Datakorrigering i tid och otid

ANDREAS MÅRDÉN
BACHELOR´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Datakorrigering i tid och otid

Av: Andreas Mårdén.

Handledare: Mats Lilja.

Examinator: Christian Nyberg

LUNDS UNIVERSITET

LTH

LUNDS TEKNISKA HÖGSKOLA

© Copyright Andreas Mårdén.

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2024

Sammanfattning

Klockdrift kan orsaka problem vid datainsamling och negativt påverka kvaliteten på insamlad data. Denna klockdrift är vanligt förekommande i de klockor som används i digitala och självrapporterande sensorer och beror på kvartsoscillatorns kvalitet och konstruktion.
I sammanhang där vanliga synkroniseringsprotokoll inte kan användas krävs en lösning som tillåter att insamlad data korrigeras i efterhand för att ge en tydlig bild av hur ett processförlopp sett ut. Situationer då denna lösning behövs kan vara då sensorers nätverkstillgång inte kan garanteras eller är mycket begränsad. Den föreslagna lösningen är att under en kortare period mäta sensorernas klockdrift mot en gemensam punkt och sedan med denna information korrigera mätningars tidstämplar utifrån den data som insamlats.

Nyckelord: Sensorer, klockdrift, nätverk.

Abstract

Clock skew can cause issues with data collection and the reliability of collected data. This skew is common in the types of clocks used by digital sensors and is influenced by the design of the quartz oscillator within. In situations where regular time synchronization protocols are impractical, there is a need for data to be corrected post-collection to give a clear picture of the process monitored at the time of sampling. Situations like these could arise if the network access of the sensors is unreliable and/or greatly limited. My suggested solution is to measure the skew of these sensor clocks for a limited time against a common relative clock and use this skew to correct the timestamps of the sampled data.

Keywords: Sensors, clock skew, networking.

Förord

Ambitionen i detta examensarbete är att lösa ett mycket nischat problem i en vardaglig arkitektur. Modellen som beskrivs här försöker kringgå problem som kan uppstå vid otillförlitliga nätverksanslutningar och minimera nätverkstrafik på mycket begränsade nätverksanslutningar.

I en tid där allt fler föremål kopplas upp på nätverk i vad som brukar kallas IoT eller "Internet of things" skapas risker för skenande ökning av datamängden om alla dessa skall sända och mottaga statusuppdateringar kring bl.a tidhållning. Majoriteten av denna ökade nätverkskommunikation kommer säkerligen mötas av 5G-nätverk, men för en del tidskänsliga system och applikationer ser vi[1] en ökad användning av anslutningar som kringgår det vanliga nätverket med markant begränsad bandvidd som konsekvens. Exempel på detta kan vara anslutningar via långvågig radio där tekniker och protokoll som LoRa och MQTT förekommer. I dessa situationer kan traditionella protokoll för tidhållning bli problematiska då de baserar sig på jämkning och återkommande tiduppdateringar, där båda bidrar till ökad nätverkstrafik, vilket lösningen som presenteras i detta arbete undviker.

Idén till examensarbetet uppstod i företaget Enersize Advanced Research AB:s möte med otillförlitliga nätverksanslutningar ute i sina kunders fabriker. Företaget arbetar med att analysera och underhålla tryckluftssystem i tillverkningsindustrin, där de använder ett flertal sensorer för att skaffa sig en uppfattning om tryckluftssystemens hälsa. När dessa sensorer dock förlorade anslutningen till företagets centrala server blev ofta den data de insamlade i bästa fall bristande eller i värsta fall helt obrukbar.

Lösningen på problemet blev både en modell och en implementation där båda kommer att beskrivas i detalj nedan.

Innehållsförteckning

1. Inledning. 7
   - 1.1. Bakgrund.
   - 1.2. Syfte.
   - 1.3. Mål.
   - 1.4. Problemformulering.
   - 1.5. Avgränsningar.
   - 1.6. Liknande arbeten.
2. Tekniska förutsättningar och omständigheter. 13
   - 2.1. Kvartsoscillatorer.
   - 2.2. NTP.
   - 2.3. Verktyg som använts.
3. Metod. 15
   - 3.1. Algoritmen.
   - 3.2. Allanvarians.
   - 3.3. Källkritik.
4. Genomförande 22
   - 4.1. Implementationen.
   - 4.2. Användning av implementationen.
5. Analys. 27
   - 5.1. Analys av exempeldata.
   - 5.2. Omständigheter med sensorenheterna.
   - 5.3. Är analysprogrammet användbart?
   - 5.4. Svar på problemformuleringen i 1.4.
6. Slutsats. 32
   - 6.1. Det som återstår.
   - 6.2. Vidareutveckling.
   - 6.3. Etisk reflektion.
7. Terminologi. 34
8. Referenser. 35
   - 8.1. Figurer & tabeller

1. Inledning

1.1 Bakgrund

1.1.1 Klockdrift och dess stabilitet.

Moderna digitala sensorer innehåller klockor för att kunna sända sina mätningar med jämna mellanrum och för att tidsstämpla när en mätning togs. Dessa klockor, likt de flesta andra, är baserade på kvartskristaller och inverterad piezoelektricitet[2]. Ingen klocka av denna typ är perfekt och alla har en viss grad av fel som i de flesta sammanhang är så litet att det inte orsakar några problem och som kan åtgärdas genom att tiden justeras då och då av användaren. I industriella system kan dock detta fel vara tillräcklig för att försvåra analyser av systemets hälsa.

En klockas korrekthet består av två delar: dess precision och dess stabilitet.
En klockas precision är dess förmåga att hålla rätt tempo, det vill säga huruvida kvartskristallen svänger rätt antal gånger per sekund eller ej. Vi mäter precision i $\Delta t/t$ eller $\Delta f/f$, beroende på om vi vill mäta i tidsenheter eller i hertz.[7] Slutresultatet blir detsamma i form av en dimensionslös variabel som kan användas för att jämföra olika oscillatorer. En förväntad frekvens kan exempelvis vara 32 528 Hz, men om frekvensen istället är 32 525 Hz så har klockan ett absolutfel på 3 Hz. Vi använder då $\Delta f/f$, vilket skulle ge: 3 Hz/ 32 528 Hz $\approx$ $9{,}223 * 10^{-5}$.

Räknemetod	Ingångsvärden	Precision
$\Delta t/t$	1 / 10 800	$9{,}259 * 10^{-5}$
$\Delta f/f$	3 / 32 528	$9{,}223 * 10^{-5}$

Tabell 1: Räkneexempel på precision.

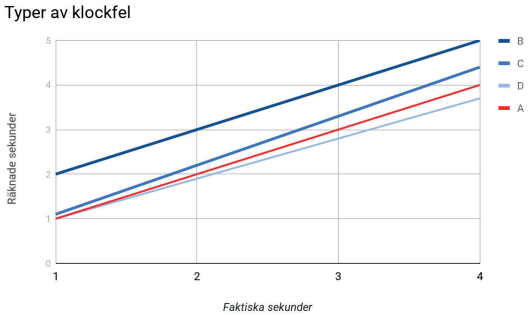
Vi ser att klockan förlorar nästan en sekund var tredje timme och räknar man vidare så ser man att det är 0,996 sekunder.

Stabilitet är istället kvartskristallens förmåga att bibehålla samma precision över tid. Om den tidigare nämnda kvartskristallen med en precision inom 3 Hz alltid tappar 0,996 sekunder var tredje timme så anses den vara stabil, om än oprecis. Om det däremot varierar hur många sekunder som vinnes eller förloras under en given period så har kvartskristallen lägre stabilitet.

Tillsammans bildar dessa två (precision och stabilitet) det fel som alla kvartsoscillatorer har till någon grad[2]. Dessa värden påverkas av oscillatorns ålder, material, konstruktion, nuvarande temperatur men också av mer svårmätta parametrar såsom luftfuktighet, lufttryck, matningsströmmens styrka och strålningspåverkan.[11] Precisionen hos kvartsoscillatorer brukar

anges i PPM, (eng. parts per million), alltså hur många miljontedelar fel de som mest förväntas ha per sekund.

En klocka kan på olika vis vara i otakt med tiden:
Exempel 1: Klocka A är perfekt i tid och används som referens. B är felställd men har ett konstant fel i förhållande till A. C går för snabbt och får ett ökande fel i förhållande till A. D går för långsamt och får på samma sätt ett ökande fel i förhållande till A.



Figur 1: Exempel på olika typer av klockfel.
Om A håller tiden perfekt så har C en positiv drift, D en negativ drift och B saknar drift men är felställd.

Grafen ovan ger dock en förenklad bild då den antyder att klockdriften skulle vara helt linjär i förhållande till realtiden, något som sällan är fallet. De tidigare nämnda faktorerna såsom ålder, temperatur m.m (se ovan) samverkar och ger upphov till en oregelbunden variation, även om varje faktor i sig kanske är periodisk. I arbetet kallas detta för driftens variation.

