NTNU
Kunnskap for en bedre verden

Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmålere ved bruk av trådløs M-Bus

Håkon Hardy Lier

Master i kybernetikk og robotikk
Innlevert: juni 2018
Hovedveileder: Geir Mathisen, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

NTNU

Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmålere ved bruk av trådløs M-Bus

Håkon Hardy lier

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

HOVEDOPPGAVE/MASTER THESIS

Kandidatens navn: **Håkon Hardy Lier**

Fag: **Teknisk kybernetikk/Engineering Cybernetics**

Oppgavens tittel (norsk): **Demonstrering av konsept for innsamling og sammenstilling av data fra flere vannmålere ved bruk av trådløs M-Bus**

Oppgavens tittel (engelsk): **Demonstration of Concept for Collecting and Compiling Data from Several Water Meters Using Wireless M-Bus**

Oppgavens tekst:

Vannmålere som måler vannvolum og som har fjernavlesning vil komme i alle hus i flere og flere kommuner. Disse installeres for å lettere og mere korrekt kunne avlese forbruket og fakturere kunden på en korrekt måte. Avlesing av disse målerne vil typisk være tilpasset til faktureringsfrekvensen mot kunde.

Dersom disse vannmålerne også utstyres med trykkmålere og avlesing av begge målere ble gjort nær sann tid, vil en få nær samtidige målinger fra alle hus. Da kan målingene fra flere hus også brukes til å si noe om tilstanden i vann-nettet som knytter husene sammen.

Vi ønsker å se på hvordan en ved hjelp av vannmålere med nær sann tid måledata fra flere hus kan si noe om tilstanden til vann-nettet utenfor huset.

Oppgaven består av følgende 3 punkter:

1. Foreta et litteratursøk rundt tilstandsovervåking av offentlige vannrørsnett.
2. Foreslå et system for innsamling av måledata relatert til vannrørsnett og sammenstilling av måledataene.
3. Så langt tiden tillater, implementer det foreslåtte systemet.

Oppgaven gitt: 8. Januar, 2018

Besvarelsen leveres: 11. Juni, 2018

Utført ved Institutt for Teknisk kybernetikk

Faglig veileder: Geir Mathisen

Trondheim, den 08.01.2018

Geir Mathisen

Faglærer

Forord

Denne oppgaven er basert på arbeidet i en prosjektoppgave skrevet av Håkon H. Lier i fjor. Tittelen på denne prosjektoppgaven er *Utvikling av et embedded system for avlesing av trådløs M-BUS*. Hovedoppgaven kan anses som en utvidelse av denne prosjektoppgaven.

I dette prosjektet har følgende utstyr og programvare blitt stilt til disposisjon;

Fire SLWSTK6220A utviklerkort fra Silicon Labs,

Dell Optiplex 9010 desktop datamaskin og tilhørende IO-enheter,

IAR Embedded Workbench for ARM version 8.20.1.14188,

Kamstrup trykkmåler.

Veileder Geir Mathisen har assistert med utledning av bakenforliggende teori i oppgaven.

Sammendrag

Flere typer måleinstrumenter kommer med et grensesnitt for trådløs avlesning, deriblant trykkmålere, gassmålere, eller strømmålere. For overføring av måledata fra disse sensorene er trådløs M-Bus blitt en mye brukt standard. Den trådløse M-Bus protokollen blir også i stor grad benyttet i denne oppgaven.

Oppgaven har gått ut på å sette opp et system bestående av flere simulerte trykkmålere og en sentral enhet, heretter omtalt som konsentrator. Hensikten til konsentratoren er å hente måledata fra trykkmålerne trådløst ved bruk av wM-Bus protokollen, og deretter sende dataene til en datamaskin for enkel behandling og videresending av dataene til en database i en skytjeneste. Både de simulerte trykkmålerne og konsentratoren i oppsettet er utviklerkort utstyrt med radio transceivers for trådløs kommunikasjon, og diverse IO-enheter.

Mye av arbeidet gikk ut på å håndtere uønsket oppførsel og bugs i software stacken for den trådløse M-Bus kommunikasjonen mellom målerne og konsentratoren. Denne software stacken var tilgjengelig som et tredjepartsbibliotek, og all funksjonalitet for kommunikasjon mellom målerne og konsentratoren var inkludert i stacken.

En annen viktig del av arbeidet var å håndtere sending av data fra konsentratoren til datamaskinen, og fra datamaskinen til skytjenesten. Datamaskinens oppgave var å foreta beregning av gjennomsnitt, minimum, og maksimum av dataene, før de ble sendt til en skytjeneste med et fast tidsintervall. Data ble sendt oftere fra konsentratoren til datamaskinen enn fra datamaskinen til skyen, så det var behov for å mellomlagre dataene på datamaskinen før videresendingen.

Den endelige løsningen fungerte bra med tanke på spesifikasjonen. Tidskravene ble overholdt og dataene som ble lagret på skyen var konsistente med de forventede verdiene fra de simulerte trykkmålerne.

Abstract

Several different gauges have an interface for wireless remote reading, including pressure meters, gas meters, and electricity meters. Wireless M-Bus has become the de facto standard for reading such devices. The wireless M-Bus protocol is also widely used in this project.

The task in this project has been to set up a system consisting of multiple simulated pressure meters and a central unit, hereby referred to as collector. The purpose of the collector is to remotely read the measurement data from the pressure meters by means of the wM-Bus protocol, and in turn send the data to a personal computer for simple processing and forwarding of the data to a database in a cloud service. Both the simulated pressure meters and the collector in the setup are development boards equipped with radio transceivers and various peripheral devices.

A substantial part of the work was dealing with bugs and unwanted behavior in the wireless M-Bus software stack. This software stack was available as a third-party library, and all functionality for the meter-collector communication was included in that stack.

Another important issue was handling the sending of data from the collector to the computer, and from the computer to the cloud. One of the responsibilities of the computer was to compute the average, minimum, and maximum of the data before they were passed on to the cloud with a fixed time interval. The data were sent more frequently from the collector to the computer than from the computer to the cloud, so caching the data on the computer before forwarding to the cloud was necessary.

The final solution worked well and was in compliance with the requirements in the specification. The timing of the transmissions followed the requirements and the data stored in the cloud were consistent with the expected values from the pressure meters.

Innhold

Forord.....4

Sammendrag.....5

Abstract.....6

Brukte forkortelser og begreper.....9

1. Innledning.....10
    - 1.1 Bakgrunn.....10
    - 1.2 Motivasjon.....11
    - 1.3 Begrensninger.....11
    - 1.4 Disposisjon av oppgaven.....11
2. Litteraturstudie.....13
    - 2.1 Overvåking av vanndistribusjonsnett.....13
    - 2.2 Oppsummering.....17
3. Teori.....18
    - 3.1 Behandling av måledata.....18
    - 3.2 Trådløs M-Bus.....21
4. Design av system.....24
    - 4.1 Funksjonell spesifikasjon.....24
    - 4.2 Overordnet design.....25
5. Implementasjon.....27
    - 5.1 Hardwarebeskrivelse.....27
    - 5.2 Beskrivelse av utviklermiljø og software.....31
    - 5.3 Biblioteker og ekstern kildekode.....31
    - 5.4 Bruk av utviklerverktøy.....34
    - 5.5 Modifikasjon av wM-Bus template-koden.....36
    - 5.6 Utvikling av brukergrensesnitt.....39
    - 5.7 Logging av data i skytjeneste.....41
    - 5.8 Plotting av data.....43
6. Resultater.....45

6.1 Initiell testing av wM-Bus kommunikasjon……..45
6.2 Kommunikasjon mellom collector og brukergrensesnit……46
6.3 Kommunikasjon med database……47
7. Diskusjon……56
7.1 Tolkning av resultater……56
7.2 Ikke-simulert trykkmåler……58
7.3 Svakheter……58
7.4 Erfaringer……59
8. Konklusjon……61
9. Videre arbeid……62
Referanseliste……64
Vedlegg……65

Brukte forkortelser og begreper

AWS – Amazon Web Services

BCD – Binary Coded Decimal

M-Bus – Meter-bus

wM-Bus – Wireless Meter-bus

PDA – Personal Digital Assistant

Broadcasting – Sending av informasjon med uspesifisert destinasjon

Andre bemerkninger; begrepene collector og konsentrator blir brukt om hverandre og refererer til det samme. Det samme gjelder meter og vannmåler/trykkmåler.

Kapittel 1 – Innledning

1.1 Bakgrunn

I et vannforsyningsnettverk er det et stort antall vannmålere fordelt utover husstander, og i inn og utløp av nettet. Disse vannmålerne har behov for jevnlig avlesing, både for å få informasjon om vannforbruk til kunder, men også for å generelt overvåke tilstanden til vanndistribusjonsnettet av sikkerhetsmessige årsaker. Rørene skal blant annet sikres mot sprekker og lekkasje, noe som krever at det opprettholdes et mer eller mindre konstant trykk i rørledningen.

Avlesing og inspeksjon av vannmålere har tradisjonelt sett blitt utført manuelt ved at vannverket sender dedikerte ansatte til hver vannmåler i nettet for å samle inn data. Denne løsningen er ugunstig på flere måter. Det er en veldig ineffektiv løsning med tanke på både bruk av tid og bruk av arbeidskraft. Denne løsningen er også utsatt for menneskelig svikt, ettersom avlesningen foregår manuelt.

Ved å implementere et system for automatisk innsamling av måledata, vil flere av ulempene med manuell avlesning elimineres. Innsamlingssystemet vil kunne gjøre målinger mye oftere og mer jevnlig, med større presisjon, og uten behov for store mengder med menneskelig arbeidskraft. Innsamlingssystemet vil også være mye mer effektivt med tanke på å detektere feiltilstander i forsyningsnettet, ettersom måledata er tilgjengelig i tilnærmet sanntid. Et slikt system kan ha en hierarkisk utforming, hvor målerne sender måledata til en sentral enhet, som i sin tur sender dataene til en database over internett, ved å for eksempel bruke et modem. Ved å lagre dataene eksternt på denne måten vil man kunne få tilgang til måledataene fra hvor som helst, forutsatt at man har internettforbindelse.

Formålet med dette prosjektet er å demonstrere konseptet for et slikt distribuert innsamlingssystem for vannmålinger. Hensikten er å undersøke gjennomførbarheten, i tillegg til funksjonaliteten av en slik løsning. Arbeidet kan brukes som et utgangspunkt for et utviklingsprosjekt av et faktisk system, noe som kan være nyttig for eksempelvis vannleverandører.

1.2 Motivasjon

Det er flere problemstillinger som kan være interessante i et utviklingsprosjekt for et datainnsamlingssystem. Oppgaven krever en overveielse av hvilken hardware og software som er egnet til de forskjellige oppgavene i systemet. En annen relevant problemstilling er utforming av systemarkitektur, og den spesifikke implementasjonen i det som er tilgjengelig av hardware og kildekode. Prosjektet involverer også en trådløs applikasjon, som medfører flere utfordringer med tanke på rekkevidde og andre parametere som har innvirkning på signalkvaliteten.

1.3 Begrensninger

Den største begrensningen i prosjektet var mangelen på faktiske trykkmålere. I utgangspunktet var planen å benytte en slik trykkmåler i oppsettet, noe som viste seg å være vanskeligere å gjennomføre en først antatt. Konsentratoren i oppsettet så ikke ut til å kunne motta wM-Bus telegrammer korrekt, selv om den kjente adressen og krypteringsnøkkelen til trykkmåleren. Det ble gjort flere forsøk på å prøve å få trykkmåleren til å kommunisere med konsentratoren, men ingenting førte frem.

1.4 Disposisjon av oppgaven

Litteraturstudie

Her står informasjon om lignende vannmålerløsninger som har blitt implementert tidligere av diverse bedrifter.

Teori

Her står relevant teori om hvordan wM-Bus protokoll-stacken fungerer på ulike lag, og om hvordan måledata fra trykk og strømningsmålere kan behandles.

Design av system

Her står den funksjonelle spesifikasjonen til vår løsning av oppgaven, i tillegg til hvordan systemet er utformet på et høyt nivå.

Implementasjon

Her står alle detaljene vedrørende implementasjonen av løsningen. Dette innebærer valg av hardware, software, bruk av utvikler verktøy og fremgangsmåte for implementasjonen.

Resultater

Denne delen inneholder en beskrivelse av prosessen med å teste ut funksjonaliteten i systemet, og resultatene av uttestingen.

Diskusjon

Her blir resultatene fra prosjektet analysert og tolket. Det blir også fokusert på svakheter ved gjennomføringen av prosjektet og erfaringer gjort i prosjektet.

Konklusjon

Videre arbeid

Her foreslås det hva man kan fokusere på dersom man ønsker å bruke dette prosjektet som et utgangspunkt for videre prosjekt arbeid, og hvilke utbedringer som kan gjøres.

Kapittel 2 – Litteraturstudie

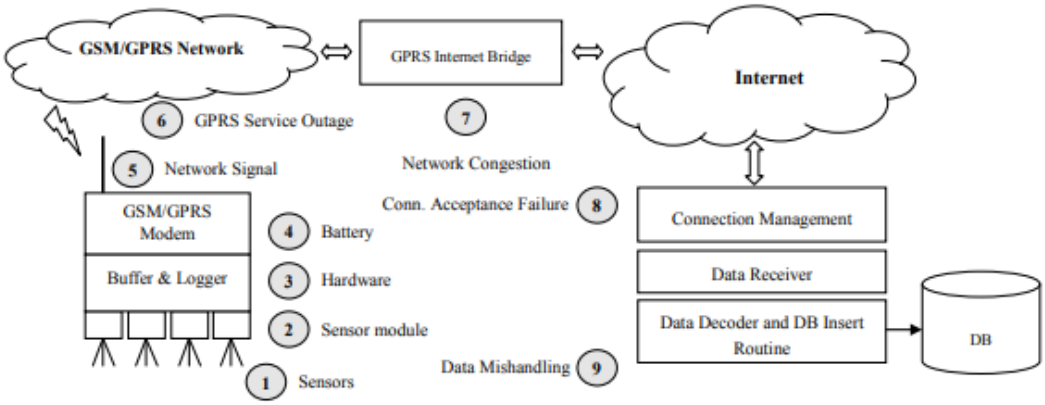
Følgende kapittel vil presentere et litteratursøk for hvordan innsamling og sammenstilling av måledata gjøres per dags dato. I kapittelet vil innholdet i relevante artikler bli oppsummert og diskutert.

2.1 Overvåking av vanndistribusjonsnett

Følgende informasjon er hentet fra;

D. D. Ediriweera, I. W. Marshall, 2010.

I (D. D. Ediriweera, I. W. Marshall, 2010) analyseres et system for online overvåking av måledata i form av trykkmålinger og strømningsmålinger fra et sensornettverk installert i et vanndistribusjonsnett. Arbeidet i undersøkelsen går i stor grad ut på å analysere ytelse og tap av data i forskjellige deler av nettverket, men inkluderer også design av systemarkitekturen i et sensornettverk for overvåking av vanndistribusjon. Figur 1 er hentet fra artikkelen og viser utformingen av sensornettverket.



Figur 1, systemarkitektur for overvåking av vannforsyning (D. D. Ediriweera, I. W. Marshall, 2010)

Sensornettverket i analysen består av flere komponenter, og bruker en internettforbindelse til å kommunisere med en database. Sensornettverket benytter over 520 sensornoder,

spredt utover et 50 km^2 landareal. Sensorene i nettverket er ikke direkte koblet til internett, men kommunikasjonen med internett går via flere grensesnitt for å til slutt få tilgang databasen. Hver sensornode bruker et GSM modem for kommunikasjon med et GSM/GPRS nettverk. Kommunikasjonen mellom sensorene og internett går derfor via GPRS kjernenettet.

Sensorene i nettverket foretar målinger hvert 15. minutt. Måledata fra sensorene blir mellomlagret i en logger, før de sendes videre til en sentral data collector. Loggeren sender dataene til data collectoren hvert 30. minutt. Trykkmålerne og strømningsmålerne opererer på forskjellig måte. Trykkmålerne sender en momentanverdi til loggeren, mens strømningsmålerne beregner et gjennomsnitt av målingene og sender dette gjennomsnittet videre. Sensornettverket implementerer med andre ord et hierarki av komponenter hvor data blir konsentrert, mellomlagret, behandlet, og videresendt.

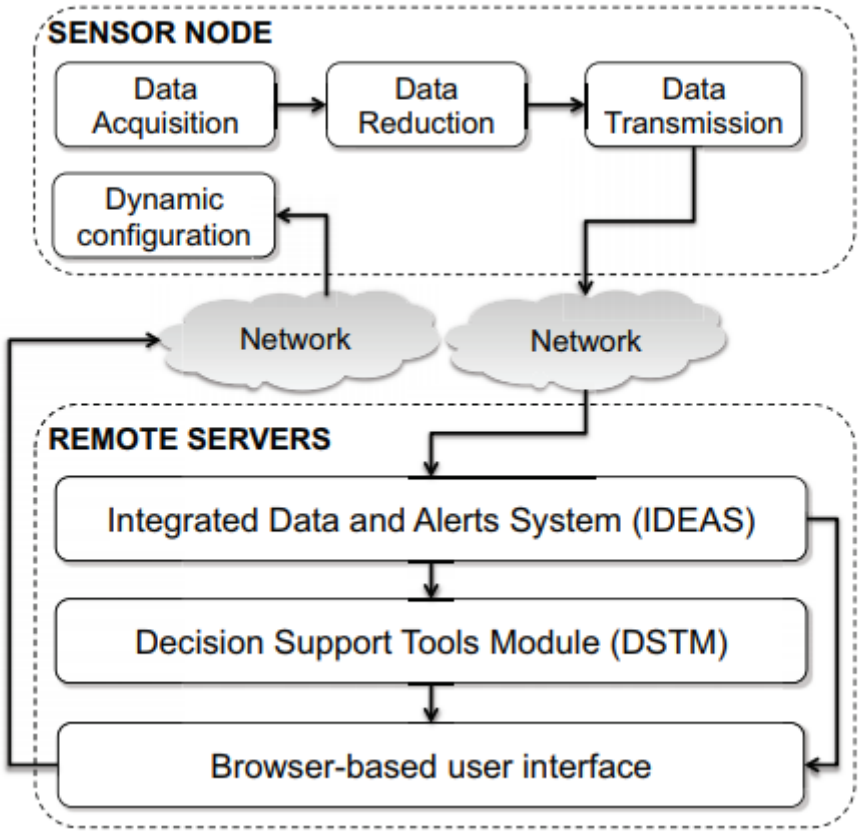
Følgende informasjon er hentet fra;

Michael Allen, Ami Preis, Mudasser Iqbal, Andrew J. Whittle, 2013.

