BACHELOROPPGAVE:

BO19E-38 STANDARDISERT IOT

Aleksander Husebø
Christoffer Angeltvedt Wollertsen

31. mai. 2019

Dokumentkontroll

Rapportens tittel: BO19E-38 Standardisert IOT	*Dato/Versjon* 31. mai. 2019
	Rapportnummer: B019E-38
Forfatter(e): Aleksander Husebø Christoffer Angeltvedt Wollertsen	*Studieretning:* HKOM
	Antall sider m/vedlegg 16
Høgskolens veileder: Andreas Ramstad Urke	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Høgskolen på Vestlandet	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): Andreas Ramstad Urke 92054062 arur@hvl.no	

Revisjon	**Dato**	**Status**	**Utført av**
0.11	31.01.19	Første utkast	Aleksander Husebø Christoffer Angeltvedt Wollertsen
0.2	26.05.19		Aleksander Husebø Christoffer Angeltvedt Wollertsen
0.21	29.05.19		Aleksander Husebø Christoffer Angeltvedt Wollertsen

1 Innhold

1 Innledning

1.1 Oppdragsgiver

Oppdragsgiver er Høgskolen på Vestlandet(HVL), campus Bergen

HVL ble opprettet gjennom en sammenslåing av høgskolen i Bergen, Sogn og Fjordane, og Stord/Haugesund. Sammenslåingen ble utført 1 januar 2017, men skolen har beholdt de individuelle studiestedene.

1.2 Problemstilling

Hovedmålet med oppgaven er å utforske IPv6 over the TSCH mode of IEEE 802.15.4e (6tisch), og utføre en proof-of-concept test av standarden. 6TiSCH er en samling åpne standarder for å knytte energi konservative trådløse enheter mot internett med høy pålitelighet, altså rettet mot Internet of things (IoT). Det finnes andre IoT løsninger også, som ZigBee og WirelessHart, men i motsetning til 6TiSCH er disse lukket.

Proof of concept testen vil bli utført ved å bruke billige sensor-noder som skal bruke 6TiSCH, en raspberry pi som skal fungere som aksesspunkt, og operativsystemet Contiki-NG som støtter 6TiSCH.

En potensiell utvidelse vil være å sende sensor-data til en skyløsning for å visualisere den, og vise at den nåværende 6TiSCh versjonen fungerer i et reelt og nyttig scenario.

De fleste utfordringene i oppgaven kommer av at teknologien er under utvikling. Mye av dokumentasjonen til 6TiSCH er i store, tekniske dokumenter, og Contiki-NG har en relativt komplisert installasjonsmetode.

1.3 Hovedidé for løsningsforslag

HVL vil utforske 6TiSCH med et node-nettverk, der vi skal dokumentere stegvis både egne erfaringer og eksisterende løsninger som brukes i testing eller oppsett av nettverket. Dette sensornettverket skal så brukes i et mer nyttig test-scenario, der parkeringssensorer har blitt foreslått. Det er ikke viktig hva de spesifikt brukes til så lenge det er en god nokk test til å utforske modenheten og funksjonaliteten til 6TiSCH.

Vi har valgt å bruke lyssensor til å måle aktiviteten i et grupperom. Vi valgte dette scenarioet fordi parkeringssensorscenarioet ble for komplisert med de sensorene vår sensortag har.

2 Kravspesifikasjon

Sensor-node: Noden skal måle sine sensorer og sende dataene til en server ved å benytte 6TiSCH stacken. Vi bruker Texas Instruments CC2650 Sensortag som har en rekke sensorer (inkludert lyssensor), den støtter Contiki-NG, er svært liten, og er ikke kostbar. For eksempel kan lys-sensoren muligens brukes som parkeringssensor, men sensortaggen kan også brukes til mer eller mindre ambisiøse bruksområder.

Gateway: Markerer grensesnittet og videresender trafikk mellom det trådløse 6TiSCH nettverket og et annet nettverk. Den må dermed også kjøre 6TiSCH stacken, og være koblet til et annet nettverk. Vi bruker en Raspberry Pi + CC2650 Launchpad.

