": {
    "acc.x": 0.24609,
    "acc.y": 0.34375,
    "acc.z": 1.0625
  }
}
```


Kuva 14. Esimerkki aikasarjadatan kirjoittamisesta vaihtoehtoiseen JSON-rakenteeseen.

Ylätason rakenne on objekti, jonka avaimena on tietoon liittyvä aikaleima. Aikaleiman arvona on objekti, missä signaalin nimi toimii avaimena ja signaalin mittaustieto arvona. Tähän rakenteeseen viitataan testituloksissa tunnisteella ”JSON KV”.

Tämän mallin ongelma on kuitenkin, että muun muassa Go JSON -kirjasto, Rust Serde -kirjasto, C# JSON.net -kirjasto ja Java Jackson -kirjasto perustuvat malliin, missä JSON-dokumentti luodaan suoraan ohjelmointikielen tietorakenteiden kuvauksista. Yleensä ohjelmassa luodaan luokka tai struktuuri, mikä on mahdollista sarjallistaa suoraan JSON-muotoon. Tällöin tarvitaan kiinteitä avaimia JSON-dokumentissa, joten avainkenttien käyttäminen hyötydatalle ei ole mahdollista. Yleensä kaikissa ohjelmointikielissä voidaan kuitenkin määritellä myös sarjallistaminen siten, että tiedon rakennetta ei tarvitse tuntea etukäteen. Tästä syntyy yleensä dynaamisesti luotu tietorakenne, joka kuvaa tiedon sisältöä ja rakennetta. Esimerkiksi Pythonin JSON-kirjasto tuottaa dictionary-tyyppisen tietorakenteen luetusta tiedosta.

5.3 XML-muotoinen telemetria

XML-rakenteen alussa on annettu ohje dokumentin lukijalle, että kyseessä on XML-dokumentin 1.0 versio ja se käyttää UTF-8-merkistöä. Ylätasolla on tagi ”d”, jonka alla on elementti ”m”, jonka attribuuttina on aikaleima mittauksen ajanhetkeen. Tämän elementin alapuolella on yksi tai useampi ”s” elementti, joka sisältää mitatun signaalin tunnisteen attribuuttikentässä ja signaalin arvon elementin tietokentässä. Tagien tunnisteet on valittu mahdollisimman lyhyiksi tilan säästämiseksi. Kuva 15 havainnollistaa testidatan kirjoittamisen XML-muotoon.

```
<?xml version='1.0' encoding='utf-8'?>
<d>
    <m ts="1484885573.056958">
        <s n="gyro.x">-1.8087</s>
        <s n="gyro.y">1.04348</s>
        <s n="gyro.z">-9.6</s>
    </m>
    <m ts="1484885573.057703">
        <s n="acc.x">0.0</s>
        <s n="acc.y">0.0</s>
        <s n="acc.z">0.0</s>
    </m>
    <m ts="1484885573.07438">
        <s n="obd.05">16</s>
    </m>
    <m ts="1484885573.099399">
        <s n="gyro.x">-1.87826</s>
        <s n="gyro.y">-0.76522</s>
        <s n="gyro.z">-9.94783</s>
    </m>
    <m ts="1484885573.100571">
        <s n="acc.x">0.24609</s>
        <s n="acc.y">0.34375</s>
        <s n="acc.z">1.0625</s>
    </m>
</d>
```

Kuva 15. Esimerkki aikasarjadatan tallentamisesta XML-dokumentiksi.

5.4 CSV-muotoinen telemetria

CSV-muodossa ensimmäisellä rivillä on sarakkeiden otsikot eli aikaleima ja signaalien tunnisteet. Sarakkeet on erotettu pilkulla (,) ja rivi päättyy Windows-tyylisen uuden rivin merkkiin

(CR+LF). Sarakkeiden arvot ovat seuraavilla riveillä vastaavan aikaleiman rivillä. Kuva 16 havainnollistaa testidatan kirjoittamista CSV-muotoon.

```
ts,acc.x,acc.y,acc.z,gyro.x,gyro.y,gyro.z,latitude,longitude,altitude,obd.04,obd.05,obd.0B,obd.0C,obd.0D,obd.0F,obd.10,obd.11,obd.33,obd.3c
1484885573.056958,,,,-1.8087,1.04348,-9.6,,,,,,,,,,,,
1484885573.057703,0.0,0.0,0.0,,,,,,,,,,,,,,,
1484885573.07438,,,,,,,,,,16,,,,,,,,
1484885573.099399,,,,-1.87826,-0.76522,-9.94783,,,,,,,,,,,,
1484885573.100571,0.24609,0.34375,1.0625,,,,,,,,,,,,,,,
1484885573.103495,,,,,,,,,,,122,,,,,,,
1484885573.123402,,,,,,,,,,,,1227,,,,,,
1484885573.141177,,,,-2.50435,1.66957,-10.4348,,,,,,,,,,,,
1484885573.14289,0.30469,0.30078,1.09766,,,,,,,,,,,,,,,
```

Kuva 16. Esimerkki aikasarjadatasta CSV-muodossa.

5.5 BSON-muotoinen telemetria

BSON-rakenne on hyvin lähellä JSON-tyyppistä rakennetta. BSON sarjallistetaan binäärisessä muodossa tekstimuodon sijaan. BSON ei tue listatyyppistä rakennetta dokumentin juuressa, joten käytössä on objekti, jonka sisällä käytetään arvon avaimena juoksevaa numeroa. Tämän avaimen alla on varsinainen telemetria omana objektina, jonka avain ”ts” kertoo aikaleiman ja ”d” listan sisällytetyistä signaaleista aikaleiman osoittamalla hetkellä. Jokainen signaali on muotoiltu objektiksi, jonka avain ”k” kertoo signaalin nimen ja avain ”v” signaalin arvon. Kuva 17 antaa esimerkin dokumentin sisällöstä tekstimuotoon muutettuna.

```
{
  "59999": {
    "ts": 1662583791.568,
      "d": [
        {
          "k": "sig3",
          "v": 412
        }
      ]
  }
}
```

Kuva 17. Esimerkki aikasarjadatan paketoinnista BSON-muotoon.

Myös BSON-rakenteesta määriteltiin vaihtoehtoinen rakenne, missä ei ole käytössä kiinteitä avaimia. Myös avaimina toimivat kentät on täytetty hyötydatalla. Tässä rakenteessa dokumentin juuressa on edelleen objekti, jonka avaimena on tekstimuotoinen esitys aikaleimasta.

Aikaleiman arvona on objekti, jonka avain–arvopareina on signaalin tunniste ja arvo kyseessä olevan ajan hetkellä. Kuva 18 havainnollistaa tätä rakennetta.

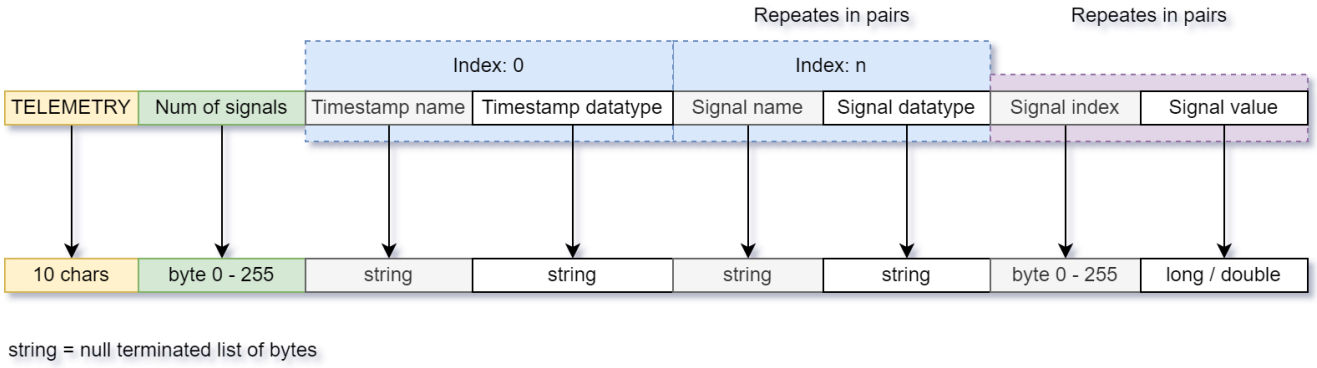
```
{
  "1662582707.422": {
    "sig3": 740
  }
}
```

Kuva 18. Esimerkki aikasarjadatan paketoinnista BSON-muotoon ilman kiinteitä avaimia (tekstimuotoisena esitettynä).

Myös tätä vaihtoehtoista rakennetta ei voida käyttää suoraan kaikissa ohjelmointiympäristöissä. BSON-tuen toteuttava kirjasto määrittelee, täytyykö dokumentin sisältää kiinteitä avaimia, että se voidaan lukea tietorakenteiksi.

