



# UTVENDIG IoT LAB

Jørgen Høyér

Aleksander Isaksen Dahl

Eivind Indahl Helle

**BO17E-12**

**14HEEL**

**31.mai 2017**

## Dokumentkontroll

<i>Rapportens tittel</i> BO17E-12 Utvendig IoT Lab	<i>Data/Versjon</i> 29. mai. 2017/2.7
	<i>Rapportnummer:</i> B017E-12
<i>Forfatter(e):</i> Eivind I. Helle Aleksander I. Dahl Jørgen Høyter	<i>Studieretning:</i> HEEL14
	<i>Antall sider m/vedlegg</i> 99
<i>Høgskolens veileder:</i> Knut Øvsthus	<i>Gradering:</i> Åpen
<i>Eventuelle Merknader:</i> Vi tillater at oppgaven kan publiseres.	

<i>Oppdragsgiver:</i> Sensario AS	<i>Oppdragsgivers referanse:</i> Infrastructure for a large scale lab for wireless sensor networks
<i>Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon):</i> Nils Jacob Berland Email: njberland@gmail.com	

Revisjon	Dato	Status	Utført av
1.1	06.05.17	Første utkast	Eivind I. Helle, Aleksander I. Dahl, Jørgen Høyter
1.2	10.05.17	Revidert utkast	Eivind I. Helle, Aleksander I. Dahl, Jørgen Høyter
1.3	11.05.17	Korreksjon av innholdsfortegnelse og nummerering	Jørgen Høyter
1.4	12.05.17	Lagt til delkapitler og bildetekst	Aleksander I. Dahl, Eivind I. Helle, Jørgen Høyter
1.5	16.05.17	Lagt til forkortelser og ordforklaringer	Jørgen Høyter
1.6	18.05.17	Lagt til Gant, drifts og vedlikehold dokumentasjon og risikotabell	Aleksander I. Dahl
1.7	19.05.17	Lagt til prosjektorganisering i appendiks c	Jørgen Høyter
1.8	24.05.17	La til to tester	Eivind I. Helle
1.9	26.05.17	Fikset på teori kapittel	Jørgen Høyter
2.0	26.05.17	Lagt til konklusjon og organisering av rapport	Aleksander I. Dahl, Jørgen Høyter
2.1	26.05.17	Fikset feil i overskrifter	Eivind I. Helle, Jørgen Høyter
2.2	27.05.17	Korrektur	Aleksander I. Dahl
2.3	27.05.17	La til resultat og brukerdokumentasjon	Eivind I. Helle
2.4	28.05.17	Korrektur	Jørgen Høyter
2.5	28.05.17	Korrektur	Eivind I. Helle
2.6	29.05.17	Korrektur	Eivind I. Helle, Jørgen Høyter, Aleksander I. Dahl
2.7	29.05.17	Ferdigstilling	Eivind I. Helle, Jørgen Høyter, Aleksander I. Dahl

## Forord

Etter tre spennende år på elektroinstituttet ved Høgskolen på Vestlandet (HVL) markeres slutten med denne bacheloroppgaven. Det har vært en utrolig lærerik og utfordrende oppgave som vi har investert mye tid og kjærlighet i. Oppgaven er utarbeidet av Sensario AS og Høgskolen på Vestlandet og tar utgangspunkt i et realistisk og reelt ingeniørproblem. Vi har fått bryne oss på balansen mellom å kombinere digital- med analog elektronikk og jobbe på tvers av fagfelt. Dette er også noe vi kan få stor nytte av når vi skal ut i arbeidslivet.

Vi vil takke Sensario AS og da spesielt vår veileder Nils Jacob Berland for en utfordrende og spennende oppgave. Nils Jacob har stilt opp til møter på kort tid og er alltid tilgjengelig på mail og har et utrolig smittende engasjement for ny teknologi.

Takk til vår faglige veileder Knut Øvsthus som har bidratt med den teoretiske tyngden som oppgaven har krevd.

Takk til Contiki ekspert Andreas Urke for all hjelp og rettledning.

Takk til omsorgslabben og Thomas Aasebø for hjelp med testing og rutertilgang.

Takk til Farzan Jouleh som tok seg av bestillinger og adgang til PCB labben når skolen var full.

Vi vil også takke våre venner og familie for all tålmodighet og støtte de siste 3 årene.

## Sammendrag

HVL tilbyr studier og gjennomfører forskning innen kommunikasjonsteknologi og elektronikk. Innenfor de nevnte disiplinene er den teknologiske utviklingen i rask vekst og dette skaper et behov for oppdaterte test- og læringsplattformer. I samarbeid med bedriften Sensario AS ønsker HVL å utforske muligheten for å danne tre store radionettverk i Bergensdalen. Hensikten er å undersøke hvordan slike radionettverk oppfører seg i praksis, samt å tilby studenter muligheten til å teste relevante algoritmer og teknologi. Denne prosjektrapporten dokumenterer et forslag til design og realisering av infrastrukturen til radionettverkene.

Kravspesifikasjonen fra oppdragsgiver var som følgende:

- Sentralisert opplasting av programmer til et sett med trådløse noder i systemet.
- Sentralisert monitorering og logging av alle trådløse noder i systemet.
- Hardware med plass til 3 separate trådløse noder for hver fysisk lokasjon.

Ved å ta i bruk langdistanse radionoder har vi utviklet en løsning som danner infrastrukturen for tre parallelle radionettverk med 3 separate noder for hver fysisk lokasjon. Hvert av nettverkene består av 4 noder og rekkevidden mellom dem er opptil 1 km. Rekkevidden kan utvides ved å montere på eksterne antenner. Ved hjelp av en tilkoblet enhet logges informasjon fra nodene og videresendes til en server. Dette gir muligheten for å overvåke nettets ytelse og stabilitet. Fra serveren er det også mulig å laste opp ny programvare til nodene. Nettverkene krever svært lav effekt og størrelsen på dem kan enkelt skaleres opp ved å legge til flere noder. For å sette opp nye noder er tilgang til internett og strøm det eneste som er nødvendig. Støv- og vanntett innkapsling gjør det mulig å plassere maskinvaren utendørs.

Med hensyn til kravspesifikasjonen oppfyller løsningen oppgavens målsetning. Systemet er klargjort for installasjon og fra en sentralisert server kan radionettverkene overvåkes og oppdateres.

Nettverkene danner i seg selv en testplattform for trådløs dataoverføring, men kan også videreføres til et trådløst sensornettverk for datainnsamling.

## Innholdsfortegnelse

1	Innledning .....	11
1.1	Oppdragsgiver .....	11
1.2	Problemstilling.....	11
1.3	Hovedidé for løsningsforslag.....	11
1.4	Kravspesifikasjon .....	11
1.5	Modulbeskrivelse .....	12
1.6	Avgrensning .....	13
1.7	Organisering av rapport .....	13
2	Metode .....	14
3	Teori.....	16
3.1	Internet of Things .....	16
3.2	Trådløse sensornettverk.....	16
3.3	Protokoller.....	16
3.3.1	Transmission Control Protocol .....	16
3.3.2	Internett protokollen.....	16
3.3.3	Serial Line IP .....	17
3.3.4	802.15.4 Mesh-nettverk.....	18
3.3.5	Transport Layer Security.....	18
3.3.6	Secure Hash standard.....	18
3.3.7	The Constrained Application Protocol.....	19
3.3.8	The Universal Asynchronous Receiver .....	19
3.3.9	Joint test action group.....	19
3.4	Standarder .....	19
3.4.1	International Protection Marking.....	19
3.4.2	Driftstemperatur .....	20
3.5	Maskinvare .....	20
3.5.1	BeagleBone Green.....	20
3.5.2	BeagleBone Black Wireless.....	20
3.5.3	CC1350 Launchpad .....	21
3.5.4	CC2650 Launchpad .....	21
3.6	Programvare .....	21
3.6.1	VMWare Workstation .....	21
3.6.2	SmartRF Flash Programmer 2.....	21

3.6.3	GitHub.....	21
3.6.4	Python .....	22
3.6.5	C.....	22
3.7	Operativsystemer .....	22
3.7.1	Debian.....	22
3.7.2	Contiki.....	23
3.7.3	Ubuntu 16.10.....	23
4	Design .....	24
4.1	Valgt løsning .....	24
4.2	Fysisk design og realisering .....	25
4.2.1	Komponenter.....	25
4.2.1.1	<i>Beaglebone Green</i> .....	25
4.2.1.2	<i>CC1350 Launchpad</i> .....	25
4.2.1.3	<i>Strøm kilde</i> .....	26
4.2.1.4	<i>Koblingsboks</i> .....	26
4.2.2	Realisering .....	27
4.2.2.2	<i>Fargekoding</i> .....	27
4.2.2.3	<i>Modifikasjoner</i> .....	28
4.3	Oppsett av systemet.....	29
4.3.1	Github.....	29
4.3.2	OpenVPN .....	29
4.3.3	UART & I/O PIN.....	29
4.3.4	Python skript .....	30
4.3.5	Server.....	31
4.3.6	Contiki OS .....	31
4.3.7	Oversikt over protokoller .....	32
4.4	Valg .....	33
4.4.1	Bokser .....	33
4.4.2	UART .....	33
4.4.3	OS .....	33
4.4.4	Ubuntu 16.04.....	33
4.4.5	OpenVPN .....	33
4.4.6	HW .....	34
4.4.7	Antenne .....	34

4.4.8	Contiki bakdør .....	34
4.4.9	BeagleBone Black wireless .....	34
5	Testing .....	35
5.1	Rekkeviddetest 1 .....	35
5.1.1	Hensikt.....	35
5.1.2	Testoppsett.....	35
5.1.3	Observasjoner.....	36
5.1.4	Diskusjon .....	36
5.1.5	Konklusjon .....	36
5.2	Rekkeviddetest 2 .....	37
5.2.1	Hensikt.....	37
5.2.2	Testoppsett distanse .....	37
5.2.3	Observasjoner rekkeviddetest.....	38
5.2.4	Testoppsett for hopp.....	39
5.2.5	Observasjoner for hopp.....	39
5.2.6	Diskusjon .....	40
5.2.7	Konklusjon .....	40
5.3	Stabilitetstest.....	41
5.3.1	Hensikt.....	41
5.3.2	Testoppsett.....	41
5.3.3	Observasjoner.....	41
5.3.4	Diskusjon .....	43
5.3.5	Konklusjon .....	43
5.4	Samlet systemtest .....	44
5.4.1	Hensikt.....	44
5.4.2	Testoppsett.....	44
5.4.3	Observasjoner.....	45
5.4.4	Diskusjon .....	49
5.4.5	Konklusjon .....	49
6	Resultateter .....	50
7	Diskusjon .....	51
7.1	Forutsetninger .....	51
7.2	Fremdrift.....	51
7.3	Risiko .....	51

7.3.1	Bestilling og levering av CC1350 Launchpad .....	51
7.3.2	Fase 4 i Gant-diagram.....	52
7.3.3	Arbeidssted.....	52
8	Konklusjon .....	53
8.1	Videreføring av oppgaven .....	53
	Litteraturliste.....	54
Appendiks A	Forkortelser og ordforklaringer.....	57
Appendiks B	Prosjektledelse og styring.....	60
B.1	Prosjektorganisasjon .....	60
B.2	Fremdriftsplan .....	60
B.3	Risikoliste.....	62
Appendiks C	Brukerdokumentasjon.....	63
C.1	Brukerdokumentasjon.....	63
C.2	Drifts- og vedlikeholdsdocumentasjon .....	68
Appendiks D	Kildekoder og Bill of Materials .....	99
D.1	Kildekode .....	99
D.2	Bill of Materlials.....	99

## Figurliste

Figur 1:	Modulbeskrivelse .....	12
Figur 2:	Oppdeling av systemet.....	14
Figur 3:	Modell av valgt metode.....	15
Figur 4:	Inforgrafikk .....	24
Figur 5:	Beaglebone Green .....	25
Figur 6:	CC1350 Launchpad .....	25
Figur 7:	Hensel koblingsboks .....	26
Figur 8:	Koblingsboks med hardware .....	27
Figur 9:	Tilkobling for XDS110, CC1350 Launchpad.....	28
Figur 10:	Koblingsskjema .....	28
Figur 11:	Debug skript .....	30
Figur 12:	Bakdør-skript .....	30
Figur 13:	Protokollskjema.....	32
Figur 14:	Testområde, rekkeviddetest 1 .....	35
Figur 15:	Testområde, rekkeviddetest 2 .....	37
Figur 16:	Testoppsett for hopp, rekkeviddetest 2.....	39
Figur 17	OpenVPN server logg. frakobling av client.....	41
Figur 18	OpenVPN serverlogg. siste kontakt mellom server, client2 og 5 .....	41
Figur 19	OpenVPN serverlogg. siste kontakt mellom server og client3.....	41

Figur 20 Første linje lagret i debug logg .....	42
Figur 21 Siste linje lagret i debug logg nett1 .....	42
Figur 22 utdrag fra debug logg til nett2 .....	42
Figur 23 utdrag fra debug logg til nett3 .....	43
Figur 24 Testoppsett samlet systemtest .....	44
Figur 25 Boks etter tilkobling av strøm .....	45
Figur 26 OpenVPN liste over tilkoblede klienter .....	45
Figur 27 Oppstart av tunslip6 .....	46
Figur 28 skript sekvens for flashing .....	46
Figur 29 Arkivering av gammel .bin fil .....	47
Figur 30 .bin fil i riktig mappe på BBG .....	47
Figur 31 flashing av node 2 på client2 .....	47
Figur 32 flashing av node2 på client3 .....	48
Figur 33 overføring av logg filer fra BBG til server .....	48
Figur 34 tidsstempeling av debug mapper .....	49
Figur 35 innhold i debug mappe .....	49
Figur 36: Initiale Gant-diagram uke 1-13 .....	60
Figur 37: Initiale Gant-diagram uke 14-24 .....	61
Figur 38: Revidert Gant-diagram uke 1 til 13 .....	61
Figur 39: Revidert Gant-diagram uke 14 til 24 .....	62
Figur 40: win 32 diskimager .....	68
Figur 41: Putty BBG ssh .....	69
Figur 42: TCP innstillingar for deling av internett .....	70
Figur 43: BBG resolv.conf .....	70
Figur 44: Skript for Internett av Derek Molloy .....	71
Figur 45: BBG UART pins for minicom .....	74
Figur 46: Putty Minicom .....	74
Figur 47: Putty Minicom .....	75
Figur 48: BBG Minicom Uart for to CC1350 .....	75
Figur 49: Debug data fra node .....	77
Figur 50: Debug data fra BR .....	78
Figur 51: Rpl debug data fra node .....	82
Figur 52: Debug data fra BR .....	82
Figur 53: Koble CC1350 til Contiki .....	83
Figur 54: Starte BR i Contiki .....	84
Figur 55: Ipv6 adresse BR .....	84
Figur 56: Firefox BR oversikt over noder .....	84
Figur 57: Firefox COAP server .....	85
Figur 58: OpenVPN konfigurasjon- CA variabler .....	86
Figur 59: OpenVPN konfigurasjon- CA variabler eksempel .....	87
Figur 60: OpenVPN konfigurasjon- Key name .....	87
Figur 61: OpenVPN konfigurasjon- key server .....	88
Figur 62: OpenVPN konfigurasjon- key direction .....	89
Figur 63: OpenVPN konfigurasjon- public network interface .....	90
Figur 64: OpenVPN konfigurasjon- rules .....	90

Figur 65: OpenVPN konfigurasjon- accept .....	91
Figur 66: OpenVPN konfigurasjon- openVPN status .....	92
Figur 67: OpenVPN konfigurasjon- tun0 .....	92
Figur 68: OpenVPN konfigurasjon- Autostart.....	98

## Tabelliste

Tabell 1: Observasjoner, rekkeviddetest 1.....	36
Tabell 2: Observasjoner, rekkeviddetest 2 .....	38
Tabell 3: Observasjoner for hopp, rekkeviddetest 2 .....	39

## 1 Innledning

### 1.1 Oppdragsgiver

Sensario AS er en oppstartsbedrift som arbeider med testing og utvikling av trådløse sensornettverk. Deres pågående prosjekter omhandler måling av luftkvalitet, vannstand i båter og registrering av sauер på beite. Bacheloroppgaven er utformet av ekstern veileder fra Sensario AS, Nils Jacob Berland. Vår interne veileder på Høgskolen på Vestlandet er Knut Øvsthus.

### 1.2 Problemstilling

HVL og Sensario AS ønsker å bygge og drive et stort testnett for trådløse sensornettverk. Testing betyr i praksis at vi skal kunne logge informasjon om ytelse og stabilitet samt å kunne administrere flere parallelle nettverk på hver fysisk lokasjon. Dette vil gjøre det mulig for studenter og andre å teste forskjellige algoritmer og teknologier på store trådløse radionettverk - dette finnes i praksis ikke i dag. På hvilken måte kan løsningen for et slikt nettverk realiseres?

### 1.3 Hovedidé for løsningsforslag

Hovedidéen for oppgaven er å danne infrastruktur for testnettet. I praksis vil dette utgjøre 15 radionoder utplassert på 5 separate lokasjoner, tre noder pr lokasjon, som danner tre separate nettverk. Det skal være mulig å koble til sensorer på nodene og dataen fra sensorene skal videreføres gjennom radionettet via en gateway til en server. I tillegg skal de tre nodene på hver lokasjon være koblet til en node-kontroller som skal logge feilsøkningsinformasjon, debug-data, fra nodene og videresende den til en sentral server. Gjennom node-kontrollerne skal det også være mulig å laste opp ny programvare fra serveren. Når infrastrukturen for testnettet er dannet skal det være lett skalarbart slik at det i fremtiden er mulig å utvide nettet til et sted mellom 10 og 100 lokasjoner og kunne dekke området fra NHH til Nesttun.

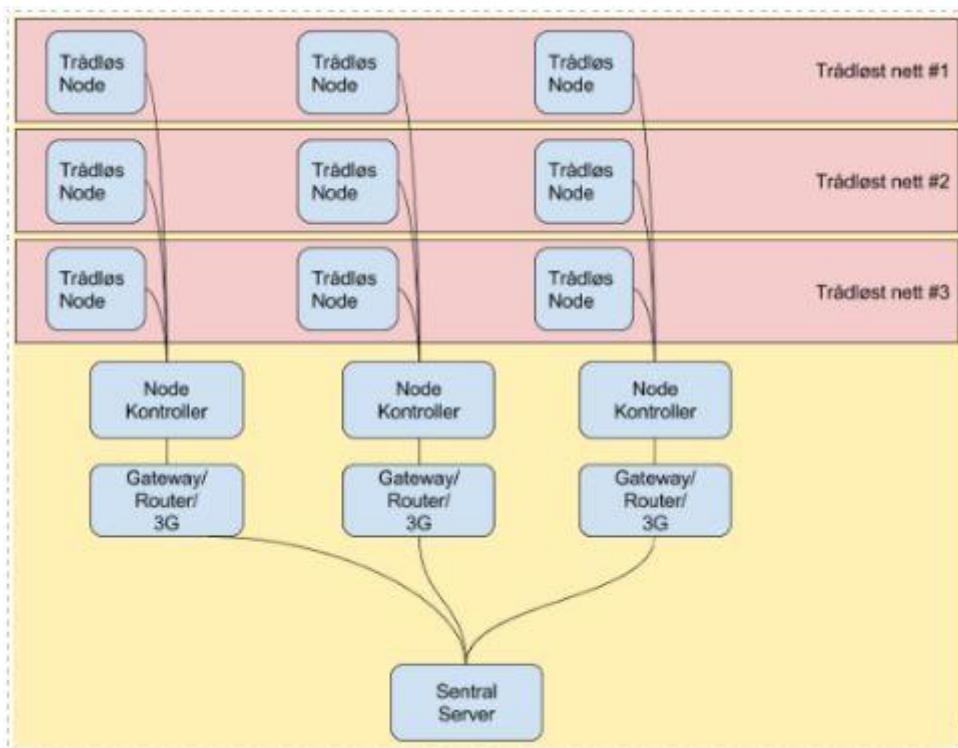
### 1.4 Kravspesifikasjon

Systemet som skal designes, bygges og driftes over en periode skal tilfredsstille følgende:

- Sentralisert opplasting av programmer til et sett med trådløse noder i systemet.
- Sentralisert monitorering og logging av alle trådløse noder i systemet.
- Hardware med plass til 3 separate trådløse noder for hver fysisk lokasjon.

## 1.5 Modulbeskrivelse

- Back-bone nettverk basert på BeagleBone (node-kontroller) samt server infrastruktur.
- Kommunikasjon mellom node kontroller og server - denne må kunne fungere selv om node-kontroller står bak brannmurer.
- Trådløse moduler basert på Texas Instruments CC1350.
- Programmene som lastes opp til de trådløse modulene skal i utgangspunktet være basert på Contiki OS.



Figur 1: Modulbeskrivelse

## 1.6 Avgrensning

Oppgaven, slik den er presentert fra veileder, er i seg selv krevende og berør flere fagområder som vi ikke hadde kjennskap til når vi startet på prosjektet. I tillegg satt vi oss enkelte utvidede mål som vi kunne jobbe mot dersom vi fikk tid. Dette gjorde det nødvendig å avgrense oppgaven i oppstartsfasen av prosjektperioden, slik at den ikke skulle bli for tidkrevende og utfordrende for gruppen.

Den første avgrensningen vi gjorde var knyttet til operativ systemet Contiki som etter krav fra oppdragsgiver skulle kjøres på nodene. Vi vurderte det som en svært krevende oppgave å gå i dybden til et operativ system som er svært annerledes enn de vi har kjennskap til fra før. Derfor ble det bestemt, i samarbeid med ekstern veileder, og ikke bruke for mye tid på dette, men heller kun fokusere på de funksjonene i Contiki som var nødvendig for å løse oppgaven.

I tillegg bestemte vi tidlig at vi ikke skulle koble til sensorer på nodene fordi vi ikke så på dette som en del av infrastrukturen til nettet. Det er derimot tilrettelagt for plass i innkapslingen og ledige serielle porter på nodene slik at fremtidige administratorer av nettet kan koble til sensorer.

## 1.7 Organisering av rapport

**Kapittel 2** vil beskrive valgt design-, implementasjons- og verifikasjonsmetode.

**Kapittel 3** gir en innføring i standarder, protokoller og komponenter vi har tatt i bruk.

**Kapittel 4** vil beskrive design og implementasjon og valgene vi har tatt knyttet til dette.

**Kapittel 5** gjennomgår tester og testresultat.

**Kapittel 6** gir en samlet oppsummering av testresultatene.

**Kapittel 7** inneholder diskusjon knyttet til gjennomføring av oppgaven.

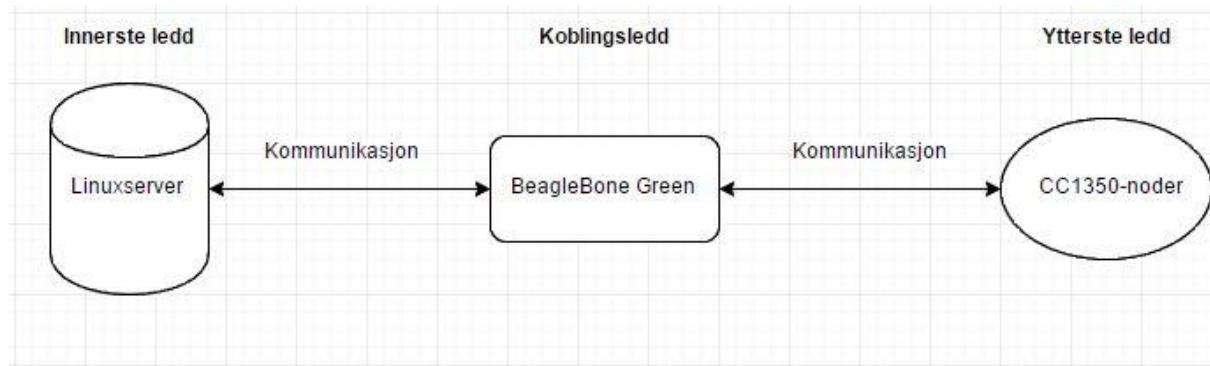
**Kapittel 8** inneholder konklusjon.

## 2 Metode

Siden store deler av innholdet i bacheloroppgaven ikke har vært en del av undervisningsløpet i bachelorgraden startet vi prosjektperioden med å søke etter dokumentasjon fra troverdige kilder som omhandlet generelle beskrivelse av Internet of Things (IoT) og trådløse sensornettverk (WSN). Et av kravene til bacheloroppgaven var å ta i bruk Contiki OS og TI-noder basert på CC13xx-chipen. Gjennom søk på TI sine nettsider fant vi boken «IoT in 5 days», utgitt av International Centre for Theoretical Physics (ICTP) [1]. Denne boken ga oss grunnleggende forståelse av IoT-teknologi. Etter ytterligere kildesøk fant vi også white paper publikasjonen «Internet of Things: Wireless Sensor Networks» av International Electrotechnical Commision (IEC) som bidro til å gi oss en bredere forståelse av den nære koblingen mellom IoT og WSN [2]. I oppstartsfasen av prosjektet var det også nødvendig med hyppige møter med ekstern oppdragsgiver som har utarbeidet prosjektets rammer. Kravene fra oppdragsgiver var tydelige og vi forsto tidlig hva systemet skulle tilfredsstille. Gjennom møtene dannet vi oss også et oversiktsbilde over hvordan systemet kunne designes.