Software: På node og gateway benyttes Contiki-NG som støtter 6TiSCH stacken. På selve Raspberry Pi-en bruker vi vanlig Raspbian.

Data-behandling: En egen-utviklet løsning som gjør dataen fra nodene tilgjengelig for brukere. Programmet skrives i Python og skal logge sensornavn, sensordata og tidspunkt i en tekstlogg.

Dokumentering: Prosessen skal dokumenteres stegvis, med spesielt fokus på erfaringer med implementering, grad av kompleksitet for å få et fungerende system, og hvor lang tid det tar å oppnå resultater.

3 Analyse av problemet

Det skal undersøkes om hvor modent 6TiSCH er som en standard. Hva som må til for å kunne sette opp et nettverk som benytter 6TiCH og som er knyttet opp mot en skytjeneste

3.1 Utforming av mulige løsninger

HW som skal brukes er, spesifisert av HVL, Sensortag som trådløse noder, og en Launchad som mottaker, såkalt border router, koblet til en datamaskin med internett-tilkobling via SLIP. De trådløse nodene skal kommunisere med 6TiSCH og RPL.

Hver node skal flashes slik at den støtter 6TiSCH og RPL og kobles i et nettverk. Disse nodene er koblet til en mottaker som fungerer som border router, og sender sensordata ut av 6TiSCH nettverket i et leselig format til en server.

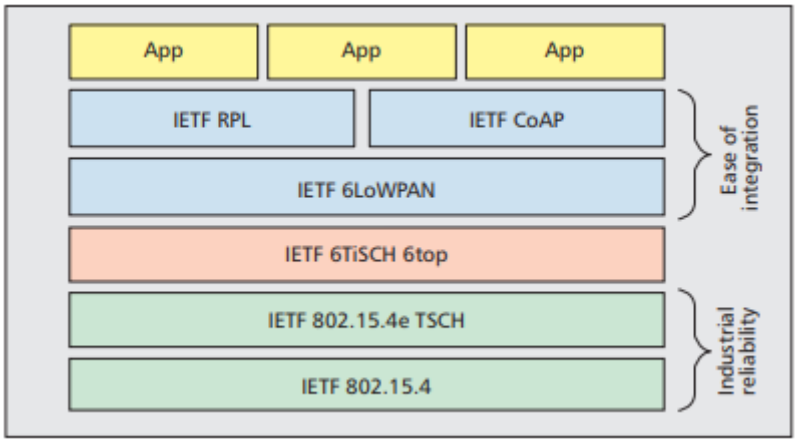
Programvaren som flashes på nodene og mottaker må utvikles delvis av oss. Contiki-NG [1] stiller med rammerverk. Programvaren på Raspberry pi må helt eller delvis utvikles selv.

3.1.1 Vurderinger i forhold til verktøy og HW/SW komponenter

Det er to tilgjengelige programvarer tilgjengelig for å flashe til mikrokontrollere, UniFlash [1] og SmartRF Flash programmer 2 [2]. Vi har valgt å bruke SmartRF Flash fordi den fungerte uten problemer mens UniFlash ikke hadde drivere inkludert. Uniflash ble også vurdert å bruke direkte på Rasperry Pi, men det støttes ikke. Det er krav om en 64-bit prosessorarkitektur, men Raspberry PI har en 32-bit arkitektur.

For å generere filene til å flashe Sensortagene og Launchpadene har det vært brukt Contiki-NGs miljø gjennom et Docker bilde. Det har ikke vært noe alternativ til dette da vi har brukt SmartRF Flash som kun støtter Windows, Docker bildet er også den anbefalte måten av Contiki-NG

4 Teori



[3]

6TiSCH stacken

4.1 6TiSCH overordnet teori

IPv6 over TSCH er en samling åpne protokoller for LLN/IoT. Hoveddelene av den er 6LoWPAN, TSCH, 6top, RPL og CoAP. Alle protokollene har sin egen rolle i 6TiSCH.