5.6 Räätälöity binäärimuoto

Yksi telemetrian paketointimuoto on itse määritelty binäärirakenne. Tässä rakenne koostuu kahdesta osasta: otsikkotiedoista, joissa kuvataan sisältö, ja itse datasta avain–arvopareina. Kuva 19 havainnollistaa tietorakenteen sisältöä.



Kuva 19. Kuvaus binäärisen telemetriarakenteen sisällöstä.

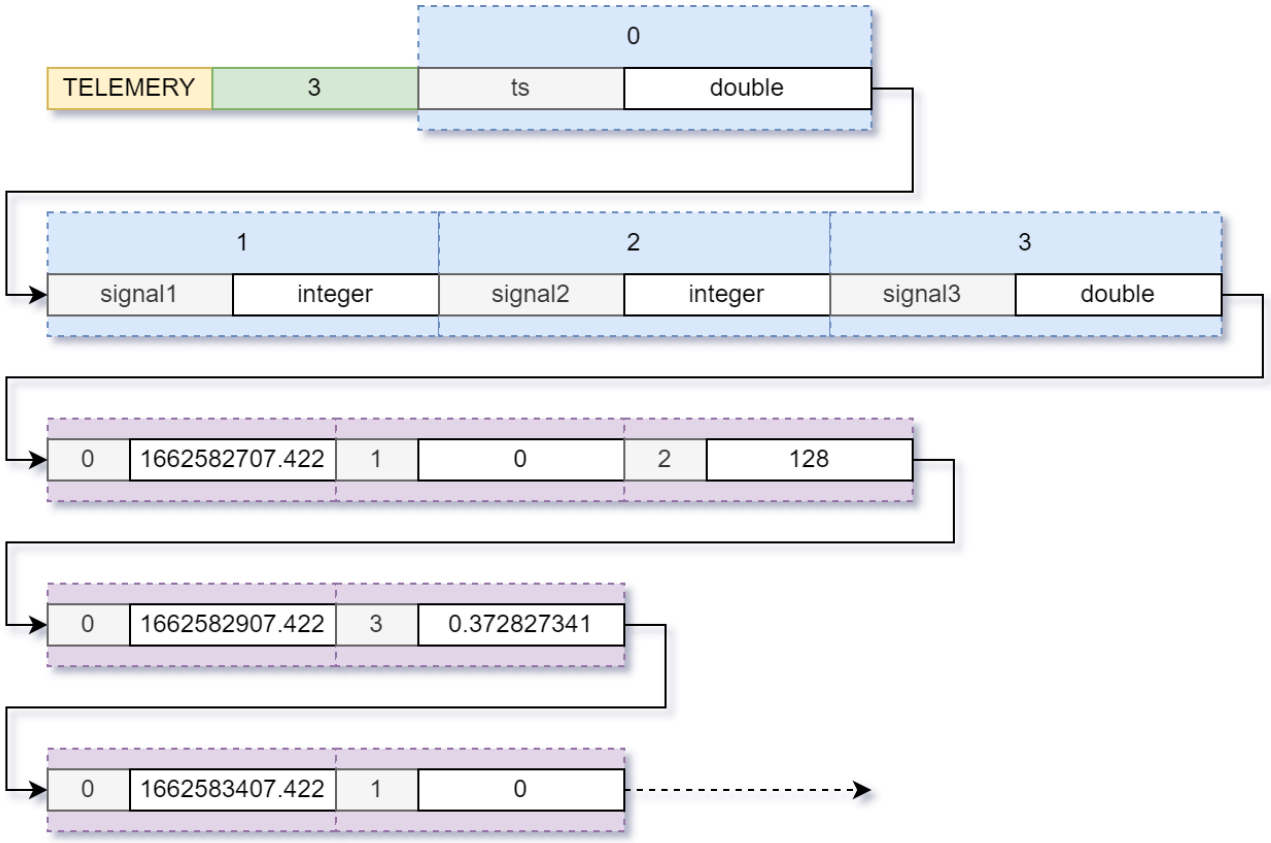
Tiedosto alkaa tunnisteella, jossa on teksti ”TELEMETRY” ja päättyy merkkiin ”\0”. Kaikki tässä muodossa käytetyt tekstikentät päättyvät ilman erillistä mainintaa merkkiin ”\0”. Numerot ovat least significant byte -muodossa (LSB). Tämän jälkeen tulee numero 8-bittisenä

etumerkittömänä kokonaislukuarvona, joka ilmaisee tiedostossa olevien signaalitunnisteiden määrän.

Seuraavaksi toistuu kaksi kenttää, jotka kuvaavat yksittäisen signaalin rakenteen. Ensimmäisenä on tekstimuotoinen signaalin tunniste ja tämän jälkeen signaalin tietotyypin tunniste. Tunniste voi olla yksi seuraavista: ”integer”, ”double” tai ”string”. ”Interger” on 32-bittinen kokonaisluku, ”double” 64-bittinen liukuluku ja ”string” kenttä on tekstikenttä, joka päättyy ”\0” merkkiin.

Ensimmäisenä parina on aina aikaleiman tunniste ja tietotyyppi. Seuraavana toistuu sama rakenne niin monta kertaa kuin on ilmoitettu signaalien lukumäärän kertovassa kentässä. Lukijan täytyy määrittää jokaiselle otsikkokentässä olevalle tunnisteelle indeksinumero. Numero lasketaan juoksevana kokonaislukuna alkaen nollasta (0) ja päättyen arvoon 255. Jokainen uusi signaali kasvattaa indeksiä arvolla 1, ja yhdessä tiedostossa voi olla korkeinaan 256 erilaista signaalia.

Näiden kenttien jälkeen alkaa varsinainen data. Signaalien arvot ovat samaan tyyliin kirjotettu pareina, joissa ensimmäisenä on signaalin indeksinumero ja heti perään arvo metadatan ilmoittamassa muodossa. Kun tiedostossa tulee aikaleimaksi tunnistettu arvo vastaan, kaikki signaalit käyttävät tätä aikaleimaa ennen seuraavan aikaleiman löytymistä. Kuva 20 antaa esimerkin tiedoston lukemisesta.



Kuva 20. Esimerkkidatan lukeminen binäärimuodosta.

5.7 CBOR-muotoinen telemetria

CBOR-muotoisessa datassa käytetään samaa rakennetta kuin aikaisemmin esitellyissä luvuissa JSON-muotoinen telemetria (5.2) ja .

BSON-muotoinen telemetria (5.5). Ainoa ero on, että tieto on kirjoitettu levylle CBOR-standardin määrittelemään muotoon.

5.8 AVRO-muotoinen telemetria

Tähän formaattiin määriteltiin skeema (kuva 21), jossa tieto on jäsennelty ”record” kenttiin, jotka sisältävät aikaleiman ja mittauspisteiden tiedot avain–arvopareina. Aikaleima on määritelty loogiseksi aikaleimatyypiksi, jonka arvo on kirjoitettu millisekuntitarkkuudella long-tyyppisenä binääriarvona. AVRO tukee assosiatiivista listaa eli parilistaa, jota käytetään signaalien arvojen tallentamiseen joko long- tai double-tyyppisenä binääriarvona.

```
{
    'doc': 'telemetry data',
    'name': 'telemetry',
    'type': 'record',
    'fields': [
        {'name': 'timestamp', 'type': {'logicalType': 'timestamp-millis', 'type': 'long'}},
        {'name': 'values', 'type': {'type': 'map', 'values': ['long', 'double']}}
    ]
}
```

Kuva 21. AVRO-tiedostomuodossa käytetty skeema.

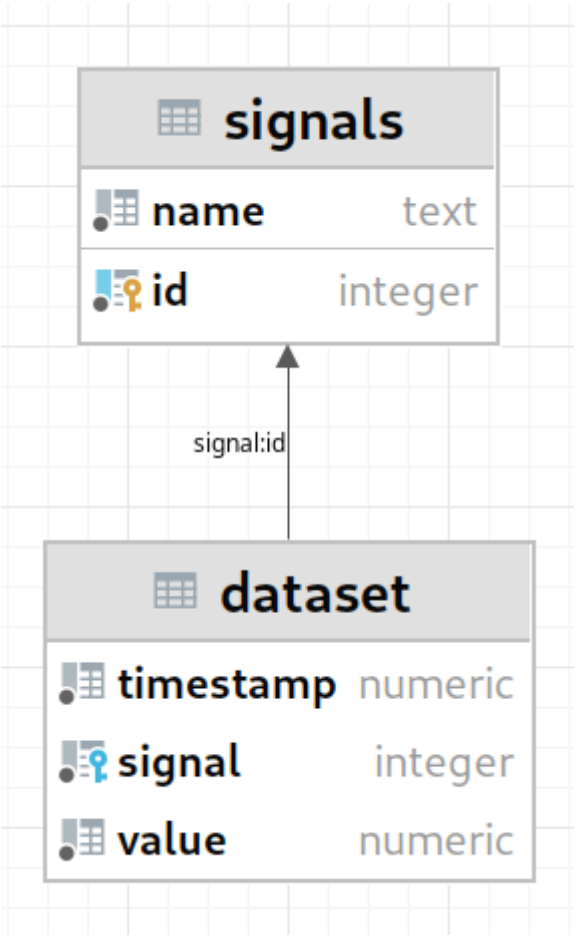
5.9 SQLite-tietokanta telemetrian talletusmuotona

SQLite on tietokantana mielenkiintoinen, koska siinä koko tietokanta voidaan kirjoittaa yksittäiseen tiedostoon. Näin koko telemetria voidaan kirjoittaa levyllä olevaan tietokantatiedostoon ja lähettää tiedosto eteenpäin. Koska kyseessä on suhteellisen monipuolinen SQL-toteutus, voidaan tietokantaa käyttää hyvin erilaisilla tavoilla.