Artikkelen (Michael Allen, Ami Preis, Mudasser Iqbal, Andrew J. Whittle, 2013) utforsker hvordan tilstanden i vanndistribusjonsnettet kan overvåkes i sanntid ved å bruke et system de kaller WaterWise.

Det diskuteres hvordan vanndistribusjonsnett bruker SCADA systemer til å hente data fra punkter hvor vann kommer inn i vanndistribusjonsnettet, istedenfor å analysere selve kjernen av vanndistribusjonsnettet. I artikkelen foreslås det en løsning for å utbedre denne begrensningen. Løsningen går ut på å hente måledata fra et sensornettverk som er installert i vannforsyningsnettet, og deretter analysere dataene i sanntid ved hjelp av WaterWise plattformen. Analysen går blant annet ut på deteksjon av lekkasjer og overvåking av vannkvalitet, og basert på denne analysen tilbyr WaterWise en form for beslutningstøtte for utbedring av feiltilstand i vannforsyningen. Figur 2 viser hvilke komponenter WaterWise

består av, og hvordan kommunikasjonen mellom dem foregår.



Figur 2, WaterWise komponenter (Michael Allen, Ami Preis, Mudasser Iqbal, Andrew J. Whittle, 2013)

I WaterWise blir analysen av måledataene håndtert av IDEAS. IDEAS har ansvaret for å behandle strømmer av måledata, i tillegg til å detektere feiltilstander i forsyningsnettet. Feiltilstander i nettet vil generere varslinger som sendes til abonnenter over SMS eller epost. Relevant sensor data blir sendt videre til DSTM, hvor dataene brukes til modellering av tilstanden i forsyningsnettet. Basert på denne tilstanden tilbys det flere verktøy for assistanse av operasjon av forsyningsnettet.

Følgende informasjon er hentet fra;

Mompoloki Pule, Abid Yahya, Joseph Chuma, 2018.

Arbeidet i (Mompoloki Pule, Abid Yahya, Joseph Chuma, 2018) går ut på å undersøke trådløse sensornettverk-løsninger for overvåking av vannkvaliteten i vannforsyningen.

I rapporten diskuteres flere teknologier for overføring av data i trådløse sensornettverk. Noe av det som tas opp er ulike kommunikasjonsstandarder for utveksling av sensor data. ZigBee og Bluetooth low energy er eksempler på slike kommunikasjonsstandarder som nevnes i rapporten. Ifølge rapporten er det flere parametere som bør tas i betraktning når en kommunikasjonsstandard skal velges for en trådløs sensornettverksapplikasjon, deriblant frekvensbånd, rekkevidde, og overføringsrate.

Noen argumenter for bruken av trådløse sensornettverk som tas opp er at de har lav kostnad, kan operere mer eller mindre uten menneskelige inngrep, og gi tilgang til måledata i sanntid eksternt.

Følgende informasjon er basert på litteratursøket i (Håkon H. Lier, 2017, *Utvikling av et embedded system for avlesing av trådløs M-BUS*).

Når det kommer til trådløs avlesing av vannmålere finnes det flere kommersielle løsninger på markedet. Eksempler på eksisterende leverandører av slike løsninger er WaterWare, Apator, Kamstrup, og AxFlow.

De forskjellige løsningene har mye til felles. For eksempel benytter alle løsningene bortsett fra AxFlow wM-Bus for kommunikasjon av måledata. Noen av løsningene bruker også en form for extender for å øke rekkevidden på wM-Bus overføringene. Mulighet for å overvåke målingene fra en PDA eller datamaskin ser også ut til å være utbredt blant de forskjellige systemene.

For mer informasjon angående de forskjellige leverandørene henvises det til (Håkon H. Lier, 2017).

2.2 Oppsummering

I dette kapittelet har ulike state-of-the-art teknologier innen overvåking av vannforsyning blitt presentert og diskutert. Det er flere likheter og forskjeller mellom de ulike løsningene som har blitt diskutert. Trådløse sensornettverk ser ut til å være en mye brukt teknologi for innhenting av måledata i sanntid. For lesing av målere ser wM-Bus ut til å være en dominerende standard, men protokoller som Bluetooth low energy og ZigBee virker som gode alternativer. Måten dataene blir behandlet og lagret varierer litt fra løsning til løsning, men ekstern lagring i database ser ut til å være en måte å gjøre det på. Overføring av måledata over GPRS til en ekstern server er en løsning som har blitt tatt opp. Noe av behandlingen av måledataene blir ofte gjort før de når sluttdestinasjonen. Dette er ofte på grunn av et behov for å synkronisere data fra flere målere.

Kapittel 3 – Teori

3.1 Behandling av måledata

Følgende teori har blitt utledet av Håkon H. Lier i samarbeid med veileder Geir Mathisen.

I et måleroppsett bestående av trykkmålere og strømningsmålere fordelt over flere husstander er det flere betraktninger som trengs å gjøres med tanke på hvordan måledataene brukes og tolkes.

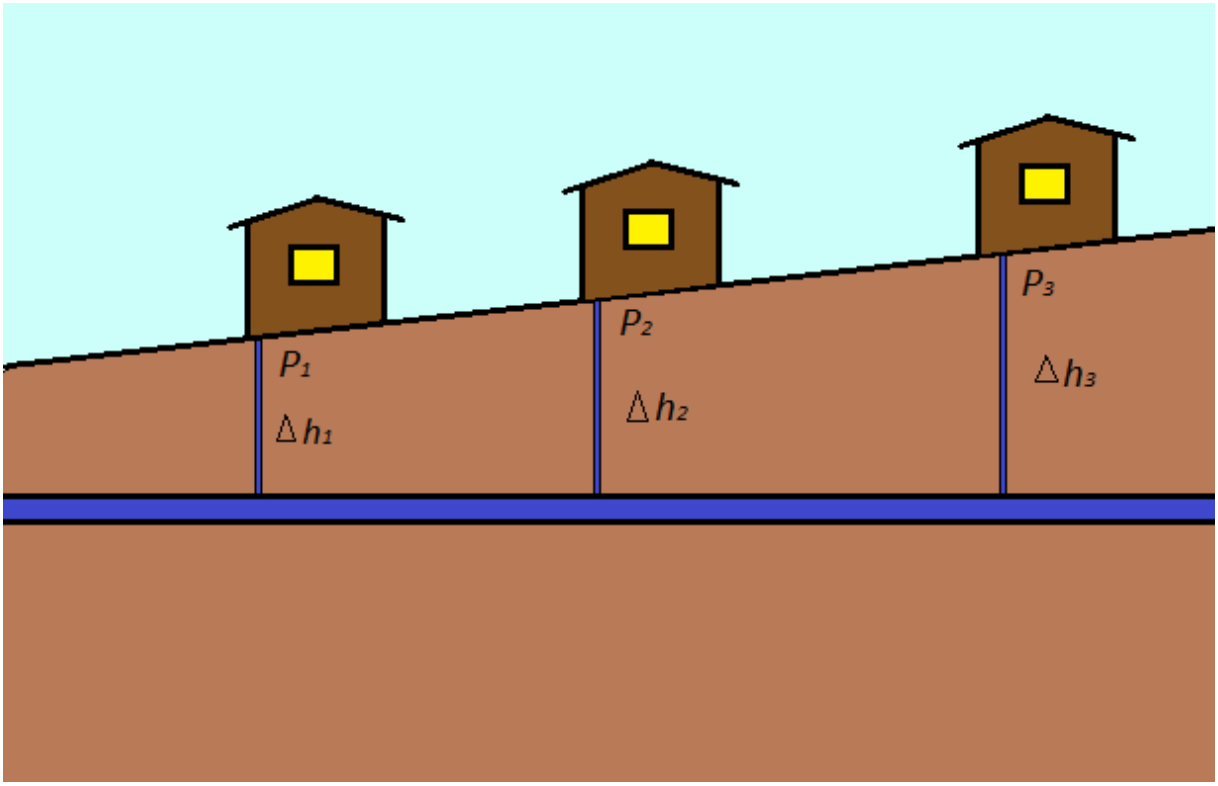
En av de viktigste grunnene til å foreta trykkmålinger er for å forsikre at det absolutte trykket holder seg innenfor en viss margin i alle deler av vannforsyningsnettet. Dersom trykket blir for høyt vil det kunne lekke ut store mengder vann og i verste fall vil rørene kunne sprekke. Hvis trykket blir for lavt vil det potensielt lekke inn i rørene, noe som kan føre til forurensing av vannet i vannforsyningsnettet.

Det vil kunne være flere husstander koblet til samme rørledning i vannforsyningsnettet. Fordi de er koblet til samme rørledning vil det være en korrelasjon mellom vanntrykkene til de respektive husstandene. Vanntrykket i de forskjellige boligene vil ikke være det samme, ettersom trykket også vil avhenge av høydeforskjeller mellom vanninntaket til boligene, i tillegg til andre trykkfall i rørledningen som følge av for eksempel friksjon. Trykkendringen som følge av høydeforskjell kan beregnes med formelen

$$\Delta P = \rho g \Delta h$$

, hvor ρ er tettheten til vann, g er tyngdeakselerasjon ved jordoverflaten, og Δh er høydedifferensialet. Figur 3 illustrerer hvordan høydeforskjeller har innvirkning på trykk i

vannforsyningsnettet.



Figur 3, Høydeforskjeller og trykk

Ved å ta hensyn til høydeforskjeller og andre kilder til trykkendringer vil det være mulig å beregne et estimat for hva trykket bør være forskjellige steder i vannforsyningsnettet. Ved å sammenligne dette estimatet med det målte trykket i vanninntaket i de respektive boligene kan man finne ut om det faktiske trykket blir påvirket av en annen uforutsett kilde, som for eksempel en lekkasje eller en blokade i rørledningen.

For at en slik analyse skal være gyldig er det viktig at vannet står mest mulig stille mens man gjør målingen av trykket hos forbrukeren. Det vil si at dersom det blir tappet vann til boligen på samme tid som det foretas en trykkmåling så vil man ikke kunne bruke denne målingen. Grunnen til dette er at når man forbruker vann så vil hastigheten til vannet øke, og dermed også trykket. Dette er tydelig utfra ligningen

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = konstant$$

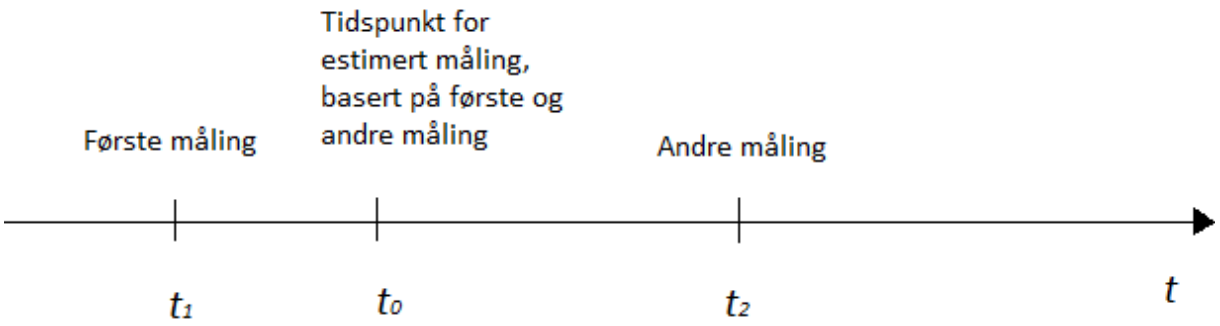
, også kjent som Bernoulli's ligning. Hvis man i tillegg til å foreta en trykkmåling også foretar en strømningsmåling vil man kunne avgjøre når vannet står i ro og når det er i bevegelse. Dermed kan man basert på denne strømningsmålingen avgjøre hvorvidt en trykkmåling er

gyldig. En annen grunn til å gjøre strømningsmålinger i hver husstand er at man ved å summere strømningen til hver husstand kan undersøke hvorvidt summen er lik det som kommer inn av vann fra reservoaret til nettet. Dersom summen er mindre vil det tyde på lekkasje.

Når man skal samle inn måledata fra flere målere i et distrikt med formål om å beregne trykkfall på måten nevnt ovenfor, er det viktig å ta hensyn til synkroniseringen mellom de forskjellige målerne. Dersom vannmålerne foretar målinger på forskjellige tidspunkter, og trykket i rørledningen endrer seg betydelig mellom hver måling, vil man ikke kunne kombinere målingene til å beregne en øyeblikkverdi av trykkfallet i rørledningen. Hvis målingene er relativt nærme hverandre i tid vil ikke feilen bli så stor dersom man bruker de samme målingene ukorrigert for tidsforskjeller, men ettersom tidsforskjellene øker vil det være ønskelig å kompensere for tidsforskjellen. En måte å gjøre dette på er å benytte gjennomsnittet mellom to målinger foretatt av en måler, og kombinere dette gjennomsnittet med en måling gjort av en annen måler i samme tidsrom. En annen potensielt bedre løsning vil være å beregne trykkfallet på et tidspunkt uavhengig av alle målerne, og estimere trykket ved hver måler med følgende formel

$$P = (1 - a)P_1 + aP_2$$

, hvor a er fraksjonen $\frac{t_0 - t_1}{t_2 - t_1}$, t_0 er et «virtuelt» måletidspunkt som er felles for alle målere, t_1 er tidspunktet for første måling, t_2 er tidspunktet for andre måling, P_1 er det første målte trykket, og P_2 er det andre målte trykket. Figur 4 viser en grafisk representasjon av sammenhengen mellom de forskjellige målingene i tid.



Figur 4, forskjellige måletidspunkter

3.2 Trådløs M-Bus

Følgende teori er hentet fra;

Olivier Hersent, David Boswarthick and Omar Elloumi (2012), *The Internet of Things: Key Applications and Protocols, First Edition*, John Wiley & Sons, Ltd.

introduction-to-wireless-mbus.pdf, Silicon Labs.

Teoridelen om wM-Bus er i stor grad basert på teoridelen i (Håkon H. Lier, 2017).

Trådløs M-Bus er en kommunikasjons protokoll for overføring av måledata. Den benytter seg av tre forskjellige protokoll-lag for å oppnå dette, fysisk lag, link-lag, og applikasjonslag.

Fysisk lag

Wmbus protokollen opererer med flere forskjellige overføringsmodi (transmission modes); S-Mode (stationary mode), T-Mode (frequent transmit mode), R-Mode (frequent receive mode), og N-Mode (narrowband mode), C-Mode (compact mode), F-Mode (frequent receive and transmit mode). For hver av disse finnes det både uni og bidirectional transfer mode. Hvilken overføringsmodus som brukes avgjør hvilken retning kommunikasjonen går, hvor ofte telegrammer blir sendt, hvilken frekvens som brukes i overføringsignalet etc. Figur 5 viser en oversikt over forskjellige overføringsmodi for wM-Bus.

Mode	Frequency(MHz)	Notes
S (Stationary)	868	Meters send data few times a day
T (Frequent Transmit)	868	Meters send data several times a day
C (Compact)	868	Higher data rate version of mode T
N (Narrowband)	169	Long range, narrow band system
R (Frequent Receive)	868	Collector reads multiple meters on different frequency channels
F (Frequent Tx and Rx)	433	Frequent bi-directional communication

Figur 5, overføringsmodi (introduction-to-wireless-mbus.pdf, Silicon Labs)

I denne oppgaven blir overføringsmodusen C-mode benyttet. C-mode opererer med en overføringsfrekvens på 868 MHz, og enheter konfigurert i C-mode er beregnet for å sende med en hyppighet på flere ganger daglig. Til sammenligning sender S-mode kun noen få ganger daglig. C-mode skiller seg fra T-mode ved at overføringsraten er høyere. C-mode er delt inn i to under kategorier, C1 og C2, som er henholdsvis uni og bidirectional. I ett system som benytter C1 går kommunikasjonen utelukkende fra meter til collector, mens for C2 kan kommunikasjonen gå begge veier. Tilsvarende har man to slike under kategorier for R-mode, S-mode, og T-mode. Det er også mulig for meter og collector å være konfigurert forskjellig med tanke på kommunikasjonsretningen. For eksempel, i denne oppgaven blir C1 benyttet for meter og C2 for collector.

Link-lag

Datalink-laget definerer oppbygningen av wM-Bus telegrammer, i tillegg til feildeteksjon og korrigering, og fysisk adressering. Detaljene i datalinklaget er av liten relevans for dette prosjektet, og vil derfor bli utelatt her.

Applikasjonslag

Dataene i ett wmbus telegram er organisert i såkalte data records. En data record inneholder data til en målt størrelse. Hvert data record starter med en data information block (DIB) som beskriver diverse informasjon som lengde og type til dataene som følger. Deretter kommer en blokk kalt value information block (VIB). De første bitene i VIB (antallet kan variere) indikerer enheten til måledataene (for eksempel m^3) og de siste bitene indikerer multiplikatoren til enheten (for eksempel dm^3 eller cm^3). Deretter kommer en blokk med selve dataene i binær kodet desimal form. Etter denne data blokken kommer det eventuelt en ny DIB dersom det er flere data records i telegrammet. Figur 6 viser oppbygningen av et data record.

<table>
<tr><td>DIF</td><td>DIFE</td><td>VIF</td><td>VIFE</td><td>Data</td></tr>
<tr><td>1 Byte</td><td>0-10 (1 Byte each)</td><td>1 Byte</td><td>0-10 (1 Byte each)</td><td>0-N Byte</td></tr>
<tr><td colspan="2">Data Information Block DIB</td><td colspan="2">Value Information Block VIB</td><td rowspan="2"></td></tr>
<tr><td colspan="4">Data Record Header DRH</td></tr>
</table>

Figur 6, Data record (http://www.m-bus.com/files/w4b21021.pdf, Prof.Dr.H.Ziegler)

Kapittel 4 – Design av system

Noen av kravene i den funksjonelle spesifikasjonen ble endret underveis i prosjektet på grunn av ny informasjon om hva som var nødvendig og hva som ikke var nødvendig med tanke på funksjonalitet. Det overordnede designet prøver i stor grad å følge det som er etterspurt i spesifikasjonen. Følgende kapittel vil presentere spesifikasjonen slik den foreligger per dags dato, i tillegg til systemets overordnede design.

