Tampereen ammattikorkeakoulu

OpenWrt-reitittimen laiteohjelmiston kääntäminen ja etähallintaohjelmiston integrointi

Teemu Tuovinen

OPINNÄYTETYÖ
Huhtikuu 2024

Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TUOVINEN, TEEMU:
OpenWrt-reitittimen laiteohjelmiston kääntäminen ja etähallintaohjelmiston integrointi

Opinnäytetyö 36 sivua
Huhtikuu 2024

Opinnäytetyön tavoitteena oli kehittää toimeksiantajayrityksenä toimineen Jidoka Technologies Oy:n Smart Hub 4G -etäohjattavan reitittimen uudelle 5G-versiolle laiteohjelmisto. Uudella 5G-reitittimellä pyritään vastaamaan paremmin asiakkaiden tarpeisiin ja tätä kautta säilyttämään kilpailukyky yrityskäyttöön suunnattujen reitittimien markkinoilla.

Työssä hyödynnettiin käyttöjärjestelmän osalta 5G-reitittimen laitevalmistajalta saatua lähdekoodia ja etähallintaohjelmiston osalta Jidokan 4G-reitittimen lähdekoodia. Käyttöjärjestelmälle luotiin käännösympäristö, jolla OpenWrt-laiteohjelmisto voidaan kääntää tarvittavien pakettien ja Linux-ytimen päivitysten kanssa. Etähallintaohjelmisto päivitettiin tukemaan 5G-reititintä sekä uusia käyttötarkoituksia. Lopuksi näistä koostettiin yhtenäinen laiteohjelmisto, joka on asennettavissa reitittimelle.

Tuloksena saatiin käyttökelpoinen 5G-reitittimen laiteohjelmisto halutuilla ominaisuuksilla pitäen samalla taaksepäin yhteensopivuus 4G-reitittimen kanssa. Uusi 5G-reititin lanseerataan markkinoille vuoden 2024 kesään mennessä.

Asiasanat: openwrt, laiteohjelmisto, 5g, etähallinta, go

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

TUOVINEN, TEEMU:
OpenWrt Router Firmware Compilation and Remote Management Software Integration

Bachelor's thesis 36 pages
April 2024

The aim of the thesis was to develop firmware for the new 5G version of the Smart Hub 4G remote control router of Jidoka Technologies Oy which acted as the commissioning company. The new 5G router aims to better meet the needs of the customers and thereby maintain competitiveness in the market of routers aimed at business use.

The thesis used source code from the 5G router device manufacturer for the operating system and Jidoka's 4G router source code for the remote management software. A build environment was created for the operating system with which the OpenWrt firmware could be compiled with the necessary packages and Linux kernel patches. The remote management software was updated to support the 5G router and new use cases. Finally, these were compiled into a unified firmware that could be installed on the router.

The result was a usable 5G router firmware with the desired features while maintaining backward compatibility with the 4G router. The new 5G router will be launched on the market by the summer of 2024.

Key words: openwrt, firmware, 5g, remote management, go

SISÄLLYS

1 JOHDANTO 6
2 TAVOITE JA TARKOITUS 7
3 TEKNOLOGIAT 8
3.1 OpenWrt 8
3.2 Buildroot 9
3.3 Image Builder 9
3.4 U-Boot 10
3.5 Go 10
4 TOTEUTUS 12
4.1 Valittu laite 12
4.2 Laitevalmistajan ohjelmisto 12
4.3 Käynnistyslataajan muutokset 13
4.4 Käyttöjärjestelmän muutokset 14
4.5 Etähallintaohjelmiston muutokset 17
4.5.1 Go kielen version päivitys 17
4.5.2 Signaalitietojen kysely 18
4.5.3 Viestijono 19
4.5.4 MWAN3 20
4.5.5 Lokiviestit 26
4.5.6 Järjestelmäasetusten hallinta 28
4.5.7 Diagnostiikka 29
4.6 Laiteohjelmiston käännösympäristö 31
5 POHDINTA 35
LÄHTEET 36

LYHENTEET JA TERMIT

OpenWrt	Open wireless router
GNU	General public license
UCI	Universal configuration interface
SPL	Secondary program loader
QSDK	Qualcomm Atheros switch software development kit
QMI	Qualcomm mobile station modems interface
USB	Universal serial bus
SSH	Secure shell
Mb/s	Megabittiä sekunnissa
AMQP	Advanced message queuing protocol
AMQPS	AMQP over SSL
WAN	Wide area network
LAN	Local area network
MWAN3	MultiWAN, paketti useiden WAN yhteyksien hallintaan
IPv4	Internet protocol version 4
JSON	JavaScript object notation
DNS	Domain name system
NSA	Non-standalone
SA	Standalone
SHA256	Hajautusalgoritmi
WSL	Windows subsystem for Linux
IPK	Itsy package
Visual Studio Code	Ohjelmointiympäristö

1 JOHDANTO

Alati kasvavien datamäärien edessä tarve nopeammille ja luotettavammille internetyhteyksille on kasvanut. Yritysmaailmassa viimevuosina yleistyneet etätyöt ja videokonferenssit lisäävät entisestään tarvetta nopeammille yhteyksille. Verkkoyhteyksiä tarvitaan paitsi toimistoilla myös esimerkiksi rakennustyömailla tai tien päällä. Nopea verkkoyhteyden pystyttäminen on myös tärkeää ja kiinteiden kaapeliyhteyksien rakentaminen voi pahimmillaan viedä kuukausia.

Näitä tarpeita vastaamaan Jidoka Technologies Oy on jo pitkään kehittänyt yrityskäyttöön suunnattua Smart Hub -etähallittavaa mobiilireititintä. Reititin voidaan kytkeä niin tavalliseen pistokkeeseen kuin auton akkuunkin, jolloin verkkoyhteys seuraa käyttäjäänsä minne tahansa. Reitittimestä on kuitenkin tähän mennessä saatavilla vain 4G-versio, ja yritysasiakkailta on enenevissä määrin tullut kyselyitä 5G-versiosta. Osa yritysasiakkaista jopa eivät enää halua ostaa 4G-reititintä ollenkaan koska teknologia nähdään vanhentuneena.

Tässä opinnäytetyössä pyritään vastaamaan näihin asiakastarpeisiin kehittämällä uudelle 5G-reitittimelle olemassa olevien järjestelmien kanssa yhteensopiva laiteohjelmisto. Näin myös säilytetään Smart Hub -tuotteen kilpailukyky tulevaisuudessa.

2 TAVOITE JA TARKOITUS

Opinnäytetyön tavoite on jaettavissa kahteen osaan. Ensimmäisenä on luotava laitevalmistajalta saadun lähdekoodin pohjalta OpenWrt-käyttöjärjestelmän käännösympäristö. Lisäksi käyttöjärjestelmään tehdään tarvittavat muutokset käytetyn laitteiston tukemiseksi.

Seuraavaksi edellisen sukupolven OpenWrt-pohjaiselle 4G-reitittimelle tehdyn etähallintaohjelmiston koodipohjaan tuodaan tuki uudelle 5G-reitittimelle. Samalla on pidettävä huoli siitä, että yhteensopivuus 4G-reitittimen kanssa pysyy, jotta sama koodi on käytettävissä sekä vanhalla että uudella laitteistolla. Uuteen reitittimeen tuodaan samat ominaisuudet kuin edellisen sukupolven reitittimellä, sekä joitain lisäominaisuuksia uusia käyttötarpeita mukaillen. Lisäksi uudet ominaisuudet pyritään mahdollisuuksien mukaan toteuttamaan siten, että ne ovat päivitettävissä myös edellisen sukupolven laitteille.

Tavoitteena on, että käyttäjän näkökulmasta uuden reitittimen käyttö tapahtuu aivan kuten vanhan reitittimellä. Näin vanha reititin voidaan korvata saumattomasti uudella ilman, että käyttäjä näkee muuta eroa kuin suuremmat tiedonsiirtonopeudet.

3 TEKNOLOGIAT

3.1 OpenWrt

OpenWrt on sulautetuille järjestelmille erityisesti verkkoyhteyksien reititykseen tarkoitettu Linux-jakelu. Se juontaa juurensa 2003 julkaistuun Linux-pohjaiseen Linksys WRT54G reitittimeen. Linux on julkaistu GNU-lisenssin alla, jonka yhtenä vaatimuksena on muokatun ja uudelleen levitettävän ohjelmiston lähdekoodin julkaisu. Linksys oli täten pian laitteen julkaisun jälkeen pakotettu julkaisemaan muokkaamansa Linux lähdekoodin. Vuonna 2007 julkaistu ensimmäinen OpenWrt-versio koodinimeltään White Russian perustui WRT54G:n lähdekoodiin. (Eric Sandler 2005; OpenWrt 2023c.)

Nykyisin OpenWrt käyttää virallista GNU/Linux-lähdekoodia, johon kääntövaiheessa lisätään päivitystiedostoilla tarvittavat yleiset sekä laitekohtaiset muutokset. OpenWrt:n suurin hyöty tulee sen aktiivisesta kehittäjäyhteisöstä ja paketinhallintajärjestelmästä. Laajan pakettivalikoiman ansiosta OpenWrt reitittimet ovat muokattavissa monenlaisiin tarkoituksiin kuten useamman internetyhteyden kuorman tasapainotukseen tai tiedostonjakopalvelimeksi. OpenWrt tukee satoja kaupallisia reitittimiä, joiden valmistajan laiteohjelmisto voidaan korvata OpenWrt:llä. (OpenWrt 2023a.)

