TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAY40LT

Artur Vainola 120414

CONTIKI OPERATSIOONISÜSTEEM ATMELI MIKROKONTROLLERIL

Bakalaureusetöö

Priit Ruberg
MSc
Nooremteadur

Tallinn 2015

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: Artur Vainola

12.06.2015

Annotatsioon

Käesoleva töö eesmärgiks on selgitada Contiki operatsioonisüsteemi erinevaid kasutusvõimalusi Atmel AVR-Atmega128RFA1 mikrokontrolleril. Töös kasutatakse Tallinna Tehnikaülikooli Arvutitehnika Instituudis kasutusel olevaid Dresden Elektroniku poolt toodetud deRFmega128 raadiomooduli ja deRFnode arendusplaadi komplekte.

Töö käigus tutvutakse põhjalikult kasutatava riistvara ja Contiki operatsioonisüsteemiga. Selgitatakse Contiki operatsioonisüsteemi ülesehitust ja töö põhimõtteid, lisatakse Contiki operatsioonisüsteemile ohjurid deRFnode arendusplaadi USB liidese ja temperatuuri-, valgusheleduse- ja kiirendusandurite kasutamiseks, luuakse tähe- ja silmustopoloogiaga juhtmevabad sensorvõrgud ning ühendatakse loodud võrgud Internetiga. Eesmärkide saavutamiseks ja tulemuste testimiseks kirjutatakse C programmeerimiskeeles neli ohjurit ja kaheksa Contiki rakendust.

Töö tulemuseks on töötav Internetiga ühendatud juhtmevaba sensorvõrk, mille põhjal on võimalik edasi arendada rakendusi erinevatele valdkondadele, alustades tarkadest hoonetest ja liiklusohutusest kuni looduskaitse ja tervishoiuni.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 4 peatükki, 9 joonist, 1 tabel.

Abstract

Contiki operating system on Atmel microcontroller

The goal of the thesis was to explore different possibilities of using Contiki operating system on Atmel AVR-Atmega128RFA1. Used devices are Dresden Elektronik deRFmega128 radiomodule and deRFnode development board set, which are available in Department of Computer Engineering in Tallinn University of Technology.

During the work, used hardware and Contiki operating system are thoroughly studied. Contiki operating system structure and working principles are ascertained, new drivers for deRFnode USB interface and temperature, luminosity and acceleration sensors are written, wireless sensor network with star and mesh topology are established and connected to the Internet. To acheive these goals and test the results, four drivers and eight Contiki applications are written in C programming language.

The final result is a working wireless sensor network connected to Internet, which provides a ground for futher developing of applications for different fields like smart buildings, road safety, healthcare and many more.

The thesis is written in estonian and contains 38 pages of text, 4 chapters, 9 figures, 1 table.

Lühendite ja mõistete sõnastik

6loWPAN	*IPv6 Low Power Wireless Private Area Network*, juhtmevaba andmeedastusprotokoll
ADC	*analog-to-digital converter*, analoog-digitaalmuundur
API	*Application Programming Interface*, rakendusliides
ARM	*Advanced RISC Machine*, mikroprotsessori arhitektuur
AVR	*modified Harvard architecture machine*, mikrokontrolleri arhitektuur
CMOS	Complementary *Metal-Oxide-Semiconductor*, komplementaarne metall-oksiid-pooljuht
CoAP	*Constrained Application Protocol*, piiratud ressurssidega seadmetel kasutatav võrgurakenduste protokoll
CPU	*Central Proccessing Unit*, kesktöötlusseade
CSMA	*Carrier Sense multiple access*, kandjatajuriga hulgipöördus
EEPROM	*Electrically Erasable Programmable Read-Only Memory*, elekterkustutusega püsimälu
GPIO	*General-purpose input/output*, üldotstarbeline sisend-väljund
GUI	*graphical user interface*, graafiline kasutajaliides
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastamise protokoll
HVSP	*High Voltage Serial Programming*, Atmel mikrokontrollerite programmeerimise liides
I2C	*Inter-Integrated Circuit*, kahesuunaline kahesooneline järjestiksiin
IEEE	*Institute of Electrical and Electronics Engineers*, Elektri- ja Elektroonikainseneride Instituut
IETF	*Internet Engineering Task Force*, Internetiehituse töörühm
IPv4	*Internet Protocol version 4*, Interneti protokolli versioon 4
IPv6	*Internet Protocol version 6*, Interneti protokolli versioon 6
JTAG	*Joint Test Action Group*, Ühendatud Testimisrühm
LCD	*liquid-crystal display*, vedelkristallkuvar
LED	*Light-Emitting Diode*, valgusdiood
LGA	*land grid array*, riistvaras kasutatav pindühendusliides
LPP	*Low-Power Probing*, meediumipöörduse juhtimise protokoll
MAC	*Media Access Control*, meediumipöörduse juhtimine
MIPS	*million instructions per second*, miljon käsku sekundis

PDI	*Program and Debug Interface*, Atmel mikrokontrollerite programmeerimise liides
PP	*Parallel Programming Interface*, Atmel mikrokontrollerite programmeerimise liides
PPP	*Point-to-Point Protocol*, kakspunktprotokoll
RDC	*Radio Duty-Cycling*, raadio töötsükkel
REST	*Representational State Transfer*, tarkvaraarhitektuur
RESTful	*Web service APIs that adhere to the REST architectural constraints*, REST arhitektuuril põhinev võrguteenus
RISC	*reduced instruction set computer*, kärbitud käsustikuga arvuti
RPL	*Routing Protocol for Low-power and Lossy Networks*, juhtmevabades sensorvõrkudes kasutatav ruutimisprotokoll
RXD	*Received Data*, andmete vastuvõtmise signaal
SLIP	*Serial Line Internet Protocol*, jadaliini internetiprotokoll
SPI	*Serial Peripheral Interface*, sünkroonne järjestikliides
TCP	*Transmission Control Protocol*, edastusohje protokoll
TTL	*Transistor–transistor logic*, TTL-loogikaskeem
TWI	*Two-Wire Interface*, kahe juhtme liides
TXD	*Transmitted Data*, andmete edastamise signaal
UART	*universal asynchronous receiver/transmitter*, järjestikühenduse liides
UDP	*User Datagram Protocol*, kasutajadatagrammi protokoll
USB	*Universal Serial Bus*, universaalne jadaliides

Sisukord

Jooniste nimekiri

Tabelite nimekiri

Sissejuhatus

Teema valikul lähtus autor sellest, et töö hõlmaks nii uue ja huvitava valdkonna uurimist, kui ka praktiliste lahenduste rakendamist. Oluliseks kaalukeeleks oli kindlasti ka võimalus tegeleda praktilises osas riistvara programmeerimisega. Lisaks eelnevale motiveeris just seda teemat valima teadmine, et tehtud tööl võib hiljem reaalne väljund olla.

Töö eesmärgiks on selgitada Contiki operatsioonisüsteemi erinevaid kasutusvõimalusi Atmel AVR-Atmega128RFA1 mikrokontrolleril. Töös kasutatakse Tallinna Tehnikaülikooli Arvutitehnika Instituudis kasutusel olevaid Dresden Elektroniku poolt toodetud deRFmega128 raadiomooduli ja deRFnode arendusplaadi komplekte. Konkreetsemateks eesmärkideks on arendusplaadi USB liidese ja temperatuuri-, valgusheleduse- ja kiirendusandurite kasutamine, tähe- ja silmustopoloogiaga juhtmevabade võrkude loomine ning loodud võrgu ühendamine Internetiga. Lisaks eelnevale on eesmärgiks koostada juhend Contiki arenduskeskkonna seadistamiseks ja rakenduste arendamiseks kasutatavale platvormile.

Töö on jagatud nelja ossa. Esimeses osas tutvutakse lähemalt kõikide töös kasutavate seadmetega. Antakse ülevaade nende olulisematest omadustest ja võimalikest kasutusaladest.

Teises osas õpitakse tundma Contiki operatsioonisüsteemi. Tutvutakse selle eriliste omaduste ja arenguga. Põhjalikumalt uuritakse Contiki ülesehitust ja selles kasutusel olevaid tehnoloogiaid, mudeleid ja protokolle.

Kolmandas osas tutvutakse Contiki operatsioonisüsteemi arendamisega, alustades erinevatel platvormidel arenduskeskkonna üles seadmisest. Seejärel uuritakse põhjalikumalt Contiki lähtekoodis kasutusel olevat *Makefile* konstruktorit ja lõpuks selgitatakse, kuidas toimub programmide kompileerimine ja seadmetele laadimine.

Neljas osa käsitleb eesmärkide saavutamiseks kasutatud praktilisi lahendusi. Selgitatakse töö käigus tekkinud probleeme ja leitud lahendusi. Lahendamata probleemide puhul kirjeldatakse uurimise tulemusi ja võimalikke põhjusi, või viidatakse valdkonnale, mida oleks vaja põhjalikumalt uurida.

1. Kasutatud riistvara ülevaade

Töös kasutatakse Dresden Elektroniku poolt toodetud deRFmega128 raadiomoodulit, mis baseerub Atmel AVR-Atmega128RFA1 mikrokontrolleril. Raadiomoodul on ühendatud samuti Dresden Elektroniku poolt toodetud deRFnode arendusplaadiga. Seadmete programmeerimiseks kasutatakse Atmeli AVR Dragon programmaatorit ning abivahendina võrguliikluse jälgimiseks Dresden Elektroniku deRFusb raadiomoodulit. Järgnevad peatükid annavad ülevaate kõikide kasutatavate seadmete kohta.

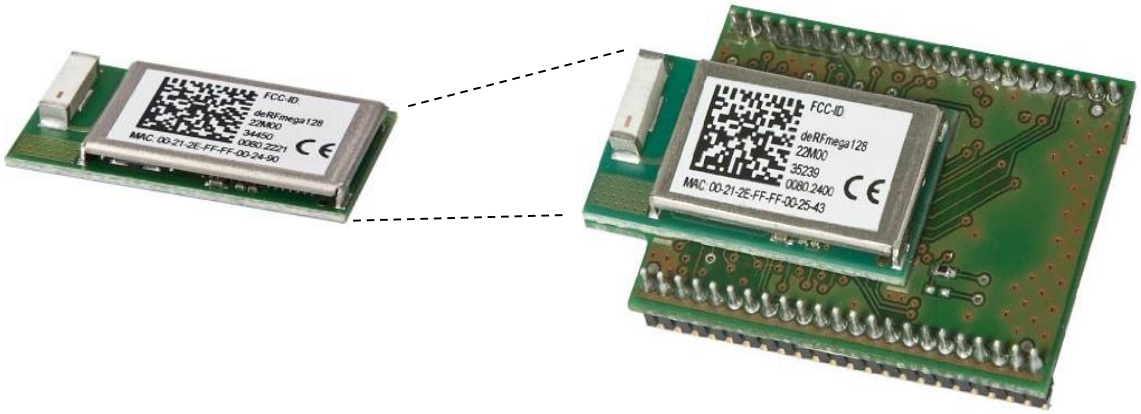
1.1. Atmel ATmega128RFA1 mikrokontroller

ATmega128RFA1 on madala võimsusega 8-bitine CMOS mikrokontroller, mis põhineb täiustatud RISC arhitektuuril. Efektiivne käsustik võimaldab saavutada läbilaskevõime kuni 1 MIPS MHz kohta, mis pakub arendajale võimaluse valida töökiiruse ja madala energiatarbe vahel. Kiipi on integreeritud kõrge andmeedastuskiirusega 2,4 GHz sagedusala transiiver. Andmeedastuskiiruseks on võimalik seadistada 250 kb/s kuni 2Mb/s. Transiiveril on väga tundlik vastuvõtja (-100 dBm) ja kõrge väljundvõimsus (3.5 dBm). Maksimaalne taktsagedus on 16MHz ja toitepinge 1.8V – 3.6V. Programmi ja andmete hoidmiseks on 128 kilobaiti välkmälu ja 4096 baiti EEPROM mälu. [3]

1.2. Raadiomoodul deRFmega128

Töös kasutatav Dresden Elektronik deRFmega128-22M00 (Joonis 1) on ATmega128RFA1 mikrokontrolleri ja integreeritud antenniga raadiomoodul. Mooduli ühendamiseks on alumisel küljel asuv 51st 0,80 mm sammuga kontaktist koosnev LGA liides. Seadme tehnilised andmed leiab Lisa 1-s. [14]

Mooduli ühendamise lihtsustamiseks on see joodetud deRFmega128-22T00 vaheplaadi külge (Joonis 1). Vaheplaat on nii öelda adapter, mis asendab joodetava LGA liidese mugavamate viikudega (2 x 23 1,27 mm sammuga viiku), tagades sealjuures ligipääsu pea kõikidele Atmega128RFA1 portidele ja võimaldades väga lihtsalt mooduli ühendamist Dresden Elektronik arendusplaadiga deRFnode. [1]



Joonis 1. Raadiomoodul deRFmega128-22M00 eraldi (vasakul) ja koos deRFmega128-22T00 adapterplaadiga (paremal).