4.1 Funksjonell spesifikasjon

1. Systemet skal bestå av flere trykkmålere, en konsentrator, en form for ekstern lagringsteknologi, eksempelvis en skytjeneste, og en kommunikasjonslink eller grensesnitt mellom konsentratoren og lagringsmediet.

2. Konsentratoren skal kunne samle inn data trådløst fra trykkmålerne, ekstrahere dataene, og deretter sende dem videre til et grensesnitt for ytterligere prosessering. Grensesnittet kan for eksempel være en PDA eller datamaskin, eller et enkelt modem.

3. Måledataene skal kunne bestå av strømningsmålinger og trykkmålinger, og sendes med et intervall på 10 sekunder. Konsentratoren skal kunne motta data med dette tidsintervallet. Konsentratoren skal også kunne sende dataene videre med samme hyppighet.

4. Målerne og konsentratoren skal være i stand til å opprettholde kommunikasjon over store nok distanser til at konsentratoren kan være plassert utenfor eiendommen til forbrukeren hvor målerne er installert. Ideelt sett er det ønskelig å kunne hente data fra så mange husstander som mulig uten bruk av en extender.

5. Vannmålerne skal kommunisere med konsentratoren ved hjelp av trådløs M-Bus. Konsentratoren skal være konfigurert for overføringsmodusen C2, og vannmålerne skal være konfigurert for C1.

6. Vannmålerne skal ikke trenge å ha noen informasjon om konsentratoren de sender til, inkludert konsentratorens adresse. Dette innebærer at målerne vil trenge å kunne

broadcaste telegrammer, og konsentratoren trenger å være i stand til å motta slike telegrammer.

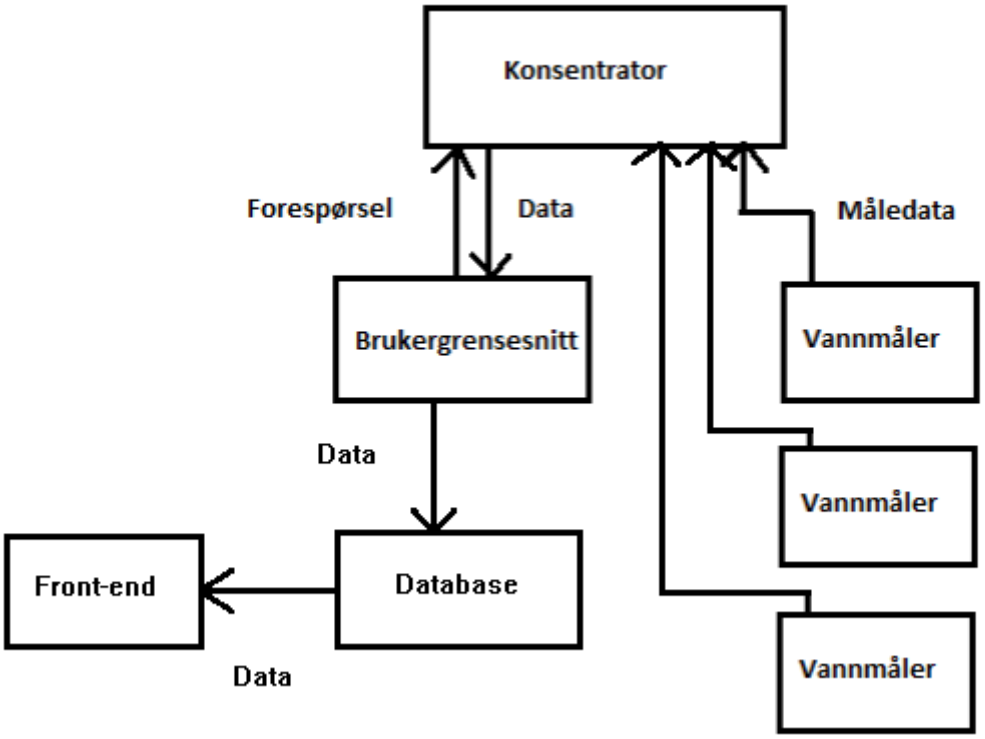
7. Konsentratoren skal ha et interface for tilkobling til et brukergrensesnitt. Dette kan for eksempel være et USB-interface. Brukergrensesnittet skal kunne overvåke data som blir mottatt i konsentratoren i sanntid og gjøre nødvendig konfigurasjon.

8. Konsentratoren trenger også en et grensesnitt for å kunne overføre data til databasen. Data skal kunne sendes til skyen med et 90 sekunders intervall, og det er essensielt at forbindelsen mellom konsentratoren og databasen har stor nok kapasitet til å håndtere at data fra tre forskjellige målere sendes med 90 sekunders intervall. Dataene som sendes til skyen skal inneholde gjennomsnittet, minimum, og maksimum av de siste målte verdiene, målerens id, og timestamp.

9. Det skal være mulig å hente ut de lagrede måledataene fra databasen i form av grafer som viser sammenhengen mellom tid, måler, og målt størrelse, for eksempel trykk.

4.2 Overordnet design

Figur 7 viser et blokkdiagram av systemets tiltenkte utforming på et høyt nivå. Diagrammet viser komponentene i systemet og informasjonsflyten mellom dem.



Figur 7, overordnet design

Det ble bestemt at kommunikasjonen med databasen skulle gå gjennom samme enhet som brukes til overvåking og konfigurering av konsentratoren, altså brukergrensesnittet. Dette reduserte antallet nødvendige komponenter i systemet. Fra brukergrensesnittet sendes forespørsler om å få tilsendt data eller gjøre konfigurasjon. Ved forespørsel om data vil dataene sendes fra konsentratoren så fort de er tilgjengelige og vises i et terminalvindu i brukergrensesnittet, som betyr at overvåkingen skjer i sanntid. Dataene sendes også videre til en database for lagring. I databasen lagres også informasjon om tid og hvilken måler dataene samsvarer med. Måledataene kan hentes fra databasen i en front-end applikasjon hvor dataene vil bli presentert grafisk.

Kapittel 5 – Implementasjon

Etter at systemarkitekturen hadde blitt bestemt på et overordnet nivå var neste del av oppgaven å realisere systemet. Det ble implementert en prototype for å demonstrere konseptet, og er på ingen måte ideell med tanke på ytelse og brukervennlighet. Det ble gjort en omstendelig prosess med å velge hvilke hardware-komponenter som skulle benyttes til de forskjellige oppgavene i systemet, i tillegg til hvilken software og hvilke utviklermiljø som skulle brukes. Følgende kapittel vil presentere implementasjonsdetaljene i realiseringen av systemet, og en begrunnelse for hvilke valg som ble gjort i den forbindelse.

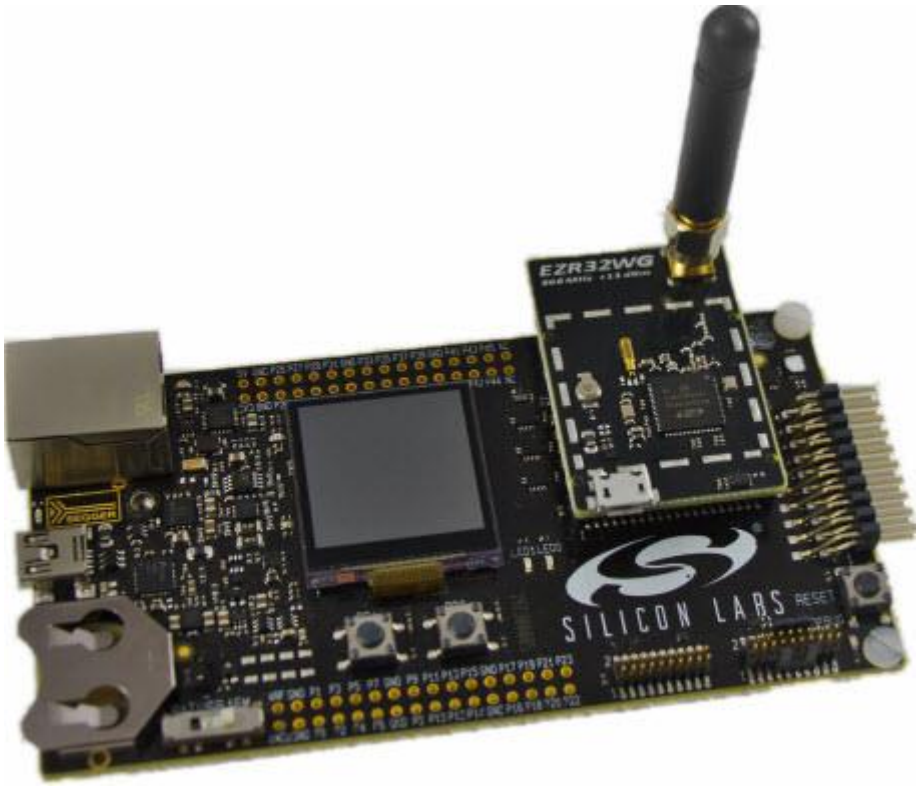
5.1 Hardwarebeskrivelse

Konsentrator

Det ble bestemt at konsentratoren i systemet skulle realiseres med et SLWSTK6220A utviklerkort fra Silicon Labs (figur 8), i likhet med hva som ble gjort i (Håkon H. Lier, 2017). Kortet er utstyrt med radio transceivers, noe som er essensielt for den trådløse M-Bus kommunikasjonen. I tillegg har kortet andre peripheral devices som er nyttige for konsentratorens oppgaver, deriblant en USB 2.0 interface for kommunikasjon med brukergrensesnittet, og LEDs som er nyttige for debugging og statusinformasjon på kortet. Kortet har en USB-konnektor for J-Link flashing/debugging og strømforsyning. Kortet kan også poweres av et batteri dersom det er nødvendig å koble kortet fra USB-konnektoren under kjøring.

Trykkmåler

Av grunner som vil bli utdypet i avsnitt 7.2 ble det bestemt at trykkmålerne i oppsettet skulle simuleres. Trykkmålerne trenger en måte å kommunisere med konsentratoren trådløst over M-Bus, og derfor er det hensiktsmessig å bruke SLWSTK6220A kort til dette formålet også.



Figur 8, SLWSTK6220A utviklerkort

Brukergrensesnitt

En datamaskin var et naturlig valg av brukergrensesnitt, ettersom det var behov for et USB-interface for kommunikasjon med konsentratoren, en form for display, og en internettilkobling. Datamaskinen var en desktop computer av typen Dell Optiplex 9010. Valg av datamaskin-modell var ikke av betydning for funksjonaliteten.

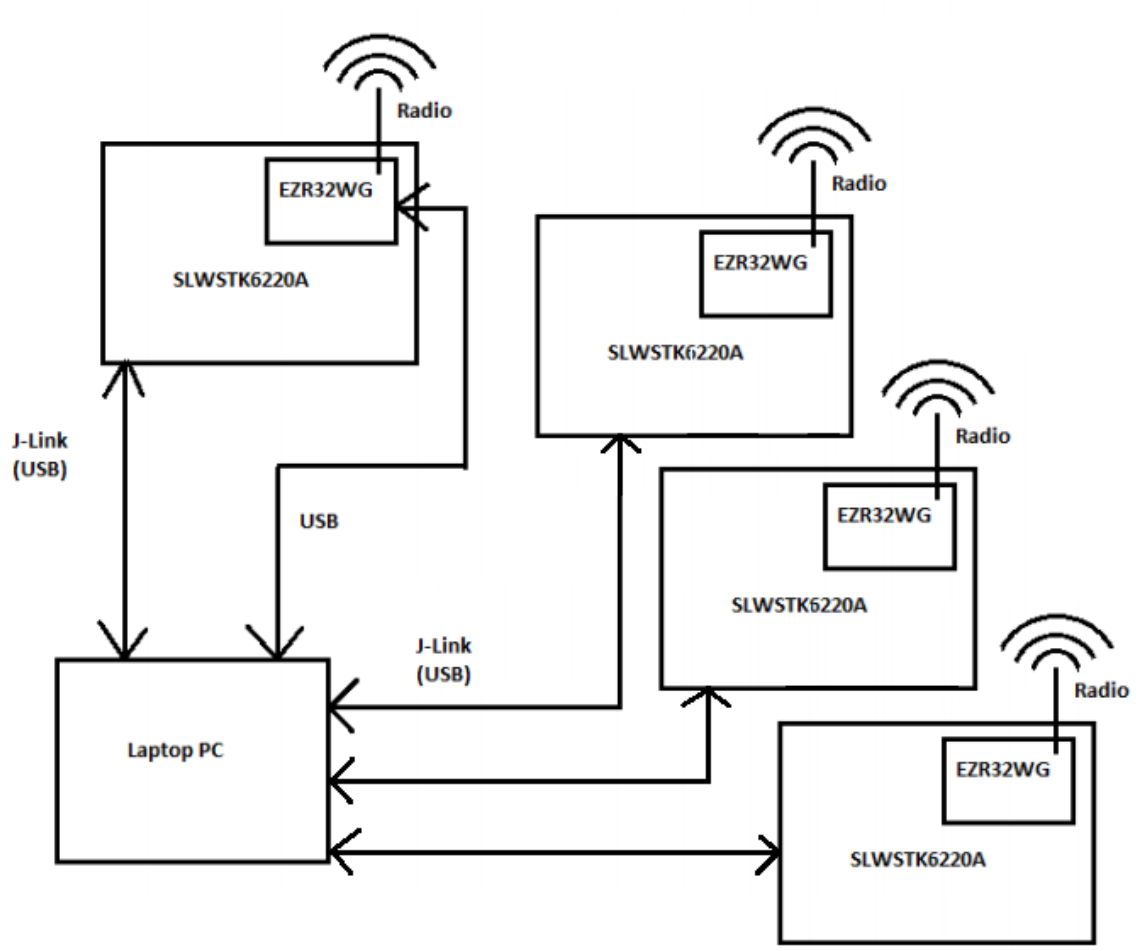
Database

Databasen var en ekstern tjeneste og hardware-spesifikke implementasjonsdetaljer var derfor ikke en del av analysen i oppgaven.

Front-end

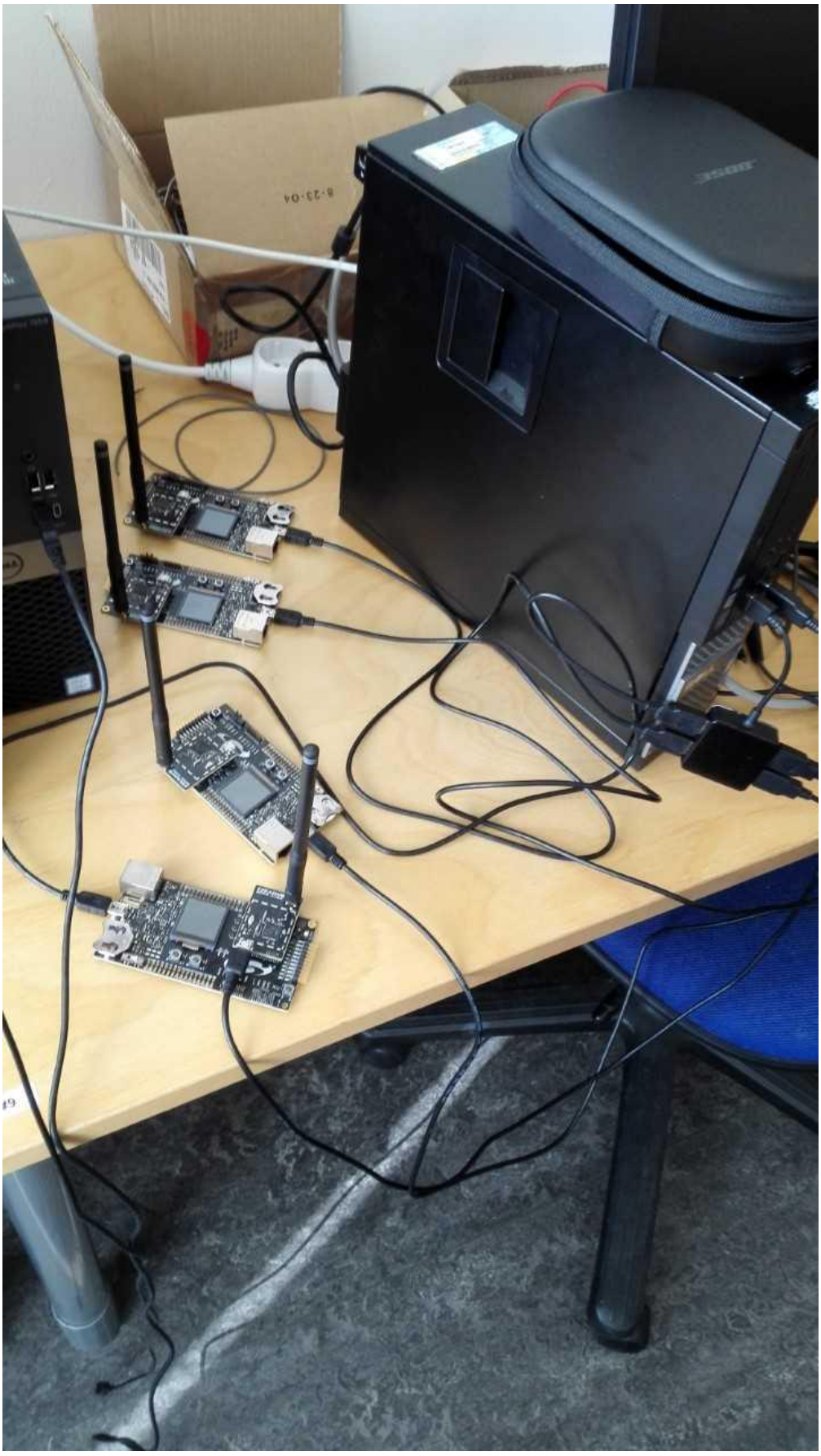
Front-enden i systemet skal i utgangspunktet kunne kjøre på en hvilken som helst plattform såfremt den har en internettforbindelse og kan kjøre Python 3.6.

