TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

HEIKKI HEMMINKI

KAATUMISVEDOSTEN RAPORTOINTI SULAUTETUSSA AUTOMAATIOJÄRJESTELMÄSSÄ

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
30. marraskuuta 2017

TIIVISTELMÄ

HEIKKI HEMMINKI: Kaatumisvedosten raportointi sulautetussa automaatiojärjestelmässä
Tampereen teknillinen yliopisto
Diplomityö, 52 sivua
Toukokuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Hannu-Matti Järvinen

Avainsanat: Linux, Ethernet, TCP/IP, sulautetut laitteet, kaatumisvedokset

Tässä työssä käydään läpi suunnitelma, jonka perusteella on toteutettu kaatumisvedosten raportointijärjestelmä erääseen sulautettuun automaatiojärjestelmään. Automaatiojärjestelmä koostuu pienistä sulautetuista laitteista, jotka ohjaavat ja seuraavat erästä teollisuusjärjestelmää. Laitteet toimivat räätälöidyllä Linux-käyttöjärjestelmällä. Kaatumisvedoksia muodostetaan, kun laitteilla ajettava ohjelma kaatuu tai jos Linux-käyttöjärjestelmän ydin kaatuu. Automaatiojärjestelmässä on myös erillinen työpöytäsovellus, jolla voidaan konfiguroida sulautettujen laitteiden toimintaa tai seurata niiden tilaa. Raportointijärjestelmä on integroitava osaksi automaatiojärjestelmän toimintaa, mutta automaatiojärjestelmän toimintaa ei saa häiritä, sillä automaatiojärjestelmän toiminnalle on asetettu tiukat reaaliaikaisuusvaatimukset. Sulautetuilla laitteilla ei ole paljoa säilytystilaa, joten kaatumisvedoksia ei voida kerätä loputtomasti.

Työpöytäsovellus on yhteydessä vain yhteen tietyn tyyppiseen sulautettuun laitteeseen. Työpöytäsovelluksen ja sulautetun laitteen välillä hyödynnetään TCP/IP-protokollayhdistelmää. Sulautetut laitteet on kiinnitetty toisiinsa Ethernet-kaapeleilla. Laitteiden välillä hyödynnetään Ethernet-tekniikkaan perustuvaa protokollaa, jolla saavutetaan redundanssisuus. Kun Linux-käyttöjärjestelmän ydin tai ajettava ohjelma kaatuu, kaatumisesta luodaan kaatumisvedos. Kun laite käynnistyy uudelleen, kaatumisvedokset pakataan tilan säästämiseksi alkulatausohjelmassa. Kaatumisvedokset raportoidaan työpöytäsovelluksen avulla. Työpöytäsovellus voi siirtää tietyn tyyppiseltä laitteelta kaatumisvedoksia SFTP-protokollan avulla. Muiden laitteiden kaatumisvedokset siirretään kyseiselle laitteelle Ethernet-tekniikan päälle rakennetun protokollan avulla. Kun kaikki kaatumisvedokset on siirretty työpöytäsovellukselle, kaatumisvedokset lähetetään kehittäjille Internetin yli.

Lopputuloksena saatiin aikaan raportointijärjestelmä, joka on integroitu automaatiojärjestelmään ja siihen kuuluviin työkaluihin ja prosesseihin. Raportointijärjestelmä ei häiritse automaatiojärjestelmän toimintaa missään vaiheessa. Kaatumisvedokset lähetetään kehittäjille, jolloin he saavat selvitettyä ajettavan ohjelman tai Linux-käyttöjärjestelmän ytimen kaatumisen syyn.

ABSTRACT

HEIKKI HEMMINKI: Crash dump reporting in an embedded automation system
Tampere University of Technology
Master of Science Thesis, 52 pages
May 2018
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Hannu-Matti Järvinen

Keywords: Linux, Ethernet, TCP/IP, embedded devices, crash dumps

This master's thesis covers a plan that was used to implement a crash dump reporting system to an embedded automation system. The automation system consists of small embedded devices that control and monitor an industry system. The devices run on a customized Linux operating system. Crash dumps are created when a runnable program or Linux kernel crashes on the devices. There is also a separate desktop application that is used for configuring the automation system and monitoring its state. The reporting system must be integrated as a part of the automation system, but it must not interfere with any of its operations because the automation system has strict real time computing requirements. The embedded devices do not have a lot of storage space, so the devices cannot accumulate crash dumps indefinitely.

The desktop application is connected to only certain types of embedded devices. TCP/IP protocol combination is used between the desktop application and the embedded device. The embedded devices are connected to each other with Ethernet-cables. A protocol based on Ethernet is utilized between the embedded devices, which offers redundancy. When Linux kernel or a runnable program crashes, a crash dump is created. When the embedded device reboots, the crash dumps are compressed in the bootloader of the device to save storage space. The crash dumps are reported with the desktop application. It can transfer files from a certain type of device with SFTP protocol. The crash dumps from other embedded devices must be transferred to the device in question with a protocol that has been created on top of Ethernet. When all crash dumps have been transferred to the desktop application, they are sent to the developers over the Internet.

The result is a crash dump reporting system that is integrated as part of the automation system and its tools and processes. The reporting system does not interfere with the operations of the automation system. The crash dumps are sent to developers, so they can investigate the reason for a runnable program or Linux kernel crashing.

ALKUSANAT

Tässä työssä käsitelty ominaisuus tehtiin osana automaatiojärjestelmää, jota kehitetään Wapice Oy:n asiakasyritykselle. Ominaisuutta on kehitetty vaiheittain monen vuoden ajan, mutta tässä diplomityössä käsiteltyä osuutta on kehitetty noin viisi kuukautta. Itse diplomityö kirjoitettiin ominaisuuden kehityksen aikana. Haluan kiittää kaikkia Wapice Oy:n työntekijöitä, jotka ovat olleet mukana kehittämässä osia, joita ominaisuudessa on hyödynnetty. Lisäksi haluan kiittää Otto Bothasta, joka on ollut mukana ominaisuuden toteutuksen eri vaiheissa ja jolta olen saanut arvokasta palautetta ja neuvoa ominaisuuden kehityksessä. Lopuksi haluan kiittää ohjaajaani Hannu-Matti Järvistä, jolta olen saanut neuvoa ja palautetta diplomityöhön.

Tampereella, 24.04.2018

Heikki Hemminki

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ASCII	American Standard Code for Information Interchange
BEEP	Blocks Extensible Exchange Protocol
CSMA/CD	Carrier Sense Multiple Access With Collision Detection
GCC	GNU Compiler Collection
FTP	File Transfer Protocol
GDB	GNU Debugger
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
MAC	Medium Access Control
MTD	Memory Technology Device
NTP	Network Time Protocol
OSI-malli	Open Systems Interconnection Reference Model
RAM	Random Access Memory
SCP	Secure Copy Protocol
SFTP	SSH File Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language

1. JOHDANTO

Teollisuudessa tuotantoprosessit ovat hyvin monimutkaisia. Esimerkiksi lääketieteen tarvikkeita valmistavassa laboratoriossa tai sähkölaitteita valmistavassa tehtaassa on monia itsenäisiä komponentteja, jotka muodostavat kokonaisen tuotantoprosessin. Komponentit voivat esimerkiksi varmistaa, ettei työpisteen lämpötila kasva tietyn rajan yli. Itsenäisten komponenttien ei aina tarvitse olla tietoisia toisistaan, mutta ylemmällä tasolla halutaan varmistaa, että tuotantoprosessissa kaikki komponentit toimivat halutulla tavalla. Teollisuusjärjestelmien kompleksisuuden ja tuotantotehokkuuden vuoksi niitä tyypillisesti ohjataan erilaisilla automaatiojärjestelmillä. Automaatiojärjestelmä voi koostua erilaisista laitteista, jotka voivat ohjata tai valvoa teollisuusjärjestelmän komponenttien tilaa. Hierarkiatasolla laitteet voivat olla yhdistettynä toisiinsa sähkökytkennällä tai ylemmällä tasolla sijaitsevaan yksikköön. Hierarkiatason ylimmällä tasolla on tyypillisesti keskitetty valvomoyksikkö, josta voi seurata tai ohjata koko automaatiojärjestelmää.

Automaatiojärjestelmän eri tasoilla sijaitsevat laitteet sekä valvomoyksikkö voivat olla automaatiojärjestelmästä riippuen joko yksinkertaisia antureita, tiettyyn tarkoitukseen valmistettuja laitteita (ts. sulautettuja laitteita) tai yleiskäyttöisiä tietokoneita. Kompleksisuudesta riippuen laitteissa voidaan suorittaa ohjelmia tai Windowsin kaltaisille käyttöjärjestelmille tehtyjä graafisia työpöytäsovelluksia. Kaikki ohjelmat voivat kaatua ohjelmointivirheen takia. Windowsilla graafinen käyttöliittymä voi ilmoittaa käyttäjälle ohjelman kaatumisesta ja mahdollisista virheviesteistä tai -koodeista. Ohjelma kaatuu, jos ohjelma suorittaa jonkin laittoman operaation, kuten luvun jakamisen nollalla. Käyttöjärjestelmä reagoi tähän lähettämällä ohjelmalle signaalin, jonka jälkeen ohjelma suljetaan. Kaatumisen yhteydessä voidaan luoda kaatumisvedos, johon tallennetaan ohjelman muistin tila. Kaatumisvedoksia voidaan analysoida virheenjäljittäjällä (*debugger*), jolloin koodista voidaan selvittää ohjelman kaatumisen syy.

Kaatumisvedokset voidaan tallentaa tietokoneen tiedostojärjestelmään, josta ne voidaan lähettää ohjelman kehittäjälle Internetin yli. Vaihtoehtoisesti kaatumisvedokset voidaan siirtää tietokoneelta ulkoiselle muistille. Automaatiojärjestelmän sulautetuissa laitteissa kaatumisvedosten lähettäminen ei aina ole helppoa. Sulautetuissa laitteissa ei ole välttämättä käyttöliittymää tai sovittimia ulkoiselle muistille. Ne voivat myös olla paikoissa, joihin ihmisten on vaikea päästä. Tämän lisäksi ohjelman kaatumista ei ole aina helppo huomata. Automaatiojärjestelmän laitteet on voitu suunnitella vikasietoiseksi, jolloin yhden laitteen ohjelman kaatuminen ei välttämättä vaikuta merkittävästi automaatiojärjestelmän toimintaan. Laite voi myös automaattisesti pyrkiä käynnistymään uudelleen ohjelman kaatuessa. Ilman käyttöliittymää laitteet eivät voi ilmoittaa ohjelman kaatumisesta.

Tässä diplomityössä tutkitaan ja suunnitellaan, miten erääseen olemassa olevaan automaatiojärjestelmään voidaan integroida sulautettujen laitteiden kaatumisvedosten raportointijärjestelmä. Automaatiojärjestelmä ohjaa ja seuraa erästä teollisuusjärjestelmää, jossa teollisuusjärjestelmän on toimittava ilman merkittäviä vasteaikoja tai häiriöitä. Työssä keskitytään, miten sulautettujen laitteiden kaatuminen huomataan, miten se ilmoitetaan käyttäjälle, miten kaatumisvedokset saadaan talteen ja miten ne lähetetään kehittäjille hyvää käytettävyyttä unohtamatta. Työn tutkimuskysymys on siis seuraava:

- Miten kaatumisvedosten raportointi voidaan integroida osaksi automaatiojärjestelmän toimintaa häiritsemättä automaatiojärjestelmän reaaliaikaista toimintaa?

Luvussa 2 on taustatutkimusta kohdeautomaatiojärjestelmästä. Kohdeautomaatiojärjestelmästä kuvaillaan yleinen toimintaperiaate ja eri komponentit. Tämän lisäksi kuvaillaan tämän diplomityön tarkemmat tavoitteet.

Luvussa 3 kerrotaan tarkemmin kaatumisvedoksista. Luvussa kerrotaan, mitä kaatumisvedokset ovat ja miten niitä voidaan käsitellä.

Luvussa 4 käsitellään tietoliikennetekniikat, joiden pohjalta kohdeautomaatiojärjestelmän kommunikaatioprotokollat on kehitetty. Lisäksi esitellään tietoliikennetekniikat, joiden avulla kaatumisvedosten siirtäminen on mahdollista.

Luvussa 5 esitetään kohdeautomaatiojärjestelmän räätälöidyt protokollat ja kommunikaatiotekniikat. Luvussa käsitellyt asiat esitellään luvun 4 tietojen perusteella.

Luvussa 6 esitellään kaatumisvedosten raportointiprosessi edellisten lukujen tietojen perusteella. Luvussa kerrotaan, miten kaatumisvedoksia käsitellään sulautetuilla laitteilla ja miten ne raportoidaan kehittäjille Internetin yli.

2. TYÖN TAUSTA JA KONTEKSTI

Tässä diplomityössä suunnitellaan, miten erään automaatiojärjestelmän laitteiden muodostamien kaatumisvedosten raportointi voidaan integroida osaksi automaatiojärjestelmän toimintaa. Tässä luvussa esitellään tarkemmin kohdeautomaatiojärjestelmän toiminta, taustatutkimusta työlle sekä vähimmäisvaatimukset kaatumisvedosten raportoinnille.

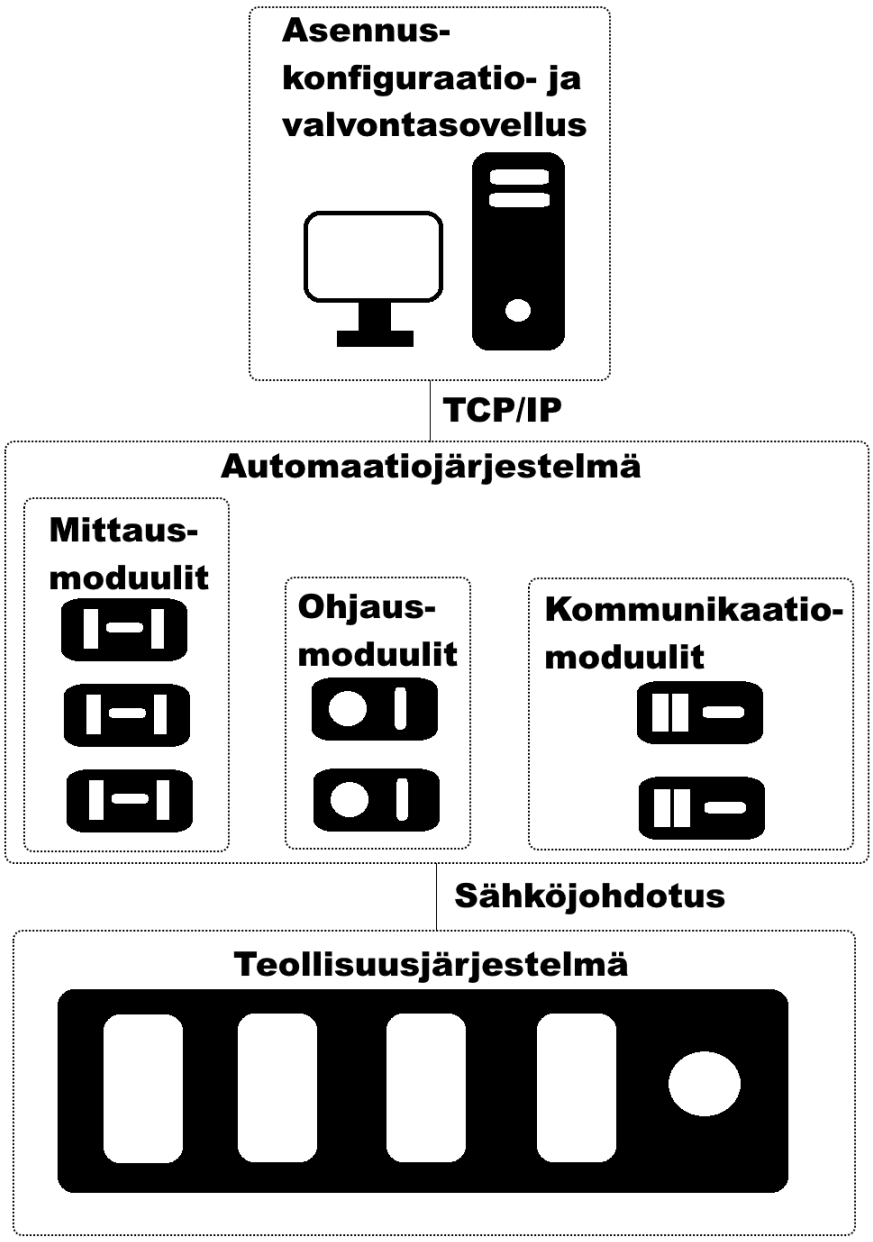
Tiivistettynä kohdeautomaatiojärjestelmä (tästä lähtien kohdejärjestelmä) koostuu laitteista, joilla suoritettava ohjelma ohjaa erästä teollisuusjärjestelmää. Erillisellä Windows-työpöytäsovelluksella voidaan konfiguroida kohdeautomaatiojärjestelmän laitteet.

2.1 Automaatiojärjestelmä

Automaatiojärjestelmät ovat suunniteltu niin, ettei niissä tarvita ihmisiä. Nykyajan teollisuusjärjestelmät ovat niin monimutkaisia, että ne tyypillisesti halutaan jakaa pienempiin ja helpommin hallittaviin kokonaisuuksiin, joita automaatiojärjestelmä käsittelee. Tässä diplomityössä käsiteltävä kohdejärjestelmä seuraa ja ohjaa erästä teollisuusjärjestelmää. Kohdejärjestelmä koostuu pienistä sulautetuista laitteista, jotka ohjaavat suurempaa mekaanista teollisuusjärjestelmää.

Kohdejärjestelmään kuuluu myös Windows-työpöytäsovellus, joka ei kuitenkaan osallistu teollisuusjärjestelmän ohjaamiseen. Työpöytäsovelluksella voidaan konfiguroida kohdejärjestelmän toimintaa, säätää ajettavan ohjelman parametreja, seurata kohdejärjestelmän tilaa ja asentaa ja päivittää ohjelmistoja sulautetuille laitteille.

Kohdejärjestelmä voidaan ajatella modulaarisena kodin seurantajärjestelmänä. Kotiin voidaan asentaa pieniä moduuleja, jotka voivat seurata ja ohjata kodin laitteiden toimintaa. Käyttäjä voi esimerkiksi asentaa jokaiseen huoneeseen mittausmoduulin, jonka vastuulla on huoneen lämpötilan ja kosteuden mittaaminen. Kaikissa kodin huoneissa voi olla yhteinen ilmastointi- ja ilmankostutinlaite, johon käyttäjä voi asentaa ohjausmoduulin. Mittausmoduulit ja ohjausmoduulit voivat olla yhteydessä kommunikaatiomoduuliin, jonka tehtävänä on ylläpitää moduulien välistä kommunikaatiota ja toimia yhdyskäytävänä kodin lähiverkkoon. Samassa lähiverkossa käyttäjä voi konfiguroida seurantajärjestelmän toimintaa älypuhelinsovelluksella. Käyttäjä voi esimerkiksi asentaa säännön, että huoneen kosteusprosentin ollessa alle 30 prosenttia kodin ilmankostutinlaite aktivoidaan, kunnes kosteusprosentti on yli 40 prosenttia. Toinen sääntö voi olla, että ilman lämpötilan ollessa yli 25 °C kodin ilmastointilaite aktivoidaan, kunnes huoneen lämpötila on alle 22 °C. Seurantajärjestelmä ei vaadi älypuhelinsovellusta toimiakseen, mutta käyttäjä voi halutessaan seurata ilmankosteutta tai lämpötilaa älypuhelinsovelluksella.



***Kuva 1**. Kohdejärjestelmän rakenne.*

Kuvassa 1 on esitelty kohdejärjestelmän rakenne. Kohdejärjestelmän toiminta on verrattavissa edellä olevaan esimerkkiin. Työpöytäsovelluksen avulla voidaan asentaa uutta ohjelmistoa, konfiguroida ohjelmiston ajoparametreja ja valvoa järjestelmän toimintaa. Edellisen esimerkin tavoin myös kohdejärjestelmä on hyvin modulaarinen. Laitteita, tai tästä eteenpäin moduuleita, voidaan lisätä teollisuusjärjestelmän tarpeiden mukaan. Moduuleja voidaan korvata toisilla vastaavilla moduuleilla, mikäli esimerkiksi moduulissa havaitaan jokin vika. Todellinen työssä käsiteltävä kohdejärjestelmä ja teollisuusjärjestelmä ovat kuitenkin huomattavasti monimutkaisempia kuin edellä oleva esimerkki.

Tässä diplomityössä ei kuitenkaan esitetä kohdejärjestelmän tai teollisuusjärjestelmän toimintaa tarkasti, sillä työn pääpaino on moduuleissa ja niissä ajettavassa ohjelmistossa.

2.1.1 Moduulit

Kohdejärjestelmän moduulit ovat vähätehoisia sulautettuja laitteita. Tästä syystä kaikki moduuleilla ajettava ohjelmisto on räätälöity moduulien ja teollisuusjärjestelmän tarpeiden mukaan. Moduuleilla on räätälöity alkulatausohjelma (*bootloader*) ja Linux-käyttöjärjestelmän ydin (*kernel*). Moduuleilla on paljon muita räätälöityjä ohjelmistokomponentteja, mutta niiden toimintaan ei käydä läpi.

Alkulatausohjelma löytyy kaikista tietokoneista. Se lataa tietokoneeseen käyttöjärjestelmän ytimen ja aloittaa ytimen suorituksen [39]. Kohdejärjestelmän moduuleilta löytyy kuitenkin kaksi alkulatausohjelmaa. Ensimmäinen alkulatausohjelma tallennetaan moduulin pysyväismuistiin moduulin valmistuksen aikana. Sen tehtävä on ladata toinen alkulatausohjelma, jota sovellusohjelmoijat voivat kehittää. Toinen alkulatausohjelma lataa varsinaisen ajettavan ohjelman. Mikäli ajettava ohjelma on korruptoitunut, epävakaa tai se kaatuu jatkuvasti, moduuli pysyy toisessa alkulatausohjelmassa.

Moduulien Linux-ydin toimii yhdessä reaaliaikaisen Xenomai-ohjelmistokehyksen kanssa. Xenomai-ohjelmistokehyksessä reaaliaikaisuus taataan ajamalla reaaliaikaydintä Linuxin ytimen rinnalla [31]. Teollisuusjärjestelmän vaatimuksien vuoksi moduuleilta vaaditaan kovaa reaaliaikasuoritusta. Käytännössä reaaliaikaisuus voi olla kova, eli järjestelmän täytyy vastata pyyntöihin täysin pyydetyssä ajassa, tai pehmeä, eli järjestelmän on yleensä pystyttävä vastaamaan pyyntöihin pyydetyssä ajassa [25]. Kovassa reaaliaikaisuudessa pyyntöihin ei saa vastata liian myöhään tai liian aikaisin, muuten kyseessä on toimintavirhe. Pehmeässä reaaliaikaisuudessa myöhästyminen ei ole välttämättä toimintavirhe, kunhan pyyntö suoritetaan loppuun asti.