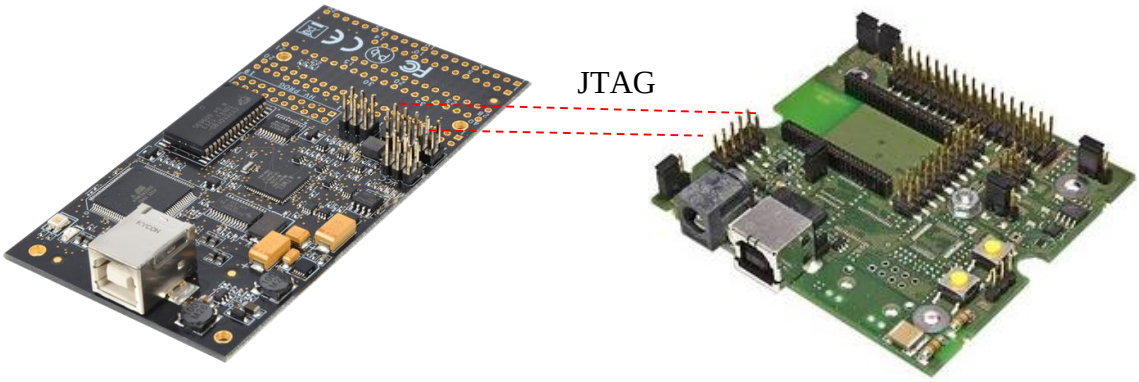
1.3. Arendusplaat deRFnode

Arendusplaat deRFnode (Joonis 2) on väga mitmekülgne ja ühildub kõikide Dresden Elektonik raadiomoodulitega. Tänu kokkusobivale ning energiasäästlikule disainile võib kombineeritult deRFmega128 mooduliga seade töötada patareide toitel kuni mitu aastat ja madala voolutarbe tõttu on võimalik toiteks kasutada ka taastuvenergiat. [6]

Arendusplaat deRFnode on varustatud järgmiste liidestega: 2 x JTAG (AVR + ARM), USB, I2C, UART, SPI, GPIO, ADC. Seadme programmeerimine toimub JTAG või SPI liidese ja programmaatori abil. Programmi silumiseks on võimalik kasutada USB ja UART liideseid. Sisseehitatud temperatuuri-, valgusheleduse- ja kiirendusandurid võimaldavad kasutada seda erinevateks mõõtmisteks. Seadmel on kolm valgusdioodi ja kaks surunuppu, ning 34-viiguse laiendusliidese abil on võimalik ühendada erinevaid lisaseadmeid. [6]

1.4. Programmaator AVR Dragon

Töös kasutatav AVR Dragon on Atmeli programmaator (Joonis 2), mille abil on võimalik programmeerida kõiki AVR perekonna mikrokontrollereid. Programmaatoril on arvutiga ühendamiseks USB 2.0 liides (tüüp B pistik) ja programmeerimiseks SPI, JTAG , HVSP (*High Voltage Serial Programming*), PP (*Parallel Programming*), PDI (*Program and Debug Interface*) ja aWire liidesed. AVR-Atmega128RFA1 kontrollereid saab siiski programmeerida vaid SPI ja JTAG liideste abil, millest sai kaabli olemasolu tõttu valitud viimane. Tööks vajaliku toitevoolu saab programmaator üle USB liidese. [4]



Joonis 2. AVR Dragon programmaator (vasakul) ja Arendusplaat deRFnode (paremal). Punktiirjoonega on tähistatud JTAG ühendus.

1.5. Raadiomoodul deRFusb

Dresden Elektoniku deRFusb-23E00 seadet (Joonis 3) koos vastava tarkvaraga kasutati töös abivahendina raadioside pakettide jälgimiseks. Seade kasutab Cortex-M3 arhitektuuri, sellel on 256 kilobaiti välkmälu, 64 kilobaiti RAM-i ja 2.4 GHz sagedusala raadiosaatja AT86RF231. Tänu nendele omadustele sobib seade hästi IEEE 802.15.4 võrgu liikluse jälgimiseks. [18]

Joonis 3. Raadiomoodul deRFusb.

2. Contiki operatsioonisüsteemi ülevaade

Contiki algversioon loodi aastal 2002 Rootsi Arvutiteaduste Instituudis (*Swedish Institute of Computer Science*) ja selle loojaks on Adam Dunkels. Edasine arendus on toimunud rahvusvahelise meeskonnana, kus on osalenud juba palju arendajaid erinevatest ettevõtetest ja ülikoolidest, nagu näiteks Atmel, Cisco, ENEA, ETH Zurich, Redwire, RWTH Aachen Univercity, Oxford University, SAP, Sensinode, ST Microelectronics, Zolertia ja paljud teised. Contiki nimi tuleb Thor Heyerdahl'i kuulsast Kon-Tiki parvest. [19]

Contiki on avatud lähtekoodiga operatsioonisüsteem piiratud mälu, jõudluse ja energiatarbega seadmetele, võimaldades neid lihtsasti ühendada Internetiga. Tüüpilisel Contiki süsteemil on mälumaht kilobaitides, energiatarve millivattides, protsessori kiirus megahertsides ja võrguühenduse kiirus sadades kilobaitides/sekundis. [19][10]

Contiki süsteem vajab keskmiselt kõigest 10 kilobaiti RAM-i ja 30 kilobaiti ROM-i (täielik süsteem koos graafilise kasutajaliidesega vajab 30 kilobaiti RAM-i). Mälu kokkuhoidmiseks, tagades seejuures hea käsuvoo (*control flow*), kasutab Contiki spetsiaalset multitegumtöötluse mehhanismi *Protothread*, mis on segu sundmusjuhitavast- ja lõimtöötlusmehhanismist. [19]

Contiki toetab valikulist tõrjuv multitegumtöötlust (*per-process optional preemptive multi-threading*) – protsesside vahelist suhtlust sündmustepõhisel sõnumite edastamisel. Lisaks on võimalik kasutada graafilise kasutajaliidese alamsüsteemi, mis toetab lokaalselt ühendatud terminali kasutamist ja üle võrgu pildi edastamist. [10]

Contiki toetab täielikult nii IPv4 ja IPv6 standardeid nagu UDP, TCP ja HTTP protokollid, kui ka IETF-i poolt standardiseeritud madala võimsusega juhtmevabades võrkudes kasutatavaid IPv6 võrguprotokolle nagu 6loWPAN muganduskihi protokoll, RPL ruutimisprotokoll ja CoAP RESTful rakenduskihi protokoll. [10]

Põhiline energiatarbe kokkuhoid saavutatakse selleks loodud raadio töötsükli (*radio duty-cycle*) protokollide abil, millest enim levinud on ContikiMAC. Madala energiatarbega seadmete arendamise edendamiseks on Contikil sisseehitatud energiatarbe monitoorimise ja prognoosimise mehhanism. [10]

Contiki toetab mitmeid müügil olevaid riistvara platvorme ning on lihtsasti porditav. Contiki arendamine on lihtne ja kiire: programmeerimine toimub C keeles ja lähtekoodiga kaasas olevad simulaatorid teevad koodi silumise väga mugavaks. Contiki konstruktor (*build system*) teeb erinevatele platvormidele kompileerimise väga lihtsaks. Cooja simulaator võimaldab paljusid platvorme emuleerida, mis annab arendajale võimaluse rakendusi luua ilma füüsilisel kujul seadet omamata. [10]

Levinud kasutusalade näitena võib tuua tänavavalgustussüsteemid, müra- ja radiatsiooni monitoorimise süsteemid ning valvesüsteemid. [19]

2.1. Tuum

Contiki tuum sisaldab sündmuste plaanurit (*event scheduler*), mis saadab jooksvatele protsessidele sündmuseid ja kutsub perioodiliselt välja protsesside hääletuse töötlejaid (*processes polling handlers*). Kogu programmi täitmine toimub kas tuuma poolt saadetud sündmuste või hääletusmehhanismi alusel. [9][16]

Kasutusel on kahte sorti sündmused: asünkroonsed ja sünkroonsed. Asünkroonsed sündmused lisatakse tuuma poolt ootejärjekorda ja saadetakse suunatud protsessile edasi järjekorra alusel. Sünkroonsed sündmused seevastu võetakse plaanuri poolt kohesele täitmisele. Hääletusmehhanismi võib vaadata kui kõrge prioriteediga sündmuseid, mis võetakse täitmisele iga asünkroonse sündmuse vahel ja täitmine toimub prioriteetsuse järjekorras. [9][16]

Tuum kasutab kõikide protsesside täitmiseks ühte jagatud pinu (*single shared stack*). Asünkroonsete sündmuste kasutamine võimaldab vähendada pinu mahuvajadust, kuna iga sündmusetöötleja käivitamise vahel pinuviit taastatakse (*stack is rewound between each invocation of event handlers*). [9][16]

2.2. Programmeerimise mudel

Töötamaks efektiivselt piiratud mäluga süsteemidel, põhineb Contiki programmeerimismudel niinimetatud protolõimedel (*Protothreads*). *Protothread* on mäluefektiivne programmeerimise abstraktsioon, millel on nii lõimtöötluse, kui ka sündmusjuhitava programmeerimise omadused. Selle tulemusena saavutatakse madal mälu üldkulu (*memory overhead*) iga *Protothread*-i kohta. [10][9]

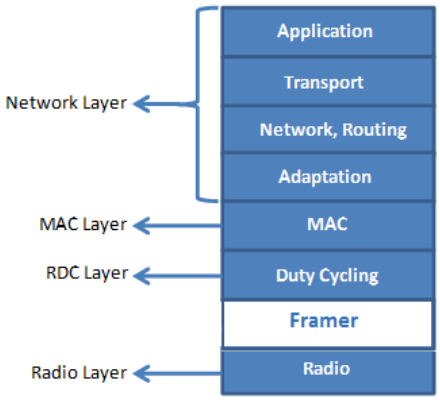
Protothread-id töötavad ühendatud plaanuriga (*cooperatively scheduled*). See tähendab, et Contiki toimingud peavad alati regulaarse intervalliga tagastama kontrolli tuumale. Tuum käivitab protsesside *Protothread*-e sisemiste või väliste sündmuste peale. Sisemisteks sündmusteks on taimerid või sõnumid teistelt protsessidelt. Välisteks sündmusteks on näiteks anduritelt või raadiosaatjalt tulevad signaalid. Protsessides on võimalik kasutada spetsiaalseid *Protothread*-e, mis blokeerivad toimingu kuni sündmuse saabumiseni ja tagastavad kontrolli tuumale sündmuste vaheliseks ajaks. [10][9]

Protothread-ide mehhanism ei määratle kindlat meetodit nende planeerimiseks ja täitmiseks, selle defineerib *Protothread*-e kasutav süsteem. *Protothread*-i tööpõhimõtteks on seda jooksutava funktsiooni korduvad väljakutsed - iga kord kui funktsiooni välja kutsutakse, jookseb *Protothread* kuni blokeeriva tingimuse või väljumiseni. Kuna *Protothread*-id ei salvesta pinu sisu blokeeriva tingimuse korral, siis lokaalsed muutujad ei säili. [10]

Protothread-id on teostatud kasutades lokaalseid järjehoidjaid (*local continuations*) - programmi jooksvaid täitmise olekuid. Lokaalseid järjehoidjaid saab funktsiooni sees seada, võimaldades hiljem jätkata funktsiooni täitmist pooleli jäänud olekust. Järjehoidja sisaldab infot käsuviida ja vajalike protsessori registriväärtuste kohta. [10]

2.3. Võrgupinu

Contikis on kasutusel tavalisest 5-kihilisest TCP/IP mudelist veidi erinev teostus. Nimelt võrgukihi ja füüsilise kihi vahel olev MAC kiht on jagatud omakorda erinevateks kihtideks (Joonis 4). [2]



Joonis 4. Contiki võrgupinu.