Figur 9 viser hvordan utstyret var satt opp i prosjektet.



Figur 9, hardwareoppsett

Figur 10 er et fotografi av det faktiske oppsettet.



Figur 10, bilde av oppsett

5.2 Beskrivelse av utviklermiljø og software

Utviklingen av firmware som kjøres på SLWSTK6220A kortene gjøres på en Asus F550L laptop PC som kjører 32-bit Windows 7. Kompilering, linking, og assembling av kildekode for utviklerkortene gjøres i et IDE kalt IAR Embedded Workbench, som er en del av IAR toolchain for ARM. Grunnen til at IAR blir brukt, istedenfor for eksempel GCC for ARM, er på grunn av kompatibilitet og vil bli ytterligere forklart i neste avsnitt.

Brukergrensesnitt-applikasjonen er en C++ desktop-applikasjon som kjøres på en Windows maskin. For dette formålet er det flere kompilatorer og IDE'er som kan brukes, men i prosjektet ble MinGW brukt, i kombinasjon med en teksteditor og makefile. MinGW er et GCC lignende verktøy for building av C/C++ applikasjoner for Windows. MinGW ble primært valgt på grunn av personlige preferanser.

For utvikling av grensesnittet for kommunikasjon med databasen brukes Python 3.6, sammen med en teksteditor. Grunnen til at Python 3.6 blir brukt vil bli tydeligere senere i rapporten når valg av databaseteknologi blir diskutert.

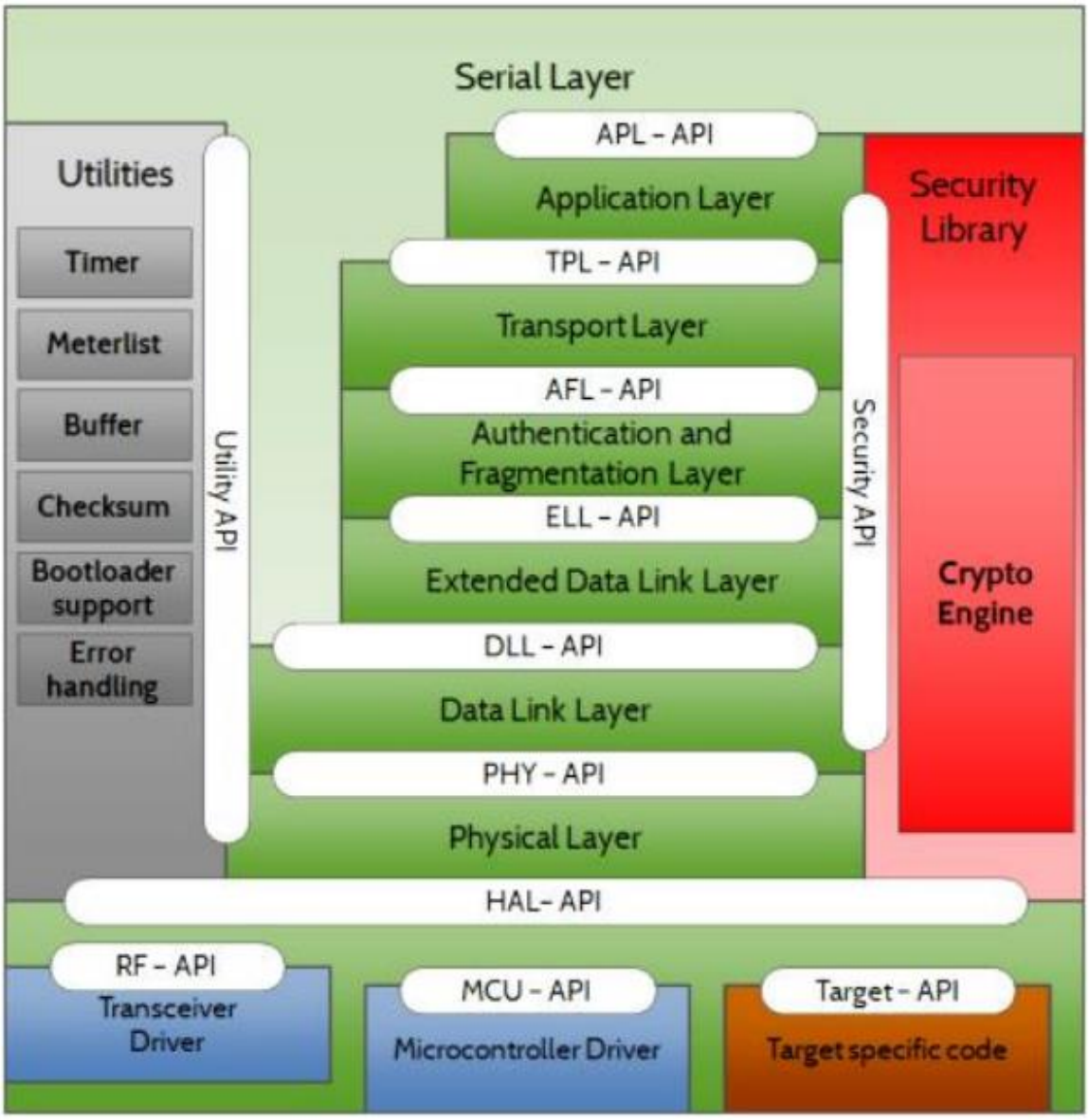
5.3 Biblioteker og ekstern kildekode

I flere deler av prosjektet ble det brukt biblioteker og kildekode utviklet av en tredjepart, ellers hadde omfanget av oppgaven blitt for stort. Alt av ekstern kildekode og biblioteker som ble brukt i prosjektet vil bli diskutert under.

WM-Bus stack

Bibliotekene som ble brukt for å bygge wM-Bus applikasjonen på SLWSTK6220A kortene var tilgjengelige som en kombinasjon av kildekode og binærfiler. Bibliotekene var opprinnelig utviklet av Stackforce, og var en del av en software-pakke som ble lastet ned fra Silicon Labs sine nettsider, på *https://www.silabs.com/products/development-tools/software/wireless-m-bus* (lastet ned september, 2017).

Figur 11 (QuickStartGuide_1.0.6.pdf, Stackforce) viser Stackforce sin implementasjon av wM-Bus protokoll stacken.

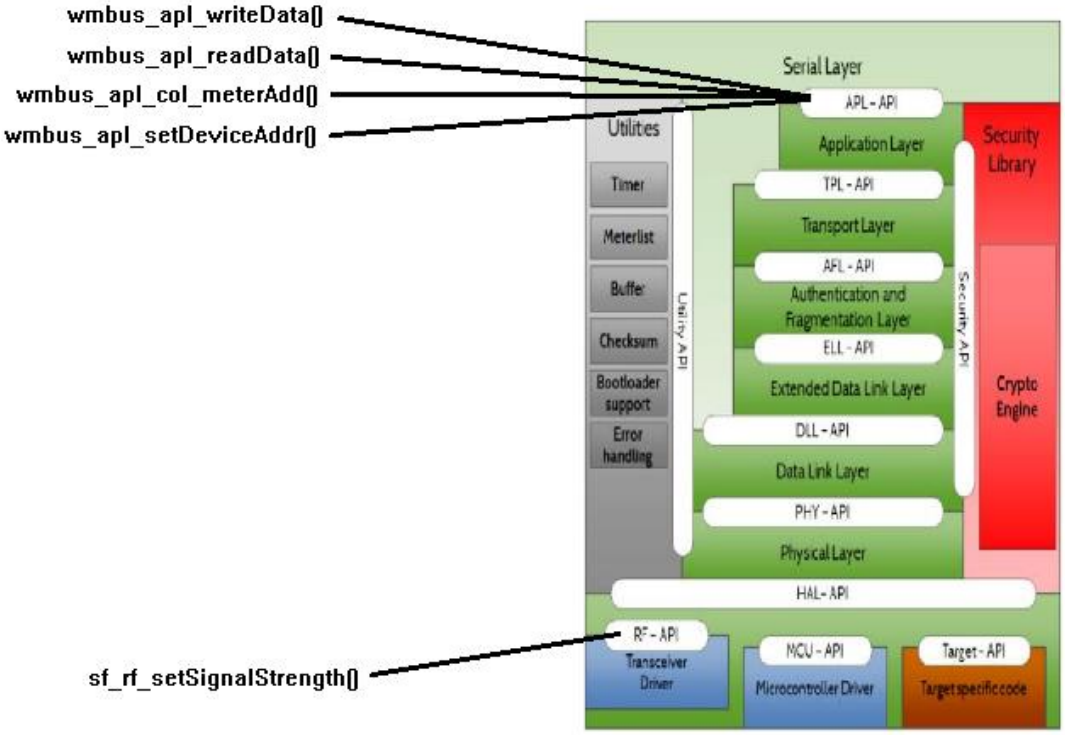


Figur 11, wM-Bus stack (QuickStartGuide_1.0.6.pdf, Stackforce)

Denne utgaven av wM-Bus protokoll stacken inkluderer tre ekstra protokoll lag, i tillegg til datalink-laget og applikasjonslaget. Et av disse lagene er et transportlag som Stackforce selv har definert. I dette laget implementeres blant annet kryptering av telegrammer.

Stackforce sitt bibliotek kommer med flere API'er for ulike lag i wM-Bus stacken. APL-API'et (applikasjonslaget), TPL-API'et (transportlaget), DLL-API'et (datalink-laget) er eksempler på

API'er som henvender seg til hvert sitt lag i stacken. Det finnes også et eget lag for hardware-interaksjon, kalt HAL-API'et (hardwareabstraksjonslaget). Dette API'et består av flere underordnede API'er, deriblant RF-API'et (radio frequency). I prosjektet ble kun APL API'et og HAL API'et brukt av praktiske grunner. Figur 12 viser noen essensielle kall som ble gjort i kildekoden og hvor i stacken de ble gjort.



Figur 12, kall til wM-Bus stacken

En del av wM-Bus biblioteket består av ferdig kompilerte binærfiler. For å kompilere en fungerende wM-Bus applikasjon kreves det to slike filer, en for stack kalt *libstack.a* og en for RF kalt *librf.a*. Det finnes flere utgaver av disse filene. Hvilke filer som skal velges er avhengig av flere kriterier. Disse inkluderer overføringsmodus, API (APL eller TPL), prosessorkjerne,

radio, og device-konfigurasjon. I prosjektet ble det bygget en applikasjon for hver device-konfigurasjon, altså en for meter og en for collector. Som antydet i spesifikasjonen skal meter benytte C1 som overføringsmodus og collector skal benytte C2. API'et som ble brukt var som tidligere nevnt APL. Prosessorkjernen på EZR32WG kortet er Cortex M4, som kan leses fra brukermanualen til EZR32WG kortet, og radio transceiveren er Si4460 som kan leses fra følgende dokument, https://www.silabs.com/documents/public/application-notes/AN888-EZR32-QuickStart.pdf (lastet ned 10.06.18). Basert på riteriene nevnt over ble de riktige binærfilene for prosjektet valgt.

USB host-API

I tillegg til wM-Bus spesifikke biblioteker var det også nødvendig med et bibliotek for utviklingen av USB host-applikasjonen på PC'en, hvor data skulle mottas fra collector. For dette formålet ble Libusb-win32 API'et brukt. Libusb er et bibliotek for utvikling av user-space USB applikasjoner. Libusb virket ideelt for prosjektets formål ettersom det er relativt simpelt og har støtte for asynkron bulk transfer mode, noe som er passende for overføring av større mengder med måledata.

Boto 3

For lagring og prosessering av måledata ble det bestemt at Amazon Web Services (AWS) skulle brukes som skytjeneste. I den forbindelse ble det installert et Python bibliotek kalt Boto 3. Boto 3 gjør det mulig å utvikle Python applikasjoner som benytter seg av flere av AWS sine tjenester, inkludert S3, EC2, og DynamoDB. Boto 3 er tilgjengelig for nedlasting på https://github.com/boto/boto3 eller ved bruk av Pythons package installer, pip.

5.4 Bruk av utviklerverktøy

Avsnitt 5.2 beskrev hvilke utviklermiljø som ble valgt og hvorfor de ble valgt. Dette avsnittet beskriver hvordan utviklerverktøyene og andre viktige ressurser ble satt opp og konfigurert for de forskjellige formålene i prosjektet.

Template wM-Bus-prosjekt

I nedlastingen av wM-Bus software-stacken fulgte det med en mappe kalt *Wireless M-BUS stack*. Denne mappen inneholder wM-Bus biblioteket og andre relevante filer for bygging av en wM-Bus applikasjon. En template-prosjektfil kan finnes på følgende plassering

Wireless M-BUS stack\WMBUS Tools\Firmware\APL firmware\ide\iar\Demo_SLWSTK6220A.

Filen heter *Demo_appapl.ewp* og i prosjektet ble denne filen brukt som utgangspunkt for utviklingen av både meter og collector-applikasjonen. Den inkluderer blant annet en main-fil for både collector og meter, som fungerer som et skjelett for videre applikasjonsutvikling. Filen ble importert i IAR Embedded Workbench, og modifisert for oppgavens formål, noe senere avsnitt vil beskrive i større detalj.

IAR build configurations

Etter å ha importert template-prosjektet til IAR var det nødvendig å konfigurere IAR til å passe med hardwaren og annen kildekode i prosjektet. Det var behov for en egen konfigurasjon for henholdsvis meter og collector. For en grundigere beskrivelse av hvordan build configurations ble satt opp henvises det til (Håkon H. Lier, 2017). Her står også en beskrivelse av hvordan bygging og debugging av applikasjonen ble gjort i det tidligere prosjektet, noe som ikke skiller seg vesentlig fra hvordan det ble gjort i dette prosjektet.

C++ applikasjonsutvikling på PC

For å kunne kompilere og kjøre C++ applikasjonen på PC'en var det nødvendig å installere MinGW kompilatoren. MinGW kan lastes ned fra sourceforge via linken https://sourceforge.net/projects/mingw/files/. På samme måte som i GCC opprettes en makefile hvor alle build configurations er spesifisert, sånn som include paths og library paths og source filer som skal kompileres.

Håndteringen av USB kommunikasjon var en sentral del av prosjektet, og for dette formålet ble Libusb benyttet. Libusb ble installert ved hjelp av installasjons filer som fulgte med source koden i Silicon Labs' application note, an0065, som ble lastet ned fra linken

https://www.silabs.com/documents/public/example-code/an0065-efm32-usb-device.zip. For å kunne bruke Libusb trenger man å installere den tilhørende kernel space driveren, libusb0.sys, på datamaskinen. Dette ble gjort ved å kjøre filen inf-wizard.exe i an0065_efm32_usb_device\host\libusb-win32-bin-1.2.6.0\bin, mens USB-enheten som driveren skulle kommunisere med, altså USB-interfacen på EZR32WG kortet, var koblet til.

Utvikling av Pythonapplikasjon

For utvikling av applikasjonen for kommunikasjonen med AWS ble Python 3.6 benyttet. Python er tilgjengelig for eksempel på nettstedet https://www.python.org/downloads/. I tillegg til Python 3.6 var det i tillegg nødvendig å installere Boto 3 biblioteket. Det ble gjort ved å kjøre kommandoen *pip install boto3*. Dette krever at man har installert pip, noe som i prosjektet ble gjort i forbindelse med Python 3.6 installasjonen. Det er også mulig å installere Boto 3 ved å klone https://github.com/boto/boto3.

For å kunne bruke AWS sine tjenester trenger man en AWS account. Man trenger også en AWS access key for autentisering før man kobler til en AWS ressurs. Det ble opprettet en account på AWS sine nettsider og AWS access key ble generert fra AWS' IAM Management Console.

5.5 Modifikasjon av wM-Bus template-koden

I tillegg til LED funksjonalitet for debugging og endring av signalstyrke ble det også gjort en hel del andre modifikasjoner til den opprinnelige template kildekoden. Den opprinnelige wM-Bus template applikasjonen var kun ment for å sende konstante test data mellom to wM-Bus devices, og hadde ikke noe praktisk formål utover det. Det var derfor nødvendig å gjøre endringer, blant annet på hvor mange meter devices collectoren kunne motta data fra, hvor ofte dataene ble sendt, og hva som ble sendt. Dette underkapittelet vil beskrive hvilke modifikasjoner som ble gjort til kildekoden og hvorfor de ble gjort.

En uheldig tendens med wM-Bus software stacken ble observert ved flere anledninger, deriblant når det ble gjort forsøk på å endre meter-devicens sendeintervall. Sendeintervallet

settes i initialiseringsstrukturen til meter, og er spesifisert i millisekunder. For eksempel vil en verdi på 10000U i den siste attributten i initialiseringsstrukturen indikere at sendeintervallet skal settes til 10 sekunder. Forsøk på å endre denne verdien var ikke alltid vellykket. Det krevdes ofte flere flashinger av både collector og meter for å få en observert endring i sendeintervall. Det var også problematisk å endre andre former for parametere, dersom de var hardkodet. Det var uvisst hva årsaken til denne oppførselen var, og det var ikke mulig å lese fra dokumentasjonen til stacken. Det ble gjort et forsøk på å wipe flashminnet etter initialisering. Tanken bak dette var at meter og collector kildekoden tilsynelatende lagret all konfigurasjonsdata i flashminnet på kortene, ettersom konfigurasjonsdataene vedvarte selv etter flashing av kortene. For å unngå å skrive over kritiske deler av firmwaren, som for eksempel bootloaderen, var det viktig å først identifisere hvilke deler av minnet config-dataene ble skrevet til. Ved å bruke IARs debuggingverktøy var det mulig å se hvilke deler av minnet som inneholdt hva. Konklusjonen var at all config-data ble skrevet til nest siste page i flashminnet. Ved å kjøre følgende kode etter initialiseringen var problemet med hardkoding av config-data løst.

```
void erase_mem(void)

{

 uint32_t *addr = (uint32_t *)0x0003F800;

 MSC_Init();

 MSC_ErasePage(addr);

 MSC_Deinit();

}
```

MSC_ErasePage() ble kalt med adressen 0x0003F800 ettersom dette var starten av nest siste page. Etter denne endringen var det mulig å sette alle parametere i koden uten at de ble «overridet» av config-data som eksisterte i minnet.