Systemet består av flere ulike enheter som samhandler og kommuniserer med hverandre. For enkelhetsskyld valgte vi å dele det opp i tre ledd, der hvert av dem er knyttet opp mot maskinvare:

- Ytterste ledd: CC1350-noder med tilhørende programvare.
- Koblingsledd: BeagleBone Green med tilhørende programvare.
- Innerste ledd: Linuxserver med tilhørende programvare.

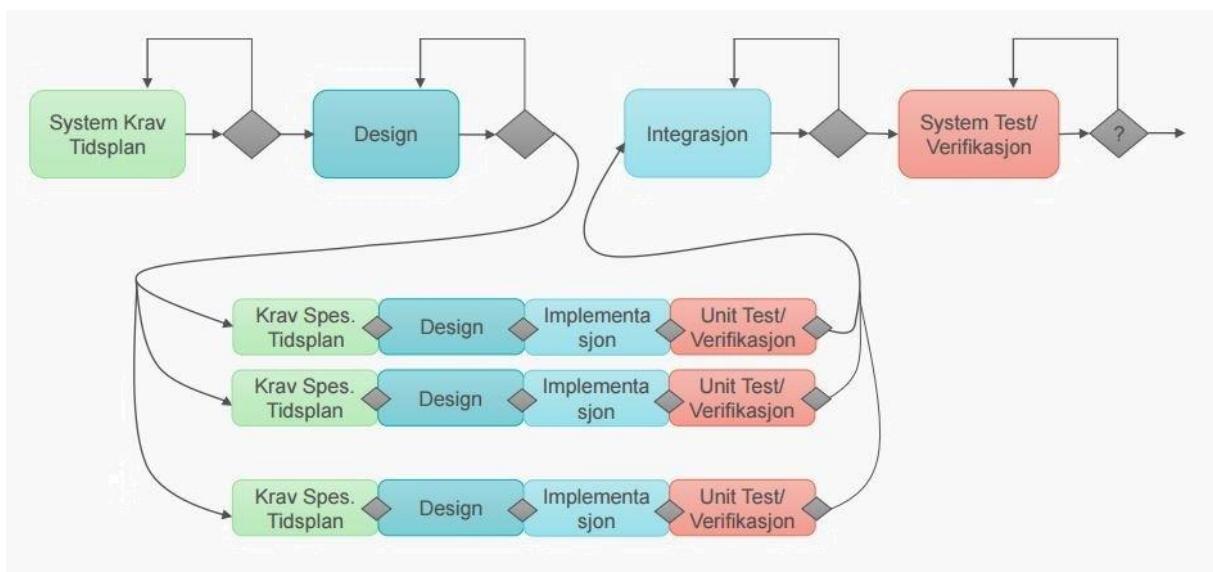


Figur 2: Oppdeling av systemet

Hvert ledd kan oppfattes som et sub-system som måtte designes, implementeres og testes før de kunne integreres i systemet. Hvert sub-system består av en rekke byggelklosser der oppgavene for å realisere dem varierer i størrelse. Derfor så vi det nødvendig å samarbeide om de større oppgavene, mens vi i andre tilfeller kunne jobbe parallelt med ulike mindre oppgaver. For at alle gruppedellemmene skulle få bred forståelse av systemet var det også viktig for oss at alle fikk være med på utarbeidingen av de viktigste byggelklossene. Derfor samarbeidet vi også alltid om de oppgavene som ga oss et innblikk i prosjektets kjerne som for eksempel oppgaver knyttet til uthenting av feilsøkingsinformasjon fra nodene, opplasting av ny programvare til nodene og kommunikasjon mellom nodene.

En av de største utfordringene for oss gjennom prosjektperioden var at deler av teknologien vi har tatt i bruk er relativt ny. Dette medfører at det fremdeles er få realiseringer og mangelfull dokumentasjon av systemer som baserer seg på teknologien. På bakgrunn av dette krevde deler av arbeidet at vi innhentet informasjon fra åpne forum og blogger der fagfolk diskuterer og presenterer mulige løsninger. Dette gjaldt i stor grad arbeidet knyttet til CC1350 Launchpads (Lp) der vi i flere tilfeller var nødt til å overføre kunnskap innhentet fra dokumentasjonen av systemer som baserer seg på eldre versjoner av samme teknologi til vårt bruk. Samtidig hadde vi en god dialog med ressurspersoner på HVL med kunnskap om deler av vårt system, noe som var til stor hjelp når vi sto fast underveis. Etter at kravene til en byggekloss var utarbeidet kunne vi bruke innhentet informasjon til å forstå hvordan den skulle designes.

Arbeidet videre krevde gjentatte runder med implementasjon og testing før vi kunne verifisere at hver del var klar for integrasjon i sitt respektive ledd av systemet. Ettersom kommunikasjon mellom leddene var nødvendig for å verifisere at deler av hvert sub-system oppførte seg slik vi ønsket, ble det nødvendig å opprette kommunikasjonen før hvert enkelt sub-system var ferdigstilt. Dette medførte at en rekke større og mindre systemtester ble utført underveis som kunne gi oss en indikasjon på om vi var på riktig vei.



*Figur 3: Modell av valgt metode*

## 3 Teori

### 3.1 Internet of Things

Begrepet Internet of Things (IoT) ble utarbeidet av Kevin Ashton i 1999 og referer til unike identifiserte objekter som for eksempel store bygninger, biler, dyr eller planter og deres virtuelle representasjoner i en «internett-aktig» representasjon. Selv om IoT ikke definerer bruk av spesiell kommunikasjonsteknologi er det naturlig å tenke seg at trådløs kommunikasjonsteknologi vil spille en viktig rolle for systemer som kan knyttes til dette begrepet. Derfor har ideen om IoT utviklet seg parallelt med trådløse sensornettverk [1].

### 3.2 Trådløse sensornettverk

Et trådløst sensornettverk (WSN) kan generelt beskrives som et nettverk av noder som samarbeider om å overvåke et miljø ved hjelp av å registrere og prosessere målbare verdier. Dette gjør det mulig for samvirking mellom mennesker, datamaskiner og omgivelser. Registrering, prosessering og kommunikasjon av data må i mange tilfeller gjøres med begrenset mengde tilgjengelig energi som krever et systemdesign med felles hensyn på laveffekt signal/data prosessering og kommunikasjons protokoller. Et av de viktigste bruksområdene for WSN er innenfor industriell automatisering, i stor grad knyttet til prosessindustri som olje- og gassutvinning. WSN er tett knyttet opp mot IoT [1].

### 3.3 Protokoller

#### 3.3.1 Transmission Control Protocol

Transmission Control Protocol (TCP) er standardisert av *the Internet Engineering Task Force* (IETF), publikasjon RFC793. TCP brukes som en høyt pålitelig *host-til-host* protokoll i pakke-basert datakommunikasjon og andre systemer som er knyttet til slike nettverk [2].

#### 3.3.2 Internett protokollen

Internett protokollen (IP) er standardisert av IETF, publikasjon RFC791. Den gir muligheten for overføring av blokker av data mellom to punkt, ofte kalt *hosts*, som er identifiserte gjennom fastsatte adresselengder [3].

### 3.3.2.1 IPv6

IP versjon 6 (IPv6) er en nyere versjon av internett protokollen og er etterkommeren av IP versjon 4 (IPv4). Det er primært fem endringer fra IPv4 til IPv6:

1. Utvidet antall tilgjengelige adresser:  
IPv6 øker antall adresser fra 32 til 128 bits for gjøre tilgjengelig et mye større antall adresserbare noder og enklere automatisk konfigurering av adressene.
2. Forenkling av IPv4-hodets format:  
Noen av IPv4-hodets felt har blitt fjernet for å redusere prosesserings kostander og begrense båndbredde kost for IPv6 header.
3. Forbedret støtte for utvidelser og andre muligheter:  
Endringer i hvordan IP-hodet er kodet gir bedre effektivitet og fleksibilitet.
4. Evnen til *Flow Labeling*:  
IPv6 behersker å merke IP-pakker som tilhører en bestemt pakkeflyt i spesielle tilfeller.
5. Forbedret autentifisering og personvern:  
Utvidelser som i større grad støtter autentifisering, dataintegritet og datakonfidensialitet er spesifisert for IPv6.

IPv6 er standardisert av IETF, publikasjon RFC2460 [4].

#### 3.3.2.1.1 6LoWPAN

6LoWPAN er et IPv6 format for overføring av IP-pakker over IEEE 802.15.4-protokollen i nettverk. Dette er en protokoll for datakommunikasjon for enheter som bruker lav datarate, lav effekt og lav kompleksitets radiobølgeoverføring over korte avstander i et trådløst personlig nettverk (WPAN). 6LoWPAN-nettverk støtter både stjerne og mesh topologi [5].

### 3.3.3 Serial Line IP

Serial Line IP (SLIP) protokollen er de facto standard for seriell tilkobling over TCP/IP protokollen. SLIP definerer en sekvens av tegn som utgjør IP-pakker, men tilbyr ingen adressering, pakketype identifikasjon, feildeteksjon eller kompresjons mekanismer. Siden protokollen gjør relativt lite er den som regel enkel å implementere. SLIP er som regel brukt til dedikerte seriell linker og i enkelte tilfeller til dial-up formål. Protokollen er svært nyttig for kommunikasjon mellom en miks av rutere og verter, enten det er vert til vert, ruter til vert eller ruter til ruter [6].

### 3.3.4 802.15.4 Mesh-nettverk

Et trådløst mesh-nettverk (WMN) er et kommunikasjonsnettverk som består av radionoder organisert i en mesh topologi. I et 802.15.4 Mesh-nettverk knyttes radionodene i nettverket sammen gjennom robust, multi-hopp kommunikasjon med endepunkt i en router-node, også kalt gateway. Mesh-nettverk er svært fleksible, men krever høyere kompleksitet enn for eksempel nettverk med stjerne topologi fordi det kreves ende til ende tilkobling mellom alle enhetene. Mesh-nettverk, på norsk maskenettverk, og dets topologi er beskrevet av IEEE [7].

### 3.3.5 Transport Layer Security

Transport Layer Security (TLS) protokollen er standardisert av IETF, publikasjon RFC5246 (2008), og har som primær oppgave å tilby personvern og dataintegritet mellom to kommuniserende applikasjoner. Protokollen består av to lag: *TLS Record Protocol* og *TLS Handshake Protocol*. TLS-protokollen tilbyr forbindelsessikkerhet som har to grunnleggende egenskaper:

- Forbindelsen er privat.
- Forbindelsen er pålitelig.

TLS Record protokollen brukes til innkapsling av diverse protokoller i høyere lag. En av dem er TLS Handshake Protokollen som gjør det mulig for serveren og klienten og autentisere hverandre.

Forbindelsessikkerhet som TLS Handshake tilbyr har tre grunnleggende egenskaper:

- Identiteten kan autentiseres gjennom krypterte algoritmer og kryptografiske nøkler (RSA, DSA)
- Forbindelsen mellom serveren og klientene er sikker: blant annet basert på en felles hemmelig delnøkkel som er utilgjengelig for angripere som prøver å avlytte forbindelsen.
- Forbindelsen er pålitelig: ingen angriper kan modifisere kommunikasjonen uten å bli oppdaget.

En fordel med TLS er at den er uavhengig av applikasjonsprotokoller. Det vil si at protokoller i høyere lag kan legges transparent over TLS-protokollen [8].

### 3.3.6 Secure Hash standard

Secure Hash Standard (SHA) spesifiserer fire sikre *hash algoritmer*, SHA-1, SHA-256, SHA-384 og SHA-512, som brukes til å beregne en pakket representasjon av elektronisk data. Eller enklere sagt, en pakket representasjon av en beskjed. Standardiseringen er gjort av Federal Information Processing Standards (FIPS) [9].

#### 3.3.6.1 SHA256

SHA-256 bruker seks logiske funksjoner der hver funksjon opererer med 32-bits ord og hvert ord er representert med x, y og z koordinater. Resultatet av hver funksjonen er et nytt 32-bit ord. Standardiseringen er gjort av FIPS [9].

### 3.3.7 The Constrained Application Protocol

Begrensete nettverk (constrained networks), som for eksempel 6LoWPAN, støtter fragmentering av IPv6 pakker. Dette medfører for øvrig en signifikant reduksjon i sannsynligheten for at pakker blir levert. The Constrained Application Protocol (CoAP) har som hovedmål å tilby en generisk web protokoll som tar seg av de spesielle behovene for slike begrensete nettverk. Dette er hovedsaklig med tanke på energiforbruk, automatisering og andre maskin-til-maskin (M2M) applikasjoner. CoAP er standardisert av IETF [10].

### 3.3.8 The Universal Asynchronous Receiver

The Universal Asynchronous Receiver/Transmitter (UART) er en enhet som utfører seriell til parallel konvertering av data mottatt fra en perifer enhet og parallel til seriell konvertering av data mottatt fra en prosessor (CPU). UART inkluderer kontroll kapabilitet og et prosessor interrupt system som kan skreddersys for å minimalisere software behandling av kommunikasjons linken. UART er basert på industri standarden TL16C550 og er beskrevet av Texas Instruments [11].

### 3.3.9 Joint test action group

Joint test action group (JTAG) protokollen implementerer on-chip verifikasierte, testing og vedlikehold av kretskort (PCB). JTAG er standardisert av IEEE, standard 1149.1-1990 [12]. JTAG kretser har et standard grensesnitt hvor instruksjoner og testdata blir sendt og mottatt.

## 3.4 Standarder

### 3.4.1 International Protection Marking

International Protection Marking (IP) klassifiserer og graderer beskyttelsen mot blant annet støv og vann for koblingsbokser. Standardiseringen er gjort av International Electrotechnical Commission (IEC), standard EN 60529 [13]. Graderingen er uttrykt med de to bokstavene IP fulgt av to siffer. Kort forklart angir det første sifferet innkapslingens beskyttelse mot støv og kan variere fra 0 til 6. Det andre sifferet angir innkapslingens beskyttelse mot inntrenging av vann og kan variere fra 0 til 8.

#### 3.4.1.1 IP66

En koblingsboks med graderingen IP66 har komplett beskyttelse mot støvgjennomtrengning og har vanntetthet tilnærmet kraftig spyling mot innkapslingen fra alle kanter.

### 3.4.2 Driftstemperatur

Ut i fra våre undersøkelser finnes det ingen offisiell standardisering av driftstemperatur for elektroniske kretser, men følgende gradering er bredt akseptert:

- Kommersiell: 0 ° til 70 °C
- Industriell: -40 ° til 85 °C
- Militær: -55 ° til 125 °C

Som følge av at det ikke finnes en offisiell standardisering er det svært viktig å lese spesifikasjonene i kretsens datablad siden ulike produsenter tar i bruk egne graderinger.

For elektronikk brukt til militære formål har det amerikanske forsvaret definert en standardisering av temperatur påvirkning, standard MIL-STD-810G, for deres utstyr [14].

## 3.5 Maskinvare

### 3.5.1 BeagleBone Green

BeagleBone Green (BBG) fra SeedStudio er en lavkost og ett-kretskort datamaskin designet for utviklere. Den er basert på åpen kilde maskinvare fra BeagleBone Black. Den kan kjøre en rekke Linux operativsystemer som for eksempel Debian, Android eller Ubuntu. BeagleBone Green kan brukes innenfor områder som robotikk, internet of things (IoT) og automasjon- og prosesskontroll for å nevne noen. Tekniske beskrivelser og design er tilgjengelig på SeedStudio sitt eget nettleksikon [15].

### 3.5.2 BeagleBone Black Wireless

I likhet med BBG er BeagleBone Black Wireless (BBBW) en lavkost og ett-kretskort datamaskin fra SeedStudio, designet for utviklere. Den kan også kjøre en rekke Linux baserte operativ systemer som for eksempel Debian, Android eller Ubuntu. I motsetning til BBG, som har en påmontert 10/100 Ethernet port, har BBBW innebygd 802.11 b/g/n WiFi og bluetooth som gir muligheten for trådløs internetttilkobling. BBBW er spesielt egnet for bruk til IoT baserte prosjekter, men har også andre bruksområder [16].

### 3.5.3 CC1350 Launchpad

SimpleLink CC1350 LaunchPad (Lp) fra Texas Instruments (TI) er en trådløs mikrokontroller (MCU) som kombinerer Sub-1 GHz med en blåtann lav-effekt radio. Mikrokontrolleren trekker svært lav effekt under bruk, men har samtidig svært lang rekkevidde, opp mot 20 km med ekstern antenn, for trådløs overføring av data. CC1350 chippen består blant annet av 32-bit ARM Cortex-M3 prosessor som kjører 48MHz som hovedprosessor og har en rekke perifere funksjoner som for eksempel en ultra-lav-effekts grensesnitt for eksterne sensorer. Dette er ypperlig for innsamling av data mens resten av systemet er i hvilemodus. Mikrokontrolleren kommer i to versjoner:

- LAUNCHXL-CC1350US, som er optimalisert for bruk på radiofrekvens 915 MHz.
- LAUNCHXL-CC1350EU, som er optimalisert for bruk på radiofrekvens 868 MHz.

Tekniske beskrivelser og design av SimpleLink CC1350 Lp er utarbeidet av det produsent TI [17].

### 3.5.4 CC2650 Launchpad

I likhet med SimpleLink CC1350 Lp er SimpleLink CC2650 Lp en trådløs MCU som har som formål å tilby svært lavt effektbruk som trådløskommunikasjons enhet. Den er i hovedsak identisk til CC1350 Lp, men opererer med 2.4 GHz som radiofrekvens som gir den en god del kortere rekkevidde [18].

## 3.6 Programvare

### 3.6.1 VMWare Workstation

VMWare Workstation er et test- og utviklingsmiljø fra VMWare Inc. som gjør det mulig å kjøre en eller flere virtuelle maskiner (VM) fra et skrivebord på en enkelt fysisk datamaskin. VMWare Workstation er kompatibelt med Microsoft Windows og Linux baserte operativ systemer og de virtuelle maskinene kan kjøre en rekke versjoner av operativ systemene Microsoft Windows, Linux, BSD og MS-DOS. VMWare Workstation 12.5 er siste versjon, utgitt i 2016 [19].

### 3.6.2 SmartRF Flash Programmer 2

SmartRF FlashProgrammer 2 er en programvare fra Texas Instruments som brukes til å programmere flashminne i Texas Instruments ARM baserte laveffekts RF trådløse mikrokontrollere over et serielt debug grensesnitt. I de fleste tilfeller er dette grensesnittet av typen JTAG [20].

### 3.6.3 GitHub

GitHub er en fritt tilgjengelig web basert utviklingsplattform for blant annet programmering, applikasjonsutvikling og generell koding. Hver bruker har et eller flere oppbevaringssted, *repository*, der hvert prosjekt er organisert. Dette gjør det mulig å dele, evaluere og samarbeide om hverandres prosjekter uavhengig av fysisk lokasjon [21].

### 3.6.4 Python

Python er et fritt tilgjengelig objekt orientert høynivå programmeringsspråk. Pythons syntaks blir av mange oppfattet som enklere enn andre programmeringsspråk grunnet utviklernes fokus på lesbarhet. Python støtter en rekke ulike moduler og pakker. Python er utgitt i tre større versjoner med en rekke mindre sub-versjoner for hver enkelt. Siste versjon er Python 3.6 utgitt i desember 2016 [22].

### 3.6.5 C

C er et lavnivå programmeringsspråk som danner grunnlaget for språkene Java og C++. At språket er lavnivå innebærer at det gir brukeren direkte tilgang til datamaskinens minne og flere ulike maskinvarenære operasjoner. På grunn av dette er C et nyttig programmeringsspråk for forskning på og utvikling av mikrokontroller baserte systemer [23].

## 3.7 Operativsystemer

### 3.7.1 Debian

Debian er et fritt tilgjengelig operativ system for datamaskiner. Det bruker Linux-kjernen, en helt fri programvare. Mesteparten av grunnlaget for Debian kommer fra GNU prosjektet, et av de tidligste fritt tilgjengelige operativ systemene [24], og blir derfor ofte referert til som Debian GNU/Linux operativ system. Formålet med Debian er å tilby et brukervennlig, fleksibelt, stabilt og sikkert operativ system fritt tilgjengelig for utviklere og entusiaster, som de kan dele kunnskap om og erfaringer fra [25].

#### 3.7.1.1 *Debian 8.7 Jessie*

Debian 8.7 Jessie, utgitt 01.04.2017, er siste versjon av Debian operativ systemet. Det er den syvende oppdateringen av Debian Jessie serien og som de tidligere oppdateringene tar den sikte på å forbedre operativ systemet gjennom diverse feilrettinger. Debian 8.7 Jessie er kompatibelt for en rekke plattformer som for eksempel ett-bretts datamaskiner [26].

### 3.7.2 Contiki

Contiki er et fritt tilgjengelig operativsystem for IoT enheter. Det tilbyr laveffekts internett kommunikasjon og er kompatibelt med både IPv6- og IPv4-protokollen i tillegg til standarder med fokus på lavt effektbruk som for eksempel 6LoWPAN, RPL og CoAP. Contiki applikasjoner er bygget i standard C kode og operativ systemet kan kjøre på en rekke trådløse enheter som for eksempel Texas Instruments CC26xx og CC13xx. Contiki 3.0, utgitt 2015, er siste versjon av Contiki operativsystem [27].

#### 3.7.2.1 Instant Contiki 3.0

Instant Contiki er et fullstendig Contiki utviklingsmiljø. Det er en virtuell Ubuntu Linux maskin som kan kjøres i VMWare og inneholder alle verktøy for Contiki utvikling i tillegg til nødvendige kompilatorer og simulatorer [28].

### 3.7.3 Ubuntu 16.10

Ubuntu er et Linux Debian basert fritt tilgjengelig operativ system. Det er tilgjengelig til bruk på en rekke ulike enheter som for eksempel servere og mobile enheter. Det er av mange regnet som det mest brukervennlige Linux baserte operativ system. Ubuntu 16.10 er den nest siste versjonen av operativ systemet og ble utgitt i oktober 2016 [29].

## 4 Design

### 4.1 Valgt løsning

Sensario AS ga oss spesifiserte krav til oppgaven som må tilfredsstilles gjennom vårt design. Målet har vært å dekke kravene for problemstillingen på en best mulig måte, samtidig som vi vil rette oppgaven mot vårt fagfelt. Komponentene og protokollene som er brukt er forhåndsspesifiserte, men hvordan vi løser oppgaven med tanke på koblinger, programvare og oppsett har vi stått fritt til å velge.

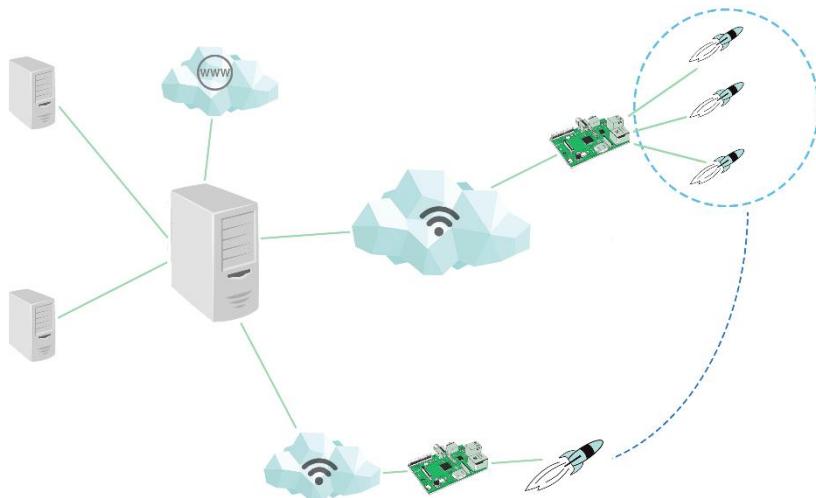
Designet og løsningen vi presenterer er bygd opp av maskinvare og programvare i flere moduler, der hver komponent samhandler med hverandre. Valgt og forhåndsspesifisert design er å ta i bruk:

- 5 stk. BBG
- 15 stk. CC1350 Lp
- 5 stk. bokser.