Tietokannan käyttämisestä telemetriatiedon siirtämiseen on tässä simuloinnissa muodostettu kaksi mallia. Ensimmäinen malli käyttää tavanomaisia SQL-rakenteita ja signaalien tiedot on kirjoitettu erilliseen tauluun varsinaisesta aikasarjadatasta. Tällöin taulujen rakenteet pysyvät vakioina riippumatta siitä, miten monta signaalia datasarjassa on. Toisessa mallissa kaikki on yhdessä taulussa, mutta taulun rakenne muuttuu dynaamisesti sen mukaan, miten monta signaalia siihen halutaan kirjoittaa.

5.9.1 SQLite-tietokanta kiinteällä rakenteella

Tietokanta rakentuu kahdesta taulusta. Ensimmäisessä taulussa on kuvattu signaalit, joita tietokannassa on käytössä. Jokaiselle signaalille on annettu uniikki numeerinen avain. Tämän taulun nimi on ”signals”. Toisessa taulussa on varsinainen data, joka sisältää aikaleiman ja arvon, sekä viittauksen ”signals” tauluun, missä signaali on määritelty. Datan sisältämän taulun tunniste on ”dataset”. Koska jokaisesta signaalista kirjoitetaan uusi rivi ”dataset” tauluun, aikaleima saattaa toistua monta kertaa taulun sisältämässä tiedossa, joten sitä ei voi määritellä uniikiksi. Tähän muotoon viitataan testituloksissa nimikkeellä ”SQLite”. Kuva 22 esittää tietokannan rakenteen ER-kaaviona.



Kuva 22. Tietokantarakenne aikasarjadataa varten ER-kaaviona.

“signals” taulu on luotu SQL-komennolla:

CREATE TABLE "signals" ("id" INTEGER NOT NULL UNIQUE, "name" TEXT NOT NULL, PRIMARY KEY("id" AUTOINCREMENT))

“dataset” taulu vastaavasti on luotu SQL-komennolla:

CREATE TABLE "dataset"("timestamp" NUMERIC NOT NULL, "signal" INTEGER NOT NULL, "value" NUMERIC NOT NULL, FOREIGN KEY("signal") REFERENCES "signals"("id"))

5.9.2 SQLite-tietokanta dynaamisella rakenteella

Toisessa mallissa käytössä on vain yksi taulu, jossa sarakkeiden määrä vaihtelee sen mukaan, miten monta signaalia tauluun on kirjoitettu. Taulu luodaan aina uudestaan, kun aletaan kirjoittamaan aikasarjadataa tietokantaan. Tarvittava signaalien määrä täytyy olla

tiedossa ennen kuin tietokantataulu luodaan, koska tietokantaulun luontiin vaadittava SQL-komento muodostetaan sen mukaan, miten monta signaalia tauluun ollaan kirjoittamassa. Tähän telemetriamuotoon viitataan testituloksissa nimikkeellä ”SQLite table”.

”dataset” taulu luodaan seuraavanlaisella SQL-lausekkeella:

```
CREATE TABLE IF NOT EXISTS "dataset" ("timestamp" NUMERIC NOT NULL, "signal_name" NUMERIC NOT NULL);
```

SQL-komennossa oleva ”signal_name NUMERIC NOT NULL” komento toistetaan niin monta kertaa kuin tauluun ollaan kirjoittamassa signaaleita. Signaalin nimi (signal_name) täytyy aina vaihtaa lausekkeeseen signaalin mukaan.

SQLite-tietokantaan voi antaa tietotyypin, jota sarake käyttää, mutta SQLite ei ole tarkka tietotyyppien käytön suhteen, joten taulussa olevaan sarakkeeseen voidaan kirjoittaa vapaasti minkä tyyppisiä arvoja tahansa. Tämä saattaa kuitenkin aiheuttaa ongelmia tietokannan lukemisessa.

Tämän rakenteen ongelma on, että tietokantaa lukevan sovelluksen täytyy tutkia, miten tietokanta on rakennettu ja miten monta saraketta siihen kuuluun. Tietokannen rakenteen pystyy selvittämään käyttämällä komentoa ”PRAGMA table_info()”. Komento palauttaa kuvauksen tietokantataulun rakenteesta.

5.10 Telemetrian pakkaus

Jokainen määritelty telemetriamuoto on pakattavissa häviöttömästi. Tässä simuloinnissa on valikoitu joitain yleisimmin pakkaustapoja. Jokainen telemetriamuoto pakattiin yhdellä tavalla käyttäen kahta eri asetusta: nopein ja tehokkain pakkaus. Taulukko 2 esittelee käytetyt työkalut ja komennot.

Taulukko 2. Simuloinnissa käytetyt pakkausohjelmat ja komennot.

Pakkausohjelma ja versio	**Asetus**	**Komentoriviko-mento tiedoston pakkauksessa**	**Komentoriviko-mento tiedoston purkamisessa**
Gzip **1.10**	Nopea	gzip -1 <tiedosto>	gzip -d <tiedosto>
	Tehokas	gzip -9 <tiedosto>	gzip -d <tiedosto>
Bzip2 **1.0.8**	Nopea	bzip2 -1 <tiedosto>	bzip2 -d <tiedosto>
	Tehokas	bzip2 -9 <tiedosto>	bzip2 -d <tiedosto>
ZStandard **1.4.8**	Nopea	zstd -1 <tiedosto>	zstd -d <tiedosto>
	Tehokas	zstd -19 <tiedosto>	zstd -d <tiedosto>
XZ **5.2.5**	Nopea	xz -1 <tiedosto>	xz -d <tiedosto>
	Tehokas	xz -9 <tiedosto>	xz -d <tiedosto>
LZ4 **1.9.3**	Nopea	lz4 -1 <tiedosto>	lz4 -d <tiedosto>
	Tehokas	lz4 -12 <tiedosto>	lz4 -d <tiedosto>
Brotli **1.0.9**	Nopea	brotli -1 <tiedosto>	brotli -d <tiedosto>
	Tehokas	brotli -9 <tiedosto>	brotli -d <tiedosto>

Komennot suoritettiin automatisoidusti samalle lähdetiedostolle. Testit ajettiin Raspberry Pi 3B+ -laitteen 4 gigatavun mallilla. Käytössä oli Raspberry Pi OS 64-bittinen käyttöjärjestelmä ja käyttöjärjestelmän versio 2022-09-26.

Pakkauskerroin määritettiin mittaamalla pakkaamattoman ja pakatun tiedoston koko tiedostojärjestelmässä.

Pakkauskerroin lasketaan kaavalla (Pu, 2006, s. 11):

$$CF = \frac{SBC}{SBA} \quad (1)$$

Missä

CF	on pakkauskerroin (compression factor)
SBC	on tiedostokoko ennen pakkausta tavuina (size before compression)
SBA	on koko pakkauksen jälkeen tavuina (size after compression)

Pakkausnopeus mitattiin tarkastelemalla pakkausohjelman suoritusaikaa. Ohjelma käynnistettiin ulkopuolisen ohjelman toimesta testattavilla argumenteilla. Ennen ohjelman käynnistystä otettiin monotonisesta järjestelmäkellosta aloitusaika nanosekunteina. Kun ohjelman suoritus oli päättynyt, otettiin samasta kellosta lopetusaika nanosekunteina. Suoritusaika saatiin vähentämällä lopetusajasta ohjelman aloitusaika.