I kildekoden for meter ble det gjort noen endringer til hva som sendes i wmbus_apl_evt_userDataRequested(). I den opprinnelige koden ble det kun sendt konstante data, med andre ord meter sendte den samme dataen til collector hver gang. Hensikten med

meter-devicen var at den skulle simulere en trykkmåler, og en trykkmåler vil åpenbart kunne sende tidsvarierende data. Derfor var det nærliggende å endre koden slik at meter også sendte tidsvarierende data. Koden ble endret slik at meter sendte en tilfeldig verdi mellom 10 og 15 bar. Applikasjonslaget i M-Bus bruker binærkodet desimal formatet (BCD) til å kode måledataene. For å få måledataene til å variere mellom 10 og 15 ble LSB satt til et tilfeldig tall mellom 0 og 5, og MSB ble satt til 1. For å sørge for at alle tre meterne genererte forskjellige sekvenser av måledata ble srand() kalt med unike seed verdier for alle tre. Den nest siste byten i hvert telegram ble satt for å indikere hvilken meter telegrammet ble sendt fra, med andre ord en slags id.

En annen endring som ble gjort i meter koden var relatert til broadcasting av måledata, i henhold til krav nummer seks i spesifikasjonen. For å få meter-devicene til å sende data uten noen spesifisert destinasjonsadresse var det nødvendig å gjøre enda en endring i initialiseringstrukturen til meter. Ved å sette attributt nummer fire til NULL istedenfor collector adressestrukturen vil meter sende til en uspesifisert adresse, altså broadcaste måledataene.

Collectoren trengte fortsatt informasjon om meterne for å kunne motta telegrammene riktig, både adressen og krypteringsnøkkelen til hver meter-device. Meter-adressene og nøklene er også en form for konfigurasjon som lagres i flashminnet, så det var viktig å legge dem til etter kallet til erase_mem(), siden den overskriver dataene i minnet som tidligere nevnt. Funksjonen wmbus_apl_col_meterAdd() ble kalt tre ganger med forskjellige verdier for meter-adressen. Meter-adressene trengte for øvrig å endres til trykkmåler-adresser, ved å sette siste felt i adressen til *WMBUS_DEV_TYPE_PRESSURE*. Nøkkelen var den samme for alle meterne.

En siste endring som ble gjort var til callback funksjonen for håndtering av mottatte telegrammer i collector, wmbus_apl_evt_tlgAvailable(). Siden måledataene skulle sendes fra collector til en PC over USB var det behov for å gjøre noen kall til USB-stacken. For utvikling av USB funksjonaliteten på SLWSTK6220A kortet ble Silicon Labs sitt USBD-API benyttet. Neste avsnitt vil beskrive kommunikasjonen med brukergrensesnittet og vil gå nærmere inn på hvilke kall som ble gjort og hvor i koden de ble gjort.

5.6 Utvikling av brukergrensesnitt

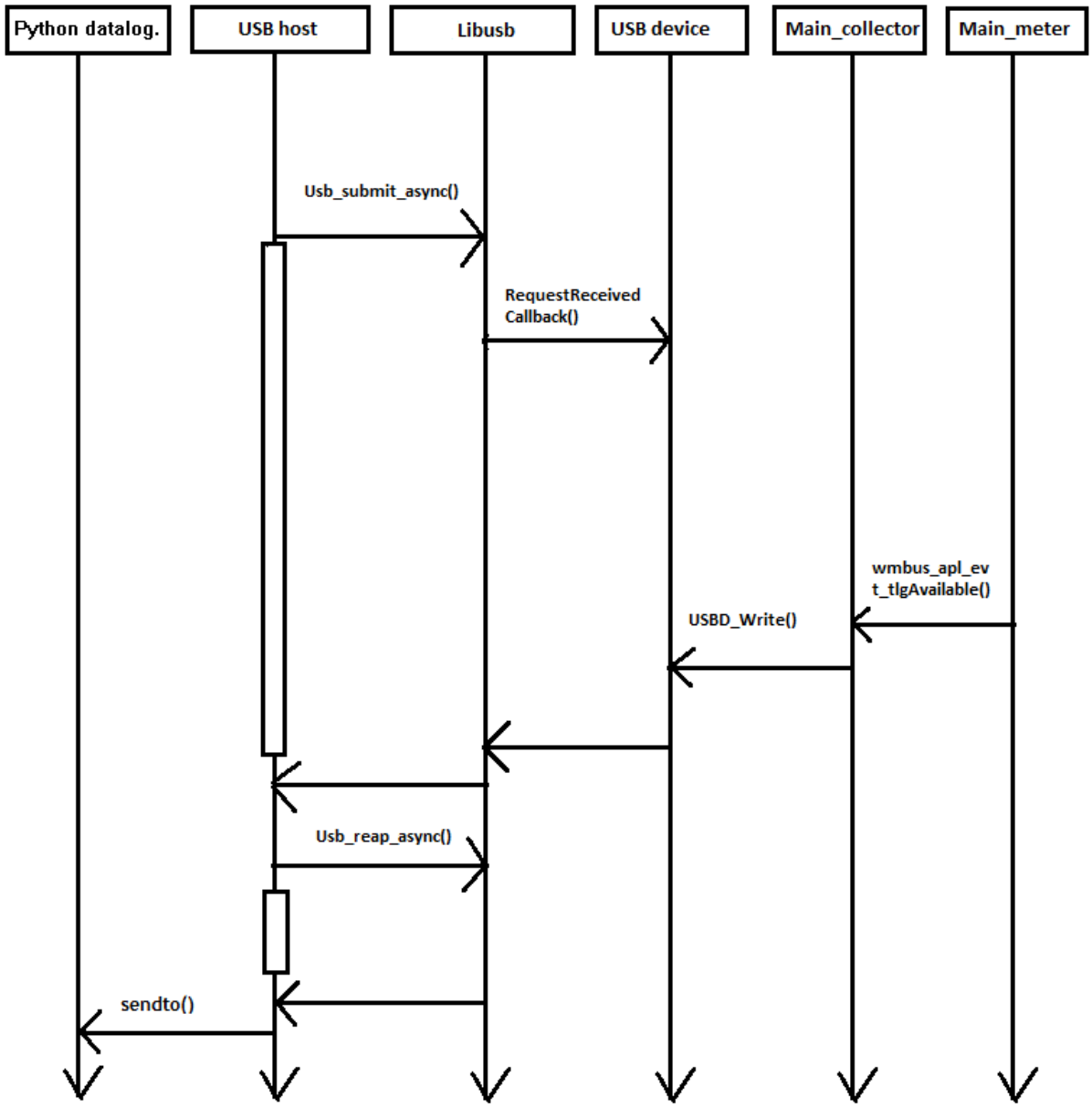
Brukergrensesnittet er en viktig del av systemet, ettersom den i tillegg til å gi brukeren et interface til collector også har ansvar for å sende dataene videre for å kunne lagres i databasen. Dataene overføres fra collector til PC'en over USB, og brukergrensesnittet trenger derfor å håndtere en USB-forbindelse for å kunne motta data. For dette formålet blir som tidligere nevnt Libusb benyttet.

USB-kommunikasjon foregår mellom en såkalt USB-host og en USB-device. Forskjellen mellom de to er grovt sett at hosten er den som starter kommunikasjonen og devicen kun responderer når den blir forespurt. I prosjektet er collector devicen konfigurert som USB device og PC'en er konfigurert som host, og vil i resten av kapittelet bli referert til som sådan.

Kommunikasjonen mellom device og host foregår i bulk transfer mode. Denne måten å overføre data på er ideelt for større datamengder. Brukergrensesnittet på host siden er implementert som en enkel konsollapplikasjon. På host siden blir tre forskjellige funksjoner implementert; forespørsel om data, endring av collector adresse, og legge til meter device i collector. Hver funksjon aktiveres ved at en forespørsel sendes fra brukergrensesnittet. Forespørselen håndteres på device-siden i requestReceivedCallback() i callbacks.c. Ved forespørsel om data vil host sende en forespørsel til device og device vil respondere ved å sende måledata til host fortløpende. På device-siden implementeres dette ved at det gjøres et kall til USBD_Write() i wmbus_apl_evt_tlgAvailable(), som er en callback for mottatte telegrammer. Overføringen av data vil opphøre etter et intervall spesifisert av brukeren. Brukeren kan også velge å la overføringen foregå over en ubegrenset periode. Dataene som ankommer brukergrensesnitt-applikasjonen vil bli skrevet til et terminalvindu og videresendt til et lokalt Python-program over sockets. Python-programmet er ansvarlig for kommunikasjonen med databasen og vil bli nærmere beskrevet i avsnitt 5.8. Ved forespørsel om endring av adresse vil host be brukeren om å skrive inn en åtte byte adresse og sende denne adressen videre til device med en forespørsel om å endre collector adressen til denne adressen. Ved forespørsel om å legge til meter vil brukeren bli bedt om å spesifisere en adresse og en krypteringsnøkkel. Deretter vil host sende denne informasjonen til device med en forespørsel om å legge til en meter med denne adressen og nøkkelen til collectorens liste

av meter devices. Listen er en datastruktur som inneholder informasjon om meter devices som har «tillatelse» til å sende telegrammer til collector. Alle andre meter devices vil i teorien bli ignorert.

Figur 13 viser et enkelt sekvensdiagram for dataoverføringsfunksjonen. Piler uten beskrivelse representerer interrupts eller annen form for overføring av kontroll.



Figur 13, dataoverføring sekvensdiagram

5.7 Logging av data i skytjeneste

Noe av hensikten med å sende måledata til PC'en var å kunne observere måledataene i sanntid, men kanskje viktigere var å kunne logge dataene i en database. Det er flere grunner til at denne databasen burde hostes i en skytjeneste, deriblant at dataene vil være tilgjengelige fra flere steder med en internettilkobling. Andre grunner vil bli diskutert i avsnitt 7.4.

Det ble til slutt bestemt at loggingen skulle foregå ved å bruke Amazon Web Services sine tjenester. En av fordelene med AWS er at en del av tjenestene deres, inkludert de som ble benyttet i prosjektet, er gratis opptil en viss grense. Denne grensen er definert i AWS sin såkalte free tier limit. I dette prosjektet var det ikke behov for å overskride denne grensen så AWS var et naturlig valg når det kom til skytjeneste som skulle benyttes. AWS har også et veldig simpelt og allsidig Python API, kalt boto 3, som ble benyttet i prosjektet. I tillegg til funksjonalitet for logging av data ble det også implementert en funksjon for å hente data fra databasen og plotte informasjonen i en graf. Dette underkapittelet vil i hovedsak dreie seg om bruken av boto 3 og behandling av måledata i Python applikasjonen.

Som nevnt i forrige underkapittel blir måledataene sendt fra C++ applikasjonen til et Python program over sockets. Det er flere måter å overføre data mellom en C++ applikasjon og Python, men sockets virket som en relativt enkel og lettvektig måte å gjøre det på. Dataene som blir overført er som tidligere nevnt i BCD form, siden brukergrensesnitt-applikasjonen ikke gjør noen form for parsing på måledataene før de sendes videre. Konverteringen fra BCD til integer form ble håndtert i Python applikasjonen.

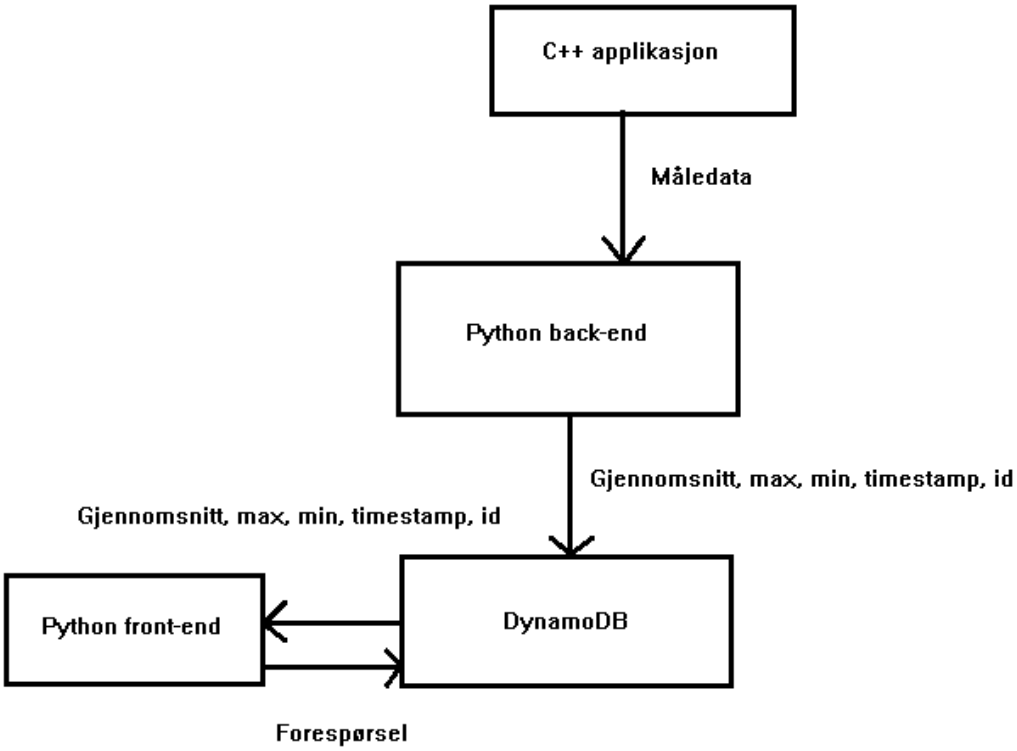
Et av kravene i spesifikasjonen er at data skal sendes til skyen med et 90 sekunders intervall. Derfor kan ikke måledataene sendes videre til databasen med en gang de ankommer Python applikasjonen, men trenger isteden å bli mellomlagret før de sendes videre. I tillegg ble det bestemt at kun gjennomsnittet, i tillegg til minimum og maksimum av dataene skulle sendes til skyen, noe som begrenser mengden med data som overføres. Gjennomsnitt skal beregnes fra de ni siste måledataene som ankommer Python applikasjonen, såkalt «moving average». Det samme gjelder minimum og maksimum. For dette formålet ble det implementert en egen Python module, kalt comp.py. Den implementerer en FIFO kø med funksjoner for å hente ut minimum, maksimum, og gjennomsnitt av et bestemt antall elementer. Dataene

lagres i et deque objekt som er en del av Pythons collections package. I Python applikasjonen opprettes det tre slike datastrukturer, en for hver meter. Størrelsen ble satt til ni elementer, ettersom Python applikasjonen mottar ni målinger fra hver meter mellom hver gang den skriver til databasen.

Python applikasjonen har to oppgaver å håndtere. Den ene går ut på å motta og lagre data fra C++ applikasjonen. Den andre går ut på å beregne gjennomsnitt/minimum/maksimum og sende det til databasen. Den siste oppgaven skal skje med et fast tidsintervall, så det gir mening å fordele disse oppgavene på to tråder, hvor tråden som skriver til databasen kjører en gang hvert 90. sekund, og den andre tråden kjører når den første ikke kjører. Begge trådene aksesserer køen med måledata; den ene skriver til den og den andre henter ut gjennomsnitt/minimum/maksimum. Det benyttes derfor for ordens skyld et mutex lignende objekt til å gi trådene eksklusiv aksess til køen.

Databasen som blir skrevet til er en AWS DynamoDB database. For å bruke AWS ressurser i boto 3 kreves det at man har en access key. I Python applikasjonen brukes denne nøkkelen i et kall til boto3.resource(), hvor nøkkelen og nøkkelens id inngår som parametere. Type ressurs (dynamoDB) og region blir også spesifisert i dette kallet. Hvert item som blir skrevet til databasen har fem attributter, gjennomsnitt, maksimum, minimum, timestamp, og id. Gjennomsnitt blir skalert med en faktor 100 fordi dynamoDB ikke klarer å håndtere flyttall, og det er ønskelig å beholde en viss nøyaktighet på dataene. Timestamp er antall sekunder siden 1. jan 1970. Id indikerer hvilken meter-device dataene tilhører, og går fra 0 til 2. For å unikt identifisere hvert item i databasen ble det brukt to key attributter. Den ene er timestamp. Det finnes tre items med samme timestamp i databasen til enhver tid, en for hver meter. For å unikt identifisere hvert item var det derfor nødvendig å i tillegg bruke id'en til hver meter som key. Timestamp ble valgt som sort key, altså «KeyType» ble satt til «Range».

Figur 14 viser kommunikasjonen mellom de forskjellige komponentene i Python applikasjonen og medvirkende komponenter.



Figur 14, python-dataloggerapplikasjon

5.8 Plotting av data

Front-enden i applikasjonen er et Python program som genererer grafer basert på måledataene i databasen. For plotting av grafer brukes biblioteket matplotlib. Matplotlib støtter både generering av 2-dimensjonale og 3-dimensjonale grafer, noe som gjorde matplotlib ideelt for plotting av måledataene. 2-dimensjonal graf er nødvendig for å vise variasjonen av trykket med hensyn på tiden, og 3-dimensjonal graf er nødvendig for å vise variasjonen i trykk basert på tid og måler.

Python programmet ber brukeren spesifisere et tidsintervall og genererer deretter en graf av dataene som ble skrevet til databasen i dette tidsintervallet. Det genereres da en 3-

dimensjonal wireframe-graf med tiden i sekunder på x-aksen, meter-id på y-aksen, og trykket i bar på z-aksen. Brukeren kan også velge å generere 2-d grafer for en spesifikk meter-device. Dette vil være mer oversiktlig enn en 3-d graf, men viser ikke sammenhengen mellom trykket i de forskjellige målerne.

Kapittel 6 – Resultater

Etter implementasjonen av funksjonene i systemet ble de forskjellige funksjonene testet med hensyn på kravene i spesifikasjonen. Følgende kapittel vil beskrive testene som ble gjort og de respektive resultatene.

6.1 Initiell testing av wM-Bus kommunikasjon

Allerede etter at template-prosjektet hadde blitt kompilert og flashet på SLWSTK6220A kortene med små modifikasjoner til kildekoden var det klart for å teste deler av funksjonaliteten i henhold til spesifikasjonen. Dette var tester som ikke var avhengig av at resten av systemet fungerte, så disse testene ble gjort på et relativt tidlig tidspunkt. Det var primært krav nummer fire, fem, og seks som ble testet i denne fasen.