Slik kan vi danne tre uavhengige nett på hver sine respektive kanaler som danner mesh-nett over 6LoWPAN protokollen. Hvert system består av:

- 1 stk. ip66 gradert boks
- 1 stk. BBG
- 3 stk. CC1350 Lp
- 1 stk. 5V strømforsyning

I alt har vi fem bokser hvor en er forbeholdt som gateway. Dette skal være nok til å dekke store deler av Bergensdalen og danne infrastrukturen for vår IoT-lab



Figur 4: Inforgrafikk

## 4.2 Fysisk design og realisering

### 4.2.1 Komponenter

#### 4.2.1.1 Beaglebone Green

Backbone node-kontroller med 1GHz ARM prosessor, 512MB DDR3 minne og 2x46 GPIO. BBG brukes for å motta debug-data fra nodene, for å flashe nodene med ny programvare og er koblet til internett gjennom TCP/IP. Hver BBG har 5 UART porter hvor 4 av dem kan brukes til å sende og motta data. Av de 2x46 GPIO benytter vi 6 av dem for å styre bakdørene til nodene samt 5V og jord.



Figur 5: Beaglebone Green

#### 4.2.1.2 CC1350 Launchpad

Node som har en oppgitt rekkevidde på 20km med ekstern antenn [30] og benyttes for å danne mesh-nettverket som dataflyten skal sendes over 6LoWPAN protokollen. CC1350 Lp kommer med debug chip XDS110 som er koblet til CC1350-chipen gjennom de midtplasserte koblingene på brettet. Disse gjør at du kan flashe CC1350-chipen med ny programvare gjennom JTAG og få debug-data serielt fra USB-porten. XDS110 er også koblet til en 3,3v spenningsomformer som vi benytter. Av pinnene på CC1350 Lp har vi tatt i bruk 5V, jord, DIO2(TX), DIO3(RX), DIO13(sel) og BPRST(reset).



Figur 6: CC1350 Launchpad

#### 4.2.1.3 **Strøm kilde**

For å levere tilstrekkelig med strøm benytter vi oss av en 5V USB-lader som kan levere 2,1A. Vi må unngå spenningsomformeren til BBG og dette gjør vi ved å stripe USB ledningen slik at vi sitter igjen med 5V og jord. De kobles så direkte til 5V og jord på BBG. Da kan vi parallell koble tre CC1350 Lp til 5V som igjen benytter XDS110 til å transformere spenningen ned til 3,3V. Vi får da en elegant løsning med en USB ledning som dekker strømforbruket til hele systemet.

#### 4.2.1.4 **Koblingsboks**

I og med at nodene skal plasseres ute i vær og vind har vi benyttet en industribasert IP66 gradert koblingsboks fra OBO. Boksen måler 240x190x95 mm har buede kanter og komponentene sitter godt på plass.

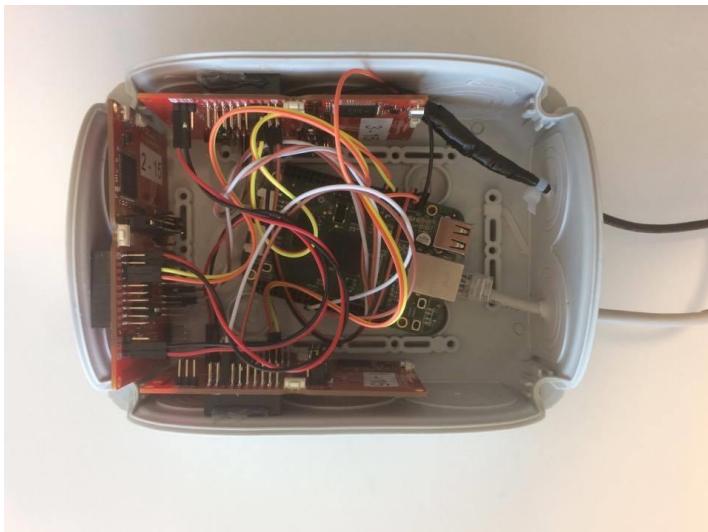


*Figur 7: Hensel koblingsboks*

## 4.2.2 Realisering

### 4.2.2.1 Oppkoblet system

Som en kan se på bildet av boksen [Figur 8: Koblingsboks med hardware] er det ikke mer plass enn hva som nødvendig for vårt system. CC1350 Lp er limt fast til boksen i «toppen» av brettet, og kan således enkelt fjernes om det skulle være nødvendig. BBG blir sikret på plass av borrelås limt med dobbeltsidig teip, den er også enkel å løfte ut for inspeksjon eller utskifting.



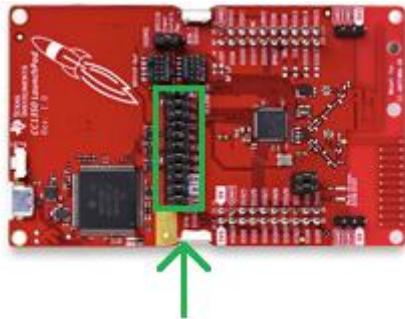
Figur 8: Koblingsboks med hardware

### 4.2.2.2 Fargekoding

Hver port på brettet som er brukt (RX, TX, RST, SEL, 5V og GND) har fått sin egen fargekode. Dette er konsekvent brukt for å lette feilsøking og holde oversikt. Det er blitt brukt standard «han til hun» lasker og «han til han» lasker. Vi har dessverre ikke hatt nok ledninger for å opprettholde en standard fargekode for alle boksene så hver boks varierer.

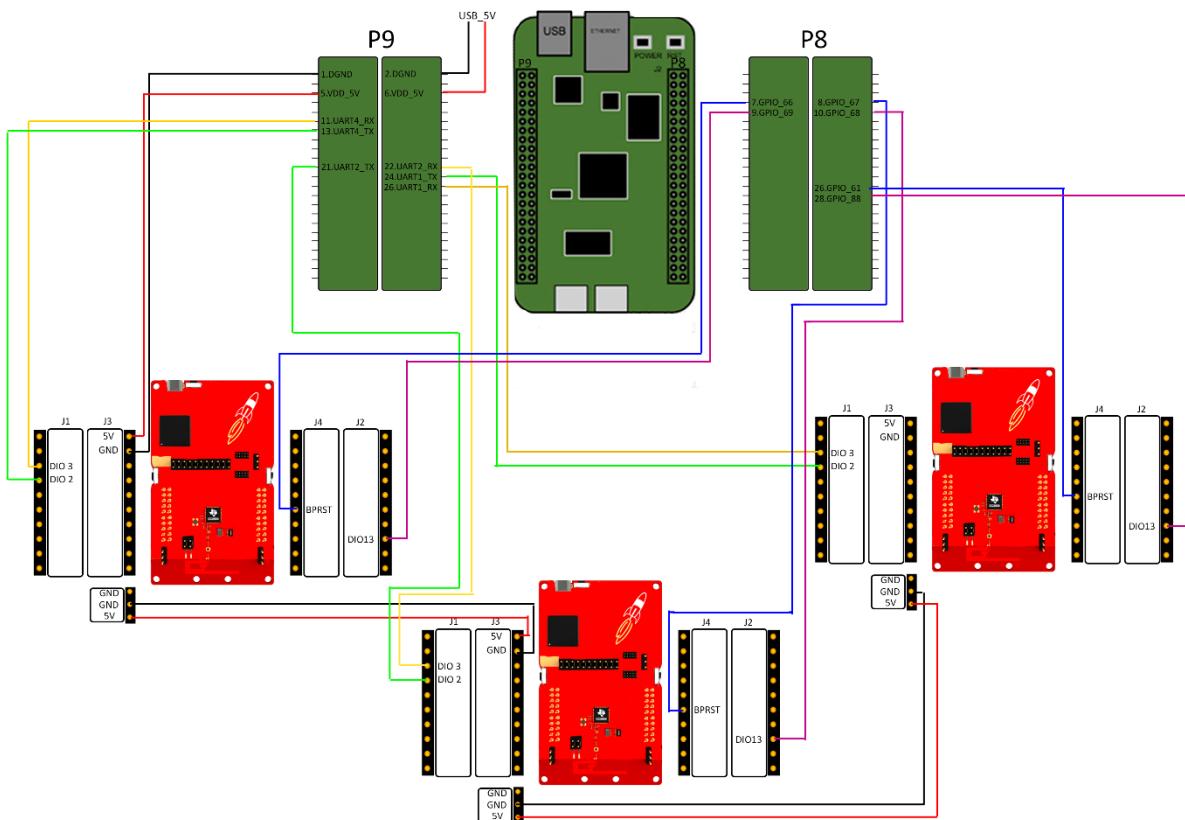
#### 4.2.2.3 Modifikasjoner

Dette gjelder launchpad utgaven av CC1350 som kommer med en «debug-chip» XDS110. Denne muliggjør flashing over JTAG som må gjøres første gang en tar i bruk brettet. Etter du har gjort den første flashingen og fått åpnet bakdøren, skal denne chipen kobles fra brettet. Dette er fordi det ikke sendes data over UART når XDS110 er koblet til. Fjern alle disse «broene» [Figur 9] utenom 5V, 3,3V, GND og RST.



Figur 9: Tilkobling for XDS110, CC1350 Launchpad

#### 4.2.2.4 Koblingsskjema



Figur 10: Koblingsskjema

## 4.3 Oppsett av systemet

Gjennom hele prosessen har det vært svært viktig for oss å ha kontroll over modulene vi skal jobbe med. Et stort antall komponenter, protokoller, skript og binærfiler (binfiler) skal holdes orden på. Vi innså derfor raskt at vi trengte et system for dette. De første steget var å innføre nummerering av all maskinvare og logging av hva som har blitt gjort av endringer. Vi opprettet også en Github som sentraliserte alle filer som skulle inn på hver enkelt BBG. Hver BBG trenger også en del programmer som må installeres manuelt ved hjelp av *apt-get install*. En liste over programmene er lagt til i Appendiks C.

### 4.3.1 Github

Alle kodene og binfilene vi bruker kan du finne på vår github ([https://github.com/Jorgenhoyer/Bachelor\\_v2017](https://github.com/Jorgenhoyer/Bachelor_v2017)) [30]. Github gjør jobben med å oppdatere BBG sømløs i form av git-push og git-fetch kommandoer. Det vil si, at hvis vi gjør en endring i et av skriptene er det enkelt å få de ut til alle nodene. Slik vi har satt det opp nå skal Bachelor\_v2017 finnes på /root til alle BBG.

### 4.3.2 OpenVPN

For å utveksle data fra serveren og ut til nodene trenger vi et program som lar oss kjøre datastrømmen gjennom en sikker TLS-tunnel . BBG 1 til 5 har fått statiske IP adresser som vi kan SSH-kommunisere gjennom (se appendiks C). Vi kan dermed koble alle BBGer til en hvilken som helst ruter og få kontakt.

- BBG1 ssh root@10.0.0.10
- BBG2 ssh root@10.0.0.6
- BBG3 ssh root@10.0.0.22
- BBG4 ssh root@10.0.0.8 (ikke tilkoblet)
- BBG5 ssh root@10.0.0.18

### 4.3.3 UART & I/O PIN

BBG ble valgt av vår oppdragsgiver på grunn av et større antall UART-porter på brettet sammenlignet med andre tilsvarende enheter. Vi bruker 3 av UART-portene som kobling til hver CC1350 Lp i hver boks. En betydelig del av oppgaven vår har gått ut på å fjern-flashe nodene og det er derfor helt kritisk at vi får til å både sende og motta data serielt. Dette løser BBG helt OK, den klarer å sende med baudrate 50 0000 til en node om gangen, men det kan oppstå problemer med lese/skrive rettigheter av ukjente årsaker.

BBG har 2x46 GPIO hvor 65 av de kan operere som digitale I/Os. For å fjernflashe nodene må vi sette 2 signal på CC1350 høy/lav for å åpne bootloader backdoor. Vi bruker 6 av de digitale I/O, 2 til hver node.

#### 4.3.4 Python skript

Alle skript kan kjøres remote over server eller lokalt på BBG. De fleste skriptene har vi skrevet selv, men noen er modifikasjoner av andre sitt arbeid. Der vi har gjort modifikasjoner krediterer vi opphavsmannen.

##### Debug

For logging av debug informasjon benytter vi oss av et debug skript. Vi kjører 3 skript på hver BBG som lytter til hver sin UART port. Skriptet tidsstempler debug informasjonen og lagrer det i en enkel tekstfil for videre behandling.

```
f = open('ser_logg.txt', 'w')
ser = serial.Serial("/dev/ttyS1", baudrate=115200, stopbits=1, parity="N", timeout=1)
```

Figur 11: Debug skript

##### Bakdør

For å åpne bakdør på CC1350 nodene må vi sette 2 signal høy/lav, avhengig av hvordan Contiki filen er «bygget».

```
import Adafruit_BBIO.GPIO as GPIO #import GPIO Library

outPin_RST="P9_12"           #set outPin for Reset to "P9_12"
outPin_SEL="P9_14"           #set outPin for Select to "P9_14"

GPIO.setup(outPin_RST,GPIO.OUT) #make outPin_RST an Output
GPIO.setup(outPin_SEL,GPIO.OUT) #make outPin_SEL an Output

from time import sleep        #so we can use delays
for i in range(0,1):          #loop 1 time
    GPIO.output(outPin_SEL, GPIO.LOW)
    sleep(1)
    GPIO.output(outPin_RST, GPIO.LOW) # Set outPin_RST LOW (Active)

    sleep(1)                   #Pause
    GPIO.output(outPin_RST, GPIO.HIGH) #Set outPin_RST HIGH (Inactive)
    sleep(1)
    GPIO.output(outPin_SEL, GPIO.HIGH) #Set outPing_SEL HIGH (Inactive)
    sleep(1)                   #Wait
GPIO.cleanup()                 #Release your pins|
```

Figur 12: Bakdør-skript

##### Fjern-Flashing

Et stort bidrag til oppgaven vår er JelmerT på github [32] foreslått av Andreas Urke. JelmerT har laget et skript som kan slette, programmere, verifisere og lese flash-minne på CC13xx & CC26xx System on Chip (SoC). Skriptet automatiserer fjernflashing over UART fra BBG til CC1350 uten å ta i bruk XDS110. Vi har modifisert skriptet med å «bake» inn bootloader bakdør skriptet slik at vi bare trenger å kjøre en kommando i terminalen for å fjernflashe en node.

### Automatiserings skript

For å gjøre systematiserings prosessen enklere for fremtidige brukere av systemet, har vi laget et automatiserings skript som kjøres fra server. Slik vi ser for oss bruken av nettene skal alle kommandoer kjøres remote via server. Skriptet utfører oppgavene i denne orden:

- Tidsstemplar og lagrer gammel binær fil som skal overskrives på server
- Kopierer og flytter ny binær fil til BBG
- Stopper debug skript på BBG
- Flasher noder med ny binær fil via BBG
- Starter debug skript på BBG

### Border router skript

For å starte samtlige border routere har vi laget et skript som kjøres fra server. Dette skriptet løser et problem med å starte border router direkte fra BBG der systemet kan låse seg.

### Debug innsamling

Dette skriptet samler all debug innformasjon som er lagret på BBG fra server. Skriptet henter tekstfilene og tidsstemplar den.

### 4.3.5 Server

De 5 BBG styres fra en server levert av Digital Ocean, denne er vår «vei» inn til nodene. Alle kommandoer og skript kan kjøres sentralt her.

### 4.3.6 Contiki OS

Alle nodene våre kjører Contiki OS som er et operativsystem for IoT. Det finnes en del modifikasjoner og endringer vi har gjort som er beskrevet i detalj i appendiks C. En oversikt over hva systemet vårt kjører beskrives her.

### Drivere

CC1350 drivere som ligger på github [31]. Dette er en pull-request, som betyr at den ikke er verifisert enda. For vårt formål har den fungert utmerket og virker stabil.

### CC26xx-web-demo

Denne eksempel filen tar i bruk protokoller som COAP, 6LBR og 6LoWPAN. Det viktigste for vår oppgave er at den inneholder 6LoWPAN protokollen, er enkel å modifiseres til ønsket funksjonalitet og støtter mesh/star/extended-star. Vi har endret kanalvalg, lagt til utskrift av RSSI, åpnet bakdør og endret debug-flag.

### RPL-border-router

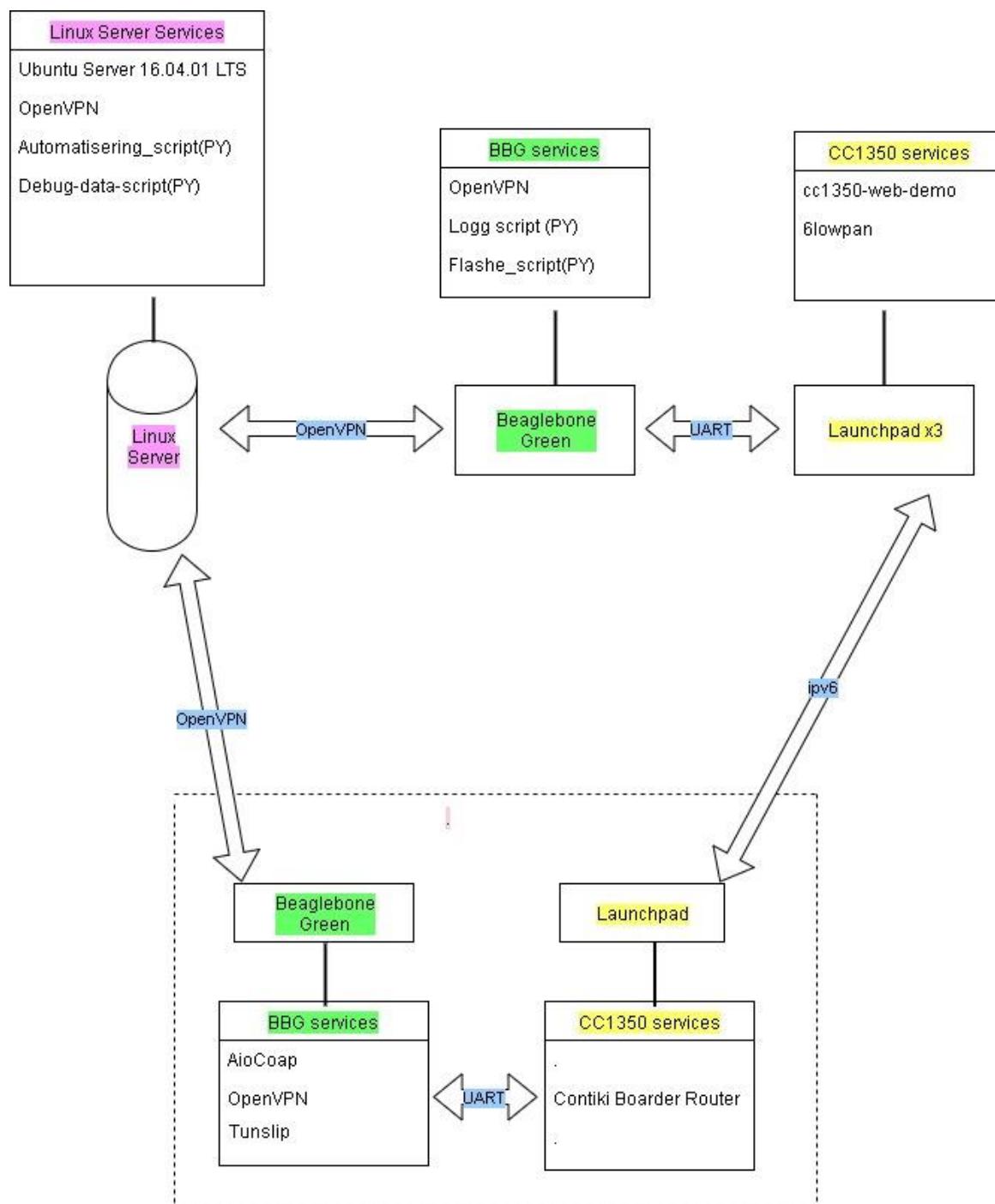
En enkel border router demo som inneholder bare det aller mest nødvendige. Dette er en gateway løsning som ruter trafikken fra 6LoWPAN nettet. Stabilitets endringer er beskrevet i appendiks C.

## SLIP-radio

Vi har lastet ned Contiki på BBG4 som er koblet til de 3 border routerne. Slik kan vi kjøre SLIP-radio på samtlige border routere. Dette programmet ligger som et verktøy i Contiki.

### 4.3.7 Oversikt over protokoller

Hver del av systemet vårt kjører en del forskjellige protokoller og skript. De viktigste er ramset opp i systemarkitekturen.



Figur 13: Protokollskjema

## 4.4 Valg

Selv om vi har hatt rammer å forhold oss til har vi og så vært nødt å ta en del valg med tanke på utførelse, design og hensikt.

### 4.4.1 Bokser

Eivind Helle er utdannet elektriker og har i den sammenheng jobbet mye med komponenter som skal støv, støt og vannsikres. Vår boks ble valgt med tanke på at den er:

- Enkel å feste (følger med skruer).
- Vann, støv og støtsikker
- IP66
- Nipler som opprettholder IP66 krav ved utføring av strøm, nettverkskabel og antenner.
- God tilkomst.
- Grei pris.

### 4.4.2 UART

BBG kommer med 1 USB tilkobling på brettet som støtter dataoverføring til CC1350. Dette er en enkel løsning som krever lite ressurser, men vi mangler da 2 USB tilkoblinger for vår oppgave. En mulighet er å løse dette ved hjelp av en USB-hub. Dette er en forgrening som plugges i USB og gir oss x antall USBer, men et av problemene som oppstår da er tildeling av port når BBG restartes eller slås på. Vi trenger kontroll på hvilken USB port som brukes av tildelt node, dette er alfa omega. Løsningen ble å bruke UART på GPIO. Dette krever flere koblinger og er teknisk sett en mer krevende løsning, men gir oss kontroll over hvilken node som kjører på hvilken port uavhengig av restart eller oppstart.

### 4.4.3 OS

Det finnes alternativer som Angstrom, SolarNode og en rekke andre OS for BBG. Valget vårt falt på Debian Jessie som er et veldig dokumentert og enkelt OS å jobbe med for oss.

### 4.4.4 Ubuntu 16.04

Valgt på grunn av vår forhåndskunnskap med Linux. Hvordan Ubuntu brukes er veldig dokumentert gjennom en rekke guider på internett som vi har brukt for å løse problemer underveis.

### 4.4.5 OpenVPN

For å nå BBG fra serveren trengte vi en enkel og fleksibel løsning som fungerer uansett hvilken ruter vi kobler BBG til. OpenVPN har mulighet for et stort antall koblinger og kan enkelt skaleres uten problemer.

#### 4.4.6 HW

Sensario AS bruker CC1310 på sine egenproduserte kort og det var ytret et ønske om at vi skulle bruke samme chip til vår oppgave. I diskusjon sammen med Andreas Urke falt valget vårt på CC1350 Lp. Grunnen var at den har blåtann som kan tilby en rekke funksjoner til videre utforskning. Det er ellers ikke noe annet som skiller de CC1350 Lp fra CC1310 Lp.

#### 4.4.7 Antenne

For å koble antenner til nodene trenger vi 15 stykker JSC til SMA kobling og eksterne antenner. Dette vil koste oss i overkant av 3500,- (ikke medregnet eksterne antenner). Vi har derfor ikke kunnet forsvere dette i budsjettet og har ikke blitt kjøpt inn.

#### 4.4.8 Contiki bakdør

I Contiki kan en velge om bakdøren skal åpnes ved aktivt høyt eller aktivt lavt signal. Dette har ikke noen betydning for maskinvaren, men for hvordan vi skriver bakdør skriptet vårt. Etter en del feilsøking hvor CC1350 Lp ikke ville resette seg korrekt etter åpning av bakdør og flashing, fant vi ut at BBG har problemer med å resette GPIOene. Dette løste vi ved å velge aktiv høy bakdør i Contiki.

#### 4.4.9 BeagleBone Black wireless

Under oppgaven sin sluttfase fikk vi problem med BBG4 og måtte gå til innkjøp av en ny. Vi valgte da å kjøpe inn BeagleBone Black wireless(BBBW) som erstatning. Denne kommer ikke med ethernetport og må kobles til internett via WiFi. Dette er ønskelig for oss ettersom det da kreves mindre logistikk med kabler for internett-tilkobling. Denne fungerte dessverre ikke på HVL grunnet eduroam sin strenge sikkerhetskrav for tilkobling og har derfor ikke blitt benyttet.