Man kan likna det vid ett reglersystem, där flertalet effekter samverkar och motverkar varandra, med olika styrka vid olika tidpunkter. Detta ger att en ögonblicksbild av driften uppmätt vid ett enskilt tillfälle inte nödvändigtvis är tillräckligt för att ge helhetsbilden av driften.

1.1.2 Drift som problem i industriella system.

Företaget Enersize utvecklar produkter för att analysera och underhålla tryckluftsystem som används i tillverkningsindustrin. Deras produkt används för att identifiera läckor och minimera användning av systemens kompressorer. Dessa tryckluftssystem är ofta mycket dyra i drift[3] då kompressorer i nästan alla fall har mycket hög strömförbrukning för den effekt de levererar.

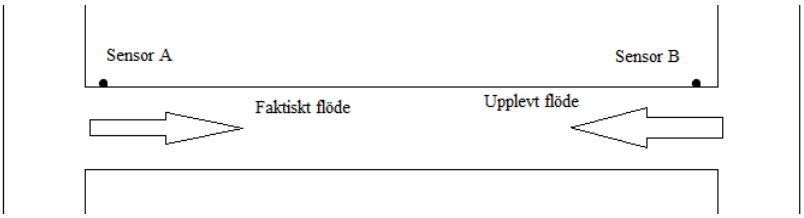
För att kunna göra de analyser som krävs behöver företaget tillförlitlig data insamlad över lång tid från ett stort antal sensorer. Om en sensor under ett par dagar saknar nätverksanslutning och

därför inte kan motta tidsuppdateringar eller sända mätdata, så kommer mätningarnas tidsstämplar bli allt mer avvikande på grund av klockdrift.

Problemet som då uppstår är att om data från flera sensorer skall användas för att förstå vad som skett i ett system under en given tid, och dessa sensorers klockor drivit i olika riktningar så ser det ut som att mätvärden insamlats i en annan kronologisk ordning än vad som faktiskt är fallet.

Den kumulativa tidsförskjutningen mellan sensorernas klockor kan sålunda förhindra upptäckten av samband när datan analyseras. Ett exempel skulle kunna vara att man inte kan utläsa tiden det tar för tryck att propagera över en given distans, eller relationen mellan luftflödet och kompressorns energiförbrukning. Är driften tillräckligt stor och olika klockor har drivit åt olika håll så kan det se ut som att händelser i ett system har inträffat i en annan ordning än de faktiskt gjorde.

Exempel 2:



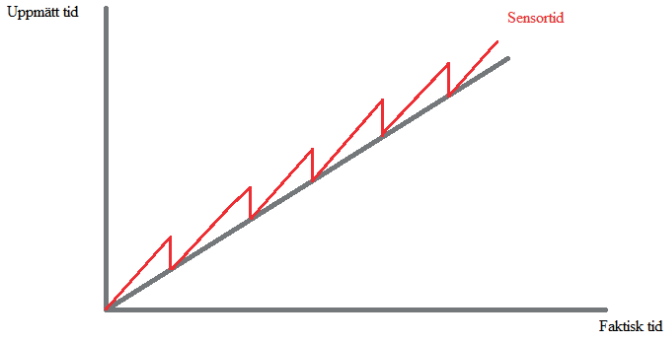
Figur 2: Ett exempel på hur drift kan påverka datas kvalitet.

Figur 2 ovan visar hur två mätare längs ett rör registrerar en tryckökning, först vid punkten A, sedan vid punkten B, vilket visar att flödet ökade i riktning från A till B. Om nu sensorklockorna för dessa har drivit åt olika håll så kan det se ut som att trycket först uppkom vid B för att sedan propagera till A, vilket skulle antyda att flödet ökade i motsatt riktning, eller att det tog orimligt lång tid för trycket att propageras genom rören, vilket kan antyda en läcka.

Redan existerande lösningar[4][5][6] på problemet finns i form av olika protokoll där sensorernas tidshållning justeras med jämna mellanrum, antingen gentemot en central server eller att de jämkar sinsemellan. Nackdelen med dessa lösningar är att de är beroende av kontinuerlig nätverksanslutning och genererar mer nätverkstrafik än vid normalfallet, något som i företagets Enersize system inte alltid kan garanteras eller tillåtas.

Vidare kan lösningar där klockor återkommande korrigeras ge upphov till större osäkerhet då det är okänt om korrigering skett nyligen eller ej. Figur 3 visar ett exempel på hur dessa korrigeringar kan ge upphov till “hack” i tidshållningen. Två prov som tages nära varandra

riskerar att falla på var sin sida om korrigeringen och skapa en situation där det andra provets tidsstämpel ligger innan den förra.



Figur 3: Ett exempel på hur återkommande tidsuppdateringar kan skapa större osäkerhet.

1.2 Syfte

I detta examensarbete beskrivs en metod för att lösa problemen orsakade av klockdrift vid datainsamling. Den erbjuder ett sätt att hantera denna drift som tillåter insamlad sensordata att vara tillförlitlig även om nätverksanslutning inte kunnat garanteras under insamlingen. Detta skall göras utan att vara beroende av konstant nätverksanslutning och utan att öka mängden kommunikation som krävs över nätverket.

Examensarbetet syftar till att förbättra möjligheten att tidsbestämma data i situationer där vanliga synkroniseringsprotokoll är otillförlitliga eller olämpliga. Därför utvecklades en modell som korrigerar data i efterhand snarare än något nytt protokoll för tidsuppdatering.

Slutresultatet av examensarbetet innebär att företag såsom Enersize på ett mer noggrant sätt skulle kunna identifiera problem och brister i sina kunders tryckluftssystem och därigenom minska sina kunders energiförbrukning.

1.3 Mål

Arbetets målsättning har varit att skapa en metod och ett program som kan identifiera och dokumentera precisionen hos en given sensorklocka och skatta dess stabilitet. Programmet skall genom dataanalys leta efter förskjutningar i tidsangivelserna och därigenom hitta sensorklockans avvikelse från realtid. Samtidigt ska programmet skatta stabiliteten för avvikelsen; detta är nödvändigt då driften inte nödvändigtvis är konstant.

Metoden jämför tiden då data anländer till en central server med den tidsstämpel sensorn rapporterat som samplingstid. Man kan då, över loppet av ett stort antal mätningar, se hur förhållandet i tid mellan de två förändras. Är förhållandet för de två tiderna konstant så är enheten i takt med realtiden, är förhållandet växande eller sjunkande så är enheten i otakt med realtiden, och varierar förhållandet mellan de två så är enheten i varierande otakt med realtiden. Detta kräver naturligtvis att sensorerna avser att mäta och leverera data regelbundet och inte vid slumpmässiga intervaller.

Modellens resultat blir dels en tabell där varje enhet är listad med värden för senaste synkronisering, drift och största uppmätta fel, och dels en korrigering av sensordata för den uppmätta driften. Om dessa resultat kan användas för att korrekt förutsäga antalet sekunder en analyserad sensorklocka kommer att räkna för en viss given realtid, får metoden och programmets implementation anses vara lyckad.

1.4 Problemformulering

Examensarbetet ämnar att besvara följande frågor:

- Kan sensordata synkroniseras utifrån empirisk analys av sensorernas klockfrekvens?
- I vilken grad påverkas sensorernas tidshållning i märkbar grad av sin omgivning avseende strömförsörjning?
- I vilken upplösning behöver dataanalysen ske för att klockornas drift skall vara mätbar?
- I vilken grad påverkar nätverksförseningar tidsstämplarnas tillförlitlighet?

1.5 Avgränsningar:

1.5.1

Något som inte har studerats är korrektheten hos systemklockan i den mottagande enheten. Systemklockans klockdrift kan antingen förminska eller överdriva sensorernas dito.

1.5.2

Tid antas i sammanhanget vara kontinuerlig och något som existerar självständigt från omvärlden. Ingen som helst hänsyn tas till att uppmätt tid kan variera.[11]

1.5.3

När bedömningen av en sensor är klar så kan data korrigeras oavsett sensorns nätverksuppkoppling, under själva analysen krävs dock en stabil anslutning. Tiden som analysen tar varierar med olika sensorer men bör ges åtminstone ett par dagar. Den långa analystiden beror på att variationer i driften kan ha perioder på flera dygn. Om alltför få av dessa perioder insamlats så kommer variationen i alltför hög grad att påverka analysens slutvärden.

1.5.4

Implementationen saknar grafiskt användargränssnitt.

1.6 Liknande arbeten:

Ett arbete som med hjälp av driftanalys identifierar nätverksnoder:
“Clock skew based node identification in wireless sensor networks”
Ding-Jie Huang, Wei-Chung Teng, Chih-Yuan Wang, Hsuan-Yu Huang och Joseph M. Hellerstein. National Taiwan University of Science and Technology, Taipei, Taiwan. 2008.[12]

Ett arbete som undersöker temperaturers påverkan på drift, och föreslår förbättringar av traditionella protokoll:
“Temperature-Compensated Clock Skew Adjustment.”
Jose María Castillo-Secilla, Jose Manuel Palomares och Joaquín Olivares.
University of Cordoba, 2013.[13]

Ett arbete som presenterar en alternativ metod för synkronisering och driftkorrigering där man strävar efter bland annat mindre nätverkskommunikation i processen:
“RTSP: An Accurate and Energy-Efficient Protocol for Clock Synchronization in WSNs.”
Muhammad Akhlaq och Tarek R. Sheltami. IEEE, 2013.[14]

Det finns gott om studier i ämnet, dessa tre är exempel som var och en motsvarar något specifikt i arbetet: Individuell drift, påverkan av sensorns omständigheter och strävan efter att minska nätverksbelastning.