6LoWPAN: 6LoWPAN er «IPv6 over low-power wireless personal area networks». 6LoWPAN inneholder blant annet mekanismer som utfører header-kompresjon og innkapsulering. Dette gjør det mulig å bruke IPv6 i «constrained-devices», ettersom IPv6 headeren vanligvis er for stor til å møte kravene til en constrained device. Kompresjonen skjer i Border router, altså grensen mellom sensornettverket og det «vanlige» nettet. Hvis IPv6 pakkene har flere headere kan 6LoWPAN komprimere både den eksterne og interne headeren. Generelt har IPv6 pakker en minste størrelse på 1280 oktetter, og headeren er 40 oktetter, noe som er mye større enn IEEE 802.15.4 MTU på 127 bytes/oktetter. Med 6LoWPANs header kompresjon kan IPv6 headeren komprimeres til 7 bytes/oktetter. [4]

TSCH: Nodenettverk som bruker 6TiSCH forventes å ha høy skalerbarhet. Det er derfor viktig at frekvensspekteret blir fordelt, enten manuelt eller automatisk, slik at det ikke blir overlapp hos nærliggende noder.

Time slotted channel hopping, eller time syncronised channel hopping er en kombinasjon av Time division multiple access(TDMA) og frequency-devision multiple access(FDMA), og sender pakker delt over både tid og frekvens. Alle nodene i et nettverk vil bli tildelt en schedule og vil dermed vite når den skal sende eller motta informasjon, og når den kan skru radioen av for å spare strøm. Hvis det blir kollisjoner mellom pakker vil noden bytte frekvens og prøve på nytt hvis dette er tillatt, derav «channel hopping». Denne løsningen er valgt da det er mer sannsynlig at pakkene når fram enn hvis noden bytter timeslot.

Hard cells: Hard cells er celler som har fått en designert plass av en ekstern kontrollenhet og ikke kan flyttes dynamisk. De har en fastsatt slotOffset og channelOffset. [5]

Soft cells: Soft cells er celler som kan flyttes dynamisk i TSCH schedule. Disse cellene overvåker sin egen ytelse og forhandler fram en ny plass i schedulen hvis de ikke oppnår den ønskede ytelsen. I motsetning til hard cells har ikke soft cells fastsatt slotOffset og channelOffset. I stedet har de krav for båndbredde og QoS. [5]

For å enkelt kunne kommunisere og forhandle mellom noder har alle nodene en mottaker celle der slotOffset og channelOffset er basert på en hash av nodens MAC adresse. [6]

6top er en logical link control/logisk link kontroller, som sitter mellom IP laget, og MAC laget. Denne protokollen gir blant annet muligheten til å kontrollere schedulen eksternt.

Scheduling mechanisms:

Det er 4 måter å håndtere TSCH schedules på i 6TiSCH node nettverket. Disse fire er Static scheduling, neighbor-to-neighbor scheduling, remote monitoring and scheduling managment, og hop-by-hop scheduling. Det er lagt opp til at man skal kunne bruke flere av disse implementasjonene samtidig

Static scheduling er den enkleste implementasjonen og er som navnet tilsier en statisk schedule, forhåndsbestemt av nettverksadministrator. Denne schedulen kan være nyttig når et nytt nettverk blir opprettet, eller som en sikring i tilfelle de andre implementasjonene/tidsplanene skulle feile. [7]

neighbor-to-neighbor scheduling bruker neighbor oversikten til å forhandle fram en schedule og dynamisk adaptere båndbredden etter behov [8]

remote monitoring and scheduling managment går i hovedsak ut på å styre og overvåke nodene fra en sentral plass. Dette vil ofte være den mest nyttige implementasjonen da endringer kan gjøres enkelt fra et kontor, uten å behøve å gå til hver enkelt node. [9]

Hop-by-hop schedule benytter seg av en rekke av soft cells, og lager en bane fra node til node. Hver enkelt node vet hvilke celler de skal motta pakker og hvilke de skal bruke for å sende pakkene videre. En rekke med hop fra avsender til destinasjon blir kalt et «track», og kan inneholde en til mange noder. Hensikten med et track er å reservere celler hos nodene mellom avsender og destinasjon, slik at det ikke blir noe tap forårsaket av at noden er opptatt. Dette gir også fordeler for batteriet da noden kan kjøre i sleep/low-power mode når det er de ubrukte cellene sin tur.