Pakkausnopeus määritettiin kaavalla:

$$Pakkausnopeus = \frac{Tiedostokoko}{Suoritusaika} \quad (2)$$

Missä

Pakkausnopeus	on tiedoston käsittelynopeus (MB/s)
Tiedostokoko	on pakkaamattoman tiedoston koko tiedostojärjestelmässä (MB)
Suoritusaika	on ohjelman suoritusaika (sekuntia)

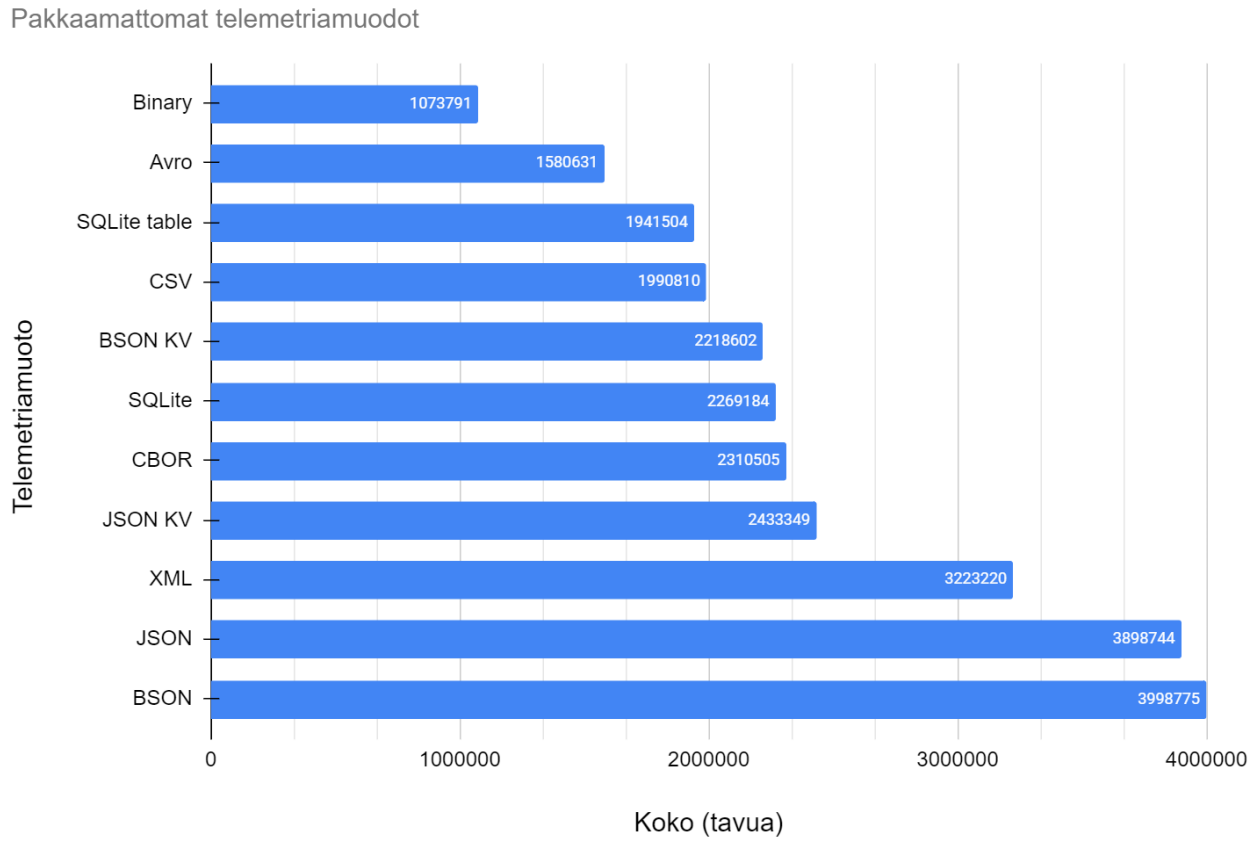
Pakkausnopeudessa huomattiin olevan hajontaa eri suorituskertojen välillä, joten jokaista pakkaustyökalun ja tiedostomuodon yhdistelmää testattiin viisi kertaa ja mitatuista pakkausnopeuksista laskettiin keskiarvo. Mittaustapa ei ole täydellinen, koska se ei mittaa pelkkää pakkausnopeutta, vaan ohjelman suoritusaikaa, mihin kuuluu myös esimerkiksi puskurien varauksia järjestelmämuistista. Absoluuttiset tulokset voivat siis poiketa hiukan todellisista, mutta tulokset ovat vertailukelpoisia eri tiedostomuotojen kesken. Pakkausohjelmien ulos tuleva tieto ohjattiin Linuxin ”/dev/null”-tiedostoon, jolloin pakkausnopeuteen vaikuttaa vain

tiedoston lukunopeus laitteen microSD-muistikortilta. Pakkaustehokkuus ja pakkausnopeus mitattiin erillisillä testauskerroilla, mutta käyttäen samoja lähdetiedostoja.

6 SIMULOINTITESTIEN TULOKSET

Telemetriatiedon kirjoittamisessa valikoituihin tiedostomuotoihoin ja pakkaamalla sisältö ulkoisella pakkaustyökalulla saatiin seuraavanlaiset testaustulokset. Koko lista kaikista erilaisista formaateista ja pakkaustyökaluista on liitteessä 2.

Kuva 23 esittää formaatin käyttämän levytilan tavuina, kun käytössä ei ole ulkoista pakkausta. Kuvaajassa pienempi arvo on parempi.



Kuva 23. Telemetriatiedon koko kirjoitettuna eri tiedostomuotoihin.

Tuloksista voidaan tulkita, että oma binääriformaatti on tehokkain tapa kirjoittaa aikasarjatietoa levylle. Standardoiduista formaateista AVRO olisi paras valinta tiedon siirtämiseen ja paras tekstimuotoinen tapa olisi CSV-muoto. Muut tekstimuotoiset formaatit (XML ja JSON) vaativat enemmän tilaa siitä syystä, että näissä muodoissa käytettään kiinteitä avainsanoja tiedon erottelemiseen. Mielenkiintoinen yksityiskohta on BSON-muodon suurin tiedostokoko, joka on koko ryhmän suurin. BSON-dokumentin tiedostokoko kuitenkin pienenee merkittävästi, kun tieto jäsennellään eri tavalla tiedoston sisällä (BSON KV).

6.1 Pakkauksen vaikutus tiedostokokoon

Telemetriatiedosto pakattiin valituilla pakkaustyökaluilla käyttäen kahta eri asetusta: nopea ja paras pakkaustehokkuus. Pakkaamattomien ja pakattujen tietomuotojen tiedostokoot on listattu taulukossa 3. Taulukko 4 esittää kaikki yhdistelmät ja kunkin pakkausmenetelmän tehokkuuden pakkauskertoimena, eli miten monta kertaa enemmän tietoa voidaan varastoida pakatussa muodossa. Suurempi arvo on parempi ja tehokkaampi pakkaustapa. Taulukon solun värillä on korostettu pakkausmenetelmiä, jotka toimivat suhteessa paremmin verrattuna muihin tapoihin. Viimeisessä sarakkeessa on myös mukana keskiarvo eri formaattien välillä valitulle pakkaustekniikalle. Liitteessä 2 on kuvaaja kaikista yhdistelmistä kuvattuna pylväskaaviona.

Taulukko 3. Mitatut tiedostokoot ennen pakkausta ja pakkauksen jälkeen.

Tavua	Avro	Binary	BSON	CBOR	CSV	JSON	SQLite	XML	BSON KV	JSON KV	SQLite table
Pakkaamaton	1580631	1073791	3998775	2310505	1990810	3898744	2269184	3223220	2218602	2433349	1941504
GZIP -1	414915	402582	734217	520117	498285	620360	948014	603214	537296	542881	666093
GZIP -9	309520	343801	536648	381087	383853	412777	822023	408558	405364	389567	561761
BZip -1	252117	306453	465008	336362	291685	311434	796010	304628	317481	297319	490324
BZip -9	214090	279070	375463	281155	272188	271095	724331	270096	267859	268809	446563
ZStd -1	381010	385567	578127	438091	422479	501348	942290	484920	448191	462143	605981
ZStd -19	264419	305279	447580	326136	312427	334377	655611	323751	325356	316623	460254
XZ -1	314256	316320	484320	368800	400720	426116	616360	424944	397800	404676	475984
XZ -9	238876	278016	407836	298912	297624	321744	532096	315772	302448	300716	413632
LZ4 -1	649215	553355	998732	660917	675401	838898	1203129	834847	714310	749892	819289
LZ4 -12	357063	374452	606155	437605	489676	552216	966041	535211	507929	505523	635799
Brotli -1	453276	419424	694991	505330	453885	544866	984665	534754	530135	500163	634263
Brotli -9	285436	317555	460201	351140	373856	394889	718514	389022	377462	376369	473246

Taulukko 4. Telemetriaformaatin ja pakkausmenetelmän pakkauskerroin.

