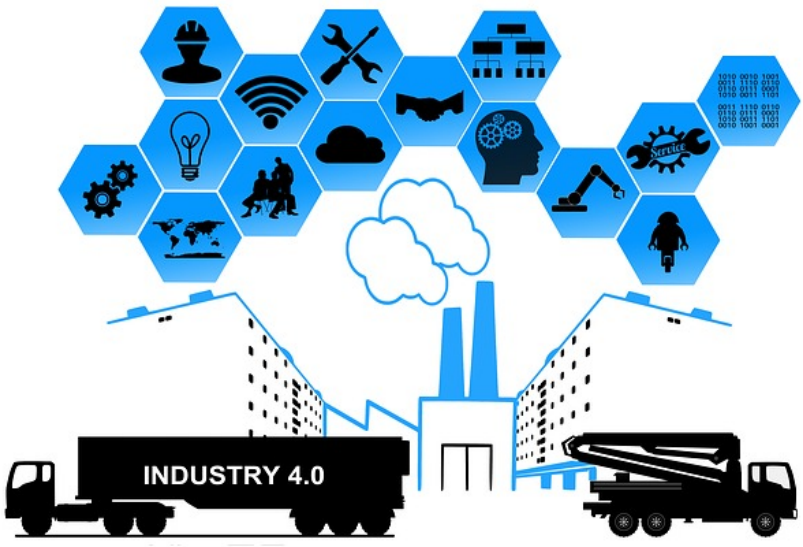
CHALMERS



Trådlös kommunikation över mobilt nätverk i industriell miljö

Koncepttest och analys av industriellt bussprotokoll över trådlöst nätverk

Examensarbete inom högskoleingenjörsprogrammet Datateknik

OSCAR HALL
RASMUS LINDY

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

EXAMENSARBETE

Trådlös kommunikation över mobilt nätverk i industriell miljö

Koncepttest och analys av industriellt bussprotokoll med trådlös kommunikation

OSCAR HALL
RASMUS LINDY

CHALMERS

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
Göteborg, Sverige 2018

Trådlös kommunikation över mobilt nätverk i industriell miljö
Koncepttest och analys av industriellt bussprotokoll med trådlös kommunikation
OSCAR HALL
RASMUS LINDY

Examinator: Peter Lundin, Institutionen för Data- och Informationsteknik
Handledare: Joachim von Hacht, Institutionen för Data- och Informationsteknik

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Framsida: Konceptbild för IIOT, Industrial Internet of Things.

Sammanfattning

Rapporten beskriver utförandet av ett koncepttest där ett virtuellt styrsystem kommunicerar med en industrirobot via mobilnätet. Koncepttestet utfördes för att undersöka om det går att ersätta den trådade kommunikationen inom en fabriksstation med trådlös kommunikation över ett mobilt nätverk. Koncepttestet har bestått av att konstruera en mjukvarulösning och utföra tester för att mäta vilken lägsta cykeltid som går att uppnå. Testerna utfördes på en uppsättning där mjukvarulösningen exekverades i ett serverkluster och skickade Ethernet-ramar via UDP i en L2TP tunnel till en Raspberry Pi. Det industriella protokollet Powerlink användes och nätverket under koncepttestet bestod av en master och slavnod. Den lägsta uppmätta cykeltiden blev 40 ms med en fördröjning på 15 ms över mobilnätet. Det fungerade bra att skicka Ethernet-ramar över det mobila nätverket, men 40 ms cykeltid är för långsamt för att kunna ersätta existerande trådad kommunikationslösning. Däremot kan det appliceras på andra tidskritiska system inom industrin, exempelvis kommunikation med överordnade system.

Nyckelord: LTE-Advanced, Powerlink, trådlös fältbuss, Industri 4.0, Industrial IoT.

Abstract

The report describes an implementation of a proof of concept where a virtual control system communicates with an industrial robot over the mobile network. The proof of concept was used to analyse the possibility of replacing wired communication within a factory station with wireless communication over a mobile network. The proof of concept consisted of creating a software application and performing tests to analyse the lowest possible cycle time. The tests consisted of a setup where the software application was placed and executed on a server sending Ethernet frames through UDP packets in a L2TP tunnel to a Rasperry Pi. The industrial fieldbus protocol Powerlink was used and the network consisted of one master and slave node. The lowest possible cycle time achived was 40 ms with a latency of 15 ms over the mobile network. Sending Ethernet frames over the mobile network worked fine, but a cycle time of 40 ms is too slow to replace a wired communication. However, it could be applied on other time critical systems within the industry, for example communication with overall systems.