Radio kihi ja RDC kihi vahel on tegelikult veel ka niinimetatud *Framer* kiht, aga kuna see ei ole tüüpiline mudeli kiht, vaid pigem kogum abistavaid funktsioone, mida kasutatakse piltlikult öeldes saadetava info pakettidesse jagamiseks ja saabuvate pakettide kokku panemiseks, siis seda ei käsitleta eraldi kihina. *Radio* kiht on tegelikult lihtsalt raadiosaatja ohjur ja see sõltub kasutusel oleva raadiosaatja tüübist. Ülejäänud kihtide puhul on Contikis olemas mitu erinevat realisatsiooni ja arendajal on võimalik valida oma projekti jaoks sobivaim. [2]

2.4. Võrgukiht

Contikis on kasutusel kolm erinevat võrgukihi mehhanismi: uIP TCP/IP pinu IPv4 võrgu jaoks, uIPv6 pinu IPv6 võrgu jaoks ja Rime pinu, mis on kogum lihtsaid spetsiaalselt madala võimsusega juhtmevaba võrgu (*low-power wireless network*) jaoks loodud protokolle. [19]

Contikis kasutatav IPv6 pinu arendati välja Cisco poolt ja valmimise järel oli see väikseim *IPv6 Ready* sertifikaadiga IPv6 pinu. IPv6 pinu sisaldab RPL (*Routing Protocol for Low-power and Lossy Networks*) madalavõimsusega ja kadudega (*low-power and lossy*) IPv6 võrkude ruutmisprotokolli ning 6LoWPAN päise kokku pakkimise ja adapteerimise (*header compression and adaptation*) kihti IEEE 802.15.4 side jaoks. [19]

Rime pinu on sensorvõrkude jaoks loodud väike kihiline kommunikatsioonipinu. Rime on mõeldud kasutamiseks siis, kui IPv4 ja IPv6 pinude kasutamiseks ei ole piisavalt mälu. Rime erineb traditsioonilistest kihilistest võrguprotokollidest selle poolest, et selle kihid on väga primitiivsed. Rime põhineb suuresti hajusprogrammeerimise põhimõtetel, kus lihtsate abstraktsioonikihtide kombineerimisel saavutatakse võimas kõrgtaseme abstraktsioon. Rime kasutab põhiliselt viite kommunikatsiooni primitiivi: *single-hop unicast*, *single-hop broadcast*, *multi-hop unicast*, *network flooding* ja *address-free data collection*. Kõik ülejäänud keerukamad protokollid ja mehhanismid saadakse nende kombineerimise teel. [7][9]

Rime eesmärk on lihtsustada sensorvõrkude protokollide teostamist ja hõlbustada koodi taaskasutamist. Rime koodi vajab mälu 10 kilobaiti ja andmed kõigest mõnikümmend baiti. [7][9]

2.5. Meediumipöörduse juhtimise kiht

MAC (*Medium Access Control*) ehk meediumipöörduse juhtimise kiht asetseb RDC kihi peal ja vastutab raadiokanalil edastatavate pakettide kokkupõrgete vältimise ja kokkupõrke korral uuesti saatmise eest. MAC kiht saab saabuvad paketid RDC kihi käest ja kasutab RDC kihti pakettide saatmiseks. Kui RDC kihis tuvastatakse paketi saatmisel kokkupõrge, siis MAC kiht hoolitseb selle eest, et mõne aja möödudes proovitakse sama paketti uuesti saata. [5]

Contikis on kasutusel kaks MAC ohjurit: CSMA (*Carrier Sense Multiple Access*) ja NullMAC. CSMA on ainuke ohjur, mis kokkupõrke korral pakke uuesti saadab, sest NullMAC ohjur ei teosta üldse MAC taseme töötlust. [5]

2.6. Raadio töötsüklite kiht

Paljud Contiki süsteemid on karmide energiatarbe piirangutega. Patareitoitel töötavad juhtmevabad sensorid peavad tihti töötama aastaid ilma sekkumiseta. Selliste süsteemide puhul on tihti suurimaks energia tarbijaks raadiosaatja. Seetõttu peab madala võimsusega võrkudes raadiosaatja olema võimalikult vähe sisse lülitatud, et säästa energiat. Contikis saavutatakse see raadio töötsüklite abil. [8][5]

RDC ehk raadio töötsükli ohjur hoiab raadiosaatjat väljas niipalju kui võimalik ja kontrollib perioodiliselt, kas raadiokanalil toimub liiklus. Kui tuvastatakse liiklus, siis ohjur hoiab raadiosaatja sees, et saadetud pakett vastu võtta. Kanali kontrollimissagedus on määratud hertsides (Hz), mis näitab mitu korda sekundis kanalit kontrollitakse. Vaikimisi on kontrollimissageduseks 8 Hz. Paketti saatmist on vaja korrata seni, kuni paketti saaja lülitab oma raadiosaatja sisse ja võtab paketi vastu. See suurendab aga saatja energiakulu ja kulutab eetriaega, mis segab teiste seadmete omavahelist suhtlust. [8]

RDC ohjurid kasutavad suurema efektiivsuse saavutamiseks erinevaid meetodeid, nagu näiteks faasi optimeerimine ContikiMAC ohjuris. Igal meetodil on omad plussid ja miinused ning vastavalt võrgu kasutamise eripärale võivad ohjurid anda väga erinevaid tulemusi. Võimaldamaks arendajal enda tingimustes efektiivseim tulemus saavutada, on Contikis kasutusel mitmed erinevate omadustega RDC ohjurid - ContikiMAC, X-MAC, CX-MAC (*Compatibility X-MAC*), LPP (*Low-Power Probing*) ja NullRDC - millest levinuimaks on ContikiMAC. ContikiMAC on spetsiaalselt Contiki jaoks loodud ohjur,

mis tagab väga hea energiasäästlikkuse, aga on loodud eelkõige CC2420 802.15.4 raadiosaatjat silmas pidades. ContikiMAC võimaldab seadmetel võrgus suhelda ja samal ajal hoida raadiosaatjat väljas kuni 99% ajast. X-MAC on vanem mehhanism ja ei ole energiasäästu mõttes niivõrd efektiivne, aga samas ka mitte sedavõrd rangete ajaliste nõudmistega. CX-MAC on X-MAC-i edasiarendus, millel on veelgi vabamad ajastuspiirangud ja seetõttu töötab suuremal hulgal erinevatel raadiosaatjatel. LPP on vastuvõtja-algatuslik RDC protokoll. NullRDC ongi "null" RDC ohjur, mis hoiab raadiosaatjat kogu aeg sees ja sobib seetõttu testimiseks ja teiste RDC ohjurite võrdlemiseks. [8][5]

2.7. Jadaliini internetiprotokoll

Võimaldamaks Contiki 6loWPAN võrku ühendada teiste IP võrkudega, on Contikis kasutusel SLIP (*Serial Line Internet Protocol*). SLIP ehk jadaliini internetiprotokoll on protokoll IP käituseks kaht süsteemi ühendavatel jadaliinidel. See on IP kapseldus loodud andmete edastamiseks üle jadaportide ja modemite. SLIP on dokumenteeritud RFC 1055-s.

Personaalarvutites on SLIP suures osas asendunud uuema ja keerulisema kakspunktprotokolliga (*Point-to-Point Protocol*), kuna sellel on rohkem funktsioone. Mikrokontrollerites on aga SLIP siiani eelistatum meetod IP pakettide kapseldamiseks tänu selle väikesele lisakulule. [16]

SLIP lisab standardsele TCP/IP paketile spetsiaalse „END" baidi (Tabel 1), mille järgi saab eraldada pakette baitide voos. Tavaline END bait asendatakse kahe järjestikuse baidiga, millest esimene on ESC ja teine on ESC_END. Tavaline ESC bait asendatakse kahe järjestikuse baidiga, millest esimene on ESC ja teine on ESC_ESC. [15]

Tabel 1. SLIP protokolli paketi lõpubaidid.

Hex value	**Abbreviation**	**Description**
0xC0	END	Frame End
0xDB	ESC	Frame Escape
0xDC	ESC_END	Transposed Frame End
0xDD	ESC_ESC	Transposed Frame Escape

SLIP vajab kindlat jadapordi konfiguratsiooni: 8 andme bitti ning ilma paarsusbitita ja UART töörežiimiks riistvaras voo reguleerimine (*hardware flow control*) või kolme-juhtme-nullmodem (Joonis 7). SLIP ei sisalda veatuvastust, jättes selle kõrgema kihi protokollide hooleks, ja seetõttu ei sobi ta eraldiseisva protokollina kasutamiseks veaaltite ühenduste puhul. [15]

Contiki operatsioonisüsteemiga seadmetes on SLIP ühenduse kasutamiseks ohjur, mis võimaldab luua ühendust üle UART liidese. Arvuti poolel ühenduse loomiseks on Contiki lähtekoodiga kaasas programm *tunslip6*, mis tekitab SLIP tunneli füüsilise jadapordi ja virtuaalse võrguliidese (*tun*) vahele, võimaldades ühendada seadmetes kasutatav 6loWPAN võrk ja arvutis loodud virtuaalne Ethernet võrk. Tulemusena on võimalik arvutist pääseda ligi kõikidele 6loWPAN võrgus olevatele seadmetele, kasutades selleks nende IP aadresse. Juhised tunslip6 programmi kasutamiseks leiab Lisa 6-s.

3. Contiki arenduskeskkonna seadistamine ja kasutamine

Kuna Contiki arendustööriistad töötavad Linuxi keskkonnas, siis on arenduskeskkonda kõige lihtsam üles seada Linuxi tööjaamale. Alustamiseks pole tarvis teha muud, kui installeerida arendustööriistad ja laadida alla lähtekood. Täpsemad juhised Linuxi keskkonnas arendustööriistade installeerimiseks ja programmikoodi alla laadimiseks leiab Lisa 2-s.

Windowsi tööjaama kasutajatel on lihtsaim viis kasutada Linuxi virtuaalmasinat. Contiki arendajad pakuvad oma kodulehelt tasuta alla laadimiseks eelseadistatud Ubuntu Linuxi virtuaalmasinat „InstantContiki“. See teeb kasutaja jaoks alustamise väga lihtsaks, kuna sinna on juba eelnevalt installeeritud kõik vajalik: arendustööriistad, kompilaatorid, simulaatorid ja operatsioonisüsteemi lähtekool. InstantContiki kasutamine on nii mugav, et isegi kirglikud Contiki arendajad kasutavad seda. Täpsemad juhised Windows keskkonnas InstantContiki virtuaalmasina üles seadmiseks leiab Lisa 3-s.

Alternatiivina on Windowsi keskkonnas võimalik kasutada Cygwin keskkonnamoodulit, aga selle kasutamist käesolevas töös ei käsitleta.

3.1. Contiki lähtekoodi struktuur

Contiki lähtekoodi kaust koosneb järgmistest alamkataloogidest:

- apps/ – arhitektuurist sõltuvad rakendused
- core/ – süsteemikood, alamkataloogid erinevate süsteemiosade jaoks
    - cfs - failisüsteemid
    - sys
    - lib
    - dev - ühised ohjurid (liidesed)
    - net - võrguprotokollipinud
- cpu/ – CPU-spetsiifiline kood, üks alamkataloog CPU kohta
- doc/ – üldine dokumentatsiooni kood
- examples/ – näidisrakendused
- platform/ – platvormispetsiifiline kood, üks alamkataloog platvormi kohta
- tools/ – tööriistad konstrueerimiseks, simuleerimiseks, testimiseks

3.2. Contiki *Makefile* konstruktor

Contikis on kasutusel keerulise struktuuriga aga mugav konstruktor (*build system*), mis koosneb mitmest *Makefile*-ist. See teeb erinevate platvormide jaoks kompileerimise väga lihtsaks. Kasutaja ei pea vaeva nägema keeruliste *Makefile*-ide muutmise pärast, olemasolevaid faile on võimalik kasutada erinevate projektide jaoks. [5]

Contiki konstruktor koosneb järgmistest *Makefile*-idest:

- **Makefile**: projekti *Makefile*, asub projekti kaustas
- **Makefile.include**: Contiki operatsioonisüsteemi *Makefile*, asub Contiki koodi juurkataloogis
- **Makefile.$(TARGET)** (kus $(TARGET) on platvorm, mille jaoks rakendust kompileeritakse): platvormispetsiifiliste juhistega *Makefile*, asub platvormi kataloogis platform/$(TARGET)
- **Makefile.$(CPU)** (kus $(CPU) on platvormil kasutusel oleva protsessori või mikrokontrolleri arhitektuur): protsessoriarhitektuurispetsiifiliste juhistega *Makefile*, asub protsessoriarhitektuuri kataloogis cpu/$(CPU)
- **Makefile.$(APP)** (kus $(APP) on app/ kataloogis asuva rakenduse nimi): rakendusespetsiifiliste juhistega *Makefile*.