OpenWrt käyttää järjestelmän asetustiedostoille yhtenäistä syntaksia, jota myös moni yhteisön kehittämä paketti myös tukee. Näiden asetusten hallintaan käytetään UCI-ohjelmistoa. Tämän tarkoituksena on keskittää järjestelmän asetukset samaan paikkaan sekä yhtenäistää eri asetustiedostojen formaatti asetusten hallinnan helpottamiseksi. (OpenWrt 2023d.)

Asetusten hallintaan on myös olemassa web-käyttöliittymä, joka tarjoaa komentoriviohjelmistoon verrattuna edistyneempiä ominaisuuksia kuten asetusten automaattisen palautuksen, jos laitteeseen ei saada enää yhteyttä asetusten muuttamisen jälkeen. Käyttöliittymään voi luoda myös tuen omille UCI-formaattia käyttäville konfiguraatioille. (OpenWrt 2023b.)

3.2 Buildroot

Buildroot on sulautetuille järjestelmille tarkoitettu Linuxin käännösympäristö. Sulautetuissa järjestelmissä ei usein ole tarpeeksi resursseja ohjelmistojen käännösten tekemiseen tai käännösten tekemiseen kuluu tarpeettoman paljon aikaa. Tämän vuoksi on tärkeää pystyä tekemään käännökset tehokkaalla tietokoneella, jolta käännetyt binäärit voidaan siirtää sulautetulle järjestelmälle. Sulautetuissa järjestelmissä on kuitenkin usein käytössä erilainen arkkitehtuuri kuin tavallisissa tietokoneissa ja käännökset lähdekoodista konekieleen tehdään arkkitehtuurikohtaisesti. Buildroot mahdollistaa käännökseen tarvittavien työkalujen ristiinkäännöksen kohdelaitteen arkkitehtuurille. Näitä työkaluja käyttäen voidaan kääntää sulautetulle järjestelmälle tehty lähdekoodi konekieleen siten, että käännöksen tuloksena saatu binääri voidaan ajaa kyseistä arkkitehtuuria käyttävällä laitteella. Buildrootilla voi myös luoda valmiin levykuvan sellaisenaan laitteen tallennustilaan kirjoitettavaksi. Levykuvaan voi sisällyttää kaiken alkulatausohjelmasta Linux ytimeen ja ohjelmistopaketteihin tarpeen mukaan. (Buildroot 2023.)

Buildrootilla on myös mahdollista luoda kohdelaitteelle oma Image Builder, jolla säästäisi aikaa, kun laiteohjelmiston käännöstä lähdetään tekemään puhtaalta pöydältä. Tätä ei kuitenkaan vielä ole nähty tarpeelliseksi sillä Buildroot osaa kääntää vain ne osat laiteohjelmistosta, joihin on tehty muutoksia nopeuttaen käännösprosessia huomattavasti ensimmäisen käännöksen jälkeen.

3.3 Image Builder

Kevyempi vaihtoehto OpenWrt-laiteohjelmiston kääntämiseen on Image Builder, jolla voi koota valmiista osista haluamansa ominaisuudet laiteohjelmistoon. Image Builder käyttää oikealle arkkitehtuurille valmiiksi käännettyä Linux-ydintä ja paketteja. Image Builderia käytettäessä ei siis tarvitse aluksi kääntää kyseiselle arkkitehtuurille käännösympäristöä, minkä ansiosta se on sekä helppokäyttöisempi, että nopeampi. Tämä on hyvä vaihtoehto, jos lähdekoodiin ei tarvitse tehdä muutoksia tai Linux-ytimen asetuksia ei tarvitse muuttaa. Jos

laiteohjelmistoon on tehtävä muutoksia, joihin Image Builder ei kykene, on käytettävä Buildroottia. (OpenWrt 2023e.)

3.4 U-Boot

U-Boot on suosittu sulautetuilla järjestelmillä käytettävä käynnistyslataaja. Alun perin PowerPC-arkkitehtuurille PPCBoot nimellä kehitetty ja vuonna 2000 julkaistu käynnistyslataaja sai vuonna 2002 tuen myös työssä käytetylle ARM-arkkitehtuurille. Samalla se sai nykyisen U-Boot nimensä, joka on lyhennys sanoista Universal Boot Loader. Tätä seuraavien vuosien aikana U-Boot sai tuen lukuisalle muullekin arkkitehtuurille. (DENX Software Engineering 2021.)

U-Boot on toimintaperiaatteeltaan yksinkertainen. Sen tarkoituksena on alustaa laitteisto lopullista käyttöjärjestelmää varten, ladata käyttöjärjestelmä laitteen muistiin ja käynnistää se halutuilla argumenteilla. U-Boot on itsessään jaettavissa kahteen osaan: SPL ja täysi U-Boot. Laitteen käynnistyminen alkaa SPL:llä, joka alustaa laitteen muistin ja sarjaportin. Tämän jälkeen SPL lataa U-Bootin muistiin ja käynnistää sen. Vasta tämän jälkeen alustetaan kaikki käyttöjärjestelmän käynnistämiseen tarvittava. (U-Boot GitHub 2023.)

3.5 Go

Monista OpenWrt paketeista poiketen etähallintaohjelmisto on tehty Go-kielellä C-kielen sijaan. Go, josta välillä käytetään myös nimitystä Golang, on Googlen kehittämä syntaksiltaan C-kieltä muistuttava ohjelmointikieli. Se on alun perin kehitetty Googlen omien massiivisten palvelinprojektien tarpeisiin mutta soveltuu myös sulautettujen järjestelmien ohjelmointiin. (Rob Pike 2012.)

Etuina C-kieleen verrattuna on helppolukuisempi syntaksi, muistinhallinta roskienkerääjällä, riippuvuuksien hallinta ja helpompi koodin ajo samanaikaisesti. Näiden ansiosta koodin kirjoittaminen Go-kielellä on nopeampaa. (Rob Pike 2012.) Lisäksi etuna sulautettujen järjestelmien kanssa on koodin kääntäminen helposti eri arkkitehtuureille riippumatta isäntälaitteen arkkitehtuurista.

Käännöskomennolle voidaan yksinkertaisesti antaa parametrina kohdelaitteen arkkitehtuuri ja käännetty binääri on ajettavissa kyseisellä laitteella.

Haittapuolena on binäärin huomattavasti suurempi koko C-kieleen verrattuna. Yksinkertainen “hello world” binääri C-kielestä käännettynä on joitain kilotavuja siinä missä vastaava Go-kielellä tehtynä on noin kaksi megatavua. Tämä voi sulautettujen järjestelmien tapauksessa koitua ongelmaksi vähäisen tallennustilan vuoksi.

4 TOTEUTUS

4.1 Valittu laite

Työssä käytettäväksi reitittimeksi valikoitui IPQ6018-piirisarjaan perustuva valmis piirilevy Quectelin 5G-modeemilla varustettuna. Valmistajalta oli saatavilla vain 2015 julkaistuun OpenWrt 15 -versioon perustuva toimiva laiteohjelmisto mutta heillä oli myös OpenWrt 21 -pohjainen laiteohjelmiston lähdekoodi, jonka saimme projektille pohjaksi. Vaikka versio 21 onkin vuodelta 2021 ja täten viimeisintä OpenWrt-julkaisua jäljessä todettiin se tarpeeksi uudeksi kehitystyön aloittamiseen. Virallisen OpenWrt:n parissa on myös tehty työtä IPQ6018-piirisarjan tukemiseksi eli tulevaisuudessa todennäköisesti käyttöjärjestelmä on päivitettävissä uusimpaan versioon.

4.2 Laitevalmistajan ohjelmisto

Aluksi tutkittiin olisiko mahdollista, että valmistajan tekemien muutoksien perusteella voitaisiin tehdä tarvittavat muutokset uusimpaan OpenWrt-versioon. Nopeasti kuitenkin selvisi, että nämä muutokset eivät olisi läheskään tarpeeksi vaan projektiin pakattuna tiedostona sisällytettyyn QSDK Linux-ytimeen tehdyt muutokset olisi myös tuotava OpenWrt:n käyttämään Linux-ytimeen. Nämä muutokset QSDK:n kehittäjä Qualcomm oli tehnyt suoraan lähdekoodiin päivitystiedostojen sijaan, mikä entisestään vaikeuttaisi muutosten löytämistä.

Toinen vaihtoehto oli käyttää valmista QSDK-pohjaista OpenWrt 21 -projektia. Tämän haittapuolena olisi kaksi vuotta vanha OpenWrt-versio sekä kahdeksan vuotta vanhaan Linux-ytimeen perustuva QSDK. Varsinkin vanhan ytimen pelättiin aiheuttavan ongelmia tulevaisuudessa mutta lähdekoodia tarkastelemalla havaittiin, että Qualcomm on tuonut ytimeen ainakin osittain uudempiin ytimen versioihin tehtyjä tietoturvapäivityksiä.