RPL: RPL er en IPv6 protokol for LLNs. RPL bruker en DODAG topologi som starter fra root node og lager en tre struktur til de nodene. Root noden vil i de fleste tilfeller også være border router, men den må ikke være det om det er behov for at den skal være en annen. RPL bruker en objective function til å sette mål for topologien, og bygger topologien etter den. RPL støtter flere senderettninger, upward routing: fra hvilken som helst node til root, downward routing: fra root til hvilken som helst node, og any-to-any routing: der nodene vil sende pakkene mot root til de kommer til nærmeste delte node og deretter til destinasjonen. [10]

CoAP: CoAP, constrained application protocol, er en web overførings protokoll for constrained networks. CoAP gir nodene i nettverket en request/respons funksjon slik at en bruker enkelt kan

hente inn sensordata når de ønsker det. CoAP er bygget og designet slik at den enkelt kan brukes med vanlige webløsninger og i HTTP, men møter også kravene for LLN som liten header og simpelhet. [11]

4.2 6TiSCH Minimal

6TiSCH minimal mode er betegnelsen på den simpleste formen for implementasjon for 6TiSCH. Den inneholder de anbefalte protokollene og konfigurasjonene for å få 6TiSCH til å fungere. I IETFs 6TiSCH minimal draft vil en 6TiSCH implementasjon bli regnet som minimal mode dersom 802.15.4 TSCH mode, 6LoWPAN rammeverket, RPL, og objective function zero(OF0) blir brukt umodifisert.

De umodifiserte innstillingene er:

TSCH mode: Denne implementasjonen bruker en enkelt slotframe. En node som kjører minimal mode vil få informasjon om størrelsen på slotframen via enhanced beacon. Noden får også en enkelt celle markert ADVERTISING for å sende eller motta enhanced beacon. Når cellen ikke sender vil den høre etter innkommende pakker, og bare bytte til sendingsmodus når den har en pakke å sende. Når noden sender vil den spørre om en acknowledgement, hvis den ikke får acknowledgement vil den vente en stund før den prøver igjen. I 6TiSCH minimal er anbefalt mengde forsøk 3 retransmissions, så totalt 4 sendinger.

6LoWPAN: Rammeverket blir implementert som det er i en full 6TiSCH versjon. Det kan forventes at en spesifikk konfigurasjon vil komme i framtiden.

RPL: 6TiSCH minimal må implementere OF0. OF0 bruker bare informasjon fra RPL og er ikke avhengig av utenforliggende protokoller og funksjoner.

5 Utførelse og testing

5.1 Første gang

Den aller første testen var å bli kjent med byggemiljøet. Det finnes et demoeksempel til platformene vi bruker som kringkaster navnet på plattformen, f.eks Sensortag, via Bluetooth. Demoeksempelet demonsterer mye mer enn bare dette, men for vår første test var dette nok. Vi støtte på et problem ved kompilering av demoeksempelet da det var en skrivefeil i makefilen. Denne ble redigert bort og eksempelet fungerte. Feilen har siden blitt rettet av Contiki-NG

5.2 Hello world

Vi testet hello world eksempelet på sensortaggene. Eksempelprogrammet gjør ikke så mye annet enn å printe hello world til seriell porten, men det viser også at Contiki-NG virker som det skal, og har blant annet fungerende RPL implementasjon i likhet med alle Contiki-NG eksemplene.