Käytännön esimerkkinä auton turvatyynyn laukeaminen luottaa reaaliaikasuoritukseen, jossa reaaliaikaisuuden on oltava kova. Kun auton ohjainlaite huomaa törmäysanturista auton törmäyksen, ohjainlaitteen on laukaistava turvatyyny tietyssä ajassa ennen kuin kuljettajan pää osuu rattiin. Myöhään lauennut turvatyyny on toimintavirhe.

Moduuleilla on käytännössä sama koodipohja, mutta moduuleilla ajettava ohjelmisto on räätälöity jokaiselle moduulille. Kaikilla moduuleilla on monia sovittimia, joista kytketään sähköjohdotus teollisuusjärjestelmän komponentteihin. Sovittimet eroavat moduulin käyttötarkoituksen mukaan. Jotkin sovittimet on tarkoitettu tietyn teollisuusjärjestelmän komponentin mittaamiseen ja jotkin ohjaamiseen. Tämän lisäksi jotkin moduulit on erikoistettu kommunikaatioon. Näillä kommunikaatiomoduuleilla on Ethernet-portti, josta voidaan kytkeä Ethernet-kaapeli esimerkiksi verkkokytkimeen. Tämän diplomityön kannalta voidaan tehdä erottelu kommunikaatiomoduulin ja muiden moduulien välille, sillä

tässä työssä keskitytään enimmäkseen kommunikaatiomoduulin toimintaan kaatumisvedosten siirtämisen kannalta.

Moduulit kommunikoivat keskenään räätälöidyllä kommunikaatioprotokollalla, joka on kehitetty Ethernet-tekniikan päälle. Moduulit ovat kiinnitettynä toisiinsa kahdella Ethernet-kaapelilla. Moduulien välillä käytetään myös redundanssisuusprotokollaa, jossa viestit lähetetään molempiin suuntaan. Tämä redundanssisuusprotokolla ei ole tämän diplomityön kannalta olennainen, joten sitä ei käsitellä tarkasti. Kommunikaatiota tarkastellaan tarkemmin luvuissa 4 ja 5.

2.1.2 Työpöytäsovellus

Kohdejärjestelmän työpöytäsovellus ei kuulu varsinaisen kohdejärjestelmän toimintaan, mutta se on merkittävässä osassa kohdejärjestelmän konfiguroinnissa. Ennen varsinaista konfiguraatiota moduuleille kehitetään ohjelmisto. Toinen alkulatausohjelma, Linuxin ydin ja ajettava ohjelma räätälöidään moduuleille kohdejärjestelmän ja teollisuusjärjestelmän tarpeiden mukaan.

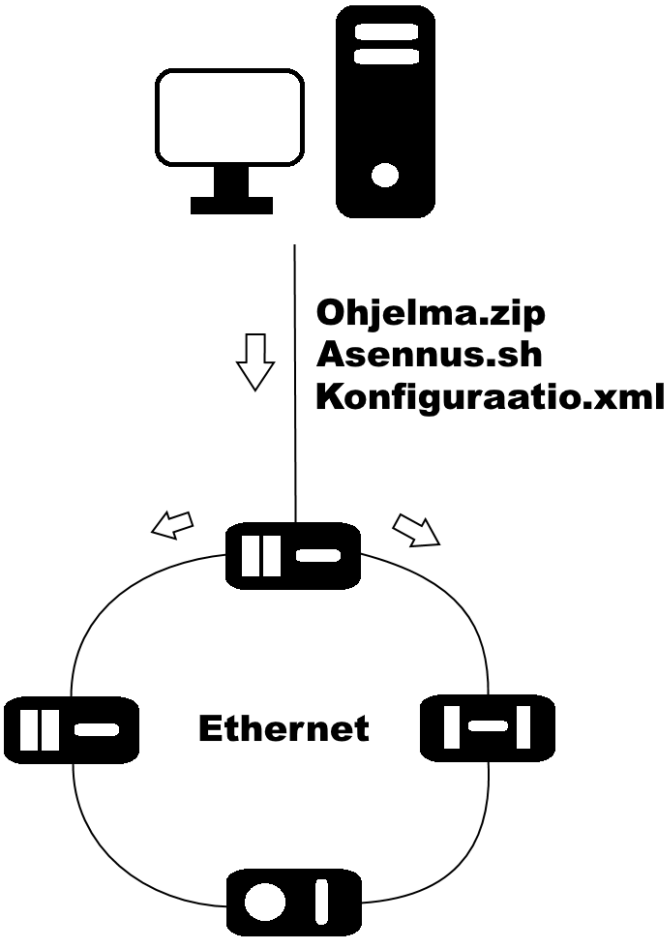
Työpöytäsovelluksella voidaan ottaa yhteyttä mihin tahansa kommunikaatiomoduuliin, kunhan työpöytäsovellus ja kommunikaatiomoduuli ovat samassa verkossa. Kohdejärjestelmässä voi olla useita kommunikaatiomoduuleja, mutta työpöytäsovelluksella voidaan olla yhteydessä kerralla vain yhteen kommunikaatiomoduuliin, jonka kautta kaikkien kohdejärjestelmän moduulien kommunikaatio kulkee.

Moduuleilla ajettava ohjelma noudattaa konfiguraatiopakettia, joka luodaan työpöytäsovelluksella. Konfiguraatiopaketissa määritellään ajettavan ohjelman parametrit. Kommunikaatiomoduuleille asetetaan muun muassa IP-osoite (*Internet Protocol*), aliverkon peite ja yhdyskäytävän osoite, jotta työpöytäsovelluksella voidaan kommunikoida kommunikaatiomoduulin kanssa. Kaikille moduuleille voidaan moduulien sovittimien mukaan konfiguroida teollisuusjärjestelmän mittaukseen ja ohjaamiseen tarvittavia arvoja.

Olennaisena osana kohdejärjestelmän toimintaa on ohjelmiston asennus moduuleille verkon yli. Työpöytäsovelluksessa voidaan valita, mitkä ohjelmistokomponentit asennetaan kohdejärjestelmän moduuleille. Valintojen perusteella työpöytäsovellus luo komentosarjatiedoston (*script file*), jossa moduuleja ohjataan asentamaan ohjelmistokomponentit. Työpöytäsovellus lähettää kaikkien moduulien komentosarjatiedostot sekä asennettavat ohjelmistokomponentit kommunikaatiomoduulille. Kommunikaatiomoduuli jakaa vastaanotetut ohjelmistokomponentit ja komentosarjatiedostot kaikille kohdejärjestelmän moduuleille, jonka jälkeen kaikki moduulit asentavat ohjelmistokomponentit itselleen. Kuvassa 2 on esitelty kyseinen toiminta.

Kuten edellä mainittiin, kohdejärjestelmä on hyvin modulaarinen. Mikäli yksi moduuli hajoaa, se voidaan korvata vastaavalla moduulilla. Kun kohdejärjestelmän muut moduulit

havaitsevat uuden moduulin, muilla moduuleilla oleva ohjelmisto asennetaan automaattisesti uudelle moduulille, jotta kaikilla kohdejärjestelmän moduuleilla on sama ohjelmisto.



***Kuva 2**. Asennusprosessi kohdejärjestelmän moduuleille.*

Latauksen ja asennuksen jälkeen kohdejärjestelmä ei tarvitse työpöytäsovellusta toimiakseen. Työpöytäsovelluksella voi kuitenkin muokata joitain ladatun konfiguraation arvoja dynaamisesti. Tätä ominaisuutta käytetään konfiguraatiopaketin testaus- ja kehitysvaiheessa.

2.2 Kaatumisvedosten noutamisen tarve ja toiminnallisuus

Kohdejärjestelmässä moduulit ovat täysin vastuussa teollisuusjärjestelmän ohjauksesta ja valvomisesta. Ohjattava järjestelmä ja kommunikaatio ovat hyvin monimutkaisia, joten moduulien ohjelmiston ohjelmoinnissa voi tapahtua ohjelmointivirheitä, jotka voivat joh-

taa ajettavan ohjelman kaatumiseen. Tästä syystä myös moduuleilta on hyvä kerätä kaatumisvedoksia, jotta saadaan selville ohjelman kaatumisen syy. Moduuleilla ajettavan Linux-käyttöjärjestelmän ydin voi myös kaatua.

Ethernet-tekniikan käyttö moduulien välillä on uutta. Aiemmissa kohdejärjestelmän sukupolvissa käytettiin kenttäväylätekniikoita, joilla tiedostonsiirto oli hyvin rajoitettua. Ethernet-tekniikan päälle rakennetun räätälöidyn kommunikaatioprotokollan myötä raportointijärjestelmä on ensimmäistä kertaa mahdollista toteuttaa. Kohdejärjestelmässä ei ole ennen ollut mitään helppoa keinoa saada kaatumisvedoksia moduuleilta. Koska kohdejärjestelmä on hyvin virhesietoinen, aina ei ole voitu edes tietää, ovatko moduulilla ajettavat ohjelmat kaatuneet.

Teollisuusjärjestelmän toimittajan kannalta kaatumisvedosten raportointi on arvokasta. Mikäli toimitettavassa ohjelmistossa on ohjelmointivirhe, joka aiheuttaa ajettavan ohjelman kaatumisen kriittisessä kohdassa, kaatumisesta muodostettava kaatumisvedos saadaan raportoitua nopeasti kehittäjille. Tällöin toimittaja voi toimittaa ohjelmiston korjaukset kaikille asiakkaille. Kustannusvaikutukset voivat olla suuret, sillä raportointijärjestelmällä voidaan parhaimmillaan minimoida monen teollisuusjärjestelmän häiriöaika.

Kaatumisvedokset voidaan tallentaa moduulien tiedostojärjestelmään, mutta suurempi ongelma on ratkaista, miten kaatumisvedokset saadaan kaikilta laitteilta talteen ja miten kaatumisvedokset voidaan lähettää eteenpäin kehittäjille. Monet automaatiojärjestelmän moduulit on kiinnitetty teollisuusjärjestelmän osiin, joihin ihmisten on vaikea päästä. Tämän lisäksi tietoturvan kannalta on tärkeää, että vain kehittäjillä on pääsy moduulien tiedostojärjestelmään. Vain kehittäjät voivat tarkastella tai ladata tiedostoja tiedostojärjestelmästä joko verkon yli tai sarjaportilla. Automaatiojärjestelmän loppukäyttäjillä ei ole oikeuksia moduulien tiedostojärjestelmään.

Kaatumisvedosten tarkistamiseen ja noutamiseen on oltava helppokäyttöinen mekanismi, jolla automaatiojärjestelmän loppukäyttäjät voivat lähettää moduulien kaatumisvedokset suoraan kehittäjille. Kaatumisvedosten tarkistus, noutaminen ja edelleen lähetys on luontevinta tehdä automaatiojärjestelmän työpöytäsovelluksella, sillä työpöytäsovelluksella konfiguroidaan sulautettujen moduulien toiminta sekä ladataan ajettava ohjelmisto laitteille. Mutta kuten sanottua, työpöytäsovellus ei ole aktiivisesti yhteydessä tai osana kohdejärjestelmän toimintoa, joten moduulien on säilytettävä kaatumisvedoksia tarpeen mukaan pitkiä aikavälejä. Kohdejärjestelmän tilaa voidaan myöhemmin seurata ottamalla kohdejärjestelmään yhteyttä työpöytäsovelluksella esimerkiksi huoltotoimenpiteiden yhteydessä.

2.3 Vähimmäisvaatimukset toiminnallisuudelle ja tavoitteet

Kohdejärjestelmän moduulien on muodostettava kaatumisvedoksia ajettavan ohjelman tai Linux-käyttöjärjestelmän ytimen kaatuessa. Minimitoiminnallisuudessa molemmat

kaatumisvedostyypit on siirrettävä kaikilta kohdejärjestelmän moduuleilta tietokoneelle, jossa työpöytäsovellusta ajetaan. Työpöytäsovellus on yhteydessä vain yhteen kommunikaatiomoduuliin kerralla, joten kaatumisvedokset siirretään kyseisen moduulin kautta.

Moduuleille on varattu säilytettävälle datalle hyvin rajattu määrä tilaa. Kohdejärjestelmä voi kuitenkin olla käytössä vuosia, joten säilytyksen kanssa on oltava tarkkana. Kaatumisvedoksia ei voida kerätä yhdelle kommunikaatiomoduulille, johon työpöytäsovellus on yhteydessä, sillä kommunikaatiomoduulilla ei ole välttämättä tarpeeksi tilaa kaikkien moduulien kaatumisvedosten säilyttämiseen kerralla. Jokaisen moduulin on siis säilytettävä omat kaatumisvedoksensa, kunnes ne siirretään. Kaatumisvedoksia täytyy myös pakata tilan säästämiseksi. Siirron jälkeen kaatumisvedokset on poistettava kaikilta moduuleilta.

Kaatumisvedokset on arkistoitava yhteen tiedostoon, joka nimetään moduulin mukaan, jotta kaatumisvedosten käsittelyssä nähdään nopeasti, mistä moduulista kaatumisvedokset tulivat. Tiedostonsiirrossa toiselta moduulilta voidaan pyytää tiedosto, mikäli tiedetään tiedoston polku ja nimi. Tiedostonsiirron yksityiskohdat selitetään aliluvussa 6.4.

Kaatumisvedosten käsittely ja siirto ei saa häiritä kohdejärjestelmän reaaliaikaista toimintaa. Moduuleilla on hyvin rajattu määrä resursseja, joten prosessorikuorma ei saa nousta liian korkeaksi. Korkea prosessorikuorma voi haitata laitteiden reaaliaikasuorituksessa olevaa ohjelmaa tai muuta toimintaa. Korkea prosessorikuorma voi aiheuttaa esimerkiksi aikakatkaisuja moduulien kommunikaatiossa.

Työpöytäsovelluksen täytyy tietää, millä moduuleilla kaatumisvedoksia löytyy, jotta työpöytäsovellus tietää, miltä moduuleilta kaatumisvedoksia täytyy hakea. Kaatumisvedoksista on ilmoitettava käyttäjälle. Koska kohdejärjestelmä toimii itsenäisesti ilman työpöytäsovellusta, työpöytäsovelluksen täytyy ilmoittaa käyttäjälle löytyneistä kaatumisvedoksista heti, kun työpöytäsovelluksesta otetaan yhteyttä mihin tahansa kohdejärjestelmän kommunikaatiomoduuliin.

Kun työpöytäsovellus on tallentanut kaikkien moduulien kaatumisvedokset, ne täytyy lähettää kehittäjille. Lähetyksen on tapahduttava mahdollisimman huomaamattomasti, mutta lähetyksen tilasta on annettava riittävästi tietoa käyttäjälle.

3. KAATUMISVEDOKSET JA NIIDEN KÄSITTELY

Tässä luvussa kerrotaan, mitä kaatumisvedokset ovat. Kaatumisvedoksia muodostetaan, mikäli moduuleilla ajettava ohjelma kaatuu tai Linux-käyttöjärjestelmän ydin kaatuu.

Tämän lisäksi kerrotaan kaatumisvedosten käsittelystä. Kaatumisvedokset ovat moduulien tallennustilan määrään verrattain isoja tiedostoja, joten ne täytyy pakata tilan säästämiseksi.

3.1 Ajettavan ohjelman kaatuminen

Tietokoneissa ajettava ohjelma ladataan väliaikaiseen RAM-muistiin (*Random Access Memory*), josta tietokoneen prosessori hakee dataa suoritukseen. Muistissa olevaa ohjelmaa voidaan analysoida interaktiivisesti virheenjäljittäjällä, kuten esimerkiksi GDB-ohjelmalla (*GNU Debugger*). Virheenjäljittäjällä voidaan esimerkiksi pysäyttää ohjelman suoritus haluttuun kohtaan koodissa ja tarkastella muuttujien arvoja. Virheenjäljittäjällä saadaan selville myös *call trace*, joka on raportti ohjelman suorituksen pinon rakenteesta. [28, s. 47–59]

Useimmilla moderneilla käyttöjärjestelmillä on mahdollista tallentaa ajossa olevan ohjelman muisti tiedostoon. Tätä tiedostoa kutsutaan tyypillisesti muistivedokseksi. Tallennetussa muistivedoksessa voi olla täydellinen kuva kaikesta RAM-muistin datasta tai tietyn ohjelman muistialueen datasta. GDB-ohjelman kaltaisella virheenjäljittäjällä muistivedosta voidaan analysoida samalla tavalla kuin ohjelma olisi vielä RAM-muistissa. Muistin tallentaminen voi olla hyödyllistä esimerkiksi silloin, kun ohjelman suoritus on jäätynyt. [34]

Ohjelman kaatuessa muodostettavaa muistivedosta kutsutaan kaatumisvedokseksi [34]. Linux-käyttöjärjestelmällä *rlimit*-tietueella (*resource limits*) voidaan määrittää, kuinka paljon ja mitä resursseja ydin sallii ohjelman tekevän [27, s. 204–205]. Tietueella voidaan esimerkiksi määrittää, kuinka suuria kaatumisvedoksia ohjelma saa luoda. Mikäli ohjelma suorittaa laittoman operaation, käyttöjärjestelmän ydin lähettää ohjelmalle signaalin, joka lopettaa ohjelman suorituksen [27, s. 305–306]. Esimerkiksi ohjelma voi yrittää kirjoittaa dataa muistialueelle, joka on kirjoitussuojattu. Tällöin käyttöjärjestelmän ydin lähettää ohjelmalle SIGSEGV-signaalin, jonka jälkeen ohjelma lopettaa suorituksensa [27, s. 109]. Ohjelma voi ottaessaan vastaan signaalin tallentaa muistinsa kaatumisvedokseen, mikäli kaatumisvedoksen tallennus on *rlimit*-tietueessa sallittu. Ohjelmassa voi olla erillinen signaalinkäsittelykomponentti, joka kuuntelee käyttöjärjestelmän lähettämiä signaaleja [22, s. 416–417].

```
 1 #include <stdio.h>
 2
 3 void getWorld(char* b)
 4 {
 5     sprintf(b, "%s", "world");
 6 }
 7
 8 void printHelloWorld(void)
 9 {
10     char* a = "hello";
11     char* b;
12     getWorld(b);
13     printf("%s %s\n", a, b);
14 }
15
16 int main(int argc, char* argv[])
17 {
18     printHelloWorld();
19     return 0;
20 }
program_crash $ulimit -c unlimited
program_crash $make
gcc -g test.c -o test
program_crash $./test
Segmentation fault (core dumped)
program_crash $gdb test core
Reading symbols from test...done.
[New LWP 13735]
Core was generated by `./test'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000400532 in getWorld (b=0x0) at test.c:5
5           sprintf(b, "%s", "world");
(gdb) bt
#0  0x0000000000400532 in getWorld (b=0x0) at test.c:5
#1  0x000000000040055d in printHelloWorld () at test.c:12
#2  0x000000000040058e in main (argc=1, argv=0x7ffe90484158) at test.c:18
(gdb)
```

***Kuva 3**. Esimerkki kaatuvasta ohjelmasta ja kaatumisvedoksen tutkimisesta GDB-ohjelmalla.*

Kuvassa 3 on kaatuva ohjelma. Alemmassa osassa *ulimit*-komennolla, joka käyttää sisäisesti *rlimit*-tietuetta, sallitaan ohjelmien luoda rajattoman kokoisia kaatumisvedoksia [27, s. 208]. Itse koodissa *char*-osoittimelle *b* ei ole varattu muistialuetta, johon merkkijono voitaisiin kirjoittaa. Näin ollen *sprintf* yrittää kirjoittaa merkkijonon kirjoitussuojattuun muistialueeseen. Käyttöjärjestelmän ydin lähettää ohjelmalle SIGSEGV-signaalin, joka lopettaa ohjelman suorituksen. Sen jälkeen ohjelma luo kaatumisvedoksen. Kun kaatumisvedosta tutkitaan GDB-ohjelmalla, nähdään, millä ohjelman rivillä ohjelma kaatui, kaatumisen syy sekä *call trace*.

Ohjelmoinnissa voidaan tehdä muitakin virheitä, jotka johtavat ohjelman kaatumiseen. Eri tyyppisille virheille on eri signaaleja, jotka helpottavat kaatumisen syyn selvittämistä. Matemaattisissa operaatioissa voidaan tehdä vahingossa luvun jakaminen nollalla, jolloin käyttöjärjestelmä lähettää ohjelmalle SIGFPE-signaalin. Toisaalta ohjelma voi yrittää suorittaa käyttöjärjestelmän järjestelmäkutsun laittomilla parametreilla, jotka voisivat

vaikuttaa käyttöjärjestelmän toimintaan. Tällöin käyttöjärjestelmä lähettää ohjelmalle SIGSYS-signaalin. [6, s. 138–140]

Kaatumisvedos itsessään ei sisällä tietoa kirjoitetusta koodista. Tietokoneilla ajettavat ohjelmat koostuvat tietokoneen prosessorin ymmärtävistä konekäskyistä. Prosessori ei osaa ajaa suoraan esimerkiksi C- ja C++-kielten koodia. Sen sijaan koodi käännetään konekäskyiksi. Ohjelman kaatuessa kaatumisvedokseen tallennetaan tietokoneen väliaikaisessa RAM-muistissa olevat konekäskyt. [38, s. 12–28]

GDB-ohjelma ei osaa suoraan päätellä, mitkä konekäskyt vastaavat esimerkiksi C-kielellä kirjoitetun koodin eri osia. GCC-kääntäjälle (*GNU Compiler Collection*) on mahdollista antaa ylimääräinen parametri *-g*, jota käytetään myös kuvan 3 esimerkissä. Parametri lisää symbolitaulun ajettavaan ohjelmaan. Symbolitaulu kertoo, mitkä käännetyt konekäskyt vastaavat kutakin kirjoitetun koodin riviä. Tällöin GDB-ohjelma osaa kertoa, mikä koodin rivi aiheutti ohjelman kaatumisen. [18, s. 135–136]

Kaatumisvedosten analysoimiseen tarvitaan vähintään itse kaatumisvedos, sitä vastaava ajettava ohjelma ja ajettavaa ohjelmaa vastaava lähdekoodi. Symbolitaulun avulla GDB-ohjelma voi näyttää *call tracen* rivinumerot kuvan 3 esimerkin tavoin, mutta jos lähdekoodi ei vastaa käännettyä ohjelmaa, rivinumerot ovat hyödyttömiä. Tästä syystä kaatumisvedosten tutkimisessa on oltava lähdekoodi, josta kaatunut ohjelma on alun perin käännetty. Tähän ongelmaan palataan aliluvussa 6.1.