Projekti *Makefile* on sihilikult tehtud väga lihtsaks. Esmalt antakse selles projektile nimi, kirjutades see muutujasse CONTIKI_PROJECT. Seejärel määratakse Contiki lähtekoodi asukoht projekti kataloogi suhtes (*relative path*), kirjutades see muutujasse CONTIKI. Viimasena kaasatakse (*include*) Contiki operatsioonisüsteemi *Makefile*, kasutades selleks käsku „include $(CONTIKI)/Makefile.include“. Lisaks on projekti *Makefile*-is võimalik APPS muutuja abil projekti kaasata apps/ kataloogis asuvaid rakendusi. Igal rakendusel on samuti oma *Makefile*. [5]

Makefile.include asub alati Contiki koodipuu juurkataloogis ja see sisaldab kõikide Contiki süsteemikoodi C failide definitsioone. Lisaks kaasab Makefile.include platvormispetsiifilise Makefile.$(TARGET)-i ja kõikide projekti *Makefile*-is lisatud rakenduste Makefile.$(APP)-id. [5]

Makefile.$(TARGET), mis asub platvorm/$(TARGET) kataloogis, sisaldab nimekirja kõikidest C failidest, mille platvorm lisab Contiki süsteemile. Nimekiri kirjutatakse

muutujasse CONTIKI_TARGET_SOURCEFILES. Lisaks sellele määratakse ära platvormil kasutatavad süsteemi moodulid, kirjutades nimekiri muutujasse MODULES. Lõpuks kaasab Makefile.$(TARGET), sõltuvalt platvormil kasutatava protsessori või mikrokontrolleri arhitektuurist, cpu/$(CPU)/ kataloogis asuva protsessoriarhitektuurispetsiifilise Makefile.$(CPU)-i. [5]

Makefile.$(CPU) sisaldab üldjuhul selle protsessori jaoks kasutatava C kompilaatori definitsioone. Sõltuvalt protsessorist kasutatakse erinevaid kompilaatoreid. Kui protsessori jaoks on kasutusel mitu erinevat kompilaatorit, siis Makefile.$(CPU) sisaldab mõlema kompilaatori definitsioone, või need defineeritakse Makefile.$(TARGET)-is, lükates sellega eelnevas defineeritu ümber. [5]

3.3. Kompileerimine

Contiki kompileeritakse nagu *Makefile* süsteemi puhul ikka „*make*“ käsuga, millele saab lisada ka parameetreid. Üheks tähtsamaks parameetriks on TARGET (lisatakse kujul TARGET=“platvorm“), millega määratakse ära kompileerimise sihtplatvorm. Töös kasutatavale AVR-Atmega128RFA1 seadmele kompileerimine toimub käsuga „make TARGET=avr-atmega128rfa1“. Kui TARGET parameeter jätta määramata, siis vaikimisi kompileeritakse kood *native* ehk arenduskeskkonna platvormile. [5]

Selleks, et iga kord ei peaks kompileerimisel TARGET parameetrit välja kirjutama, on võimalik käsuga „make TARGET=AVR-ATmega128RFA1 savetarget“ määrata uus vaikimisi sihtplatvorm. Sellega luuakse projekti kataloogi uus fail nimega Makefile.target, kuhu salvestatakse sihtplatvormi nimi, määrates see nii uueks vaikimisi sihtplatvormiks antud projekti jaoks. [5]

Teine oluline parameeter on DEFINES, mis võimaldab defineerida suvalisi muutujaid C eelprotsessori jaoks. Muutujad esitatakse komaga eraldatud loendina (nt. DEFINES=MYTRACE,MYVALUE=4711). Sarnaselt TARGET-ile on savedefines käsuga võimalik projekti jaoks DEFINES-e salvestada, vältimaks vajadust igal kompileerimisel neid välja kirjutada. [5]

Kompileerimise tulemusena luuakse projekti kataloogi uus fail laiendiga *. $(TARGET) (*.avr-atmega128rfa1, kui TARGET=AVR-ATmega128RFA1). [5]

3.4. Seadmele laadimine

Kompileeritud rakenduse seadmele laadimiseks on nii Linuxi kui ka Windowsi keskkonnas olemas spetsiaalsed tööriistad. Linuxi keskkonnas on selle jaoks käsureatööriist “avrdude” ja Windowsi keskkonnas programm AVR Studio 4. Täpsemad juhised nende kasutamise kohta leiab Lisa 3-s (avrdude) ja Lisa 4-s (AVRStudio 4).

Mugavuse huvides on võimalik rakendust seadmele laadida ka *Makefile* konstruktorit kasutades. Selleks on lisatud protsessorispetsiifilisse Makefile.$(CPU)-sse üleslaadimise tööriista käsud, AVR seadmete puhul on tööriistaks eelpool mainitud avrdude. Avrdude parameetrid on defineeritud platvormi *Makefile*-is (*Makefile.avr-atmega128rfa1*). Rakendust saab seadmele laadida lisades kompileerimise käsus kompileeritava faili nimele .u (*upload*) laiend. Näiteks *hello-world* programmi laadimiseks tuleb sisestada käsk „make hello-world.u“. Selle tulemusena kõigepealt programm kompileeritakse ja seejärel laetakse seadmele, eeldades loomulikult et see on programmaatoriga ühendatud.

4. Probleemid ja lahendused

Järgnev peatükk kirjeldab lähemalt kõiki probleeme, mis kompileerimisel ja testimisel tekkisid. Lahenduse leidnud probleemide puhul on kirjeldatud ka võimalikke lahendusi. Koodi alla laadida ja kõiki muudatusi täpsemalt näha saab spetsiaalselt selleks loodud Git hoidlas (*repository*) aadressil *https://github.com/avainola/contiki-iasb/*. Samuti on kood kättesaadav juhendajalt.

4.1. Lähtekoodiga tutvumine

Contiki koodi tundmaõppimiseks sai alustatud konfiguratsioonifaili uurimisest ja koodi kompileerimisest. Platvormi konfiguratsioonifail "contiki-conf.h" (asub kaustas contiki/platvorm/avr-atmega128rfa1/) sisaldab kõiki eelprotsessori definitsioone (*#define*) ja makrosid AVR-ATmega128RFA1 platvormi jaoks. Seal määratakse ära kõik platvormiga seonduv, alates nimest ja tüübist kuni kasutatavate ohjuriteni.

Juba vaikimisi seadetega kompileerimisel ootas ees ebameeldiv üllatus, kui linkur andis veateate. Õnneks ei olnud lahenduse leidmine väga keeruline. Platvormi moodulite nimekirjast oli puudu *nullsec_driver* moodul, mis on ebaturvaline lülikihi turvaohjur (*insecure link layer security driver*). Pärast selle lisamist (Makefile.avr-atmega128rfa1 failis defineeritud muutujale MODULES oli vaja lisada "core/net/llsec") õnnestus kompileerimine ja linkimine.

Sarnane viga tekkis ka CX-MAC RDC ohjuriga kompileerides, kuna platvormi moodulite nimekirjast oli puudu ka *cxmac_driveri* moodul ("core/net/mac/cxmac").

4.2. USB ohjuri lisamine

Programmi silumist ilma ekraanile printimiseta on väga raske ette kujutada ja seetõttu oli järgmiseks eesmärgiks teksti ekraanile printimise võimaldamine. Kuna kasutataval seadmel on olemas USB liides, siis tundus mõistlik suunata standard väljund (*stdout - printf*) USB liidesele ja arvutis käivitada terminaliprogramm, mis kuvab sõnumid ekraanile.

Kuna Contiki lähtekoodis puudus ohjur deRFnode arendusplaadi USB liidese juhtimiseks, siis oli vaja see lisada. Dresden Elektroniku poolt pakutavate

näidisrakenduste abil sai suhteliselt väikse vaevaga kirjutatud uus ohjur USB liidese juhtimiseks (*usb-derfnode*) ja lisatud see platvormi kausta. Nüüd oli vaja lisada see platvormi poolt Contiki süsteemile lisatavate C failide nimekirja, mis on defineeritud failis Makefile.avr-atmega128rfa1 muutujaga CONTIKI_TARGET_SOURCEFILES. Välistamaks vajadust igas rakenduses eraldi seda teha, sai ohjuri initsialiseerimine lisatud platvormi contiki-main.c faili.

Tulemuse testimiseks sobis väga hästi vana tuttav *"Hello, world"*. Seetõttu saigi lähtekoodis olemasoleva *hello-world* näite alusel kirjutatud lihtne programm (asub kaustas contiki/IASB/hello-world), mis trükib iga 10 sekundi möödudes ekraanile *"Hello, world!"*.

Testimise tulemusena selgus, et kui USB oli ühendatud, siis töötas kõik planeeritult, aga kui USB kaabel eemaldada, siis lülitas seade ennast mõne hetke pärast välja. Läbiviidud testide analüüsimise põhjal võis arvata, et probleem on tõenäoliselt seotud USB liidese jaoks kasutatava puhvri täitumisega. Kuna töö eesmärgiks ei olnud seda probleemi väga süvitsi uurida ja aega selle jaoks nappis, siis sai lisatud lihtne moodus kompileerimisel määrata, kas liidest soovitakse kasutada või mitte (*stdout* suunatakse USB-le ainult siis, kui kompileerimisel defineeritakse DEFINES=LOGTOUSB). Sellest hoolimata vajaks tekkinud probleem kindlasti põhjalikumat uurimist.

4.3. Andurite ohjurite lisamine

Kuna kasutataval deRFnode arendusplaadil on sisse ehitatud kaks surunuppu, 3 valgusdioodi ning temperatuuri-, valgusheleduse- ja kiirenduse andurid, siis järgmiseks eesmärgiks oligi nende tööle seadistamine. Taaskord olid abiks Dresden Elektroniku kodulehelt alla laetavad näidisrakendused, mille hulgas olid ka nuppude ja andurite kasutamise näited. Nende põhjal sai platvormi kausta lisatud uued failid: io_access (nupud ja LEDid), i2c_sensors (sensorid) ja twi_master (TWI kasutamise baasfunksionaalsus). Sarnaselt USB ohjurile, oli need loomulikult vaja lisada ka platvormi C failide nimekirja (failis Makefile.avr-atmega128rfa1-s olevale muutujale CONTIKI_TARGET_SOURCEFILES).

Testimiseks sai taaskord tehtud lihtne programm (asub kaustas /contiki/IASB/sensors), mis loeb iga 15 sekundi järel andurite väärtused ja prindib need välja. Energia kokkuhoiu mõttes oli mõistlik vahepeal andurid välja lülitada.

Testimise käigus selgus, et valgusheleduse- ja kiirenduse andurite mõõtmistulemused olid alati nullid. Pärast andurite spetsifikatsiooni uurimist ja mõningast testimist olid põhjused selged ja võimalikud lahendused välja mõeldud.

Valgusheleduse anduri puhul pidi anduri iseloomu tõttu mõõtmise alustamise ja väärtuse lugemise vahel olema vähemalt 100 millisekundit. Probleemi lahendamiseks sai proovitud kahte erinevat meetodit. Esimene variant on hoida andur pidevalt sees, kuid see oleks energiakulu mõttes raiskamine. Teine ja parem meetod on lisada mõõtmise alustamise ja väärtuse lugemise vahele taimer. Kuna eesmärke arvesse võttes tundus energia kokkuhoid mõõtmise ajahetke täpsusest olulisem, siis saigi kasutatud teist varianti.