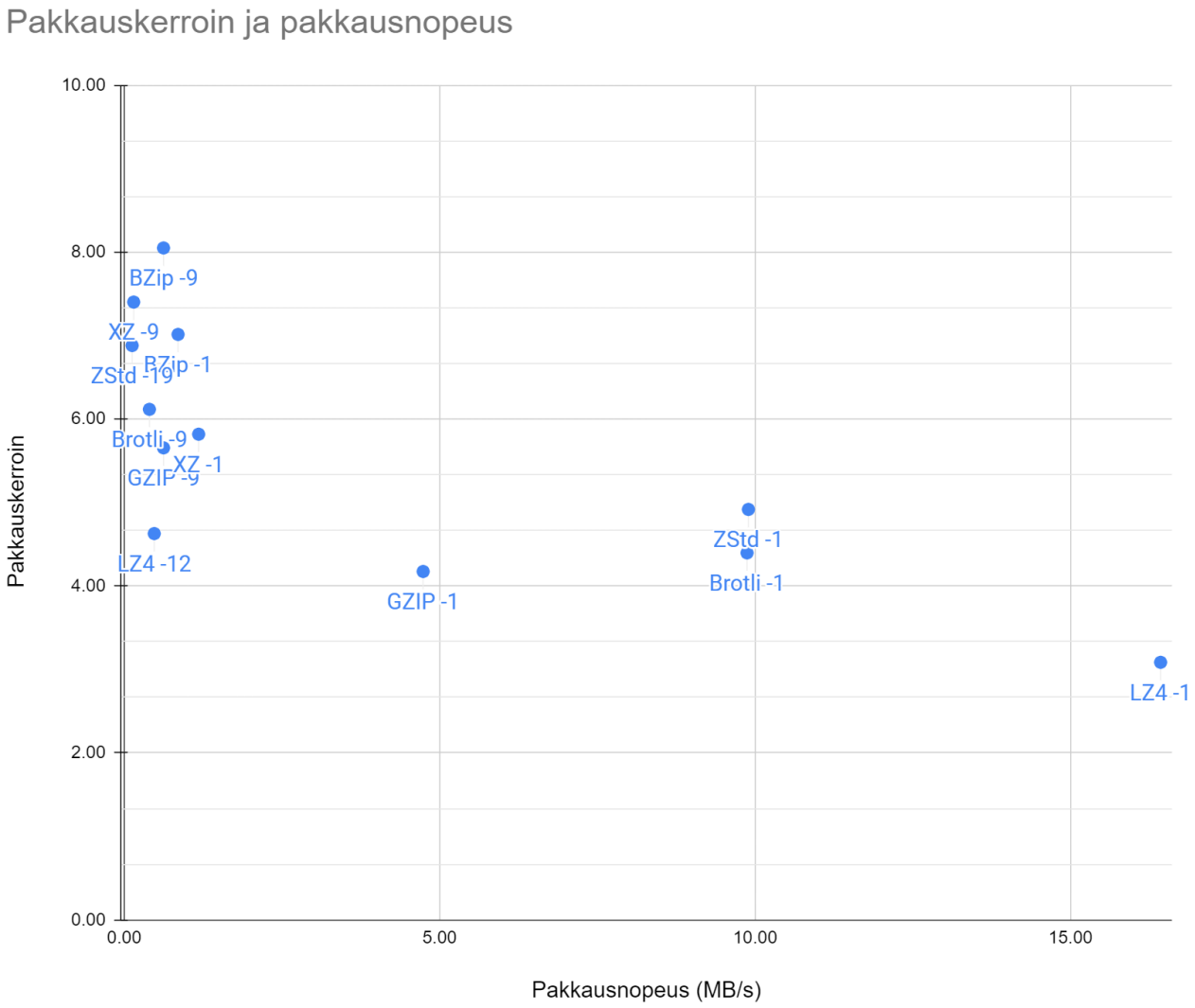
	Avro	Binary	BSON	CBOR	CSV	JSON	SQLite	XML	BSON KV	JSON KV	SQLite table	Keskiarvo
GZIP -1	3,81	2,67	5,45	4,44	4,00	6,28	2,39	5,34	4,13	4,48	2,91	4,17
GZIP -9	5,11	3,12	7,45	6,06	5,19	9,45	2,76	7,89	5,47	6,25	3,46	5,65
BZip -1	6,27	3,50	8,60	6,87	6,83	12,52	2,85	10,58	6,99	8,18	3,96	7,01
BZip -9	7,38	3,85	10,65	8,22	7,31	14,38	3,13	11,93	8,28	9,05	4,35	8,05
ZStd -1	4,15	2,78	6,92	5,27	4,71	7,78	2,41	6,65	4,95	5,27	3,20	4,92
ZStd -19	5,98	3,52	8,93	7,08	6,37	11,66	3,46	9,96	6,82	7,69	4,22	6,88
XZ -1	5,03	3,39	8,26	6,26	4,97	9,15	3,68	7,59	5,58	6,01	4,08	5,82
XZ -9	6,62	3,86	9,80	7,73	6,69	12,12	4,26	10,21	7,34	8,09	4,69	7,40
LZ4 -1	2,43	1,94	4,00	3,50	2,95	4,65	1,89	3,86	3,11	3,24	2,37	3,09
LZ4 -12	4,43	2,87	6,60	5,28	4,07	7,06	2,35	6,02	4,37	4,81	3,05	4,63
Brotli -1	3,49	2,56	5,75	4,57	4,39	7,16	2,30	6,03	4,18	4,87	3,06	4,40
Brotli -9	5,54	3,38	8,69	6,58	5,33	9,87	3,16	8,29	5,88	6,47	4,10	6,12

Keskiarvoltaan parhaat tavat pakata tietoa tässä käyttötapauksessa ovat BZip2-ohjelman tehokkaalla asetuksella (8,05) ja XZ-ohjelman tehokkaalla asetuksella (7,40). BZip2 toimi paremmin sekä nopealla, että tehokkaalla asetuksella kuin XZ kaikissa muissa datamuodoissa paitsi SQLite-tietokannoissa. Paras pakkauskerroin oli JSON-muodolla ja BZip2-ohjelman tehokkaalla asetuksella (14,38).

Yllättävää oli AVRO formaatin pakkautuminen. Formaatti pärjäsi jo hyvin pakkaamattomanakin (2. sija), mutta tiedosto pakkaantui vielä melko hyvällä pakkaustehokkuudella. SQL-tietokannat eivät tulosten valossa pienene pakkaamalla enää niin paljon kuin muut formaatit, ja myös oma binäärimuoto pakkautui muita vähemmän.

6.2 Pakkausnopeus

Pakkausalgoritmien ja työkalujen nopeudesta mitattiin keskiarvo viidestä peräkkäisestä pakkausoperaatiosta yhdelle tiedostomuodolle. Pakkausnopeudesta laskettiin keskiarvo, joka on esitetty kuvaajassa (kuva 24) suhteessa työkalun pakkaustehokuuden keskiarvoon. Taulukko 5 esittää kaikkien tiedostomuotojen ja pakkaustyökalujen pakkausnopeuksien tulokset.



Kuva 24. Keskimääräinen pakkauskerroin ja keskimääräinen pakkausnopeus X-Y taulukkona.

Taulukko 5. Pakkausnopeus eri tiedostomuodoilla MB/s.

MB/s	Avro	Binary	BSON	CBOR	CSV	JSON	SQLite	XML	BSON KV	JSON KV	SQLite table	Keskiarvo
GZIP -1	6,24	7,03	3,41	5,09	5,06	3,67	3,24	4,04	4,89	4,58	4,89	4,74
GZIP -9	0,69	0,66	0,47	0,69	0,46	0,68	0,33	0,71	0,84	0,65	0,70	0,62
BZip -1	1,07	1,49	0,37	0,70	1,28	0,33	0,83	0,50	0,83	0,74	1,25	0,85
BZip -9	0,77	1,16	0,23	0,49	0,91	0,17	0,73	0,31	0,61	0,48	0,98	0,62
ZStd -1	11,33	13,20	8,22	10,42	10,38	8,73	7,84	9,05	9,81	10,17	9,70	9,89
ZStd -19	0,21	0,36	0,04	0,10	0,10	0,05	0,12	0,07	0,12	0,10	0,13	0,13
XZ -1	1,66	1,71	0,89	1,19	1,24	0,95	0,82	1,08	1,20	1,13	1,12	1,18
XZ -9	0,18	0,30	0,07	0,12	0,15	0,08	0,18	0,12	0,14	0,13	0,18	0,15
LZ4 -1	19,69	22,92	12,30	17,48	17,66	12,94	12,77	15,99	16,71	15,98	16,27	16,43
LZ4 -12	0,71	0,80	0,39	0,52	0,36	0,46	0,18	0,41	0,54	0,58	0,29	0,48
Brotli -1	10,81	11,63	8,20	11,13	11,04	9,16	6,26	10,02	9,70	11,46	9,21	9,87
Brotli -9	0,63	0,70	0,27	0,42	0,37	0,30	0,21	0,31	0,41	0,38	0,40	0,40

Nopeat pakkaustyökalut korostuvat tässä testissä. LZ4 nopealla asetuksella oli selkeästi koko joukon nopein (16,43 MB/s), mutta samalla pakkauskerroin oli huonoin (3,09). Brotli (9,87) ja ZStd (9,89) ovat nopealla asetuksella hyvin samalla tasolla pakkausnopeudessa, mutta ZStd on hiukan parempi pakkauskertoimessa (4,92 vs. 4,40). Gzip nousi myös erilleen muusta ryhmästä nopealla asetuksella, mutta pakkaustehokkuus ei ollut niin hyvä kuin Brotlissa tai ZStd:ssä. Tästä voi tehdä johtopäätöksen, että Brotli tai ZStd ovat parempia valintoja kuin Gzip, kun haetaan nopeaa pakkaustyökalua.