3.2 Ytimen kaatuminen

Linux-käyttöjärjestelmän ytimeen voidaan ladata dynaamisesti ajettavia moduuleita. Esimerkiksi laiteajurit ja tiedostojärjestelmät ovat ajettavia moduuleita. Moduulien täytyy olla dynaamisesti linkitettyjä, sillä staattisesti linkitetyt moduulit kasvattaisivat ytimen kokoa liikaa. Kuten kaikki muutkin ohjelmat, ytimen moduulit voivat kaatua ohjelmointivirheen takia. [36, s. 80–81]

Jos ydintä ajettaessa huomataan ongelma, joka johtaisi ytimen kaatumiseen, ydin pyrkii lopettamaan ongelmallisen prosessin tai moduulin ja raportoimaan virheestä. Tämä virheraportti on nimeltään *kernel oops*. Ydin ei ole välttämättä niin epävakaassa tilassa, että ytimen suoritus olisi lopetettava. [32, s. 93–95]

Kuvassa 4 on esitelty moduuli, joka aiheuttaa ytimessä *kernel oopsin*. Virheraportin alussa esitetään lyhyesti virheen luonne *BUG*-osassa, joka kertoo, että moduuli yritti hakea *NULL*-osoittimen päässä olevan arvon. *IP*-kenttä kertoo, missä *instruction pointer* oli virheen tapahtuessa. Alemmista kentistä löytyy muun muassa moduulin *call trace*, ajettavien ohjeiden osoitteet prosessorin rekistereistä ja lista ytimen funktioista, jotka ajettiin ennen moduulin koodia. Vaikka virheraportti ei ole sama kuin kaatumisvedos, virhera-

portin avulla voidaan analysoida ytimen moduulia GDB-ohjelmalla annettujen muistiosoitteiden avulla. Virheraportti voidaan siis mieltää kaatumisvedokseksi, sillä sitä käsitellään samankaltaisesti, kuin ajettavan ohjelman kaatumisvedoksia. [17]

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static void buggy_function(void)
{
    *(int*)0 = 0;
}

static int __init oops_init(void)
{
    printk("Initializing kernel module\n");
    buggy_function();
    return 0;
}