For å verifisere at hello world eksempelet fungerer kan man lese serielldata. Baudraten på seriellporten var ikke nevnt i brosjyrene som fulgte med, heller ikke på Tis sider. Dette førte til at serielloutput ikke var tekst. Vi fant til slutt ut at baudraten var 115200, og fikk da tekst i terminalen. Contiki-NG bruker kun LF som end-of-line-karakter. Dette er ikke standard i hverken Putty eller Teraterm, som forventer CR. Resultatet er at hver nye linje i terminalen blir forskjøvet mot høyre, som trappetrinn.

5.3 Shell

Ved å legge til linjen «MODULES += os/services/shell» i prosjektets makefile, kompilere og flashe på nytt fikk vi lagt til shell på hello world nodene. Med shell skulle vi kunne sende ping fra nodene, se routing tables og RPL rute informasjon. Vi brukte Putty til å koble til nodene via seriell og fikk opp hello world på skjermen, men shell responderte ikke. Etter å ha lest litt videre om shell fant vi ut Contiki-NG seriell linje kode bare gjenkjenner LF karakteren, og ikke CR karakteren som Putty sender. Vi løste dette med å bruke Teraterm til å koble til serielt i stedet for Putty. I Teraterm kan man velge om man skal sende CR, LF eller begge deler. Når vi brukte Teraterm og sendte riktig end-of-line-karakter fungerte shell som det skulle og vi fikk tilgang til kommandoene vi var ute etter, men Contiki-NG viser ikke karakterene som blir skrevet i terminalen så instillingen «local echo» måtte slås på.

5.4 TSCH

Ved å legge til linjen «MAKE_MAC = MAKE_MAC_TSCH» i prosjektets makefile vil sensornoden bruke TSCH. Hvis noden bruker IPv6 og RPL, som er standard i Contiki-NG, vil det også sette opp 6TiSCH slik at det er en enkel samhandling mellom TSCH og IPv6. Schedule vil være 6TiSCH minimal og et TSCH nettverk blir opprettet. Nodene begynner å gjøre seg kjent gjennom enhanced beacon og setter opp et nettverk sammen. Etter dette steget vil nodene kunne sende ping til hverandre og vises i hverandres RPL-neighbour liste og routing table. Dette fungerte feilfritt for oss.

5.5 Selvskrevet kode

Kode ble skrevet to steder. En i Contiki-NG-miljøet skrevet i C, og i Python 3.7 med CoAPthon3 [12] biblioteket. Dokumentasjon var i begge tilfeller mangelfull. I tilfellet for begge fantes det relevante eksempler. I Contiki-NG var dokumentasjonen tungvint å finne frem til da man måtte lese kildekoden, dette gjelder kun plattformspesifikke funksjoner da sentrale funksjoner som f.eks timere er dokumentert. I tilfellet for CoAPthon fantes det utdatert dokumentasjon.

5.6 Border router

En border router knytter sammen sensornettverket med en datamaskin over en seriell tilkobling. Contiki-NG har klar border router kode som kan bygges og flashes på en Launchpad uten modifisering. For å koble Launchpad til en datamaskin har Contiki-NG inkludert kildekoden til verktøyet tunslip6. Tunslip6 ble bygget på en Raspberry Pi som også måtte konfigureres til å videresende pakke mellom nettverksgrensesnitt ettersom tunslip6 åpner et tunnelgrensesnitt når det kjører. I utgangspunktet trodde vi at rutingtabellen måtte modifiseres, men dette var ikke nødvendig. Får å kunne nå sensornettverket utenifra via Raspberry Pi med en laptop var det nødvendig å ha en rute for sensornettverket pekende på Raspberry Pi.

5.7 Signalrekkevidde

Vi testet mottaker rekkevidden i en enkel hello world implementasjon. Vi hadde border router satt opp i et grupperom i fabrikkgaten, en sensortag i gangen utenfor rommet, og vi plasserte den siste sensortagen i enden av gangen Ca 30 meter unna. Originalt var intensjonen å få den fjerntliggende sensortaggen til å bytte parent til noden i gangen og gå gjennom den for å nå border router. Resultatet ble at den fjerntliggende noden fortsatt gikk direkte til border router. Signalrekkevidden var mye lengre enn antatt, spesielt med tanke på veggene rundt grupperommet. I løpet av samme

dag observerte vi at en tredje node gikk gjennom noden i gangen for å nå border router selv om den selv var plassert på samme bord som border router og da var fysisk nærmere enn noden i gangen.