Keywords: LTE-advanced, Powerlink, Wireless fieldbus, Industry 4.0, Industrial Iot

Förord

Vi vill tacka Ericsson och ABB för att fått möjligheten att arbeta med detta intressanta examensarbete. Ett särskilt tack till Lars Egeland, Ericsson, och Magnus Seger, ABB, som ordnat möten och hjälpt oss att komma i kontakt med rätt personer. Ett stort tack till Joachim von Hacht som handlett och gett oss värdefull feedback under rapportskrivandet.

Rasmus Lindy, Oscar Hall, Göteborg, Juni 2018

Innehållsförteckning

Förkortningar

3GPP 3 Generation Partnership Project. 11

AGV Automated Guided Vehicle. 37
APN Access Point Name. 21
ASnd Asynchronous Send. 13

CN Controlled Node. 13
CodeSys Controller Development Systems. 14

EPL Ethernet Powerlink. xiv, 12
ERP Enterprise Resource Planning. 8

HDLC High-Level Data Link Control. 10
HMI Human Machine Interface. 9

IIoT Industrial Internet of Things. 1

L2TP Layer Two Transfer Protocol. 11
LTE Long Term Evolution. 10

MN Managing Node. 13

PLC Programmable Logic Controller. 14
PPP Point-to-Point Protocol. 10
PReq Poll Request. 13
PRes Poll Respone. 13

SoA Start of Asynchronous phase. 13
SoC Start of Cycle. 13

Figurer

Tabeller

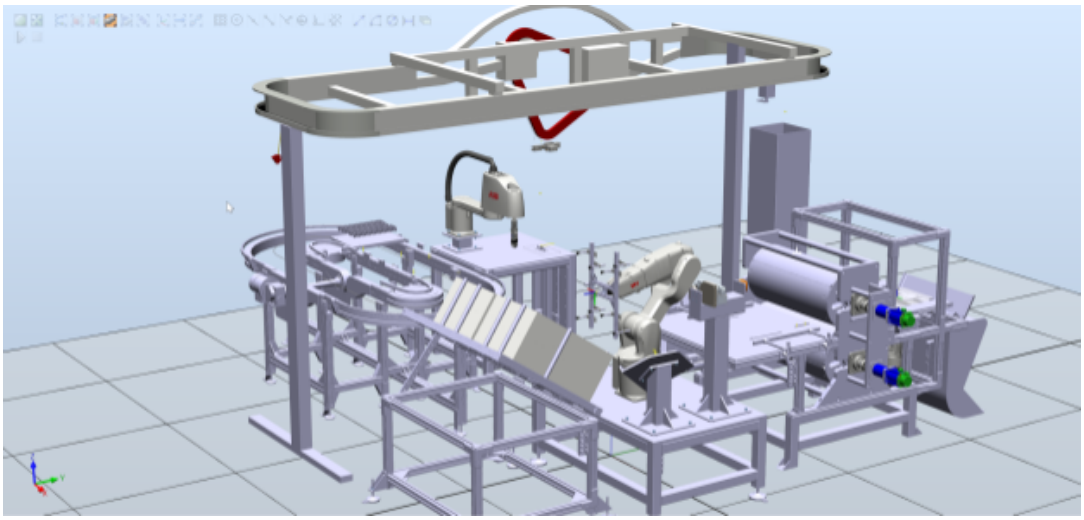
Kod

1
Inledning

1.1 Bakgrund

Industri 4.0, även känd som den fjärde industriella revolutionen eller Industrial Internet of Things (IIoT), är en samling av olika teknologier och koncept inom automation, processindustriell IT och produktionsteknologier. En del av Industri 4.0 är att varje produkt i produktionskedjan ska ha med sig information om vart den ska och hur den ska bearbetas. På detta sätt ska fabriken kunna organisera sig själv i större grad. Ett mer dynamiskt flöde i produktionen skulle även medföra kortare omställnings- och ledtider.[1]