static void __exit oops_cleanup(void)
{
    printk("Cleaning up module.\n");
}

module_init(oops_init);
module_exit(oops_cleanup);
```

```
[  100.876180] Initializing kernel module
[  100.876185] BUG: unable to handle kernel NULL pointer dereference at           (null)
[  100.876236] IP: oops_init+0x10/0x1000 [test]
[  100.876266] PGD 0
[  100.876317] Oops: 0002 [#1] SMP
[  100.876592] CPU: 0 PID: 2359 Comm: insmod Tainted: P  OE   4.10.0-28-generic #32~16.04.2-Ubuntu
[  100.876691] task: ffff913af539ad00 task.stack: ffff9c82429dc000
[  100.876727] RIP: 0010:oops_init+0x10/0x1000 [test]
[  100.876757] RSP: 0000:ffff9c82429dfc98 EFLAGS: 00010282
[  100.876790] RAX: 000000000000001a RBX: ffffffffc0596000 RCX: 0000000000000006
[  100.876825] RDX: 0000000000000000 RSI: 0000000000000202 RDI: ffff913affc0dc80
[  100.876862] RBP: ffff9c82429dfc98 R08: 0000000000000001 R09: 00000000000001db
[  100.876897] R10: ffff913af53705e0 R11: 00000000000001db R12: ffffffffc03ab000
[  100.876945] R13: 0000000000000000 R14: ffff913af56c39c0 R15: 0000000000000001
[  100.876980] FS:  00007f3cd079d700(0000) GS:ffff913affc00000(0000) knlGS:0000000000000000
[  100.877019] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  100.877051] CR2: 0000000000000000 CR3: 00000000380d3000 CR4: 00000000000406f0
[  100.877090] Call Trace:
[  100.877118]  do_one_initcall+0x53/0x1c0
[  100.877151]  ? kmem_cache_alloc_trace+0x152/0x1c0
[  100.877183]  do_init_module+0x5f/0x1ff
[  100.877216]  load_module+0x1825/0x1bf0
[  100.877245]  ? __symbol_put+0x60/0x60
[  100.877276]  ? ima_post_read_file+0x7d/0xa0
[  100.877306]  ? security_kernel_post_read_file+0x6b/0x80
[  100.877341]  SYSC_finit_module+0xdf/0x110
[  100.877370]  SyS_finit_module+0xe/0x10
[  100.877402]  entry_SYSCALL_64_fastpath+0x1e/0xad
[  100.877431] RIP: 0033:0x7f3cd02c9499
[  100.877822] RSP: 002b:00007ffdc43c3548 EFLAGS: 00000206 ORIG_RAX: 0000000000000139
[  100.878207] RAX: ffffffffffffffda RBX: 00007f3cd058cb20 RCX: 00007f3cd02c9499
[  100.878596] RDX: 0000000000000000 RSI: 000055c3e628c26b RDI: 0000000000000003
[  100.878990] RBP: 0000000000001011 R08: 0000000000000000 R09: 00007f3cd058eea0
[  100.879379] R10: 0000000000000003 R11: 0000000000000206 R12: 00007f3cd058cb78
[  100.879756] R13: 00007f3cd058cb78 R14: 00000000000270f R15: 00007f3cd058d1a8
[  100.880145] Code: <c7> 04 25 00 00 00 00 00 00 00 00 31 c0 5d c3 00 00 00 00 00 00 00
[  100.880519] RIP: oops_init+0x10/0x1000 [test] RSP: ffff9c82429dfc98
[  100.880884] CR2: 0000000000000000
```

***Kuva 4**. Esimerkki kaatuvasta ytimen moduulista ja kernel oops -virheviestistä.*

Ytimen *kernel oops* ei ole aina vakava virhe. Ydin voi jatkaa toimintaansa normaalisti, mikäli ytimen prosessi voidaan turvallisesti lopettaa, kuten kuvassa 4. Prosessi voi tosin olla ytimen kannalta ehdoton. Mikäli lopetettava prosessi onkin tiedostojärjestelmä, käyttöjärjestelmä ei voi palautua virheestä. Tällainen virhetilanne on nimeltään *kernel panic*.

Siinä virheviesti on vastaava kuin *kernel oopsissa*, eli virheviestistä nähdään lyhyesti kaatumisen syy ja *call trace*. *Kernel panicissa* ydin on epävakaassa tilassa, joten käyttöjärjestelmän normaalia suoritusta ei jatketa, sillä epävakaassa tilassa käyttöjärjestelmän data voi korruptoitua. [30, s. 119–126][36, s. 87]

3.3 Kaatumisvedosten tallennus ja pakkaaminen

Ajettavan ohjelman kaatumisvedokset tallennetaan oletuksena tiedostojärjestelmään. Kuvan 3 esimerkin mukaan kaatumisvedos tallennetaan samaan sijaintiin kuin ajettava ohjelma. Linux-käyttöjärjestelmässä on mahdollista määrittää kaatumisvedoksen nimi ja tiedostosijainti tarpeen mukaan [22, s. 449].

Ytimen kaatumisvedoksien tallentamiseen voidaan käyttää Linux-käyttöjärjestelmän MTD-alijärjestelmää (*Memory Technology Device*), joka abstrahoi käytettyä flash-muistia [36, s. 167–168]. Käytetty flash-muisti halutaan tyypillisesti jakaa eri osioihin, joihin voidaan sijoittaa esimerkiksi alkulatausohjelma tai tiedostojärjestelmä. Ytimen virheraporteille voidaan myös luoda oma MTD-osio [36, s. 172–173]. Ytimen virheraportit halutaan talteen ytimen puskurimuistista, ennen kuin tietokone käynnistyy uudelleen ja kadottaa virheraportit. Linux-käyttöjärjestelmä tarjoaa tähän *MTDoops*-moduulin, joka kirjoittaa virheraportit MTD-osiolle turvallisesti ennen tietokoneen uudelleenkäynnistystä.

Kaatumisvedokset halutaan pakata tallentamisen jälkeen, sillä kaatumisvedosten koko voi ohjelman suorituskohdasta riippuen olla suuri. Tiedonpakkauksessa tiedosto pyritään kuvaamaan vähemmällä bittimäärällä. Tiedonpakkaus voi olla häviöllistä tai häviötöntä. Häviöllisessä tiedonpakkauksessa tiedostosta säilytetään vain tärkein data. Tarpeeton tai vähemmän tärkeä data hävitetään. Häviöttömässä pakkauksessa mitään dataa ei hävitetä. Tiedoston on oltava sama ennen ja jälkeen pakkaamisen. Häviöllistä pakkausmenetelmää käytetään esimerkiksi kuvissa ja videoissa, sillä niissä dataa voidaan poistaa niin, että kuva tai video näyttää ihmissilmälle samalta. [3]

Käännetyissä ohjelmissa tai kaatumisvedoksissa kaikki data on tärkeää, sillä ilman osaa dataa ohjelma on korruptoitunut ja kaatumisvedoksesta ei saada järkevää *call tracea*. Tästä syystä kaatumisvedosten pakkaamiseen ei voida käyttää häviöllistä pakkausmenetelmää.

ZIP-tiedostonpakkausmenetelmässä käytetään häviötöntä tiedonpakkausta, jossa datasta pyritään tunnistamaan redundanssisuus ja esittämään redundantti data lyhyemmin. Käytännössä datasta etsitään identtisiä tavuja ja niiden esiintymistiheyksiä. Pakatussa datassa kerrotaan, kuinka monta kertaa identtinen tavu esiintyy tietyissä kohdissa. ZIP-pakkauksessa käytetään DEFLATE-algoritmia, joka on yhdistelmä LZ77-algoritmia ja Huffmanin koodausta. [24]

Tiedostoja voidaan lisätä zip-tiedostoihin myös ilman pakkausta. Tätä operaatiota kutsutaan arkistoinniksi (*archiving*). Alun perin Unix-järjestelmissä käytetty tar-formaatti kehitettiin vain tiedostojen arkistoimiseen. Vanhat nauha-asemat pystyivät lukemaan vain tietyn kokoisia datalohkoja, joten pienempien datalohkojen lukeminen ei hyödyntänyt lukuoperaatiota täysin. Lisäämällä kaiken datan yhteen tiedostoon lukuoperaatioita pystyttiin nopeuttamaan. [15]

Nykyään samalla periaatteella, mutta eri toimintatavalla, tiedostoja voidaan haluta lisätä zip- tai tar-arkistotiedostoihin, mikäli tiedostot ovat jo pakattuja tai tiedostot halutaan lisätä yhteen arkistotiedostoon nopeasti. Tietokoneen I/O-operaatioissa (*Input/Output*) yhden ison tiedoston siirtäminen on tehokkaampaa kuin monen pienen tiedoston siirtäminen. Monen tiedoston siirtämisessä jokaiselle tiedostolle on suoritettava tiedostojärjestelmästä ja käyttöjärjestelmästä riippuen monia I/O-operaatioita, jotka sisältävät muun muassa monia raskaita järjestelmäkutsuja. Yhdellä arkistoidulla tiedostolla käyttöjärjestelmän suorittamat operaatiot joudutaan tekemään vain kerran, joten tiedonsiirto on nopeampaa. [22, s. 233–244]

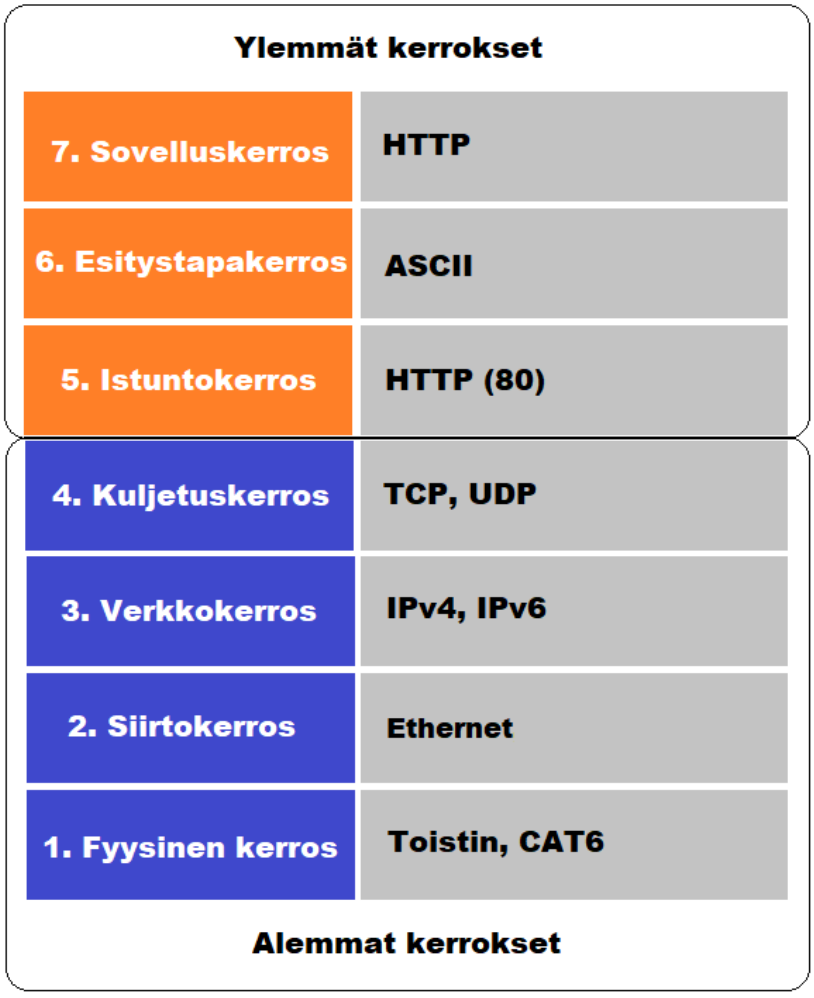
4. TIETOLIIKENNETEKNIIKAT

Tässä luvussa esitellään tietoliikennetekniikat, joita hyödynnetään kohdejärjestelmään kehitellyissä protokollissa. Lisäksi esitellään tietoliikennetekniikat, joita tarvitaan kaatumisvedosten siirtämisessä.

Aluksi käsitellään OSI-malli (*Open Systems Interconnection model)*, jonka pohjalta käydään läpi tietoliikennetekniikat ja -protokollat myöhemmissä aliluvuissa. Ne käsitellään mallin alimmasta kerroksesta ylimpään kerrokseen.

4.1 OSI-malli

Tietoliikennetekniikat voidaan esittää konseptuaalisesti OSI-mallissa, joka kuvaa eri tietoliikennetekniikoita, laitteita ja protokollia [20]. Tällä hetkellä OSI-malli voidaan jakaa seitsemään eri kerrokseen. OSI-malli voidaan esittää kuvan 5 tavalla.



***Kuva 5**. OSI-malli [35, s. 275–289][13, s. 47–48].*

OSI-mallissa seuraava taso rakentuu edellisen tason päälle. Alimmalla tasolla oleva fyysinen kerros vaaditaan kaikkiin sitä ylempiin kerroksiin, sillä ilman fyysisiä laitteita tietoliikenne ei voi toimia. OSI-mallissa viestin lähettäminen etenee ylimmästä kerroksesta alimpaan kerrokseen ja viestin vastaanottaminen etenee alimmasta kerroksesta ylimpään kerrokseen. Käytännössä jokainen kerros vastaanottaa datan, suorittaa kerroksen oman tehtävän ja välittää datan seuraavalle kerrokselle. Kerrokset 5-7 luokitellaan ylemmiksi kerroksiksi, sillä ne sisältävät datan, jota käsitellään korkean tason sovelluksissa. Kerrokset 1-4 luokitellaan alemmiksi kerroksiksi, sillä niissä keskitytään datan siirtämiseen. [35, s. 275–289][13, s. 47–48]

Seitsemäs kerros on sovelluskerros. Se on abstraktiotasolla korkeimpana. Sovelluskerros on myös lähimpänä käyttäjää, sillä siihen lasketaan tietoliikennetekniikat, jotka ovat suoraan tekemisissä tietokoneen sovellusohjelmien kanssa. Esimerkkinä sovelluskerroksen tietoliikenneprotokollasta on tiedonsiirtoprotokolla SFTP (*SSH File Transfer Protocol*), jota käsitellään aliluvussa 4.4. Sovellustasolla ohjelmoija voi esimerkiksi ladata tiedoston toiselta samassa verkossa olevalta tietokoneelta. [14, s. 30]

Kuudes kerros on esitystapakerros. Sen tehtävä on muuttaa sovelluskerroksen data neutraaliin muotoon siirtoa varten. Sen tehtävä voi olla esimerkiksi datan pakkaaminen tai tiedoston merkistökoodauksen muuttaminen ASCII-muotoon (*American Standard Code for Information Interchange*). [14, s. 20]

Viides kerros on istuntokerros. Se alustaa, ylläpitää ja purkaa yhteyden kahden prosessin välillä. Sen vastuulla on ylläpitää istunnon aikaisia palveluita, kuten kykyä palautua tiettyyn kohtaan istuntoa. [2, s. 73–88]

Neljäs kerros on kuljetuskerros. Se on OSI-mallissa tärkein kerros, sillä sen vastuulla on pakettien luotettava lähetys. Käytännössä tämä tarkoittaa pakettien lähetystä, uudelleen lähettämistä paketin hukkuessa ja pakettien järjestämistä oikeaan järjestykseen. Kuljetuskerros myös huolehtii vuonhallinnasta pakettien ruuhkien välttämiseksi. Esimerkiksi TCP-protokolla (*Transmission Control Protocol*) toteuttaa kuljetuskerroksen. Se huolehtii kaikista edellä mainituista tehtävistä. [13, s. 51][2, s. 73–88]

Kolmas kerros on verkkokerros. Sen vastuulla on pakettien reititys. Paketeilla voi olla useita eri reittejä tai kiinteä reitti määränpäähän. Käytännössä verkkokerroksessa jokaisella eri laitteella on verkko-osoite, joka toimii pakettien osoitteena. Jokainen paketti tietää, mikä on kohdelaitteen verkko-osoite, ja esimerkiksi reitittimet ohjaavat pakettia oikeaan suuntaan. [14, s. 21]

Toinen kerros on siirtokerros. Siihen kuuluu merkittävänä osana Ethernet-tekniikka, joka kuuluu myös osittain fyysiseen kerrokseen. Ethernet-tekniikasta kerrotaan enemmän seuraavassa aliluvussa. Siirtokerroksen vastuulla on datan siirto verkon eri solmujen välillä.

Verkon datasta muodostetaan loogisia paketteja, jotka välitetään verkkokerrokselle. Siirtokerroksen vastuulla on myös mahdollisten datan virheiden korjaus, sillä data voi korruptoitua siirrossa. [1][13, s. 52–53]

Ensimmäinen kerros on fyysinen kerros. Siihen kuuluvat laitteet, jotka siirtävät raakoja bittejä. Esimerkiksi CAT6-kaapeli ja toistimet kuuluvat fyysiseen kerrokseen. Toistimella voidaan esimerkiksi yhdistää kaksi kaapelia yhdeksi. [2, s. 73–88]

4.2 Ethernet

IEEE 802.3 (*Institute of Electrical and Electronics Engineers*) on lähiverkkotekniikka Ethernet-pohjaisiin lähiverkkoihin. Ethernet ei ole yksittäinen tekniikka, vaan se koostuu eri lähiverkkojen tekniikoista. Siihen kuuluvat muun muassa OSI-mallin fyysisen kerroksen kaapelointitekniikat ja toistimet. Lisäksi siihen kuuluu siirtokerroksen tekniikoita, kuten Ethernet-kehyksen hallitseminen. [13, s. 41]

Ethernetin fyysinen kerros ja siirtokerros kattavat eri osa-alueita. Seuraavissa aliluvuissa esitellään Ethernetin fyysinen kerros ja siirtokerros erikseen.

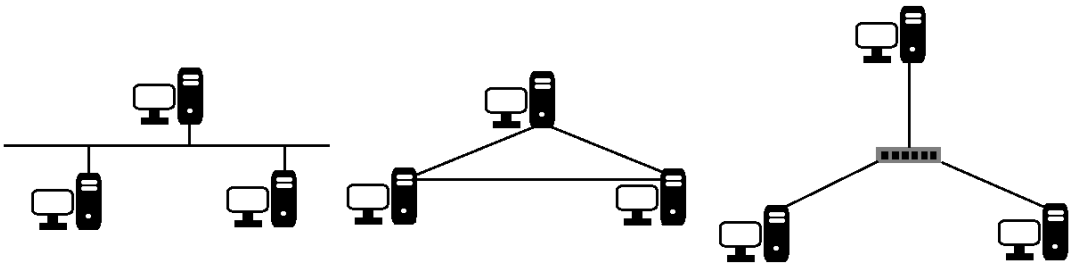
4.2.1 Ethernetin fyysinen kerros

Ethernet-laitteissa on oltava verkkosovitin, jotta ne voidaan liittää esimerkiksi lähiverkkoon. Verkkosovittimen avulla tietokone voi kommunikoida lähiverkossa. Se lähettää tai vastaanottaa raakaa dataa, joka vastaanotettaessa järjestellään verkkopaketeiksi ja tarpeen mukaan korjataan, mikäli data ei ole järkevässä muodossa. Jokaisella verkkosovittimella on MAC-osoite (*Medium Access Control*), joka yksilöi laitteen Ethernet-verkossa [13, s. 64–66].

Verkkosovittimen johdotuksessa käytetään yleisesti Ethernet-kaapelia [13, s. 126]. Se on parikaapelia, joka on valmistettu kuparista. Parikaapelointia käytetään ulkopuolisten häiriösignaalien minimoimiseksi. Ethernet-kaapeleissa on eri standardeja, mutta nykyään yleisin standardi on Cat5e, jolla saavutetaan 1000 Mt/s siirtonopeudet [4, s. 39–41].

Ethernet-tekniikassa käytetään toistimia ja kytkimiä verkkoliikenteen ohjaukseen. Toistin kuuluu OSI-mallissa fyysiseen kerrokseen. Se toistaa vastaanotetut paketit kaikille toistimeen liitetyille Ethernet-laitteille, mutta se ei tarkastele läpi kulkevaa dataa. Kytkin kuuluu OSI-mallissa siirtokerrokseen, joten sitä käsitellään seuraavassa aliluvussa. [37, s. 34]

Ethernet-laitteilla voidaan muodostaa erilaisia verkkotopologioita. Topologia tarkoittaa tapaa, miten laitteet ovat fyysisesti kaapeloitu [7, s. 132–135]. Perustopologioihin kuuluvat väylä, rengas ja tähti. Topologiat on esitelty kuvassa 6.



***Kuva 6**. Väylä-, rengas- ja tähtitopologia [7, s. 132–135].*

Väylätopologiassa kaikki laitteet on yhdistetty yhteen kaapeliin, jossa laitteet keskustelevat vuorosuuntainesti (*half duplex*). Mikäli kaikki väylätopologian laitteet yrittävät kommunikoida yhtä aikaa, verkko ruuhkaantuu. Vuorosuuntaisissa Ethernet-verkoissa on käytettävä CSMA/CD-menetelmää (*Carrier Sense Multiple Access With Collision Detection*), jossa törmäyksiä pyritään välttämään. Mikäli huomataan törmäys, odotetaan satunnainen ajankohta ja yritetään uudestaan. Kaksisuuntaisen (*full duplex*) verkkoliikenteen ansiosta myös CSMA/CD-menetelmä on tarpeeton tilanteissa, joissa laite haluaa lähettää ja vastaanottaa dataa samanaikaisesti. Nykyaikaisissa verkoissa ei juurikaan käytetä vuorosuuntaista verkkoa tai väylätopologiaa. [37, s. 1–10]

Rengastopologiassa Ethernet-verkossa olevat laitteet ovat yhteydessä niin, että ne muodostavat renkaan. Rengastopologiassa viestit lähetetään rengasta pitkin määritettyyn suuntaan. Huonona puolena rengastopologiassa on vikasietoisuus. Mikäli yksi laite ei lähetä viestejä eteenpäin, koko verkon toiminta häiriintyy. [7, s. 133–134]

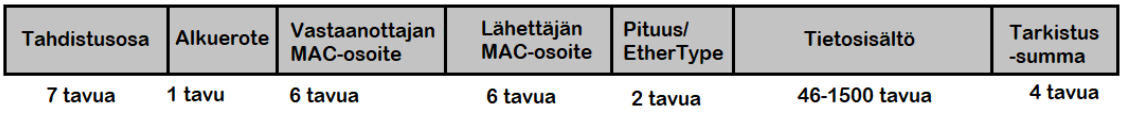
Tähtitopologiassa kaikki verkkoliikenne kulkee yhden laitteen lävitse. Laite voi olla kotiverkoissa reititin, jolla laitteet voidaan yhdistää Internetiin. Tähtitopologia on yleisin käytetty topologia nykyaikana. Vikasietoisuuden kannalta verkko ei tosin toimi, jos tähden keskusta ei toimi. [37, s. 242–244]

4.2.2 Ethernetin siirtokerros

Kytkimiä voidaan käyttää Ethernet-verkon osien yhdistämiseen. Esimerkiksi kaksi pienempää verkkoa voidaan yhdistää yhdeksi isommaksi verkoksi yhden kytkimen avulla, jolloin kytkin toimii siltana kahden verkon välillä. Kun kytkin vastaanottaa paketin, se tarkastaa paketista lähettäjän MAC-osoitteen, joka tallennetaan kytkimen osoitetauluun. Kun kytkin tietää kaikkien yhdistettyjen laitteiden MAC-osoitteet, se osaa välittää paketit oikeille Ethernet-laitteille. [37, s. 299–305]

Siirtokerrokseen kuuluvat olennaisena osana Ethernet-kehykset, jotka kuvaavat yhden Ethernet-paketin sisällön. Yhdessä kehyksessä on tarvittava metadata, jonka avulla paketti

voidaan luotettavasti lähettää määränpäähänsä. Kuvassa 7 esitellään IEEE 802.3 standardin mukainen Ethernet-kehys. [4, s. 22–23]



Kuva 7. *Ethernet-kehyksen rakenne IEEE 802.3 standardin mukaan [4, s. 22–23].*

Ethernet-kehyksen ensimmäiset 8 tavua on varattu tahdistusosalle (*preamble*) ja kehyksen alkuerotteelle (*start-of-frame delimiter*). Ne kuuluvat OSI-mallissa fyysiseen kerrokseen. Muut Ethernet-kehyksen kentät kuuluvat OSI-mallissa siirtokerrokseen. Tahdistusosa on tarkoitettu lähettäjän ja vastaanottajan pakettilähetysten synkronoimiseen. Käytännössä tahdistusosa kertoo vastaanottajalle, että kehystä ollaan lähettämässä. Alkuerote kertoo vastaanottajalle tarkan kohdan, mistä kehys alkaa. [4, s. 22–24]

Kehyksessä seuraavat 12 tavua on varattu vastaanottajan ja lähettäjän MAC-osoitteisiin (*destination MAC-address, source MAC-address*). Kuten aliluvun 4.2.1 alussa mainittiin, MAC-osoite yksilöi Ethernet-laitteen verkossa. Paketissa on oltava vastaanottajan MAC-osoite, jotta paketti osataan ohjata oikealle Ethernet-laitteelle. Jos vastaanottaja huomaa, että kohdeosoite ei ole sama kuin oman verkkosovittimen osoite, paketti hylätään. Paketissa on oltava myös lähettäjän MAC-osoite, jonka avulla esimerkiksi verkkokytkin voi rekisteröidä laitteen osoitteen. [4, s. 24–25]

Seuraavat 2 tavua on varattu joko datakentän pituuden ilmaisemiseen tai *EtherType*-merkintään. Vastaanottajan täytyy tietää, kuinka monta tavua datasta täytyy lukea, sillä datassa voi olla 46-1500 tavua. *EtherType* kertoo, mikä protokolla on koteloitu Ethernet-kehyksen sisään. [4, s. 25]

Viimeiset neljä tavua on varattu tarkistussummaan (*cycle redundancy check*). Tarkistussumma on algoritmilla laskettava luku, joka kuvaa koko Ethernet-kehystä. Vastaanottaja laskee vastaanotetun Ethernet-kehyksen tarkistussumman ja vertaa sitä kehyksessä olevaan tarkistussummaan. Jos tarkistussummat ovat samat, paketti lähetettiin onnistuneesti. Muuten paketti hylätään. [4, s. 26]