2. Tekniska förutsättningar och omständigheter

2.1 Kvartsoscillatorer[2][7]

Grunden för de flesta moderna klockor är kvartsoscillatorn och piezoelektricitet; när en kvartskristall utsätts för elektrisk spänning deformeras den och kan på så vis användas som en tillförlitlig oscillator. En omgivande mekanism räknar sedan antalet svängningar och sänder en impuls till resten av systemet om att en sekund har passerat när rätt antal svängningar har räknats. Ofta är denna räknare binär med ett maxvärde på 2^{15}, dvs 32'767, som vid överslag sänder klockpulsen.

I designen finns flera källor till att beräkningen av en sekund kanske inte blir exakt: kvartskristallens ålder, dess skärning, strömförsörjningen både till räknaren och kristallen, temperatur, lufttryck är några av de saker som har en effekt. Det finns ett antal olika lösningar på dessa problem, bland annat att hålla kristallen jämnt tempererad eller användning av självkorrigering efter uppmätt temperatur och lufttryck, men återkommande för dessa lösningar är att de är ökar klockans kostnad och de används därför sällan till mer än de mest känsliga sammanhang såsom i rymd-, forsknings-, eller militära sammanhang.

För mer vardagliga användningar brukar man mäta precisionen i "PPM", vilket står för "Parts Per Million", där värden kring 5ppm ($5x10^{-6}$) till 100ppm (10^{-4}) är att betrakta som normala. Om en drift är stabil så är en tidsskillnad på 0.07- till 0.09 sekunder per dag att betrakta som normalt. Då sensorerna på industriella system mäter förändringar som kan propagera genom systemet på markant mindre tid än fulla sekunder så är det tydligt varför drift av denna grad kan vara ett problem.

2.2 NTP[8][9]

Hårdvaran som i utvecklingen av modellen har använts som tidsmätningsserver använder i sin tur "Network Time Protocol", för att korrigera sin klocka och därigenom motarbeta drift. Detta är ett av de tidigare nämnda tidsutbytesprotokollen som är en vanlig lösning för klockdriftsproblematik. NTP används i detta fall av systemprocessen "timedatectl" för att återkommande försäkra sig om att systemtiden är korrekt, denna kontroll och korrigering genomförs efter 32 sekunder som minst till 34 minuter och 8 sekunder (d.v.s 2048 sekunder) som mest. NTP är ett komplext protokoll som i flera steg beräknar vad systemklockan bör vara där den tar i beaktande faktorer såsom anslutningens propageringstid bland annat. Vissa effekter kan vara värda att påpeka: bristande tillförlitlighet hos systemklockan och användande av NTP kan i värsta fall påverka modellens datainsamling och beräkningar i fall där slutsatser dras på små

datamängder. Detta sker om man återkommande justerar tiden i ett system där mätningar kontinuerligt pågår då det inte går att ur datan utläsa om den interna tiden justerats.

Exempel:
Om en första mätning togs vid tiden 02:00 och en andra mätning togs vid 03:00 så verkar det rimligt att en timme förflöt mellan dessa. Om dock klockan justerades med en timme, exempelvis på grund av byte mellan normaltid och sommartid, så är detta ett fel som är mycket svårt att upptäcka i efterhand. Samma sak sker i minuter, sekunder eller ännu mindre tidsenheter förekommande i IT-system.

Det är därför viktigt att man i användandet av modellen tillåter en tillräckligt stor datamängd att insamlas, så att skillnaden mellan mätningarnas tidsperiod och dessa justeringar blir mycket stor. Implementationen varnar därför om datamängden som bearbetas riskerar att vara för liten för att resultatet skall bli tillförlitligt.

2.3 Verktyg som använts

C++: Ett programmeringsspråk som utvecklades under 1980-talet som en vidareutbyggnad av C med möjligheter till objektorienterad programmering. Programmen i arbetet kompilerades med C++17.

G++/Gmake: En kompilator för C++ och dess instruktioner.

MySQL: Open source Structured Query Language för rationella databaser. Termen används både för språket i sig och de databaser språket är avsett att användas på.

Python: Ett dynamiskt typat programmeringsspråk. Språket utvecklades också under 1980-talet och kom i bruk efter 1990.

Flask: Ett Pythonramverk för att skapa enkla servrar. Flaskprojektet lanserades under 10-talet och erbjuder möjligheten att sätta upp enkla webservrar som erbjuder HTTP för ett begränsat antal anslutningar.

GMU191: Ionsigns datainsamlingsplatform. Då sensorerna i arbetet har varit analoga så har denna datainsamlingsplatform konverterat infomationen till digital form och skapat mätpunkterna. Det är dessas kvartsoscillatorer som mätningarna har utförts på.

3. Metod

En modell och implementation har skapats som insamlar sensordata som rapporteras via nätverk, beräknar och dokumenterar dessa sensorers fel, variation m.m och som sedan korrigerar insamlad data från dessa sensorer utefter den skapade tabellen.

Beräkningarna nedan förväntas ske på storleksintervallet mikrosekunder eller tiondels mikrosekunder (10^{-6} eller 10^{-7}). Dock anges datapunkterna på nanosekundsnivå, men då utrustningen som använts inte har specifikt utformats eller valts för sin exakthet i den upplösningen så betraktas värden på tiotals nanosekunder eller nanosekunder (10^{-8} eller 10^{-9}) som otillförlitliga även om de förekommer.

3.1 Algoritmen

Modellen börjar med en mängd av indata på formatet {$Servertid_n$, $Sensortid_n$} där $Servertid_n$ är tiden när en mottagare mottog en sensorrapport "n" och $Sensortid_n$ tiden då sändaren av rapporten "n" anser att densamma sänts (hädanefter benämns dessa som "Ser_n" och "Sen_n").

En ny mängd, C_n, skapas från den ursprungliga genom $C_n = Ser_n - Sen_n$. Resultatet av beräkningen är tiden i realtid som passerat sedan 01-01-2000 klockan 00:00:00 till mängdens första prov togs, där "n" är ett godtyckligt värde där $0 \leq n < N$. Dessa värden är inte av särskilt stort intresse var för sig, men det som är relevant är att C_n inte är konstant. Det är denna variation som är värd att ta till vara då detta är driftsumman för varje provtagning: $D_n = C_{n+1} - C_n$. Detta ger oss en mängd med förändringen i sekunder per sekund per provtagning. Denna variation är klockans stabilitet, eller snarast frånvaro av stabilitet.

Det vi här har gjort är snarlikt Allanvarians. Se mer om Allanvarians i 3.2.

Tabell 2: Visualisering av värdena Ser-D med de första fem värdena i en insamlad talserie om drygt 14 000 värden. I raderna Ser och C är de inledande siffrorna "6196190" trunkerade för läslighetens skull.

Ser_n	73,60714	78,60667	83,60742	88,62723	93,60662
Sen_n	4	9	14	19	24
C_n	69,60714	69,60667	69,60742	69,62723	69,60662
D_n	N/A	-0,00047	+0,00075	+0,00198	-0,00206

D_n ovan är antalet sekunder extra som passerat sedan den senaste mätningen. Detta är klockdriftens summa per mätning, antalet sekunder som sensorklockan räknat extra eller för lite

under ett rapportintervall. I exemplet ovan så är rapportintervallet fem sekunder, där man ser att även om sensorn räknade fem sekunder mellan prov 1 och 2, så förflöt 5 - 0,0047 faktiska sekunder, och mellan prov 2 och 3 förflöt 5 + 0,0075 sekunder. Om man dividerar driftssumman per prov med tidsintervallet får vi driften per sekund mellan dessa två prov, -0,00094 och 0,00015 respektive.

Rapportintervallet kan, om det är känt, anges som inparameter till modellen, annars kan rapportintervallet skattas via $\lceil (Sen_{N-1} - Sen_0) / N \rceil$ där "N" är antalet uppmätta datavärden och "⌈⌉" är avrundning uppåt. Naturligtvis kräver detta att sensorn faktiskt anser sig sända rapporter med jämna intervaller.

Gör man nedslag enligt tabell 2 förväntas det att en klocka har både positiva och negativa fel, men med en större tendens åt det ena eller andra hållet. Det är denna tendens som är klockans drift; en klocka med positiv drift kan mycket väl ha en negativ drift vid enskilda mätpunkter, det är vilken sida om korrekt tid som är övervägande som avgör huruvida klockan har positiv eller negativ drift.