Industri 4.0 innebär också att sensorer och ställdon kopplas samman med fabrikens övriga IT-system för att få bättre överblick över produktionsflödet. Det innebär dessutom möjligheter för att kunna utöva förebyggande underhåll på ett mer effektivt sätt.[1] Det finns ett pågående projekt, ”Smarta Fabriker”, på Lindholmen i Göteborg som fungerar som en plattform för industriell digitalisering och drivs utav Göteborgs Tekniska College. Projektet är finansierat av näringslivsdepartementet och Västra Götalandsregionen samt ett flertal olika industriella partners som deltar i olika grad. I projektet har man byggt upp en modern fabriksstation med hjälp av studenter som handletts av industriföretag. Denna ska användas för att pröva ut och visa upp olika koncept inom Industri 4.0.[2] Se figur 1.1.

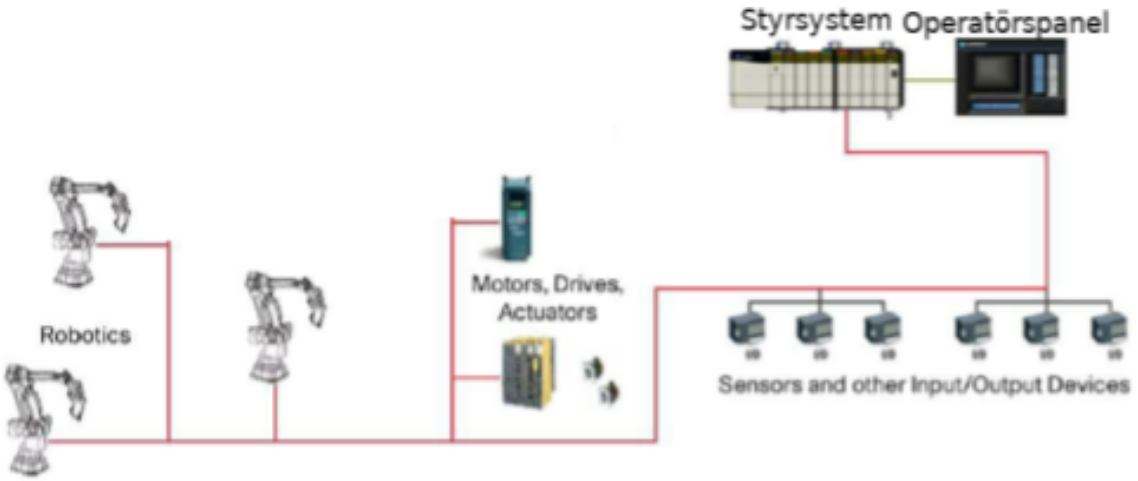


Figur 1.1: Modell av Smarta Fabrikers fabriksstation i simuleringsprogrammet RobotStudio.

Detta examensarbetet utförs i samarbete med ABB och Ericsson samt ”Smarta Fabriker”. ABB är ett av de största företagen inom kraft- och automationsteknik, medan Ericsson är ett av de största företagen inom nätverk och kommunikationsteknik. Med detta examensarbete så vill ABB och Ericsson undersöka om det går att ersätta den trådade kommunikationen i fabriken med trådlös kommunikation och låta ett virtuellt styrsystem styra en fabriksstation via mobilnätet. Tanken är att det koncepttest som kommer att skapas senare ska kunna visas upp i ”Smarta Fabrikers” fabriksstation.

1.2 Fabriksstation

Som det ser ut idag i en fabrik brukar det finnas ett styrsystem vid varje station längs en fabrikslinje som ansvarar för styrningen för den stationen. Styrsystemet kan sedan vara ihopkopplat med underordnade enheter som industrirobotar, motorer eller enheter med tillkopplade sensorer och givare. Dessa underordnade enheter är ofta kopplade till styrsystemet via en industriell kommunikationsbuss baserad på Ethernet som medium.[3] Se figur 1.2.



Figur 1.2: Exempel på hur kommunikationen inom en fabriksstation i en fabrik kan se ut. Den röda linjen representerar den industriella kommunikationsbussen.

Kommunikationsbussens transportprotokoll är utformat på ett sådant sätt att det kan uppfylla hårda realtidskrav för datakommunikationen. Detta innebär att garantier ges för att den data som skickas alltid kommer att komma fram till mottagaren inom ett visst tidsintervall. Denna bortre tidsgräns sätts när man konfigurerar och driftsätter bussystemet.