Kiirenduse anduri puhul selgus aga spetsifikatsioonist, et peale selle sisse lülitamist on alati esimene lugemine ebatäpne. Lahendatud sai probleem nii, et igal mõõtmisel loetakse tulemust kaks korda.

Seejärel tundus kõik töötavat, ent põhjaliku testimise tulemusena selgus, et andurite töötamine sõltub ka kasutatavast RDC ohjurist. Kui ülejäänud ohjurite korral probleeme ei ole, siis ContikiMAC RDC ohjuri korral jääb seade anduri sisse lülitamisel *twi_master* ohjuris lõpmatusse kordusahelasse (*while loop*) kinni (faili *twi_master.c* rida 83 “while (!(TWCR & (1 << TWINT)));”). Tõenäoliselt on probleemi põhjuseks see, et mõlemad ohjurid kasutavad TWI liidest (Two Wire Interface) ja segavad üksteist.

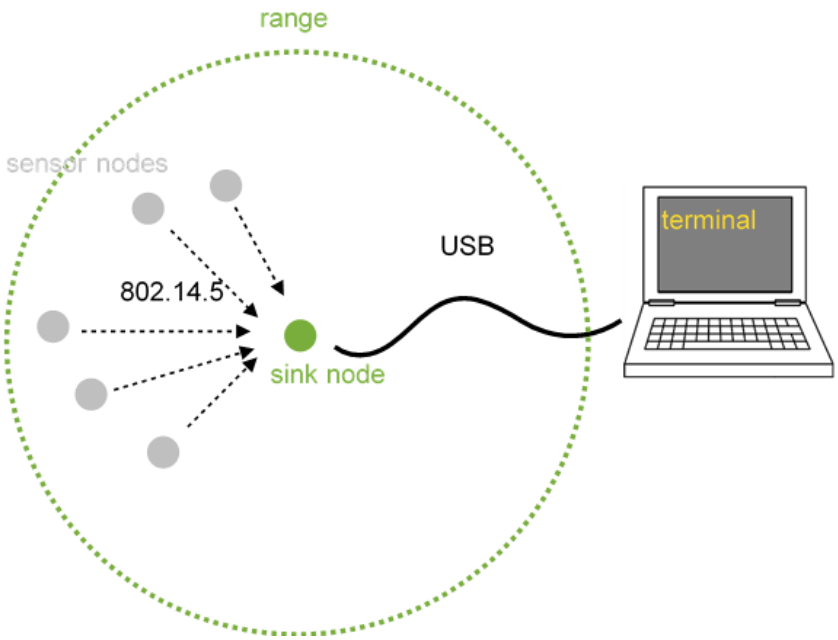
See on kindlasti probleem, mida peaks põhjalikumalt edasi uurima, aga ajalistel kaalutlustel ei olnud võimalik täpsemaid põhjusi selle töö käigus välja selgitada.

4.4. Tähtvõrgu loomine kasutades Rime pinu

Järgmiseks eesmärgiks oli tähekujulise topoloogiaga võrgu loomine. Kuna ühtegi piirangut ei olnud, siis sai võrgukihiks valitud Rime.

Selle jaoks sai kirjutatud kaks eraldi programmi (asuvad kaustas contiki/IASB/rime): üks andmeid koguva ja arvutiga ühendatud seadme jaoks ning teine juhtmevabade

sensorseadmete jaoks (Joonis 5). Joonisel on tähistatud andmeid koguv seade rohelise punktiga (*sink node*) ja juhtmevabad sensorseadmed hallide punktidega (*sensor nodes*). Rohelise punktiirjoonega on tähistatud andmeid koguva seadme raadiosaatja leviala ja musta pidevjoonega USB ühendus arvutiga. Arvutis jookseb terminaliprogramm saabuvate andmete kuvamiseks.



Joonis 5. Tähtvõrk.

Võrgu suhtlusloogika sai üles ehitatud nii, et andmeid koguv seade annab *broadcast* sõnumi abil enda olemasolust teada iga 30 sekundi järel, samal ajal kuulates võrku, tuvastamaks sissetulevaid *unicast* sõnumeid. Juhtmevabad sensorseadmed kuulavad aga võrku, et tuvastata läheduses oleva andmeid koguva seadme poolt välja saadetud *broadcast* sõnumeid. Kui sõnum tuvastatakse, siis salvestatakse selle saatja aadress ja hakatakse sellele iga 17 sekundi järel oma andurite mõõtetulemusi edastama.

Programmides on kasutatud Rime-i *broadcast* ja *unicast* primitiive. Mõlemas programmis jookseb kaks *Protothread*-i: üks *broadcast*-i ja teine *unicast*-i jaoks. Sensorseadmetes on andurite andmete lugemiseks kasutatud eelnevalt andurite testimiseks kirjutatud koodi. Andmeid koguva seadme programmis on andmete ekraanile printimise jaoks kirjutatud eraldi funktsioon.

Esimesel testimisel selgus, et *broadcast* sõnumid jõudsid kõikide seadmeteni, aga *unicast* sõnumitega oli probleem. Siinkohal sai abiks võetud deRFusb seade ja *BitCatcher*

programm, millega oli näha, et sensorseade saadab küll *unicast* sõnumi välja, aga andmeid koguv seade seda kätte ei saa.

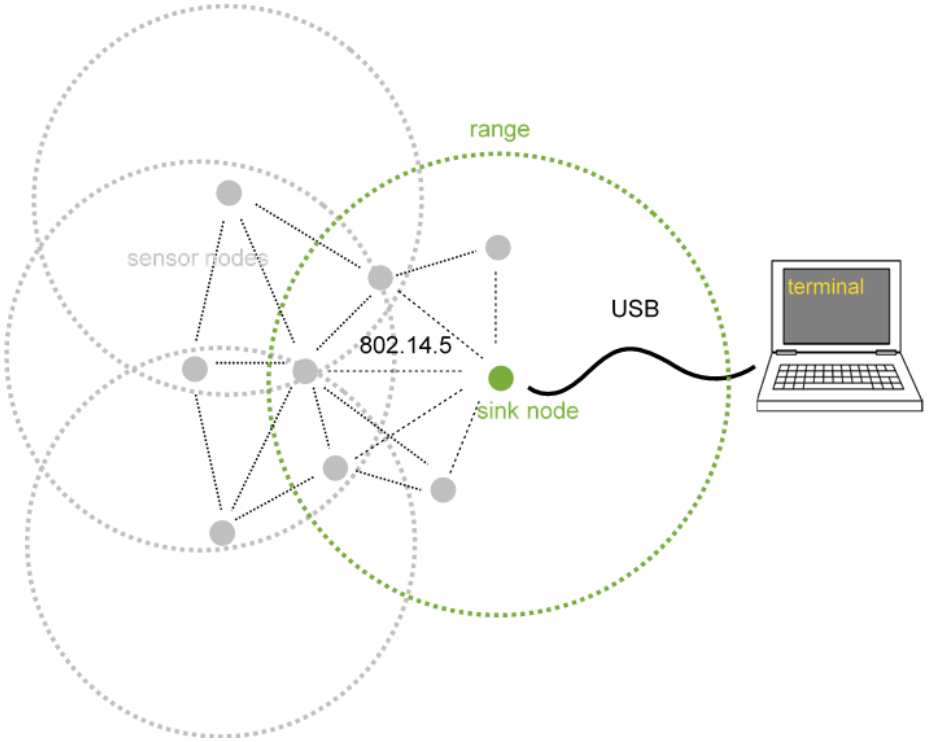
Põhjuse välja selgitamiseks sai koodi võrgupinu (*network stack*) kõikidesse kihtidesse lisatud silumissõnumeid (*debug messages*), aga tulemust see ei andud. Andmeid koguvas seadmes ei jõudnud need sõnumid isegi madalal asuva RDC kihini. Selle põhjal on alust arvata, et probleem võis olla pakettide adresseerimises, kuna pakette ignoreeriti juba raadiosaatja ohjuri poolt.

Pärast põhjalikku otsimist õnnestus Contiki arendajate kogukonna diskussioonirühmast (*mailing list*) leida üks kirjavahetus, kus oldi kokku puututud sarnase probleemiga. Selgus, et Contikis kasutatakse raadiosaatjat nii öelda laiendatud režiimil (*extended mode*), mis aga AVR-ATmegaRFA1 seadme raadiosaatjaga võib probleeme tekitada. Seega sai platvormi konfiguratsioonifailis contiki-conf.h laiendatud režiimi seadistus maha võetud (RF230_CONF_AUTOACK = 0) ja tulemusena jõudsid ka *unicast* sõnumid kohale. Sellega ei saa küll probleemi lahendatuks pidada, aga selle seadistusega sai seadmeid testitud erinevate ohjuritega (väljaarvatud ContikiMAC ohjuriga eelmises peatükis kirjeldatud probleemi tõttu) ja sõnumite liikumisega probleeme ei esinenud.

Ilmselt tuleks selle probleemi täpseid põhjusi hakata uurima raadiosaatja ohjurist ja sõnumite adresseerimisest, aga lähtuvalt eesmärkidest ei jõutud käesoleva töö raames sellega tegeleda.

4.5. Silmusvõrgu loomine kasutades uIPv6 pinu

Kuna eelmise võrgu piiranguks on see, et kõik sensorseadmed peavad andmeid koguva seadme raadiosaatja levialas olema, siis järgmiseks eesmärgiks oli luua silmusvõrk (*mesh network*) (Joonis 6). Joonisel on tähistatud andmeid koguv seade rohelise punktiga (*sink node*) ja juhtmevabad sensorseadmed hallide punktidega (*sensor nodes*). Ringikujulised punktiirjooned tähistavad seadmete raadiosaatjate leviala ja mustad sirged punktiirjooned tähistavad juhtmevabas võrgus olevate seadmete omavahelisi seoseid. Musta pidevjoonega on tähistatud andmeid koguva seadme USB ühendus arvutiga. Arvutis jookseb terminaliprogramm saabuvate andmete kuvamiseks.



Joonis 6. Silmusvõrk.

Seekord sai võrgukihiks valitud uIPv6 pinu, kuna see sisaldab RPL ruutimisprotokolli, mis peaks silmusvõrgu loomise väga lihtsaks tegema. Võrgu loomiseks oli taaskord tarvis ühte programmi andmeid koguva seadme (*sink node*) jaoks ja teist programmi andmeid edastavate seadmete (*sensor nodes*) jaoks. Contiki lähtekoodis oli selle ülesande lahenduseks väga hea näide (asub contiki/examples/ipv6/rpl-udp), kus kasutatakse andmete edastamiseks UDP protokolli. Aja kokkuhoiu huvides saigi väikeste muudatustega seda kasutatud.

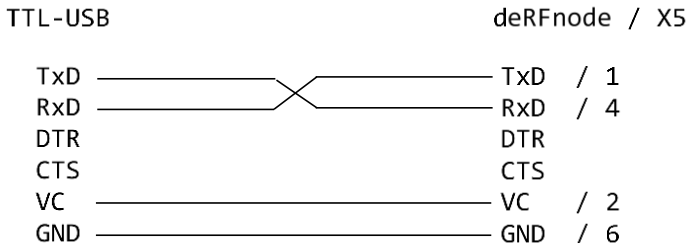
Andmeid koguva seadme jaoks on programm udp-server.c ja andmeid edastava seadme jaoks on programm udp-client.c (asuvad kataloogis contiki/IASB/rpl). Programmide põhimõte on sama, mis eelmises peatükis, et sensorseadmed saadavad oma info andmeid koguvale seadmele ja see prindib üle USB liidese need arvutis jooksva terminali ekraanile. Andurite lugemiseks ja andmete välja printimiseks sai kasutatud endiselt samu funktsioone, mis eelmistes peatükkides.

Testitud sai erinevate ohjuritega (väljaarvatud ContikiMAC) ja testimise tulemused olid head, andmete edastamisel probleeme ei esinenud. Tähelepanu väärib see, et IPv6 stacki kasutades töötas ka Rime pinu kasutamisel probleeme tekitanud laiendatud režiim

(*extended mode*). Seega on taaskord põhjust arvata, et tegu võib olla Rime pinu adresseerimise probleemiga.