## 5 Testing

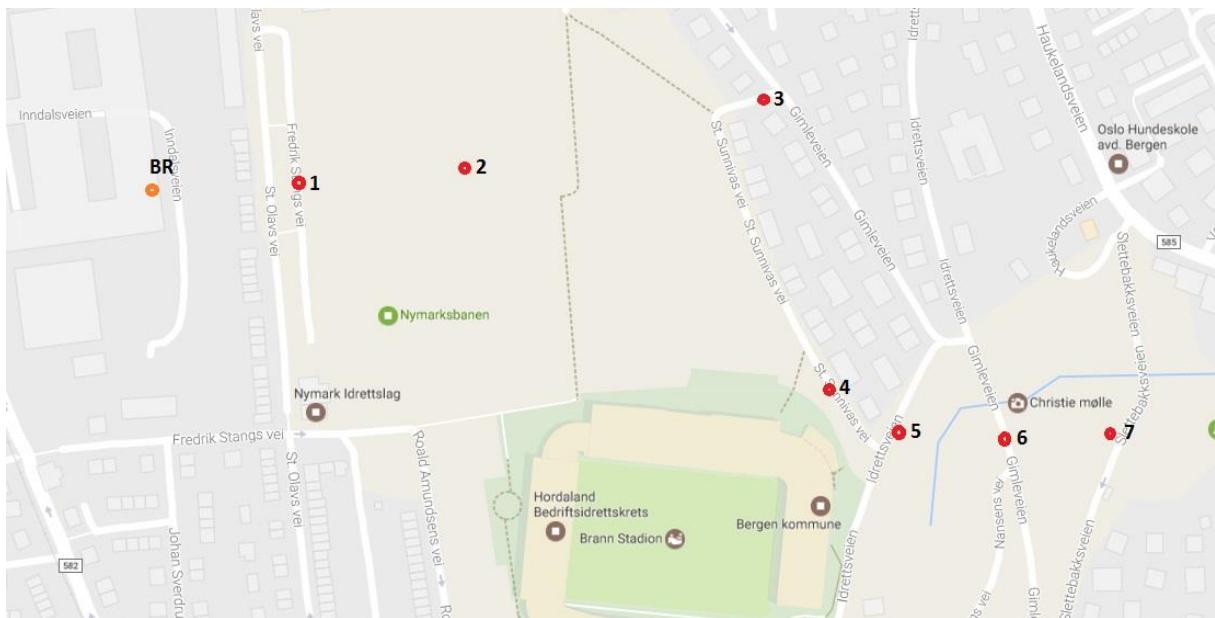
### 5.1 Rekkeviddetest 1

#### 5.1.1 Hensikt

Hensikten med denne testen var å sjekke hvor lang rekkevidde vi kan oppnå mellom to stk. CC1350 Lp uten eksterne antenner. Vi prøvde å teste under så optimale forhold som mulig der ingen av mikrokontrollerne var innkapslet og med de integrerte antennene pekende rett mot hverandre. Vi forventet en rekkevidde på ca. 500m.

#### 5.1.2 Testoppsett

Testen ble gjennomført fra østsiden av HVL på Mikkien terrasse i andre etasje mot Nymarksbanen og Christieparken. Vi sendte en person ut med en CC1350 som kjørte siste utgave av Contiki som var koblet til en batteripakke for å få strøm. Informasjon om signal styrke (RSSI) ble lest av fra en pc som var koblet til en CC1350 Lp. Denne kjørte siste utgaven av border router for Contiki. Vi sjekket også temperaturen til noden som vi flyttet rundt for å sjekke at det faktisk ble overført data. Alle distanser ble bestemt ved hjelp av Google Maps. Oppsett for både border router og node finner man i appendiks C.2.



Figur 14: Testområde, rekkeviddetest 1

### 5.1.3 Observasjoner

Test nr:	Plassering node:	RSSI (dBm):	Distanse luftlinje (m):
1	Parkeringsplass, vest for bane	-75	85
2	Grusbane, øst for bane	-80	200
3	Bilvei, st sunnivas vei	-79	360
4	Hjørne, nordøst, brannstadion	-87	400
5	I bakken ovenfor	-81	450
6	I bakken høyere opp	-87	530
7	Toppen av parken/haugen	-87	600

Tabell 1: Observasjoner, rekkeviddetest 1

Som vi kan se på målingene varierte ikke signalstyrken særlig mye. Målingene ble gjentatt flere ganger på de samme plassene og vi leste av samme signalstyrke gjentatte ganger. Vi hadde et avvik som ble gjort i måling nummer 4. dette tror vi var fordi det ikke var like god sikt mellom border routeren og noden som på de andre testene. Noden ble ikke slått av mellom målingene og holdt stort sett kontakten mens den var i bevegelse. Når vi nådde 600 meter kunne vi ikke bevege oss lengre uten å miste fri sikt til border routeren. Noden var i tillegg plassert mellom flere trær og hadde derfor ikke helt fri sikt som på de andre målingene. Når vi forsøkte å bevege noden enda lengre unna fikk vi verken målt RSSI eller temperatur. Dette var nok fordi noden ikke lenger hadde fri sikt til border routeren. Vi observerte også at det tok lengre tid før border router og node koblet seg sammen jo lengre distansen mellom de var.

### 5.1.4 Diskusjon

I ettertid ser vi at vi skulle haft et større område å teste på. For å finne den maksimale avstanden vi kan ha mellom en node og en border router må vi finne et område med bedre plass og fri sikt. Testene kunne også vært gjennomført både med og uten innkapsling slik at det er mulig å observere hvor stor påvirkning innkapslingen har på signalstyrken. Videre hadde det vært interessant å se om flere noder ville påvirket signalstyrken til hverandre. I senere tester skulle vi også likt å se hvor lang distanse vi klarer å oppnå når nogene gjør hopp fra en node til en annen.

### 5.1.5 Konklusjon

Vi kom opp i en avstand mellom noden og border routeren som var lengre enn forventet, men vi planlegger å teste enda lengre distanser på et annet område med bedre sikt.

## 5.2 Rekkeviddetest 2

### 5.2.1 Hensikt

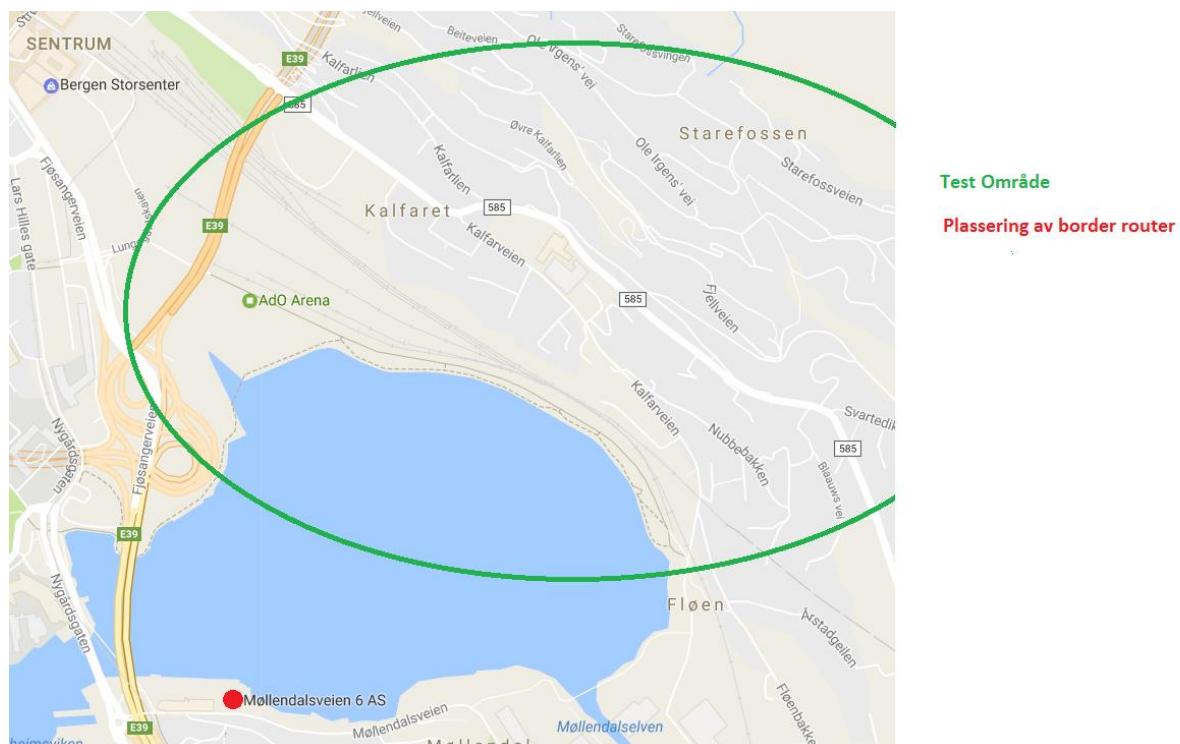
Tidligere hadde vi en utvendig test der vi gjorde et forsøk på å måle den maksimale rekkevidden til CC1350 Lp. I den forrige testen var distansen med fri sikt mellom noden og border routeren ikke tilstrekkelig for å registrere målinger over 600m. Det er derfor nødvendig å finne et testområde for å kunne fastslå:

- Maksimal distanse mellom en node og border router uten innkapsling
- Maksimal distanse mellom en node og border router med innkapsling
- Om to noder kobler seg til hverandre og danner mesh-nettverk

Det er viktig for oss å finne ut hvilken påvirkning innkapsling har på signalstyrken hvis boksene skal plasseres ut forskjellige steder i Bergensdalen. Slik boksene er konstruert bruker vi PCB antennene som er montert på hver CC1350 Lp av leverandøren.

### 5.2.2 Testoppsett distanse

Området vi valgte denne gang var ved store Lungegårdsvann i Bergen. Vi plasserte border routeren på takterrassen til HVL i Møllendalsveien. Nodene ble flyttet rundt Lungegårdsvannet og oppover i fjellsiden. Alle nodene kjørte siste utgave av Contiki. Border routeren var koblet til en PC som kjørte siste utgaven av tunslip6. Oppsett for både border router og node finner man i appendiks C.2.



Figur 15: Testområde, rekkeviddetest 2

### 5.2.3 Observasjoner rekkeviddetest

Alle avstander ble bestemt ved hjelp av Google Maps. RSSI og temperatur ble avlest fra CoAP server som kjørte på PC.

Test nr:	Tidspunkt:	Plassering av node:	RSSI uten innkapsling(dBm):	RSSI med innkapsling(dBm):	Distanse(m):	Temperatur uten boks(°C):	Temperatur med boks(°C):
1	11:42	Regnhytten	-83	-89	785	19	13
2	12:00	Sør for Regnhytten	-86	ingen måling	790	20	ikke målt
3	12:23	Gressplen øst for ADO arena	-78	-81	630	22	12
4	12:35	Basketball bane ADO arena	-90	ingen måling	630	20	ikke målt
5	12:07	Washingtonsvei 6	-91	-91	1100	ikke målt	ikke målt
6	11:45	Fjellveien 133	-88	ingen måling	1000	ikke målt	ikke målt

Tabell 2: Observasjoner, rekkeviddetest 2

I den første testen vi gjorde koblet noden seg fint til border routeren og vi fikk målt RSSI både med og uten innkapsling. Det tok ikke lang tid før noden og border routeren koblet seg sammen og vi fikk målt temperaturen i begge tilfellene.

Test nummer 2 ble utført kun et lite stykke fra test nummer en. Av erfaring fra andre tester visste vi at signalstyrke i underkant av -90dBm ville bli for lav til å få overført data fra noden. Forventningen var at noden fortsatt kom til å klare tilkobling med og uten innkapsling. Uten innkapsling klarte noden helt fint å koble seg opp mot border routeren, men med innkapsling koblet den seg ikke opp. Vi prøvde å resette noden for å se om det hjalp, men dette hadde ingen effekt. Det fungerte heller ikke å gi den 5 minutter for å se om den trengte lengre tid enn for tilfellet uten kapsling.

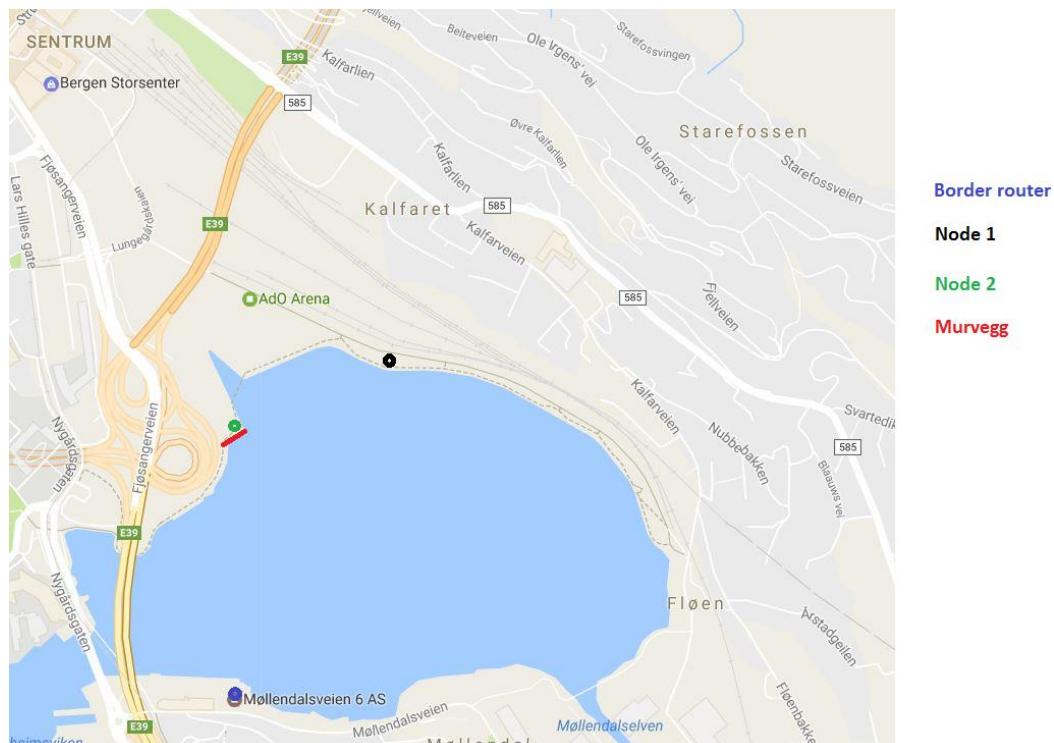
I test nummer 3 flyttet vi noden et godt stykke nærmere border routeren for å sjekke om vi hadde samme tap med innkapsling her som vi hadde i test nummer 1. Her observerte vi at tapet var litt lavere etter innkapslingen var satt på. Noden og border routeren koblet seg til hverandre nesten umiddelbart og vi fikk lest av temperatur i begge tilfellene.

Den fjerde testen ble gjennomført rett ved siden av ADO arena. Her kan vi se at i luft linje er det like langt. Vi hadde forventet at test nummer 3 og 4 skulle gi samme resultat men det gjorde de ikke. Uten innkapsling målte vi en RSSI på -90dBm. Vi forsøkte flere ganger å peke antennen på både border routeren og node rettet mot hverandre, men det ga oss ikke noe bedre resultat. Så fort noden og border routeren ble innkapslet forsvant signalet.

Test nummer 5 og 6 ble gjennomført mellom de andre testene ved hjelp av en annen node. Her var målet å sjekke hvor langt unna border routeren noden kunne stå før de mistet kontakt. I disse testene kunne vi se at border routeren fikk kontakt med nodene begge ganger, men vi fikk ikke lest av temperaturen. Vi ser igjen at når vi nærmer oss -90dBm får vi ikke overført data.

### 5.2.4 Testoppsett for hopp

I denne testen hadde vi det samme oppsettet for noden og border routeren som i distanse testene. Det eneste som endret seg her var hvor vi plasserte noden som skulle ha kontakt med border routeren, og noden som skulle gjøre et hopp via den første noden. Noden som skulle gjøre et hopp ble plassert bak en murvegg slik at den kun hadde fri sikt til var den andre noden. Distansen mellom border routeren og node 1 var på 630m og avstanden mellom de to nodene var 300m.



Figur 16: Testoppsett for hopp, rekkeviddetest 2

### 5.2.5 Observasjoner for hopp

Test nr:	Tidspunkt:	Plassering av node1:	Plassering av node2	RSSI node1 (dBm):	RSSI node2(dBm):	Temperatur node 1(°C):	Temperatur node 2(°C):
1	13:05	Gressplen øst for ADO arena	Ved brannstasjon	-78	-89	19	16
2	13:07	Gressplen øst for ADO arena	Ved brannstasjon	-80	-84	22	16
3	13:09	Gressplen øst for ADO arena	Ved brannstasjon	-88	-86	22	16

Tabell 3: Observasjoner for hopp, rekkeviddetest 2

Det første vi sjekket var at border routeren og node1 fikk kontakt. Etter dette skrudde vi på node2 for å se om vi fikk kontakt med den på border routeren. Den første noden koblet seg fort opp til border routeren uten problemer og vi fikk både målt RSSI og temperatur. Vi slo så på node2 for å sjekke at denne koblet seg til border routeren. Etter litt tid koblet node2 seg opp mot border

routeren. Vi var ikke sikker på om node2 gjorde et hopp eller om den fikk direkte kontakt med border routeren. For å teste dette slo vi av node1 for å se om node2 også koblet seg fra. Nesten umiddelbart etter at vi slo av node1 forsvant også node2. dette forsøket gjorde vi 3 ganger for å forsikre oss om at node2 faktisk var avhengig av node1.

### 5.2.6 Diskusjon

Vi skulle gjerne hatt noen antenner koblet til hver node. Vi ser for oss at hvis boksene skal plasseres rundt i Bergensdalen må vi kunne ha større rekkevidde mellom hver node enn 800m. Etter hva vi har lest i dokumentasjonen til Texas Instruments skal man kunne oppnå avstander på opp mot 20 kilometer med ekstern antenn. Videre ser vi at testene skulle vært utført over lengre tid for å sjekke stabiliteten og signalstyrken over tid. En av observasjonene vi ikke skrev ned var hvor lang tid nodene brukte på å koble seg opp mot border routeren. Hvis vi hadde skrevet ned denne målingen kunne vi potensielt sett, fastslått en sammenheng mellom oppkoblingstid og distanse.

Siden resten av nodene våre sto inne og kjørte en datainnsamlingstest på HVL, fikk vi ikke muligheten til å sjekke hvilken påvirkning det har å kjøre tre nett på tre kanaler parallelt. Det ideelle hadde vært om vi hadde fått alle boksene våre plassert ut på forskjellige steder, mens de var koblet til serveren.

Vi er også usikker på hvor mye støy det er i testområdet. Det hadde vært greit å gjenta samme test på flere lokasjoner for å se hvor mye nærliggende bygningsmasse har å si for måleresultatene. Dette kunne vært nyttig for å finne den ideelle lokasjonen til den endelige IoT-labben.

Den siste testen vi gjorde var en hopptest. Dette var av ren nysgjerrighet for å finne ut om nodene faktisk gjorde hopp mellom hverandre eller om de bare koblet seg direkte til en border router. Vi skulle gjerne ha satt opp en test med flere noder der de gjorde opp til flere hopp. Dette hadde også vært til hjelp for å finne en passende plassering for IoT-labben. Da kunne vi observert om det var forskjell på distanser man kan ha mellom node til node og node til BR.

### 5.2.7 Konklusjon

Når de ikke er plassert i innkapsling kan de se ut som CC1350 Lp med PCB-antenner har en rekkevidde på rundt 800m. Når man legger CC1350 Lp i innkapsling, synker signalstyrken og dermed rekkevidden. Videre kom vi fram til at ved rundt -90dBm tar det lang tid for CC1350 Lp å koble seg opp mot border routeren og vi opplevde at vi ikke fikk hentet temperatur data fra noden.

## 5.3 Stabilitetstest

### 5.3.1 Hensikt

I denne testen skal vi se på hvordan systemet oppfører seg over tid, det som skal testes er:

1. At BBG holder kontakten med server gjennom hele perioden.
2. At skriptene som skal samle inn debug data fungerer hele perioden.
3. Hva som skjer hvis BBG mister kontakt med server.

### 5.3.2 Testoppsett

Boksene ble plassert på omsorgslabben til HVL i Møllendalsveien 6. Dette ble gjort fordi de har et nett som ikke er regulert av IT avdelingen på HVL. Her skal de plasseres i noen dager før vi henter dem. Alle boksene ble koblet til en nettverks switch ved hjelp av patche-kabler. De blir tilført strøm av 4 separate 230v til USB-adapttere (5V).

### 5.3.3 Observasjoner

#### Kontakt

Vi sjekket at boksene var koblet til serveren hver ukedag i testperioden. Dette gjorde vi ved å sjekke log filen til OpenVPN fra serveren. Her så alt helt fin ut frem til dag 5. Klientnummeret samsvarer med nummeret vi har gitt hver BBG.

```
Mon May 15 10:35:22 2017 us=354789 client3/51.174.18.178:39300 [client3] Inactivity timeout (--ping-restart), restarting
Mon May 15 10:35:22 2017 us=355317 client3/51.174.18.178:39300 SIGUSR1[soft,ping-restart] received, client-instance restarting
Mon May 15 10:35:22 2017 us=355776 TCP/UDP: Closing socket
Mon May 15 10:35:30 2017 us=839463 client5/51.174.18.178:49098 [client5] Inactivity timeout (--ping-restart), restarting
Mon May 15 10:35:30 2017 us=839527 client5/51.174.18.178:49098 SIGUSR1[soft,ping-restart] received, client-instance restarting
Mon May 15 10:35:30 2017 us=840027 TCP/UDP: Closing socket
Mon May 15 10:35:30 2017 us=916256 client2/51.174.18.178:41576 [client2] Inactivity timeout (--ping-restart), restarting
Mon May 15 10:35:30 2017 us=916330 client2/51.174.18.178:41576 SIGUSR1[soft,ping-restart] received, client-instance restarting
Mon May 15 10:35:30 2017 us=916789 TCP/UDP: Closing socket
```

Figur 17 OpenVPN server logg. frakobling av client

På omtrent samme tidspunkt koblet client2, 3 og 4 seg fra serveren. Vi kan lese i loggfilen til OpenVPN at litt før dette hadde alle 3 clientene kontakt med serveren.

```
Mon May 15 10:02:22 2017 us=234668 client2/51.174.18.178:41576 VERIFY OK:
Mon May 15 10:02:22 2017 us=234858 client2/51.174.18.178:41576 VERIFY OK:
Mon May 15 10:02:22 2017 us=404572 client5/51.174.18.178:49098 VERIFY OK:
Mon May 15 10:02:22 2017 us=404749 client5/51.174.18.178:49098 VERIFY OK:
```

Figur 18 OpenVPN serverlogg. siste kontakt mellom server, client2 og 5

```
Mon May 15 10:02:10 2017 us=303855 client3/51.174.18.178:39300 VERIFY OK:
Mon May 15 10:02:10 2017 us=304037 client3/51.174.18.178:39300 VERIFY OK:
```

Figur 19 OpenVPN serverlogg. siste kontakt mellom server og client3

Vi vet ikke hva som har skjedd her. Selv om de 3 andre klientene koblet seg fra fortsatte client1 å fungere som den skulle. Etter en restart av BBG2, 3 og 5 koblet de seg til serveren igjen og alt fungerte som forventet.

### Innsamling av debug data

Vi startet skriptene for innsamling av debug-data i starten av testperioden. Her kan vi se at debug-skriptet starter å samle inn data så fort det er slått på.

```
2017-05-10 13:28:45
RPL: rpl_process_parent_event recalculate_ranks
```

*Figur 20 Første linje lagret i debug logg*

```
2017-05-11 09:52:48
RPL: Sending prefix info in DIO for aaaa::

2017-05-11 09:52:48
RPL: Sending unicast-DIO with rank 256 to fe80::212:4b00:e01:4801

2017-05-11 09:52:48
RPL: MOP 2 OCP 1 rank 256 dioint 20, nbr count 2

2017-05-11 09:52:48
RPL: nbr 5 128,
```

*Figur 21 Siste linje lagret i debug logg nett1*

Etter cirka 20 timer sluttet debug skriptet å lese av noe data fra nett1. Dette skjedde på alle 3 nodene. Da vi åpnet boksene etter testen kunne vi se at nodene fortsatt hadde strøm men det virket som om de fikk et reset signal. Det eneste som er koblet til reset pinnene på CC1350 er ledninger fra BBG som skal styre reset pinnene. Siden dette skjedde i samtlige bokser mistenker vi at det kan være et problem med GPIO innstillingene på BBG.

```
2017-05-16 15:09:52
RPL: Sending prefix info in DIO for aaaa::

2017-05-16 15:09:52
RPL: Sending a multicast-DIO with rank 256
```

*Figur 22 utdrag fra debug logg til nett2*

```
2017-05-16 15:09:36
RPL: nbr 128    128,    128 =>   256 -- 16 fp (last tx 0 min ago)

2017-05-16 15:09:36
RPL: nbr 134    384,    142 =>   526 -- 16 f  (last tx 2 min ago)
```

Figur 23 utdrag fra debug logg til nett3

Selv om boksene mistet kontakt med serveren fortsatte de å logge debug data fra nett 2 og nett 3. Vi kan se at tidsstempelen stemmer overens med når vi avsluttet testene. Vi observerte flere steder at det skjer opp til 5 hendelser innenfor samme sekund. Ut i fra denne observasjonen kunne vi se at vi bør øke oppløsningen på tidsstempelen.

### Bortfall av nett

Når boksene blir frakoblet nettet kan vi se at de ikke kobler seg opp igjen som de skal. Her er vi usikker på om det er problemer med nettet vi koblet oss til eller OpenVPN. Slik som OpenVPN er konfigurert skal den hvert 5. minut se om den får kontakt med serveren.