5.8 Grupperom

Vi plasserte sensortagger i 3 forskjellige grupperom, henholdsvis D229, D230 og D231, og målte lysstyrken i rommet. Hensikten med dette var å teste sensortaggene over tid i et miljø der vi kunne få reelle resultater. I dette eksperimentet måler sensorene hvor ofte et grupperom blir brukt i forhold til hvor lenge det er bestilt. Vi har målt at timeren for at lyset slås av på et grupperom er fire minutter, med mindre det er bevegelse i rommet.

Først ble hvilke rom, hvor testingen skulle foregå, valgt. Sensortagene fikk nye batterier og ble plassert oppå vegghengende lydabsorbenter, med sensorene pekende opp. Det ble også vurdert å legge dem oppå lysarmaturene som finnes i rommene, men det ble gått bort i fra for å unngå eventuell overopphetning av sensortagene. Hver sensortag tar en måling av lyset hver 120. sekund, samme periode brukes også av klientprogrammet. Hver spørring logges med IPv6 adresse, dato og klokkeslett, og lysstyrke i lux. Det blir også spurt om spenningen til batteriet hvert 10. minutt.

Testingen viser at lysnivået når lyset er av er under 1.00 lux, en sensor har målt 0,32-0,48 lux mens de to andre har målt 0,72-0,96 lux. Ved lys på har det vært målt 41,80-72,00 lux, men ser man bort fra de første fire målingene etter lyset ble slått på har det vært målt 70,00-72,00 lux. På en annen sensortag har det vært målt 77,83-86,88 lux og på den tredje 94,24-106,48 lux. Variasjonen mellom sensorene kan skyldes plassering i rommet, tilstand på lysrør og -armatur, samt fettmerker på sensordeksel og variasjoner i sensorene selv. Selv om variasjonen av lysstyrke målt mellom hver sensortag har variert en del, har det ikke vært tvil om at verdien målt har representert hvorvidt lyset i grupperommet har vært av eller på.

fd00::212:4b00:11a7:3206	2019-05-09 09:39:09	Light=0.88 lux
fd00::212:4b00:11a7:3206	2019-05-09 09:41:09	Light=0.88 lux
fd00::212:4b00:11a7:3206	2019-05-09 09:43:12	Light=95.52 lux
fd00::212:4b00:11a7:3206	2019-05-09 09:45:12	Light=102.00 lux

Utdrag fra logg

5.9 Batterilevetid

Det har blitt observert at batterilevetiden på programmer som printer ut med noen sekunders mellomrom, som hello world eksempelet i Contiki-NG tømmer batteriene til sensortagene i løpet av noen dager. Programmet som ble brukt i 5.8 bruker hovedsakelig energi når det mottar og svare på forespørsler, der en spørring hvert 2. minutt senket batterispenningen med 43-47 mV over en periode på 4,5 timer. Uten at det ble sendt spørringer til sensortagene så senket batterispenningen med 46 mV over en periode på 19,5 timer.

6 Diskusjon

Den originale oppgaveteksten har parkeringssensor som et forslag til praktisk anvendelse av sensorene. Det spesifikke bruksområdet er ikke viktig og det ble fort klart at nodene ikke egnet seg som parkeringssensor og bruksområdet ble derfor forandret til romsensor, som måler om lyset er av eller på i et rom og dermed om rommet er i bruk eller ikke.