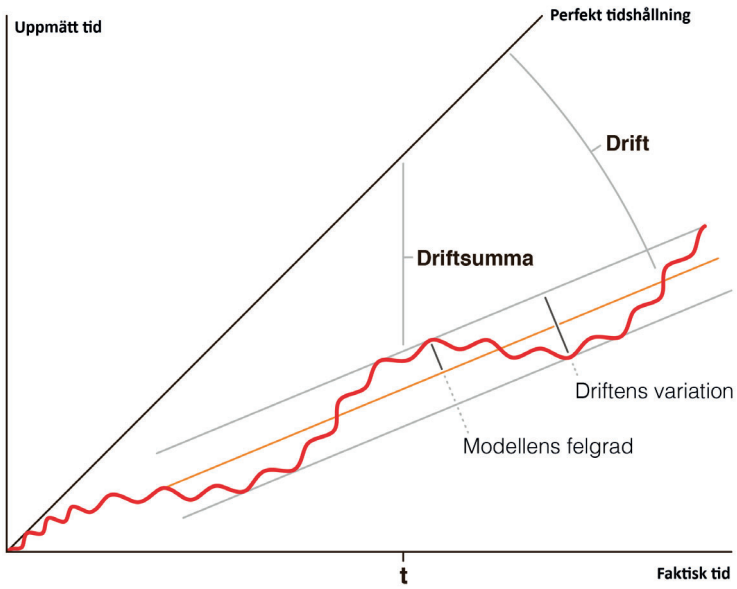
Efter att differensen D_n tagits fram beräknas den totala driftsumman för mängden och den största positiva och negativa variationen. Driftsumman för hela mängden är skillnaden mellan vårt första och sista värde $D_{summa} = C_n - C_1$. Dock så itereras och summeras också mängden av D-värden för att spara undan de största positiva och negativa värdena. Som i exemplet ovan är $D_{max} = +0{,}01981$ och $D_{min} = -0{,}02061$, dessa används sedan som referenser för att bedöma om driftsumman är tillräcklig för att få ett användbart slutresultat. Om den ligger inom intervallet mellan dessa extremvärden så anses driften för låg eller variationerna (jitter) för kraftiga för att någon drift skall vara möjlig att uttyda. Detta löses enklast genom att utöka mängden datapunkter, då tiden som förskjutits totalt behöver bli mer än någon enskild variation mellan två datapunkter. Driften per sekund tas sedan fram genom att dividera den totala driftsumman med antalet mätpunkter multiplicerat med intervallet som avrundas uppåt:
$D_{/S} = D_{summa} / (N * \lceil (Sen_{N-1} - Sen_0) / N \rceil)$.

Avslutningsvis beräknas ett intervall för klockans stabilitet. Detta behövs då modellen hittills behandlar klockdriften som om den vore helt linjär och inga störningar förekommer under mätningarna, men dels varierar driften och dels finns det en risk att exempelvis nätverksfördröjningar skapar brus i beräkningarna. Intervallet som beräknas är stabilitetens amplitud. Beräkningen för detta sker i flera steg: Först beräknas vad sensorklockan borde ha angivit för tid T, vid en given mätpunkt n_x, om driften varit helt linjär:
$T = Sen_0 + n_x * I + n_x * I * D_{/S}$

T är tiden sensorklockan bör visa om driften beräknats korrekt med hjälp av:

Sen_0: Sensortiden för mängdens första element.
n_x: Nuvarande provpunkts plats i ordningen.
I: Intervallet mellan provtagningar = $\lceil (Sen_{N-1} - Sen_0) / N \rceil$.
$D_{/S}$: Driften per sekund.

Så om driften har beräknats korrekt och sensorklockans drift är helt linjär så bör sensortiden vid punkten n_x vara T.



Figur 4: En visualisering av begreppen, där orange visar modellens beräkning av driften och rött klockans faktiska drift. Modellens felgrad ges av avståndet mellan orange och röd linje och driftens variation är spannet inom vilket driften varierar.

Som tidigare påpekats så är inte driften helt linjär; så vid jämförelse av T och Sen_x så kommer de sällan vara lika. Felet dem emellan måste dock divideras med tiden som gått från mätningarnas start och n_x, detta då man annars får den driftsumma som ansamlats från mängdens start till det nuvarande provet. Det vill säga: man får antalet sekunder som provet avviker från modellen, inte antalet sekunder per sekund som provet avviker från modellen. Antalet sekunder per sekund som modellen har fel om en given punkt får man då istället via: $T_{fel} / (I * n_x) = D_{fel}$. Detta kommer här att refereras till som felgrad.

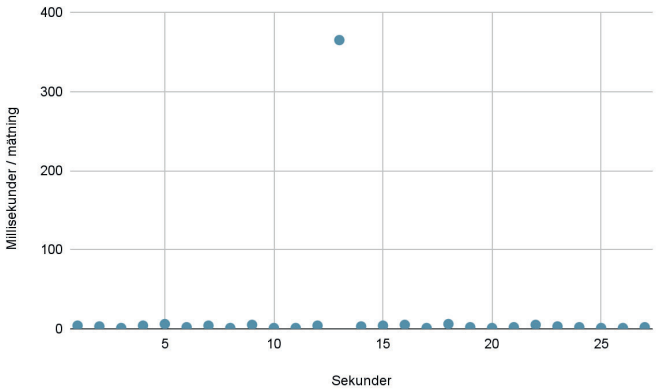
Exempel: Om en klocka som testas räknar 10 002 sekunder under en viss tid, men beräkningarna som utförts på klockan ger att den bör ha räknat 10 001 sekunder, så har modellen en felgrad på 0,0001 för just den mätpunkten.

Det är normalt och förväntat att modellen skall ha en felgrad för de flesta mätpunkterna då modellen baserar sig på ett genomsnitt, men också då storleken på driften varierar. Se figur 4.

Ett medelvärde M beräknas sedan av alla felgrader, D_{fel},: $M = (D_{fel\ 1} + \ldots + D_{fel\ N-1})/N$. Där N är mängdens storlek.

Detta medelvärde behövs som referens för att identifiera det brus som kan orsakas av nätverksfördröjningar. Med medelvärdet går det att filtrera bort extremvärden som klockan som analyseras omöjligt kan vara ansvarig för.

Förväntningen var att hitta stora felgrader orsakade av nätverksfördröjningar som skulle bli markant större än de felgrader som uppkommer av driftens variation, som i sin tur exempelvis orsakas av temperaturförändringar i sensorns oscillator. Därför skapades en filtrering (eller tröskelsättning) för att undvika att beräkningarna inkluderade felaktiga extremvärden såsom exemplet i Figur 5.



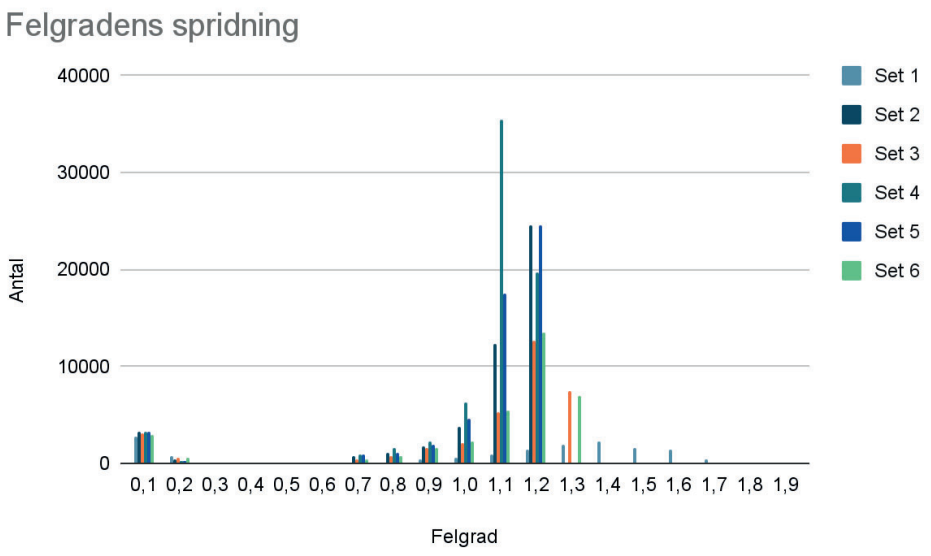
Figur 5: Ett exempel på problematiken filtreringen skulle åtgärda.
I beräkningen av medelvärdet så skulle extremvärdet filtreras.

Dessa nätverksfördröjningar väntades kraftigt påverka det beräknade medelvärdet med extremvärden orelaterade till kvartsoscillatorn. För att undvika detta så räknades medelvärdet om där alla värden som överstiger det tidigare medelvärdet med 2,5 gånger uteslöts.
$(D_{fel\ n} < M*2{,}5 \rightarrow D_{fel\ n} = D_{fel\ n}) \wedge (D_{fel\ n} \geq M*2{,}5 \rightarrow D_{fel\ n} = 0)$.