Aikarajoitteiden vuoksi päädyttiin jälkimmäiseen vaihtoehtoon. Lisäksi OpenWrt:lle on tulossa tuki IPQ6018-piirisarjalle vuoden 2024 aikana

julkaistavaan versioon 24, jolloin tehdyt muutokset ja ohjelmistot voitaisiin siirtää toimimaan sillä.

4.3 Käynnistyslataajan muutokset

Tietoturvan parantamiseksi U-Boottiin haluttiin tehdä muutoksia. Tätä varten saimme valmistajalta täyden lähdekoodin, jota pystyimme muokkaamaan tarpeidemme mukaan ja lähettämään takaisin valmistajalle käännettäväksi ja testattavaksi. Käännöksen ja testaamisen hoiti valmistaja sen vuoksi, että toimimattoman U-Bootin voi korvata vain oikeilla työkaluilla, joilla saadaan kirjoitettua suoraan laitteen flash-muistiin. Nämä laitteet löytyvät vain valmistajalta, joten valmistaja lähetti meille käännettyjä U-Boot versioita vain silloin kun olivat testanneet ne itse toimiviksi.

U-Boottiin lisättiin salasanasuojaus, jolle käytetyssä U-Boot versiossa oli jo tuki olemassa mutta uudemmassa versiossa korjattu SHA256-tiivisteen parsintaan liittyvä ohjelmointivirhe esti pääsyn U-Bootin komentoriville täysin. Tätä varten etsittiin ja tuotiin korjaus uudemmasta versiosta käytössä olevaan vanhaan versioon, minkä jälkeen salasanasuojaus alkoi toimia.

U-Bootissa on bootargs-ympäristömuuttujaan tallennettu lista komennoista, jotka annetaan eteenpäin Linux ytimelle käynnistyksen yhteydessä. Muokkaamalla näitä komentoja on mahdollista avata Linux-komentorivi pääkäyttäjän oikeuksilla. Muuttujan perään voidaan lisätä kutsu avaamaan Linux-komentorivi, jolloin komentorivi aukeaa käytettäväksi sarjaportin kautta. Tämän laitteen Linux-ytimelle annetaan komennoiksi ”console=ttyMSM0,115200n8 ubi.mtd=rootfs root=mtd:ubi_rootfs rootfstype=squashfs rootwait cnss2.bdf_pci0=0xa0”, missä tärkein komento on rootwait, jolla odotetaan tiedostojärjestelmän alustamista. Lisäämällä komento /bin/sh rootwait-komennon jälkeen voidaan OpenWrt-komentorivi käynnistää täysillä pääkäyttäjän oikeuksilla tiedostojärjestelmään. Tämän estämiseksi kyseinen ympäristömuuttuja tarkistetaan muutosten varalta ennen Linux-ytimen käynnistystä ja käynnistys keskeytetään muutoksia havaittaessa.

4.4 Käyttöjärjestelmän muutokset

Lähdekoodi ei sellaisenaan ollut vielä valmis käytettäväksi eikä sitä edes pystynyt uusimmilla kääntäjillä kääntämään. Lähdekoodin dokumentaatio oli hyvin puutteellinen ja alkuun piti käyttää aikaa paljon pelkästään käännösympäristön rakentamiseen. Lisäksi Linux-ytimenä käytettiin Qualcommin Linux 4.4 versioon perustuvaa QSDK-ydintä virallisen OpenWrt 21 käyttäessä versiota 5.4. Puutteita koodissa oli 5G-modeemin ajurituessa.

Näin vanhassa Linux-ytimessä ei ollut tukea Quectelin modeemien käyttämään QMI-protokollaan USB-yhteydellä. QMI itsessään oli tuettuna mutta vain ethernetin yli. Tätä varten onneksi löytyi valmiit päivitystiedostot käytetylle Linux-ytimen versiolle, joiden avulla qmi_wwan-ajuriin saatiin tuotua tuki myös USB-yhteyksille. Lisäksi qmi_wwan-ajuriin täytyy erikseen lisätä jokainen tuettu modeemi käyttäen valmistajan ja tuotteen tunnistetta modeemin yksilöimiseksi. QMI:tä käytettäessä USB:n yli on myös lisättävä samat tunnisteet USB-ajuriin. Tätä varten tehtävät muutokset voitiin tehdä uudemman Linux-ytimen pohjalta, jossa käytetty RM500Q-modeemi oli jo tuettuna.

SSH-yhteyden suojaamiseksi sisäänkirjautuminen käyttäjänimen ja salasanan avulla poistettiin käytöstä. Näin sisäänkirjautuminen on mahdollista vain SSH-avaimen avulla. Edellisen sukupolven laitteeseen oli jo lisätty mahdollisuus avaimen käyttöön, joten avain oli jo valmiiksi mukana etäkäyttöohjelmiston paketissa. Asetusten muutokset lisättiin omana shell-skriptinään (kuva 1) IPK-paketin sisäiseen kansioon, joka sisältää asennuksen yhteydessä ajettavia shell-skriptejä.

```
if ! uci -q get dropbear.@dropbear[0].PasswordAuth | grep -iq '0' ; then

        echo "Disable password auth"

        uci -q batch <<-EOF >/dev/null
                set dropbear.@dropbear[0].PasswordAuth="0"
                commit dropbear
EOF
fi

exit 0
```

KUVA 1. SSH-palvelimen asetusten muutokset.

Laitteen suorituskykyä testatessa huomattiin, että vaikka laitteissa käytettiin moniydinprosessoreja, ei pakettien reititykseen käytetty kuin yhtä ydintä. Tämän vuoksi tiedonsiirtonopeus rajoittui noin 300:aan Mb/s. Linux-ytimen verkkoajuri käyttää tällä laitteella useampaa jonoa syöttämään ja vastaanottamaan paketteja verkkokortin ja Linux-ytimen välillä. Koska kaikki nämä jonot käyttivät yhtä ja samaa prosessoriydintä muodostui pullonkaulaksi yksittäisen ytimen suorituskyky, joka ei ollut tarpeeksi 5G-yhteyden pakettimäärän prosessointiin. Sallimalla jonojen käyttää kaikkia prosessoriytimiä päästiin tästä pullonkaulasta eroon. (Dan Siemon n.d.)

Jonojen tiedot löytyvät Linuxin sysfs-järjestelmästä kansiosta /sys/class/net/*interface*/queues, missä *interface* on fyysisen verkkosovittimen nimi. Näiden kansioiden sisältä löytyy kansiot lähetys- ja vastaanottojonoja varten. Jonokansion sisällä on jonon tyypin mukaan joko rps_cpus tai xps_cpus tiedosto, joka määrittää bittimaskina mitkä prosessoriytimet ovat jonon käytettävissä. Neliydinprosessorilla kun halutaan kaikki ytimet käyttöön, tulee bittimaskiksi 1111, joka on heksadesimaalina f. Näiden muutosten jälkeen suurimmat nopeudet testeissä olivat 700 Mb/s luokkaa.

Verkon käytön ollessa satunnaista huomattiin, että yhteyksien alussa ylimääräistä viivettä ennen kuin yhteys alkoi toimia täydellä nopeudella. Tämän korjaamiseksi asetettiin prosessori Linuxin-ajureista suorituskyky tilaan. Tässä tilassa prosessorin kellotaajuus pidetään jatkuvasti suurimmassa sallitussa arvossa. Oletuksena kellotaajuus säädetään kuormituksen mukaan. Kellotaajuuden nostamisen havaittiin kuitenkin tapahtuvan liian hitaasti mikä

aiheutti yhteyksien alkuun viivettä. Lisäksi minimi kellotaajuus asetettiin samaksi kuin maksimi siltä varalta, että joissain tilanteissa käyttöjärjestelmä ottaa suorituskykytilan pois päältä.

Lisäämällä edellä mainitut muutokset /etc/rc.local tiedostoon saadaan asetukset otettua käyttöön aina järjestelmän käynnistyksen yhteydessä. Muutosten teko lisättiin omana shell-skriptinään (kuva 2) IPK-paketin sisäiseen kansioon, joka sisältää asennuksen yhteydessä ajettavia shell-skriptejä.

```
if grep 'find /sys/class/net' -iq /etc/rc.local ; then
 echo "/etc/rc.local already added"
else
echo "" >  /etc/rc.local
tee -a /etc/rc.local << END
find /sys/class/net/*/queues/[rt]x-[01234567]/[rx]ps_cpus -exec sh -c '[ -w {} ] && echo f > {} 2>/dev/null' \;

echo "performance" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo "performance" > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
echo "performance" > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
echo "performance" > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
echo "710000" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
echo "710000" > /sys/devices/system/cpu/cpu1/cpufreq/scaling_min_freq
echo "710000" > /sys/devices/system/cpu/cpu2/cpufreq/scaling_min_freq
echo "710000" > /sys/devices/system/cpu/cpu3/cpufreq/scaling_min_freq

exit 0
END
fi
```

KUVA 2. Prosessorin suorituskykyasetusten muutokset.

Oletuksena OpenWrt sallii sarjaportin kautta sisäänkirjautumisen ja järjestelmän lokien lukemisen. Vaikka nämä ovat kehityskäytössä hyödyllisiä ominaisuuksia haluttiin sarjaportti poistaa käytöstä lopullisesta tuotteesta tietoturvasyistä.