Vi har holdt oss relativt godt til framdriftsplanen, men har vært nødt til å være ganske dynamisk da selve hensikten med oppgaven er å finne ut hvor bra Contiki-NG og 6TiSCH fungerer. Det har forekommet situasjoner der vi har undervurdert vanskelighetsgraden og vært nødt til å oppjustere framdriftsplanen, som når vi prøvde å finne ut hvordan vi skulle programmere våre egne programmer. Det har også forekommet situasjoner der vi har undervurdert vanskelighetsgraden og blitt ferdig mye fortere enn antatt. Begge disse situasjonene var forventet da Contiki-NG og 6TiSCH er under utvikling og har diverse problemer med Bugs, manglene dokumentasjon og manglende funksjoner.

7 Konklusjon

Både 6TiSCH og Contiki-NG er under utvikling og har derfor manglende funksjoner og Bugs. Derimot er de største problemene med Contiki-NG mangel på dokumentasjon og strømlinjeforming. Installasjonsguiden for Contiki-NG er god og svarer på de fleste problemer, men krever likevel at brukeren er erfaren med Linux eller Windows PowerShell, og docker.

For å flashe til nodene brukte vi SmartRF Flash, som er betalt programvare. Vi brukte studentversjoner vi fikk tilgang til gjennom HVL, men en bedrift vil måtte betale for tilgang. Gratisversjonen UniFlash mangler driver til USB tilkobling. Denne driveren er mulig å laste ned separat men den er vanskelig å finne og krever en del søking.

Contiki-NG har flere fult fungerende kode eksempler man kjapt og enkelt kan legge på nodene. Å skrive sin egen kode er langt vanskeligere fordi kode eksemplene er den eneste dokumentasjonen på den Contiki spesifikke koden, resten er vanlig C.

6TiSCH er allerede implementert i Contiki-NG og blir enkelt skrudd på ved å legge til TSCH modulen. Noden vil da bruke 6TiSCH minimal uten at man trenger å gjøre noe ekstra arbeid. I 6TiSCH minimal blir TSCH og RPL brukt umodifisert med bare basis innstillinger. 6LoWPAN bruker hele rammeverket slik det er, men det forventes at en mer minimal konfigurasjon vil bli utviklet.

Funksjonsmessig fungerer 6TiSCH veldig bra når man har fått opp en border router og et par noder. Hos nodene er signalrekkevidden er lengre enn forventet og batteritiden er god med mindre man har programmert nodene til å gjøre en handling konstant, slik som i det originale hello world eksempelet.

Rett ut av boksen vil vi ikke anbefale å bruke 6TiSCH og Contiki-NG, med mindre bruker har tid til å lære seg systemene og konfigurasjonene. Med riktig noder/microkontrollere, og rikelig med kunnskap til å programmere og konfigurere disse, vil 6TiSCH være brukbart. Det er usikkert om sikkerhetsmekanismene og motstandsdyktigheten mot pakketap har kommet langt nok til at 6TiSCH trygt kan brukes til kritiske oppgaver.

8 Videre arbeid

Vi har ikke fått anledning til å teste en del av funksjonene til CoAP. Vi har sendt Request fra PC, gjennom border router til nodene men det kan være mer interessant å la nodene styre selv når de skal sende sensorinformasjon. Et eksempel kan være når en verdi øker kraftig, eller går over en satt grenseverdi.

Vi har ikke fått anledning til å ta en lengre test av batteritid og tallfeste hvor lenge nodene kan kjøre med et gitt program. Det er spesielt interessant å sammenligne samme program med og uten “sleepy mode”, der noden går inn i en form for batterisparingsmodus når den ikke trenger å sende data.

Contiki-NG kommer med både ende til ende og hop til hop kryptering, men har foreløpig bare implementert pre-shared keys. [14] Andre metoder for key exchange vil sannsynligvis implementeres i framtiden.

Referanser

[1] Contiki-NG, «Contiki-NG: The OS for Next Generation IoT Devices,» [Internett]. Available: https://github.com/contiki-ng/contiki-ng. [Funnet 31 Mai 2019].