Just “2,5” valdes som en första filtrering, utifrån att ha noterat att normala variationer låg inom det spannet, med avsikten att sedan korrigeras allteftersom man kunde se hur många mätpunkter som låg utanför intervallet. I och med den stora skillnaden mellan värden som kan skapas av

nätverksfördröjningar och variationer hos en oscillator kunde man tänka sig variationer så höga som 100 gånger medelvärdet.

Filtreringen kom aldrig till användning, då några så extrema outliers inte förekom i de insamlade datamängderna. Figur 6 visar hur datamängdernas mätpunkter är fördelade kring driftens variationsgenomsnitt utan någon filtrering. Den horisontella axeln är olika multiplar av medelvärdet, där filtreringen skulle avlägsna alla datapunkter som var minst 2,5 gånger så stora. Som kan ses i figuren förekommer inga värden över 1,7 gånger medelvärdet, vilket innebär att inga tydliga effekter av nätverksfördröjningar kan ses i variationerna. Ansamlingen kring 1,0 är väntad då det representerar det beräknade medelvärdet.



Figur 6: Felgradens spridning från medelvärdet, här representerat som 1,0.
Den ansamling som syns vid 0,1 är värden som har en avvikelse från medelvärdet på nästintill 0.

Grafen ovan har ett fel; ansamlingen vid 0,1 beror på att det egentligen bör stå "≤0,1", vilket innebär att alla mätpunkter som har en felgrad på 0,1 eller mindre hamnar på den platsen. Den mindre ansamlingen vid 0,2 kan tolkas som att det skulle bli en förhållandevis jämn normalfördelning kring 0,1 om det fanns fler alternativ därinunder.

Det finns fler frågor som uppkommer när man studerar grafen: Varför har vi en tydlig normalfördelning kring 1 men tomrum mellan 0,2 och 0,7? Varför finns inga värden högre än 1,2 för set 2, 4 och 5? Varför ansamlas just set 4 så kraftigt vid 1,1? Alla är intressanta, men då

avsikten med att beräkna ett intervall på driftens variation var att hitta och kunna exkludera mätningar där exempelvis nätverksfördröjningar, paketförluster eller nätverksstockning kraftigt påverkat mätvärdena så har valet blivit att konstatera att inga sådana fall verkar finnas i exempeldatan. Då mätdatan har insamlats med enbart en nätverksnod mellan server och sensorer så är det heller inte så överraskande.

3.2 Allanvarians.[15]

En snarlik beräkningsmetod är Allanvarians, en algoritm för att beräkna en klockas eller oscillators stabilitet över tid. Algoritmen utvecklades av David W. Allan under 1960-talet och beräknas genom att ta mätningar av objektet som undersöks och ställa dem i relation till en betrodd yttre källa, dvs jämföra vår testklocka med en referensklocka som vi förutsätter går exakt rätt. Metoden går ut på att betrakta hur skillnaden mellan två på varandra följande stickprov förändras över tid. Allanvarians beskrivs som:

$$\sigma^2_E(T) = \frac{1}{2(N-1)} \sum_{i=1}^{N-1} (E_{i+1} - E_i)^2$$

där E är en mängd mätningar, N är antalet mätningar, T är mätperioden och i representerar index.[7]

Detta kan ses i relation till statistikens vanliga varians där skillnaden tas från varje mätvärde i förhållande till mängdens genomsnitt, istället för den direkt föregående mätpunkten.

Vidare så görs här skattning av stabilitet för en klocka med Allanavvikelse $\sigma_E(T)$, kvadratroten ur Allanvariansen, som ger ett värde på stabiliteten i systemet som testas.

I en väl kontrollerad miljö kan en mätpunkt tas då en referensklocka anser att en given tid har passerat. Något som inte låter sig göras i arbetets givna förutsättningar, skillnaden ligger i att mätningarna i det här arbetet inkommer när klockan som mäts anser att fem sekunder har passerat sedan den senaste sändningen. Detta gör att mätningar inte konsekvent tas med jämna intervall enligt referensklockan, vilket är en förutsättning för att använda Allanvariansens algortim.

Med andra ord, en central skillnad mellan Allanvariansen och metoden här, samt anledningen till att Allanvariansen inte används, är att mätning i Allanvariansen sker när referensklockan anser att en viss tid har passerat. Mätning i denna metod sker när klockan under test anser att en viss tid har passerat, vilket tillåter att intervallen mellan mätpunkterna varierar.
Värden likt Allanavvikelsen, som används för att jämföra klockor, blir inte heller aktuella för arbetet då arbetets syfte är förbättrad datakvalitet genom att korrigera data med hjälp av analyserad klockdrift, inte att jämföra klockor med varandra.

3.3 Källkritik.

Vissa källor som används i arbetet är skapade av vinstdrivande organisationer eller med ekonomiska motiv. Detta är framförallt referenserna 1, 3 och 10. De används i arbetet för att skapa en kontext av problematiken som arbetet söker möta. Referens 1 ger kontext kring problematiken med begränsade eller otillförlitliga nätverksanslutningar, referens 3 ger kontext hur sensorer används i industriella system, och referens 10 är för att beskriva den typ av hårdvara som användes i arbetet.

Referenserna 8 och 9 är publicerade av Network Time Foundation - ett opensourceprojekt som ligger till grund för tidhantering i nästan alla delar av modern IT-infrastruktur. Referenserna beskriver NTP (Network Time Protocol) och hur det hanterar tid.

Referens 4 - 6 samt 11 - 12 är akademiska publikationer som inte bedöms ha någon anledning att vara avsiktligt vilseledande. Referenserna 2, 7, 14 -15 är istället publicerade av Förenta staternas institut för standarder och teknologi (NIST) eller av IEEE. De är standardiseringsorganisationer som inte bör behöva någon vidare presentation.

4. Genomförande

4.1 Implementationen:

Utvecklingen har skett i C++ med vissa stödjande funktioner i Python och MySQL. Programmen som utvecklats har inga grafiska gränssnitt men använder standard-out och har någon enstaka läsning från input. De exekveras med fördel antingen som ett CRON-jobb eller i en terminal. Programmen bör inte vara känsliga för vilken plattform de exekveras på, men har inte testats på något annat än Debian-derivatet Lubuntu.

Programmet som beräknar och korrigerar klockdriften är litet och omfattar mindre än 500 rader C++, men utöver detta innehåller projektet diverse pythonskript, SQL och make. Insamlandet av testdata gjordes via "GMU 191"-enheter av Ionsign[10] kopplade till en analog trycksensor. Skall man vara strikt korrekt så är GMU 191 inte en sensor, men erbjuder samma typ av sampling och kommunikationsmöjligheter för analoga sensorer som en digital sensor hade gjort. Dessas leveranser av data mottogs av en webserver baserat på Pythonramverket "Flask", som i sin tur inordnade relevant data i en MySQL-tabell. Flaskservern lyssnar på port 8000 efter GET-begäran av hostname/loggerdata/. I denna GET inkluderas en rapport om insamlad data och en tidstämpel för när rapporten sändes. Detta sparas i tabellen sensordata.

Tabellen sensordata består av fyra kolumner: Nyckeln "ID" av typen int, "sensor_id" av typen varchar, "arrival_time" och "report_time" av typen decimal. Ur analysprogrammets perspektiv så behövs enbart de två sista för att analys och justering skall kunna göras.

Det rekommenderas att man vid användande av programmet själv implementerar en datainsamlingsmetod som passar den arkitektur man har.

Huvudprogrammet är den exekverbara filen "Analysis" och består av klasserna: Analysis-main, Analysis, DBconnection och den fria funktionen DataCorrection.

4.1.1 Klassen Analysis-main.cc

Mainklassen har ansvar för att skapa instanser av de övriga klasserna och hantera allt dataflöde där emellan. Klassen har tre hjälpfunktioner:

- Read_config, som läser in anslutningsinformation till databasen från config.ini.
- Commit_results, som kontrollerar om analysen lyckades och om så, sänder resultaten vidare till databasanslutningen.
- Fix_dataset, som om ett visst nollställningsfel upptäckts i Sensordata, kan korrigera för detta.

Även om de två första är uppenbara i sin funktion så kan den tredje kräva en förklaring: Funktionen existerar för att hantera en egenskap hos den sensorenhet som Enersize använder i sina system och som har använts under utvecklingen: Enheten gör en ansats att korrigera sin klockdrift var sjätte timme genom att begära en tidsstämpel från servern. Detta är något man vill undvika då dessa ansatser inte är särdeles noggranna och kan ge upphov till större osäkerhet i provernas tidstämplar (se avsnitt 1.1.2 och Figur 3 om varför). För att undvika dessa korrigeringar så ges enheten inget svar vid förfrågan på tidsstämpel, vilket gör att enheten nollställer sin interna räknare efter dessa sex timmar (18 000 sekunder). Detta är en alltför kort tid för att en tydlig klockdrift skall kunna konstateras, varför funktionen finns för att beräkna om rapporternas tidsstämpel i databasen som om denna nollställning inte skedde.

Alla dessa funktioner är menade att omforma data till en lista av decimaltalspar (en sk "map") och exekvera analysklassen med denna som argument.