Lokiviestien poistamiseksi muutettiin Linux-ytimen käyttämän printk-lokifunktion asetuksista käytetyksi lokiviestien tasoksi 0, joka vastaa tasoa KERN_EMERG. Tällä tasolla tulostetaan vain kaikkein kriittisimmät lokiviestit, jolloin viestejä laitteen tavallisesta toiminnasta ei tulosteta ollenkaan.

Sisäänkirjautuminen sarjaportin kautta poistettiin käytöstä poistamalla järjestelmän käynnistyessä avattavia prosesseja hallitsevasta /etc/inittab tiedostosta rivit, joilla käynnistetään komentorivi sarjaportille. Varmuuden vuoksi sarjaportin kautta sisäänkirjautumiseen asetettiin salasanan kysely päälle, vaikka tähän pisteeseen ei sarjaportin kautta pitäisi ikinä päästä.

4.5 Etähallintaohjelmiston muutokset

4.5.1 Go kielen version päivitys

Reitittimelle asennettava etähallintaohjelmisto päätettiin päivittää uusimpaan Go-versioon, jolloin siirryttiin versiosta 1.13 versioon 1.21. Ohjelmisto käytti vielä edellisen sukupolven laitteen kehitystyön aikaista versiota kielestä ja sen jälkeen esimerkiksi kielen standardikirjastoihin oli tullut useita ohjelmointia helpottavia lisäyksiä. Moni kolmannen osapuolen kirjasto myös käyttää näitä uusia ominaisuuksia, minkä takia monista kirjastoista ei ole olemassakaan versioita vanhalle kielen versiolle tai pitäisi käyttää vanhempaa kirjaston versiota, jota ei myöskään tulevaisuudessa voi päivittää. Tulevaisuuden kannalta todettiin parhaaksi vaihtoehdoksi päivittää uusimpaan Go-versioon nyt sen sijaan, että päivitystä viivytettäisiin niin pitkään kuin mahdollista.

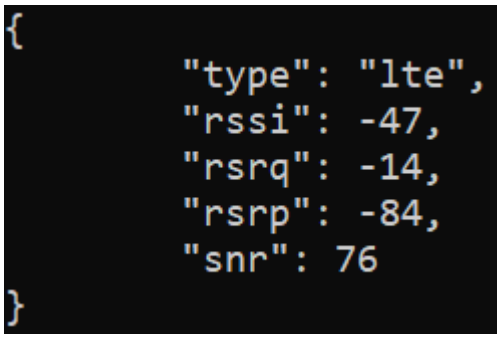
Samalla täytyi ottaa käyttöön uusi Go-kielen paketinhallintajärjestelmä Go modules, jolla korvattiin aiemmin projektissa käytetty Go dep. Alkuperäistä koodia kirjoitettaessa Go-kielelle ei ollut vakiintunutta paketinhallintajärjestelmää. Go dep oli virallinen mutta vielä kokeellinen paketinhallintajärjestelmä. Samalla kehitettiin myös Go modules -paketinhallintajärjestelmää ja sille oli olemassa alustava tuki versiosta 1.11 alkaen. Go dep oli kuitenkin ollut pidempään käytössä, minkä vuoksi se päätettiin aiemmin tässä projektissa ottaa käyttöön. Vuosi projektin alun jälkeen Go modulesista tehtiin Go:n virallinen paketinhallintajärjestelmä ja Go dep kehitys lopetettiin. Teoriassa siirtyminen Go modulesiin on helppoa, sillä komento, jolla projekti alustetaan käyttämään Go modulesia osaa lukea lukuisten aiempien paketinhallintajärjestelmien hallintatiedostojen, mukaan lukien Go dep, formaatteja ja automaattisesti uudelleen luoda ne Go modulesin formaattiin.

Käytännössä kuitenkin ongelmia aiheutui, sillä joissain paketeissa oli tehty taaksepäin yhteensopimattomia rajapintamuutoksia eikä vanhempia versioita ollut enää saatavilla. Tämä pakotti tekemään muutoksia koodiin näiden pakettien osalta. Linux-ytimen netlink-moduulin kanssa keskusteleva Go-paketti oli

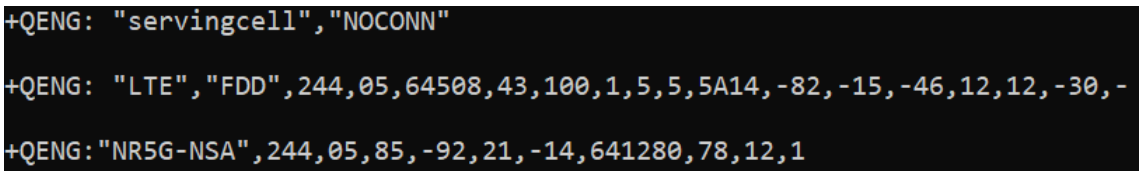
päivittynyt rajapintansa osalta erilaiseksi kuin Go 1.13 kanssa käytetty paketti, joten netlinkkiä käyttävä osa koodista oli osittain kirjoitettava uudelleen. Netlinkkiä käytetään dataliikenteen seurantaan kysymällä Linux-ytimeltä tasaisin väliajoin tietoja reititetyistä paketeista.

4.5.2 Signaalitietojen kysely

Aiemmin signaalitietojen kyselyyn käytetty uqmi-ohjelmisto pystyi palauttamaan verkon tyypin ja signaalin voimakkuuden kaikille eri verkkotyypeille. Uusin versio ei kuitenkaan vielä tue 5G NSA -verkkoja vaan antaa verkon tyypiksi LTE. Tämän vuoksi verkon signaalitietojen kyselyssä täytyi siirtyä käyttämään sarjayhteyden kautta modeemille lähetettäviä AT-komentoja. Haittapuolena tässä on se, että uqmi palauttaa tiedot modeemista riippumatta samalla tavalla (kuva 3) mutta AT-komennoista osa on valmistajakohtaisia ja juuri tarkempien signaalitietojen hakeminen on käytetyllä modeemilla tehtävä Quectelin oman AT-komennon kautta (kuva 4).



KUVA 3. Uqmi-komennon palauttamat signaalitietodot 5G NSA -verkossa.



KUVA 4. AT komennon palauttamat signaalitietodot 5G NSA -verkossa.

4.5.3 Viestijono

Palvelimen kanssa viestintään käytetään RabbitMQ-viestijonoja, jotka hyödyntävät AMQP-protokollaa. RabbitMQ tukee myös TLS-salausta viestien sisällön suojaamiseksi käyttävää AMQPS-protokollaa mutta edellisen sukupolven laite ei tätä käyttänyt. Tietoturvan parantamiseksi viestijono muutettiin käyttämään AMQPS-protokollaa. Viestijonon asetuksiin lisättiin tieto onko TLS tuettuna vai ei (kuva 5).

```
// Queue contains data needed for the hub to connecto to RabbitMQ
type queue struct {
    Host     string `json:"host,omitempty"`
    Port     int    `json:"port,string,omitempty"`
    Vhost    string `json:"vhost,omitempty"`
    User     string `json:"user,omitempty"`
    Password string `json:"password,omitempty"`
    Exchange string `json:"exchange,omitempty"`
    TLS      bool   `json:"tls,omitempty"`
}
```

KUVA 5. Viestijonon asetukset.

Viestijonon asetukset saadaan laitteelle palvelimelta ensimmäisen käynnistyksen yhteydessä. Laite lähettää provisiointijärjestelmään viestin, jonka tiedoilla laite voidaan yksilöidä. Vastauksena tähän viestiin laite saa omat yksilölliset asetuksensa. Viestiin lisättiin jonon asetuksista saatava tieto siitä, tukeeko laite AMQPS-protokollaa ja tämän perusteella provisiointijärjestelmä palauttaa laitteelle sopivat viestijonoasetukset. Ensimmäisellä käynnistyksellä käytetään laiteohjelmiston mukana tulevaa viestijonon asetuksia. Tämän jonon kautta laite saa haettua itselleen lopullisen viestijonon asetukset. AMQPS:än käyttämiseksi täytyy kirjasto ohjeistaa käyttämään AMQPS-protokollaa ja käytettävä portti on muutettava. RabbitMQ-kirjasto voidaan ohjeistaa käyttämään AMQPS-protokollaa muuttamalla palvelimen URL-skeema AMQP:sta AMQPS:ään. Oletuksena AMQP käyttää porttia 5672 ja AMQPS porttia 5671 ja näitä portteja laitteen käyttämä viestijono palvelin myös käyttää.