For å indikere sending og mottak av wM-Bus telegrammer ble det benyttet LEDs som var en del av periferi-enhetene på kortet. I APL-API'et finnes det to callback-funksjoner, *wmbus_apl_evt_tx()*, og *wmbus_apl_evt_tx()*. Disse kalles ved henholdsvis sending og mottak av gyldige wM-Bus telegrammer til applikasjonslaget. Disse ble brukt i prosjektet til å bekrefte vellykket sending og mottak av telegrammer. LED lampen på meter ble programmert til å blinke hver gang telegrammer ble sendt, og LED lampen på collector ble programmert til å blinke hver gang telegrammene ble mottatt korrekt.

Selv om alle telegrammer ble overført riktig når collector og meter var plassert i nærheten av hverandre, var det fortsatt viktig å sørge for at de også kommuniserte på lengre avstander, med tanke på krav nummer fire i spesifikasjonen. Man kunne, ved å endre strømforsyningen til batteri, bevege collectoren bort fra meter, og observere når LED lampen sluttet å blinke. Det betydde at maksimal rekkevidde var nådd.

Rekkevidden ble testet inne i en bygning, så den maksimale rekkevidden ble sannsynligvis påvirket av vegger og andre objekter som befant seg mellom senderen og mottakeren. Det var også mest hensiktsmessig å gjøre denne testingen innendørs, ettersom vannmålere normalt er installert innendørs.

Den første gangen rekkevidden ble testet stoppet overføringen når distansen overskred cirka 20 meter. Dette var åpenbart ikke tilfredsstillende, så det var nødvendig å gjøre endringer i kildekoden til meter og collector. Følgende kall ble lagt til i koden etter initialiseringen av både meter og collector;

```
sf_rf_setSignalStrength(0xFE);
```

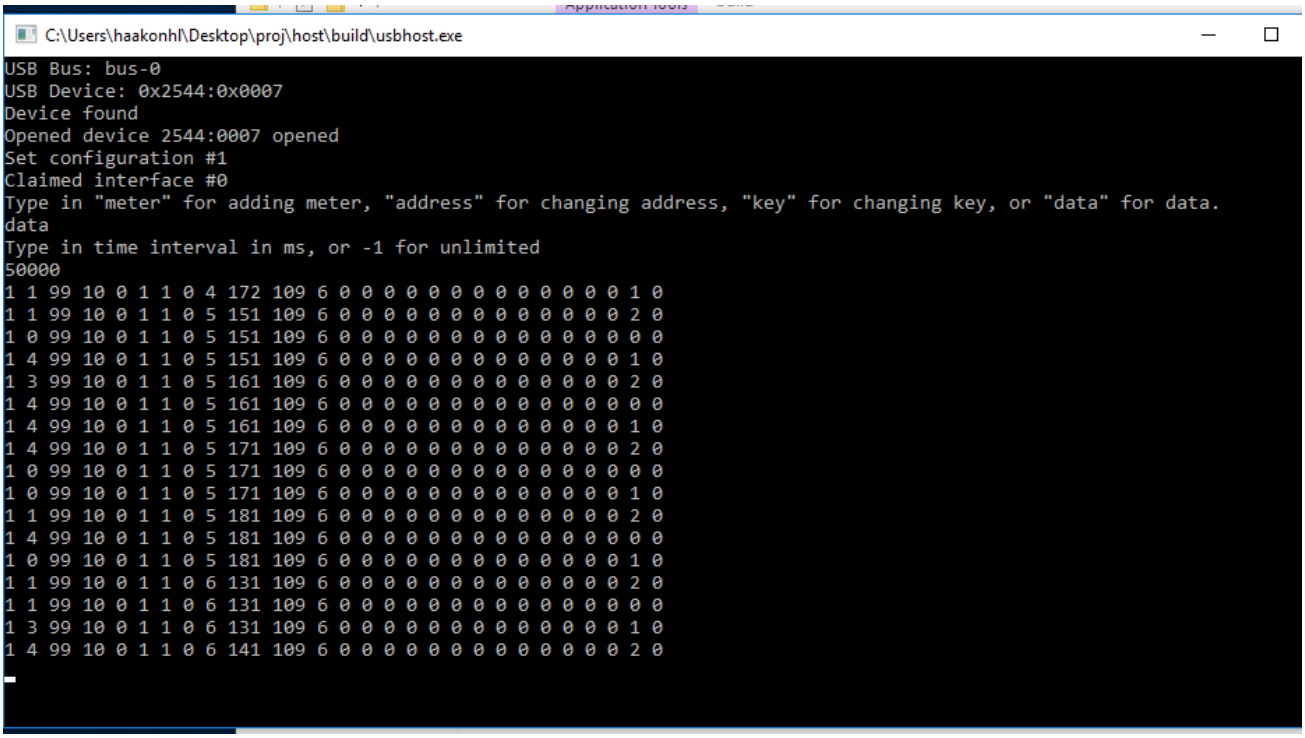
Denne delen av koden setter signalstyrken til senderen til høyest mulige verdi. Etter at denne endringen hadde blitt implementert ble samme test som over foretatt igjen, og resultatet var at rekkevidden hadde økt til cirka 100 meter, noe som fortsatt er mindre enn ønskelig, men en betydelig forbedring.

6.2 Kommunikasjon mellom collector og brukergrensesnitt

En del av testingen gikk ut på å verifisere kommunikasjonen mellom collector og brukergrensesnitt-applikasjonen på PC'en. Denne fasen av testingen adresserte krav nummer to, tre og syv i spesifikasjonen, ettersom disse har med kommunikasjonen mellom collector og brukergrensesnittet å gjøre.

Først ble det sendt en forespørsel om data fra brukergrensesnittet. Figur 15 viser et skjermbilde av det som ble skrevet til terminalvinduet som et resultat av forespørselen. De fire første bytene utgjør record 1. De neste bytene utgjør record 0, som er et timestamp. Nest siste byte angir som tidligere nevnt opprinnelsen til telegrammet. Fra figuren er det tydelig at både dataene i record 2 og timestampen endrer seg fra telegram til telegram, i tillegg til «id-byten».

En observasjon som ble gjort var at data ble skrevet til terminalvinduet umiddelbart etter at LED lampen i meter indikerte at et telegram ble sendt. LED lampen i collector viste tilsvarende at et telegram ble mottatt. Tiden mellom hver sending ble ikke nøyaktig målt, men det virket som om tiden lå på cirka 10 sekunder ved manuell telling.



Figur 15, data, brukergrensesnitt.

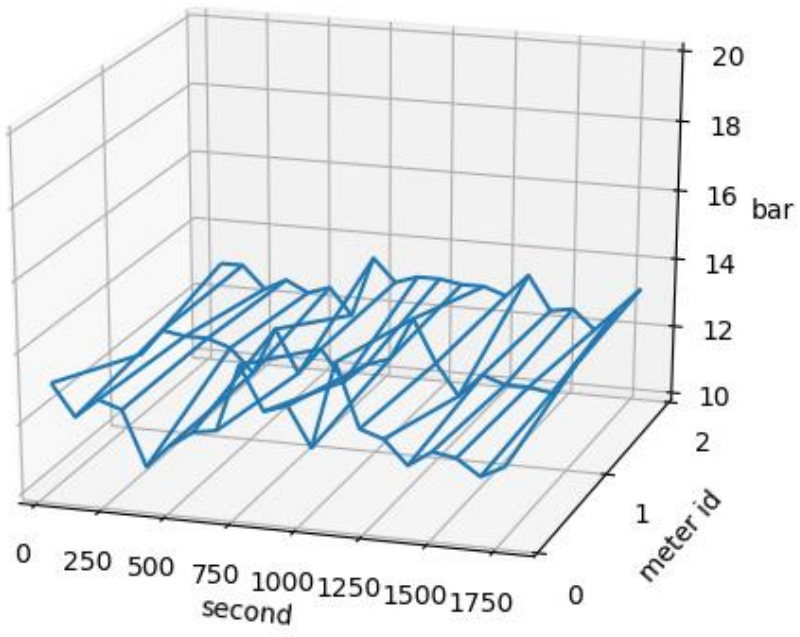
Andre del av krav nummer syv var at collector skulle kunne konfigureres fra brukergrensesnittet. En slik konfigurasjon involverer å kunne legge til meter-devices til collector ved å spesifisere nøkkel og adresse i brukergrensesnittet. Dette var en funksjon som ble utviklet og testet i forbindelse med arbeidet i (Håkon H. Lier, 2017), og det ble ikke gjort noen eksplisitte tester av denne funksjonen i dette prosjektet. Funksjonen fungerte i henhold til spesifikasjonen i det tidligere prosjektet, og så langt den ble brukt i dette prosjektet var det ingen tegn til avvik fra spesifikasjonen heller. For en grundigere beskrivelse av hvordan funksjonen ble testet ut henvises det til (Håkon H. Lier, 2017).

6.3 Kommunikasjon med database

De neste funksjonene som ble testet ut var de som hadde med kommunikasjonen med databasen å gjøre. Henholdsvis krav nummer åtte og krav nummer ni ble testet. Tre meter-devices ble programmert med hver sin adresse, og de ble lagt til i collector. Deretter ble collectorens USB interface koblet til en Dell Optiplex 9010 desktopmaskin. Alt av nødvendig software og drivere ble installert på desktop maskinen og det ble bekreftet at data ble

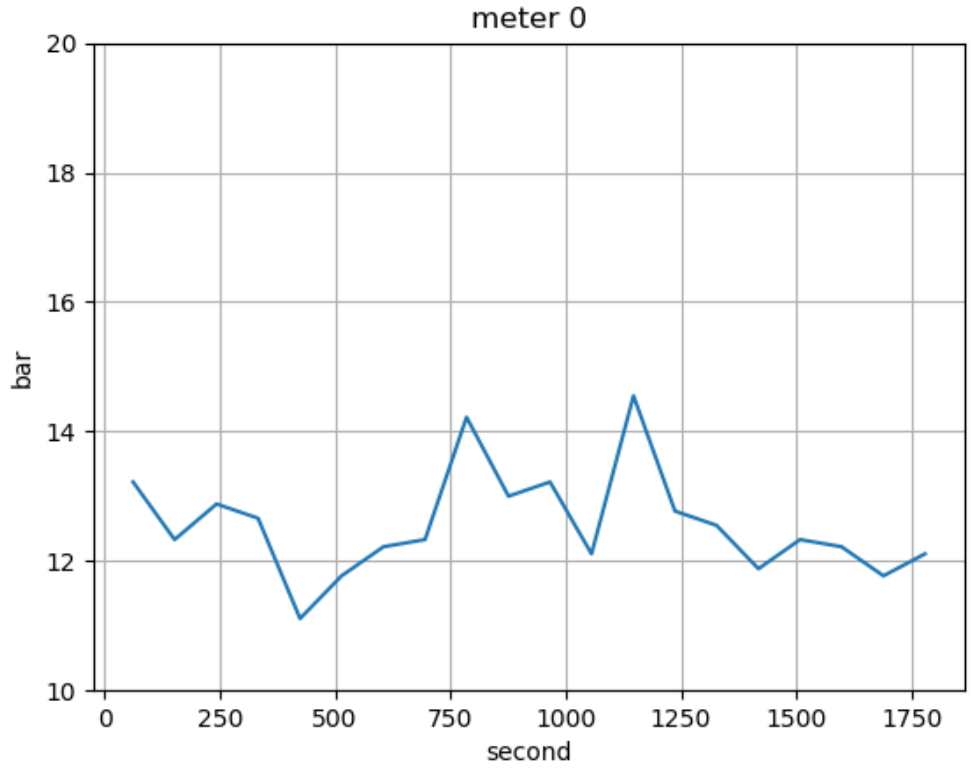
mottatt i terminalvinduet i brukergrensesnitt-applikasjonen. Deretter ble Python applikasjonen kjørt på desktop maskinen og data ble tilsynelatende skrevet til databasen. Loggingen foregikk fra tirsdag midt på dagen til fredag midt på dagen. Loggingen ble avbrutt på grunn av et strømbrudd, men det burde ikke ha noe å si for dataene som allerede var lagret i databasen.

Deretter ble det gjort et forsøk på å generere en 3-dimensjonal graf ved å kjøre plotdata3d.py. Tidsintervallet ble spesifisert til å være 2018-05-23 kl. 05.00-05.30 og typen data ble spesifisert til gjennomsnitt. Figur 16 viser den resulterende grafen.

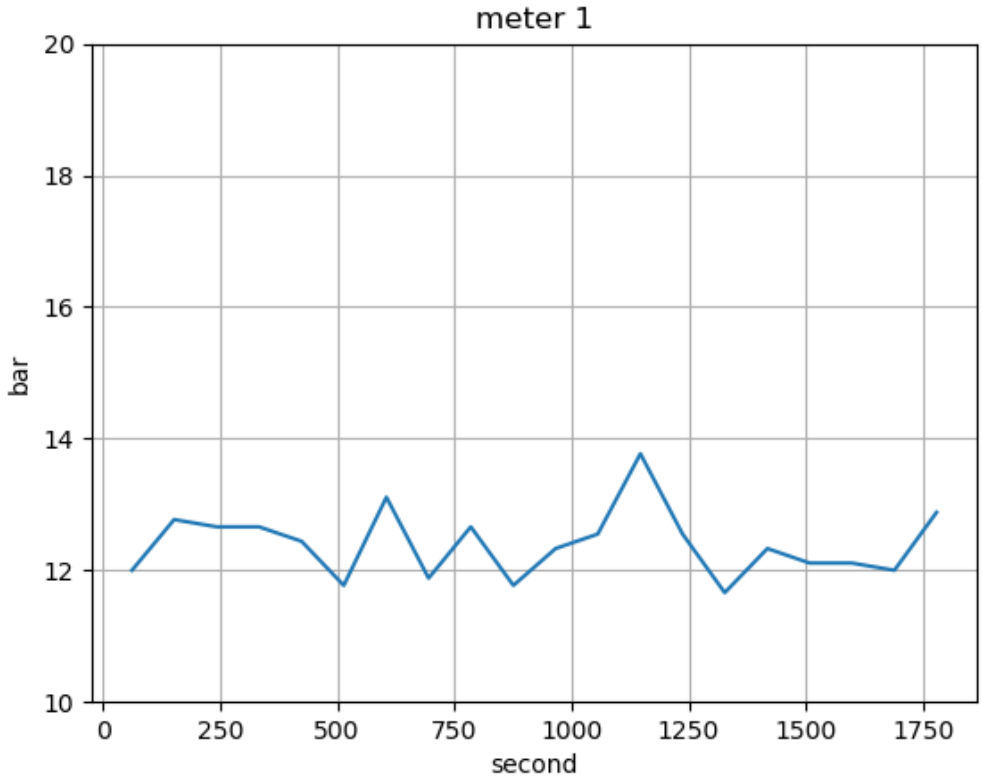


Figur 16, gjennomsnittlig trykk for alle målerne

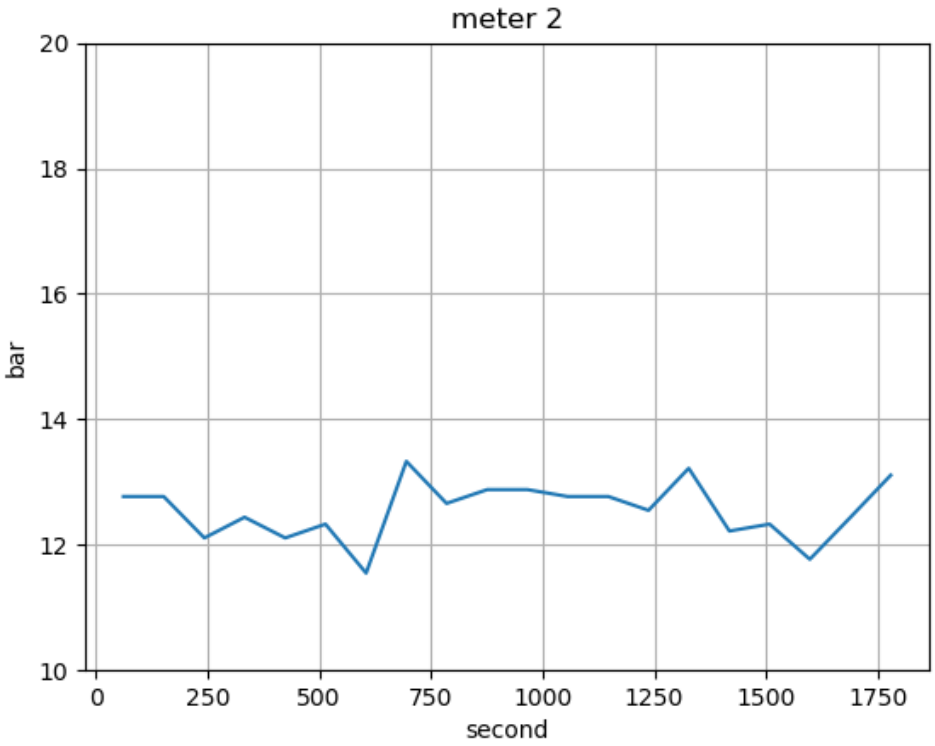
Det ble også gjort forsøk på å generere 2-grafer for hver meter-device. Figur 17 viser grafen for meter 0, figur 18 viser grafen for meter 1, og figur 19 viser grafen for meter 2. Alle 2-d grafene er fra samme tidsintervall som 3-d grafen i figur 16 og typen data er her også gjennomsnitt. Figur 20 viser en punktvis utgave av figur 17, for å gjøre det mer synlig hvor hvert enkelt datapunkt befinner seg.