4.3 TCP/IP

TCP/IP on usean protokollan yhdistelmä [16, s. 25–26]. TCP- ja IP-protokollat ovat kuitenkin oleellisimmat, sillä niiden avulla kommunikaatio Internetissä ja Ethernet-verkoissa on mahdollista. Kuten aiemmissa aliluvuissa mainittiin, IP-protokolla sijoittuu OSI-mallissa verkkokerrokseen ja TCP-protokolla sijoittuu OSI-mallissa kuljetuskerrokseen. TCP- ja IP-protokollien toiminta on huomattavan monimutkaista, joten tämän diplomityön laajuudessa vain perusteet käydään läpi.

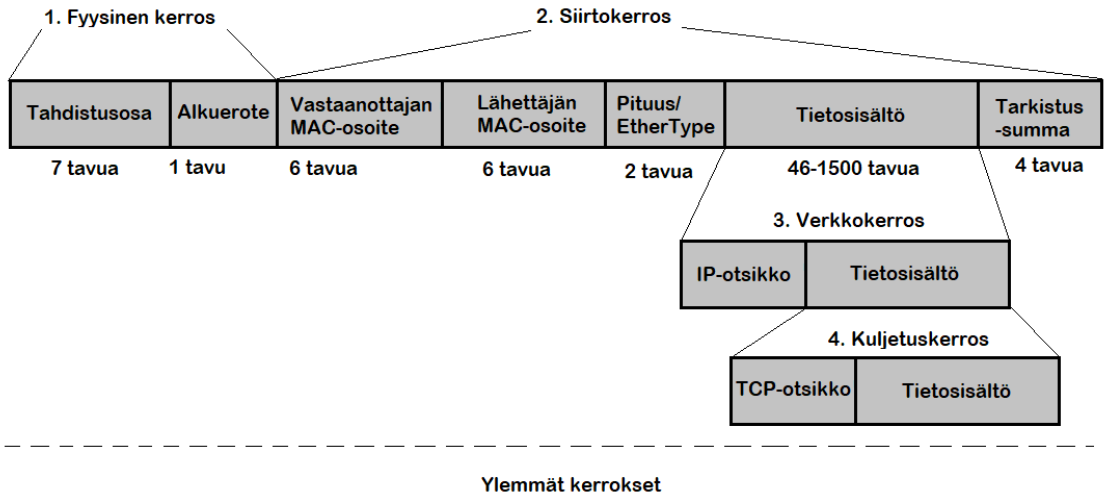
IP-protokollan tehtävä on pakettien reititys [19, s. 56–57]. Kaikki alemmat OSI-mallin kerrokset tukevat IP-protokollaa. Yksinkertaistettuna IP-osoite on jokaiselle verkossa olevalle laitteelle asetettu osoite. IP-osoitteen perusteella lähiverkoissa ja Internetissä paketit voidaan reitittää oikeaan määränpäähän. Tällä hetkellä IP-protokollasta on käytössä kaksi versiota [16, s. 6]. IPv4 on vielä nykyään laajasti käytetty IP-protokollan versio, joka kehitettiin jo vuonna 1983. IPv6 on uudempi versio IP-protokollasta, jonka tärkeimpänä ominaisuutena on laajempi osoiteavaruus, sillä IPv4 ei tarjoa riittävästi osoitteita Internetin suosion seurauksena.

TCP- ja UDP-protokollat (*User Datagram Protocol*) ovat OSI-mallissa kerroksen ylempänä. IP-protokolla tarjoaa pakettien reitityksen, mutta ei pakettien luotettavaa reititystä, johon TCP-protokolla perustuu. UDP-protokolla perustuu yhteydettömyyteen. UDP-protokollassa pakettien lähetys varmistetaan vain seuraavaan verkon solmuun, mutta UDP-protokolla ei varmista, että paketti meni perille saakka. Tästä syystä UDP-protokolla sopii verkkotapahtumiin, joissa muutaman paketin hukkuminen ei haittaa viestintää merkittävästi. TCP-protokollassa pitää ennen datan lähetystä avata koneiden välille yhteys. Protokolla huolehtii, että jokainen paketti saapuu perille oikeassa järjestyksessä. [19, s. 59–61]

TCP- ja UDP-protokollissa jokaiselle IP-osoitteelle tarjotaan monta porttia, joiden kautta kommunikaatio voi tapahtua, sillä yhden IP-osoitteen kautta ei voisi muuten ylläpitää useaa kommunikaatiota. Asiakaspalvelin-mallissa palvelin kuuntelee aktiivisesti tiettyä porttia. Kun asiakas ottaa yhteyttä palvelimeen, pyyntö lähetetään tiettyyn IP-osoitteeseen ja porttiin. Tämän jälkeen asiakas ja palvelin voivat keskustella portin kautta, kunnes yhteys suljetaan. Porttien numerot on tyypillisesti standardisoitu käytön mukaan. Esimerkiksi verkkopalvelimet kuuntelevat liikennettä TCP-portissa 80, SSH (*Secure Shell*) kuuntelee liikennettä TCP-portissa 22 ja NTP-palvelin (*Network Time Protocol*) kuuntelee liikennettä UDP-portissa 123. [23, s. 31–35]

IP-protokollassa paketeilla on IP-otsikko ja tietosisältö. IP-otsikossa on tärkeimpinä kenttinä tieto käytetyn IP-protokollan versiosta, otsikon pituus, paketin pituus, ylemmän protokollan tunniste, lähettäjän IP-osoite, vastaanottajan IP-osoite ja paketin tarkistussumma. [23, s. 35–38]

TCP- ja UDP-protokollissa paketeilla on TCP- tai UDP-otsikko ja tietosisältö, joka on jälleen varattu ylemmille kerroksille. Molemmilla protokollilla on otsikoissaan tärkeimpinä kenttinä lähettäjän portti, vastaanottajan portti, paketin koko ja paketin tarkistussumma. TCP-protokollan otsikko sisältää enemmän kenttiä, kuten esimerkiksi juoksevan sarjanumeron, jonka avulla tiedetään, kuinka paljon suuresta paketista on lähetetty vastaanottajalle. [23, s. 25–31]



Kuva 8. *OSI-mallin kerrokset 1-4 havainnoituna [1].*

Kuvassa 8 on esitelty tähän asti käsitellyt OSI-mallin kerrokset. Jokainen edellinen taso rakentuu edellisen kerroksen päälle. Siirtokerroksesta lähtien jokaisen kerroksen tietosisältö on vain seuraavaa kerrosta varten. [1]

4.4 SSH ja SFTP

SSH- ja SFTP-protokollat ovat OSI-mallissa korkeimmalla applikaatiokerroksella. SSH-protokolla on tietoliikenteen salaukseen tarkoitettu protokolla [21]. SFTP-protokolla ei kuitenkaan ole FTP-tiedostonsiirtoprotokolla (*File Transfer Protocol*) SSH-salauksen yli, vaan SFTP on täysin oma protokollansa [5, s. 82]. Ennen SFTP-protokollan selostusta on käytävä läpi SSH-protokolla, sillä SFTP-protokolla on rakennettu SSH-protokollan sisään. SSH-protokolla tarjoaa salatun kanavan asiakkaan ja palvelimen välille suojaamattomissa verkoissa symmetrisellä salauksella, epäsymmetrisellä salauksella ja tarkisteeseen perustuvalla salauksella [21].

Tarkisteeseen perustuvassa salauksessa viesteistä voidaan muodostaa tarkistesummia tarkistefunktiolla. Kuten monilla alemmilla OSI-mallin kerroksilla, tarkistesummaa käytetään tarkistamaan, ettei viesti ole muuttunut matkan varrella. [5, s. 42–43]

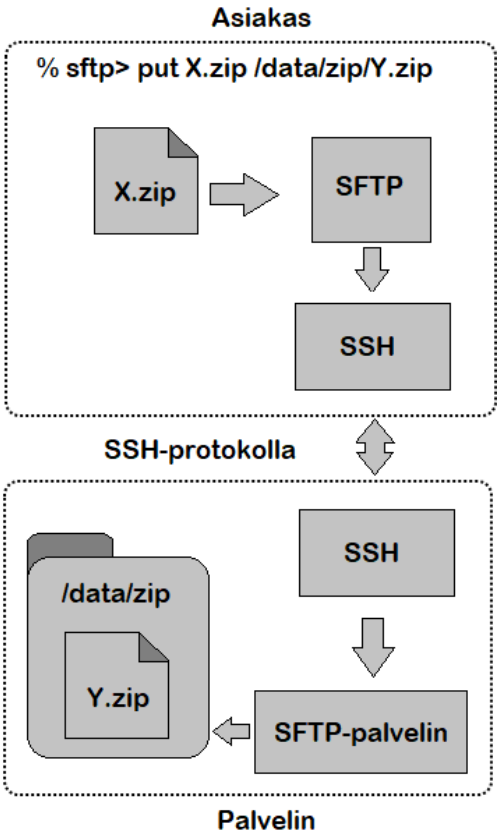
Symmetrisessä salauksessa käytetään yhteistä salausavainta, jolla viestit salataan ja puretaan. Kun asiakas tai palvelin lähettää viestin, yhteistä salausavainta käytetään viestin salaamiseen, jonka jälkeen viesti lähetetään salattuna vastaanottajalle. Viestin vastaanottaja käyttää samaa salausavainta purkaakseen salauksen. Symmetrinen salaus on yksinkertainen, joten se on myös nopea ja helppo toteuttaa. Suurin ongelma on kuitenkin salausavaimen jako, sillä salausavainta ei voi jakaa salaamattomassa verkossa. [21]

Asymmetrisessä salauksessa käytetään julkisen ja yksityisen avaimen salausta. Asiakkaalla ja palvelimella on olemassa julkinen ja yksityinen avain. Mikäli asiakas haluaa lähettää viestin, palvelin ensin toimittaa julkisen avaimen asiakkaalle. Asiakas voi salata

viestin julkisella avaimella, mutta vain palvelimen yksityinen avain voi purkaa salauksen. Asymmetrinen salaus siis ratkaisee symmetrisen salauksen suurimman ongelman, eli salausavaimen jakamisen. Salaamattomassa verkossa palvelin voi lähettää asiakkaalle julkisen avaimen, sillä julkinen avain toimii vain viestin salaamiseen, ja vain yksityinen avain voi purkaa salauksen. Julkisen ja yksityisen avaimen salaus on matemaattisesti hyvin raskasta, joten verkon salaus on hidasta. [21]

SSH-protokolla käyttää symmetristä ja asymmetristä salausta yhdessä, jolloin saadaan molempien salaustekniikoiden hyödyt. SSH-salauksen alussa käytetään asymmetristä salausta. Palvelin lähettää julkisen avaimensa asiakkaalle, jonka jälkeen asiakas voi lähettää salattuja viestejä palvelimelle. Sen jälkeen siirrytään symmetriseen salaukseen, eli asiakas ja palvelin voivat vaihtaa yhteisen salausavaimen asymmetrisen salauksen yli. [5, s. 41–42]

SSH-protokolla tarjoaa kaksi tiedostonsiirtoprotokollaa, jotka toimivat SSH-prosessin sisällä. Ensimmäinen tiedostonsiirtoprotokolla on SCP-protokolla (*Secure Copy Protocol*). Historiallisesti SCP-protokolla oli ensimmäinen tiedostonsiirtoprotokolla, jonka SSH-protokolla tarjosi. Tässä diplomityössä ei kuitenkaan käsitellä SCP-protokollaa, sillä sen tarjoamat toiminnallisuudet eivät ole riittäviä kaatumisvedosten käsittelyyn. SCP-protokolla ei tarjoa esimerkiksi tiedostojen listausta tai tiedostojen poistoa. Sen sijaan käsitellään uudempaa SFTP-tiedostonsiirtoprotokollaa. [5, s. 81–82]



Kuva 9. *SFTP-protokollan toiminta havainnollistettuna [5, s. 83].*

Kuvassa 9 on havainnollistettu SFTP-protokollan toimintaa. Asiakkaan päädyssä ajetaan SFTP-asiakasta, joka toimii SSH-prosessin sisällä. Kun käyttäjä pyytää SFTP-asiakasta siirtämään tiedoston X.zip palvelimelle sijaintiin /data/zip/Y.zip, pyyntö lähetetään SSH-protokollan yli palvelimelle, jossa ajetaan SFTP-palvelinta. Tällä tavoin asiakkaan ei tarvitse tietää mitään palvelimen toteutuksesta, sillä toteutusyksityiskohdat hoidetaan SSH-protokollan sisällä. Palvelimella voidaan käyttää esimerkiksi *OpenSSH*-ohjelmistoa tai *Tectia SSH* -ohjelmistoa. Mikäli tiedostonsiirto on mahdollista määritettyyn tiedosto-sijaintiin, tiedonsiirto voidaan aloittaa. [5, s. 323–324][21]

SFTP-protokolla tarjoaa tiedostonsiirron lisäksi monia muita komentoja. SFTP-asiakas voi hakea tai lähettää yhden tai useamman tiedoston palvelimelle, poistaa tiedostoja ja nimetä uudelleen tiedostoja. SFTP-asiakas voi myös listata kaikki tiedostot tietyssä hakemistossa. Lisäksi hakemistoja voidaan luoda tai poistaa. [5, s. 323–324]

5. KOHDEJÄRJESTELMÄN KOMMUNIKAATIO

Edellisessä luvussa käytiin läpi verkkotekniikat, joita hyödynnetään kohdejärjestelmän protokollissa. Lisäksi käytiin läpi verkkotekniikoita, joita vaaditaan kaatumisvedosten siirrossa.

Tässä luvussa esitellään kohdejärjestelmän kommunikaatioprotokollat tarkemmin. Tämän luvun perusteella voidaan lopulta esitellä kaatumisvedosten raportointi.

5.1 Kohdejärjestelmän kommunikaatioprotokollat

Aliluvussa 4.1 käytiin läpi OSI-mallin eri kerrokset. Sen jälkeen kaikki käsitellyt tietoliikenneprotokollat ja tekniikat selitettiin OSI-mallia hyödyntäen. Samalla tyylillä jatketaan, kun käydään läpi kohdejärjestelmän tietoliikenneprotokollat.

Kuten aiemmin on mainittu, kommunikaatiomoduulin ja työpöytäsovelluksen välillä käytetään TCP/IP-protokollayhdistelmää. Kommunikaatiomoduulilla on IP-osoite, aliverkon peite ja yhdyskäytävän osoite, joiden avulla työpöytäsovellus voi ottaa yhteyttä kommunikaatiomoduuliin. TCP-protokolla varmistaa, että lähetetyt paketit saapuvat perille oikeassa järjestyksessä. Se ei kuitenkaan ota kantaa, miten asiakas ja palvelin varsinaisesti kommunikoivat. On siis oltava kommunikaatioprotokolla, jonka avulla kommunikaatiomoduuli ja työpöytäsovellus osaavat kommunikoida. OSI-mallissa protokolla sijoittuu sovelluskerrokseen.

Moduulien välillä hyödynnetään Ethernet-tekniikkaa. Ethernet-tekniikka ei tarjoa kommunikaatiotapoja, mutta sen päälle voidaan rakentaa protokollia. Teollisuusjärjestelmän vaatimusten vuoksi kohdejärjestelmän on oltava redundantti. Tästä syystä moduulien välillä käytetään redundanssisuusprotokollaa, joka mahdollistaa kohdejärjestelmän toiminnan yhden moduulin kaatuessa. Protokolla koteloidaan Ethernet-kehyksen sisään viestin lähetyksessä, jolloin kehyksen pituuskenttä korvataan *EtherType*-kentällä. Viestin vastaanottaja poistaa koteloidun protokollan, jolloin vastaanottaja voi käsitellä viestiä tavallisena Ethernet-kehyksenä. Protokolla ei ole viestien käsittelyn kannalta relevantti, sillä se on käytännössä näkymätön korkeammalla tasolla. Se ei myöskään liity kaatumisvedosten siirtämiseen, joten sitä ei myöskään käsitellä tarkemmin.

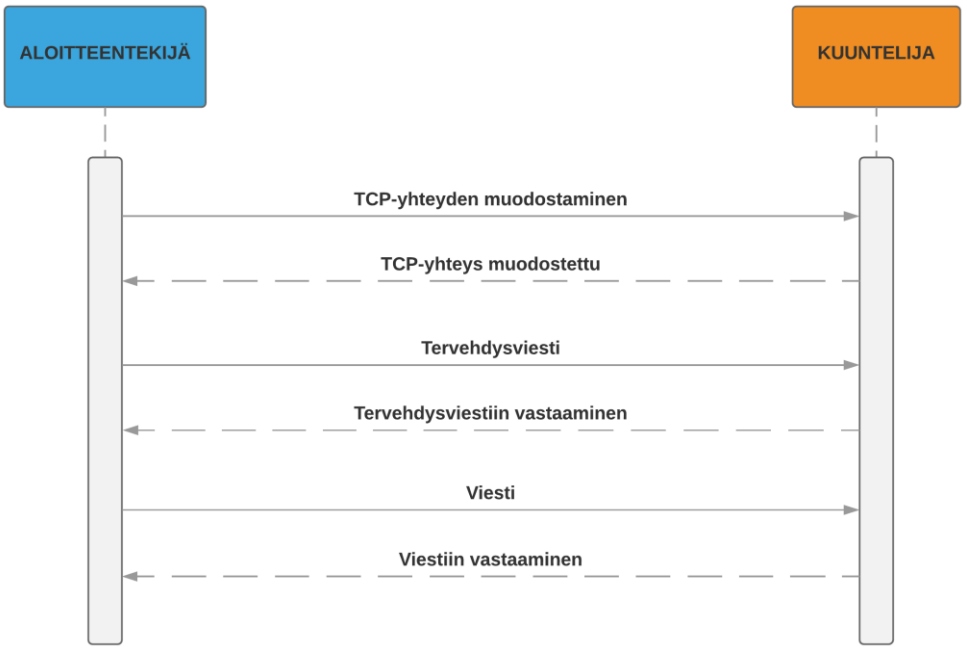
Moduulien välille on kehitetty räätälöity kommunikaatioprotokolla, joka toimii Ethernet-tekniikan päällä. Jotta kaatumisvedoksia voidaan siirtää moduulien välillä, tarvitaan tiedostonsiirtoprotokolla. Aiemmin käsitelty SFTP-protokolla kuuluu OSI-mallissa sovelluskerrokseen. Räätälöity protokolla kuuluu OSI-mallissa huomattavasti alempaan kerrokseen, joten tiedostonsiirtoon on kehitettävä toinen ratkaisu.

Lopuksi tarvitaan keino, jonka avulla työpöytäsovellus voi keskustella jonkun muun, kuin kommunikaatiomoduulin kanssa. Työpöytäsovellus voi lähettää esimerkiksi komennon, jolla tietty moduuli systeemissä käynnistetään uudelleen. Komennon on ensin kuljettava työpöytäsovelluksen ja kommunikaatiomoduulin välisen protokollan yli ja sen jälkeen moduulien välisen protokollan yli. Seuraavissa aliluvuissa kerrotaan tarkemmin, miten edellä mainitut protokollat ja tekniikat toimivat.

5.2 Kommunikaatiomoduulin ja työpöytäsovelluksen kommunikaatio

Kommunikaatioprotokolla kommunikaatiomoduulin ja työpöytäsovelluksen välillä on rakennettu TCP-protokollan päälle. Tämä tarkoittaa sitä, että kommunikaatioprotokolla toimii vasta silloin, kun TCP-yhteys asiakkaan ja palvelimen välillä on avattu. Kommunikaatioprotokolla hyödyntää TCP-protokollan tarjoamia ominaisuuksia, kuten TCP-otsikossa olevaa sarjanumeroa suurten pakettien lähettämiseen osissa. Kommunikaatioprotokollasta käydään vain ne osat läpi, jotka liittyvät kaatumisvedosten siirtoprosessiin.

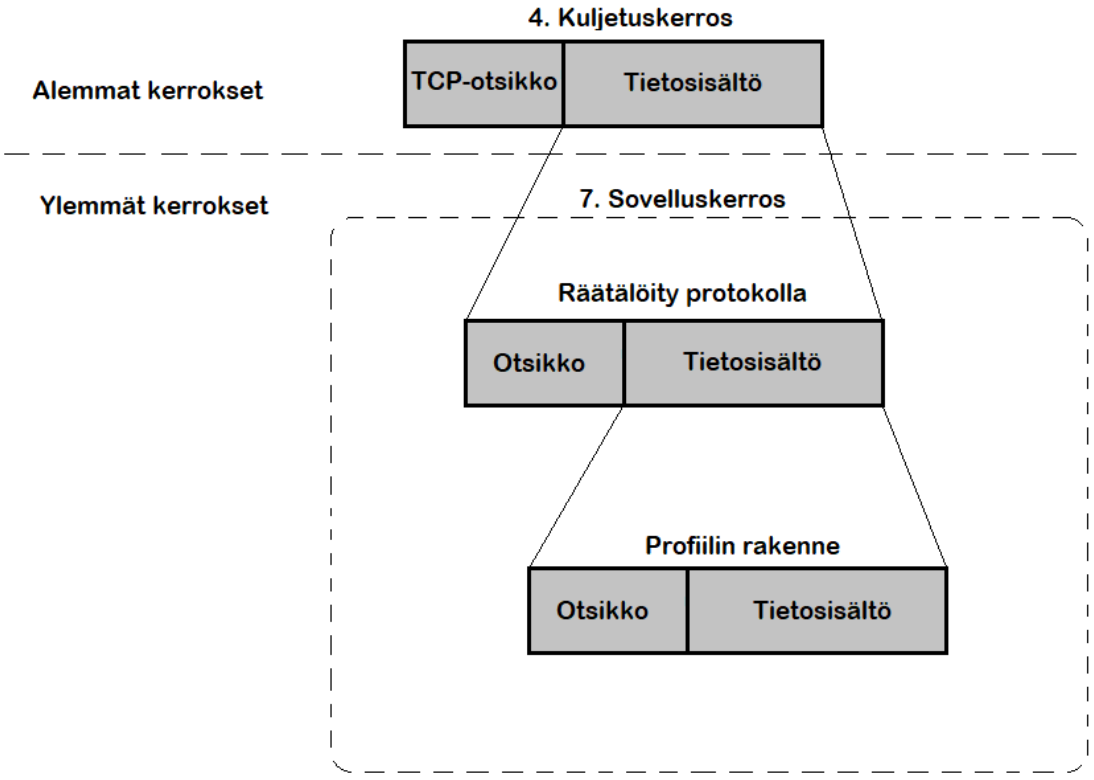
Kommunikaatioprotokolla on räätälöity kohdejärjestelmälle sopivaksi BEEP-protokollakehyksen (*Blocks Extensible Exchange Protocol*) mukaisesti. BEEP-protokollakehys ei ole protokolla, vaan joukko hyväksi havaittuja käytäntöjä, joista on kehitetty uudelleen käytettävä protokollakehys. BEEP-protokollakehys toimii vertaisverkkoperiaatteella, joten siinä ei ole asiakasta tai palvelinta. Jonkin osapuolen on kuitenkin aloitettava kommunikaatio. Tätä osapuolta kutsutaan aloitteentekijäksi (*initiator*). Toinen osapuoli on kuuntelija (*listener*). [12]



Kuva 10. *BEEP-protokollakehyksen toiminta [33].*

BEEP-protokollakehyksessä on kanavia ja profiileja, joiden avulla kommunikointi tapahtuu. Profiili on joukko sääntöjä, miten viesti muodostetaan ja luetaan. Kommunikaation luonnissa jokaiselle profiilille luodaan oma kanava, jonka läpi kommunikoidaan. Yksi kommunikaatioistunto voidaan siis limittää (*multiplex*) moneksi eri kanavaksi. [12]

Kun aloitteentekijä ottaa yhteyttä kuuntelijaan, aloitteentekijä lähettää tervehdysviestin, jossa on lista profiileja, joita aloitteentekijä tukee. Kuuntelija lähettää takaisin tervehdysviestin, jossa on lista kuuntelijan tukemista profiileista. Kun aloitteentekijä tietää profiilit, joita aloitteentekijä ja kuuntelija tukevat, profiilien välille luodaan kanavat. Tämän jälkeen aloitteentekijä voi lähettää kuuntelijalle viestejä, joihin kuuntelija vastaa. Kuvassa 10 on esitetty protokollan toiminta. [33]



Kuva 11. *TCP/IP-protokollayhdistelmän päälle rakennetun protokollan rakenne.*

BEEP-protokollakehys perustuu XML-kieleen (*Extensible Markup Language*) [33]. Kohdejärjestelmässä kommunikaatio on työpöytäsovelluksen ja kommunikaatiomoduulin välillä. Moduulit ovat vähätehoisia sulautettuja laitteita, joten XML-kielen jäsentäminen olisi turha raskas operaatio kommunikaatiossa. Sen sijaan käytetään räätälöityä protokollaa, jossa jokaisessa viestissä on otsikko ja tietosisältö. Otsikossa on tieto viestin pituudesta, käytetystä kanavasta ja profiilista, joka on viestin tietosisällössä. Tämä on esitetty kuvassa 11.

Myös profiililla on otsikko ja tietosisältö. Otsikossa tärkeimpinä kenttinä ovat moduulin tunniste, jolle viesti lähetetään, viestin tyyppi ja tietosisällön pituus. Viestin tyyppi voi

olla pyyntö tai vastaus, riippuen siitä lähettääkö aloitteentekijä viestin kuuntelijalle vai vastaako kuuntelija aloitteentekijälle.

Kohdejärjestelmässä on monia profiileja, jotka liittyvät ajettavan ohjelman eri osa-alueisiin. Niitä ei käsitellä tässä diplomityössä. Yksi profiileista on tarkoitettu systeemikomentoihin, joita tarkastellaan tässä diplomityössä tarkemmin.

Kohdejärjestelmässä systeemikomennot ovat viestejä, joilla voidaan ohjata moduulien perustoiminnallisuuksia. Näihin kuuluvat muun muassa moduulin uudelleenkäynnistys, SSH-palvelun käynnistäminen tai moduulin pakottaminen alkulatausohjelmaan. Systeemikomennot täytyy toimia työpöytäsovelluksen ja kommunikaatiomoduulin välillä. Kommunikaatiomoduuli pystyy jäsentelemään systeemikomennon osaksi protokollaa, jota käytetään moduulien välillä. Tätä käsitellään seuraavassa aliluvussa.

Systeemikomentojen suunnitteluperiaatteiden mukaan niiden on liityttävä moduulin perustoiminnallisuuteen. Vain välttämättömät toiminnallisuudet voidaan mieltää systeemikomennoiksi. Systeemikomennoissa on taattava täysi taaksepäin yhteensopivuus, joten mitään luotuja systeemikomentoja ei poisteta missään tulevissa ohjelmistoversioissa. Kaatumisvedosten noutaminen voidaan luokitella perustoiminnallisuudeksi sen tärkeyden vuoksi. Tällä tavoin kaatumisvedosten raportointi saadaan integroitua osaksi moduulien toimintaa. Kaatumisvedosten noutamista systeemikomentojen avulla tarkastellaan aliluvussa 6.4.

Systeemikomennoissa tietosisällössä voi olla esimerkiksi parametreja, jotka lähetetään komennon mukana, ja mahdollisia paluuarvoja. Esimerkiksi kohdejärjestelmässä työpöytäsovellus voi lähettää komennon kommunikaatiomoduulille SSH-palvelun aktivoimiseksi. Parametrissa voidaan antaa *true*- tai *false*-arvo, riippuen halutaanko SSH palvelu aktivoida vai kytkeä pois päältä. Kommunikaatiomoduuli voi lähettää takaisin viestin, jossa kerrotaan, onnistuiko komennon suorittaminen.

5.3 Moduulien välinen kommunikaatio

Moduulit ovat yhdistetty toisiinsa Ethernet-kaapeloinnilla rengastopologian mukaisesti. Korkeamman tason protokollien sijaan käytetään räätälöityä protokollaa, joka on rakennettu Ethernet-tekniikan päälle. Protokollasta tarkastellaan vain osat, jotka liittyvät kaatumisvedosten siirtämiseen.

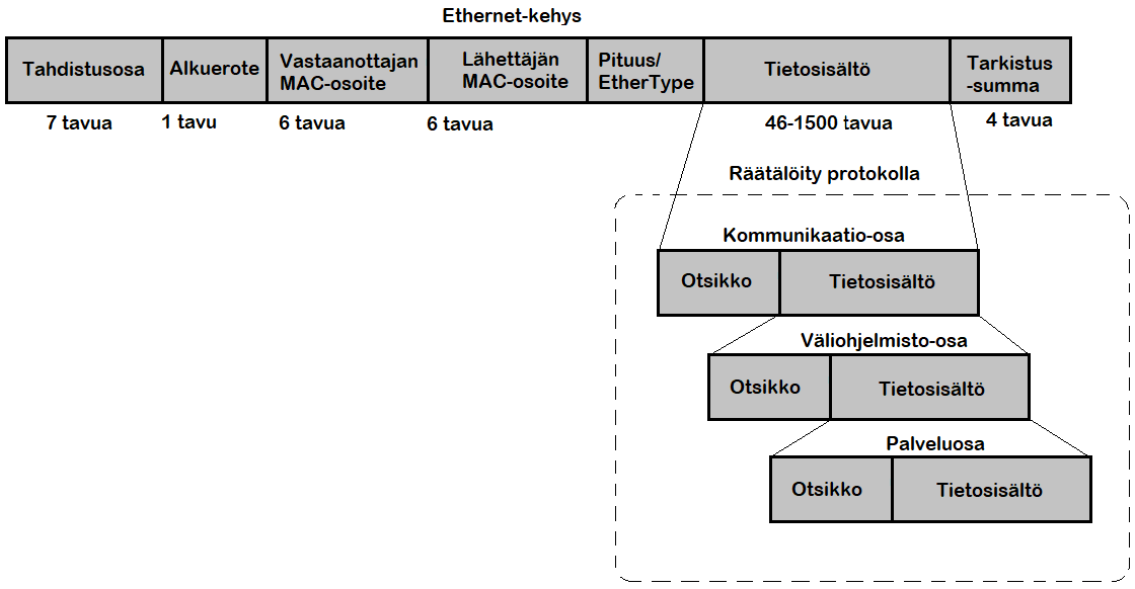
Työpöytäsovelluksen ja kommunikaatiomoduulin välinen kommunikaatioprotokolla rakentuu TCP/IP-protokollayhdistelmän päälle. Protokollassa ei tarvitse miettiä pakettien reititystä, sillä IP-protokolla huolehtii siitä. Protokollassa ei myöskään tarvitse miettiä pakettien luotettavaa lähetystä, sillä TCP-protokolla huolehtii siitä. Räätälöity protokolla on OSI-mallissa IP- ja TCP-protokollien alapuolella, joten räätälöidyssä protokollassa ei voida hyödyntää niiden tuomia etuja.

Räätälöidyn protokollan suunnittelussa on otettu TCP/IP-protokollayhdistelmästä joitain vaikutteita. Protokolla on jaettu kolmeen osaan, jotka ovat kommunikaatio-osa, väliohjelmisto-osa ja palveluosa. Jokainen osa on rakennettu otsikkotietosisältö-mallilla toistensa päälle, eli kommunikaatio-osan tietosisältö on varattu väliohjelmisto-osalle ja niin edelleen.