4.5.4 MWAN3

5G-reitittimelle uutena käyttökohteena laite voidaan toimittaa kiinteän yhteyden rakentamisen ajaksi väliaikaiseksi yhteydeksi. Kiinteän yhteyden valmistumisen jälkeen laite jätetään varayhteydeksi. Tätä varten otettiin käyttöön MWAN3-paketti, jolla voi yhdistellä useista internetyhteyksistä erilaisia kokonaisuuksia. Oletuksena laite on asetettu varayhteystilaan, jossa pääyhteytenä toimii ethernet WAN ja toissijaisena yhteytenä 5G-modeemi. Ethernet-yhteyden katketessa vaihdetaan 5G-yhteys käyttöön muutaman sekunnin sisällä. Ethernet-yhteyden palautuessa otetaan se takaisin käyttöön ja 5G-yhteys jää varalle.
Toinen MWAN3-paketin käyttökohde on kuormituksen tasaaminen usean yhteyden välillä. Tämän avulla voidaan yhdistää useamman internetyhteyttä tarjoavan laitteen verkkoyhteys yhdeksi, siten että yksittäiset yhteydet sisäverkon ulkopuolelle kulkevat satunnaisen verkkolaitteen kautta. Kaikki laitteet ovat silti samassa sisäverkossa kiinni ja voivat nähdä toisensa, jolloin esimerkiksi tulostimet ovat käytettävissä miltä tahansa verkkoon kytketyltä laitteelta. Vaikka yksittäisen yhteyden nopeutta ei tällä tavalla voida nostaa saadaan kuitenkin kokonaisen toimiston, jossa voi olla kymmeniä tai satoja internetyhteyttä tarvitsevia laitteita, käytettävissä olevaa kaistaa kasvatettua. Samalla myös lisätään toimintavarmuutta, sillä yhden internetyhteyttä tarjoavan laitteen yhteyden katketessa verkkoyhteys toimii edelleen muiden laitteiden läpi. Aiemmin tähän tarkoitukseen on käytetty erillisiä laitteita, joissa ei ole ollut mahdollisuutta etähallintaan. MWAN3-pakettia käyttämällä saatiin yhdellä laitteella katettua myös tämä tarve ja samalla kuormituksen tasaajasta tehtiin myös etähallittava.

MWAN3-paketin toimintaa hallitaan UCI-formaattisilla asetustiedostoilla. Laiteohjelmistoon on sisällytetty viisi valmista asetustiedostoa, joilla voidaan valita varayhteyden, kuormantasauksen ja vain modeemin tai ethernetin käytön väliltä. Asetuksissa voidaan määrittää kuinka eri verkkosovittimet toimivat yhdessä, asettamalla niille metric ja weight -arvot. Metric määrittää missä järjestyksessä verkkosovittimia käytetään siten, että pienimmällä arvolla on suurin prioriteetti ja samaan arvoon asetettujen verkkosovittimien välillä tehdään kuormantasausta. Kuormantasauksen ollessa käytössä weight-arvo määrittää missä suhteessa verkkoliikennettä ohjataan verkkosovittimien välillä.

Verkkosovitinasetuksista muodostetaan erilaisia menettelytapoja, joita sitten otetaan käyttöön halutulle verkkoliikenteelle. Oletusasetuksissa (kuva 6) käytetään ethernettiä pääyhteytenä ja modeemia varayhteytenä kaikelle IPv4-liikenteelle.

```
config member 'wan_m1_w3'
        option interface 'wan'
        option metric '1'
        option weight '3'

config member 'modem_m2_w2'
        option interface 'modem'
        option metric '2'
        option weight '2'

config policy 'wan_modem'
        list use_member 'wan_m1_w3'
        list use_member 'modem_m2_w2'

config rule 'default_rule_v4'
        option dest_ip '0.0.0.0/0'
        option family 'ipv4'
        option use_policy 'wan_modem'
```

KUVA 6. MWAN3-paketin varayhteysasetukset.

Etähallintaohjelmiston asetuksiin on lisätty MWAN3-asetukset (kuva 7), joilla valitaan mikä edellä mainituista asetustiedostoista otetaan käyttöön sekä minkä nimisiä loogisia WAN-verkkosovittimia laitteesta löytyy. MWAN3-asetustiedosto otetaan käyttöön (kuva 8), jos asetuksiin on havaittu tehdyn muutoksia. Modeemille voidaan asettaa vain yksi verkkosovitin, sillä laitteeseen voi kytkeä vain yhden modeemin. Ethernet-verkkosovittimia voi asettaa useampia. Vaikka oletuksena WAN ethernet -portteja on vain yksi, on LAN ethernet -portteja on mahdollista asettaa toimimaan WAN-portteina tarvittaessa.

```go
// MultiWAN data for mwan3 OpenWRT package
type multiwan struct {
    Setup              string   `json:"setup,omitempty"`
    ModemInterface     string   `json:"modem_interface,omitempty"`
    EthernetInterfaces []string `json:"ethernet_interfaces,omitempty"`
}
```

KUVA 7. Etähallintaohjelmiston asetukset MWAN3-paketin hallintaan.

```go
func SetMultiWAN(configName string) error {
    if len(configName) == 0 {
        return errors.New("multiwan settings not given in config")
    }

    configPath := filepath.Join("/etc/smarthub/multiwan", configName)
    if _, err := os.Stat(configPath); err != nil {
        return fmt.Errorf("multiwan template %s not found", configPath)
    }

    _, errStream, err := tr4util.Command(`cat ` + configPath + ` | uci import mwan3`)
    if len(errStream) > 0 {
        err = errors.New(string(errStream))
    }
    return err
}
```

KUVA 8. MWAN3-paketin asetustiedoston käyttöönotto.

Edellisen sukupolven laitteessa vain modeemin kautta tuleva internetyhteys oli tuettuna ja yhteyttä testattiin ja tarvittaessa palautettiin, jos yhteys RabbitMQ-jonoon menetettiin. Yhteyden palauttamisen logiikkaa täytyi muokata, jotta myös ethernet-yhteydet voidaan katketessaan palauttaa ja usean internetyhteyden ollessa käytössä seurata jokaista yhteyttä erikseen.

Joitain yhteyksien tarkkailuun käytettäviä komentoja varten täytyy ensin selvittää loogista verkkosovitinta vastaava fyysinen verkkosovitin (kuva 9). Esimerkiksi looginen ethernet WAN -verkkosovitin voi olla nimeltään wan ja käyttää fyysistä verkkosovitinta eth0. Fyysisen verkkosovittimen selvittämiseen käytetään ubus-komentoa, jolla voidaan palauttaa kaikkien verkkosovittimien tiedot JSON-muodossa.

```go
func getPhysicalNetworkInterface(virtualInterface string) (string, error) {
    outStream, errStream, err := tr4util.Command("ubus call network.interface dump")
    if len(errStream) > 0 {
        return "", errors.New(string(errStream))
    } else if err != nil {
        return "", err
    }

    // Ubus returns all network interfaces in JSON format
    var interfaces map[string][]networkInterface
    if err := json.Unmarshal(outStream, &interfaces); err != nil {
        return "", err
    }

    // Find the L3 device of the interface
    for _, iface := range interfaces["interface"] {
        if iface.Name == virtualInterface {
            // Only interfaces that are up have L3 device listed
            if iface.Up {
                if iface.L3Device == "" {
                    return "", fmt.Errorf("no l3 device found for interface '%s'", iface.Name)
                }
                return iface.L3Device, nil
            } else {
                return "", errInterfaceDown
            }
        }
    }

    return "", fmt.Errorf("interface '%s' not found", virtualInterface)
}
```

KUVA 9. Loogista verkkosovitinta vastaavan fyysisen verkkosovittimen nimen haku.

Ensimmäisenä täytyy selvittää mitkä asetuksissa mainituista verkkosovittimista ovat oikeasti käytössä. Ethernet-porttien ollessa kyseessä Linuxin sysfs-tiedostojärjestelmästä voidaan lukea, onko porttiin kytketty kaapelia kiinni (kuva 10). Modeemille tällaista tapaa selvittää onko yhteyttä järkevää tarkkailla ei ole löytynyt, joten modeemin oletetaan aina olevan käytettävissä.

```go
func CheckCarrier(interf string) (bool, error) {
    interf, err := getPhysicalNetworkInterface(interf)
    if err != nil {
        // Trying to read carrier with 'cat' while interface is down returns 'invalid argument'.
        // Return that there is no carrier.
        if err == errInterfaceDown {
            return false, nil
        }
        return false, err
    }

    Command := fmt.Sprintf("cat /sys/class/net/%s/carrier", interf)
    cmd := exec.Command("/bin/sh", "-c", Command)
    output, _ := cmd.Output()
    if strings.TrimRight(string(output), "\n") == "1" {
        zap.S().Debug("Found carrier / cable connected: ", interf)
        return true, nil
    } else {
        zap.S().Info("Not found carrier / cable not connected: ", interf)
        return false, nil
    }
}
```

KUVA 10. Ethernet-verkkosovittimen tilan selvitys.

Yhteyksiä tarkkaillaan pingaamalla Googlen DNS-palvelimia tasaisin väliajoin. Tätä varten täytyi luopua aiemmin käytetystä go-ping-kirjastosta ja siirtyä käyttämään Linuxin ping-komentoa, sillä go-ping ei tue tietyn verkkosovittimen valintaa. Ping-komennolle voidaan antaa fyysinen verkkosovitin –I optiolla. Komennon paluuarvon lukemalla saadaan tieto siitä, saatiinko kohdepalvelimelta vastausta vai ei. Paluuarvo 0 tarkoittaa, että vastauksia on saatu vähintään yksi. 1 tarkoittaa, että vastausta ei saatu ollenkaan (kuva 11). Oletuksena on, että Googlen DNS-palvelin 8.8.8.8 vastaa tarpeeksi luotettavasti, jotta yhteys voidaan todeta katkenneeksi, jos vastausta ei saada.