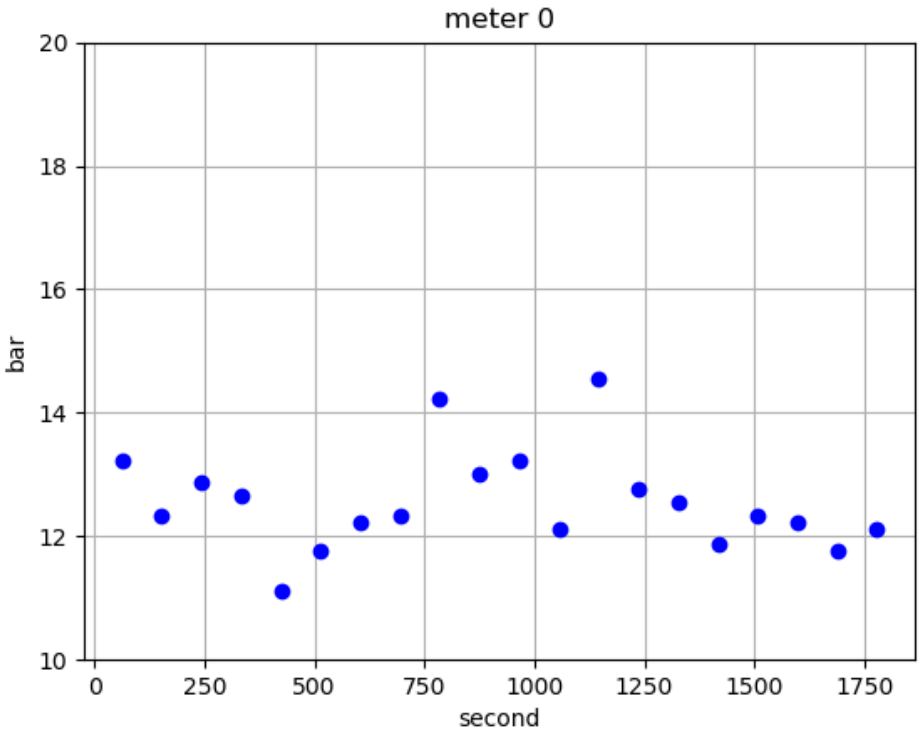
Figur 17, gjennomsnittlig trykk for meter 0



Figur 18, gjennomsnittlig trykk for meter 1

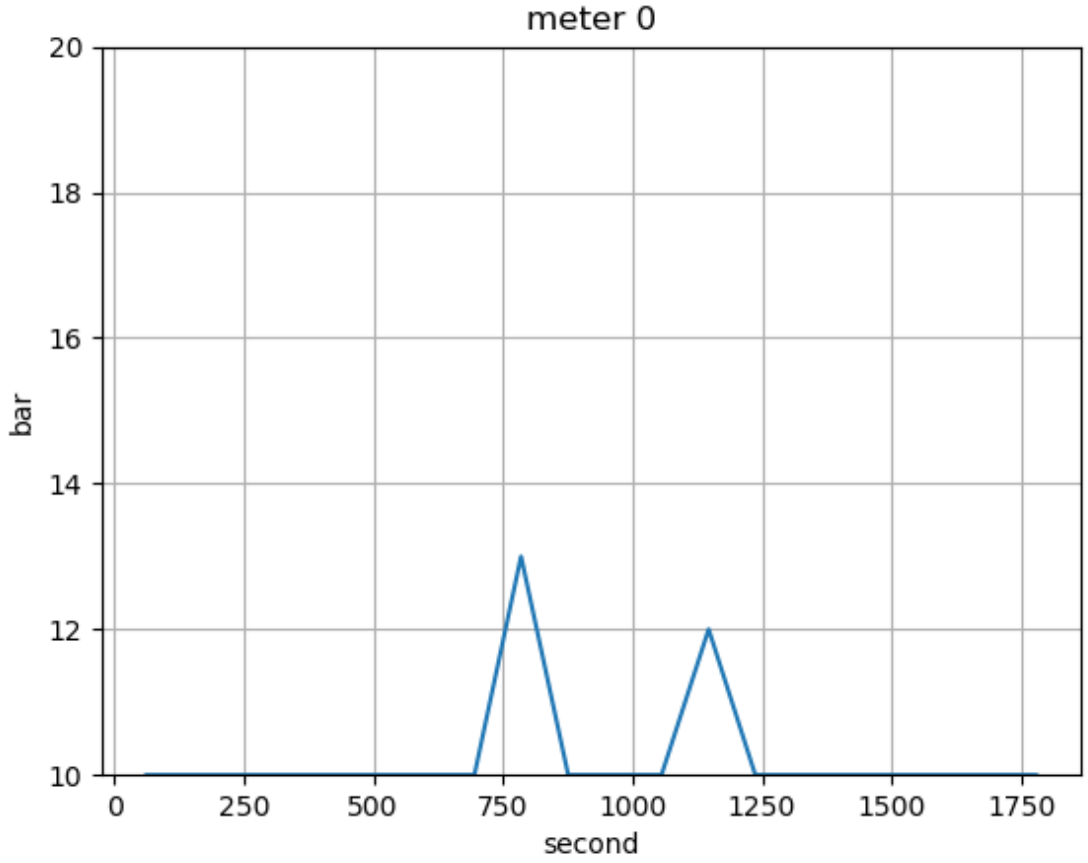


Figur 19, gjennomsnittlig trykk for meter 2

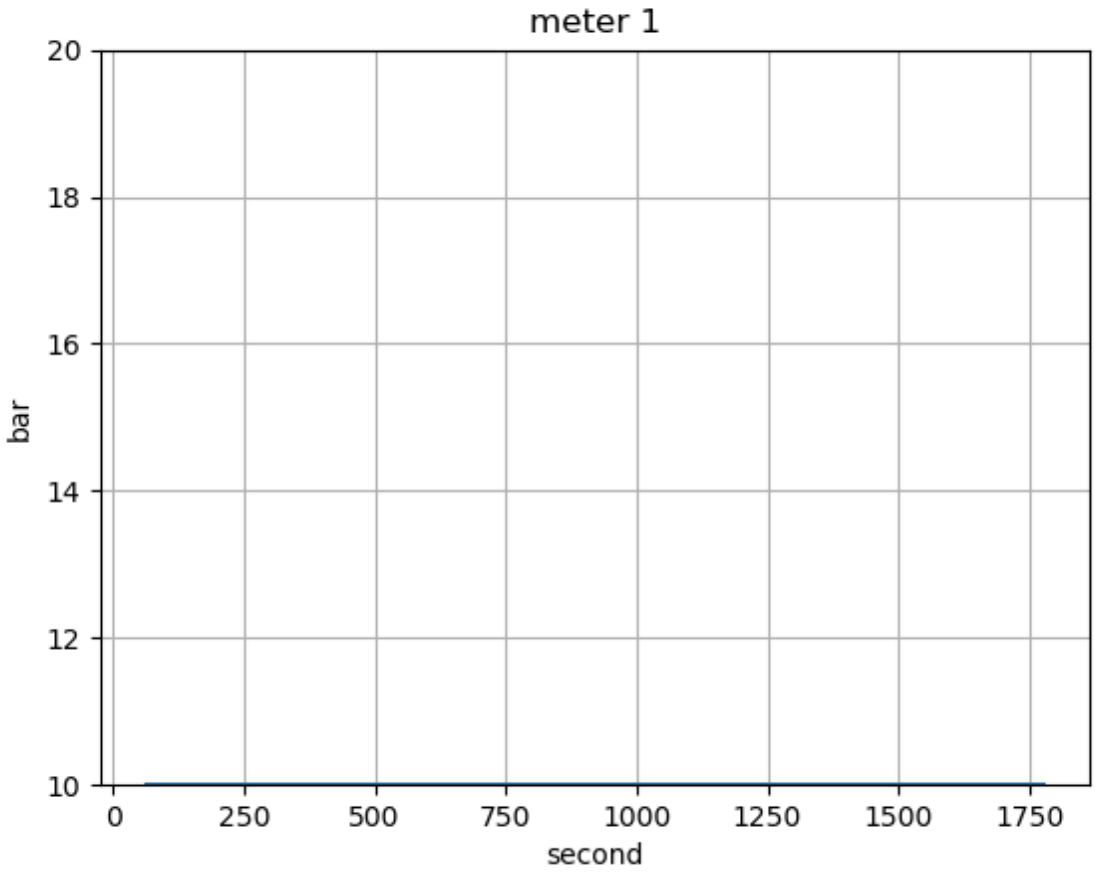


Figur 20, punktplot utgave av figur 17

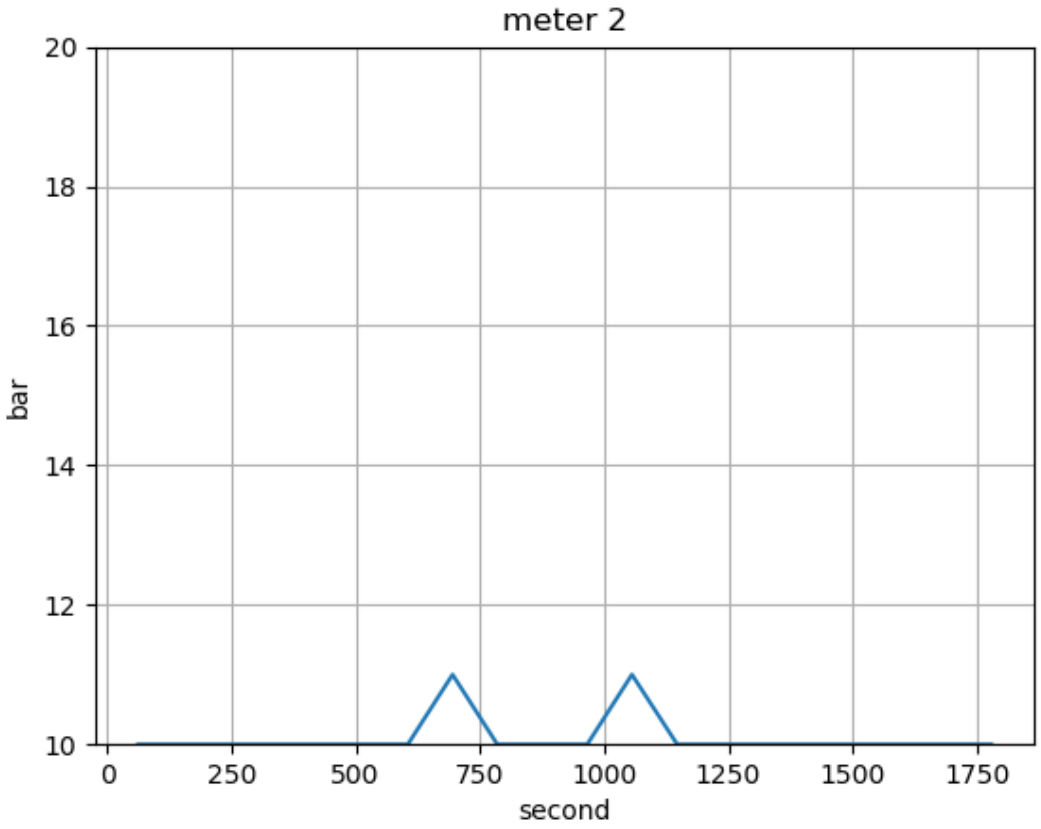
For å teste plottingen av minimale og maksimale trykk ble følgende grafer generert. Figur 21, 22 og 23 er minimale trykk for henholdsvis meter 1, 2, og 3. Figur 24, 25, og 26 er maksimale trykk for henholdsvis meter 1, 2, og 3.



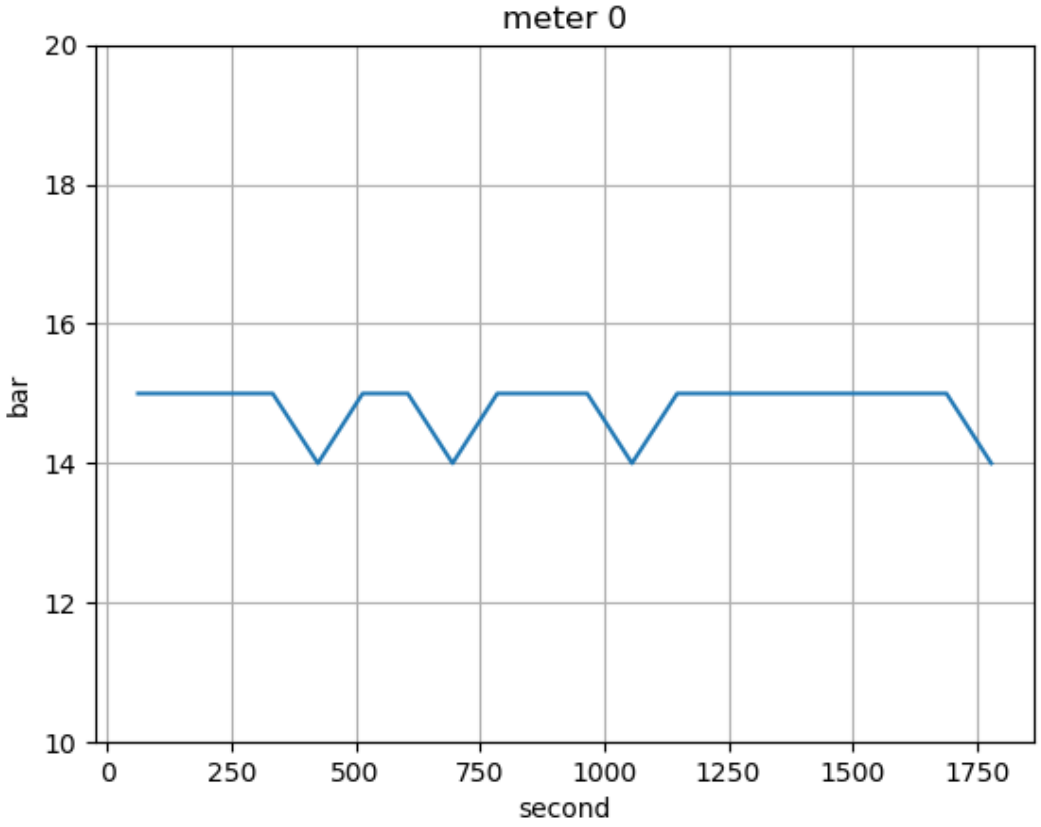
Figur 21, minimalt trykk for meter 0



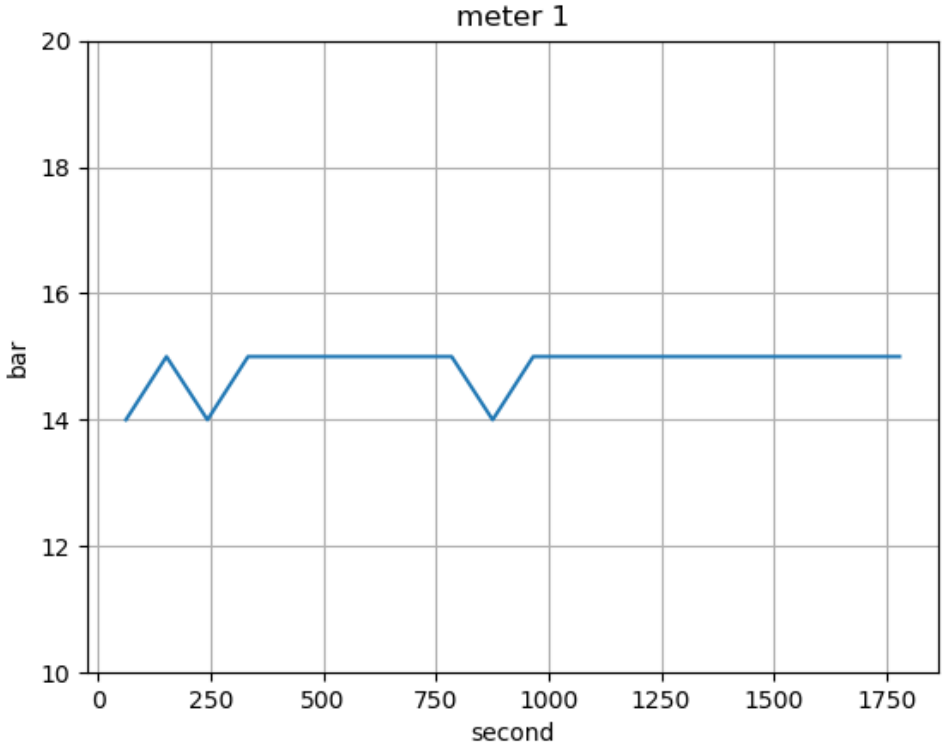
Figur 22, minimalt trykk for meter 1



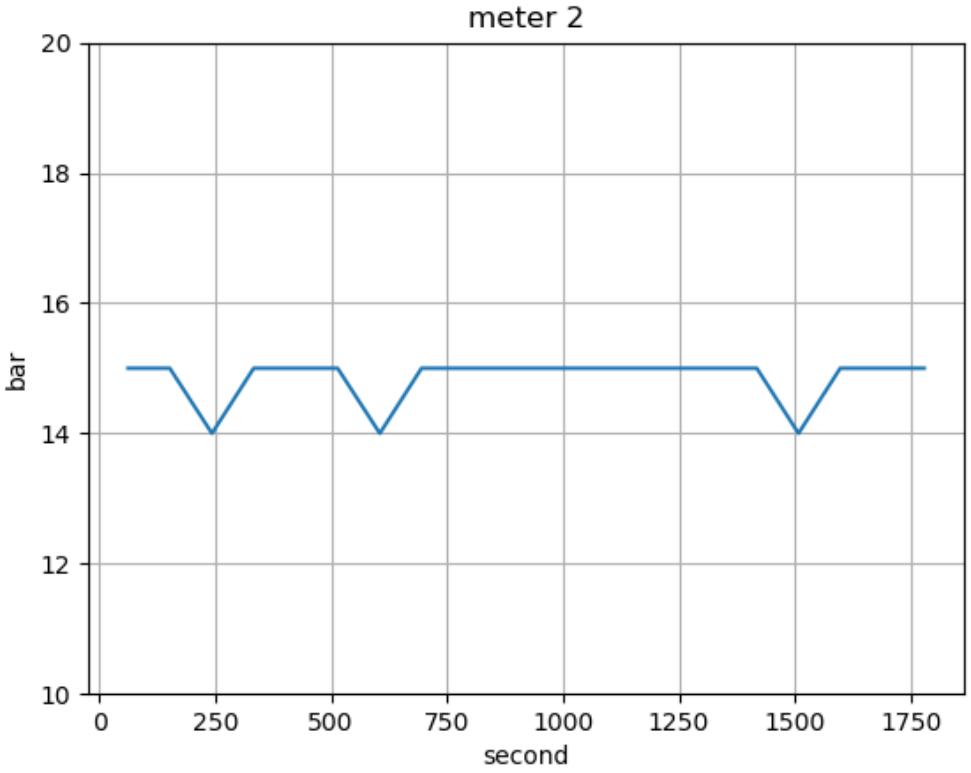
Figur 23, minimalt trykk for meter 2



Figur 24, maksimalt trykk for meter 0



Figur 25, maksimalt trykk for meter 1



Figur 26, maksimalt trykk for meter 2

Kapittel 7 – Diskusjon

7.1 Tolkning av resultater

Den initielle testingen av wM-Bus kommunikasjonen viser at systemets overensstemmelse med krav nummer fem i spesifikasjonen er intakt. Blinkingen av LEDs i både meter og collector betyr at telegrammene som blir sendt er gyldige så langt opp i stacken som til applikasjonslaget, så det er tydelig at meter og collector kommuniserer. Krav nummer seks er dermed også oppfylt, ettersom meterne var konfigurert til å broadcaste.