4.1.2 Klassen Analysis.h & .cc

Analysklassen är ryggraden av programmet och består av den publika funktionen `Run` och de privata funktionerna Difference, Derive, Calculate_skew, Avg_variation, Compose_return och Interval_correction.

4.1.2.1 Datatyper:

Analysklassen använder en användardefinierad datatyp, SkewResults, vilken existerar enbart för att kunna returnera flera värden utan globala variabler, sk "getters" eller utparametrar. Datatypen är en struct med medlemsvariablerna skew, skewsum, maxVariation och status. Alla dessa värden sätts i funktionen Calculate skew. Det kan nämnas att "skew" är klockdriften och "skewsum" är summan som av klockdriften adderats till tidsstämpeln vid någon given punkt. Mer om det senare.

4.1.2.2 Funktionen Difference:

Funktionen itererar över en parlista där varje par består av sensortiden (Sen) för en rapport och servertiden (Ser) då rapporten mottogs. Sensortiden subtraheras från servertiden för att skapa en vektor där, i fallet med en hypotetiskt perfekt sensor, alla värden (C_n) kommer att vara det ögonblick sensorn började insamla data. För alla faktiska sensorer kommer dock variationer att finnas mellan vektorns värden, något nästa metod Derive utnyttjar.

Exempel från ett test under utvecklingen (värdena i rad ett och tre har trunkerats):

Tabell 3: Exempel på insamlad data.

Ser_n	73,60714	78,60667	83,60742	88,62723	93,60662
Sen_n	4	9	14	19	24
Result (C_n)	69,60714	69,60667	69,60742	69,62723	69,60662

4.1.2.3 Funktionen Derive:

Funktionen itererar över den vektor skapad av Difference ovan och tar fram förändringen i tid mellan varje prov (D_n) och sparar detta i en ny vektor.

Exempel från ett test under utvecklingen:

Tabell 4: Exempel på data extraherad ur insamlad data.

Difference (C_n)	69,60714	69,60667	69,60742	69,62723	69,60662
Derive (D_n)	N/A	-0,0047	+0,0075	+0,0198	-0,0206

4.1.2.4 Funktionen Calculate_skew:

Funktionen använder vektorn som skapats av Derive för att hitta sensorns drift ($D_{/s}$) och maximalt diftvariationsintervall hos driftberäkningen.

Tabell 5: En del av tabell 4.

D_n	N/A	-0,0047	+0,0075	+0,0198	-0,0206

Driftsumma (D_{summa}) = -0,0047 + 0,0075 + 0,0198 - 0,0206 = +0,0020. Dvs, sensorns interna klocka verkar ha lagt på 0,0020 sekunder under de 20 sekunder som dessa fem prover insamlats. Driften per sekund ($D_{/S}$) blir då $\frac{1}{10'000}$ sekund. Maximalt fel (D_{max} och D_{min}) är +0,0198 och -0,0206, med absolutvärdet (D_{abs}) 0,0404.

Resultatet $D_{/s}$ är för sammanhanget ett väldigt högt resultat och beror på att alltför få mätpunkter användes i exemplet. Implementationen skulle heller inte godkänna beräkningen och begära fler datapunkter då den största enskilda förändringen (D_{abs}) överstiger driftsumman (D_{summa}).

4.1.2.5 Funktionen Avg_variation:

Funktionen använder driften ($D_{/s}$), intervallet (I) och rådatan för att hitta hur mycket driften varierar i genomsnitt. Detta görs genom att för varje mätpunkt (n) jämföra en utifrån modellen förväntad rapporteringstid med den faktiska rapport tiden för mätpunkten. Den beräknade rapporttiden är framräknad för en given mätpunkt via formeln:
första mätpunktens tid + (mätpunktens nummer i ordningen * rapporteringsintervallet) + (mätpunktens nummer i ordningen * rapporteringsintervallet * driften).

$$D_1 + (n * I) + (n * I * D_{/s}) = Sen_n.$$

När detta framräknade värde inte helt överensstämmer med rapporttiden för mätpunkten så har man punktens avvikelse från den genomsnittliga driften. Alla dessa variationer sparas var för sig i en vektor och summeras till ett genomsnitt. Vektorn med värden itereras sedan över där varje värde jämförs med det beräknade medelvärdet: om något värde avviker med mer än 2,5 gånger medelvärdet så utesluts det när medelvärdet beräknas på nytt. Detta är filtreringen som diskuteras i sektion 3.1 samt figurerna 6 och 7.

4.1.2.6 Funktionen Compose_return:

Funktionen sammanställer beräkningarnas resultat i en parlista (`map<string, double>`) för att enkelt kunna sätta in resultatet i en databas.

4.1.3 Klassen DBconnection.h & .cc

Klassen DBconnection sköter programmets anslutning till mySQL-databasen och kan antingen sända en SQL-fråga direkt från mainklassen eller konstruera sin egen SQL-fråga utefter en parlista som den som Compose_return ovan tillhandahåller.

4.1.4 Klassen DataCorrection.h:

Den fria funktionen DataCorrection är inte att sammanblanda med den tidigare nämnda funktionen fix_dataset. DataCorrection finns för att använda sig av den framräknade driften och korrigera dataset som inte används till grund för beräkningarna. Funktionen är menad att kunna användas separat från det övriga programmet givet att en analys har gjorts av en given sensors drift. Med driften funnen skall DataCorrection kunna användas på dataset genererade av denna sensor och korrigera dem för den beräknade driften.

4.2 Användning av implementationen:

Programmen har under utvecklingen kompilerats via “gcc” med hjälp av “gmake”, men bör kunna kompileras med andra verktyg. Vid kompilering är implementationen beroende av biblioteket “mysqlcppconn”. Vid exekvering behövs en SQL-server med rådata.

Vid kompilering skapas två exekveringsbara filer: Analysis och Datacorrection. Analysis identifieras driften utifrån rådatan och korrigerar sedan rådatan. Datacorrection korrigerar enbart rådatan och är tänkt att användas när en sensor redan analyserats, men där sensorn har genererat ett nytt dataset som behöver korrigeras för drift.

Indata och utdata:
Programmet förutsätter att databastabellen Sensordata innehåller tidsangivelser för när ett sensorprov rapporterades in och den av sensorn satta tidsstämpeln för denna provtagning.
Den mottagande serverns tid (Ser_n) förväntas stå i kolumnen arrival_time, den av sensorn satta provtagningstiden (Sen_n) förväntas stå i kolumnen report_time och sensorns id förväntas stå i kolumnen sensor_id. Det är viktigt att just detta första dataset insamlats konsekvent och utan avbrott eller störningar såsom, men inte exkluderande: nollställningar, tillfälliga pauser i uppräkningen eller leverans av flera rapporter från olika tidpunkter samtidigt. Om detta sker under insamlingen av den första datamängden så kommer fel att uppstå i samband med beräkningarna. Analysprogrammet kan hantera en vissa typer av fel som tenderar att uppkomma med just den utrustning som använts under utvecklingen, men man bör göra en översyn på sin insamlade data för eventuella brister beroende på vilken hårdvara man själv använder.

I tabellen Skew kommer en sensors drift och detaljer om denna att skrivas ut. Om en sensor sedan tidigare förekommer i tabellen, när data från tabellen Sensordata läses in av programmet, så kommer sensorn ignoreras. Vill man att en sensor skall analyseras på nytt så behöver man avlägsna den ur tabellen Skew.

I tabellen Results kommer datakorrigeringsdelen av programmet skriva ut de driftkorrigerade värdena från tabellen Data.

5. Analys

5.1 Analys av exempeldata.

Arbetet har använt sig av sex olika dataset insamlade under arbetet:

- Set 1 med 14 464 mätpunkter, där sensorn matades med en okänd spänning.
- Set 2 med 47 629 mätpunker, där sensorn matades med 21,85 volt.
- Set 3 med 33 733 mätpunker, där sensorn matades med 28,98 volt.
- Set 4 med 69 818 mätpunker, där sensorn matades med 22,06 volt.
- Set 5 med 54 279 mätpunker, där sensorn matades med 23,40 volt.
- Set 6 med 34 416 mätpunker, där sensorn matades med 26,26 volt.

Spänningen mättes med voltmeter vid uppstart och sensorerna gavs avsiktligt olika styrka för att undersöka vilken, om någon, effekt strömstyrka kan ha på en sensors tidhållning. Att spänningen varierar till synes oordnat mellan de olika mätningarna beror på att strömförsörjningen sattes till min- och maxspänning för set två och tre, för att sedan gå från låg till hög med set fyra, fem och sex. Volttalen har inte valts med någon exakthet utan sattes med hjälp av skruvmejsel.