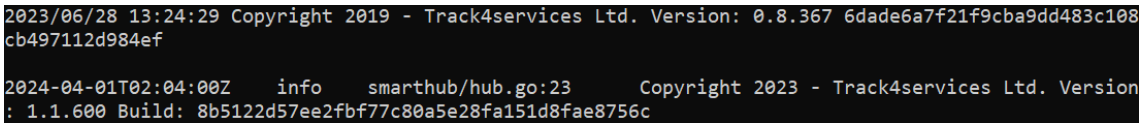
```go
func ConnectionTestThroughInterface(interf string) (bool, error) {
    interf, err := getPhysicalNetworkInterface(interf)
    if err != nil {
        if err == errInterfaceDown {
            return false, nil
        }
        return false, err
    }

    params := " "

    if len(interf) > 0 {
        params = fmt.Sprintf(" -I %s ", interf)
    }

    command := fmt.Sprintf("ping %s -4 -c 4 -i 2 -W 1 8.8.8.8", params)
    cmd := exec.Command("/bin/sh", "-c", command)
    output, err := cmd.CombinedOutput()
    if err != nil {
        if exitErr, ok := err.(*exec.ExitError); ok {
            // Ping command returned error
            zap.S().Debugf("Ping %s exit code: %d", interf, exitErr.ExitCode())
            if exitErr.ExitCode() == 1 {
                // Exit code 1 -> ping received no response
                // Partial packet loss will return exit code 0
                zap.S().Info("Ping failed ", interf)
                return false, nil
            }
            return false, err
        } else {
            // Command execution returned error
            return false, err
        }
    }
    zap.S().Debugf("Ping %s output: %s", interf, string(output))
    zap.S().Info("Ping successful ", interf)
    return true, nil
```

KUVA 11. Internetyhteyden testaus tietyn verkkosovittimen kautta.

Yhteyden palauttamiseksi yritetään ensimmäisenä käynnistää verkkosovitin uudelleen. Verkkosovitin käynnistetään uudelleen ifup-komennolla, jolle parametrina annetaan loogisen verkkosovittimen nimi. Tämä tehdään sekä modeemin, että ethernetin tapauksessa. Modeemin ollessa kyseessä seuraavaksi käynnistetään modeemi uudelleen. Jos tämänkään jälkeen ei yhteyttä saada palautettua eikä yhteyttä ole olemassa minkään muunkaan verkkosovittimen kautta käynnistetään koko laite uudelleen.

4.5.5 Lokiviestit

Etäkäyttöohjelmiston lokitiedoston kirjoittamiseen käytettiin edellisen sukupolven laitteella Go-kielen tarjoamaa log-lokikirjastoa. Tämä kirjasto on ominaisuuksiltaan hyvin rajallinen ja ei esimerkiksi tarjoa erillisiä lokiviestien tasoja. Hyödyllisempiä lokiviestejä varten lokiviestikirjasto korvattiin Uberin kehittämällä zap-kirjastolla (kuva 12). Zap tarjoaa valmiiksi kattavat lokiviestien tasot, jotka ovat konfiguroitavissa siten, että esimerkiksi debug-tason viestit voidaan jättää lokeista pois.



KUVA 12. Info-tasoisen lokiviestin formaatti log ja zap -kirjastoilla.

Etäkäyttöohjelmiston asetuksissa määritetään käytettävä lokiviestien taso (kuva 13). Muut zap-kirjaston asetukset päätettiin toistaiseksi pitää kovakoodattuina (kuva 14). Lokiviestien formaatti on määritelty mahdollisimman helposti ihmisen luettavaksi, sillä lokien lukemiseen ei ole käytössä erillistä ohjelmistoa. Zap tukee oletuksena myös JSON-formaattia lokiviesteille, joka olisi hyödyllinen jos viestejä esimerkiksi lähetettäisiin eteenpäin pilvipalvelulle käsittelyä varten.

```
// Logging config for zap package
type logging struct {
    Level string `json:"level,omitempty"`
}
```

KUVA 13. Etähallintaohjelmiston lokiasetukset.

```go
func InitLogger() error {
    logFile := "/tmp/smarthub/log/hub.log"
    if err := os.MkdirAll(filepath.Dir(logFile), os.ModePerm); err != nil {
        return err
    }

    cfg := zap.NewProductionConfig()
    cfg.OutputPaths = []string{logFile}
    cfg.Encoding = "console"
    cfg.EncoderConfig.EncodeTime = zapcore.TimeEncoderOfLayout(time.RFC3339)

    level = zap.NewAtomicLevelAt(zap.DebugLevel)
    cfg.Level = level

    logger, err := cfg.Build()
    if err != nil {
        return err
    }

    zap.ReplaceGlobals(logger)
    return nil
}
```

KUVA 14. Zap-lokikirjaston alustus.

Viestien sisältämien tiedostopolkujen huomattiin olevan absoluuttisia sen sijaan, että ne olisivat projektikansioon relatiivisia. Tiedostopoluissa oli siis nähtävissä esimerkiksi käyttäjänimiä binäärin kääntäneeltä tietokoneelta. Binäärin kääntävälle go build -komennolle voitiin antaa –trimpath optio, jolla tiedostopoluista jätetään pois projektikansion ulkopuoliset kansiot (kuva 15).

```
2024/03/28 02:01:17 Could not read past signal measurements: no signal strength values found

2024-04-02T02:29:41Z    error   modem/signal.go:542     Error reading past signal measurements: no s
ignal strength values found
OpenWrt/smarthub/modem.GetNetworkInfo
        OpenWrt/smarthub/modem/signal.go:542
OpenWrt/smarthub/messaging.Heartbeat
        OpenWrt/smarthub/messaging/messages.go:104
main.main.func5
        OpenWrt/smarthub/hub.go:473
```

KUVA 15. Error-tasoisen lokiviestin formaatti log ja zap -kirjastoilla.

4.5.6 Järjestelmäasetusten hallinta

Laitteiden käyttöjärjestelmän ja monien pakettien asetusten määrittelyyn käytettävät UCI-asetustiedostot on hallittu tähän mennessä uci-komentorivityökalulla, jota on kutsuttu Go-koodista exec-paketin avulla (kuva 16). Tämä on korvattu go-uci-kirjastolla, joka osaa lukea sekä kirjoittaa UCI-asetustiedostoja (kuva 17). Kirjaston avulla saadaan asetusten kirjoittamista ja lukemista helpotettua. Virhetilanteista saadaan suoraan tieto sen sijaan, että pitäisi tarkastella uci-komennon paluuarvoa tai virhetulostetta. Lisäksi argumenttien antamisesta tulee helpommin ymmärrettäviä, kun argumenttien määrät ovat nähtävissä suoraan funktiokutsussa.

```
        command := `uci batch << EOF
set network.wan.disabled='1'
set network.lan.ifname='eth0.1'
commit network
EOF`

        // Execute the command
        if _, errStream, err := tr4util.Command(command); len(errStream) > 0 {
            return errors.New(strings.TrimSpace(string(errStream)))
        } else if err != nil {
            return err
        }
    }
```

KUVA 16. UCI asetustiedoston asetusten muuttaminen uci komennolla.

```
if ok := uci.Set("network", "lan_bridge", "ports", "eth0.1"); !ok {
    return errors.New("could not add ethernet to LAN bridge")
}
if ok := uci.Set("network", "wan", "disabled", "1"); !ok {
    return errors.New("could not enable wan interface")
}
uci.Commit()
```

KUVA 17. UCI asetustiedoston asetusten muuttaminen go-uci kirjastolla.

4.5.7 Diagnostiikka

Internetyhteyden nopeuden mittaukseen käytettiin omaa speedtest.net-palvelimia vasten tehtyä koodia, joka toimi 4G-yhteyksillä tarpeeksi hyvin mutta nopeammilla 5G-yhteyksillä tulokset olivat huomattavasti alemmat kuin selaimella tehdessä. Oma koodi korvattiin speedtest-go-kirjastolla (kuva 18), jolla saadut tulokset (kuva 19) vastasivat paremmin selainta. Suurin etu speedtest-go-kirjastolla vanhaan koodiin verrattuna on tuki usean yhteyden käyttöön nopeustestin aikana samanaikaisesti, mikä auttaa varsinkin nopeiden 5G-yhteyksien kanssa tarkemman tuloksen saavuttamiseksi. Lisäksi ylläpito helpottuu, kun satojen koodirivien itse tehty paketti korvaantui yhdellä muutaman kymmenen rivin funktiolla.

```
speedtestClient := speedtest.New()
speedtestClient.Manager.Reset()
speedtestClient.SetNThread(numTesters)

user, err := speedtestClient.FetchUserInfo()
if err != nil {
    return message, err
}
speedtestClient.User = user

serverList, err := speedtestClient.FetchServers()
if err != nil {
    return message, err
}
// FindServer returns the server with the lowest latency if no server IDs are given
server, err := serverList.FindServer([]int{})
if err != nil {
    return message, err
}

// Do the speedtest
s := server[0]
if err := s.PingTest(func(latency time.Duration) {}); err != nil {
    return message, err
}
if err := s.DownloadTest(); err != nil {
    return message, err
}
if err := s.UploadTest(); err != nil {
    return message, err
}
```

KUVA 18. Nopeustestin suorittaminen speedtest-go-kirjastolla.

```
message.Speedtest.Latency = strconv.FormatFloat(float64(s.Latency.Nanoseconds())/1000000.0, 'f', -1, 64)
message.Speedtest.Download = strconv.FormatFloat(float64(s.DLSpeed), 'f', -1, 64)
message.Speedtest.Upload = strconv.FormatFloat(float64(s.ULSpeed), 'f', -1, 64)
```

KUVA 19. Nopeustestin tulosten luku.