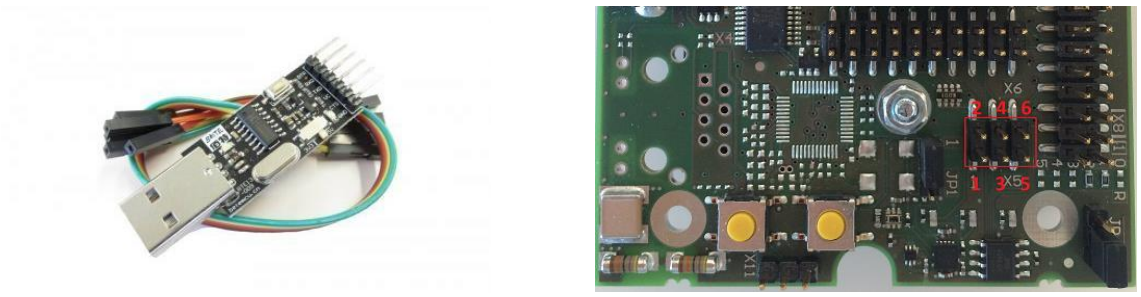
4.6. 6loWPAN ja Ethernet võrgu ühendamine

Viimaseks eesmärgiks oli luua ühendus 6loWPAN (IEEE 802.15.4) ja Ethernet (IEEE 802.3) võrgu vahel. Selleks on vaja luua seadme ja arvuti vahel SLIP ühendus. Kuna Contiki SLIP ohjurid võimaldavad luua ühendust üle UART0 ja UART1 liideste, aga kasutatava arendusplaadi USB liides ei kasuta andmete edastamiseks mikrokontrolleri UART liideseid, siis üle arendusplaadi USB liidese SLIP ühenduse loomiseks oleks pidanud kirjutama uue ohjuri. Kuna selleks aega ei olnud, siis sai SLIP ühenduse loomiseks kasutatud mikrokontrolleri UART0 liidest ja TTL-USB konverterit (Joonis 8) kolme-juhtme nullmoodemi ühendusega (Joonis 7).



Joonis 7. Kolme-juhtme-nullmoodemi ühandamine.

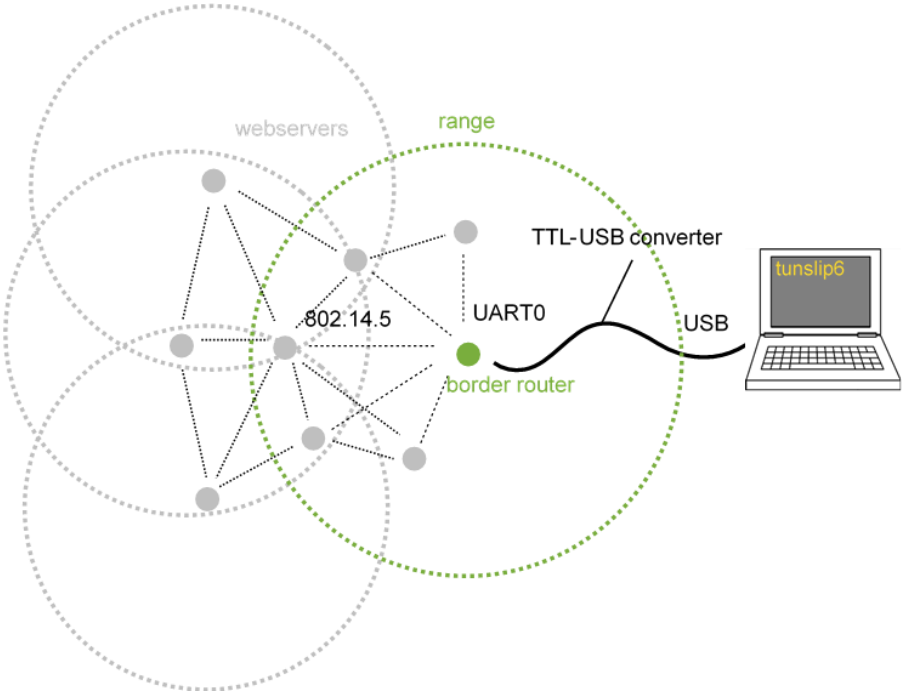
Selleks on vaja konverteri TxD ja RxD viigud ühendada AVR-Atmega128RFA1 mikrokontrolleri UART0 liidese RxD0 ja TxD0 viikudega ning konverteri GND viik arendusplaadi GND viiguga. Arendusplaadil on need kättesaadavad X5 liidese (Joonis 8) viikude 1 (TxD0) ja 4 (RxD0) ja 6 (GND) kaudu. Lisaks on võimalik liidese kaudu ka seadmele toidevoolu anda ühendades konverteri VC viigu X5 liidese 2 viiguga.



Joonis 8. TTL-USB konverter (vasakul) ja deRFnode arendusplaadi X5 liides nummerdatud viikudega (paremal).

Võrgu loomiseks sai kasutatud Contiki lähtekoodi näiteid: ruuterseadme jaoks contiki/examples/ipv6/rpl-border-router ja juhtmevabade sensorseadmete jaoks contiki/examples/webserver-ipv6-raven. Viimane on väga primitiivne veebiserver, mis kuvab infot mikrokontrolleri protsesside, võrgu, sisseehitatud sensorite jms kohta. Juhtmevabadesse seadmetesse oleks võinud valida ka mõne teise programmi, aga testimise lihtsuse huvides sai otsustatud selle kasuks. Näidisprogrammide kaustad sai muutmiseks kopeeritud projekti kausta contiki/IASB/slip.

Loodav võrk peaks võimaldama arvutist pääseda ligi kõikides juhtmevabades seadmetes jooksvatele veebiserveritele (Joonis 9). Joonisel on tähistatud ruuterseade rohelise punktiga (*border-router*) ja juhtmevabad sensorseadmed hallide punktidega (*webservers*). Ringikujulised punktiirjooned tähistavad seadmete raadiosaatjate leviala ja mustad sirged punktiirjooned tähistavad juhtmevabas võrgus olevate seadmete omavahelisi seoseid. Musta pidevjoonega on tähistatud TTL-USB konverteri ühendus ruuterseadme ja arvuti vahel. Arvutis jookseb tunslip6 programm SLIP ühenduse loomiseks.



Joonis 9. 6loWPAN ja Ethernet võrgu ühendamine.

Kui *border-router*-i programm kompileerus esimesel katsel, siis veebiserveri koodi kompileeruma saamiseks oli tarvis mõned muudatused teha. Kuna kasutatav veebiserveri programm oli loodud eelkõige AVR-Raven platvormile, millel on erinevalt deRFnode

seadmetest LCD ekraan, siis esimese asjana tuli veebiserveri programmi (asub kaustas contiki/IASB/slip/webserver-ipv6-raven) failis Makefile.webserver muutujast APPS eemaldada väärtus „raven-lcd-interface“. Lisaks oli vaja teha parandusi ka ühes veebiserveri rakenduse failis, kuna süsteemikoodis tehtud muudatused ei olnud selles kajastatud. Tehtud parandused on näha eelpool mainitud Git hoidlas.

Kui kompileeritud programmid olid seadmetele laetud ja ruuterseade seade TTL-USB konverteri abil arvuti USB liidesega ühendatud, siis sai arvutis tunslip6 programm käivitatud. Juhised tunslip6 programmi käivitamiseks leiab Lisa 6-s.

Testimisel ilmnes aga järgmine probleem. Võrgu loomine õnnestus (kõiki võrgus olevaid seadmeid oli võimalik *ping*-ida), aga ühegi veebiserveri lehekülg veebilehitsejas ei avanenud. Uurimise käigus selgus, et ruuterit õnnestus *ping*-ida vaid kuni 92 baidi suuruste pakettidega ja seadmeid kuni 72 baidi suuruste pakettidega. Selle põhjal võis järeldada, et probleem oli uIP puhvri suuruses. Nimelt *border-router*-i programmi konfiguratsioonifailis (contiki/IASB/slip/rpl-border-router/project-conf.h) oli defineeritud UIP_CONF_BUFFER_SIZE = 140 (päise lisakulu (*overhead*) on 48 baiti, seega 140 – 48 = 92 baiti) ja veebiserveri programmi failides polnud seda üldse defineeritud, mis tähendas et kasutati Contiki vaikimisi väärtust 120 (seega 120 – 48 = 72 baiti).

Probleemi lahendamiseks sai *border-routeri* programmi konfiguratsioonifailis muutuja UIP_CONF_BUFFER_SIZE väärtuseks defineeritud 1280. Valikul lähtuti sellest, et väärtus oleks võimalikult suur, mis veel kasutada olevasse muutmälusse ära mahuks, aga mitte suurem, kui protokolli maksimaalne paki suurus (*MTU - Maximum transmission unit* , IPv6 = 1280 baiti). Kuna veebiserveri programmil ei olnud eraldi konfiguratsioonifaili, siis sai see lisatud (contiki/IASB/slip/webserver-ipv6-raven/project-conf.h) ja seal samuti defineeritud UIP_CONF_BUFFER_SIZE väärtuseks 1280. Nüüd oli vaja veel veebiserveri programmi faili Makefile.webserver lisada info konfiguratsioonifaili olemasolu kohta (CFLAGS += -DPROJECT_CONF_H = \"project-conf.h\").

Selle tulemusel õnnestus ruuterseadet *ping*-ida kuni 1232 baidi suuruste pakkidega (48 baiti on päiste *overhead*) ja avanesid ka veebiserverite leheküljed. Kuna lehekülgede avamine võttis ligikaudu 5-10 sekundit aega, siis sai testitud ühenduse kiirust erinevate

RDC ohjuritega. Ühegi ohjuriga märgatavalt kiiremini leheküljed avanema ei hakanud, aga nagu ka eeldada võis, andis parimaid tulemusi nullMAC ohjur (tulemused on näha Lisa 7-s).

Veebiserveri faile uurides selgus, et lehe kujundamiseks kasutatud *style.css* faili suurus on 2030 baiti. Kuna iga lehe avamisega kaasnes ka *style.css* alla laadimine, siis oli põhjust arvata, et lehtede aeglane avanemine võis olla tingitud just sellest. Seetõttu sai *index.html* faili päises laaditabeli (*stylesheet*) kasutamine välja võetud. Selle tulemusena muutus olukord märgatavalt ja lehed avanesid keskmiselt 1 sekundiga.

Kuna andmete edastuse kiirus võib sõltuda erinevatest Contiki seadistustest, siis see on kindlasti valdkond, mida võiks põhjalikumalt uurida, et leida parim kompromiss energia säästlikkuse ja andmeedastuskiiruse vahel.

4.7. Soovitused edasiseks uurimiseks

Töö käigus tekkis mitmeid probleeme ja lisavõimalusi, andes suurepärase võimaluse teema uuringutega jätkamiseks.

Alustuseks võiks kindlasti uurida kõiki käesolevas töös esile tõusnud probleeme. Arendusplaadi USB liidese kasutamise probleemiga seoses võiks uurida põhjalikumalt, kuidas toimub andmete puhverdamine, ning mis siis juhtub, kui puhver täis saab. Andurite kasutamise probleemi kohapealt oleks vaja uurida, miks *twi_master* ohjur ContikiMAC RDC ohjurit kasutades kirjeldatud tsüklisse kinni jääb. Ilmselt peitub probleem TWI liidese kasutamises. Lisaks eelnevale vajaks uurimist ka laiendatud režiimis Rime pinu kasutamise probleem, mille põhjus on suure tõenäosusega seotud seadmete adresseerimisega.

Lisaks kirjeldatud probleemidele on olulisteks uurimisvaldkondadeks energiatarbimise minimeerimine ja andmeedastuskiiruse maksimeerimine. See võimaldaks kasutada seadmeid efektiivsemalt ja laiendada kasutusalade hulka.

Konkreetsematest rakendustest tasuks kindlasti uurida CoAP protokolli kasutades RESTful võrguteenuse loomist. See annaks väga hea võimaluse Interneti vahendusel seadmetega suhtlemiseks, koormates seejuures palju vähem võrguliiklust kui veebiserverid.

Kokkuvõte

Käesoleva töö käigus õpiti tundma Contiki operatsioonisüsteemi ja selle erinevaid kasutusvõimalusi Tallinna Tehnikaülikooli Arvutitehnika Instituudis kasutusel olevatel deRFmega128 raadiomooduli ja deRFnode arendusplaadi komplektidel.

Töö eesmärgiks oli Contiki operatsioonisüsteemi laiendamine, võimaldamaks kasutada deRFnode arendusplaadi USB liidest ja erinevaid sensoreid, ning erineva topoloogiaga võrkude loomine ja nende Internetiga ühendamine.