Muiden työkalujen tulokset jäivät samaan ryppääseen, jossa pakkausnopeus pyörii noin 1 MB/s tasolla ja pakkauskerroin on noin 5,0–8,0 tuntumassa. LZ4 tehokkaalla asetuksella on hiukan tästä ryppäästä ulkopuolella johtuen huonommasta pakkaussuhteesta. Pakkausnopeus on kuitenkin selkeästi hitaanpuoleinen.

Pakkausnopeudessa näyttäisi olevan hieman eroja myös eri tiedostomuotojen välillä. Yksikään tiedostomuoto ei ollut selkeästi parempi kuin toinen tai yksikään pakkaustyökalu ei näyttäisi toimivan merkittävästi paljon nopeampaa tietyissä tiedostomuodoissa.

7 YHTEENVETO JA POHDINTA

Loppupäätelmänä voidaan todeta, että telemetriassa käytettävän formaatin valinta ei ole aina suoraviivaista. Valinnassa tulisi huomioida käyttötapaus, vaatimukset, rajoitteet ja skaalautuvuus. Käyttötapaus määrittää, miten usein telemetriaa lähetetään laitteelta ja miten isoissa erissä. Kolme tapaa lähettää tietoa ovat viestipohjainen, tiedostojen siirtoon perustuva tapa tai tietovirran lähettäminen. Samaan aikaan valittu tiedonsiirtotapa asettaa rajoitteet, miten paljon laitteella on tiedonsiirtokaistaa käytettävissä. Laitteen tarjoamat resurssit, suoritinteho, käyttömuisti ja talletusmuistin määrä asettavat omat rajoitteensa. Tiedon puskurointi laitteen muistissa voidaan myös toteuttaa samalla formaatilla kuin missä se lähetetään eteenpäin laitteelta.

Yksiselitteisesti voidaan kuitenkin todeta, että telemetriadatan pakkaaminen on aina järkevää. Tässä tutkimuksessa tarkasteltujen pakkaustyökalujen välillä löytyi selviä erovaisuuksia pakkaustehokkuudessa. Myös valittu formaatti telemetriadatalle vaikutti pakkaustehokkuuteen. Pakkaustyökalun valinta on enemmänkin kompromissi sen suhteen, mikä on tärkeää käyttötapauksessa. Osa työkaluista käyttää enemmän resursseja pakkausvaiheessa, jolloin päästään useimmiten parempaan pakkaustehokkuuteen. Toisilla työkaluilla painopiste on nopeassa purkamisessa, joka taas heijastuu järjestelmän skaalautumisessa. Kun kaikki vastaanotettu telemetria puretaan keskitetysti pilvipalvelussa, näkyy se siellä lisääntyneinä käyttökuluina, jos purkaminen on suhteellisen raskas operaatio.

Myös valitun sarjallistamismuodon rakenteeseen on syytä kiinnittää huomiota. Oma tarkoitukseen tehty binäärimuotoinen rakenne voi olla tilan käytön suhteen tehokas, mutta samaan aikaan se voi olla vaikeasti laajennettavissa ja hankala käsitellä. Jos telemetrian vaatiman tiedonsiirtoakaistan suhteen ei ole eritysvaatimuksia, on lähes aina suositeltavaa käyttää jotain yleiskäyttöistä datamuotoa. Yksi mahdollisuus on myös tehdä toteutus, joka käyttää yleistä tapaa tiedon esittämismuotoon, kuten esimerkiksi Eclipse Sparkplug -standardia.

Tässä työssä todettiin, että paras datamuoto lähettää aikasarjadataa telemetriassa on Apache Avro -tietomuoto. Siinä on hyvänä puolena sisäänrakennettu tuki tiedon skeemalle. BZip2-työkalu osoittautui tehokkaimmaksi tavaksi pakata telemetriatietoa. Se on myös melko raskas työkalu tiedon pakkaamisen ja purkamiseen. Zstandard, Brotli ja Gzip tarjoavat hyvän kompromissin pakkaustehokkuuden ja -nopeuden välillä.

Pakkaustyökaluja olisi voinut testata vielä enemmänkin. Googlella on useampia pakkauskirjastoja, jotka se on antanut vapaaseen jakeluun. Snappy on myös nopeuteen tähtäävä työkalu, jota käytetään monissa tietokantaohjelmistoissa. Se on hyvin samanlainen kuin ZStd. Myös Apple on kehittänyt oman pakkaustyökalun nimeltään LZFSE. Se on myös nopeaan käsittelyyn erikoistunut työkalu ja julkaistu vapaaseen jakeluun. Brotlista on myös esitelty Brotli-G versio, joka käyttää GPGPU-laskentaa datan pakkauksen ja purkamisen kiihdyttämiseen.

Microsoft Azure Stream Processing for Edge ja eKuiper olivat ainoat tässä työssä esitellyt reunalaskentakäyttöön tarkoitetut työkalut. Azure Stream Processing for Edgen hankaluus on, että se sitoo käyttämään Microsoft Azuren palveluita, ainakin IoT-laitteiden hallinnan ja tiedon vastaanottamaisen kanssa. eKuiper on vielä hyvin tuore työkalu, eikä kovin laajasti käytössä. Sen käyttöönotto on myös vielä melko hankalaa ja vaatii toimiakseen myös muita palveluita laitteelle, kuten esimerkiksi EdgeX-alustan. InfluxDB ja Redis ovat muuhun käyttöön alun perin tehtyjä työkaluja, mutta niitä voidaan soveltaa myös reunalaskennan työkaluina. Nämä ovat toiminnaltaan myös lähempänä tietokantaa kuin stream processing -palvelua.

Reunalaskennan työkalut tulevat varmasti kehittymään vielä merkittävästi tulevaisuudessa. Reunalaskenta on melko monia asioita tarkoittava termi, ja sitä käytetäänkin useasti käyttötapauksissa, missä tiedonkäsittelyä pyritään hajauttamaan useaan paikkaan käyttäen ohjelmistokontteihin perustuvaa tekniikkaa. Tällä hetkellä markkinoilla on kahden ison pilvipalvelutarjoajan, Amazonin ja Microsoftin, tarjoamat ratkaisut helppoon reunalaskennan käyttöönottoon. Teollisen internetin käyttötapauksessa reunalaskennan merkitys korostuu vielä lisää, koska käyttöympäristö tuo monesti vaatimuksen palveluiden saatavuuden varmistamisesta. Reunalaskentalaite voi toimia itsenäisesti ja tarjota palveluita paikallisesti, vaikka pilvipalvelu ei olisi käytettävissä. Näin pyritään pitämään järjestelmän toimintahäiriön vaikutus teollisten prosessien toimintaan minimissään.

Yksi isoista haasteista reunalaskentajärjestelmissä on tietoturvan hallinta. Laitteet eivät ole välttämättä samalla tapaa valvottuja ympäristöjä kuin perinteiset konesalit. Laitteisiin voi päästä helpommin fyysisesti käsiksi ja laitteen ohjelmistoja ei päivitetä ja ylläpidetä samalla tasolla kuin esimerkiksi verkkopalvelimia. Laitteiden massahallinta voisi olla yksi ratkaisu tähän haasteeseen. Laitekanta voidaan päivittää keskitetysti, ilman että operaattorin täytyy käydä erikseen jokaista laitetta läpi. Kriittisten tietoturvapäivitysten automaattinen

päivittyminen laitteisiin pitäisi olla vakio-ominaisuus jokaisessa laitteessa. Reunalaskenta-sovellusten toimittaminen laitteisiin ohjelmistokonteissa helpottaa ylläpitoa jossain määrin, kun pohjalla toimiva käyttöjärjestelmä voidaan pitää kevyempänä. Ohjelmistokontit sisältävät kaikki ohjelman tarvitsevat riippuvuudet (kirjastot yms.), joten kaikki ohjelmistokomponentit voidaan päivittää kerralla, ilman riskiä riippuvuuksien versioristiriidasta. Uusi lupaava teknologia on Kata Containers, missä ohjelmistokontteja ajetaan omissa pienissä virtuaalikoneissa. Tämä pienentää riskiä siitä, että vihamielinen ohjelmisto pääsee ulos sille annetusta ”hiekkalaatikosta” ja ottaa laitteen hallintaansa.

Pilvipalveluiden puolella on tapahtunut jo pudotuspeliä IoT-tuotteiden ympärillä. Google ilmoitti elokuussa 2022 lopettavansa IoT Core -palvelun vuoden 2023 elokuussa (Clark, 2022a). Pian myös IBM ilmoitti, että he lopettavat Watson IoT -palvelun joulukuussa 2023 (Clark, 2022b). Suurista pilvipalveluiden tarjoajista jäljellä ovat siis enää Amazon ja Microsoft Azure. Kentällä on kuitenkin useita pienempiä toimijoita ja monia tuotteita, jotka tarjoavat pelkästään alustaa, joka voidaan asentaa melkein mihin tahansa pilvipalveluun tai hosting-palveluun.