Palvelevaa solua ja naapurisoluja näyttävä diagnostiikka täytyi päivittää tukemaan 5G-soluja. Diagnostiikan lähettämistä varten signaalin tiedot parsitaan modeemin palauttamasta pilkuilla erotetusta listasta JSON-formaattiin. 5G NSA -verkon tapauksessa käytössä olevia soluja palautuu kaksi, yksi 5G-solu (kuva 20) ja yksi 4G-solu. 5G SA -soluille (kuva 21) riitti arvojen parsiminen modeemin vastauksesta samalla tavalla kuin 4G ja sitä edeltäneiden verkkojen soluilla.

```
type CellNR5GNSA struct {
    NetworkType       string `json:"networkType"`      // "NR5G-NSA"
    MobileCountryCode int    `json:"mcc"`              // <MCC>
    MobileNetworkCode int    `json:"mnc"`              // <MNC>
    PhysicalCellID    *int   `json:"pcid,omitempty"`   // <PCID>
    Rsrp              *int   `json:"rsrp,omitempty"`   // <RSRP>
    Sinr              *int   `json:"sinr,omitempty"`   // <SINR>
    Rsrq              *int   `json:"rsrq,omitempty"`   // <RSRQ>
    Arfcn             *int   `json:"arfcn,omitempty"`  // <ARFCN>
    Band              uint32 `json:"band,omitempty"`   // <band>
}
```

KUVA 20. Modeemin palauttamat 5G NSA -solun signaalin tiedot.

```
type CellNR5GSA struct {
    CellType           string `json:"cellType"`              // servingcell
    State              string `json:"state"`                 // <state>
    NetworkType        string `json:"networkType"`           // "NR5G-SA"
    Mode               string `json:"mode"`                  // <duplex_mode>
    MobileCountryCode  int    `json:"mcc"`                   // <MCC>
    MobileNetworkCode  int    `json:"mnc"`                   // <MNC>
    CellID             int64  `json:"cellId"`                // <cellID>
    PhysicalCellID     *int   `json:"pcid,omitempty"`        // <PCID>
    TrackingAreaCode   *int64 `json:"tac,omitempty"`         // <TAC>
    Arfcn              *int   `json:"arfcn,omitempty"`       // <ARFCN>
    Band               uint32 `json:"band,omitempty"`        // <band>
    NRDLBandwidth      *int   `json:"nrdlBandwidth,omitempty"` // <NR_DL_bandwidth>
    Rsrp               *int   `json:"rsrp,omitempty"`        // <RSRP>
    Rsrq               *int   `json:"rsrq,omitempty"`        // <RSRQ>
    Sinr               *int   `json:"sinr,omitempty"`        // <SINR>
    Scs                *int   `json:"scs,omitempty"`         // <scs>
    SelectReceiveLevel *int   `json:"srxlev,omitempty"`      // <srxlev>
}
```

KUVA 21. Modeemin palauttamat 5G SA -solun signaalin tiedot.

4.6 Laiteohjelmiston käännösympäristö

Buildrootin vaatimuksille ei ollut olemassa dokumentaatiota. Yrityksen ja erehdyksen kautta saatiin selvitettyä millä eri työkalujen versioilla laiteohjelmisto saatiin käännettyä. Ubuntu 18 sisälsi lähes kaikista työkaluista oikean version, joten se otettiin Docker-kontin pohjaksi. Puuttuvat paketit voidaan asentaa apt-get ohjelmistolla. Python 3.10 asentamiseksi täytyi lisätä deadsnakes/ppa-pakettivarasto, joka tarjoaa vanhemmillekin Ubuntun versioille Pythonin uudempia versioita (kuva 22).

```
FROM ubuntu:18.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update &&\
    apt-get install -y software-properties-common &&\
    add-apt-repository -y -u ppa:deadsnakes/ppa &&\
    apt-get install -y sudo time git-core subversion build-essential gcc-multilib \
                       libncurses5-dev zlib1g-dev gawk flex gettext wget unzip python \
                       gcc binutils bzip2 flex perl make unzip gawk libz-dev libc-dev rsync \
                       python3.10 &&\
    apt-get clean

RUN useradd -m openwrt &&\
    echo 'openwrt ALL=NOPASSWD: ALL' > /etc/sudoers.d/openwrt

USER openwrt
WORKDIR /home/openwrt/
```

KUVA 22. Käännösympäristön Docker-kontin asetukset.

Lisäksi Buildroottiin tehtiin m4 ja Pythonin osalta muutoksia uudemman version tukemiseksi, jotta kaikki paketit saatiin kääntymään ilman virheitä. Pythonista tarvittiin versio 3.10, jonka tukemiseksi riitti Buildrootin tekemän Python version tarkastuksen muokkaaminen hyväksymään kyseinen versio (kuva 23).

```
$(eval $(call SetupHostCommand,python,Please install Python >= 3.5, \
    python3.10 -V 2>&1 | grep 'Python 3', \
    python3.9 -V 2>&1 | grep 'Python 3', \
    python3.8 -V 2>&1 | grep 'Python 3', \
    python3.7 -V 2>&1 | grep 'Python 3', \
    python3.6 -V 2>&1 | grep 'Python 3', \
    python3.5 -V 2>&1 | grep 'Python 3', \
    python3 -V 2>&1 | grep -E 'Python 3\.[5-10]\.?'))
```

KUVA 23. Python-version tarkastus.

M4 päivittämiseksi versiosta 1.4.18 versioon 1.4.19 täytyi muokata m4 kääntämiseen käytettävä makefile kääntämään uudempi versio. Tähän riitti PKG_VERSION ja PKG_HASH -muuttujien päivittäminen (kuva 24). Näillä muutoksilla Buildroot osaa ladata oikean version lähdekoodit käännöstä varten.

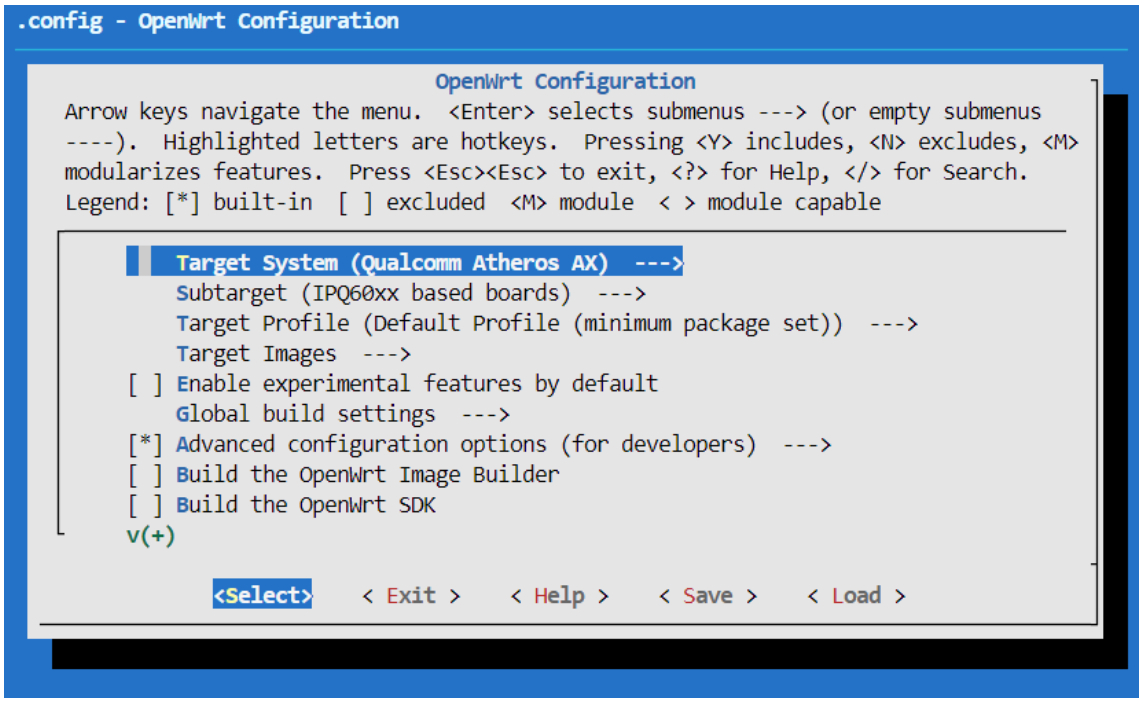
```
PKG_NAME:=m4
PKG_CPE_ID:=cpe:/a:gnu:m4
PKG_VERSION:=1.4.19

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.xz
PKG_SOURCE_URL:=@GNU/$(PKG_NAME)
PKG_HASH:=63aede5c6d33b6d9b13511cd0be2cac046f2e70fd0a07aa9573a04a82783af96
PKG_CAT:=xzcat
```

KUVA 24. M4-paketin 1.4.19 asetukset.