Krav nummer fire er også til en viss grad oppfylt, selv om det er avhengig av hvordan man tolker det. Rekkevidden er åpenbart ikke ideell, siden man gjerne vil dekke så mange boliger med en konsentrator som mulig for å redusere antallet konsentratorer. Med en slik rekkevidde kan man kanskje håpe på å hente data fra opptil ti boliger, avhengig av hvor tettbebygget området er. I en realisering av et slik datainnsamlingssystem vil det mest sannsynlig være ønskelig å benytte en sterkere transceiver for å kunne dekker et større område.

Testingen av brukergrensesnittet produserte ganske overbevisende resultater. Innholdet i record 1 var som forventet. Første byte skal alltid være 1. Andre byte skal være et pseudotilfeldig generert tall mellom 0 og 5, og i hver melding som skrives ut er byte nummer to innenfor dette intervallet. Id-byten veksler jevnt mellom 0, 1, og 2 som gir mening siden alle målerne sender med samme tidsintervall. Record 0 er som tidligere nevnt et timestamp, og for hver id blir timestamp inkrementert med et fast intervall, noe som gir mening siden målerne sender med et konstant tidsintervall. Det kan utfra denne informasjonen fastslås at krav nummer to er oppfylt. Krav nummer tre er ikke påvist fullstendig innfridd, ettersom ingen av målingene simulerer strømningsmålinger, men det er ingen grunn til å tro at det ikke vil fungere for strømning også, spesielt fordi dataene kun er simuleringer. Ellers ser krav nummer tre ut til å være oppfylt, ettersom dataene blir sendt og mottatt med det angitte tidsintervallet, og mottak og videresending skjer i samtidig med sendingen. Første del av krav nummer syv om sanntids overvåking av data ser også ut til å være oppfylt, ettersom måledataene skrives ut til terminalvinduet slik som forventet.

Andre del av krav nummer syv, som gikk ut på at collector skulle kunne konfigureres fra brukergrensesnittet, ble som nevnt testet i et tidligere prosjektarbeid, og konklusjonen var at denne funksjonen fungerte som den skulle. Dette kravet har ikke endret seg betydelig i siden den gang, og det burde derfor ikke være nødvendig med en ytterligere test i dette prosjektet for å kunne si at kravet fortsatt er oppfylt. Funksjonen ble imidlertid benyttet ved flere anledninger i det nåværende prosjektet, siden det var behov for å legge til meter-devices under utvikling og testing av andre deler av systemet. Funksjonen fungerte hensiktsmessige i alle disse tilfellene, og det var spesielt tydelig ettersom det i det nåværende prosjektet ikke var noen problemer med oppdatering av config-informasjon, på grunn av *erase_mem()* som først ble introdusert i det nåværende prosjektet.

Brukergrensesnittet har også en funksjon for å endre collector-adresse. Denne funksjonen har ikke blitt testet i det hele tatt. Det er imidlertid ingen behov for denne funksjonen per dags dato, så funksjonen forblir utestet uten at det har noen innvirkning på systemets konformitet med kravene i spesifikasjonen.

Grafene som ble generert i forbindelse med testingen av krav nummer åtte og ni var i stor grad som forventet. Den 3-dimensjonale wireframe-grafen ser ikke så nyttig ut i en rapport som dette, men i Python-applikasjonen genereres det en interaktiv graf som blant annet kan roteres og forstørres, noe som gjør den mer nyttig for brukeren enn det kan se ut som her. De 2-dimensjonale grafene i figur 17, 18, og 19 stemmer overens med hva man kan lese fra 3-d grafen i figur 16. Dette gir mening siden de er hentet fra samme tidsintervall. Ettersom applikasjonen genererer meningsfulle grafer virker det som krav nummer ni er oppfylt.

Et punktplot ble inkludert for å lettere se hvor hvert datapunkt befinner seg i grafen. En nyttig observasjon man kan hente fra denne grafen er distansen mellom hvert punkt på x-aksen. Det kan være vanskelig å avgjøre nøyaktig hvor langt det er, men det kan se ut som det tilsvarer cirka 90 sekunder. Dette kan bekreftes ved å kjøre boto 3 sin table.scan()-funksjon og skrive ut dataene direkte. Dette vil gi nøyaktig 90 sekunders forskjell mellom timestamp til hvert item. Krav nummer åtte ser derfor også ut til å være tilfredsstilt.

Det eneste kravet som ikke enda har blitt diskutert er krav nummer en. Dette kravet er, uten at det krever noen videre forklaring, oppfylt.

7.2 Ikke-simulert trykkmåler

I prosjektet ble det gjort forsøk på å motta måledata fra en Kamstrup trykkmåler. Hensikten var å bruke en slik trykkmåler i oppsettet, og deretter gå over til å bruke flere slike målere på sikt. Forsøket var dessverre ikke vellykket, og på grunn av mangel på tid trengte man derfor å prioritere gjennomføringen av andre deler av prosjektet.

Da kommunikasjonen mellom Kamstrup trykkmåleren og konsentratoren ble testet, ble adressen og krypteringsnøkkelen først hardkodet inn i firmwaren til konsentratoren, både ved å sende dem inn i *wmbus_apl_col_meterAdd()* og initialiseringstructen. Trykkmåleren skulle angivelig sende med et 90 sekunders intervall, men ingen telegrammer ble bekreftet mottatt i dette tidsrommet. Deretter ble det gjort forsøk på å legge den til som meter i brukergrensesnittet. Dette var heller ikke vellykket. Det ble undersøkt om trykkmåleren faktisk sendte telegrammer. Det ble først benyttet en spektrumanalysator, og det ble bekreftet at måleren sendte i 868 MHz båndet. Det ble på et senere tidspunkt også bekreftet at den sendte gyldige wM-Bus telegrammer.

Det er tydelig at trykkmåleren faktisk fungerte som den skulle, og at problemet med kommunikasjonen var på mottaker siden. Det er imidlertid ikke klart hva som årsaken til at telegrammene ikke ble mottatt av konsentratoren, og i hvilken del av mottaker kjeden informasjonen ble tapt. Dette er en problemstilling som kan håndteres i et senere prosjektarbeid, ettersom det potensielt kan være ganske tidkrevende og kreve mye feilsøking og debugging.

7.3 Svakheter

Det er mange nevneverdige svakheter i måten prosjektet ble gjennomført, og i følgende avsnitt vil en del av dem bli diskutert i større detalj.

Kanskje den største svakheten i analysen var at det ikke ble benyttet ekte trykkmålere, men isteden kun simulerte trykkmålere i oppsettet i prosjektet. Planen i starten av prosjektet var å benytte flere Kamstrup trykkmålere og strømningsmålere i et oppsett i en bolig for å måle faktiske trykk og strømninger i vannrør, og gjøre flere beregninger på måleverdiene. Slike

data ville vært mer verdifulle for analysen enn fabrikkerte data som ble brukt i dette prosjektet. Med faktiske måledata vil man kunne analysere nytteverdien og bruksområdene til et system for fjernavlesing av vannmålere, og ikke kun gjennomførbarheten som ble undersøkt i dette prosjektet.

En annen svakhet i gjennomføringen av dette prosjektet er måten målingene ble simulert. De tilfeldig genererte måledataene var uniformt distribuert mellom 10 og 15 bar, noe som ikke er en veldig realistisk modell for hvordan trykket varierer i en rørledning. Det var heller ingen sammenheng mellom målte verdier som var nærme i tid. Normalt ville det vært en sammenheng mellom to påfølgende målinger, med mindre trykkvariasjonene var spesielt raske. I tillegg til mangel på korrelasjon mellom målingene i tid var det kanskje enda viktigere en mangel på korrelasjon mellom det målte trykket i hver trykkmåler. Normalt skulle man forvente at trykkmålere som måler i samme rørledning nærme hverandre vil måle mer eller mindre det samme trykket, eller at differansen mellom de målte trykkverdiene er tilnærmet konstant. I oppsettet i dette prosjektet er dette åpenbart ikke tilfelle.

Det er også ugunstig at analysen ikke går nærmere inn på anvendelsen av måledataene. En av de største fordelene med et datainnsamlingssystem som det beskrevet i oppgaven er muligheten for sanntidsberegninger for deteksjon av for lavt eller høyt trykk i deler av forsyningsnettet. En analyse av et slikt system burde ta slike anvendelser i betraktning, og selv om konseptet i stor grad ble diskutert i oppgaven ble det fortsatt ikke tilstrekkelig testet ut i praksis.

7.4 Erfaringer

Underveis i gjennomføringen av prosjektet oppsto det en del uønskede hendelser som hadde innvirkning på fremgangen i prosjektet. Dette avsnittet har til hensikt å diskutere hvordan slike hendelser kan unngås, noe som kan være nyttig dersom man på et senere tidspunkt skal gjennomføre et lignende prosjekt.

I et prosjekt som krever logging eller annen generering av store mengder med data over lengre tid er det noen forbehold som bør tas. Jo lengre tid data blir logget, desto større er sannsynligheten for at loggingen blir avbrutt på grunn av en svikt i systemet, og desto større

mengde med tid er bortkastet dersom en svikt inntreffer. I dette prosjektet gikk en essensiell del av analysen ut på å logge og tolke data som ble generert fra måleroppsettet. Dataene ble logget over flere dager, og loggingen foregikk til tider uten oppsyn. Loggingen var avhengig av at mange komponenter fungerte mer eller mindre uten avbrudd. For eksempel var det helt avgjørende at PC'en hadde en konstant internett forbindelse. Dersom en TP-kabel ble dratt ut ved et uhell, eller på en eller annen måte internettforbindelsen ble brutt, ville loggingen opphøre. Dette er kun et eksempel på en svikt som ville avbryte loggingen. Et annet eksempel er strømbrudd. I prosjektet skjedde det på et tidspunkt et strømbrudd, noe som åpenbart stanset loggingen. Heldigvis var dataene lagret eksternt, slik at det som var blitt logget ikke ble tapt. Dette er en av fordelene med å lagre dataene til en skytjeneste. Dette vil ikke nødvendigvis være tilfelle for alle systemer. Dersom dataene hadde blitt lagret lokalt ville alt ha blitt tapt. I slike tilfeller bør man foreta backup lagring av dataene med jevne mellomrom. Generelt bør man alltid holde logging under oppsyn slik at dersom det forekommer en svikt så vil det bli oppdaget raskt.

Kapittel 8 – Konklusjon

Hensikten med prosjektarbeidet var å utvikle et datainnsamlingssystem for avlesing av flere simulerte vannmålere, og lagring av måledata til en skytjeneste. Oppgaven var basert på et behov for et system for å overvåke tilstanden til vannforsyningsnettet, av både kommersielle og sikkerhetsmessige årsaker. Det ble foretatt et litteratursøk for å undersøke hvordan overvåking av vannforsyningen har blitt gjort per dags dato. Løsningen som denne oppgaven foreslår involverer at måledataene avleses trådløst av sentrale enheter omtalt som konsentratorer ved bruk av wM-Bus protokollen. Disse konsentratorene vil i sin tur sende dataene videre til lagring i en database. Løsningen ble implementert og testet i henhold til en funksjonell spesifikasjon. Systemet viste i stor grad ønsket oppførsel under testingen, og de fleste kravene i spesifikasjonen ble innfridd. En vesentlig svakhet ved analysen var mangel på ordinære måleinstrumenter. Det ble isteden brukt simulerte målere, noe gav svært begrensede resultater. Testingen av systemet viste imidlertid at konseptet hadde potensial, og det er god grunn til å tro at det burde videreutvikles slik at det blir anvendelig i større skala.

Kapittel 9 – Videre arbeid

Det finnes flere svakheter ved analysen, noe som ble brakt opp i kapittel syv. Noen av disse svakhetene kan potensielt utbedres i senere prosjekter. Det er også flere andre problemstillinger forbundet med overvåking av vannforsyningen som også kan undersøkes grundigere. Dette kapittelet vil gi eksempler på hva som kan utforskes grundigere.

Videre prosessering av måledata

I prosjektet ble det kun gjort enkel behandling av måledata. Det finnes mange anvendelser for trykk og strømningsmålinger som nevnt i kapittel 3. Man kan også undersøke i hvilken grad det er mulig å synkronisere målinger mellom vannmålere i et distrikt, og eventuelt hvordan man kan omgå problemet med synkronisering. Mange interessante problemstillinger er verdt å undersøke med tanke på hva måledata kan brukes til.

Testing med ordinære vannmålere

Løsningen i oppgaven bruker simulerte vannmålere, noe som er en betydelig svakhet. Ved å teste systemet med ordinære vannmålere vil man kunne undersøke i hvilken grad systemet faktisk utfører den oppgaven det var ment å simulere, noe som er en fordel. Det ble i dette prosjektet gjort forsøk på å kommunisere med en slik vannmåler, uten suksess. Videre arbeid kan inkludere feilsøking og debugging av firmwaren, og grundigere analyse av wM-Bus stacken, for å identifisere problemet med kommunikasjonen. I tillegg til å bruke faktiske målere kan man også undersøke ytelsen til systemet når antallet målere øker og sende intervallet reduseres, eller hvordan problemet med rekkevidde kan omgås.

Konfigurasjon av konsentrator fra skytjeneste

I en løsning for fjernavlesning av vannmålere vil det kunne være ønskelig å kunne gjøre så mye som mulig eksternt med tanke på operasjon. Dette er ikke nødvendigvis begrenset til avlesing, men kan også involvere flere former for konfigurasjon, som for eksempel endring

av konsentratorens sendeintervall. Det kan utvikles en webapplikasjon, for eksempel ved å bruke AWS sin Elastic Beanstalk tjeneste, med et brukergrensesnitt hvor forskjellige parametere kan modifiseres etter behov. Det er også mulig at konfigurasjonsdata kan skrives direkte til dynamoDB og lastes ned av konsentratoren med jevne mellomrom.

Referanser

Håkon H. Lier (2017), *Utvikling av et embedded system for avlesing av trådløs M-BUS*.

D. D. Ediriweera, I. W. Marshall (2010), *Monitoring water distribution systems: understanding and managing sensor networks*, Drink. Water Eng. Sci.

Michael Allen, Ami Preis, Mudasser Iqbal, Andrew J. Whittle (2013), *Water Distribution System Monitoring and Decision Support Using a Wireless Sensor Network*, IEEE.

Mompoloki Pule, Abid Yahya, Joseph Chuma (2018), *Wireless sensor networks: A survey on monitoring water quality*, Journal of Applied Research and Technology.

Olivier Hersent, David Boswarthick and Omar Elloumi (2012), *The Internet of Things: Key Applications and Protocols, First Edition*, John Wiley & Sons, Ltd.

introduction-to-wireless-mbus.pdf, Silicon Labs (lastet ned 11.06.18).

http://www.m-bus.com/files/w4b21021.pdf, Prof.Dr.H.Ziegler (lastet ned 11.06.18).

Vedlegg

Her står informasjon om kildekodefiler i prosjektet. Kun filer det har blitt gjort modifikasjoner på er inkludert. All kildekode er tilgjengelig umodifisert fra følgende nettsteder;

Wmbus stack; https://www.silabs.com/products/development-tools/software/wireless-mbus

Libusb; https://sourceforge.net/projects/libusb-win32/files/

Gecko SDK; https://www.silabs.com/products/development-tools/software/simplicity-studio

C-filer

prosjekt\wmbus\source\demo;

main_collector.c;

Modifikasjoner har blitt gjort i følgende funksjoner;

void wmbus_apl_evt_rx(void)

void wmbus_apl_evt_tlgAvailable(E_WMBUS_RX_t e_status, uint8_t c_tlgReqId,

s_apl_tlgAttr_t *ps_tlgAttr)

void main(void)

Følgende funksjoner har blitt lagt til;

void erase_mem(void)

main_meter.c;

Modifikasjoner har blitt gjort i følgende funksjoner;

void wmbus_apl_evt_tx(uint8_t c_tlgId)

bool_t wmbus_apl_evt_userDataRequested(uint8_t c_tlgId, bool_t b_periodical)

void main(void)

Følgende funksjoner har blitt lagt til;

void erase_mem(void)

prosjekt\wmbus\source\oldusb;

callbacks.c;

Følgende funksjoner blir implementert;

void stateChange(USBD_State_TypeDef oldState, USBD_State_TypeDef newState)

int dataSentCallback(USB_Status_TypeDef status, uint32_t xferred, uint32_t remaining)

int requestReceivedCallback(USB_Status_TypeDef status, uint32_t xferred, uint32_t

remaining)

Følgende preprocessor makroer blir implementert;

BUF_TO_ADDR(buf, start)

BUF_TO_KEY(buf, start)

prosjekt\UI\src;

Følgende funksjoner blir implementert;

usb_dev_handle *open_dev(void)

int read_data(int timeout)

int main(void)

H-filer

prosjekt\wmbus\source\oldusb;

callbacks.h

usb_descriptors.h

usbconfig.h

Paths til objektfiler

prosjekt\wmbus\Demo_SLWSTK6220A\collector_s2\Obj

prosjekt\wmbus\Demo_SLWSTK6220A\SLWSTK6220A_Meter_S2\Obj

prosjekt\UI\build

Py-filer

prosjekt\datalogger;

comp.py;

Følgende funksjoner blir implementert;

__init__(self, n)

new_value(self, a)

find_avg(self)

find_max(self)

find_min(self)

logger.py;

Følgende funksjoner blir implementert;

recv_meter_data()

write_to_db()

plotdata.py

plotdata3d.py