Vi startet hver BBG på nytt og sjekket at de koblet seg til serveren. Videre prøvde vi å fjerne nettverkskabelen og plugge den inn igjen. Kort tid etter den var koblet til igjen fikk den kontakt med serveren. Vi gjentok denne testen flere ganger med lengre og lengre tid mellom frakopling til tilkobling. På det lengste, ventet vi over 3 timer før vi koblet nettverkskabelen tilbake og det fungerte fremdeles som forventet.

### 5.3.4 Diskusjon

Etter denne testen ser vi at vi må gå over alle innstillingene i OpenVPN for å sjekke om den er feil konfigurert. Videre har vi sett etter oppdateringer for OpenVPN, men vi kjører siste utgave på både serveren og på hver BBG. Vi har byttet lokasjon på alle boksene for å se om det samme problemet oppstår på nytt eller om dette var noe som skjedde en gang. Vi har i ettermiddag snakket med administrator på omsorgslabben og han kunne bekrefte at de hadde en switch som hang seg samtidig som vi mistet kontakt med boksene. Dette er nok grunnen til at det oppsto problemer i testen. Nettet er satt opp igjen hjemme hos Eivind for å se om vi opplever de samme problemene.

Samtlige noder på nett1 sluttet å virke etter kort tid. Vi vil derfor videre jobbe med å finne ut hva som gjør at disse ikke fungerer som de skal.

### 5.3.5 Konklusjon

Boksene får kontakt med serveren når de blir koblet til strøm og internett. Innsamling av debug-data fra nett2 og nett3 fungerer som det skal, men vi har problemer med nett1. Ved bortfall av internett-tilkobling fortsetter boksene å samle inn debug data. Når boksene får internett igjen, kobler de seg automatisk til serveren.

## 5.4 Samlet systemtest

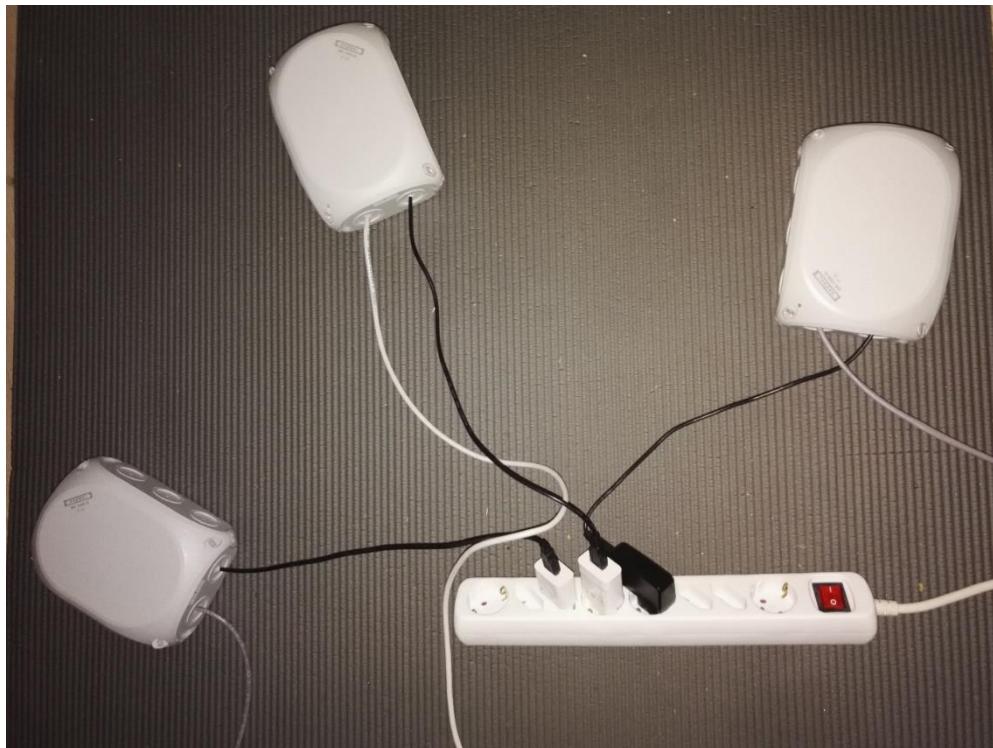
### 5.4.1 Hensikt

I denne testen skal vi teste systemet i sin helhet. Dette er viktig for undersøke om vi har tilfredsstilt alle kravene i oppgavebeskrivelsen. Videre vil denne testen hjelpe med å kvalitets sikre brukerdokumentasjon. I denne testen skal vi teste:

- Hvor enkelt det er å installere systemet.
- At man kan starte border router og SLIP-radio fra serveren.
- At skriptene for flashing av hvert nett fungerer.
- At skriptene for å hente debug log for hvert nett fungerer.

### 5.4.2 Testoppsett

Testen ble gjennomført hjemme hos Eivind. Det kobles opp 2 bokser med tre noder i hver og en boks med 3 border rutere. Distansene mellom hver boks vil være på ca. en halv meter. En Ethernet kabel som kommer ut fra hver boks kobles til en ruter som er koblet til internett.



Figur 24 Testoppsett samlet systemtest

### 5.4.3 Observasjoner

#### Installasjon

For å installere systemet skal det være nok å koble strøm til USB kabelen som kommer ut av boksen. Etter dette må man koble Ethernet kabelen til en ruter som er koblet til internett. For å sjekke at alle komponentene i boksen hadde fått strøm åpnet vi lokket og så at alle komponentene lyste grønt.



Figur 25 Boks etter tilkobling av strøm

Når vi hadde sjekket at alle komponentene i boksen lyste grønt logget vi oss inn på serveren for å sjekke at hver BBG koblet seg til via OpenVPN.

ROUTING TABLE			
Virtual Address	Common Name	Real Address	Last Ref
10.8.0.18	client5	88.89.115.217:42760	Tue May 23 19:22:20 2017
10.8.0.6	client2	88.89.115.217:55616	Tue May 23 19:22:14 2017
10.8.0.22	client3	88.89.115.217:56770	Tue May 23 19:22:15 2017

Figur 26 OpenVPN liste over tilkoblede klienter

Her ser vi at alle 3 klientene er tilkoblet serveren.

### Oppstart av border router og SLIP-radio

For å starte opp SLIP-radio måtte vi logge oss inn på client5. Deretter startet vi tunslip6 for alle kanalene.

```
*****SLIP started on ``/dev/ttys2''
opened tun device ``/dev/tun1''
ifconfig tun1 inet `hostname` mtu 1500 up
ifconfig tun1 add aaaa:1/64
aaaa:1: Resolver Error 0 (no error)
ifconfig tun1 add fe80::0:0:0:1/64
ifconfig tun1

tun1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
             UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:500
             RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

*** Address:aaaa:1 => 0000:0000:0000:0000
Got configuration message of type P
Setting prefix ::

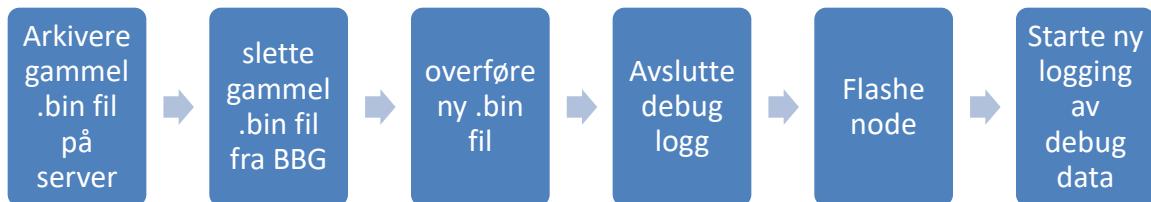
RPL: Node set to be a DAG root with DAG ID ::212:4b00:e01:6782
#A root=130
RPL: Scheduling DIO timer 472 ticks in future (Interval)
RPL: Prefix set - will announce this in DIOs
rpl_set_prefix - prefix NULL
created a new RPL dag
Server IPv6 addresses:
 ::212:4b00:e01:6782
 fe80::212:4b00:e01:6782
```

*Figur 27 Oppstart av tunslip6*

Her kan vi se at SLIP-radioen har startet og fungerer som den skal.

### Flashing av noder

For å flashe alle nodene i et nett bruker vi et skript som utfører flere oppgaver. For å sjekke at alle oppgavene blir utført deler vi skriptet opp i mindre deler.



*Figur 28 skript sekvens for flashing*

```
bachelor@bwsensorlab:~/Upload/Nett2/Gammel$ ls -l
total 512
-rw-rw-r-- 1 bachelor bachelor 131072 May 24 12:07 2017-05-24_12-07.bin
-rw-rw-r-- 1 bachelor bachelor 131072 May 24 12:08 2017-05-24_12-09.bin
-rw-rw-r-- 1 bachelor bachelor 131072 May 24 11:49 cc1350webdemo-ch6.bin
-rw-rw-r-- 1 bachelor bachelor 131072 May 24 10:08 Nett2.bin
```

Figur 29 Arkivering av gammel .bin fil

Her kan vi se at de gamle binær-filene blir arkivert i en mappe som heter Gammel. Vi la til tidsstempel til alle filene slik at det skal være enkelt å finne filen senere.

```
root@bbg2:~/Bin_filer/Nett2# ls
Nett2.bin
```

Figur 30 .bin fil i riktig mappe på BBG

På hver BBG ligger det nå kun en binær-fil til hvert av nettene. Dette viser at skriptet klarer å slette den gamle binær-filen og overføre en ny.

Skriptet avslutter debug loggingen før det starter flashing. Dette er fordi operativ systemet på BBG ikke tillater to programmer å lese og skrive til UART samtidig.

```
Opening port /dev/ttyS2, baud 500000
Reading data from /root/Bin_filer/Nett2/Nett2.bin
Firmware file: Raw Binary
Connecting to target...
CC1350 PG2.1 (7x7mm): 128KB Flash, 20KB SRAM, CCFG.BL_CONFIG at 0x0001FFD8
Primary IEEE Address: 00:12:4B:00:0E:01:0F:05
Erasing all main bank flash sectors
    Erase done
Writing 131072 bytes starting at address 0x00000000
Write 128 bytes at 0x0001FF808
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0xbd3975d8)

Starter debugging av node 2 client 2
```

Figur 31 flashing av node 2 på client2

```
Opening port /dev/ttyS2, baud 500000
Reading data from /root/Bin_filer/Nett2/Nett2.bin
Firmware file: Raw Binary
Connecting to target...
CC1350 PG2.1 (7x7mm): 128KB Flash, 20KB SRAM, CCFG.BL_CONFIG at 0x0001FFD8
Primary IEEE Address: 00:12:4B:00:0E:01:2D:04
Erasing all main bank flash sectors
    Erase done
Writing 131072 bytes starting at address 0x00000000
Write 128 bytes at 0x0001FF808
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0xbd3975d8)

Starter debugging av node 2 client 3
```

Figur 32 flashing av node2 på client3

Her kan vi se at skriptet fullfører flashing på både client2 og 3. Vi kunne også se at de koblet seg til border routeren når de var ferdig. Videre startet skriptet et annet skript som logger debug data og legger det i en tekstfil.

### Innsamling av debug data

```
bachelor@bwsensorlab:~/Debug/nett2$ python debug-data-nett2.py
2017-05-24_13-03
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2016-11-06

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debug-node2.txt                      100%   40KB  40.0KB/s  00:01
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2016-11-06

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debug-node2.txt                      100%   48KB  48.0KB/s  00:00
```

Figur 33 overføring av logg filer fra BBG til server

Her ser vi at skriptet laster ned tekstfilene fra hver BBG. Videre blir de lagret i en mappe som får samme navn som starttidspunktet for innsamlingen av debug filene.

```
bachelor@bwsensorlab:~/Debug/nett2$ ls -l
total 20
drwxrwxr-x 2 bachelor bachelor 4096 May 23 13:38 2017-05-23_13-37
drwxrwxr-x 2 bachelor bachelor 4096 May 24 13:02 2017-05-24_13-02
drwxrwxr-x 2 bachelor bachelor 4096 May 24 13:03 2017-05-24_13-03
-rw-rw-r-- 1 bachelor bachelor 789 May 22 12:18 debug-data-nett2.py
```

Figur 34 tidsstempiling av debug mapper

Hvis man går inn i mappen kan en se at tekstfilene ligger inne og nummerert med hvilke BBG de kommer fra.

```
bachelor@bwsensorlab:~/Debug/nett2/2017-05-24_13-03$ ls -l
total 88
-rw-r--r-- 1 bachelor bachelor 40960 May 24 13:03 bbg2nett2.txt
-rw-r--r-- 1 bachelor bachelor 49152 May 24 13:03 bbg3nett2.txt
```

Figur 35 innhold i debug mappe

Vi gjentok testene for de to andre nettene og fikk samme resultat.

#### 5.4.4 Diskusjon

Vi ser at vi skulle hatt flere bokser koblet til serveren, men Eivind hadde ikke flere tilkoblinger i ruterne hjemme. Videre burde vi hatt en test der boksene var koblet opp på forskjellige steder og var tilkoblet forskjellige nett. Dette får vi ikke gjennomført siden vi ikke har to lokasjoner som er nærmest nok til at nodene får dannet et nett.

#### 5.4.5 Konklusjon

Installasjonen til systemet er enkel og krever ikke mer enn strøm og internett. Alle skriptene gjorde jobben sin og fungerte som forventet.

## 6 Resultater

Fra rekkeviddetesten har vi klart å finne den maksimale avstanden vi kan ha mellom noder og border router når CC1350 Launchpadene kjører Contiki. Vi kom frem til at den maksimale avstanden var rundt 800m uten innkapsling og litt kortere med innkapsling. Videre kom vi frem til at ved en signalstyrke på -90dBm sluttet nettet å overføre data, men det var fortsatt kontakt mellom node og border router. En annen observasjon vi gjorde var at nodene brukte lengre tid på å koble seg til border routeren desto lengre distansen mellom de var. Vi observerte også at nodene gjør hopp via andre noder.

Stabilitetstesten viste at OpenVPN fungerte som det skulle. Boksene koblet seg opp til serveren når de ble koblet til strøm og internett. Vi hadde kontakt med boksene igjennom nesten hele testperioden, men grunnet en feil i en nettverks-switchen mistet vi kontakt etter 5 dager. Selv om boksene mistet kontakt med serveren fortsatte de å logge debug data på nett2 og 3. På nett1 har vi et problem med at nodene mottar et reset signal fra BBG etter noen timer.

Den samlede systemtesten viste oss at systemet fungerer som det skal. Installasjonen av boksene er enkel og de krever kun strøm og internett. Skriptet som skal laste opp ny programvare til nodene på hvert nett fungerer og er etter vår mening enkelt å ta i bruk. Fra serveren får man også kontakt med den BBG som kjører SLIP-radioen. Videre er det enkelt å hente all debug-data opp til serveren der de lagres i en tidsstemplet mappe ved hjelp av et skript.

## 7 Diskusjon

### 7.1 Forutsetninger

Under planleggingen av prosjektet har vi sammen med oppdragsgiver laget en omstendelig fremdriftsplan. Denne tok for seg de elementene som måtte på plass og hvilken orden de skulle på plass i. Vi definerte kritisk vei og tidsfrister som skulle overholdes. En risikovurdering ble også utarbeidet. Her diskuterte vi momenter som kan by på utfordringer og problemer. Dette har blitt gjort for å kunne tilpasse tid, ressursbruk og målsetning.

### 7.2 Fremdrift

Vi startet med å systematisk følge opp fremdriftsplanen. Oppgavene som var planlagt ble løst og vi fikk lagt inn bestilling av komponenter som var kritisk for oppgaven. Møter med veileder og personer med teknisk innsikt økte også vår forståelse i en tidlig fase. Det har ikke blitt gjort store endringer underveis, men etter hvert som vi har fått en dypere forståelse, har vi kunnet vært mer fleksible i forhold til planen. Noen steder var ressursbruken for høy og førte til at vi havnet langt foran i fremdrift. Andre steder var den tekniske kompetansen for lav, noe som førte til at det ble brukt mye tid på diverse nettforum og feilsøking. Mye av hensikten med å lage en fremdriftsplan var å belyse og forutse utfordringer som kunne oppstå. I ettertid kan vi se, fra revidert gant-diagram, at vi kunne lagt inn mer tid der kompetansen vår var lav og mindre tid der vi har mer erfaring.

### 7.3 Risiko

I forstudiet trakk vi spesielt frem:

- Bestilling og levering av CC1350 Launchpad.
- Fase 4 i Gant-diagram (Utviding til 3 nett, server, bruk av loggeverktøy).

Andre problemer som har medført risiko for oppgaven:

- Arbeidssted med tilgang til internett-tilkoblede ethernetporter.

#### 7.3.1 Bestilling og levering av CC1350 Launchpad

Det var en del rot med bestillingen av CC1350 Lp i starten av prosjektperioden. Vi fikk utlevert cc2650 Launchpad som er tilnærmet lik CC1350 Lp for å kompensere for tapt tid, men tidligere erfaringer viser det er ytterst sjeldent at alt er likt når det kommer til maskinvare. Det viste seg at CC1350-chipen ikke kan kjøre de samme driverne som CC2650-chippen i Contiki OS. Dette førte til den del utfordringer knyttet til flashing og kapabilitet. Dette var tid vi ikke hadde lagt inn i Gant-diagrammet som vi etter hvert måtte justere ressursbruken deretter.

### 7.3.2 Fase 4 i Gant-diagram

Dette viste seg å være en ubegrunnet risiko, men ikke av årsaken vi nevnte i forstudiet. Planen var at etter påske skulle modulen plasseres ut i felt for utendørs testing. De viktigste elementene for å gjennomføre en slik test er:

- Antenner for å øke rekkevidden til nodene.
- Plassering av bokser på egnet plass med tilgang til internett og fri sikt.

Dette lot seg ikke gjennomføre på grunn av at ingen av disse punktene var på plass, og vil heller ikke komme på plass før vi er ferdig med semesteret.

### 7.3.3 Arbeidssted

En viktig forutsetning for å løse en oppgave som baserer seg på å ta i bruk ny teknologi er et egnet arbeidssted. Etter 3 år på HVL vet vi hvor vanskelig det er å finne et sted å jobbe og vi begynte derfor å utforske alternativer. Nyskapningsparken har et Makerspace som kunne tilby oss sitteplass og internett tilkoblede ethernetporter. Dette lokalet hadde dessverre ikke ventilasjon og var uegnet for arbeid over lengre tid. Løsningen har vært å bruke HVL sine grupperom og PCB-labben i 4.etg.

## 8 Konklusjon

Vi har laget et system som oppfyller de forhåndsspesifiserte krav fra oppdragsgiver. Systemet består av fire bokser med komponenter, programvare og en server. Underveis er delmål blitt utvidet og endret med tanke på tid, teknisk kompetanse og økonomi. De valg som er tatt har bidratt til at ønsket resultat og oppgaven i sin helhet er oppnådd.

Oppgaven er delt i tre hovedmoduler som sammen danner en IoT-testlabb. Første modul operer som node kontroller. Den er tilkoblet internett, logger debug informasjon og laster opp ny programvare til nodene. Andre modul er nodene som sender informasjon på tre separate mesh-nett og kan motta ny programvare. Tredje modul er en sentralisert server som styrer node-kontrolleren og på den måten indirekte styrer alle nodene. Alt av endringer og arbeid gjøres fra denne modulen.

Løsningen vi presenterer gir et godt grunnlag for kommunikasjon over lengre avstander og testing av trådløse protokoller enn konvensjonelle IoT-labber vi har kjennskap til. Alt av prosjektets programvare og maskinvare er robust, tåler brukerfeil og kommer med en veldig dokumentert brukerhåndbok for videre arbeid.

### 8.1 Videreføring av oppgaven

Vi mener at det ligger et stort potensial for videre arbeid med oppgaven for studenter innen elektro- og datafag. Gjennom prosjektperioden har vi dannet infrastrukturen til et testnettverk som kun vil kreve fremtidige engasjerte, kreative og teknisk kompetente studenter for å videreutvikles. Våre forslag til videre arbeid er å forbedre brukergrensesnittet, gjøre tilpasninger av Contiki for å minimere effektbruk og å koble til sensorer til nodene som logger innsamlet data til en database.

## Litteraturliste

- [1] ICTP, «IoT in 5 days,» 2017. [Internett]. Available: [http://wireless.ictp.it/school\\_2015/book/](http://wireless.ictp.it/school_2015/book/). [Funnet 2017].
- [2] IEC, «Internet of Things: Wireless Sensor Networks,» 2017. [Internett]. Available: <http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf>. [Funnet 2017].
- [3] International Electrotechnical Commission, «Internet of Things: Wireless Sensor Networks,» 2014. [Internett]. Available: <http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf>. [Funnet 2017].
- [4] The Internet Engineering Task Force, «Transmission Control Protocol,» 1981. [Internett]. Available: <https://tools.ietf.org/html/rfc793>. [Funnet 2017].
- [5] The Internet Engineering Task Force, «The Internet Protocol,» 1981. [Internett]. Available: <https://tools.ietf.org/html/rfc791>. [Funnet 2017].
- [6] The Internet Engineering Task Force, «Internet Protocol Version 6 (IPv6) Specification,» 1998. [Internett]. Available: <https://tools.ietf.org/html/rfc2460>. [Funnet 2017].
- [7] IEEE, «IEEE Standards for Low-Rate Wireless Networks, DOI: 10.1109/IEEESTD.2016.7460875,» 2016. [Internett]. Available: <http://ieeexplore.ieee.org>. [Funnet 2017].
- [8] The Internet Engineering Task Force, «A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP,» 1988. [Internett]. Available: <https://www.ietf.org/rfc/rfc1055.txt>. [Funnet 2017].
- [9] IEEE, «A mesh topology formation scheme for IEEE 802.15.4 based wireless sensor networks, DOI: 10.1109/ICUFN.2015.7182523,» 2015. [Internett]. Available: <http://ieeexplore.ieee.org>. [Funnet 2017].
- [10] The Internet Engineering Task Force, «The Transport Layer Security (TLS) Protocol Version 1.2,» 2008. [Internett]. Available: <https://tools.ietf.org/html/rfc5246>. [Funnet 2017].
- [11] Federal Information Processing Standards, «Secure Hash Standard Publication 180-2,» 2002. [Internett]. Available: <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>. [Funnet 2017].
- [12] The Internet Engineering Task Force, «The Constrained Application Protocol (CoAP),» 2014. [Internett]. Available: <https://tools.ietf.org/html/rfc7252>. [Funnet 2017].
- [13] Texas Instruments, «The Universal Asynchronous Receiver/Transmitter (UART),» 2010. [Internett]. Available: <http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf>. [Funnet 2017].

- [14] IEEE, «1149.1-1990 - IEEE Standard Test Access Port and Boundary - Scan Architecture, DOI: 10.1109/IEEEESTD.1990.114395,» 1990. [Internett]. Available: <http://ieeexplore.ieee.org>. [Funnet 2017].
- [15] International Electrotechnical Commision, «IEC 60529:1989+AMD1:1999+AMD2:2013 CSV Consolidated version,» 2013. [Internett]. Available: <https://webstore.iec.ch/publication/2452>. [Funnet 2017].
- [16] United States Department of Defense, «MIL-STD-810G, DEPARTMENT OF DEFENSE TEST METHOD STANDARD: ENVIRONMENTAL ENGINEERING CONSIDERATIONS AND LABORATORY TESTS,» 2008. [Internett]. Available: [http://everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-810G\\_12306/](http://everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-810G_12306/). [Funnet 2017].
- [17] SeedStudio, «BeagleBone Green,» 2016. [Internett]. Available: [http://wiki.seeed.cc/BeagleBone\\_Green/](http://wiki.seeed.cc/BeagleBone_Green/). [Funnet 2017].
- [18] SeedStudio, «BeagleBone Black Wireless,» 2017. [Internett]. Available: <https://beagleboard.org/black-wireless>. [Funnet 2017].
- [19] Texas Instruments, «Simplelink™ Sub-1 GHz and Bluetooth® low energy CC1350 wireless MCU LaunchPad™ Development Kit,» 2017. [Internett]. Available: <http://www.ti.com/tool/launchxl-cc1350#descriptionArea>. [Funnet 2017].
- [20] Texas Instruments , «SimpleLink™ CC2650 Wireless MCU LaunchPad™ Kit,» 2017. [Internett]. Available: <http://www.ti.com/tool/launchxl-cc2650>. [Funnet 2017].
- [21] VMWare Inc., «Workstation for Windows,» 2017. [Internett]. Available: <http://www.vmware.com/products/workstation.html>. [Funnet 2017].
- [22] Texas Instruments, «SmartRF FlashProgrammer,» 2016. [Internett]. Available: <http://www.ti.com/tool/flash-programmer>. [Funnet 2017].
- [23] GitHub, «Hello World,» 2016. [Internett]. Available: <https://guides.github.com/activities/hello-world/>. [Funnet 2017].
- [24] Python, «What is Python? Executive Summary,» 2017. [Internett]. Available: <https://www.python.org/doc/essays/blurb/>. [Funnet 2017].
- [25] The University of Utah, «The C Programming Language,» [Internett]. Available: [http://www.cs.utah.edu/~germain/PPS/Topics/C\\_Language/the\\_C\\_language.html](http://www.cs.utah.edu/~germain/PPS/Topics/C_Language/the_C_language.html). [Funnet 2017].
- [26] GNU, «The GNU Project,» 2015. [Internett]. Available: <https://www.gnu.org/gnu/thegnuproject.html>. [Funnet 2017].