Buildroot toimii vain Linux-ympäristöissä. Sillä on riippuvuuksia useisiin Linux-paketteihin sekä vaatimus merkkikokoriippuvaisesta tiedostojärjestelmästä. Windows-käyttöjärjestelmällä on mahdollista käyttää Linux-ohjelmistoja WSL:n kautta. Windows-tiedostojärjestelmä on kuitenkin oletuksena merkkikokoriippumaton ja tämä on otettava huomioon Buildroottia käytettäessä. Tähän ongelmaan on kaksi ratkaisua. Windows-tiedostojärjestelmässä on mahdollista muuttaa yksittäisiä kansioita merkkijonoriippuvaisiksi, jolloin buildroot voisi olla Windows-tiedostojärjestelmän puolella. Tämä kuitenkin voi aiheuttaa ongelmia Windows-ohjelmistojen kanssa, jotka olettavan tiedostojärjestelmän olevan merkkijonoriippumaton. Toinen vaihtoehto on käyttää WSL:n sisäistä Linux-tiedostojärjestelmää, joka on merkkijonoriippuvainen. WSL:n sisäisen tiedostojärjestelmän käyttö Windowsin puolelta on kuitenkin vaikeampaa kuin Windows-tiedostojärjestelmän käyttö ja monet ohjelmat eivät osaa käyttää WSL:n tiedostojärjestelmää ollenkaan. Esimerkiksi Visual Studio Code vaatii WSL-lisäosan, jolla täytyy erikseen yhdistää WSL-järjestelmään, jotta sen sisältämät tiedostot saadaan näkyviin. Työssä päädyttiin käyttämään WSL:n sisäistä tiedostojärjestelmää, jotta Windows ja Linux -puoli saadaan pidettyä erillään toisistaan, ja Docker-konttia, jolla saadaan hallittua Buildrootin riippuvuuksia ilman että isäntäjärjestelmään täytyy asentaa paketteja.

Laiteohjelmiston kääntämisen asetukset määritellään komentorivipohjaisella menuconfigilla (kuva 25). Tällä valitaan niin laitteen käyttämä arkkitehtuuri kuin

Linux-ytimen asetukset sekä laiteohjelmiston mukaan sisällytettävät paketit. Lopputuloksena menuconfig luo .config tiedoston, jota buildroot käyttää kääntääkseen toolchainin oikealle arkkitehtuurille sekä haluttujen pakettien lataamiseen ja kääntämiseen.



KUVA 25. Menuconfigin käyttöliittymä.

Buildroot ei sisällä työkaluja Go-koodin kääntämiseen, sillä suurin osa paketeista on tehty C-kielellä. Sen sijaan että Go-koodille olisi lisätty tuki buildroottiin, päätettiin lisätä etähallintaohjelmisto laiteohjelmistoon valmiina IPK-pakettina. Etähallintaohjelmiston kääntämiseksi ja IPK-paketiksi paketoinniksi oli jo olemassa valmiit skriptit, joita voitiin tähän tarkoitukseen hyödyntää. Buildrootissa ei ole valmista tukea IPK-paketin sisällyttämiseksi. Luomalla pakettia varten makefile (kuva 26), joka kopioi IPK-paketin samaan kansioon johon käännöksen aikana IPK-paketti luotaisiin, saatiin valmis IPK-paketti laiteohjelmistoon mukaan.

```
include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=smarthub
PKG_RELEASE:=1.3.723

PKG_FORCE_PREBUILT:=1


include $(INCLUDE_DIR)/package.mk


define Package/smarthub
  SECTION:=utils
  CATEGORY:=Utilities
  TITLE:=smarthub
  DEPENDS:=+shadow-usermod +kmod-usb-serial-option +kmod-usb-serial \
   +kmod-usb-serial-wwan +usb-modeswitch +kmod-mii +kmod-usb-net \
   +kmod-usb-wdm +kmod-usb-net-qmi-wwan +uqmi +luci-proto-qmi \
   +coreutils-base64 +dropbearconvert +kmod-nfnetlink \
   +kmod-nf-conntrack-netlink +irqbalance
endef

define Package/smarthub/description
  Track4services application, smarthub.
endef



define Build/Prepare
    mkdir -p  $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Build/Compile

endef

define Package/smarthub/install
    $(INSTALL_DIR) $(1)/etc/crontabs
    $(INSTALL_DIR) $(1)/etc/dropbear
    $(INSTALL_DIR) $(1)/etc/hotplug.d/dhcp
    $(INSTALL_DIR) $(1)/etc/init.d
    $(INSTALL_DIR) $(1)/etc/smarthub
    $(INSTALL_DIR) $(1)/etc/smarthub/multiwan
    $(INSTALL_DIR) $(1)/etc/smarthub/config
    $(INSTALL_DIR) $(1)/etc/uci-defaults
endef
```

KUVA 26. Valmiiksi käännetyn paketin makefile.

5 POHDINTA

Opinnäytetyön tuloksena saatiin onnistuneesti rakennettua käännösympäristö OpenWrt-laiteohjelmiston kääntämiseksi, sekä integroitua olemassa oleva etähallintaohjelmisto laiteohjelmiston sisälle. Laitevalmistajan lähdekoodin dokumentaation puute aiheutti alkuun yllättäviä vaikeuksia käännösympäristön kanssa. Toinen yllätys lähdekoodissa oli ajurituen puute laitteen mukana toimitettavalle 5G-modeemille. Näistä ongelmista kuitenkin selvittiin ja samalla pääsi oppimaan syvemmin Linux-ytimen käännösprosessista ja ajureista.

Etähallintaohjelmiston uudet ominaisuudet olivat lopulta hyvin suoraviivaisia, joskin työläitä lisätä. Ominaisuudet saatiin myös toimimaan edellisen sukupolven laitteilla, siten että sama ohjelmistopaketti on asennettavissa sekä uusille että vanhoille laitteille. Varsinkin MWAN3-paketin tuomat ominaisuudet avaavat uusia mahdollisuuksia laitteen tullessa markkinoille vuoden 2024 kesään mennessä.

Jatkokehitystä varten olisi hyvä ottaa käyttöön Image Builder nopeampia laiteohjelmiston käännöksiä varten. Linux-ytimeen ei ole nähtävissä lisämuutoksia lähitulevaisuudessa, joten Buildrootilla voitaisiin kääntää Image Builder, jota taas käytettäisiin laiteohjelmiston rakentamiseen. Pidemmällä aikavälillä Image Builder tulee käyttöön joka tapauksessa, kun OpenWrt 24 -julkaisun yhteydessä tulee tuki IPQ6018-piirisarjalle. OpenWrt tarjoaa jokaiselle tuetulle laitteelle valmiin Image Builder Docker-kontin, joita projektissa käytetään jo nyt edellisen sukupolven laiteohjelmistojen rakentamiseen. Tämän käyttöönoton jälkeen voidaan Buildrootin käytöstä luopua täysin. Samalla päästäisiin eroon QSDK Linux -ytimestä sekä siihen tehdyistä päivitystiedostoista.

LÄHTEET

Buildroot. 2023. About Buildroot. Verkkosivu. Viitattu 10.11.2023. https://buildroot.org/downloads/manual/manual.html#_about_buildroot

Dan Siemon. n.d. Queueing in the Linux Network Stack. Viitattu 12.1.2024. https://www.coverfire.com/articles/queueing-in-the-linux-network-stack/

DENX Software Engineering. 2021. U-Boot. Verkkosivu. Viitattu 17.11.2023 https://www.denx.de/project/u-boot/

Eric Sandler. 2005. The Open Source WRT54G Story. Verkkosivu. Viitattu 10.11.2023. https://www.wi-fiplanet.com/the-open-source-wrt54g-story/

Microsoft. 2023. Adjust case sensitivity. Verkkosivu. Viitattu 10.11.2023. https://learn.microsoft.com/en-us/windows/wsl/case-sensitivity

OpenWrt. 2023a. About the OpenWrt/LEDE project. Verkkosivu. Viitattu 10.11.2023. https://openwrt.org/about

OpenWrt. 2023b. LuCI essentials. Verkkosivu. Viitattu 10.11.2023. https://openwrt.org/docs/guide-user/luci/luci.essentials

OpenWrt. 2023c. OpenWrt version history. Verkkosivu. Viitattu 10.11.2023. https://openwrt.org/about/history

OpenWrt. 2023d. The UCI system. Verkkosivu. Viitattu 10.11.2023. https://openwrt.org/docs/guide-user/base-system/uci

OpenWrt. 2023e. Using the Image Builder. Verkkosivu. Viitattu 10.11.2023. https://openwrt.org/docs/guide-user/additional-software/imagebuilder

Rob Pike. 2012. Go at Google: Language Design in the Service of Software Engineering. Verkkosivu. Viitattu 28.3.2024. https://go.dev/talks/2012/splash.article

U-Boot GitHub. 2023. README. Verkkosivu. Viitattu 17.11.2023 https://github.com/u-boot/u-boot/blob/master/README