[2] Texas Instruments, «Uniflash Standalone Flash Tool for TI Microcontrollers,» Texas Instruments, [Internett]. Available: http://www.ti.com/tool/UNIFLASH. [Funnet 31 Januar 2019].

[3] Texas Instruments, «SmartRF Flash Programmer,» Texas Instruments, [Internett]. Available: http://www.ti.com/tool/FLASH-PROGRAMMER. [Funnet 31 Januar 2019].

[4] D. Dujovne, T. Watteyne, X. Vilajosana og P. Thubert, «6TiSCH: Deterministic IP-Enabled Industrial Internet (of Things),» *IEEE Communications Magazine,* pp. 36-41, Desember 2014.

[5] J. Hui og P. Thubert, «Request for Comments: 6282,» IETF, September 2011. [Internett]. Available: https://tools.ietf.org/html/rfc6282#section-3. [Funnet 24 Mai 2019].

[6] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.3.1,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.3.1. [Funnet 24 Mai 2019].

[7] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.3.2,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.3.2. [Funnet 24 Mai 2019].

[8] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.5.1,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.5.1. [Funnet 24 Mai 2019].

[9] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.5.2,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.5.2. [Funnet 24 Mai 2019].

[10] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.5.3,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.5.3. [Funnet 24 Mai 2019].

[11] P. Thubert, «An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4 section 4.5.4,» IETF, 1 Mars 2019. [Internett]. Available: https://tools.ietf.org/html/draft-ietf-6tisch-architecture-20#section-4.5.4. [Funnet 24 Mai 2019].

[12] Z. Shelby, K. Hartke og C. Bormann, «The Constrained Application Protocol (CoAP),» IETF, Juni 2014. [Internett]. Available: https://tools.ietf.org/html/rfc7252. [Funnet 30 Mai 2019].

[13] G. Tanganelli, C. Vallati og E. Mingozzi, «CoAPthon: Easy Development of CoAP-based IoT Applications with Python,» IEEE World Forum on Internet of Things (WF-IoT 2015), [Internett]. Available: https://github.com/Tanganelli/CoAPthon3. [Funnet 30 Mai 2019].

[14] Contiki-NG, «Documentation: Communication Security,» [Internett]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Communication-Security. [Funnet 31 Mai 2019].

[15] S. Bradner, «Request for Comments: 2026,» IETF, Oktober 1996. [Internett]. Available: https://tools.ietf.org/html/rfc2026#section-7. [Funnet 31 Januar 2019].

[16] Contiki-NG, «Documentation: RPL,» 14 Februar 2019. [Internett]. Available: https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-RPL. [Funnet 25 Mai 2019].

Appendiks A Forkortelser og ordforklaringer

6TiSCH	IPv6 over the TSCH mode of IEEE 802.15.4e
CoAP	Constrained Application Protocol
DODAG	Destination Oriented Directed Acyclic Graph
FDM	Frequency Division Multiplexing
Flashe	Programmering av enheter med flashminne, f.eks mikrokontrollere
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	IP versjon 4
IPv6	IP versjon 6
MAC	Medium Access Control
Make	Automatisk byggesystem for kompilering av programvare
Makefile	Fil som definerer hvordan make skal kompilere
MCU	Microcontroller unit
Lukket	Standard som ikke er offentlig tilgjengelig
RFC	Request For Comments
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
Shell	Brukergrensesnitt, tekstbasert i denne sammenheng
SSH	Secure Shell
SLIP	Serial Line IP
TDM	Time Divivsion Multiplexing
TI	Texas Instruments
TSCH	Time Slotted Channel Hopping
Åpen	Standard som er offentlig tilgjengelig [11]

Appendiks B Prosjektledelse og styring

B.1 Prosjektorganisasjon

Det er flat struktur i gruppen, det er ingen leder. Dette virker bra da gruppen kun består av to medlemmer. Arbeidet har blitt delt etter behov

B.2 Fremdriftsplan

Se vedlegg

B.3 Risikoliste

Programmering opp mot skytjenste viser seg å være mye mer komplisert og/eller tidkrevende enn forventet