- [27] Debian, «DebianIntroduction,» 2013. [Internett]. Available: <https://wiki.debian.org/DebianIntroduction>. [Funnet 2017].
- [28] Debain, «Debain 8: 8.7 released,» 2017. [Internett]. Available: <https://www.debian.org/News/2017/20170114>. [Funnet 2017].
- [29] Contiki-OS, «Contiki: The Open Source OS for the Internet of Things,» [Internett]. Available: <http://www.contiki-os.org/>. [Funnet 2017].
- [30] Contiki-OS, «Get Started with Contiki,» [Internett]. Available: <http://www.contiki-os.org/start.html>. [Funnet 2017].
- [31] Canonical Ltd, «The Ubuntu Story,» 2017. [Internett]. Available: <https://www.ubuntu.com/about/about-ubuntu>. [Funnet 2017].
- [32] Texas Instruments, «TI,» 2017. [Internett]. Available: <http://www.multivu.com/players/English/7746254-texas-instruments-simplelink-dual-band-cc1350/>. [Funnet 2017].
- [33] BO17E-12, «Bachelor-v2017,» 2017. [Internett]. Available: [https://github.com/Jorgenhojer/Bachelor\\_v2017](https://github.com/Jorgenhojer/Bachelor_v2017).
- [34] J. T, «Github JelmerT,» 2017. [Internett]. Available: <https://github.com/JelmerT/>. [Funnet 2017].
- [35] Contiki, «Contiki pull request,» 2017. [Internett]. Available: <https://github.com/contiki-os/contiki/pull/1932>. [Funnet 2017].

## Appendiks A Forkortelser og ordforklaringer

ARM	Advanced RISC Machine
BBBW	BeagleBone Black Wireless
BBG	BeagleBone Green
Bin	Binær
BPRST	Board Package Reset
BR	BorderRouter
CC1350 Lp	CC1350 Launchpad
CoAP	Constrained Application Protocol
dBm	desiBel milliwatt
eMMC	embedded MultiMediaCard
GND	Ground
GNU	GNU's Not Unix
GPIO	General Purpose In/Out
GW	GateWay
HVL	Høgskolen på Vestlandet
HW	HardWare
ICTP	The International Centre for Theoretical Physics
IEC	International Electrotechnical Commision
IEEE	Institute of Electrical and Electronics Engineers
IETF	The Internet Engineering Task Force
IoT	Internet of Things
IP	International Protection Marking
IP	the Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JTAG	Joint Test Action Group

MCU	MicroController Unit
OS	Operating System
PCB	Printed Circuit Board
RISC	Reduced Instruction Set Computing
RPL	Routing Protocol for Low power and Lossy Networks
RSSI	Received Signal Strength Indication
RST	Reset
RX	Receiver
SEL	Select
SHA	Secure Hash Standard
SLIP	Serial Line Internet Protocol
SoC	System on Chips
SSH	Secure Shell
SSL	Secure Socket Layer
SW	SoftWare
TCP	Transmission Control Protocol
TI	Texas Instruments
TLS	Transport Layer Security
TX	Transmitter
UART	The Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
V	Volt
WMN	Wireless Mesh Network
WSN	Trådløse Sensornettverk
XDS	eXtended Development System
6LBR	6LoWPAN Border Router
6LoWPAN	internet protocol version 6 over Low power Wireless Personal Area Networks



## Appendiks B      Prosjektledelse og styring

### B.1              Prosjektorganisasjon

Som beskrevet i metodekapittelet har vi delt opp systemet i tre ledd: serverleddet, bakdørleddet (BBG) og nodeleddet (CC1350 Lp). Derfor var det naturlig å fordele det tekniske ansvaret der etter. Jørgen Høyer er hovedansvarlig for serverleddet, Aleksander I. Dahl er hovedansvarlig for bakdørleddet og Eivind I. Helle er hovedansvarlig for nodeleddet. Alle på gruppen har imidlertid arbeidet med oppgaver i alle ledd når det har vært nødvendig. Videre er Aleksander I. Dahl ansvarlig for design og utforming av presentasjoner og plakat, Eivind I. Helle har vært ansvarlig for materiell og Jørgen Høyer er ansvarlig for kontakt med veiledere og planlegging. Jørgen Høyer arbeidet som vitenskapelig assistent sommeren 2016, med et lignende prosjekt i regi av Sensario AS, og ble på grunnlag av dette valgt som prosjektleder for bachelorprosjektet.

### B.2              Fremdriftsplan

Bachelor Oppgave	Framdrift	Frister	Uke	1	2	3	4	5	6	7	8	9	10	11	12	13
Oppstartsmøte med veileder																
Planlegging av faser																
Forprosjekt	0 %															
Innlevering forprosjekt	0 %	31.jan														
<b>Fase 1</b>	0 %															
Flashing av Beaglebone med Debian Jessie	0 %															
SSH tilgang til BB	0 %															
Tilgang til BB bak brannmur	0 %															
<b>Fase 2</b>	0 %															
Contiki OS	0 %															
Flashing av cc1350 med Contiki	0 %															
Opplasting av kode til cc1350 via BB	0 %															
Kommunikasjon mellom 2 cc1350 noder	0 %															
<b>Fase 3</b>	0 %															
Oppsett av Border Router	0 %															
Sette opp nettverk med BB og remote eksekvering	0 %															
Innsamling av sensor data	0 %															
Logging av debug info fra cc1350 til BBG og videre til server	0 %															
Oppsett av sentral server	0 %															
<b>Fase 4</b>	0 %															
Utvide til 3 nettverk med BB	0 %															
Opplasting og rekonfigurering av flere noder v.h.a Ansible/Fabric	0 %															
Oppsett av server på HVL	0 %															
Opplasting til MongoDB	0 %															
Ta i bruk Fluentd	0 %															
Dokumentasjon av fase 1 til 4	0 %															
<b>Midtveispresentasjon</b>	0 %	Medio Apr.														
Ferie Aleksander Dahl																
Ferie Jørgen Høyer																
Ferie Eivind Helle																

Figur 36: Initiale Gant-diagram uke 1-13

Bachelor Oppgave	Framdrift	Frister	14	15	16	17	18	19	20	21	22	23	24
Midtveispresentasjon	0 %	Medio Apr.	■										
<b>Fase 5</b>	0 %												
Vannrette bokser med 5V forsyning	0 %												
Installasjon av nettverk i felt	0 %												
Test av nettverk i felt (stabilitet og ytelse)	0 %												
Måling av signalstyrke(RSSI)	0 %												
<b>Bachelor oppgave seminar</b>	0 %	12.mai											
<b>Fase 6</b>	0 %												
Opplasting av firmware til BB i felt	0 %												
Kjøring av algoritmer i felt	0 %												
Remote avlesning av debugginformasjon i felt	0 %												
<b>Dokumentasjon av fase 4 til 6</b>	0 %												
<b>Ferdigstilling</b>	0 %												
<b>Innlevering</b>	0 %	Slutt Mai											
Sluttrapport	0 %												
EXPO plakat	0 %	01.jun											
<b>Sluttpresentasjon av Bachelor</b>	0 %	Uke 24/25											
<b>EXPO 2017</b>	0 %	15.jun											
Ferie Aleksander Dahl													
Ferie Jørgen Høyler													
Ferie Eivind Helle													

Figur 37: Initiale Gant-diagram uke 14-24

Bachelor Oppgave	Framdrift	Frister	Uke	1	2	3	4	5	6	7	8	9	10	11	12	13
Oppstartsmøte med veileder																
Planlegging av faser																
Forprosjekt	100 %															
<b>Innlevering forprosjekt</b>	100 %	31.jan														
<b>Fase 1</b>	100 %															
Beaglebone Green HW	100 %															
Launchpad cc2650 HW	100 %															
<b>Fase 2</b>	100 %															
Contiki OS	100 %															
Flashing av cc2650 med Contiki	100 %															
Opplasting av kode til cc2650 via BB	100 %															
Kommunikasjon mellom 2 cc2650 noder	100 %															
<b>Fase 3</b>	100 %															
Oppsett av Border Router	100 %															
Sette opp nettverk med BB og remote eksekvering	100 %															
BBG HW feilsøking	100 %															
Fjernflashing	100 %															
Oppsett av server på Digital Ocean	100 %															
<b>Fase 4</b>	100 %															
Utvide til 3 nettverk med BB	100 %															
Launchpad cc1350 HW	100 %															
Server SW	100 %															
Innsamling av sensor data	100 %															
Logging av debug info fra cc1350 til BBG og videre til server	100 %															
<b>Dokumentasjon av fase 1 til 4</b>	50 %															
Ferie Aleksander Dahl																
Ferie Jørgen Høyler																
Ferie Eivind Helle																

Figur 38: Revidert Gant-diagram uke 1 til 13

Bachelor Oppgave	Framdrift	Frister	Uke	14	15	16	17	18	19	20	21	22	23	24
<b>Midtveispresentasjon</b>	<b>100 %</b>	<b>21.apr</b>		14										
<b>Fase 5</b>	<b>100 %</b>							125 timer						
Vannrette bokser med 5V forsyning	100 %													
Flashe scripting	100 %													
Test av nettverk (innendørs)	100 %													
<b>Fase 6</b>	<b>100 %</b>								350 timer					
Opplasting av firmware til BB i felt	100 %													
Test av nettverk (utendørs)	100 %													
<b>Dokumentasjon av fase 4 til 6</b>	<b>100 %</b>													
<b>Ferdigstilling</b>													150 timer	
<b>Innlevering</b>					30.mai								1	
Sluttrapport														
EXPO plakat	100 %	01.jun										1		
<b>Sluttpresentasjon av Bachelor</b>		13.jun											1	
<b>EXPO 2017</b>		14.jun											1	
<b>Ferie Aleksander Dahl</b>							1							
<b>Ferie Jørgen Høyre</b>							1							
<b>Ferie Eivind Helle</b>							1							

Figur 39: Revidert Gant-diagram uke 14 til 24

## B.3 Risikoliste

Risiko	Sannsynlighet (1-10)	Alvorlighetsgrad (1-10)	Konsekvens	Tiltak
<b>Komponenter blir ikke utlevert i tide</b>	5	7	Vi får ikke fulført oppgaven.	Benytte lignende komponenter.
<b>Sykdom</b>	8	2	Tap av arbeidstid	De som ikke er syk må ta ansvar for tapt arbeid
<b>Personlige konflikter</b>	1	4	Dårlig arbeidsmoral	Bedre arbeidsmiljøet
<b>Budsjett</b>	8	4	Får ikke utløst potensialet i oppgaven	Planlegge innkjøp og avveie hva som er viktigst
<b>Bruksfeil av komponenter</b>	7	5	HW kan slutte å virke	Feilsøking og reparasjon eller innkjøp av ny HW
<b>Tidsfrister</b>	2	9	Får ikke levert inn arbeid	Planlegge og sette interne tidsfrister

## Appendiks C Brukerdokumentasjon

### C.1 Brukerdokumentasjon

Dette systemet er laget for å kunne laste opp nye programmer til nodene og hente debug informasjon etter en test.

#### Laste opp ny .bin fil

##### Krav til program som skal lastes opp

- Prosjektet må ha riktige drivere for cc1350 Launchpad
- Må ha muligheten for å åpne bakdør ved at DIO13 og BPreset settes lav.
- Hvis man ønske debug data må dette aktiveres
- Debug data må skrives ut over UART

##### Hvordan laste inn bin fil til server

1. Last først opp .bin filen fra din PC til et github repository
2. Logg inn på server.

```
login as: bachelor
bachelor@207.154.220.121's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)
```

3. Naviger til mappen til det nettet du skal bruke (\$: cd Upload/Nett(1, 2 eller 3))

```
bachelor@bwsensorlab:~$ cd Upload/Nett1
bachelor@bwsensorlab:~/Upload/Nett1$ █
```

4. Last ned filen fra github repository til serveren  
(\$:wget https://raw.githubusercontent.com/User/Path/To/Directory.bin)

```
bin_filer/ch6/cc1350webdemo-ch6.bin
--2017-05-24 13:45:46-- https://raw.githubusercontent.com/Jorgenhoyer/Bachelor_v2017/master/bin_filer/ch6/cc1350webdemo-ch6.bin
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133, 151.101.128.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 131072 (128K) [application/octet-stream]
Saving to: 'cc1350webdemo-ch6.bin'

cc1350webdemo-ch6.bin      100%[=====] 128.00K --.-KB/s   in 0.002s

2017-05-24 13:45:46 (67.1 MB/s) - 'cc1350webdemo-ch6.bin' saved [131072/131072]

bachelor@bwsensorlab:~/Upload/Nett2$ █
```

Hvis man får opp dette bilde er filen lastet ned i riktig mappe

## Hvordan starte SLIP-radio

1. Logg inn på server

```
login as: bachelor
bachelor@207.154.220.121's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)
```

2. åpne en ssh tunell til client5 \$: ssh root@10.8.0.18

```
bachelor@bwsensorlab:~$ ssh root@10.8.0.18
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2016-11-06

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

root@10.8.0.18's password:
Last login: Thu May 25 10:11:11 2017 from 10.8.0.1
root@bbg5:~#
```

3. naviger til mappen tools. \$: cd contiki/tools

```
root@bbg5:~# cd contiki/tools
root@bbg5:~/contiki/tools#
```

4. kompiler tunslip6 \$: make tunslip6

```
root@bbg5:~/contiki/tools# make tunslip6
make: 'tunslip6' is up to date.
root@bbg5:~/contiki/tools#
```

5. start tunslip for det nettet eller de nettene du skal teste.

```
$: sudo ./tunslip6 aaaa:1/64 -s /dev/ttys(1, 2 eller 4)
```

```
root@bbg5:~/contiki/tools# sudo ./tunslip6 aaaa:1/64 -s /dev/ttys2./tunslip6 aaa
a:1/64 -s /dev/ttys2
*****SLIP started on `/dev/ttys2'
opened tun device `/dev/tun1'
ifconfig tun1 inet `hostname` mtu 1500 up
ifconfig tun1 add aaaa:1/64
aaaa:1: Resolver Error 0 (no error)
ifconfig tun1 add fe80::0:0:0:1/64
ifconfig tun1

tun1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
                      inet6 addr: fe80::1/64 Scope:Link
                        UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
                        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
                        collisions:0 txqueuelen:500
                        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

## Hvordan flashe nodene i nettet fra server

1. Logg inn på server

```
login as: bachelor
bachelor@207.154.220.121's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)
```

2. naviger til mappen til nettet du vil laste opp ny fil til. \$: cd Upload/Nett(1, 2 eller 3)

```
bachelor@bwsensorlab:~$ cd Upload/Nett2
bachelor@bwsensorlab:~/Upload/Nett2$ █
```

3. Start scriptet som laster opp ny .bin fil til alle nodene på valgt nett

```
$: python Automatisering_nett2_v2.py
```

```
bachelor@bwsensorlab:~/Upload/Nett2$ python Automatisering_nett2_v2.py
Sendert til aktive klienter...
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
Nett2.bin
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
Nett2.bin
Starter flashing av nett2
Stopper debugging av node 2 client 2..
```

```
Starter flashing av node 2 client 2
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]

Opening port /dev/ttyS2, baud 500000
Reading data from /root/Bin_filer/Nett2/Nett2.bin
Firmware file: Raw Binary
Connecting to target...
CC1350 PG2.1 (7x7mm): 128KB Flash, 20KB SRAM, CCFG.BL_CONFIG at 0x0001FFD8
Primary IEEE Address: 00:12:4B:00:0E:01:0F:05
Erasing all main bank flash sectors
    Erase done
Writing 131072 bytes starting at address 0x00000000
Write 128 bytes at 0x0001FF808
    Write done
Verifying by comparing CRC32 calculations.
    Verified (match: 0xbd3975d8)

Starter debugging av node 2 client 2
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]

Client 2 nett 2 fullfort.
```

Dette gjentar seg automatisk for alle boksene som er koblet til serveren.

## Hvordan hente debug data

1. logg inn på server

```
login as: bachelor
bachelor@207.154.220.121's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)
```

2. Naviger til mappen til det nettet du skal hente debug data fra. \$: cd Debug/nett(1, 2 eller 3)

```
bachelor@bwsensorlab:~$ cd Debug/nett2
bachelor@bwsensorlab:~/Debug/nett2$ ls
debug-data-nett2.py
```

3. Start script som henter debug data fra alle BBG. \$: python debug-data-nett(1, 2 eller 3).py

```
bachelor@bwsensorlab:~/Debug/nett2$ python debug-data-nett2.py
2017-05-22_14-57
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
debug-node2.txt
Debian GNU/Linux 8
BeagleBoard.org Debian Image 2016-11-06
Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian
default username:password is [debian:temppwd]
debug-node2.txt
bachelor@bwsensorlab:~/Debug/nett2$
```

4. Nå er det blitt laget en tidsstempel mappe som du kan gå inn i og hente filene dine.

\$: cd (ditt tidsstempel)

```
bachelor@bwsensorlab:~/Debug/nett2$ ls
2017-05-22_14-57  debug-data-nett2.py
bachelor@bwsensorlab:~/Debug/nett2$ cd 2017-05-22_14-57/
bachelor@bwsensorlab:~/Debug/nett2/2017-05-22_14-57$ ls
bbg2nett2.txt  bbg3nett2.txt
bachelor@bwsensorlab:~/Debug/nett2/2017-05-22_14-57$
```

Her ligger nå alle debug filene dine som en .txt fil.

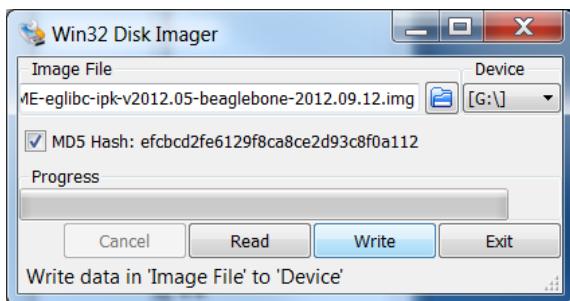
## C.2 Drifts- og vedlikeholds dokumentasjon

### Beaglebone Green

#### Flashing av eMMC

##### I Windows

1. Last ned nyeste Debian Jessie på <https://beagleboard.org/latest-images>
2. Unzip filen og legg den på en kjent plassering
3. Last ned <https://sourceforge.net/projects/win32diskimager/files/latest/download>
4. Åpne win32diskimager og velg microSD kortet du har plugget i maskinen

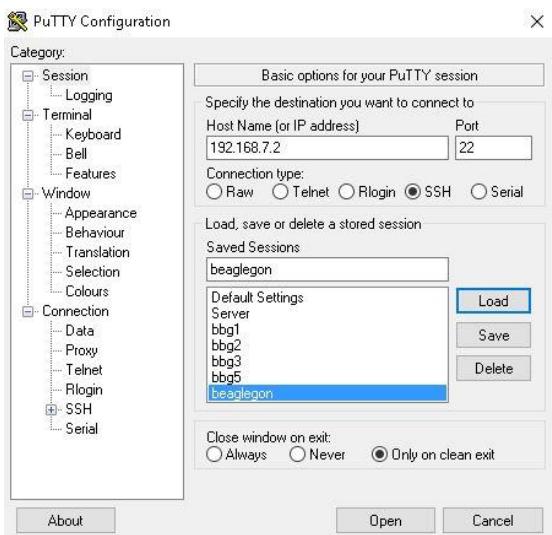


Figur 40:win 32 diskimager

5. Skriv filen du har lastet ned til microSD kortet og plugg ut når det er ferdig.
6. Installer Putty <http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

##### På Beaglebone

7. Koble BBG til PC vha. USB kabelen som følger med.
8. Åpne Putty og skriv inn adresse 192.168.7.2 og trykk Open.



**Figur 41: Putty BBG ssh**

9. Du vil bli møtt med et sikkerhetsvarsle, trykk OK.
10. Standard brukernavn og passord på BBG er *root*.
11. Naviger til `cd /boot`
12. Åpne `uEnv.txt` ved å skrive `nano uEnv.txt`
13. Endre hvor det står:

```
##enable BBB: eMMC Flasher:  
cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh
```

Til:

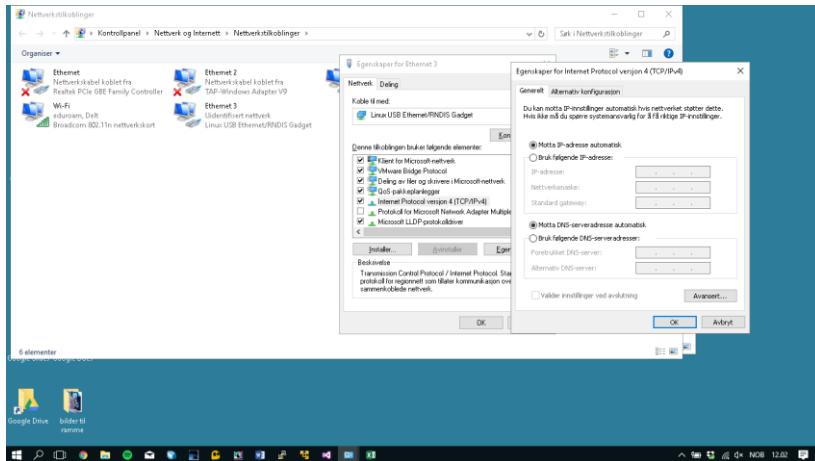
```
##enable BBB: eMMC Flasher:  
cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh
```

14. Lagre filen og slå av BB
15. Sett inn microSD kortet i BB og slå på.
16. Led lysene vil indikere at den blir flashet ved å pulsere i et «Cyclon Knight-Rider» mønster.  
<https://www.youtube.com/watch?v=Mo8Qls0HnWo>. Når BB skrur seg av er den ferdig flashet.
17. Ta ut microSD kortet og slå på BB

## BBG Internett ved HVL:

Lokale innstillingar på PC:

1. Sett opp deling av internett gjennom Ethernet (Den som USB er koblet til).
2. Velg automatisk tilføying av ip under egenskaper Internett protocoll versjon 4



Figur 42: TCP innstillingar for deling av internett

Innstillingar på BBG

1. Plugg inn BBG og koble til v.h.a Putty.
2. I command prompt skriv: `/sbin/route add default gw 192.168.7.1`
3. `cd /etc`
4. `nano resolv.conf`  
Tilføy under nameserver 127.0.0.1
  - nameserver 158.37.87.2
  - nameserver 158.37.87.5

```
192.168.7.2 - PuTTY
GNU nano 2.2.6      File: resolv.conf

Generated by Connection Manager
nameserver 127.0.0.1
nameserver ::1
nameserver 158.37.87.2
nameserver 158.37.87.5

[ Read 5 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^X Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figur 43:BBG resolv.conf

- Lagre (ctrl+O)
  - Avslutt (ctrl+x)
5. `ping www.vg.no`

Mottar du pakkdata er du koblet til internett.

## Skript for internett og tidsstilling

Skript laget av Derekmolloy som automatisk setter opp riktig nameserver, gw og tidssinstillinger.

1. git clone git://github.com/derekmolloy/ee402
2. apt-get install ntpdate
3. apt-get install ntp
4. ntpdate us.pool.ntp.org
5. Fiks datoens gjennom kommandoens dpkg-reconfigure tzdata (åpner seg et nytt vindu, velg Oslo)
6. cd ee402/
7. cd scripts/
8. nano StartUSBNetwork

Skriv inn:

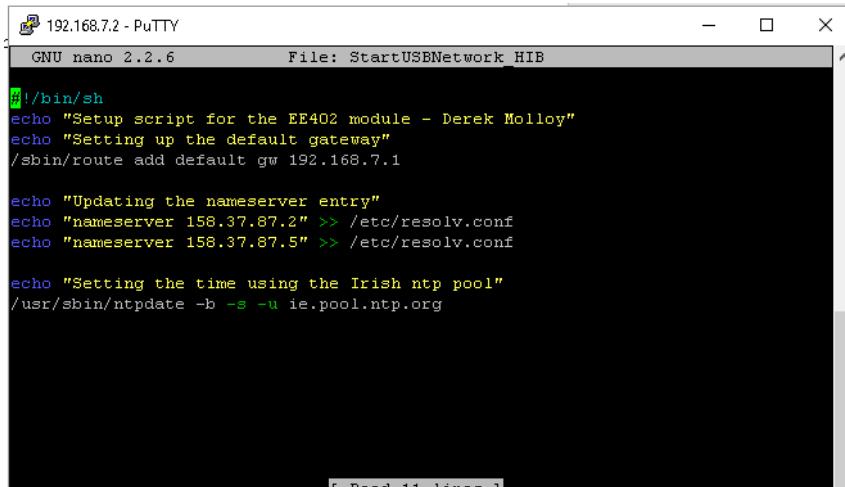
nameserver 158.37.87.2

nameserver 158.37.87.5

Du må også endre */usr/bin/ntpdate -b -s -u ie.pool.ntp.org* til

*usr/sbin/ntpdate -b -s -u ie.pool.ntp*

lagre og gå videre til steg 10.



```
#!/bin/sh
echo "Setup script for the EE402 module - Derek Molloy"
echo "Setting up the default gateway"
/sbin/route add default gw 192.168.7.1

echo "Updating the nameserver entry"
echo "nameserver 158.37.87.2" >> /etc/resolv.conf
echo "nameserver 158.37.87.5" >> /etc/resolv.conf

echo "Setting the time using the Irish ntp pool"
/usr/sbin/ntpdate -b -s -u ie.pool.ntp.org
```

Figur 44: Skript for Internett av Derek Molloy

9. cp StartUSBNetwork ~/.
10. cd
11. more StartUSBNetwork
12. chmod ugo+x StartUSBNetwork
13. reboot

Når BBG har rebootet og du har koblet til gjennom Putty skriv:

14. ./StartUSBNetwork

Og du er koblet til internett.