Tässä työssä saatuja tuloksia ja oppeja tullaan hyödyntämään yrityksessä, kun suunnitellaan asiakastarpeisiin räätälöityjä IoT-ratkaisuja. Telemetriatiedon muotoilun suunnittelu ja valinta ovat jokaisessa järjestelmässä mukana olevia haasteita, jotka täytyy ratkaista.

LÄHTEET

Alakuijala, J., & Szabadka, Z. (2016). *Brotli Compressed Data Format*. IETF. https://www.ietf.org/rfc/rfc7932.txt

Apache. (i.a.). *Apache AVRO Specification 1.1.11*. https://avro.apache.org/docs/1.11.1/specification/#schema-declaration

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. *MCC '12: Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 13-16. https://doi.org/10.1145/2342509.2342513

Bormann, C., & Hoffman, P. (2020). *RFC 8949 Concise Binary Object Representation (CBOR)*. IETF. https://www.rfc-editor.org/rfc/rfc8949.html

BSON. (i.a.). *BSON Specification Version 1.1*. https://bsonspec.org/spec.html

Charest, G., & Rogers, M. (2020). *Data Exchange Mechanisms and Considerations*. Harvard. https://enterprisearchitecture.harvard.edu/data-exchange-mechanisms

Clark, L. (2022a). *Google shuts off IoT Core services shortly after announcing API stability commitments*. The Register. https://www.theregister.com/2022/08/19/google_iot_core_axed/

Clark, L. (2022b). *IBM to fire Watson IoT Platform from its cloud*. The Register. https://www.theregister.com/2022/11/15/ibm_set_to_retire_watson/

Collet, Y. (2021). *RFC 8878 Zstandard Compression and the 'application/zstd' Media Type*. https://www.rfc-editor.org/rfc/rfc8878

Collet, Y. (2022). *lz4: Man Page*. Mankier. https://www.mankier.com/1/lz4

Collin, J., & Saarelainen, A. (2016). *Teollinen internet.* Talentum.

Donahoo, M. J., & Speegle, G. D. (2005). *SQL: Practical Guide for Developers.* Elsevier Science & Technology.

Eclipse. (2022). *Sparkplug 3.0.0 - Sparkplug Specification*. https://sparkplug.eclipse.org/specification/version/3.0/documents/sparkplug-specification-3.0.0.pdf

Ecma International. (2017). *The JSON Data Interchange Syntax*. Ecma International. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

eKuiper. (i.a.). *LF Edge eKuiper - Lightweight data stream processing engine for IoT edge*. Haettu 10.2.2023, https://ekuiper.org/docs/en/latest/

Estrada, R. (2018). *Apache Kafka Quick Start Guide: Leverage Apache Kafka 2.0 to Simplify Real-Time Data Processing for Distributed Applications.* Packt Publishing.

Fischer, J., & Ning, W. (2022). *Grokking Streaming Systems: Real-time event processing.* Manning Publications.

Gailly, J.-l. (i. a.). *gzip(1): Linux man page*. die.net. https://linux.die.net/man/1/gzip

Gilchrist, A. (2016). *Industry 4.0: The Industrial Internet of Things.* Apress. https://doi.org/10.1007/978-1-4842-2047-4

Influxdata. (2022a). *Get started with Flux*. https://docs.influxdata.com/flux/v0.x/get-started/

Influxdata. (2022b). *Why Use InfluxDB Open Source*. https://www.influxdata.com/products/influxdb/

InfluxData. (i. a.). *What is time series data?* https://www.influxdata.com/what-is-time-series-data/

Lea, P. (2020). *IoT and Edge Computing for Architects.* (2 p.). Packt Publishing.

Lin, S.-W., Bradford, M., Durand, J., Bleakley, G., Chigani, A., Martin, R., & Crawford, M. (2019). *The Industrial Internet of Things Volume G1: Reference Architecture.* Industrial Internet Consortium. https://www.iiconsortium.org/pdf/IIRA-v1.9.pdf

Marble, J. (2022). *Building Blocks of the InfluxDB Data Model* [video]. Youtube. https://youtu.be/3qTTqsL27lI

Marshall, P. (2021). Networks and Networking. *State of the Edge 2021: A Market and Ecosystem Report for Edge Computing*, 34-44.

McCain, D. (2023). *Implementing Cellular IoT Solutions for Digital Transformation.* Packt Publishing.

Microsoft. (2021). *Azure Stream Analytics on IoT Edge*. https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-edge

Microsoft. (2022a). *Welcome to Azure Stream Analytics*. https://learn.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

Microsoft. (2022b). *Understand the Azure IoT Edge runtime and its architecture*. https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime?view=iotedge-1.4

Microsoft. (2022c). *How an IoT Edge device can be used as a gateway*. https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-as-gateway?view=iotedge-1.4

Microsoft. (2022d). *Develop without using an Azure IoT Hub SDK*. https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-no-sdk

Microsoft. (2022e). *Tutorial: Deploy Azure Stream Analytics as an IoT Edge module*. https://learn.microsoft.com/en-us/azure/iot-edge/tutorial-deploy-stream-analytics?view=iotedge-1.4

Microsoft. (i. a.). *Azure IoT reference architecture*. https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/iot

Mozilla. (2022). *XML introduction*. https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction

Nummenmaa, L., Holopainen, M., & Pulkkinen, P. (2014). *Tilastollisten menetelmien perusteet.* Sanoma Pro.

Pu, I. M. (2006). *Fundamental Data Compression.* Butterworth-Heinemann.

Redis. (2022a). *Introduction to Redis*. https://redis.io/docs/about/

Redis. (2022b). *Redis commands reference: timeseries*. https://redis.io/commands/?group=timeseries

Redis. (2022c). *RedisTimeSeries*. https://redis.io/docs/stack/timeseries/

Seward, J. (i. a.). *man page - bzip2*. Helpmanual. https://helpmanual.io/man1/bzip2/

Shafranovich, Y. (2005). *RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files*. https://datatracker.ietf.org/doc/html/rfc4180

Silverman, B., & Solberg, M. (2018). *OpenStack for Architects* (2 p.). Packt Publishing

Simone, S. D. (2016). *Facebook Open-Sources New Compression Algorithm Outperforming Zlib*. InfoQ. https://www.infoq.com/news/2016/09/facebook-zstandard-compression/