Kuna töö autoril puudus varasem kokkupuude nii kasutatavate seadmete, kui ka Contiki operatsioonisüsteemiga, siis oli suureks osaks tööst teoreetiliste alusteadmiste kogumine. Autor uuris töö käigus põhjalikult kasutatava riistvara iseärasusi ning Contiki operatsioonisüsteemi ülesehitust ja tööpõhimõtteid. Eesmärkide saavutamiseks ja tulemuste testimiseks kirjutati C programmeerimiskeeles neli ohjurit ja kaheks Contiki rakendust.

Kui üksikud väiksemad probleemid kõrvale jätta, siis said kõik püstitatud eesmärgid täidetud. Töö tulemusena õnnestus deRFmega128 raadiomooduli ja deRFnode arendusplaadi komplektil Contiki operatsioonisüsteemiga kasutada arendusplaadi USB liidest ning kõiki andureid, luua seadmete vahele nii tähe- kui ka silmustopoloogiaga võrk ja ühendada loodud võrk jadaliidese abil arvuti Ethernet võrguga. Lisaks sai koostatud juhend Contiki arenduskeskkonna seadistamiseks ja rakenduste arendamiseks kasutatud platvormile.

Töös kasutatud AVR-Atmega128RFA1 mikrokontroller on küll Contiki toetatud platvormide nimekirjas, aga arendamisega alustamine osutus arvatust tunduvalt keerulisemaks. Peamisteks probleemideks võib tuua ilmselt väga kehva dokumentatsiooni ja liiga üldise koodi, mis on kirjutatud liiga paljude erinevate platvormide jaoks. Ilmselt just see, et tahetakse sama koodi kasutades toetada võimalikult palju platvorme, on põhjuseks, miks kood on väga raskesti loetav ja paljude seadistustega ei kompileeru. Nagu töö käigus selgus, on siiski piisavate alusteadmiste korral võimalik Contikit kasutada erinevateks rakendusteks, mis koos Contiki heade omadustega võib anda projektile vajamineva tõuke.

Töö teostamisel tekkisid mitmeid probleemid ja lisavõimalused, mis antud töö eesmärkidest lähtuvalt jäid lahtiseks. See annab suurpärase võimaluse tööd jätkata ning uurida võimalikke lahendusi optimaalsema energiatarbe ja parema andmeedastuskiiruse saavutamiseks.

Kasutatud kirjandus

[1] Adapter board with deRFmega128 22T00 | 22T02: Datasheet. / Dresden Elektronik Ingenieurtechnik GMBH, October 2012.

[2] ANRG Contiki page [WWW] http://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS (16.05.2015)

[3] ATmega128RFA1 Summary Datasheet. / Atmel Corporation, September 2014.

[4] AVR Dragon User Guide (updated: 08/2013). Atmel Corporation [WWW] http://www.atmel.com/webdoc/avrdragon/index.html (16.05.2015)

[5] Contiki wiki [WWW] https://github.com/contiki-os/contiki/wiki (16.05.2015)

[6] deRFnode for AVR and ARM based radio modules: Datasheet. / Dresden Elektronik Ingenieurtechnik GMBH, November 2013.

[7] Dunkels, A. Rime - a lightweight layered communication stack for sensor networks. *In Proceedings of the European Conference on Wireless Sensor Networks (EWSN)*, Poster/Demo session, Delft, The Netherlands, January 2007.

[8] Dunkels, A. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, December 2011.

[9] Dunkels, A., Asterlind, F., He, Z. An adaptive communication architecture for wireless sensor networks. *In Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys 2007)*, Sydney, Australia, November 2007.

[10] Dunkels, A., Grönvall, B., Voigt, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, November 2004.

[11] Dunkels, A., Schmidt, O., Voigt, T., Ali, M. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. *In Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, November 2006.

[12] Durvy, M., Abeillé, J., Wetterwald, P., O'Flynn, C., Leverett, B., Gnoske. E, Vidales, M., Mulligan, G., Tsiftes, N., Finne, N., Dunkels, A. Making sensor networks ipv6 ready. *In Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, poster session, Raleigh, North Carolina, USA, November 2008.

[13] Ko, J, Eriksson, J., Tsiftes, N, Dawson-Haggerty, S., Durvy, M, Vasseur, JP., Terzis, A., Dunkels, A., and Culler, D. Beyond Interoperability: Pushing the Performance of Sensornet IP Stacks. *In Proceedings of the ACM Conference on Networked Embedded Sensor Systems, (ACM SenSys 2011)*, Seattle, WA, USA, November 2011.

[14] Radio modules deRFmega128 22M00 | 22M10: Datasheet. / Dresden Elektronik Ingenieurtechnik GMBH, September 2013.

[15] Romkey, J. A Nonstandard for transmission of IP datagrams over serial lines: SLIP (RFC 1055), June 1988.

[16] Serial Line Internet Protocol Wikipedia page [WWW] http://en.wikipedia.org/wiki/Serial_Line_Internet_Protocol (16.05.2015)

[17] Zolertia Contiki page [WWW] http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_Lesson_0 (16.05.2015)

[18] USB radio stick deRFusb 23E00 | 23E06 JTAG Datasheet. / Dresden Elektronik Ingenieurtechnik GMBH, February 2013.

[19] Wikipedia Contiki page [WWW] http://en.wikipedia.org/wiki/Contiki (16.05.2015)

Lisa 1. Raadiomooduli deRFmega128-22M00 tehnilised andmed.

Technical Data	
Dimensions	23.7 x 13.2 x 3.0 mm
Operating temperature	- 40 to +85°C
Power supply	1.8 to 3.6 VDC
Power consumption @ 3.3 VDC	TX: 18 mA @ +3 dBm | RX: 18 mA | Sleep: <1 µA
Connections	51 pads
Antenna	Chip ceramic antenna
Antenna gain	+1.3 dBi (peak) | - 0.5 dBi (average)
Range	>200 m (line of sight)
Frequency range	2.4 GHz
Transmit power	+3 dBm
Receiver sensitivity	-100 dBm (250kbit/s)
Communication standard	IEEE 802.15.4
Data rate (gross)	250 kbit/s, 500 kbit/s, 1 Mbit/s, 2 Mbit/s
Microcontroller	ATmega128RFA1
Transceiver	Integrated
Interfaces	JTAG, UART, I2C, ADC, SPI, GPIO
Certification	CE, ETSI, FCC

Lisa 2. Contiki OS-i arenduskeskkonna seadistamine Linux keskkonnas

Linuxi kasutajatel ei ole Contiki arendamisega alustamiseks teha muud, kui vajalikud tööriistad installeerida ja Contiki lähtekood alla laadida.

Kõik vajalikud tööriistad saab installeerida järgmise käsuga:

sudo apt-get install build-essential binutils-msp430 gcc-msp430 msp430-libc msp430mcu mspdebug binutils-avr gcc-avr gdb-avr avr-libc avrdude openjdk-7-jdk openjdk-7-jre ant libncurses5-dev

Contiki lähtekoodi saab alla laadida aadressilt:

http://downloads.sourceforge.net/project/contiki/Contiki/Contiki 2.7/contiki-2.7.zip

Lisa 3. InstantContiki virtuaalmasina seadistamine Windows keskkonnas

Virtuaalmasina kasutamiseks on vaja installeerida mõni virtuaalmasina mängija – näiteks *VMWare player* või *VirtualBox*. Järgnevalt täpsemalt sellest, kuidas Windowsi keskkonnas *VMWare Playeri* abil Contiki arenduskeskkond üles seada.

Installeeri programmid:

VMWare player on programm, millel virtuaalmasin jookseb (Alternatiivina võib kasutada ka *Virtual Box*-i). Programm on tasuta allalaetav aadressilt:

https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/7_0

InstantContiki on Ubuntu Linuxi virtuaalmasin, milles on üles seatud Contiki arenduskeskkond. InstantContiki on allalaetav (zip 2.2GB) Contiki kodulehelt:

http://www.contiki-os.org/download.html

Ava InstantContiki VMWare Playeris:

Käivita VMWare player: Start > All Programs > VMWare > VMWare Player

VMWare player programmi aken

InstantContiki avamiseks vali VMWare Playeris „Open a Virtual Machine“, liigu allalaetud InstantContiki kausta ja ava fail laiendiga .vmx. Seejärel peaks VMWare Playeri menüüsse ilmuma InstantContiki nimeline rida. Valides selle ja vajutades „Play Virtual Machine“ käivitub InstantContiki virtuaalmasin. Virtuaalmasinasse saab sisse saab logida parooliga „**user**“.

Kõik vajalikud tööriistad on installeeritud. Contiki lähtekood asub kaustas ~/contiki . Kaust on seotud Contiki Git arendushoidlaga. Kõige viimase koodi saab alla laadida kasutades ~/contiki kaustas käsku:

git pull origin master

Lisa 4. AVRdude tööriista kasutamine

AVRdude programm toetab laia valikut erinevaid programmaatoreid. See juhend käsitlen ainult AVRDRAGON programmaatorit. Erinevate programmaatorite kasutamiseks tuleb kasutada teisi parameetreid. Toetatud programmaatorite nimekirja näeb käsuga:

avrdude -c/?

Kõik deFRmega moodulite programmeerimiseks vajalikud parameetrid on kirjeldatud all olevas tabelis.

Parameeter	**Kirjeldus**
-p <partno>	<partno> - moodul, mida programmeeritakse
-V	verifitseerimist ei teostata
-P <port>	<port> - arvuti port, kugu programmaator on ühendatud (nt usb)
-c <programmer>	<programmer> - kasutatav programmaator
-U <mem>:<op>:<file>[:<format>]	<mem> - flash, eeprom, hfuse, lfuse or efuse <op> - r|w|v (read, write, verify) <file> - fail, mida kirjutatakse <format> - faili laiend (valikuline)

Lisa infot avrdude kasutamise parameetrite kohta leiab aadressilt:

http://www.ladyada.net/learn/avr/avrdude.html

Kõik parameetrid antakse ühe reana nagu järgnevas näites:

avrdude -p m128rfa1 -V -P usb -c dragon_jtag -U flash:w:'application.hex'

Selle käsuga saab programmeerida kõikide Atmel AVR-ATmega129RFA1 mikrokontrollerite flash mälu kasutades AVRDRAGON programmaatori JTAG liidest.

Programmifail peab asuma samas kataloogis, kus AVRdude käsk käivitatakse. Enne programmi laadimist peab flash mälu kustutama, aga AVRdude teeb seda automaatselt.

Lisa 5. AVR Studio 4 kasutamine seadmete programmeerimiseks

AVR Studio 4 on programm Atmeli mikrokontrollerite tarkvara arendamiseks ja programmeerimiseks. Selles juhend tutvustab vaid väga lühidalt, kuidas programmi abil on võimalik seadmeid programmeerida.