## Liste over nyttig programmer og services på BBG

Som standard kommer BBG med et lite knippe programmer som er helt grunnleggende for vanlig funksjonalitet. Vi trenger en del ekstra programmer for å kjøre kode og logging på våre BBG.

- pip install --upgrade pip  
(oppgraderer pip-installerer)
- pip install python-magic  
(for detektering av .bin fil i bsl-skript for flashing av cc13xx)
- pip install pyserial  
(for seriell kommunikasjon i python-skript)
- pip install intelhex  
(for detektering av .hex fil i bsl-skript for flashing av cc13xx)
- apt-get install python3-serial  
(for seriell kommunikasjon i python3-skript)
- apt-get install ntp  
(dato tid)
- apt-get install minicom  
(program for seriell kommunikasjon)
- pip install --upgrade setuptools  
(oppgjører setuptools for python-pip)
- pip install ez\_setup  
(installerer ez-setup)
- apt-get install python3-pip  
(innstallerer python3 versjonen av pip)
- pip3 install ez\_setup  
(installerer python3 versjonen av ez setup)
- pip3 install --upgrade setuptools pip  
(oppgraderer setuptools for python3)
- pip3 install aiocoap  
(installerer mulighet for coap-server)
- pip install asyncio  
(installerer synkronisering program for python)
- apt-get install ntpdate  
(tid dato)
- apt-get install openvpn  
(installerer vpn program)
- pip install cryptography

- (installerer krypterings program)
- pip install zdaemon
  - (installerer program som kan gjøre andre program til daemon/service)

#### **Autoremove for overflødige filer på BBGer:**

- apt-get autoremove

#### **Nødvendig nedlasting for BBG border-routere:**

- git clone --recursive <https://github.com/contiki-os/contiki.git>

PS! Kan også være lurt å kjøre sudo apt-get update og Sudo apt-get upgrade en gang iblant for å installere de nyeste oppdateringene på BBG.

### **Aktivere UART på BBG.**

Som standard kommer ikke BBG med åpne UART porter.

1. I command prompt skriv: nano /boot/uEnv.txt
2. Finn linjen under eksempel 4.1: #cape\_enable=capemgr.enable\_partno=
3. Endre til: cape\_enable=capemgr.enable\_partno=BB-UART1,BB-UART2,BB-UART4, BB-UART5
4. Sjekk om de er gjenkjent ved å skrive: ls -l /dev/ttyS\*

### **Minicom for testing av UART**

Minicom oppfører seg som et modem kontroller og terminal emulerings program. Slik kan vi teste om vi har enablet UART gjennom pinnene på brettet.

1. apt-get install minicom

Testing: To måter å teste på. (VIKTIG! Slå av BBG når du kobler)

#### **Metode 1: Ved hjelp av et BBG kort**

For å teste UART1 og UART2:

Koble pinnene v.h.a lasker («han til han» ledninger) slik:

UART1 TXD (P9\_24) -> UART2 RXD (P9\_22)

UART2 TXD (P9\_21) -> UART1 RXD (P9\_26)

UART		P9		P8	
DGND	1	2	DGND	1	2
VDD_3_3	3	4	VDD_3V3	GPIO_38	3
VDD_5V	5	6	VDD_5V	GPIO_34	5
SYS_5V	7	8	SYS_5V	GPIO_66	7
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9
UART4_RXD	11	12	GPIO_60	GPIO_45	11
UART4_TXD	13	14	GPIO_50	GPIO_23	13
GPIO_48	15	16	GPIO_51	GPIO_47	15
GPIO_5	17	18	GPIO_4	GPIO_27	17
UART1_RTSN	19	20	UART1_CTSN	GPIO_22	19
UART2_TXD	21	22	UART2_RXD	GPIO_62	21
GPIO_49	23	24	UART1_RXD	GPIO_36	23
GPIO_117	25	26	UART1_RXD	GPIO_32	25
GPIO_115	27	28	GPIO_123	GPIO_86	27
GPIO_121	29	30	GPIO_122	GPIO_87	29
GPIO_120	31	32	VDD_ADC	UART5_CTSN+	31
AIN4	33	34	GNDA_ADC	UART4_RTSN	33
AIN6	35	36	AIN5	UART_4_CTSN	35
AIN2	37	38	AIN3	UART5_TXD+	37
AIN0	39	40	AIN1	GPIO_76	39
GPIO_20	41	42	GPIO_7	GPIO_74	41
DGND	43	44	DGND	GPIO_72	43
DGND	45	46	DGND	GPIO_70	45

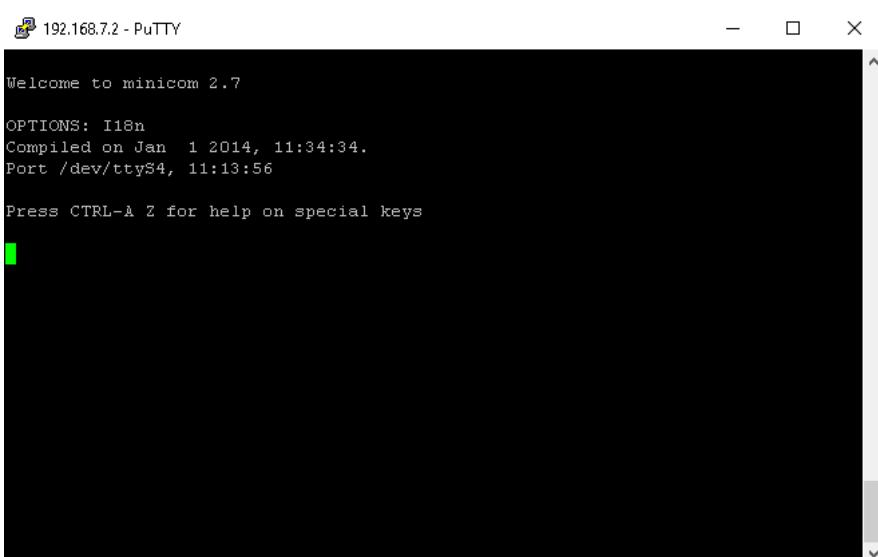
Figur 45: BBG UART pins for minicom

Åpne to terminaler i Putty.

I den ene terminalen skriver du følgende:

```
minicom -D /dev/ttys1 -b 9600
```

Det åpner seg nå en ny terminal som ser slik ut:

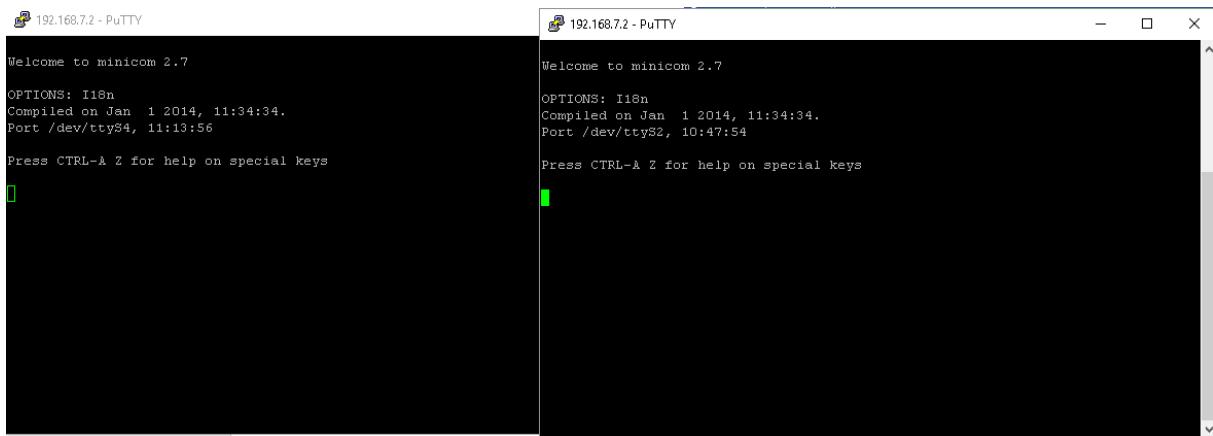


Figur 46: Putty Minicom

I den andre terminalen skriver følgende:

```
minicom -D /dev/ttys2 -b 9600
```

Du skal nå ha to terminaler åpne hvor du kan sende toveiskommunikasjon gjennom tastaturet:

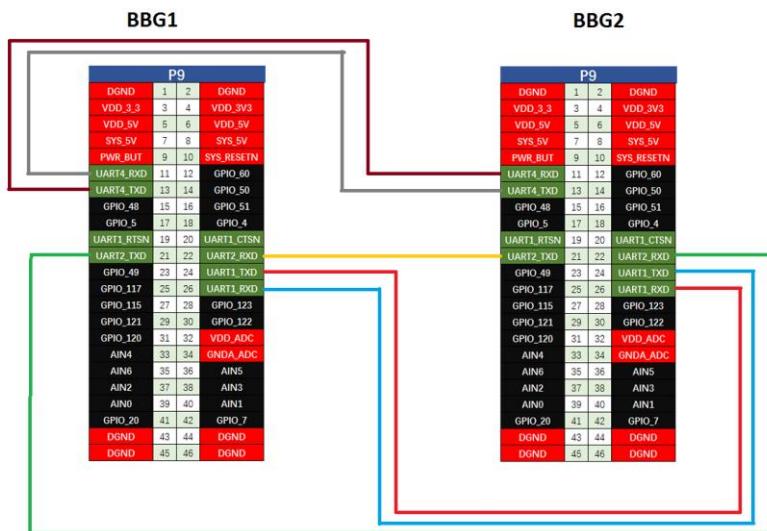


**Figur 47: Putty Minicom**

Får du kommunikasjon gjennom har du enablet UART gjennom pinnene

#### **Metode 2:** Ved hjelp av to BBG og to datamaskiner:

Koble pinnene slik skjemaet viser. (Oppsettet viser kobling for UART fra 1 til 4, dette er ikke nødvendig i en test sammenheng).



**Figur 48: BBG Minicom Uart for to CC1350**

SSH gjennom putty til BBG og skriv følgende på hver av datamaskinene:

```
minicom -D /dev/ttyS2 -b 9600
```

Får du kommunikasjon fra den ene datamaskinen til den andre har du enablet UART gjennom pinnene

## Github

Under hele Bacheloroppgaven har vi benyttet github som standard. Her finner du alle .bin filer og skript som vi bruker.

1. I ønsket mappe på BBG skriv: git clone [https://github.com/Jorgenhoyer/Bachelor\\_v2017](https://github.com/Jorgenhoyer/Bachelor_v2017)

## Flashing av CC1350 over USB

Koble CC1350 til BBG med USB. Åpne bakdøren på CC1350 manuelt, hold inne DIO11(sideknapp under XDS110) og trykk på reset knapp. Da skal dioden som blinker grønt slå seg av og kortet er klar for å bli flashet. Husk å alltid flashe med en fil som har åpen bakdør.

1. cd Bachelor\_v2017
2. python flash-node1.py -e -w -v ~/Bachelor\_v2017/*ditt filnavn her*.bin

## Flashing av CC1350 over UART

Vårt system kommuniserer over UART (se koblingsskjema) og flashes med 3 skript som opererer på UART 1,2 og 4. Dette skriptet har også en bootloader bakdør funksjon, den åpner bakdøren for deg over I/O pins på BBG. Husk å alltid flashe med en fil som har åpen bakdør.

- flash-node1.py (UART 1 kalt ttyS1 på BBG)
- flash-node2.py (UART 2 kalt ttyS2 på BBG)
- flash-node3.py (UART 4 kalt ttyS4 på BBG)

For å kjøre skriptet, naviger til riktig mappe:

1. cd Bachelor\_v2017
2. python flash-nodeX.py -e -w -v ~/Bachelor\_v2017/*ditt filnavn her*.bin
3. Bytt ut X med den porten du vil flashe.
4. Trenger du hjelp skriv: python flash-nodeX.py -help

Vi har modifisert et skript utviklet av JelmerT som kan slette, programmere, verifisere og lese flash til cc13xx & cc26xx SOC. Det som er endret er:

- Lagt til UART
- Lagt til bakdør skript

En kan enkelt legge til mer funksjon ved å åpne .py filen i din favoritt editor og skrive mer kode.

## Logge seriell data fra cc13xx (rpl debug data)

NB! BBG må ha adafruit\_BBIO og pyserial installert!

1. Start en BR (slik som i border-router guiden)
2. Start en node som er koblet serielt til BBG (se koblingsskjema)
3. På BBG naviger til cd Bachelor\_v2017
4. Kjør skriptet som viser til hvilken port du bruker
  - a. Debug-node1.py -> UART 1(ttyS1)
  - b. Debug-node2.py -> UART 1(ttyS2)
  - c. Debug-node3.py -> UART 1(ttyS4)

f.eks. python debug-node1.py

5. Debug utskriften vil printes ut i terminalen og lagres i debug-node1.txt så lenge programmet kjører.
6. For å kjøre flere skript simultant skriv: python debug-node1.py & python debug-node2.py & python debug-node3.py
7. Gå til BBG terminalen og naviger til Bachelor\_v2017 mappen (eller hent den fra **NB!** Dette vil opprette en fil: debug-node1.txt i den mappen du er i når du kjører skriptet. Det er fint å endre sti i filen hvis du ønsker det. Bruk nano for å gjøre endringer i debug-node1.py. Her kan også port endres.
8. Debug utskriften vil printes ut i terminalen og lagres i debug-node1.txt så lenge programmet kjører.
9. I Instant contiki terminalen vil debug data for border routeren printes ut i terminalen, men det vil selvsagt ikke opprettes en slik fil siden .py programmet ikke kjører der.

```
2016-05-21 23:33:48
RPL: Received consistent DIO

2016-05-21 23:34:06
RPL: Sending prefix info in DIO for aaaa::

2016-05-21 23:35:01
RPL: Sending prefix info in DIO for aaaa::

2016-05-21 23:35:28
RPL: Received a DIO from fe80::212:4b00:aff:5387

2016-05-21 23:35:28
RPL: Incoming DIO (id, ver, rank) = (30,240,128)

2016-05-21 23:35:28
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)

2016-05-21 23:35:28
RPL: DIO option 4, length: 14

2016-05-21 23:35:28
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60

2016-05-21 23:35:28
RPL: DIO option 8, length: 30

2016-05-21 23:35:29
RPL: Copying prefix information

2016-05-21 23:35:29
RPL: Prefix announced in DIO

2016-05-21 23:35:29
RPL: Prefix set - will announce this in DIOs

2016-05-21 23:35:29
rpl_set_prefix - prefix NON-NUL

2016-05-21 23:35:29
Set dag aaaa::212:4b00:aff:5387 lifetime to 3145

2016-05-21 23:35:29
```

Figur 49: Debug data fra node

```
RPL: Adding DAO route
RPL: Received a DIO from fe80::212:4b00:aff:2781
RPL: Incoming DIO (id, ver, rank) = (30,240,256)
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)
RPL: DIO option 4, length: 14
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60
RPL: DIO option 8, length: 30
RPL: Copying prefix information
RPL: Prefix announced in DIO
RPL: Prefix set - will announce this in DIOs
rpl_set_prefix - prefix NON-NULL
RPL: Received a DIO from fe80::212:4b00:aff:2781
RPL: Incoming DIO (id, ver, rank) = (30,240,256)
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)
RPL: DIO option 4, length: 14
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60
RPL: DIO option 8, length: 30
RPL: Copying prefix information
RPL: Prefix announced in DIO
RPL: Prefix set - will announce this in DIOs
rpl_set_prefix - prefix NON-NULL
*** dropping large 2000 byte packet
*** dropping large 2000 byte packet
```

Figur 50: Debug data fra BR

## CONTIKI for Launchpad-CC1350.

### Installasjon av WMware og Contiki

1. Last ned nyeste versjon av WMware fra <http://www.vmware.com/>
2. Installer VMWare
3. last ned nyeste utgave av contiki instant fra  
<https://sourceforge.net/projects/contiki/files/Instant%20Contiki/>
4. Pakk ut filene på en plass du finner de igjen (sørg for at når filen er ferdig utpakket at du finner vmdk filen. Uten denne får du ikke kjørt contiki i WMware)
5. Start VMWare, velg «Open a virtual machine», så finner du vmdk filen i contiki mappen. (det kan ta litt tid før den virtuelle maskinen starter opp)
6. passordet er: user
7. lag en ny mappe som du vet hvor ligger. Denne er vanlig å kalle git-repo
8. gå inn i mappen ved å skrive: cd git-repo  
skriv så denne linjen inn i terminalen: git clone --recursive <https://github.com/contiki-os/contiki.git>
9. Oppdater ved å skrive:  
sudo apt-get update  
sudo apt-get upgrade

### Launchpad-CC1350 drivere

Den nyeste driveren til CC1350 Launchpad ligger ikke inne i Contiki enda. Men de har en pull request på Github som vi kan bruke til vårt formål. (<https://github.com/contiki-os/contiki/pull/1932>).

1. Gå til git-repo/contiki
2. Skriv git pull origin pull/1932/head (sjekk at denne pull requesten ikke er godkjent enda, hvis den er lagt til i master så kan du benytte: git fetch)
3. CC1350 ligger nå under git-repo/contiki/platform/launchpad/CC1350

### Åpne bootloader backdoor for fjernflash.

1. Lokaliser filen git-repo/Contiki/cpu/cc26xx-cc13xx/lib/cc13xx/Startup\_files/ccfg.c (CCFG står for Customer Configuration).
2. Endre følgende registre i filen slik at den ser slik ut:

```
#####
// Bootloader settings
#####

#ifndef SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE
```

```
//#define SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE      0x00 // Disable ROM boot
loader
#define SET_CCFG_BL_CONFIG_BOOTLOADER_ENABLE      0xC5 // Enable ROM boot
loader
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_LEVEL
#define SET_CCFG_BL_CONFIG_BL_LEVEL      0x0 // Active low to open boot loader
backdoor
#define SET_CCFG_BL_CONFIG_BL_LEVEL      0x1 // Active high to open boot
loader backdoor
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_PIN_NUMBER
#define SET_CCFG_BL_CONFIG_BL_PIN_NUMBER      0x0D // DIO number for boot
loader backdoor
#endif

#ifndef SET_CCFG_BL_CONFIG_BL_ENABLE
#define SET_CCFG_BL_CONFIG_BL_ENABLE      0xC5 // Enabled boot loader
backdoor
#define SET_CCFG_BL_CONFIG_BL_ENABLE      0xFF // Disabled boot loader
backdoor
#endif
```

- Nå kan bootloaderens bakdør åpnes manuelt ved å holde inne BUTTON 1 (DIO13) og trykke på reset. (Hvis cc26xx-web-demo allerede er på CC1350 skal ledet sluttet å blinke).
- Kan endre mellom aktiv høy og aktiv lav for å åpne bakdør. Vi har benyttet aktiv høy.

## Kanalvalg

For å kunne kjøre flere nett må vi endre kanalen de respektive CC1350 setter opp nett på. Filer som må endres på:

- **cc26xx-web-demo.**
  1. git-repo/contiki/examples/cc26xx/cc26xx-web-demo/project-conf.h
  2. Gå til linje 36 og endre kanalen fra 1-25 etter behov.

```
#define IEEE802154_CONF_PANID      0xABCD
#define RF_CORE_CONF_CHANNEL      25
#define RF_BLE_CONF_ENABLED      1
```

- **border-router**
1. git-repo/contiki/examples/ipv6/rpl-border-router/project-conf.h
  2. Legg til:  
`//Valg av kanal  
#ifndef RF_CORE_CONF_CHANNEL  
#define RF_CORE_CONF_CHANNEL 25  
#endif`

## Økt stabilitet på border-router.

Dette er en fiks for border-router som kjører på SOC fra Texas instrument.

1. git-repo/contiki/examples/ipv6/rpl-border-router/project-conf.h
2. Kommenter ut følgende kode:  
`/*  
#ifndef UIP_CONF_BUFFER_SIZE  
#define UIP_CONF_BUFFER_SIZE 140  
#endif  
  
#ifndef UIP_CONF_RECEIVE_WINDOW  
#define UIP_CONF_RECEIVE_WINDOW 60  
#endif  
*/`

## Utskrift av RSSI på nodene.

1. git-repo/contiki/examples/cc26xx/cc26xx-web-demo/cc26xx-web-demo.c
2. på linje 423 står det:  
`if(uip_ip6addr_cmp(source, uip_ds6_defrt_choose())) {  
 def_rt_rssi = sicslowpan_get_last_rssi();`
3. Endre til:  
`if(uip_ip6addr_cmp(source, uip_ds6_defrt_choose())) {  
 def_rt_rssi = sicslowpan_get_last_rssi();  
 // printe ut rssi  
 printf("RSSI: %d ", def_rt_rssi);`

## Debug utskrift.

1. git-repo/contiki/core/net/rpl/
2. Endre filene:  
rpl-dag.c: endre DEBUG\_NONE til DEBUG\_FULL

rpl-icmp6.c: endre DEBUG\_NONE til DEBUG\_FULL

### 3. Debug data:

```
2016-05-21 23:33:48
RPL: Received consistent DIO

2016-05-21 23:34:06
RPL: Sending prefix info in DIO for aaaa::

2016-05-21 23:35:01
RPL: Sending prefix info in DIO for aaaa::

2016-05-21 23:35:28
RPL: Received a DIO from fe80::212:4b00:aff:5387

2016-05-21 23:35:28
RPL: Incoming DIO (id, ver, rank) = (30,240,128)

2016-05-21 23:35:28
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)

2016-05-21 23:35:28
RPL: DIO option 4, length: 14

2016-05-21 23:35:28
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60

2016-05-21 23:35:28
RPL: DIO option 8, length: 30

2016-05-21 23:35:29
RPL: Copying prefix information

2016-05-21 23:35:29
RPL: Prefix announced in DIO

2016-05-21 23:35:29
RPL: Prefix set - will announce this in DIOs

2016-05-21 23:35:29
rpl_set_prefix - prefix NON-NUL

2016-05-21 23:35:29
Set dag aaaa::212:4b00:aff:5387 lifetime to 3145

2016-05-21 23:35:29
```

Figur 51: Rpl debug data fra node

```
RPL: Adding DAO route
RPL: Received a DIO from fe80::212:4b00:aff:2781
RPL: Incoming DIO (id, ver, rank) = (30,240,256)
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)
RPL: DIO option 4, length: 14
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60
RPL: DIO option 8, length: 30
RPL: Copying prefix information
RPL: Prefix announced in DIO
RPL: Prefix set - will announce this in DIOs
rpl_set_prefix - prefix NON-NUL
RPL: Received a DIO from fe80::212:4b00:aff:2781
RPL: Incoming DIO (id, ver, rank) = (30,240,256)
RPL: Incoming DIO (dag_id, pref) = (aaaa::212:4b00:aff:5387, 0)
RPL: DIO option 4, length: 14
RPL: DAG conf:dbl=8, min=12 red=10 maxinc=896 mininc=128 ocp=1 d_l=30 l_u=60
RPL: DIO option 8, length: 30
RPL: Copying prefix information
RPL: Prefix announced in DIO
RPL: Prefix set - will announce this in DIOs
rpl_set_prefix - prefix NON-NUL
*** dropping large 2000 byte packet
*** dropping large 2000 byte packet
```

Figur 52: Debug data fra BR

## Border-router.bin

1. I terminal.
2. cd git-repo/Contiki/examples/ipv6/rpl-border-router
3. Det kan være lurt å skrive kommandoen: make clean i terminalen mellom hver gang du skal opprette ny .bin fil eller .hex fil. Da rensker du opp og sletter tidligere filer som har blitt generert av make kommandoen slik at nye konfigurasjoner som er gjort i Contiki blir tatt neste gang du skal opprette contiki image (.bin/.hex).