Med trådad kommunikation är datakommunikationen inom en fabrik stabil och säker, men det gör den också statisk. En trådlös lösning som kan åstadkomma samma kvalitet som en trådad lösning skulle kunna göra samma fabrik mer dynamisk. Därmed skulle det bli smidigare att genomföra förändringar av produktionskedjan vid omställningar av produktionen. Utmaningen med att använda trådlös kommunikation är att den bortre tidsgräns som blivit satt fortfarande måste hållas för att kunna uppfylla realtidskraven i kommunikationssystemet.

2
Syfte

Undersöka om det går att ersätta nuvarande trådade kommunikationen mellan industriella styrsystem och övriga enheter i en fabriksstation med en trådlös kommunikationslösning.

3
Mål

Målet är att utveckla ett prototypsystem för ett koncepttest där ett virtuellt styrsystem styr ABBs industrirobot via mobilnätet. Undersöka vilken lägsta bortre tidsgräns som går att uppnå med en sådan trädlös lösning.

4

Avgränsningar

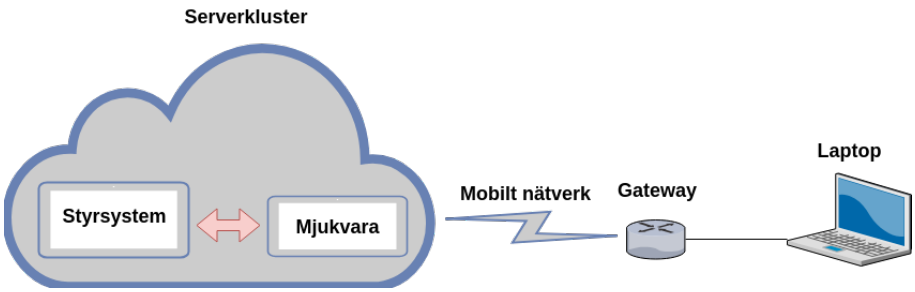
Examensarbetet kommer inte att beröra virtualisering av styrsystemet. Arbetet är begränsat till att behandla den del som ska möjliggöra att ta emot signaler från styrsystemet och skicka vidare dessa till industriroboten.

5
Metod

För att få fram underlag för att konstruera koncepttestet på ett lämpligt sätt kommer tidigare examensarbeten och forskningsrapporter som arbetat med liknande projekt att studeras. Den lägsta bortre tidsgräns som går att uppnå med koncepttestet kommer att testas fram. Med hjälp av testresultaten kommer det att analyseras om realtidskraven fortfarande kan uppfyllas med trådlös kommunikation. Man kommer också undersöka vilken lägsta bortre tidsgräns som skulle vara acceptabel dels för "Smarta Fabrikers" fabriksstation och dels generellt i industriella tillämpningar.

För att utföra testerna kommer ett system att utvecklas. Systemet kommer att vara en enkel uppsättning som efterliknar hur det kan se ut i en fabriksstation. Under utvecklingen av systemet kommer en agil utvecklingsprocess att användas med dagliga möten och veckovisa utvärderingsmöten.

Testerna kommer att utföras på en uppsättning som beskrivs i figur 5.1. Styrsystemet kommer exekveras i ett serverkluster och kommunicera över ett mobilt nätverk med hjälp av en utvecklad mjukvara. På motsatt sida av nätverket kommer en gateway ta emot signalerna och vidarebefordra de till en laptop.



Figur 5.1: Uppsättningen som ska användas under tester.

6
Teoretisk bakgrund

6.1 Master/slave

Master/Slave är en åtkomstmetod som innebär att all kommunikation initieras av en utvald masternod. Denna åtkomstmetod är vanligt förekommande hos bussprotokoll. Ett protokoll som implementerar Master/Slave låter typiskt masternoden kommunicera med varje slavnod i tur och ordning för att utbyta data.[4]

6.2 Cykeltid

I industriella bussprotokoll pratar man om cykeltid vilket innebär den tid det tar för en masternod att uppdatera och få tillbaka lägesdata från samtliga tillkopplade slavnoder. Cykeltiden är ett fast tidsintervall som konfigureras då man driftsätter bussystemet. Saker som kan påverka vilken lägsta cykeltid man kan ha är bland annat hur många noder som är tillkopplade samt vilken fördröjning som finns i systemet.