Kommunikaatio-osassa on muun muassa tietokenttä, joka kertoo protokollan version, lähettäjän osoitteen, vastaanottajan osoitteen, tietosisällön tyypin ja tietosisällön koon. Lähes vastaavat kentät löytyvät IP-protokollasta. Osoitteena ei käytetä IP-osoitteita, vaan osoite koostuu muun muassa moduulin numeraalisesta tunnisteesta ja moduuliryhmän tunnisteesta. Osoite voi esimerkiksi sanoa, että viesti lähetetään kaikille järjestelmän kommunikaatiomoduuleille tai yksittäiselle kommunikaatiomoduulille.

Väliohjelmisto-osassa on muun muassa juokseva sarjanumero, jolla pidetään kirjaa, kuinka paljon isosta paketista on siirretty, yksilöivä tunniste siirtotapahtumasta ja tietosisällön pituus. Lähes vastaavat kentät löytyvät TCP-protokollasta.

Palveluosan otsikko kertoo, mitä dataa tietosisällössä on. Tietosisällössä voi olla merkittävimpinä osina kaatumisvedosten siirron kannalta tiedostonsiirron data, systeemikomennot tai kuulutusviestit. Moduulit kuuluttavat omaa tilaansa kaikille moduuleille periodisesti, sillä erityisesti kommunikaatiomoduulia kiinnostaa, mitkä moduulit järjestelmässä ovat käynnissä, jotta kommunikaatiomoduuli voi ilmoittaa tiedon työpöytäsovellukselle. Tätä käsitellään tarkemmin seuraavassa aliluvussa. Kohdejärjestelmässä on monia muita mahdollisia tietosisältötyyppejä, mutta ne eivät liity kaatumisvedosten siirtämiseen. Kuvassa 12 on esitelty kaikki kolme osaa.



Kuva 12. *Räätälöity protokolla Ethernet-kehyksen tietosisältönä.*

Systeemikomennot ovat samoja, joita työpöytäsovellus voi lähettää kommunikaatiomoduulille. Kun ylemmällä tasolla työpöytäsovellus lähettää kommunikaatiomoduulille viestin, kommunikaatiomoduuli käsittelee viestin. Mikäli viesti on tarkoitettu kommunikaatiomoduulille, viesti käsitellään heti. Mikäli viesti on tarkoitettu toiselle moduulille, systeemikomento asetetaan palveluosaan ja lähetetään eteenpäin.

Kohdejärjestelmässä on toteutettu tiedostonsiirtoprotokolla, jonka avulla tiedostoja voidaan lähettää tai hakea toiselta moduulilta. Yksinkertaistettuna moduuli voi pyytää toiselta moduulilta tiedostoa vastaavalla tavalla kuin SFTP-protokollassa, jota käsiteltiin aiemmin aliluvussa 4.4.

Kun tiedosto pyydetään toiselta moduulilta, räätälöidyn protokollan kommunikaatio-osaan asetetaan toisen moduulin osoite, väliohjelmisto-osaan varataan tunniste tiedostonsiirto-operaatiosta ja palveluosaan asetetaan halutun tiedoston tiedostopolku. Toinen moduuli alkaa lähettämään haluttua tiedostoa pala kerrallaan samalla väliohjelmisto-osan tunnisteella. Juoksevan sarjanumeron avulla pidetään kirjaa lähetetyn datan määrästä.

5.4 Kaatumisvedosten tilan kuuntelija

Työpöytäsovellus voi lähettää viestejä mille tahansa kohdejärjestelmän moduulille kommunikaatiomoduulin kautta, mutta kommunikaation pitäisi toimia myös toiseen suuntaan. Toisin sanoen minkä tahansa moduulin täytyy voida lähettää viestejä työpöytäsovellukselle. Kohdejärjestelmässä on yleisesti hyödyllistä, että moduulit voivat itse kertoa eri komponenttien tilan periodisesti.

Kaatumisvedosten raportoinnissa työpöytäsovelluksen täytyy tietää, millä moduuleilla on kaatumisvedoksia. Lisäksi kaatumisvedosten siirtämisessä työpöytäsovelluksen on tiedettävä tiedostonsiirron tila. Työpöytäsovellus voisi kysyä tiloja erikseen, mutta kommunikaation kannalta on edullisempaa lähettää viestejä vain tarpeen mukaan.

Aliluvussa 5.2 käsiteltyä protokollaa voidaan käyttää kaksisuuntaisesti. Kun työpöytäsovellus ja kommunikaatiomoduuli ovat avanneet yhteyden protokollalla, molemmat voivat lähettää toisilleen viestejä. Tilan lähettämiselle voidaan luoda uusi profiili ja kanava. Profiilin mukaisesti lähetettävässä paketissa on mukana otsikko, joka kertoo mitä tietoa ollaan lähettämässä, ja tietosisältö, jossa on varsinainen lähetettävä data.

Muut moduulit eivät voi lähettää viestejä työpöytäsovellukselle suoraan. Sen sijaan kaikki moduulit kuuluttavat tilaansa sekunnin välein. Jokainen moduuli lähettää kaikille järjestelmän moduulille tilansa, jossa on muun muassa tieto moduulin sarjanumerosta. Tällöin jokainen järjestelmän moduuli on tietoinen kaikkien moduulien tilasta. Kaatumisvedosten tila voidaan integroida osaksi kuulutustilaa. Jokainen moduuli on tällöin tietoinen, millä moduuleilla on kaatumisvedoksia.

Systeemissä voi olla useita kommunikaatiomoduuleita. Vain se kommunikaatiomoduuli, johon ollaan yhteydessä työpöytäsovelluksella, lähettää oletuksena kaikki vastaanotetut tilat työpöytäsovellukselle 15 sekunnin välein. Kommunikaatiomoduuli voi myös lähettää kaikki tilat välittömästi, mikäli jokin komponentti vaatii välitöntä tilan ilmoittamista. Myöhemmin käsiteltävässä kaatumisvedosten tiedostonsiirrossa tila voi vaihdella kaatumisvedosten määrästä riippuen sekunnin välein, jolloin 15 sekunnin aikaväli ei ole riittävä.

Kun työpöytäsovelluksella otetaan yhteyttä mihin tahansa systeemin kommunikaatiomoduuliin, kommunikaatiomoduuli aloittaa tilojen lähettämisen työpöytäsovellukselle. Tämän jälkeen työpöytäsovellus kuuntelee kommunikaatiomoduulin tilaa jatkuvasti. Mikäli kommunikaatiomoduuli ilmoittaa, että esimerkiksi yhdellä ohjausmoduulilla on kaatumisvedoksia, työpöytäsovellus voi ilmoittaa käyttäjälle, että systeemissä on kaatumisvedoksia.

6. KAATUMISVEDOSTEN RAPORTOIMINEN

Tässä luvussa esitellään kaatumisvedosten raportointi. Esittelyssä hyödynnetään kaikkia tähän asti käsiteltyjä asioita.

Tässä luvussa lähdetään liikkeelle ohjelman tai ytimen kaatumisesta. Sen jälkeen käsitellään kaatumisvedosten pakkausta ja arkistointia. Lopuksi tarkastellaan, miten kaatumisvedokset siirretään työpöytäsovellukselle ja miten kaatumisvedokset raportoidaan.

6.1 Kaatumisvedosten käsittely

Ennen ominaisuuden toteuttamista kohdejärjestelmässä oli jo valmiina joitain komponentteja muiden osapuolten toteuttamana. Kaikille moduuleille on jo kerätty kaatumisvedoksia kohdejärjestelmän aiemmissa ohjelmistoversioissa. Tässä aliluvussa kuvaillaan nämä valmiit komponentit. Vasta seuraavissa aliluvuissa käsitellään komponentteja, jotka toteutettiin tätä diplomityötä varten.

Luvussa 3 käsiteltiin kaatumisvedoksia. Käyttöjärjestelmässä ajettavissa ohjelmissa voidaan määrittää, luodaanko kaatuneesta ohjelmasta kaatumisvedosta. Vastaavasti ytimen kaatumisvedoksen tallennukseen voidaan käyttää erilaisia taustaprosesseja. Kohdejärjestelmässä ajettavan ohjelman on sallittu luoda rajattoman kokoisia kaatumisvedoksia aina kaatumisen yhteydessä *rlimit*-tietueella. Käyttöjärjestelmän ytimen kaatumisvedosten tallentamiseen käytetään Linux-käyttöjärjestelmän *MTDoops*-moduulia, joka kytketään päälle ytimen koodissa.

Mikäli kohdejärjestelmän moduulilla ajettava ohjelma tai käyttöjärjestelmän ytimessä ajettava prosessi kaatuu, moduuli käynnistyy automaattisesti uudelleen. Käynnistyksen yhteydessä moduuli käynnistyy ensin valmistajan kehittämään ensimmäiseen alkulatausohjelmaan, joka lataa toisen alkulatausohjelman. Kaatumisvedosten käsittely alkaa toisessa alkulatausohjelmassa. Moduuleilla on erillinen osio, johon on varattu pieni määrä tallennustilaa. Osio on tarkoitettu tiedostojen säilytykseen, joten myös kaatumisvedokset säilytetään kyseisessä osiossa.

Käyttöjärjestelmän ytimen kaatumisvedokset käsitellään tekstimuodossa ytimen puskurimuistissa. Linux-käyttöjärjestelmän *MTDoops*-moduuli tallentaa kaatumisvedoksen erilliselle MTD-osiolle, joka löytyy kohdejärjestelmän moduuleilta. Alkulatausohjelmassa uudet kaatumisvedokset haetaan MTD-osiolta. Kaatumisvedosten tallentamisen jälkeen MTD-osio tyhjätään, jotta samoja kaatumisvedoksia ei haeta uudelleen myöhemmin. Ytimen kaatumisvedoksia ei pakata, sillä ne ovat vain pieniä tekstitiedostoja.

Ohjelman kaatuessa käsittelemätön kaatumisvedos tallennetaan ensin osion juureen. Eri moduuleilla on eri määrä tallennustilaa. Pahimmillaan yksi pakkaamaton kaatumisvedos vie tallennustilasta jopa 10 prosenttia. Kaatumisvedoksia täytyy säilyttää moduuleilla huomattavan pitkiä aikavälejä, joten ajettavan ohjelman kaatumisvedokset on pakattava. Ajettavan ohjelman kaatumisvedoksen pakkaaminen ei ole välitöntä. Huonolla ajoituksella moduuli voidaan sammuttaa tai moduuli voi käynnistyä uudelleen pakkauksen aikana. Vaikka se on hyvin epätodennäköistä, käytännön kokeilut ominaisuuden toteutuksessa ovat todistaneet sen olevan mahdollista, joten siihen täytyy varautua.

Zip-tiedostojen luomiseen ja käsittelyyn on olemassa libzip-kirjasto [26]. Kirjasto on suunniteltu olemaan turvallinen ja atominen. Käytännössä tämä tarkoittaa sitä, ettei tiedostoja koskaan korruptoida tai poisteta vahingossa, vaikka libzip-kirjastoa käyttävä ohjelma äkillisesti sammutettaisiin kesken pakkaamisen. Kun zip-tiedosto avataan käsittelyä varten, libzip-kirjasto luo toisen väliaikaisen zip-tiedoston, johon tiedostot pakataan. Kun tiedostot on pakattu, väliaikainen tiedosto korvaa alkuperäisen zip-tiedoston atomisesti. Jos moduuli satuttaisiin käynnistämään uudelleen ennen kuin zip-tiedosto korvataan, väliaikainen zip-tiedosto jää tiedostojärjestelmään. Tämä ei ole suuri ongelma, sillä kyseiset väliaikaiset zip-tiedostot voidaan poistaa vaivattomasti alkulatausohjelmassa.

Ominaisuuden toteutuksessa on huomattu, että libzip-kirjastolla pakatessa ajettavan ohjelman kaatumisvedos saadaan pakattua hyvin tehokkaasti. Pakattu kaatumisvedos on usein vain 10 prosenttia alkuperäisestä koosta. Näin ollen moduuleilla voidaan säilyttää useita kaatumisvedoksia. Pakattujen kaatumisvedosten tiedostonimessä on tieto muun muassa kaatuneen ohjelmakomponentin nimestä ja aikaleima, jonka avulla voidaan päätellä kaatumisien järjestys.

Kaatumisvedokset sellaisenaan eivät vielä riitä kaatumisen syiden analysointiin, sillä kaatumisvedoksista ei saada selville, mikä versio ajettavasta ohjelmasta aiheutti ohjelman kaatumisen. Ohjelman kääntämisessä ohjelmasta luodaan kohdejärjestelmässä symbolitaulu, jonka avulla kaatumisvedoksesta voidaan tutkia *call tracea* vertaamalla kaatumisvedosta lähdekoodiin GDB-ohjelmalla [28, s. 18–38]. Tästä syystä alkulatausohjelmassa tallennetaan tarvittavia metatietoja ja erilaisia lokitiedostoja, jotka tekevät kaatumisen syyn selvittämisen mahdolliseksi ja helpommaksi. Nämä tiedot tallennetaan erillisiin tiedostoihin. Tiedostot tallennetaan samaan paikkaan, mihin ajettavan ohjelman kaatumisvedokset tallennetaan.

6.2 Kaatumisvedosten arkistointi

Moduulien välillä käytettävään räätälöityyn kommunikaatioprotokollaan ei ole toteutettu tukea tiedostolistaukselle toiselta moduulilta. Aliluvussa 5.3 esiteltiin toiminnallisuus, jossa tiedostoja voidaan siirtää moduulien välillä. Moduuli voi esimerkiksi pyytää tiedostoa toiselta moduulilta. Ongelmana on kuitenkin tietää siirrettävän tiedoston nimi ja polku ilman tiedostolistausta.

Tästä syystä jokainen moduuli arkistoi omat kaatumisvedoksensa tunnettuun sijaintiin tunnetulla tiedostonimellä niin, että toinen moduuli osaa pyytää tiedoston oikealla tiedostopolulla. Kohdejärjestelmän jokaisella moduulilla on uniikki numeraalinen tunnistusarvo, joka yksilöi moduulin kohdejärjestelmässä, joten luotu arkisto nimetään tunnistusarvon mukaan. Tällöin moduuli voi hakea tiedoston toiselta moduulilta tietämällä sen tunnistearvon.

Jos kaatumisvedoksia tulee lisää, ne lisätään olemassa olevaan arkistotiedostoon ilman pakkausta. Tämä on mahdollista käytetyllä libzip-kirjastolla. Arkistoinnin ajoittamiseen on useita vaihtoehtoja.

Ensimmäinen vaihtoehto on suorittaa arkistointi heti yksittäisten kaatumisvedosten pakkauksen jälkeen alkulatausohjelmassa. Pienillä tiedostoilla arkistoiminen onnistui hyvin. Jos moduulilla on olemassa jo arkistotiedosto, jossa on esimerkiksi kolme kaatumisvedosta, yhden kaatumisvedoksen lisääminen oli silmämääräisesti välitöntä. Suuremmilla arkistotiedostoilla tiedoston lisääminen on huomattavasti hitaampaa. Mitä suurempi arkistotiedosto on, sitä kauemmin yhden tiedoston lisääminen kestää. Tämä ei johdu vain heikosta prosessorista. Moduulien tallennustilaan on hyödynnetty hidasta flash-muistia, jota ei ole tarkoitettu jatkuvaan uudelleenkirjoittamiseen. Jos arkistotiedostossa on jo kymmeniä kaatumisvedoksia, yhden kaatumisvedoksen lisääminen kestää jopa minuutin. Tämä ei ole teollisuusjärjestelmän toiminnan kannalta hyväksyttävää, sillä ajettavan ohjelman käynnistymiseen on aikarajat. Alkulatausohjelmassa on lisäksi vahtikoira-ajastin, eli ajastin, joka varmistaa alkulatausohjelman suorituksen etenemisen [8]. Mikäli ajastin huomaa, että jokin operaatio kestää 15 sekuntia tai kauemmin, moduuli käynnistetään uudelleen. Jos moduuli on käynnistetty uudelleen liian monta kertaa, moduuli ei enää yritä käynnistyä, vaan se jää pysähtyneeseen tilaan.

Toinen vaihtoehto on suorittaa arkistointi taustalla alkulatausohjelmassa. Toisin sanoen pääsäie voisi suorittaa alkulatausohjelman alustamisen ja toinen säie voisi suorittaa arkistoimisen. Tästä ideasta luovuttiin nopeasti, sillä kun alkulatausohjelma lataa ajettavan ohjelman, alkulatausohjelma lopetetaan välittömästi. Tällöin arkistointi jäisi kesken.

Kolmas vaihtoehto on suorittaa arkistointi joko alkulatausohjelmassa tai ajettavassa ohjelmassa. Aliluvussa 2.1.1 mainittiin, että moduuli pysyy alkulatausohjelmassa, jos ajettava ohjelma kaatuu useamman kerran peräkkäin tai jos ajettava ohjelma on korruptoitunut. Arkistointi voitaisiin suorittaa alkulatausohjelmassa tässä vaiheessa. Jos alkulatausohjelma lataa ajettavan ohjelman, arkistointi olisi suoritettava ajettavassa ohjelmassa. Tämä on ongelmallista, sillä silloin häirittäisiin ajettavan ohjelman reaaliaikasuoritusta. Yksi vaihtoehto olisi integroida arkistointi Xenomai-ohjelmistokehykseen luomalla arkistoinnista matalimmalla prioriteetilla oleva tehtävä funktiolla *rt_task_create* [40]. Tästä ideasta luovuttiin, sillä sen toteutus on huomattavan monimutkainen ja näin ollen virhealtis. Sen lisäksi reaaliaikasuoritus on tarkoitettu teollisuusjärjestelmän mittaamiseen ja ohjaamiseen, joten sitä ei ole hyvä jakaa arkistointioperaation kanssa.

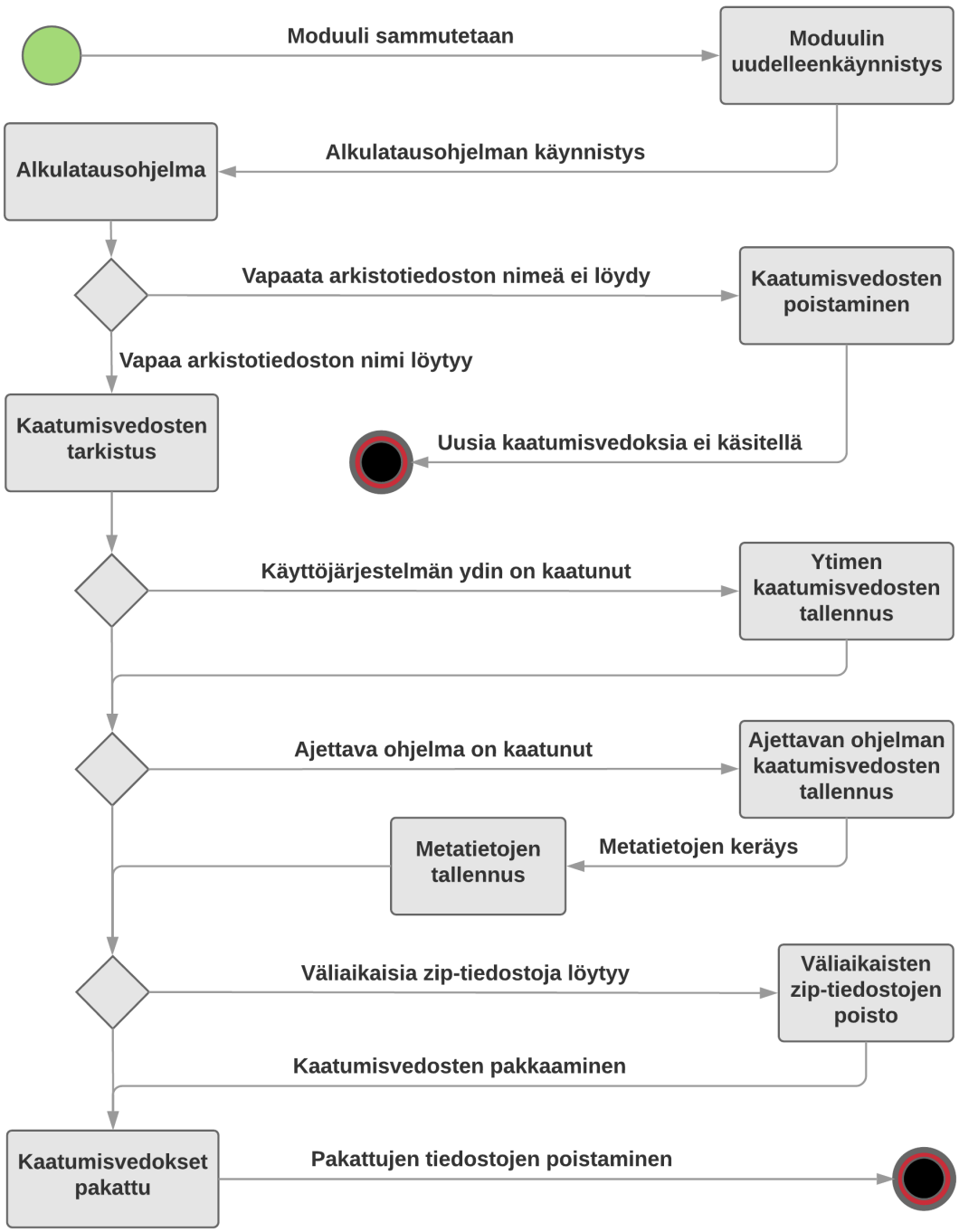
Neljäs vaihtoehto on luoda systeemikomento kaatumisvedosten pakkaamiseen ja arkistoimiseen. Automaattisen pakkaamisen ja arkistoinnin sijaan työpöytäsovellus voisi lähettää kaikille moduuleille systeemikomennon, jossa moduuleita käsketään pakkaamaan ja arkistoimaan kaatumisvedoksensa. Tällöin kaatumisvedokset voitaisiin pakata ja arkistoida hallitusti vasta silloin, kun kaatumisvedokset halutaan raportoida. Tämä lähestymistapa ratkaisisi ongelmat pakkaamisen ja arkistoimisen kanssa. Pakkaamisen ja arkistoimisen määrittelevä systeemikomento ei sopisi systeemikomentojen suunnitteluperiaatteisiin. Kuten aliluvussa 5.2 mainittiin, kohdejärjestelmässä vain välttämättömät toiminnot on hyvä toteuttaa systeemikomentoina. Kaatumisvedosten pakkaaminen ja arkistoiminen katsottiin olevan liian spesifinen toiminto systeemikomennoksi. Lisäksi toiminto ei ole välttämätön, sillä pakkaaminen ja arkistoiminen voidaan tehdä automaattisestikin.

Toteutuksessa päädyttiin ensimmäiseen vaihtoehtoon, mutta oli selvää, että arkistointi ei saa kestää kauaa. Parin sekunnin viive moduulin käynnistymisessä kaatumisen jälkeen ei haittaa kohdejärjestelmän toimintaa oleellisesti, mutta käynnistys ei saa kestää paria sekuntia pidempään. Kaikki kaatumisvedokset ja muut tiedostot on kuitenkin saatava arkistoitua yhteen tiedostoon tunnetulla nimellä, sillä tiedostonsiirrossa tiedoston nimi täytyy tietää.

Toteutuksessa päädyttiin arkistoimaan jokaisen erillisen kaatumisen kaatumisvedokset omiksi arkistotiedostoiksi. Toisin sanoen yksi arkistotiedosto kuvaa yhtä ohjelman tai ytimen kaatumista. Tällöin pakkaus ja arkistointi suoritetaan enintään parissa sekunnissa, joka on hyväksyttävää. Arkistotiedostot nimetään aiemman mallin mukaan, mutta tiedoston nimeen lisätään juokseva numerointi. Esimerkiksi mikäli tiedostonimi juoksevalla numerolla nolla on jo käytetty, koetetaan tiedostonimeä juoksevalla numerolla yksi. Tätä jatketaan, kunnes löydetään käyttämätön tiedostonimi. Toinen moduuli voi tällöin kysyä tiedostoja numerojärjestyksessä. Tätä tarkastellaan tarkemmin seuraavissa aliluvuissa.

Ratkaisussa on suurena ongelmana taaksepäin yhteensopivuus. Kaatumisvedoksia on jo kerätty moduuleilla, jotka ovat olleet tuotannossa vuosia. Hyvin epätodennäköisessä skenaariossa moduuleille on kerääntynyt suuri määrä kaatumisvedoksia. Skenaario on kuitenkin mahdollinen, joten siihen täytyy varautua. Jos moduuleille asennetaan tässä diplomityössä käsitelty ominaisuus, moduuli yrittäisi arkistoida kaikki olemassa olevat tiedostot. Tämä ei ole mahdollista, sillä aiemmin mainittu vahtikoira-ajastin katkaisisi alkulatausohjelman suorituksen, sillä arkistoinnissa kestäisi liian kauan. Moduuli joutuisi pysähtyneeseen tilaan, jolloin ainut keino moduulin käynnistämiseen olisi kaatumisvedosten manuaalinen poisto tiedostojärjestelmästä. Kohdejärjestelmän käyttäjille ei kuitenkaan ole oikeuksia tehdä tällaista operaatiota. Arkistoinnissa voitaisiin ottaa vain joitain tiedostoja arkistoitavaksi kerrallaan, mutta moduuli ei voi tietää, mitkä kaatumisvedokset kuuluvat mille metatiedostoille.

Ongelmaan ei löytynyt mitään järkevää ratkaisua. Toteutuksessa päätettiin, että vanhat kaatumisvedokset poistettaisiin, kun tässä diplomityössä käsitelty ominaisuus asennettaisiin vanhoille moduuleille. Hyvänä puolena on myös se, että näin arkistointia ei tarvitse tehdä erikseen. Kun ydin tai ajettava ohjelma kaatuu, kaikki kaatumisvedokset ja metatiedostot voidaan kerätä yhteisesti tietyn hakemiston alle kaatumisen yhteydessä ja kaatumisen jälkeen. Toteutus yksinkertaistui merkittävästi, sillä nyt riittää, että kaikki tiedostot pakataan kerralla yhteen zip-tiedostoon. Lopullinen ratkaisu on esitelty kuvassa 13.