4. make TARGET=srf06-cc26xx BOARD=launchpad/cc1350 border-router.bin
5. Flytt den nylig opprettede border-router.bin (eller .hex) til skrivebordet
6. Bruk FlashProgrammer 2 til å flashe CC1350 med border-router.bin

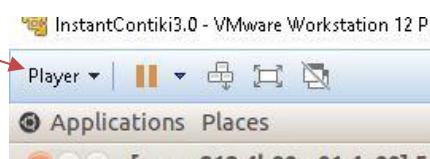
## CC26xx-web-demo.bin

1. I terminal.
2. cd git-repo/Contiki/examples/cc26xx/cc26xx-web-demo
3. Det kan være lurt å skrive kommandoen: make clean i terminalen mellom hver gang du skal opprette ny .bin fil eller .hex fil. Da renser du opp og sletter tidligere filer som har blitt generert av make kommandoen slik at nye konfigurasjoner som er gjort i Contiki blir tatt neste gang du skal opprette contiki image (.bin/.hex).
4. make TARGET=srf06-cc26xx BOARD=launchpad/cc1350 cc26xx-web-demo.bin
5. Flytt den nylig opprettede border-router.bin (eller .hex) til skrivebordet
6. Bruk FlashProgrammer 2 til å flashe CC1350 med cc26xx-web-demo.bin

## Tunslip & CoAP

For å få border-router til å rute trafikken og gjøre informasjon som kommer inn fra nodene tilgjengelig, trenger vi en SLIP-radio løsning. I Contiki ligger tunslip som et verktøy for akkurat dette. COAP er også et verktøy i Contiki, det er en tjeneste som gir toveis kommunikasjon til noden og setter opp en lokal webpage. På våre brett(CC1350) kan vi måle batteristyrke, temperatur og rssi.

**TIPS:** Det er viktig at CC1350border-router er koblet til WMware (kobles til under player->removable devices-> «din CC1350»)



Figur 53: Koble CC1350 til Contiki

1. I terminal.
2. cd git-repo/Contiki/tools
3. make tunslip6
4. sudo /tunslip6 aaaa:1/64 -s /dev/ttyACM0
5. Om du ikke skulle få noe informasjon ut kan det hende at du kjører fra feil port. Prøv da med: sudo /tunslip6 aaaa:1/64 -s /dev/ttyACM1
6. Når border-router kjører fra riktig port vil det se slik ut:

```
user@instant-contiki:~/contiki/tools
user@instant-contiki:~/contiki/tools$ sudo ./tunslip6 aaaa::1/64 -s /dev/ttyUSB1
*****SLIP started on `/dev/ttyUSB1'
opened tun device `/dev/tun0'
ifconfig tun0 inet 'hostname' mtu 1500 up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:1/64
ifconfig tun0

tun0      Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.1.1 P-t-P:127.0.1.1 Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
             inet6 addr: aaaa::1/64 Scope:Global
               UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
               RX packets:0 errors:0 dropped:0 overruns:0 frame:0
               TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
               collisions:0 txqueuelen:500
               RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

**Figur 54:** Starte BR i Contiki

7. Start en node som kjører på samme kanal som border router og vent til de har fått stabil kontakt (det grønne lyset slutter å blinke)
8. Trykk på reset knappen for å dumpe ipv6 adressen:

```
user@instant-contiki:~/contiki/tools
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Starting Contiki-3.x-1659-g103883d
With DriverLib v0.44336
TI SmartRF06EB + CC13xx EM
IEEE 802.15.4: Yes, Sub-GHz: Yes, BLE: Yes, Prop: Yes
Net: sicslowpan
MAC: CSMA
RDC: ContikiMAC, Channel Check Interval: 16 ticks
RF: Channel 25
RPL-Border router started
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Setting prefix aaaa::
created a new RPL dag
Server IPv6 addresses:
aaaa::212:4b00:7c6:2b02
fe80::212:4b00:7c6:2b02
Starting Contiki-3.x-1659-g103883d
With DriverLib v0.44336
TI SmartRF06EB + CC13xx EM
IEEE 802.15.4: Yes, Sub-GHz: Yes, BLE: Yes, Prop: Yes
Net: sicslowpan
MAC: CSMA
RDC: ContikiMAC, Channel Check Interval: 16 ticks
RF: Channel 25
RPL-Border router started
*** Address:aaaa::1 => aaaa:0000:0000:0000
```

**Figur 55 :**Ipv6 adresse BR

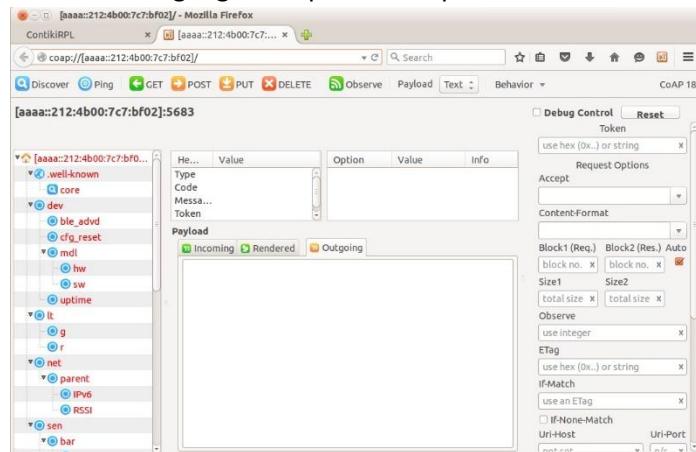
9. Åpne firefox i WMware og skriv inn denne adressen slik [aaaa:212:2b00:7c6:2b02] (bytt ut denne med din egen)

**Figur 56:** Firefox BR oversikt over noder

10. Ta ipv6 adressen som websiden lister opp og skriv i en ny fane:

coap://[ aaaa:212:4b00:7c7:bf02]/ (bytt ut med din adresse)

Du vil nå få tilgang til coap serveren på din node



Figur 57: Firefox COAP server

## Oppsett av OpenVPN for Ubuntu 16 server (fra DigitalOcean) og klienter (BBG)

Nyttige lenker:<https://www.digitalocean.com/community/tutorials/how-to-set-up-an-openvpn-server-on-ubuntu-16-04#prerequisites>

NB! Denne steg-for-steg guiden er basert på lenken over, men tilpasset vårt prosjekt.

### Installere openvpn på server

- Skriv i serverens terminal:

```
sudo apt-get update  
sudo apt-get install openvpn easy-rsa
```

Vent til installeringen er ferdig før du forsetter...

### Sett opp CA directory

- Kopier easy-rsa med make-cadir kommandoen:

```
make-cadir ~/openvpn-ca
```

- Naviger til den nylig opprettede mappen:

```
cd ~/openvpn-ca
```

### Konfigurer CA variabler

- Åpne var filen som ligger i mappen:

```
nano vars
```

- Scroll nedover i filen til du kommer til dette:



```
~/openvpn-ca/vars  
...  
  
export KEY_COUNTRY="US"  
export KEY_PROVINCE="CA"  
export KEY_CITY="SanFrancisco"  
export KEY_ORG="Fort-Funston"  
export KEY_EMAIL="me@myhost.mydomain"  
export KEY_OU="MyOrganizationalUnit"  
...
```

Figur 58: OpenVPN konfigurasjon- CA variabler

Endre til ønskelige verdier. For eksempel:

```
# These are the default values for fields
# which will be placed in the certificate.
# Don't leave any of these fields blank.
export KEY_COUNTRY="NORWAY"
export KEY_PROVINCE="HORDALAND"
export KEY_CITY="Bergen"
export KEY_ORG="HVL"
export KEY_EMAIL="h146221@stud.hvl.no"
export KEY_OU="Uni"
```

Figur 59: OpenVPN konfigurasjon- CA variabler eksempel

Endre også verdien til KEY\_NAME til server:

```
# X509 Subject Field
export KEY_NAME="server"
```

Figur 60: OpenVPN konfigurasjon- Key name

## Build Certificate Authority (CA)

- Naviger til ca mappen:  
cd ~/openvpn-ca
- Source vars filen, skriv i terminal:  
source vars

Hvis det fungerte skal outputet være:

Output

NOTE: If you run ./clean-all, I will be doing a rm -rf on  
/home/sammy/openvpn-ca/keys

- Skriv så i terminalen:

```
./clean-all
./build-ca
```

Det er ikke nødvendig å legge inn nye verdier for province, by osv. siden dette allerede er lagt inn i vars filen vi redigerte i begynnelsen av steg 3. Hvis du må legge inn to bokstaver for landkode kan du legge inn «NO» for Norge. Ellers er det bare å trykke Enter for å bekrefte forhåndstilpassede verdier.

## Lag server sertifikat, nøkkel og kryptering

- Start med å lage sertifikat og nøkkelpar. Skriv i terminalen:

```
./build-key-server server
```

Som tidligere er det bare å trykke Enter for å bekrefte forhåndstilpassede verdier fra vars filen.

**NB! Ikke legg til challenge password når du blir spurt om dette**

Til slutt må du svare 'y' på to spørsmål for å gjøre ferdig oppsettet:

```
Certificate is to be certified until May 1 17:51:16 2026 GMT (3650 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Figur 61: OpenVPN konfigurasjon- key server

- Det neste vi må gjøre er å generere Diffie-Hellman nøkler. Skriv i terminal:  
./build-dh  
**NB! Det kan ta en god stund å generere dh-nøklene.**
- Så må vi generere HMAC signatur for å styrke serverens TLS integritets verifikasjons muligheter. Skriv i terminal:  
openvpn --genkey --secret keys/ta.key

## Generer client sertifikat og nøkkelpar

- For å generere nøkler uten passord beskyttelse skriv i terminal:

```
cd ~/openvpn-ca
source vars
./build-key client1
```

Client1 er her bare et eksempel. Du kan kalle den hva du vil. Igjen må du trykke Enter for å slippe å skrive inn nye verdier.

**NB!** Hvis du ønsker passord på nøklene skriver du istedenfor:

```
cd ~/openvpn-ca
source vars
./build-key-pass client1
```

## Konfigurer OpenVPN som service

- Vi må kopiere filene vi trenger til ~/etc/openvpn. Skriv i terminal:

```
cd ~/openvpn-ca/keys
```

```
sudo cp ca.crt ca.key server.crt server.key ta.key dh2048.pem  
/etc/openvpn
```

- Så må vi kopiere og unzipe en sample konfigurasjons fil for OpenVPN, server.conf, til ~ /etc/openvpn. Skriv i terminal:

```
gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz | sudo tee  
/etc/openvpn/server.conf
```

- Vi må nå endre litt på server.conf filen. Skriv i terminal:

```
sudo nano /etc/openvpn/server.conf
```

- Scroll (eller søk med ctrl-w) til du finner HMAC seksjonen. Der skal det være en linje som inneholder tls-auth. Fjern kommentaren foran denne linjen

I tillegg må du legge til: key-direction 0

Det skal se slik ut:

```
# The second parameter should be '0'  
# on the server and '1' on the clients.  
tls-auth ta.key 0 # This file is secret  
key-direction 0
```

Figur 62: OpenVPN konfigurasjon- key direction

- Finn så linjen: ;cipher AES-128-CBC  
Fjern ; foran linjen.
- Under denne legger du til en ny linje:  
auth SHA256
- Finn så user og group og fjern ; foran de respektive linjene:

```
user nobody  
group nogroup
```

- Finn så redirect-gateway seksjonen og fjern ; foran linjen:  
push "redirect-gateway def1 bypass-dhcp"
- Finn så dhcp-options seksjonen. Fjern ; foran linjene:  
push "dhcp-option DNS 208.67.222.222"  
push "dhcp-option DNS 208.67.220.220"

NB! Dette medfører at all nettverkstrafikk fra clienten (bbg) går gjennom serveren.

- Finn linjene for port og proto og endre verdiene til:

```
port 443  
proto tcp
```

- Dobbelsjekk at linjene for cert og key ser slik ut:  
cert server.crt  
key server.key
- Lagre og gå ut av filen

## Endre serverens nettverks konfigurasjon

- Vi må gjøre noen endringer i /etc/sysctl.conf filen for at serveren skal få forwarde trafikk.  
Skriv i terminalen:  
`sudo nano /etc/sysctl.conf`
- Finn linjen: `#net.ipv4.ip_forward`. Fjern `#` foran linjen.
- Lagre og lukk filen.
- For å lese filen å endre verdier for gjeldene session, skriv i terminalen:  
`sudo sysctl -p`
- Nå trenger vi å finne public network interface for vår maskin. Skriv i terminalen:  
`ip route | grep default`
- Navnet på public network interface kommer etter dev i outputet. Her er et eksempel der public network interface har navnet wlp11s0:

```

Output
default via 203.0.113.1 dev wlp11s0 proto static metric 600

```

Figur 63: OpenVPN konfigurasjon- public network interface

Legg merke til og noter ned navnet på ditt public network interface.

- Gå så til /etc/ufw/before.rules filen, skriv i terminalen:  
`sudo nano /etc/ufw/before.rules`
- Legg til linjene skrevet i rødt i bildet under:

```

/etc/ufw/before.rules

#
# rules.before
#
# Rules that should be run before the ufw command line added rules. Custom
# rules should be added to one of these chains:
#   ufw-before-input
#   ufw-before-output
#   ufw-before-forward
#

# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to eth0
-A POSTROUTING -s 10.8.0.0/8 -o eth0 -j MASQUERADE
COMMIT
# END OPENVPN RULES

# Don't delete these required lines, otherwise there will be errors
*filter
...

```

Figur 64: OpenVPN konfigurasjon- rules

- Lagre og lukk filen
- Så må vi la UFW få lov til å forwarde pakker. Åpne /etc/default/ufw, skriv i terminal:

```
sudo nano /etc/default/ufw
```

- Finn linjen : DEFAULT\_FORWARD\_POLICY og endre «DROP» til «ACCEPT». Slik som dette:

```
/etc/default/ufw  
DEFAULT_FORWARD_POLICY="ACCEPT"
```

Figur 65: OpenVPN konfigurasjon- accept

- Det neste vi må gjøre er å endre brannmuren på serveren slik at den slipper OpenVPN trafikk gjennom. Skriv i terminal:  
`sudo ufw allow 443/tcp`  
`sudo ufw allow OpenSSH`
- Nå må vi bare re-enable ufw for at endringene skal tas i bruk. Skriv i terminal:  
`sudo ufw disable`  
`sudo ufw enable`

## Start OpenVPN-server

- Nå kan vi endelig starte OpenVPN serveren. Skriv i terminal:  
sudo systemctl start openvpn@server
- Vi kan sjekke status på serveren ved å skrive:  
sudo systemctl status openvpn@server

Hvis alt har gått som det skal vil outputet være noe slikt:

```
Output
● openvpn@server.service - OpenVPN connection to server
  Loaded: loaded (/lib/systemd/system/openvpn@.service; disabled; vendor preset: enabled)
  Active: active (running) since Tue 2016-05-03 15:30:05 EDT; 47s ago
    Docs: man:openvpn(8)
          https://community.openvpn.net/openvpn/wiki/Openvpn23ManPage
          https://community.openvpn.net/openvpn/wiki/HOWTO
   Process: 5852 ExecStart=/usr/sbin/openvpn --daemon ovpn-%i --status /run/openvpn/%i.status 1
 Main PID: 5856 (openvpn)
   Tasks: 1 (limit: 512)
  CGroup: /system.slice/system-openvpn.slice/openvpn@server.service
          └─5856 /usr/sbin/openvpn --daemon ovpn-server --status /run/openvpn/server.status 1

May 03 15:30:05 openvpn2 ovpn-server[5856]: /sbin/ip addr add dev tun0 local 10.8.0.1 peer 10.8.0.2
May 03 15:30:05 openvpn2 ovpn-server[5856]: /sbin/ip route add 10.8.0.0/24 via 10.8.0.2
May 03 15:30:05 openvpn2 ovpn-server[5856]: GID set to nogroup
May 03 15:30:05 openvpn2 ovpn-server[5856]: UID set to nobody
May 03 15:30:05 openvpn2 ovpn-server[5856]: UDPv4 link local (bound): [undef]
May 03 15:30:05 openvpn2 ovpn-server[5856]: UDPv4 link remote: [undef]
May 03 15:30:05 openvpn2 ovpn-server[5856]: MULTI: multi_init called, r=256 v=256
May 03 15:30:05 openvpn2 ovpn-server[5856]: IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
May 03 15:30:05 openvpn2 ovpn-server[5856]: IFCONFIG POOL LIST
May 03 15:30:05 openvpn2 ovpn-server[5856]: Initialization Sequence Completed
```

Figur 66: OpenVPN konfigurasjon- openVPN status

- Det er også mulig å sjekke at OpenVPN tun0 grensesnittet er tilgjengelig ved å skrive:

```
ip addr show tun0
```

Her skal outputet se slik ut:

```
bachelor@bwsensorlab:~$ ip addr show tun0
10: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 100
  link/none
  inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
    valid_lft forever preferred_lft forever
bachelor@bwsensorlab:~$
```

Figur 67: OpenVPN konfigurasjon- tun0

- Hvis alt har gått som det skal kan vi enable OpenVPN som service slik at det starter automatisk på boot. Skriv i terminalen:

```
sudo systemctl enable openvpn@server
```

## Lag infrastruktur for client konfigurering

- Lag en mappe struktur for å lagre client filene vi skal opprette senere. Skriv i terminal:

```
mkdir -p ~/client-configs/files
```

- Siden client konfigurasjonen har client nøkler innebygd må vi låse adgangen til strukturen. Skriv i terminal:

```
chmod 700 ~/client-configs/files
```

- Lag en base konfigurasjon for klienter. Skriv i terminal:

```
cp /usr/share/doc/openvpn/examples/sample-config-files/client.conf  
~/client-configs/base.conf
```

- Åpne base konfigurasjonen i nano. Skriv i terminal:

```
nano ~/client-configs/base.conf
```

- Trykk ctrl+w og skriv i søkefeltet: remote
- Endre linjen som begynner med romte slik at den er slik det er vist i rødt under:

```
...  
# The hostname/IP and port of the server.  
# You can have multiple remote entries  
# to load balance between the servers.  
remote server_IP_address 443  
...
```

NB! I vårt tilfelle er serveren på digitalocean sin IP, server\_IP\_adress: 207.154.220.121

- Endre så linjen som begynner med proto til:  
`proto tcp`
- Fjern så kommenteringen for user og group. Det skal se slik ut:

```
# Downgrade privileges after initialization (non-Windows only)  
user nobody  
group nogroup
```

- Finn så linjene som beskriver ca, cert og key og kommenter de ut. Det skal se slik ut:

```
# SSL/TLS parms.  
# See the server config file for more  
# description. It's best to use  
# a separate .crt/.key file pair  
# for each client. A single ca  
# file can be used for all clients.  
#ca ca.crt  
#cert client.crt  
#key client.key
```

- Se så til at linjene som beskriver cipher og auth er lik som i /etc/openvpn/server.conf. Det skal se slik ut:

```
cipher AES-128-CBC  
auth SHA256
```

- Legg så til key-direction et eller annet sted i filen og sett den til 1. Skal se slik ut:  
`key-direction 1`

- Legg så til et par linjer som skal være kommentert ut. Vi skal bruke disse linjene etter vi sender kopier av denne filen til klientene. Linjene som skal legge til er:

```
# script-security 2  
# up /etc/openvpn/update-resolv-conf  
# down /etc/openvpn/update-resolv-conf
```

- Lagre og gå ut av filen: Ctrl + o, ctrl + x
- Lag en ny fil, make\_config.sh, i ~/client-config mappen. Skriv i terminal:

```
nano ~/client-configs/make_config.sh
```

- Kopier inn dette i den nylig opprettede make\_config.sh filen:

```
#!/bin/bash
```

```
# First argument: Client identifier  
  
KEY_DIR=~/openvpn-ca/keys  
OUTPUT_DIR=~/client-configs/files  
BASE_CONFIG=~/client-configs/base.conf
```

```
cat ${BASE_CONFIG} \
<(echo -e '<ca>') \
${KEY_DIR}/ca.crt \
<(echo -e '</ca>\n<cert>') \
${KEY_DIR}/${1}.crt \
<(echo -e '</cert>\n<key>') \
${KEY_DIR}/${1}.key \
<(echo -e '</key>\n<tls-auth>') \
${KEY_DIR}/ta.key \
<(echo -e '</tls-auth>') \
> ${OUTPUT_DIR}/${1}.ovpn
```

- Lagre og lukk filen: ctrl + o, ctrl + x
- Gjør filen kjørbar. Skriv i terminal:

```
chmod 700 ~/client-configs/make_config.sh
```

## Generer client konfigurasjon

- Skriv i terminal:

```
cd ~/client-configs
./make_config.sh client1
```

- Hvis alt går som det skal vil det ligge en client1.ovpn fil i ~/client-configs. For å sjekke skriv i terminal:

```
ls ~/client-configs/files
```

Output:

```
client1.ovpn
```

## Installering av client konfigurering og overfør fil fra server til client

- Start opp BBG (client) og logg inn.
- Se til at den har internett tilkobling og hent filen client1.ovpn fra server ved å skrive i terminalen:

```
sftp bachelor@openvpn_server_ip:client-configs/files/client1.ovpn  
~/
```

Dette plasserer client1.ovpn i home mappen.

NB! I vårt tilfelle er serveren på digitalocean sin IP, server\_IP\_adress: 207.154.220.121

- Installer openvpn på BBG (client). Skriv i terminal:

```
sudo apt-get update
```

```
sudo apt-get install openvpn
```

- Sjekk at du har update-resolv-conf i ~/etc/openvpn. For å sjekke skriv i terminal:

```
ls /etc/openvpn
```

Ouput:

```
update-resolv-conf
```

- Endre client1.ovpn filen. Skriv i terminal:

```
nano client1.ovpn
```

- Fjern kommentering på følgende linjer:

```
script-security 2  
up /etc/openvpn/update-resolv-conf  
down /etc/openvpn/update-resolv-conf
```

NB! Dette fungerer kun hvis du har update-resolv-conf i ~/etc/openvpn/ mappen

- Lagre og lukk filen
- Nå kan vi starte openvpn på client-siden. Skriv i terminal:

```
sudo openvpn --config client1.ovpn
```

- Nå skal clienten være koblet til serveren. Dette kan sjekkes hvis du skriver i terminalen på server siden:

```
sudo cat /etc/openvpn/openvpn-status.log
```

Output:

OpenVPN CLIENT LIST

Updated,Thu Mar 9 13:01:22 2017

Common Name,Real Address,Bytes Received,Bytes Sent,Connected Since

client1,158.37.234.210:62726,6057,6523,Thu Mar 9 13:01:18 2017

ROUTING TABLE

Virtual Address,Common Name,Real Address,Last Ref

10.8.0.10,client1,158.37.234.210:62726,Thu Mar 9 13:01:21 2017

GLOBAL STATS

Max bcast/mcast queue length,0

END

Dette kan selvfølgelig gjøres på flere klienter ☺

## Autostart client-openvpn tilkobling til server

Nyttige lenker: <http://askubuntu.com/questions/464264/starting-openvpn-client-automatically-at-boot>

- Flytt client.ovpn filen fra root til /etc/openvpn/. For å flytte filen, nавигer til mappen der .ovpn filen ligger og skriv i terminal:

```
mv client.ovpn /etc/openvpn/
```

- Nавигer til /etc/openvpn, i terminal: cd /etc/openvpn/

- Endre filnavnet fra .ovpn til .conf. For å endre navn skriv i terminalen:

```
mv client.ovpn client.conf
```

- Nавигer til ~/etc/default/ og åpne openvpn filen i nano. I temrinal, på root, skriv:

```
nano /etc/default/openvpn
```

- Finn linjen: #AUTOSTART=all og fjern kommenteringen, #

```
# This is the configuration file for /etc/init.d/openvpn

#
# Start only these VPNs automatically via init script.
# Allowed values are "all", "none" or space separated list of
# names of the VPNs. If empty, "all" is assumed.
# The VPN name refers to the VPN configuration file name.
# i.e. "home" would be /etc/openvpn/home.conf
#
# If you're running systemd, changing this variable will
# require running "systemctl daemon-reload" followed by
# a restart of the openvpn service (if you removed entries
# you may have to stop those manually)
#
#AUTOSTART="all"
#AUTOSTART="none"
#AUTOSTART="home office"
#
# WARNING: If you're running systemd the rest of the
```

Figur 68: OpenVPN konfigurasjon- Autostart

Lagre og restart beaglebone. Da vil /etc/default/openvpn filen starte client.conf på oppstart. Dvs at BBG alltid vil rute nettverkstrafikken gjennom den sentrale serveren og ha mulighet for å tunellere (tls/ssl- eller ssh-protokoll) data til og fra server.

## **Appendiks D      Kildekoder og Bill of Materials**

### **D.1            Kildekode**

Kildekoder ligger som vedlegg i mappen Kildekoder-BO17E-12.zip. Kodene er også tilgjengelig på [https://github.com/Jorgenhoyer/Bachelor\\_v2017](https://github.com/Jorgenhoyer/Bachelor_v2017).

### **D.2            Bill of Materlials**

Se vedlagt excel dokument, Bill of Materials.xls