Tabellen nedan visar resultatet av analysen för de olika samlingarna. Något man kan se är att man utifrån arbetets mätningar inte kan dra några slutsatser om spänningens påverkan på driften. Notera att set 3 och 6 gavs högst spänning och därför borde båda ha värden på samma sida om set 2 och 4, något som inte är fallet nedan: Set 2 och 4 faller mellan 3 och 6 i drift. Jag är fortsatt övertygad om att spänningsstyrka har en effekt på en sensors drift[11] men utifrån dessa resultat så kan inga sådana slutsatser dras. Föreslagtsvis behövs längre mätperioder, fler insamlade mängder och kraftigare variation i försörjningspänning för att undersöka spänningens påverkan på driften. Dock så nöjer vi oss med konstaterandet här att inget samband i just detta samanhang kan påvisas.

Tabell 6: Drift, driftsumma, antal mätvärden och intervall hos exempelmängderna.

Set	Drift (D/s)	Driftsumma (D_{Summa})	Antal prov	Intervall
1	0,0000236	1,70398	14 463	5
2	0,0000227	5,41408	47 628	5
3	0,0000208	3,51292	33 732	5
4	0.0000227	7,91011	69 817	5
5	0,0000233	6,32884	54 279	5
6	0,0000259	4,45328	34 415	5

5.2 Omständigheter med sensorenheterna.

De sensorer som användes har haft vissa egenheter och tendenser som gör att den insamlade datamängden är av något sämre kvalitet än vad man teoretiskt kunde förvänta sig.

5.2.1 Heltalsrapportering.

Sensorerna som användes i detta arbete sänder som mest rapporter var femte sekund. De har heller ingen mer exakt tidsangivelse än heltalssekunder, även om de verkar ha en intern tidshållning med högre upplösning. Det vill säga att de trunkerar decimalerna för sekunden för varje rapport.

Exempel:

Tabell 7: Ett grovt exempel på hur trunkering av decimaler skapar plötsliga variationer i driftsumman.

Servertid (Ser_n)	0	1.2	2.4	3.6	4.8	6	7.2	8.4
Sensortid (Sen_n)	0	1	2	3	4	6	7	8
D_{summa}	0	0.2	0.4	0.6	0.8	0	0.2	0.4

Exemplet är grovt och förklarar bara rörelsen i den ena riktningen. Om än exakta detaljer om sensorns klockkonstruktion saknas så verkar det inte så enkelt att decimaler trunkeras, utan snarare att det rör sig om olika avrundning av sekunden i spannen 0-0,25 och 0,75-1 å ena sidan och 0,25-0,75 å andra. Eftersom relationen mellan den verkliga sekunden och sensorsekunden skiftar mycket långsamt så ger 0,75-1 spannet en större upplevd skillnad och 0-0,25 spannet en

mindre dito. Notera att detta är en effekt skapad av hur denna specifika sensor hanterar tid och förväntas se annorlunda ut eller saknas helt i analysen av andra enheter.

Beteendet ger upphov till lokala extremvärden men gör liten skillnad i det långa loppet. Den interna tidhållningen, oavsett upplösning, har en drift som framkommer i mätningar över längre tid. Dock så gör beteendet att man behöver genomföra mätningen just under längre tid, då en kort mätperiod gör resultatet känsligt huruvida de sista mätningarna var del av det övre eller undre bandet. I egna räkneexempel så har cirka två sekunder tillkommit eller försvunnit beroende på hur stor del av mängden som används och i vilket band mätningen avslutades.

5.2.2 Sensorns förväntningar och önskemål.

Som tidigare nämnts så valdes inte sensorerna i arbetet varken för sin detaljerade tidshållning eller anpassning för detta arbete, detta innebär att de förväntade sig att få tidsuppdateringar från den server de rapporterade till. Som tidigare beskrivits så är detta normalt och är vad hela detta arbete vill erbjuda ett alternativ till. Dock så innebar det att när sensorerna inte fick någon uppdatering från servern, så nollställde de sin egen klocka och började om. Detta skedde var 18 000:e sekund och tidstämplarna i databasen gick då, från analysverktygets perspektiv, tillbaka i tiden var femte timme. Det i sin tur tolkades som en kolossal negativ drift som fullkomligt överskuggade alla andra värden. Detta kringgicks med att rapporttiden räknades om utifrån sin ordning i databasen som om nollställningen inte hade skett.

Begäran på tidstämpel och nollställningen innebar också att den första rapporten efter denna begäran innehöll en stor mängd felmeddelanden och feldata som den mottagande webservern inte kunde hantera. Därför saknas var 3600:e rapport ur samlingarna.

5.3 Är analysprogrammet användbart?

Som ett test av systemet kan vi betrakta den största samlingen set fyra: Mätningen pågick i 349 281 sekunder (~97 timmar) vilket borde innebära att om den började vid tiden noll, och hade en drift om 0,0000227 sekunder per sekund ($D_{/s}$), så bör den sista tidstämpeln (Sen_n) visa tiden 349 289,019. Om vi jämför detta med den sista sensorrapportens så ser vi att det kan stämma. Tidstämpeln är 349 289 sekunder sedan uppstart. Detta visar att algoritmen korrekt beräknat driften för enheten, som över fyra dagar drivit åtta sekunder.

Set	Mätperiod i sekunder	Drift ($D_{/s}$)	Förväntat slutvärde	Faktiskt slutvärde	Genomsnittlig Felgrad
1	72 352,296	0,0000236	72 354,004	72 354	0,2571
2	238 272,586	0,0000227	238 264,985	238 265	0,2621
3	168 750,487	0,0000208	168 753,997	168 754	0,2831
4	349 281,090	0.0000227	349 289,019	349 289	0,2885
5	271 554,998	0,0000233	271 561,325	271 561	0,2518
6	172 165,547	0,0000259	172 170,006	172 170	0,3376

Tabell 8: En kontrollräkning för att undersöka om den beräknade driften är rimlig.

Analysprogrammet verkar sålunda ha beräknat driften rätt, inom någon hundra eller tusendels sekund. Det är en aning olyckligt att de förväntade slutvärdena ovan alla ligger så nära till att avrundas till det faktiska slutvärdet, då det hade varit intressant att söka efter mönster i var de faktiska slutvärdena landar när de förväntade värdet ligger mellan två hela sekunder.

5.3.1 Att använda resultatet.

Tabellen ovan antyder att driften har beräknats korrekt och nu kan användas för att korrigera seriens tidstämplar. Som exempel kan vi ta en tidstämpel i set fem: mätningen med ID 40 000 har en tidstämpel på 200 116. Vi vet inte vilken sekund det motsvarar på en korrekt klocka, men vi vet att sensorn lägger till 0,0000233 sekunder per sekund ($D_{/s}$). Eftersom sensorklockan lägger till tid enligt formeln Realtid + Realtid * Drift, så får vi realtiden via Sen_n / ($D_{/s}$ + 1).
Enligt detta var realtiden när tidstämpeln sattes 200 111,337 sedan sensorn startades. Då detta är exempeldata så vet vi att tidstämpeln sattes 200 111,389 sedan uppstart. Med en felgrad på 0,052 över 200 111 sekunder så faller exemplet väl inom förväntningarna.

5.4 Svar på problemformuleringen i 1.4:

- Kan sensordata synkroniseras utifrån analys av förändringen hos sensorernas klockfrekvens?
    - Arbetet visar att sensordata kan synkroniseras efter insamling och analys av sensorernas klockdrift.

- I vilken grad påverkas sensorernas tidshållning i märkbar grad av sin omgivning avseende strömförsörjning?

- Utifrån arbetet så kan man inte dra någon tydlig slutsats att strömförsörjningen påverkar driften hos en kvartsklocka. Dock har tidigare studier visat samband mellan en klockas drift och dess strömförsörjning.

- I vilken upplösning behöver dataanalysen ske för att klockornas drift skall vara påtaglig?
    - Då resultaten i tabell 8 huvudsakligen skiljer sig åt på mikrosekundsnivå (10^{-6}) så är detta en tillräcklig upplösning.

- I vilken grad påverkar nätverksförseningar tidsstämplarnas tillförlitlighet?
    - Återigen är jag övertygad om att närverksfördröjningar kan ha stor påverkan, speciellt om anslutningarna är globala. Men, återigen, så kan denna slutsats inte dras utifrån detta arbete.

6. Slutsats

6.1 Det som återstår att göra med existerande system.

Det är ingen tvekan om att det förekommer tillkortakommanden med modellen och med dess implementation. Det finns ett flertal områden som skulle behövas omarbetas eller utvecklas. Det finns även vissa motsägelsefulla krav som modellen ställer på hur den används:

- Trots att modellen avser att hantera osäkerheter i tidtagning så är den beroende av att systemet som används som server är någorlunda tillförlitligt, detta då modellen inte kan på förhand etablera egna fasta punkter i tiden.
- Trots att modellen avser att minska beroendet av tillförlitliga nätverksuppkopplingar så är den under mätperioden själv beroende av tillförlitliga nätverksuppkopplingar. Efter mätperioden, som rekommenderas tillåts vara ett par dagar, är dock ett tillförlitligt nätverk till sensorerna inte längre en nödvändighet.

Kvaliteten på arbetets analys kan förbättras om mätningar kunde genomföras över längre tid, över större avstånd, med större mängd och variation av sensorer, temperaturer, strömstyrkor etc, för att ge mer utförliga och välgrundade svar på frågorna två och fyra om vad som kan påverka tidshållning hos kvartsoscillatorer och hur.