Kuva 13. *Tilakaavio kaatumisvedosten pakkaamisesta.*

Paketoinnissa käytetään edelleen samaa juoksevaa numerointia kuten aiemmin. Ohjelman tai ytimen kaatumisen on harvinainen tapahtuma, joten 20 kaatumisen kaatumisvedosten säilyttäminen riittää hyvin. Kun ohjelma kaatuu, arkistotiedostojen nimet tarkistetaan. Mikäli moduulilla on jo 20 arkistotiedostoa, uudet kaatumisvedokset poistetaan. Ihanteellisesti kaikki kaatumisvedokset tallennetaan ja lähetetään kehittäjille, mutta näin ei voida tehdä, sillä tallennustilaa ei ole rajattomasti. Kehittäjiä todennäköisimmin kiinnostaa ensimmäinen kaatumisvedos, sillä ensimmäinen kaatuminen on voinut johtaa muihin kaatumisiin.

Kaikkien kaatumisvedosten pakkaamisessa on sama ongelma kuin yksittäisten kaatumisvedosten pakkauksessa. Moduuli voidaan sammuttaa tai käynnistää uudelleen kesken pakkaamisen. Tästä syystä käytetään samaa libzip-tiedostoa, jonka toiminta esitettiin edellisessä aliluvussa. Mikäli moduuli käynnistetään uudelleen pakkaamisen aikana, moduulille jää väliaikainen zip-tiedosto tiedostojärjestelmään. Kuten aiemmin mainittiin, väliaikainen zip-tiedosto voidaan poistaa alkulatausohjelmassa.

Kun kaatumisvedokset on pakattu, ne voidaan poistaa, sillä tällöin libzip-kirjasto on turvallisesti pakannut kaatumisvedokset zip-tiedostoon. Mikäli moduulilla on kaatumisvedoksia, lopputuloksena pitäisi olla arkistotiedosto, joka on nimetty moduulin tunnisteen ja juoksevan numeroinnin mukaan.

6.3 Kaatumisvedosten siirron aloittaminen

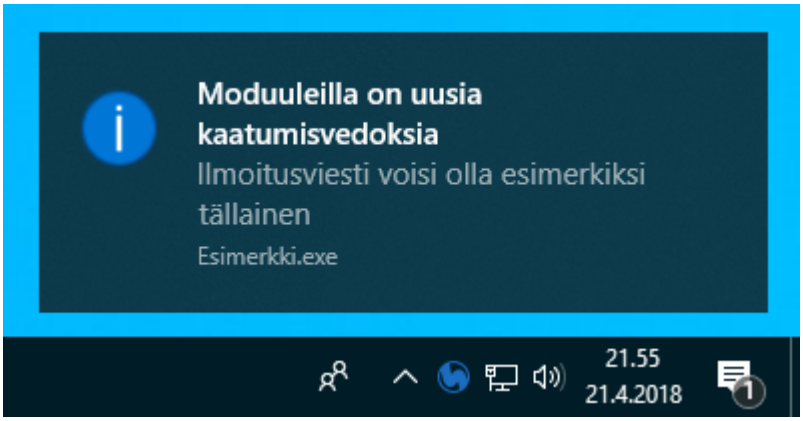
Kaatumisvedosten pakkaamisen jälkeen moduulilla on vähintään yksi valmis arkistotiedosto, joka on valmis siirrettäväksi, tai moduulilla ei ole lainkaan arkistotiedostoja. Työpöytäsovelluksen on tiedettävä molemmista tiloista.

Olemassaolon tarkistuksessa on päätettävä, ovatko moduulit passiivisia vai aktiivisia. Mikäli moduulit ovat passiivisia, moduuleilta kysyttäisiin, onko niillä kaatumisvedoksia. Käytännössä työpöytäsovellus kysyisi kaatumisvedosten olemassaoloa systeemikomennolla. Toteutettava systeemikomento olisi liian spesifinen, joten sitä ei toteutettu. Aliluvussa 5.4 kuvattu tilan kuuntelija ei ollut vielä olemassa tässä vaiheessa suunnitelmia. Kun suunnitelmat tilan kuuntelijaan alkoivat realisoitumaan, kaatumisvedosten olemassaolon tarkistus päätettiin nopeasti toteuttaa tilan kuuntelijalla. Moduulit ovat siis aktiivisesti kuuluttamassa tietoa kaatumisvedosten olemassaolosta.

Kaikilla moduuleilla kaatumisvedokset tarkastetaan kerran minuutissa, jotta moduulia ei rasitettaisi jatkuvilla I/O-operaatioilla. Varsinainen tila kuulutetaan 15 sekunnin välein, kuten aliluvussa 5.4 mainittiin. Jos moduulilla ei ole kaatumisvedoksia, tarkistuksia ei enää tehdä lainkaan ohjelman ollessa käynnissä. Mikäli ajettava ohjelma tai ydin kaatuu, moduuli käynnistetään uudelleen, ja uudet kaatumisvedokset pakataan alkulatausohjelmassa. Tällöin myös kaatumisvedosten tarkistus aloitetaan uudelleen. Moduulin ollessa

käynnissä uusia kaatumisvedoksia ei voida luoda, joten kaatumisvedosten olemassaoloa ei ole järkevää tarkastaa joka minuutti, mikäli kaatumisvedoksia ei ole olemassa.

Kaatumisvedosten olemassaolo voidaan integroida osaksi moduulien kuulutusviestejä. Moduuleilla joko on tai ei ole kaatumisvedoksia, joten tila voidaan esittää yhdellä bitillä kuulutusviestissä. Jokainen moduuli on tällöin tietoinen, millä moduuleilla kaatumisvedoksia on olemassa. Jos työpöytäsovelluksella otetaan yhteyttä mihin tahansa kommunikaatiomoduuliin, moduulien tila voidaan välittää työpöytäsovellukselle.



Kuva 14. *Esimerkki mahdollisesta ilmoitusviestistä Windows-käyttöjärjestelmässä.*

Kun työpöytäsovellus huomaa, että millä tahansa moduulilla on pakattuja kaatumisvedoksia, asiasta on ilmoitettava käyttäjälle. Windows-ympäristössä ilmoittaminen voidaan tehdä ilmaisinalueen ilmoitusviestillä (*system tray message*) [29]. Ilmoitusviestiä klikkaamalla käyttäjä voidaan ohjata erilliseen dialogiin, josta kaatumisvedosten keräysprosessi voidaan aloittaa ja josta käyttäjä näkee keräysprosessin tilan. Esimerkki mahdollisesta ilmoitusviestistä löytyy kuvasta 14.

6.4 Kaatumisvedosten siirtäminen

Kaatumisvedosten siirtäminen kommunikaatiomoduulilta työpöytäsovellukselle tehdään SFTP-protokollalla. Ennen kuin SFTP-protokollaa voi käyttää, kommunikaatiomoduulin ja työpöytäsovelluksen välille on luotava SSH-yhteys. Työpöytäsovelluksella voidaan lähettää systeemikomento kommunikaatiomoduulille SSH-palvelun käynnistämiseksi. Tietoturvasyistä SSH-palvelu ei ole jatkuvasti päällä, joten se täytyy käynnistää erikseen. Kun SSH-palvelu on käynnistetty, SFTP-istunto voidaan aloittaa.

Ominaisuuden toteutuksessa kävi ilmi, että SSH-palvelun käynnistäminen ja SFTP-istunnon aloittaminen on raskas operaatio. Kommunikaatiomoduulin prosessorikuorma on 100 prosenttia parin sekunnin ajan. Tämä aiheutti ongelmia kohdejärjestelmässä. Jotkin tärkeät kommunikaatio-operaatiot aikakatkaistiin korkean prosessorikuorman takia. Reaaliaikasuorituksen kannalta hetkellinen korkea prosessorikuorma ei ole hyväksyttävää, sillä

huonolla ajoituksella korkea prosessorikuorma voi häiritä järjestelmän toimintaa. Korkean prosessorikuorman syytä ei saatu selville, sillä syy voi olla kolmannen osapuolen SSH-kirjastossa. Näin ollen raportointi voidaan suorittaa turvallisesti vain silloin, kun teollisuusjärjestelmä ei ole käynnissä. Tämä ei ole suuri ongelma, sillä teollisuusjärjestelmän tila voidaan tarkistaa ohjelmallisesti. Käyttäjälle myös ilmoitetaan, että raportointia ei voida tehdä teollisuusjärjestelmän ollessa päällä. Vastaavat rajoitukset ovat uuden ohjelmiston lataamisessa ja asennuksessa. Tällä ratkaistaan myös ongelma, jossa liiallinen kommunikaatio häiritsisi kohdejärjestelmän toimintaa, vaikka raportointia ei voisi tehdäkään kaikissa tilanteissa.

SFTP-protokollalla tiedostonsiirto kommunikaatiomoduulin ja työpöytäsovelluksen välillä on yksinkertaista. SFTP-protokollalla työpöytäsovellus voi kysyä tiedostolistauksen kommunikaatiomoduulilta. Tiedostoja voidaan ladata ja poistaa kommunikaatiomoduulilta.

Muilta moduuleilta tiedostot on siirrettävä ensin kommunikaatiomoduulille. Sen jälkeen työpöytäsovellus voi siirtää tiedostot kommunikaatiomoduulilta SFTP-protokollalla. Tiedostot voitaisiin periaatteessa siirtää valmiiksi kommunikaatiomoduulille, sillä moduulien välinen tiedostonsiirto ei ole raskas operaatio. Näin ei voida tehdä, sillä kommunikaatiomoduulin tallennustila ei riitä kaikkien pakattujen kaatumisvedosten tallennukseen. Kohdejärjestelmä on tarpeen mukaan hyvin skaalautuva. Joissain tapauksissa kommunikaatiomoduuli voisi säilyttää kaikkien moduulien kaatumisvedoksia, mutta monissa tapauksissa tämä ei ole mahdollista. Pienien järjestelmien varalle ei ole kuitenkaan järkevää tehdä omaa erikoistoteutusta. Tämän takia arkistotiedostot ovat siirrettävä yhdeltä moduulilta kerrallaan. Tämä tarkoittaa sitä, että jokainen moduuli säilyttää omia arkistotiedostojaan niin kauan, kunnes ne siirretään kommunikaatiomoduulille.

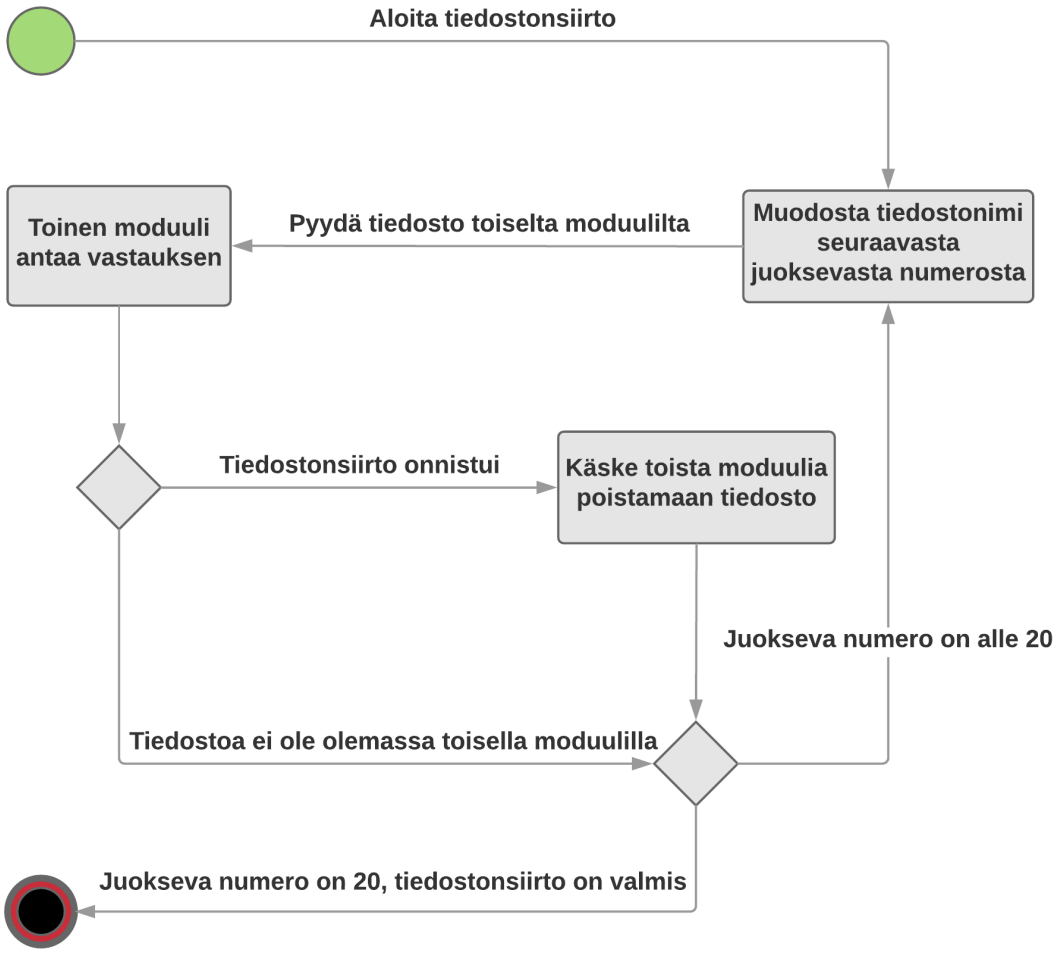
Kaatumisvedosten keräysprosessi alkaa SFTP-istunnon aloittamisessa. Tässä vaiheessa työpöytäsovellus tietää, millä moduuleilla on kaatumisvedoksia. Mikäli kommunikaatiomoduulilla on arkistotiedostoja, ne ladataan ensin. Jos lataaminen onnistuu, tiedostot poistetaan. Jos muilla moduuleilla ei olekaan kaatumisvedoksia, raportoinnissa voidaan siirtyä kaatumisvedosten käsittelyyn, jota käsitellään seuraavassa aliluvussa.

Moduulien välisessä tiedostonsiirrossa on kaksi vaihtoehtoa. Ensimmäinen vaihtoehto on lähettää kommunikaatiomoduulille systeemikomento, jolla kommunikaatiomoduulia komennetaan hakemaan tiedostot toiselta moduulilta. Toinen vaihtoehto on lähettää muulle kuin kommunikaatiomoduulille systeemikomento, jolla moduulia komennetaan lähettämään tiedostot kommunikaatiomoduulille. Kumpikin vaihtoehto on mahdollinen, sillä Ethernet-tekniikan päälle toteutetussa protokollassa on toteutettu molemmat tavat.

Ensimmäinen vaihtoehto on yksinkertainen, sillä systeemikomento voidaan lähettää vain kommunikaatiomoduulille. Kun kommunikaatiomoduuli on siirtänyt tiedostot toiselta

moduulilta, kommunikaatiomoduuli voi ilmoittaa asiasta työpöytäsovellukselle välittömästi.

Toinen vaihtoehto on hieman monimutkaisempi. Systeemikomento täytyy ensin lähettää kommunikaatiomoduulille. Sen jälkeen kommunikaatiomoduulin täytyy välittää systeemikomento eteenpäin toiselle moduulille, joka lähettää tiedostot kommunikaatiomoduulille. Ongelmana on tietää, milloin kaikki tiedostot on siirretty kommunikaatiomoduulille. Toinen moduuli tietää, milloin kaikki tiedostot on lähetetty kommunikaatiomoduulille, mutta kommunikaatiomoduuli tai työpöytäsovellus eivät tiedä tiedostosiirron tilaa. Edellisessä aliluvussa mainittiin, että kaatumisvedosten olemassaolo tarkistetaan joka minuutti. Kommunikaatiomoduuli ja työpöytäsovellus voisivat päätellä tiedostonsiirron onnistuneen, mikäli tiedostot lähettänyt moduuli kuuluttaisi, ettei sillä olisi kaatumisvedoksia enää. Tämä ei ole käytännössä mahdollista. Jos tiedostot lähetetään kommunikaatiomoduulille muutamassa sekunnissa, työpöytäsovelluksen olisi odotettava lähes minuutti seuraavan tilan vastaanottamista.



Kuva 15. *Tiedostonsiirto moduulien välillä.*

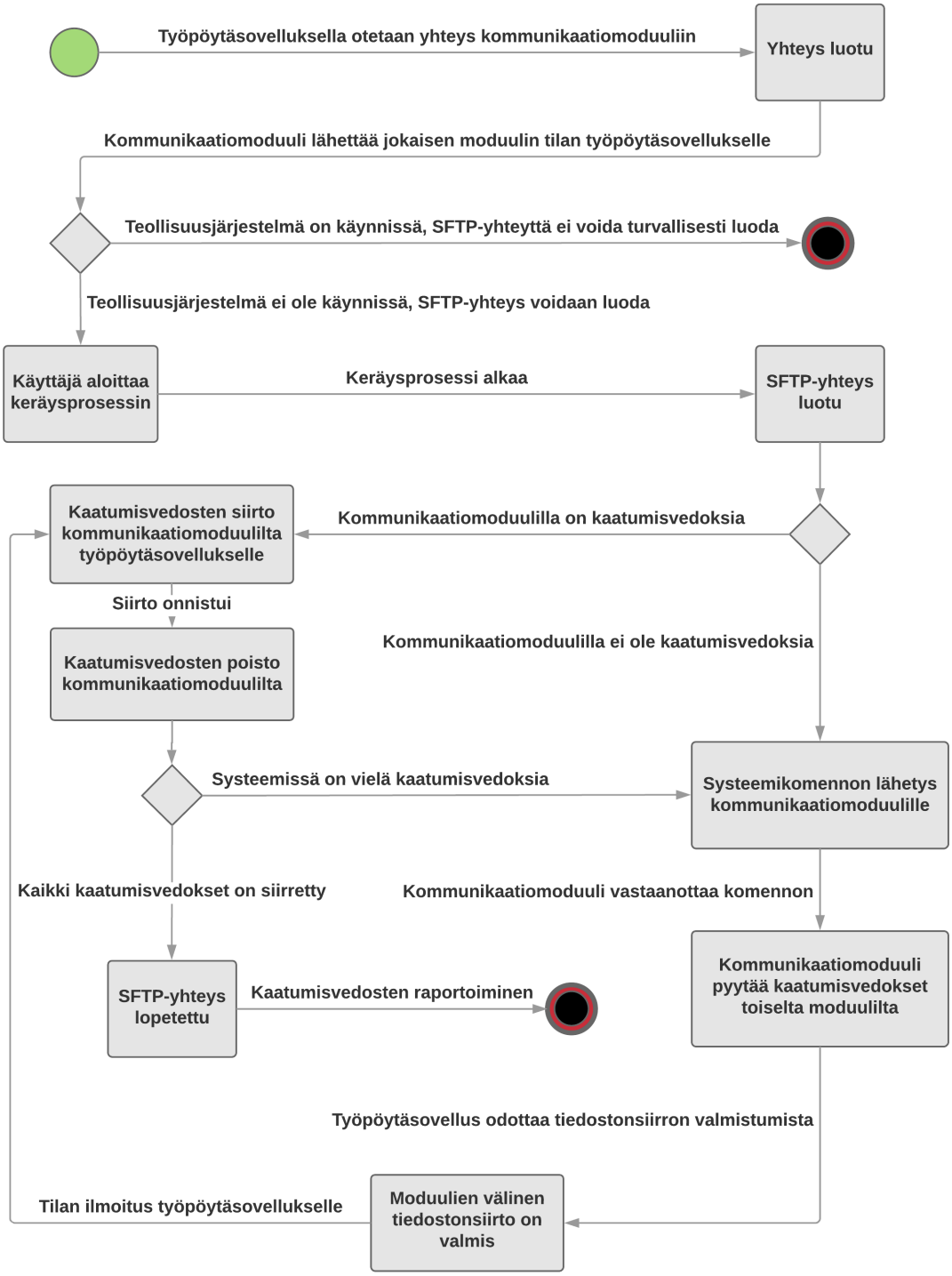
Toteutuksessa päädyttiin ensimmäiseen vaihtoehtoon, mutta siinä oli omat ongelmansa. Kuten aliluvussa 6.2 kerrottiin, alun perin ajateltiin, että kommunikaatiomoduulin olisi haettava vain yksi arkistotiedosto, joka on nimetty moduulin tunnisteen mukaan. Näin ei kuitenkaan voida tehdä, joten kommunikaatiomoduulin on vain yritettävä hakea tiedostoja. Arkistotiedostot on nimetty juoksevan numeroinnin mukaan, joten kommunikaatiomoduuli yrittää hakea tiedostoja kaikilla mahdollisilla tiedostonimillä. Mikäli tiedostoa ei ole olemassa, toinen moduuli ilmoittaa asiasta. Tämä on käytännössä sama operaatio, kuin tiedoston olemassaolon kysyminen, joten ratkaisu on riittävä. Kuvassa 15 on esitelty koko prosessi.

Kun arkistotiedosto on siirretty toiselta moduulilta, se täytyy myös poistaa, ettei samaa tiedostoa raportoida uudelleen. Tätä varten luotiin yleiskäyttöinen systeemikomento tiedoston poistamiseen toiselta moduulilta. Systeemikomento sopi systeemikomentojen suunnitteluperiaatteisiin, sillä tiedoston poistaminen on hyvin yleinen operaatio, jota muutkin moduulin komponentit voivat käyttää. Kun tiedosto on siirretty toiselta moduulilta kommunikaatiomoduulille, kommunikaatiomoduuli lähettää systeemikomennon toiselle moduulille. Systeemikomennolle annetaan parametriksi tiedoston polku ja nimi. Kun toinen moduuli vastaanottaa komennon, tiedosto poistetaan.

Kun tiedostonsiirto on käynnissä, työpöytäsovellus odottaa sen päättymistä. Työpöytäsovelluksen on tiedettävä tiedostonsiirron tila, sillä arkistotiedostoja ei voi ladata kommunikaatiomoduulilta ennen kuin tiedostot on siirretty kommunikaatiomoduulille. Tähän hyödynnetään samaa tilan kuuntelijaa, jota hyödynnetään arkistoitujen kaatumisvedosten olemassaolon ilmoittamiseen. Kun tiedostonsiirto aloitetaan, työpöytäsovellukselle lähetetään välittömästi tila, jossa ilmoitetaan tiedostonsiirron olevan käynnissä. Kun kaikki tiedostot on siirretty, työpöytäsovellukselle lähetetään välittömästi tila, jossa ilmoitetaan tiedostonsiirron olevan valmis. Samaan aikaan kommunikaatiomoduuli lähettää toiselle moduulille viestin, jossa toista moduulia käsketään poistamaan arkistoidut kaatumisvedokset.

Tiedostonsiirron jälkeen kommunikaatiomoduulilla on toisen moduulin arkistoidut kaatumisvedokset. Samalla SFTP-istunnolla tiedostot voidaan siirtää ja poistaa. Sen jälkeen työpöytäsovellus tarkistaa, onko joillain moduuleilla vielä kaatumisvedoksia. Jos on, työpöytäsovellus lähettää jälleen viestin kommunikaatiomoduulille, jossa sitä käsketään noutamaan arkistoidut kaatumisvedokset toiselta moduulilta.

Koko siirtoprosessissa on siis kolme vaihetta. Ensin työpöytäsovellus lähettää systeemikomennon kommunikaatiomoduulille. Sen jälkeen kommunikaatiomoduuli pyytää arkistoituja kaatumisvedoksia toiselta moduulilta. Lopuksi työpöytäsovellus hakee arkistoidut kaatumisvedokset kommunikaatiomoduulilta. Tätä prosessia toistetaan niin kauan kunnes kaikki kaatumisvedokset kaikilta moduuleilta on haettu.



Kuva 16. *Korkean tason tilakaavio siirto- ja raportointiprosessista.*

Kuvassa 16 on esitelty koko siirtoprosessi. Lopputuloksena työpöytäsovellus on kerännyt kaikilta moduuleilta arkistoidut kaatumisvedokset. Lisäksi millään moduulilla ei ole enää arkistoituja kaatumisvedoksia. Tämän jälkeen voidaan siirtyä kaatumisvedosten käsittelyyn ja raportoimiseen työpöytäsovelluksella.

6.5 Kaatumisvedosten käsittely ja raportointi

Kohdejärjestelmässä myös työpöytäsovellus voi kaatua. Windows-ympäristössä kaatumisvedosten käsittely on huomattavasti helpompaa kuin kaatumisvedosten käsittely usealla sulautetulla laitteella. Windows-ympäristössä on useita eri kirjastoja, jotka voivat automaattisesti kerätä kaatumisvedoksen ja tarvittavat tiedot ja lähettää ne eteenpäin.

Moduulien kaatumisvedokset on lähetettävä kehittäjille, mutta lähettäminen ei ole triviaalia. Työpöytäsovelluksen puolella täytyy olla komponentti, joka toimii asiakkaana. Kehittäjien puolella on oltava komponentti, joka toimii palvelimena.

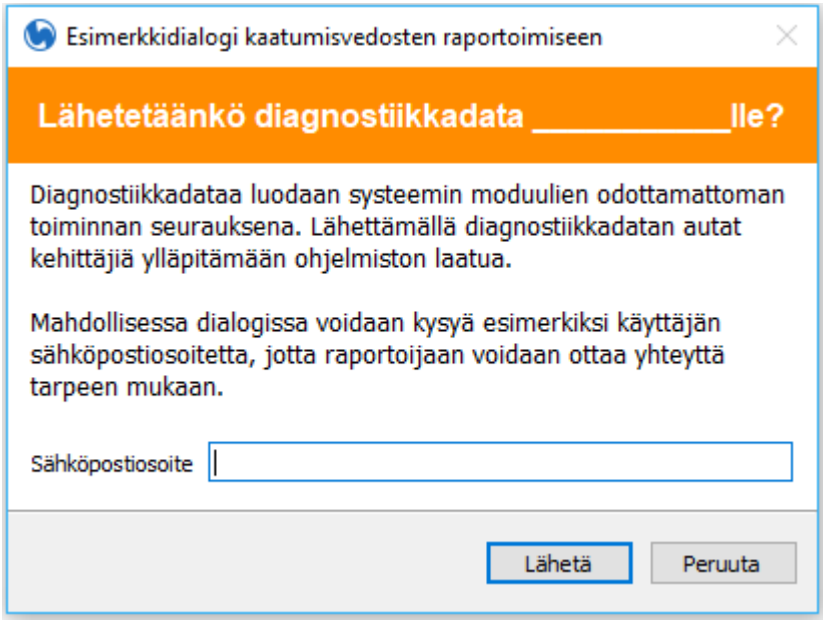
Työpöytäsovelluksella käytetään CrashRpt-kirjastoa kaatumisvedosten automaattiseen käsittelyyn ja raportoimiseen. CrashRpt-kirjasto käsittelee kaatumiseen johtavia poikkeuksia, kerää tietoa kaatumisesta ja lähettää kaatumisraportin Internetin yli. CrashRpt-kirjasto voi lähettää kaatumisraportteja HTTP-protokollan (*Hypertext Transfer Protocol*) yli tai sähköpostitse. [11]

Palvelimella, joka vastaanottaa kaatumisraportit, ajetaan CrashFix-ohjelmaa. CrashRpt-kirjasto on tuettuna CrashFix-ohjelmassa, eli CrashFix-ohjelma osaa jäsentää CrashRpt-kirjaston lähettämät kaatumisraportit. CrashFix-ohjelma tarjoaa palvelimella web-sovelluksen, jonka avulla kehittäjät voivat selata kaatumisvedoksia ja muita hyödyllisiä tietoja suoraan verkkoselaimella. [9]

Moduulien kaatumisvedosten raportoimiseen ei haluta käyttää enempää resursseja kuin on tarpeen. CrashRpt-kirjasto ja CrashFix-ohjelma tarjoavat jo työkalut kaatumisvedoksiin raportoimiseen ja käsittelyyn, joten niitä voidaan hyödyntää kaatumisvedosten raportoimiseen.

CrashRpt-kirjasto on suunniteltu kaatumisvedosten automaattiseen käsittelyyn ensisijaisesti Windows-käyttöjärjestelmässä natiivin C++-ohjelman poikkeuksille [11]. Moduulien kaatumisvedosten raportoinnissa työpöytäsovellus ei ilmiselvästi kaadu, joten kaatuminen täytyy simuloida. CrashRpt-kirjasto tarjoaa funktiot tiedostojen manuaaliseen lisäämiseen raporttiin ja kaatumisen simuloimiseen [10]. Moduulien kaatumisvedokset voidaan siis lisätä raporttiin manuaalisesti.

Kaatumisen simuloiminen on käytettävyyden kannalta ongelmallista, sillä CrashRpt-kirjaston tarjoama raportointidialogi näyttää samalta kuin työpöytäsovelluksen kaatuessa. Kirjastolla ei ole mahdollista muokata raportointidialogin rakennetta, mutta dialogin tekstejä on mahdollista muuttaa. Tästä huolimatta ominaisuuden ensimmäisessä versiossa käyttäjiltä tuli palautetta, jossa he ihmettelivät, miksi työpöytäsovellus halusi lähettää kaatumisvedoksia, vaikka työpöytäsovellus ei kaatunut. Monet käyttäjät assosioivat dialogin rakenteen välittömästi työpöytäsovelluksen kaatumiseen, sillä vastaavaa rakennetta käytetään monissa muissakin vastaavissa kirjastoissa.



Kuva 17. *Esimerkki mahdollisesta raportointidialogista.*

CrashRpt-kirjastolla on mahdollista raportoida kaatumisvedokset ilman graafista raportointidialogia [10]. Tätä vaihtoehtoa voidaan hyödyntää moduulien kaatumisvedosten raportointiin. Työpöytäsovellukseen toteutetaan erillinen dialogi, josta käyttäjälle käy selvästi ilmi, että käyttäjä on raportoimassa moduulien kaatumisvedoksia. Esimerkki mahdollisesta dialogista esitetään kuvassa 17. Kun käyttäjä lähettää kaatumisvedokset, taustalla CrashRpt-kirjastolla simuloidaan kaatumista. Näin moduulien kaatumisvedokset voidaan lähettää kehittäjien tutkittavaksi vastaavalla tavalla kuin työpöytäsovelluksen kaatumisvedokset hyödyntämällä olemassa olevia työkaluja.

Työpöytäsovelluksen puolella ei tällä hetkellä kerrota käyttäjälle, mitä kerätyt tiedostot tarkkaan ottaen ovat. Kohdejärjestelmän käyttäjä ei välttämättä ole huomannut teollisuusjärjestelmän toiminnassa mitään erityistä tai käyttäjä ei edes tiedä, mitä kaatumisvedokset ovat. Tästä syystä kaikkea kerättyä dataa kutsutaan diagnostiikkadataksi. Käyttäjille ei haluta suoraan kertoa, että moduuleilla ajettava ohjelma tai käyttöjärjestelmän ydin on kaatunut.

6.6 Työn tavoitteiden arviointi

Kokonaisuudessaan työn tavoitteet saavutettiin. Tästä huolimatta työn eri vaiheissa olisi parannettavaa. Seuraavaksi käydään läpi joitain työn keskeisiä osa-alueita. Jokaisesta osa-alueesta kerrotaan, mikä meni hyvin ja missä olisi ollut parannettavaa.

Kaatumisvedosten varsinainen luominen toteutettiin hyvin. Ytimen kaatumisvedosten luomiseen hyödynnettiin Linux-käyttöjärjestelmän valmista *MTDoops*-moduulia, joka huolehti kaatumisvedosten tallennuksesta MTD-osiolle. Kaatumisvedosten data voitiin