Smallcombe, M. (2023). *Structured vs Unstructured Data: 5 Key Differences*. Integrate. https://www.integrate.io/blog/structured-vs-unstructured-data-key-differences/

Sodabathina, R., Shan, J., & Ulloa, M. (2022). *Building event-driven architectures with IoT sensor data*. Amazon. https://aws.amazon.com/blogs/architecture/building-event-driven-architectures-with-iot-sensor-data/

SQLite. (2021). *About SQLite*. https://sqlite.org/about.html

Stocker, A., Kaiser, C., & Festl, A. (2017). *Automotive Sensor Data: An Example Dataset from the AEGIS Big Data Project.* https://doi.org/10.5281/zenodo.820576

TIBCO. (i.a.). *What is Structured Data?* TIBCO Software. https://www.tibco.com/reference-center/what-is-structured-data

Vogiatzis, D. (2022). *Your Guide to Data Transformation Techniques*. Coupler.io. https://blog.coupler.io/data-transformation-techniques/

W3C. (2016). *Extensible Markup Language (XML)*. W3C. https://www.w3.org/XML/

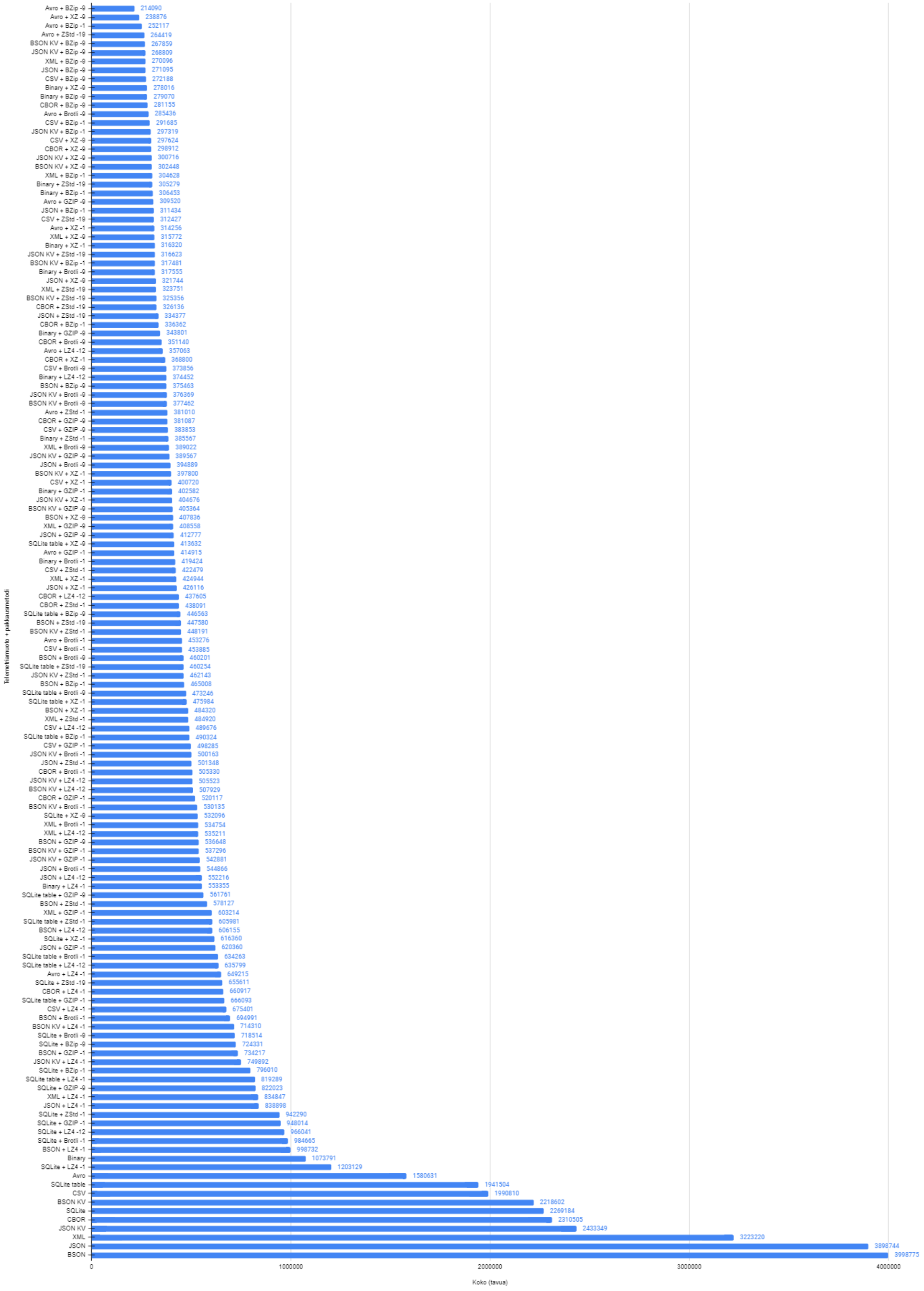
xz: Man Page. (2022). Mankier. https://www.mankier.com/1/xz

LIITTEET

Liite 1. Otanta simuloinnissa käytetystä aikasarjadatasta

Liite 2. Kaikki telemetriaformaatit ja pakkaustyökalut pienimmästä alkaen

Aikaleima	acc.x	acc.y	acc.z	gyro.x	gyro.y	gyro.z	latitude	longitude	alti-tude	obd.04	obd.05	obd.0B	obd.0C	obd.0D	obd.0F	obd.10	obd.11	obd.33	obd.3c
1484885573.80339										90									
1484885573.80952				-3.47826	-1.66957	-8.69565													
1484885573.8112	0.22266	0.24609	1.07812																
1484885573.82773							4658.4215	1520.1199	105.7										
1484885573.8335											16								
1484885573.8513				-2.92174	0.90435	-8.69565													
1484885573.8531	0.23828	0.20313	1.05859																
1484885573.85364												137							
1484885573.8834													1378						
1484885573.89317				-2.22609	0.97391	-8.69565													
1484885573.89487	0.24219	0.20313	1.03906																
1484885573.90352														34					
1484885573.93352															-6				
1484885573.93519				-0.55652	2.85217	-8.76522													
1484885573.93682	0.25391	0.15625	0.98047																
1484885573.95364																37			
1484885573.97701				-2.43478	2.22609	-7.7913													
1484885573.97861	0.19141	0.23047	0.99609																
1484885573.98345																	0		
1484885574.00344																		99	
1484885574.01885				-2.08696	2.15652	-7.16522													
1484885574.02045	0.27344	0.16797	1.0625																
1484885574.03348																			171


Telemetriamuoto + pakkausmetodi
Avro + BZip -9 214090
Avro + XZ -9 238876
Avro + BZip -1 252117
Avro + ZStd -19 264419
BSON KV + BZip -9 267859
JSON KV + BZip -9 268809
XML + BZip -9 270096
JSON + BZip -9 271095
CSV + BZip -9 272188
Binary + XZ -9 278016
Binary + BZip -9 279070
CBOR + BZip -9 281155
Avro + Brotli -9 285436
CSV + BZip -1 291685
JSON KV + BZip -1 297319
CSV + XZ -9 297624
CBOR + XZ -9 298912
JSON KV + XZ -9 300716
BSON KV + XZ -9 302448
XML + BZip -1 304628
Binary + ZStd -19 305279
Binary + BZip -1 306453
Avro + GZIP -9 309520
JSON + BZip -1 311434
CSV + ZStd -19 312427
Avro + XZ -1 314256
XML + XZ -9 315772
Binary + XZ -1 316320
JSON KV + ZStd -19 316623
BSON KV + BZip -1 317481
Binary + Brotli -9 317555
JSON + XZ -9 321744
XML + ZStd -19 323751
BSON KV + ZStd -19 325356
CBOR + ZStd -19 326136
JSON + ZStd -19 334377
CBOR + BZip -1 336362
Binary + GZIP -9 343801
CBOR + Brotli -9 351140
Avro + LZ4 -12 357063
CBOR + XZ -1 368800
CSV + Brotli -9 373856
Binary + LZ4 -12 374452
BSON + BZip -9 375463
JSON KV + Brotli -9 376369
BSON KV + Brotli -9 377462
Avro + ZStd -1 381010
CBOR + GZIP -9 381087
CSV + GZIP -9 383853
Binary + ZStd -1 385567
XML + Brotli -9 389022
JSON KV + GZIP -9 389567
JSON + Brotli -9 394889
BSON KV + XZ -1 397800
CSV + XZ -1 400720
Binary + GZIP -1 402582
JSON KV + XZ -1 404676
BSON KV + GZIP -9 405364
BSON + XZ -9 407836
XML + GZIP -9 408558
JSON + GZIP -9 412777
SQLite table + XZ -9 413632
Avro + GZIP -1 414915
Binary + Brotli -1 419424
CSV + ZStd -1 422479
XML + XZ -1 424944
JSON + XZ -1 426116
CBOR + LZ4 -12 437605
CBOR + ZStd -1 438091
SQLite table + BZip -9 446563
BSON + ZStd -19 447580
BSON KV + ZStd -1 448191
Avro + Brotli -1 453276
CSV + Brotli -1 453885
BSON + Brotli -9 460201
SQLite table + ZStd -19 460254
JSON KV + ZStd -1 462143
BSON + BZip -1 465008
SQLite table + Brotli -9 473246
SQLite table + XZ -1 475984
BSON + XZ -1 484320
XML + ZStd -1 484920
CSV + LZ4 -12 489676
SQLite table + BZip -1 490324
CSV + GZIP -1 498285
JSON KV + Brotli -1 500163
JSON + ZStd -1 501348
CBOR + Brotli -1 505330
JSON KV + LZ4 -12 505523
BSON KV + LZ4 -12 507929
CBOR + GZIP -1 520117
BSON KV + Brotli -1 530135
SQLite + XZ -9 532096
XML + Brotli -1 534754
XML + LZ4 -12 535211
BSON + GZIP -9 536648
BSON KV + GZIP -1 537296
JSON KV + GZIP -1 542881
JSON + Brotli -1 544866
JSON + LZ4 -12 552216
Binary + LZ4 -1 553355
SQLite table + GZIP -9 561761
BSON + ZStd -1 578127
XML + GZIP -1 603214
SQLite table + ZStd -1 605981
BSON + LZ4 -12 606155
SQLite + XZ -1 616360
JSON + GZIP -1 620360
SQLite table + Brotli -1 634263
SQLite table + LZ4 -12 635799
Avro + LZ4 -1 649215
SQLite + ZStd -19 655611
CBOR + LZ4 -1 660917
SQLite table + GZIP -1 666093
CSV + LZ4 -1 675401
BSON + Brotli -1 694991
BSON KV + LZ4 -1 714310
SQLite + Brotli -9 718514
SQLite + BZip -9 724331
BSON + GZIP -1 734217
JSON KV + LZ4 -1 749892
SQLite + BZip -1 796010
SQLite table + LZ4 -1 819289
SQLite + GZIP -9 822023
XML + LZ4 -1 834847
JSON + LZ4 -1 838898
SQLite + ZStd -1 942290
SQLite + GZIP -1 948014
SQLite + LZ4 -12 966041
SQLite + Brotli -1 984665
BSON + LZ4 -1 998732
Binary 1073791
SQLite + LZ4 -1 1203129
Avro 1580631
SQLite table 1941504
CSV 1990810
BSON KV 2218602
SQLite 2269184
CBOR 2310505
JSON KV 2433349
XML 3223220
JSON 3898744
BSON 3998775
0
1000000
2000000
3000000
4000000
Koko (tavua)