6.2 Vidareutveckling.

I nuläget finns heller inget sätt att avgöra hur lång en mätperiod behöver vara, något som varierar med hur pass tillförlitlig nätverksanslutningen är och hur pass stora variationer en sensorklocka har. Om variationen är större än driften så kommer implementationen i nuläget först att varna att ingen analys kan genomföras för att sedan begära ett större dataset. Dock så skulle en fingervisning om hur mycket större samlingen behöver vara i dagar vara händigt. Detta skulle kunna implementeras via en beräkning av hur snabbt den beräknade driften har närmat sig den största variationen sedan mätningens början.

Naturligtvis skulle hela arbetet må bra av att genomföras med ett atomur som referens, då det i nuläget kan vara känsligt för drift i den centrala servern eller tidskorrigering i den centrala servern enligt problemet i figur 2.

Slutligen så skulle en bättre filtreringsfunktion av tidsstämplar behövas. Något som aldrig behövdes i detta arbete, men som kommer att vara nödvändigt om modellen skall användas i mer komplexa miljöer. I arbetet här så förekom enbart en nätverksnod mellan sensorerna och servern, något som i en industriell miljö är i det närmaste otänkbart. I ett tänkbart verkligt sammanhang där lösningen skulle kunna komma att användas så blir filtreringen allt mer nödvändig ju större och vältrafikerat nätverket är.

6.3 Etisk reflektion.

Uppsatsen, programmet som producerats, och det som undersökts borde inte ge upphov till några etiska problem. I modern tid där stora datamängder insamlas och sammanställs, så bör all datainsamling och aggregering ske med eftertanke. Detta då stora dataset gör det möjligt att dra slutsatser som dataseten inte var avsedda för och som kan vara både integritetskränkande och säkerhetshotande. Med det sagt så fokuserar detta examensarbete enbart på datapunkters tidsstämplar och dessa tidsstämplars kvalitet, och befattar sig på inte med datapunkternas faktiska innehåll. Vidare så är den data som använts i arbetet insamlad med ägande företags godkännande och innehåller ingen företagskänslig data eller användardata.

Därför behöver ingen tillfrågas eller informeras om sitt deltagande, ingen tystnadsplikt är applicerbar och resultatet av arbetet skapar inga etiska konflikter.

7. Terminologi

Allanvarians - En typ av variansberäkning som används för att jämföra klockor och som utvecklades av David W Allan under 1960-talet.[15]

Driftsumma - Den ansamlade mängden drift; den skillnad i tid som (av drift) uppstått mellan idealt värde och det värde klockan visar. Antalet tidsenheter en klocka kommit ur synkronisering med en referens.

Driftens variation - Ett mått på skillnader i drift vid olika tidpunkter. Nära besläktat med en klockas stabilitet då frånvaron av stabilitet ger upphov till variationer i klockors drift. Kan beskrivas som ett spann inom vilket en klockas drift förväntas hålla sig.

Felgrad - Differensen mellan den beräknade tiden för en klocka vid en given tidpunkt (utifrån dess uppmätta drift) och klockans faktiska värde vid tidpunkten. Felgraden används av analysprogrammet för att skatta resultatets kvalitet.

Klockdrift - En klockas tendens att driva från korrekt tid. Något som sker beroende på en klockas konstruktion och hantering. Olika konstruktioner och yttre faktorer kan påverka dess korrekthet och orsaka olika magnituder av klockdrift.

Korrekthet - En klockas förmåga att hålla sig till korrekt tid, sammanräknat dess precision och stabilitet.

Precision - Ett mått på en klocka eller oscillators förmåga att hålla rätt frekvens. Ett mått beräknas som uppmätt sekund per ideal sekund ($\Delta t/t$) eller uppmätt frekvens per ideal frekvens ($\Delta f/f$) och blir enhetslöst. Detta värde används bland annat för att jämföra klockor.[7]

Stabilitet - Ett mått på en klockas/ocillators förmåga att hålla konstant precision.[7]

8. Referenser

[1] “LoRa Alliance and Wireless Broadband Alliance Demonstrate Increase in Market Opportunity and ROI for IoT Deployments that Combine LoRaWAN and Wi-Fi.“
LoRa alliance, pressrelease 17/11/2020
https://lora-alliance.org/lora-alliance-press-release/lora-alliancer-and-wireless-broadband-alliance-demonstrate-increase-market-opportunity/ (Hämtad bl.a 21/05/21.)

[2] “Introduction to quartz frequency standards”.
John R. Vig. U.S. Army Laboratory Command, 03/1992
https://apps.dtic.mil/dtic/tr/fulltext/u2/a248503.pdf (Hämtad bl.a 21/05/21.)

[3] Enersizes ABs affärskoncept.
https://enersize.com/ (Hämtad bl.a 21/05/21.)

[4] “Simulation Engine for Analysis and Comparison between Cristian’s and Berkeley clock synchronization algorithms.”
Rahul Sehgal, Department of Computer Science, Kent State University, 2007
http://antares.cs.kent.edu/~mikhail/classes/aos.f07/ProjectReports/sehgal.report.pdf (Hämtad bl.a 21/05/21.)

[5] “Fully Distributed Clock Skew And Offset Estimation In WirelessSensor Networks”
Du, J; Wu, YC, University of Hong Kong, 2013.
https://core.ac.uk/download/pdf/38032398.pdf (Hämtad bl.a 21/05/21.)

[6] “Clock synchronization of wireless sensor networks”,
Yik-Chung Wu, Qasim Chaudhari, Erchin Serpedin, IEEE Signal Processing Magazine, 01/2011.
https://www.eee.hku.hk/~ycwu/publication_download/ycwu_SPM_2011.pdf (Hämtad bl.a 21/05/21.)

[7] “The accuracy and stability of quartz watches.”
Michael Lombardi, Horological Journal, 02/2008.
https://tf.nist.gov/general/pdf/2276.pdf (Hämtad bl.a 21/05/21.)

[8] “Reference clock support”
NTPs officiella dokumentation. Network time foundation. 22/09/2010
https://www.eecis.udel.edu/~mills/ntp/html/refclock.html (Hämtad bl.a 21/05/21.)

[9] Diverse information om NTP via dokumentationssidan.
Network time foundation. 22/09/2010
http://www.ntp.org/documentation.html (Hämtad bl.a 21/05/21.)

[10] “GMU191”
Ionsigns officiella dokumentation.
https://www.ionsign.fi/en/iot-big-data/gmu191/ (Hämtad bl.a 21/05/21.)

[11] “Worst Case Clock Skew under Power Supply Variations.”
Min Zhao, Kaushik Gala, Vladimir Zolotov, Yuhong Fu, and Rajendran Panda, Motorola Inc., Austin, TX, USAR.
Ramkumar and Bhuwan Agrawal, Motorola India Design Center, Noida, India.
TAU '02: Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems. 12/2002
http://www.gtcad.gatech.edu/www/weekly/sp07/p22-zhao.pdf (Hämtad bl.a 21/05/21.)

[12] “Clock skew based node identification in wireless sensor networks.”
Ding-Jie Huang, Wei-Chung Teng, Chih-Yuan Wang, Hsuan-Yu Huang and Joseph M. Hellerstein. National Taiwan University of Science and Technology, Taipei, Taiwan. 2008.
https://www.researchgate.net/profile/Wei-Chung-Teng/publication/224356243_Clock_Skew_Based_Node_Identification_in_Wireless_Sensor_Networks/links/0c96052b0b451224d9000000/Clock-Skew-Based-Node-Identification-in-Wireless-Sensor-Networks.pdf (Hämtad bl.a 21/05/21.)

[13] “Temperature-Compensated Clock Skew Adjustment.”
Jose María Castillo-Secilla, Jose Manuel Palomares och Joaquín Olivares.
University of Cordoba, 2013.
https://www.mdpi.com/1424-8220/13/8/10981/pdf (Hämtad bl.a 27/05/22.)

[14] “RTSP: An Accurate and Energy-Efficient Protocolfor Clock Synchronization in WSNs.”
Muhammad Akhlaq och Tarek R. Sheltami. IEEE, 2013.
https://www.researchgate.net/profile/Tarek-Sheltami/publication/260304065_RTSP_An_Accurate_and_Energy-Efficient_Protocol_for_Clock_Synchronization_in_WSNs/links/573968e108ae9f741b2bf2af/RTSP-An-Accurate-and-Energy-Efficient-Protocol-for-Clock-Synchronization-in-WSNs.pdf (Hämtad bl.a 27/05/22.)

[15] “Statistics of Atomic Frequency Standards.”
David W. Allan. Proceedings of the IEEE, 1966.
https://tf.nist.gov/general/pdf/7.pdf

8.1: Figurer & tabeller:

Figur 4 är skapad av Mie Kristensen.

Printed by Tryckeriet i E-huset, Lund 2025

Series of Bachelor´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2025-1056
http://www.eit.lth.se