lukea osiolta ja tallentaa se tiedostojärjestelmään alkulatausohjelmassa. Ajettavan ohjelman kaatumisvedokset tallennetaan suoraan tiedostojärjestelmään. Ratkaisu on riittävä, sillä molemmat kaatumisvedostyypit luodaan häiritsemättä kohdejärjestelmän toimintaa.

Kaatumisvedosten käsittelyssä suurin ongelma oli tekninen velka. Kun aliluvussa 6.1 kuvatut komponentit oli toteutettu, koko toiminnallisuudelle ei selvästi ollut selkeää suunnitelmaa. Tässä vaiheessa olisi pitänyt päättää, toteutetaanko Ethernet-tekniikan päälle kehitettyyn protokollaan tiedostolistaus vai ei. Tiedostolistauksen tukea selvästi odotettiin, sillä kaatumisvedokset pakattiin erillisiin zip-tiedostoihin. Pakkaus pienensi kaatumisvedosten kokoa merkittävästi, joten pakkaaminen oli oikea ratkaisu rajatun tallennustilan rajoitteiden kanssa. Pakkaus ei kestänyt kauaa, joten pakkaus voitiin suorittaa alkulatausohjelmassa ongelmitta.

Myöhemmin keksittiin, että tiedostonsiirto voidaan tehdä ilman tiedostolistausta keräämällä kaikki pakatut kaatumisvedokset yhteen arkistotiedostoon. Tämä oli hieman ristiriitainen lähestymistapa, sillä jokaisen tiedoston pakkaaminen ja kaikkien pakattujen tiedostojen arkistoiminen sisältää tarpeettoman monta tiedosto-operaatiota. Taaksepäin yhteensopivuuden takia näin oli kuitenkin tehtävä. Toteutuksessa ei odotettu, että arkistoiminen olisi hidasta suurilla arkistotiedostoilla. Hitaan flash-muistin ja pienitehoisen prosessorin takia arkistoiminen oli mahdotonta kaikissa tilanteissa. Taaksepäin yhteensopivuus hylättiin. Sen sijaan kaikki kaatumisvedokset pakataan kerralla yhteen zip-tiedostoon, joka nimetään moduulin tunnisteen ja juoksevan numeroinnin mukaan. Ratkaisu on yksinkertainen ja toimiva.

Tästä opittiin, että mitään keskeneräistä ei saa julkaista, varsinkin jos koko toiminnallisuuden suunnitelma on puutteellinen. Muuten kaikissa tulevissa ohjelmiston versioissa on otettava huomioon tarpeetonta taaksepäin yhteensopivuutta, joka kasvattaa teknistä velkaa merkittävästi. Ominaisuuden toteutuksessa olisi päästy helpolla, jos alun perin olisi päätetty, toteutetaanko tiedostolistaus vai ei. Nykyisellä ratkaisulla tiedostolistausta ei loppujen lopuksi tarvitakaan.

Varsinaisessa kaatumisvedosten siirrossa oli ongelmia SFTP-istunnon luomisessa työpöytäsovelluksen ja kommunikaatiomoduulin välille. Istunnon luomisen korkeaa prosessorikuormaa ei saatu selville, joten rajoituksena teollisuusjärjestelmä ei voi olla käynnissä, kun kaatumisvedokset raportoidaan, muuten kohdejärjestelmän ja teollisuusjärjestelmän toiminta voisi häiriintyä oleellisesti. Toisaalta rajoituksen ansiosta ei tarvitse miettiä, häiritseekö moduulien välinen tiedostonsiirto kohdejärjestelmän toimintaa, sillä teollisuusjärjestelmän ollessa pois päältä kohdejärjestelmällä ei ole reaaliaikaisuutta vaativia tehtäviä. Tästä huolimatta ihanteellisesti SFTP-istunnon luomisen pitäisi onnistua myös silloin, kun teollisuusjärjestelmä on käynnissä.

Muuten kaatumisvedosten siirtäminen kaikilta moduuleilta työpöytäsovellukselle onnistui hyvin. Moduulien välillä tiedostonsiirrossa täytyi käyttää luovuutta, sillä ilman tiedostolistausta ei voitu tietää, millä tiedostonimillä ja -poluilla tiedostoja täytyi pyytää. Juoksevalla numeroinnilla moduulien täytyi vain kokeilla siirtää tiedostoja kaikilla mahdollisilla tiedostonimillä. Poistaminen tiedostonsiirron jälkeen onnistui hyvin uudella systeemikomennolla.

Rajallisella tallennustilalla siirrossa ei voida siirtää kaikkien moduulien kaatumisvedoksia kommunikaatiomoduulille kerralla. Yksitellen siirto onnistui hyvin, vaikka sen toteutus olikin monimutkaisempi.

Raportoimisessa käytettiin tehokkaasti hyväksi olemassa olevia työkaluja. CrashRpt-kirjasto ja CrashFix-ohjelma olivat jo käytössä työpöytäsovelluksella, joten niiden käyttämineen raportoimiseen oli luontevaa. CrashRpt-kirjastoa käytettiin epätavanomaisesti, sillä kirjasto on erityisesti suunniteltu automaattiseen kaatumisien käsittelyyn. Tästä huolimatta kirjastoa pystyttiin soveltamaan moduulien kaatumisvedosten raportoimiseen.

Kokonaisuudessaan raportointijärjestelmä saatiin integroitua kohdejärjestelmään. Eri vaiheissa oli monia haasteita, mutta lopputulos on toimiva. Raportointijärjestelmä saatiin toteutettua niin, ettei se häiritse kohdejärjestelmän toimintaa, vaikka raportointijärjestelmässä on joitain rajoitteita. Ohjelman tai ytimen kaatuminen pitäisi olla hyvin harvinainen tapahtuma, joten rajoitteet eivät haittaa raportointijärjestelmän toimintaa merkittävästi.

6.7 Jatkokehitys

Kaatumisvedosten luomisessa ja käsittelyssä ei ole tällä hetkellä parantamisen varaa. Loppujen lopuksi kaatumisvedosten pakkaaminen on melko yksinkertainen operaatio. Arkistotiedoston nimeäminen juoksevan numeroinnin mukaan takaa sen, että millään moduulilla ei säilytetä enempää kuin 20 kaatumisen kaatumisvedoksia.

Arkistotiedostojen siirto moduulien välillä onnistuu tällä hetkellä hyvin. Tiedostolistaus ei tunnu tässä vaiheessa enää toteuttamisen arvoiselta, sillä juoksevan numeroinnin ansiosta tiedostoja voidaan siirtää helposti.

SFTP-yhteyden luomisessa oli ongelmia prosessorikuorman kanssa. Korkean prosessorikuorman syytä ei saatu selville, joten on syytä selvittää, mikä sen aiheuttaa. Teollisuusjärjestelmän käytön kannalta teollisuusjärjestelmän sammuttaminen kaatumisvedosten raportoimisen ajaksi on hyvin vaivalloista. Toisaalta jos teollisuusjärjestelmä on toiminut odottamattomasti ajettavan ohjelman kaatumisen takia, teollisuusjärjestelmän sammuttaminen olisi suositeltavaa tehdä joka tapauksessa. Tästä huolimatta kaatuminen pitäisi olla hyvin harvinainen tapahtuma. Mikäli ajettava ohjelma tai ydin kuitenkin kaatuu, kaatumisvedosten raportointi on mahdollista.

Raportointijärjestelmää voidaan laajentaa tarpeen mukaan helposti. Kohdejärjestelmässä voi tulevaisuudessa olla muutakin dataa, joka kiinnostaisi kehittäjiä. Moduuleilta voitaisiin kerätä yleistä diagnostiikkadataa, kuten esimerkiksi tietoa kommunikaatiovirheistä.

Raportoiminen voisi olla paremmin automatisoitu. Tällä hetkellä koko prosessi on melko manuaalinen. Käyttäjän on ensin reagoitava ilmoitukseen uusista kaatumisvedoksista. Koska ne mielletään diagnostiikkatiedostoiksi työpöytäsovelluksessa, käyttäjä voi sivuuttaa ilmoituksen. Jos SFTP-yhteys voitaisiin luoda teollisuusjärjestelmän ollessa päällä, se voitaisiin tehdä taustalla automaattisesti. Myös muita tiedostonsiirtoprotokollia voitaisiin tutkia. Näin myös moduulien välinen tiedostonsiirto voitaisiin tehdä automaattisesti.

Raportoimisjärjestelmän tarkoitus ei ole tullut kaikille kohdejärjestelmän käyttäjille selväksi. Koska kohdejärjestelmä on hyvin vikasietoinen, monia kaatumisia ei voi huomata teollisuusjärjestelmän toimintaa seuraamalla. Jotkin käyttäjät ovatkin ihmetelleet, miten moduuleilla voi olla kaatumisvedoksia, sillä teollisuusjärjestelmä on toiminut silmämääräisesti normaalisti. Tästä syystä käyttäjät ovat luulleet, että itse raportointijärjestelmä ei toimi kunnolla. Raportointi olisi joko tehtävä automaattisesti tai raportointijärjestelmän tarkoitusta olisi selvennettävä.

Raportointijärjestelmä ei ole ollut vielä tuotannossa nykyisessä muodossaan. Vain pieni määrä kohdejärjestelmän käyttäjiä on päässyt testaamaan ominaisuutta käytännössä. Vaikka ominaisuuden testauksessa ei ole havaittu mitään erityisiä ongelmia, tuotannossa voi ilmetä erilaisia ongelmia. Kaikkiin ongelmiin ei voi varautua, sillä raportointijärjestelmässä on niin monta liikkuvaa osaa. Tässä vaiheessa ei voida tietää, onko 20 kaatumisvedosten säilyttäminen riittävää, jos kohdejärjestelmä on ollut käynnissä vuosia ilman huoltokäyntiä. Testausten perusteella kaatumisraportit ovat olleet käyttökelpoisia, mutta voi olla rajatapauksia, joissa tietoa tarvitaankin enemmän.

7. YHTEENVETO

Tässä diplomityössä suunniteltiin kaatumisvedosten raportointi sulautetussa automaatiojärjestelmässä. Työssä lähdettiin liikkeelle esittelemällä kohdejärjestelmän toiminta. Kohdejärjestelmä koostuu sulautetuista Linux-laitteista, joihin viitattiin työn aikana moduuleina, sillä kohdejärjestelmä on hyvin modulaarinen. Kohdejärjestelmään kuuluu merkittävänä osana myös työpöytäsovellus, jonka tärkeimpänä tehtävänä on uuden ohjelmiston asennus moduuleille, kohdejärjestelmän konfigurointi ja seuraaminen sekä ongelmien kartoittaminen.

Seuraavaksi käytiin läpi kaatumisvedosten teoriaa. Kohdejärjestelmässä kerätään kahdenlaisia kaatumisvedoksia: Linux-käyttöjärjestelmän ytimen kaatumisvedoksia sekä moduulilla ajettavan ohjelman kaatumisvedoksia. Kaatumisvedokset pakataan tilan säästämiseksi.

Kohdejärjestelmän normaalissa toiminnassa ja kaatumisvedosten raportoinnissa kommunikaatio on suuressa osassa. Tästä syystä ensin käytiin läpi tietoliikennetekniikoiden teoriaa. Moduulien välillä hyödynnetään Ethernet-tekniikkaa. Korkeammalla tasolla kommunikaatiomoduulin ja työpöytäsovelluksen välillä hyödynnetään TCP/IP-protokollayhdistelmää. SSH- ja SFTP-protokollat ovat merkittävässä osassa työpöytäsovelluksen ja kommunikaatiomoduulin välisessä tiedostonsiirrossa.

Sekä moduulien välillä että työpöytäsovelluksen ja kommunikaatiomoduulin välillä käytetään räätälöityjä protokollia. Ylemmällä tasolla TCP/IP-protokollayhdistelmän päälle on rakennettu räätälöity protokolla työpöytäsovelluksen ja kommunikaatiomoduulin välille. TCP/IP-protokollayhdistelmä huolehtii muun muassa reitityksestä ja luotettavasta tiedonsiirrosta. Alemmalla tasolla Ethernet-tekniikan päälle on rakennettu räätälöity protokolla, jota käytetään moduulien välillä. Protokollaan on otettu vaikutteita TCP/IP-protokollayhdistelmästä, sillä pelkkä Ethernet-tekniikka ei tarjoa muun muassa luotettavaa tiedonsiirtoa.

Kaatumisvedosten luominen ja käsittely oli toteutettu jo valmiiksi ennen tämän diplomityön ominaisuuden toteuttamista. Toteutettu osuus ei toiminut täysin uuden ominaisuuden kanssa puutteellisen aikaisemman suunnitelman ja moduulien rajoitusten takia. Vanha toteutus hylättiin lopulta. Sen sijaan toteutettiin helpommin ylläpidettävä ja hallitumpi tapa käsitellä kaatumisvedoksia.

Kaatumisvedosten siirtämisessä ei ollut suuria ongelmia. Ainut ongelma on SFTP-yhteyden luomisen ongelma prosessorikuorman kanssa, mutta se vain rajoittaa, milloin raportoiminen voidaan tehdä. Teollisuusjärjestelmän luonteen takia se ei haittaa merkittävästi.

Työhön saatiin toteutettua kaikki vaaditut osat, vaikka eri osissa olikin erilaisia ongelmia. Työn alussa esitettiin seuraava tutkimuskysymys:

- Miten kaatumisvedosten raportointi voidaan integroida osaksi automaatiojärjestelmän toimintaa häiritsemättä automaatiojärjestelmän reaaliaikaista toimintaa?

Integrointi itsessään oli haastavaa, sillä koko raportointijärjestelmässä on hyvin monta liikkuvaa osaa. Tämän lisäksi työpöytäsovelluksen ja moduulien ohjelmointiympäristöt ovat hyvin erilaisia. Niiden väliin on kehitetty räätälöityjä kommunikaatioprotokollia, mutta yhteensovittamisessa oli vaikeuksia.

Erityisesti kaatumisvedosten käsittelyssä oli monia haasteita. Jos käsittelyn vanha toteutus olisi säilytetty, uusi toteutus olisi ollut liian virhealtis. Toisaalta täysi taaksepäin yhteensopivuus menetettiin, mutta uudesta toteutuksesta saatiin yksinkertaisempi. Uusi toteutus ei myöskään häiritse kohdejärjestelmän toimintaa.

Kaatumisvedosten siirtäminen ei häiritse kohdejärjestelmän toimintaa, mutta toisaalta varsinainen ongelma saatiin vain piilotettua. SFTP-yhteyden luominen aiheuttaa edelleen hetkellisesti korkean prosessorikuorman, joten sitä ei voida tehdä esimerkiksi silloin, kun teollisuusjärjestelmä on käynnissä. Kaatumisvedosten siirto moduulien välillä on melko kevyttä, joten sen ei pitäisi teoriassa häiritä kohdejärjestelmää.

Kokonaisuudessaan raportointijärjestelmä integroitiin osaksi kohdejärjestelmän toimintaa. Kohdejärjestelmän toimintaa ei myöskään häiritä merkittävästi missään vaiheessa.

LÄHTEET

[1] M.M. Alani, Guide to OSI and TCP/IP Models, 1st ed. Springer, Cham, 2014, 69 p.

[2] N. Alpern, R. Shimonski, Eleventh Hour Network+, Syngress, 2010, 208 p.

[3] A. Apostolico, M. Comin, L. Parida, Bridging Lossy and Lossless Compression by Motif Pattern Discovery, Electronic Notes in Discrete Mathematics, Vol. 21, 2005, pp. 219-225.

[4] J. Axelson, Embedded Ethernet and Internet Complete, Lakeview Research, Madison, 2003, 482 p.

[5] D.J. Barrett, R.E. Silverman, R.G. Byrnes, SSH, The Secure Shell: The Definitive Guide, 2nd ed. O'Reilly, Sebastopol (CA), 2009, 672 p.

[6] D.P. Bovet, M. Cesati, Understanding the Linux kernel, 3rd ed. O'Reilly, Sebastopol (CA), 2006, 923 p.

[7] A. Burdett, J. Jaworski, T. Ng, M. Scheer, A. Cumming, F. Hurvid, BCS Glossary of Computing and ICT, 13th ed. BCS Learning & Development Limited, 2013, 473 p.

[8] A. Butterfield, G.E. Ngondi, A. Kerr, A Dictionary of Computer Science, 7th ed. Oxford University Press, 2016, 608 p.

[9] CrashFix, Architecture Overview. Saatavissa (viitattu 19.3.2018): http://crashfix.sourceforge.net/doc/html/architecture_overview.html.

[10] CrashRpt, CrashRpt Functions. Saatavissa (viitattu 19.3.2018): http://crashrpt.sourceforge.net/docs/html/group___crash_rpt_a_p_i.html.

[11] CrashRpt, Getting Started. Saatavissa (viitattu 19.3.2018): http://crashrpt.sourceforge.net/docs/html/getting_started.html.

[12] E. Dumbill XML Watch: Bird's-eye BEEP. Saatavissa (viitattu 17.2.2018): http://www.beepcore.org/articles/x-beep-pdf.pdf.

[13] J. Edwards, R. Bramante, Networking Self-Teaching Guide OSI, TCP/IP, LAN's, MAN's, WAN's, Implementation, Management, and Maintenance, 1st ed. John Wiley & Sons, Hoboken, 2009, 864 p.

[14] R. Fox, W. Hao, Internet Infrastructure : Networking, Web Services, and Cloud Computing, 1st ed. CRC Press, Milton, 2017, 633 p.

[15] FreeBSD, Manual Pages. Saatavissa (viitattu 10.2.2018): https://www.freebsd.org/cgi/man.cgi?query=tar&sektion=5.

[16] W. Goralski, The Illustrated Network: How TCP/IP Works in a Modern Network, 2nd ed. Morgan Kaufmann Publishers, US, 2017, 936 p.

[17] L. Guo, J. Lawall, G. Muller, Oops! where did that code snippet come from? Proceedings of the 11th Working Conference on mining software repositories, ACM, pp. 52-61.

[18] W.V. Hagen, The Definitive Guide to GCC, 2nd ed. Springer Verlag, DE, 2014, 505 p.

[19] T. Herbert, Linux TCP/IP Stack : Networking for Embedded Systems, 1st ed. Charles River Media, Herndon, 2004, 586 p.

[20] M.E. Ilchenko, A.V. Moshinskaya, L.A. Urywsky, Levels separation and merging in the OSI reference model for information–telecommunication systems, Cybernetics and Systems Analysis, Vol. 47, Iss. 4, 2011, pp. 598-605.

[21] P. Iyappan, K.S. Arvind, N. Geetha, S. Vanitha, Pluggable Encryption Algorithm In Secure Shell(SSH) Protocol, 2009 Second International Conference on Emerging Trends in Engineering & Technology, pp. 808-813.

[22] M. Kerrisk, The Linux programming Interface: A Linux and UNIX System Programming Handbook, 1st ed. No Starch Press, San Francisco, 2010, 1554 p.

[23] T. Lammle, TCP / IP, John Wiley & Sons, Newark, 2017, 114 p.

[24] A. Langiu, On parsing optimality for dictionary-based text compression -- the Zip case, Journal of Discrete Algorithms, Vol. 20, 2013, pp. 65-70.

[25] J. Lee, I. Shin, A. Easwaran, Convex optimization framework for intermediate deadline assignment in soft and hard real-time distributed systems, The Journal of Systems and Software, Vol. 85, Iss. 10, 2012, pp. 2331-2339.

[26] Libzip, Libzip documentation. Saatavissa (viitattu 17.2.2018): https://libzip.org/documentation/libzip.html.

[27] R. Love, Linux System Programming, 2nd ed. O'Reilly, Sebastopol, CA u.a, 2007, 400 p.

[28] N. Matloff, P.J. Salzman, Art of Debugging with GDB, DDD, and Eclipse, 1st ed. No Starch Press, Daly City, California, 2008, 280 p.

[29] Microsoft, Notifications and the Notification Area. Saatavissa (viitattu 11.3.2018): https://msdn.microsoft.com/en-us/library/windows/desktop/ee330740(v=vs.85).aspx.

[30] M. Moore, S. Hancock, Tru64 UNIX Troubleshooting: Diagnosing and Correcting System Problems, 1st ed. Digital Press, 2002, 336 p.

[31] P. Olivier, J. Boukhobza, A hardware time manager implementation for the Xenomai real-time kernel of embedded Linux, ACM SIGBED Review, Vol. 9, Iss. 2, 2012, pp. 38-42.

[32] A. Rubini, J. Corbet, Linux device drivers, 2nd ed. O'Reilly, Beijing [u.a.], 2001, 592 p.

[33] R. Salz Beep BEEP! Saatavissa (viitattu 17.2.2018): http://www.xml.com/pub/a/2002/10/16/ends.html.

[34] J. Seo, S. Lee, T. Shon, A study on memory dump analysis based on digital forensic tools, Peer-to-Peer Networking and Applications, Vol. 8, Iss. 4, 2015, pp. 694-703.

[35] R. Sharp, Principles of Protocol Design, Springer, Berlin, Heidelberg, 2008, 406 p.

[36] C. Simmonds, Mastering Embedded Linux Programming, Packt Publishing, Birmingham, 2015, 418 p.

[37] C. Spurgeon, J. Zimmerman, Ethernet: The Definitive Guide, 2nd ed. O'Reilly Media, 2014, 508 p.

[38] M. Stevanovic, Advanced C and C++ Compiling, 1st ed. Apress, Berkeley, CA, 2014, 326 p.

[39] L. Wang, X. Chen, A Design and Implementation of Bootloader Based on MPC8280, Applied Mechanics and Materials, Vol. 668-669, 2014, pp. 592-597.

[40] Xenomai, Task management services. Saatavissa (viitattu 18.2.2018): http://xenomai.org/documentation/xenomai-2.4/html/api/group__task.html.