Installeeri AVR Studio 4.19

AVR mikrokontrollerite programmeerimiseks soovitatakse kasutada AVR Studio 4 programmi. Programm (AVR Studio 4.19 (build 730)) on tasuta allalaetav Atmeli koduleheküljelt aadressil:

http://www.atmel.com/tools/studioarchive.aspx

Programmi kasutamine:

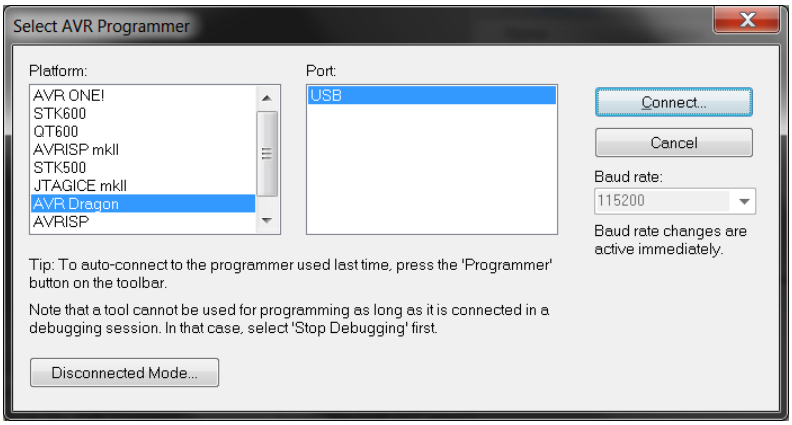
Kui AVR Studio on installitud, siis käivitamiseks vali:

Start > All Programs > Atmel AVR Tools > AVR Studio 4

Seadme programmeerimiseks vali menüüst:

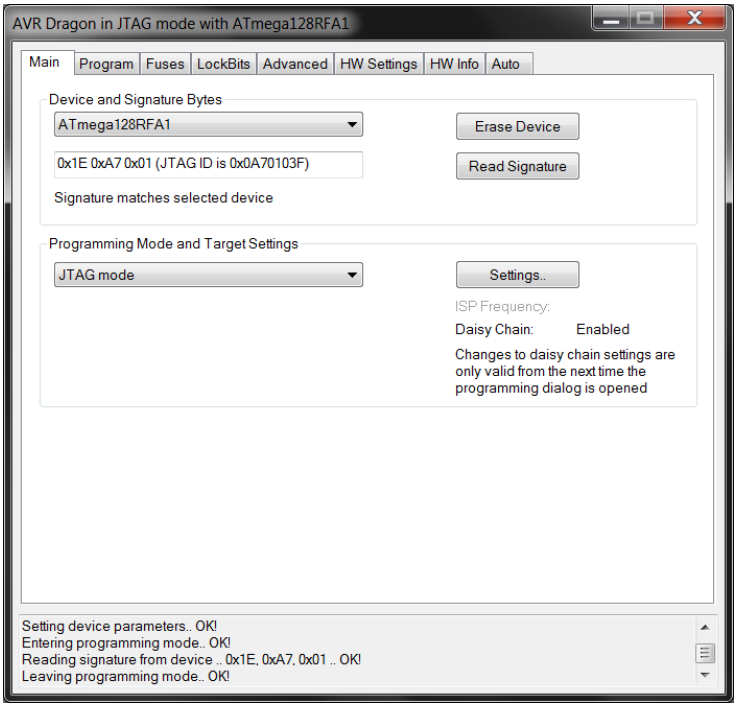
Tools > Program AVR > Connect

Avaneb aken pealkirjaga „Select AVR programmer“.

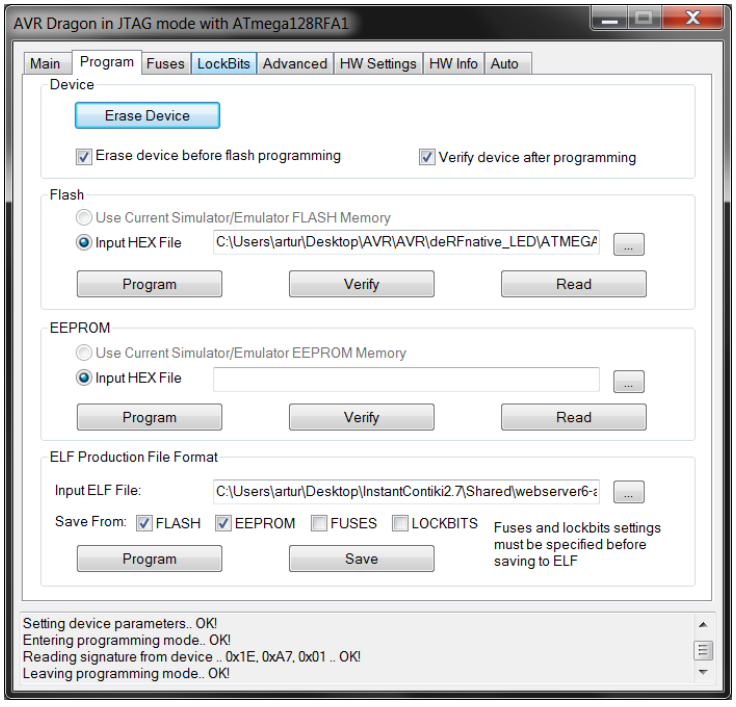


Platvormiks vali „AVR Dragon“ ja pordiks „USB“. Seejärel vajuta „Connect“.

Kui seade on JTAG kaabli abil programmaatoriga ühendatud, siis avaneb uus aken pealkirjaga „AVR Dragon in JTAG mode with Atmega128RFA1“.



„Main“ lehel saab valida seadme tüüpi ja kontrollida selle signatuuri ning muuta programmeerimise meetodit.



Programmeerimine toimub „Program“ lehel. Erinevad sektsioonid võimaldavad programmeerida erinevaid mälusid eraldi (flash, EEPROM) või kõik korraga, kasutades viimast sektsiooni ja *.elf formaadis programmifaili.

Lisa 6. Failide jagamine Windows keskkonna ja virtuaalmasina vahel.

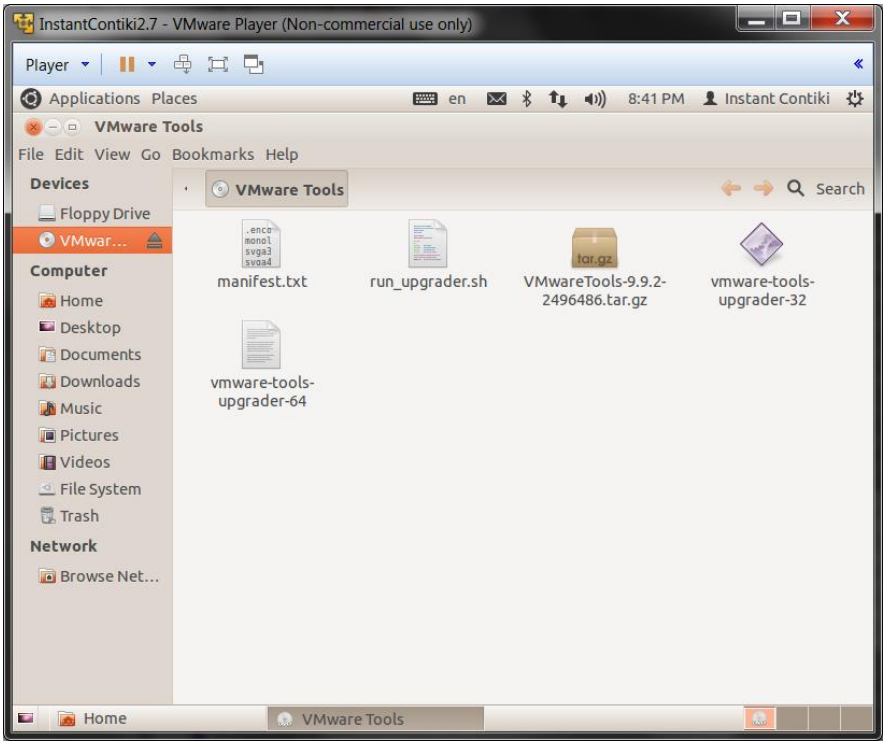
Selleks, et Windows keskkonna ja InstantContiki virutaalmasina vahel saaks faile jagada, on tarvis installida InstantContikisse VMWare Tools.

VMWare Tools installeerimine

VMWare Tools-i installimiseks peaks virtuaalmasin töötama. Vali menüüst:

Player > Manage > Install VMWare Tools

Seejärel käivitab VMWare Player virtuaalmasinas „virtual CD drive“ seadme. Virtuaalmasinas peaks nüüd näha olema uus „Device“ nimega „VMWare Tools“. Sellelt leiad faili „VmwareTools-x.x.x.tar.gz“.



Paki see fail lahti. Ava „Terminal“ ja otsi üles tekkinud kaust. Sealt leiad faili nimega „vmware-install.pl“. Käivita see käsuga:

sudo ./vmware-install.pl

NB! Installimise käigus küsib installer kasutajalt täpsustusi. InstantContiki puhul võib kõik vaikimisi väärtused jätta (võib iga küsimuse peale „Enter“ vajutada).

Kaustade jagamise aktiveerimine

Nüüd on võimalik VMWare Playeris võimalik seadistada jagatud kaustasid. Selleks vali ülevalt menüüst:

Player > Manage > VirtualMachineSettings

Avanenud aknas vali:

Options > Shared Folders

Siin saab lisada jagatavaid kaustasid ja valida, kas jagamine on alati aktiivne või ainult selle sessiooni lõpuni. Jagada on võimalik ka ainult lugemiseks (read-only).

Jagatud kaustad on virtuaalmasinas nähtavad kaustas:

/mnt/hgfs/

Lisa 7. Tunslip6 programmi kasutamine

Tunslip6 programm asub kaustas contiki/tools. Esmakordsel kasutamisel on vaja kõigepealt programm kompileerida. Selleks liigu contiki/tools kausta ja sisesta käsk:

make tunslip6

Tunslip6 programmi käivitamine toimub käsuga:

sudo ./tunslip6 [options] ipaddress

Olulised parameetrid, mis kaasa tuleb anda on:

- -B baudrate 9600,19200,38400,57600,115200 (default),230400,460800,921600
- -s siodev Serial device (default /dev/ttyUSB0)

Näiteks: *sudo ./tunslip6 –s /dev/ttyUSB1 –B 57600 aaaa::1/64*

Baudrate peab olema sama, mis on ühendatud seadme poolel kasutatava UART liidese jaoks seadistatud. UART liideste baudrate seadistatakse platvormi failis „contiki-main.c“ (vaikimisi on UART0 38400 ja UART1 57600).

Teiste parameetrite kohta saab infot kasutatades käsku:

./tunslip6 -help

Lisa 8. Border-routeri andmeedastuskiiruse mõõtmistulemused

MAC=nullmac_driver, RDC=sicslowmac_driver, FRAMER=framer_80215

```
user@instant-contiki:~$ ping6 -c 10 -s 1200 aaaa::41:1ff:fe00:5455
PING aaaa::41:1ff:fe00:5455(aaaa::41:1ff:fe00:5455) 1200 data bytes
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=1 ttl=63 time=777 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=2 ttl=63 time=607 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=3 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=4 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=5 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=6 ttl=63 time=607 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=7 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=8 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=9 ttl=63 time=608 ms
1208 bytes from aaaa::41:1ff:fe00:5455: icmp_seq=10 ttl=63 time=608 ms
--- aaaa::41:1ff:fe00:5455 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9021ms
rtt min/avg/max/mdev = 607.014/625.171/777.830/50.890 ms
```

MAC=csma_driver, RDC=contikimac_driver, FRAMER=framer_80215

```
user@instant-contiki:~$ ping6 -c 10 -s 1200 aaaa::55ff:fe14:5505
PING aaaa::55ff:fe14:5505(aaaa::55ff:fe14:5505) 1200 data bytes
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=1 ttl=63 time=858 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=2 ttl=63 time=977 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=3 ttl=63 time=729 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=4 ttl=63 time=1102 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=5 ttl=63 time=724 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=6 ttl=63 time=726 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=7 ttl=63 time=1109 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=8 ttl=63 time=735 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=9 ttl=63 time=1107 ms
1208 bytes from aaaa::55ff:fe14:5505: icmp_seq=10 ttl=63 time=733 ms
--- aaaa::55ff:fe14:5505 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 724.193/880.259/1109.174/166.331 ms, pipe 2
```

MAC=nullmac_driver, RDC=cxmac_driver, FRAMER=framer_802154

```
user@instant-contiki:~$ ping6 -c 10 -s 1200 aaaa::50:50ff:fe50:1441
PING aaaa::50:50ff:fe50:1441(aaaa::50:50ff:fe50:1441) 1200 data bytes
--- aaaa::50:50ff:fe50:1441 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9009ms
```

Kuna kõik pakid läksid kaduma, siis sai proovitud pikema intervalliga.

```
user@instant-contiki:~$ ping6 -c 10 -s 1200 -i 2 aaaa::50:50ff:fe50:1441
PING aaaa::50:50ff:fe50:1441(aaaa::50:50ff:fe50:1441) 1200 data bytes
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=1 ttl=63 time=2609 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=3 ttl=63 time=2285 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=4 ttl=63 time=2534 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=6 ttl=63 time=2335 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=7 ttl=63 time=2411 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=8 ttl=63 time=2595 ms
1208 bytes from aaaa::50:50ff:fe50:1441: icmp_seq=10 ttl=63 time=2394 ms
--- aaaa::50:50ff:fe50:1441 ping statistics ---
10 packets transmitted, 7 received, 30% packet loss, time 18029ms
rtt min/avg/max/mdev = 2285.405/2452.372/2609.078/118.640 ms, pipe 2
```