6.3 Fältbuss

En fältbuss är en industriell digital kommunikationsbuss som används för att uppnå distribuerad realtidskontroll i industriella styr- och övervakningssystem över enheter som till exempel givare eller frekvensomvandlare i en fabrik.[5] Många av de fältbussar som används idag är baserade på Ethernet som medium.[6]

6.4 Realtidskrav

Ett industriellt styrsystem kan beskrivas som ett realtidssystem och arbetar under en förutbestämd cykel. Under en cykel så skannar styrsystemet av sina ingångsvärden, exekverar sin logik och uppdaterar därefter utgångsvärden med resultatet från exekveringen. Därför är det viktigt att den data som styrsystemet skannar av är aktuell för att logiken ska fungera korrekt.[7]

Styrsystemets ingångsvärden kan hämtas från fysiska ingångsportar på styrsystemet men också via en fältbuss dit givare och sensorer är kopplade. Detta kräver att kommunikationen över fältbussen är deterministisk och garanterar att datan kommer fram inom ramen för styrsystemets cykeltid. Om datan inte kommer fram inom den bortre tidsgränsen som blivit uppsatt så betraktas den som värdelös.

För att en fältbuss ska vara användbar måste den kunna uppfylla de hårda realtidskrav som sätts på industriella applikationer. Detta behöver inte innebära att tidsintervallet för den bortre tidsgränsen, det vill säga cykeltiden, måste vara kort. Men garantera att data som blivit sänd kommer fram innan cykeltidens slut. Vilka realtidskrav som finns beror på vilken typ av applikation kommunikationssystemet ska användas till. Det finns tre olika kategoriseringar angående cykeltid inom Ethernetbaserade fältbussprotokoll, se figur 6.1.

- **Kategori A**: Applikationer med låga realtidskrav med en cykeltid upp till 100 ms.
- **Kategori B**: Applikationer med höga realtidskrav med en cykeltid upp till 10 ms. I denna kategorin finns många Ethernet baserade fältbussprotokoll inom automation.
- **Kategori C**: Applikationer med dom högsta realtidskraven med en cykeltid upp till 1 ms. Applikationer som hanterar rörelse faller inom denna kategori.

Figur 6.1: Kategorier av cykeltider inom Ethernetbaserade fältbussprotokoll.[8]

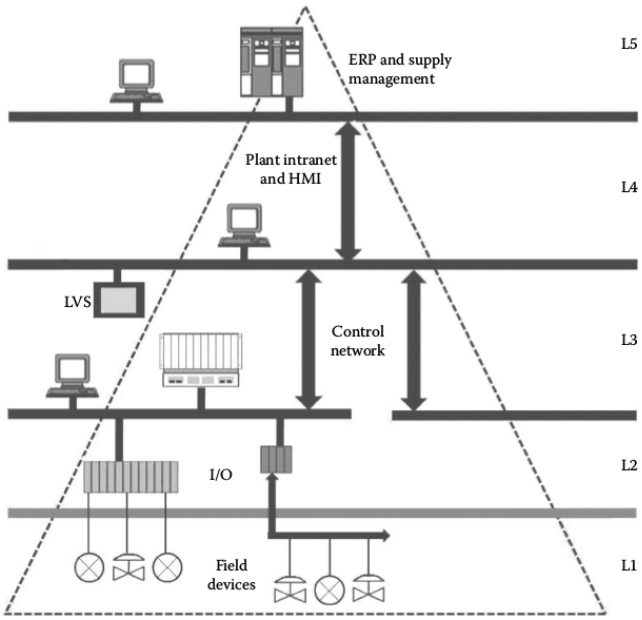
Applikationer som uppfyller realtidskraven inom **kategori A** är ofta baserade på TCP/IP. Användandet av TCP/IP innebär att protokollet inte är beroende av vilken typ av hårdvara eller medium som används. Det gör det också möjligt att kommunicera med enheter utanför det lokala nätverket eftersom adresseringen bygger på IP-adresser. Applikationer inom den här kategorin kan typiskt vara kommunikation till överordnade system i fabriken som samordnar eller övervakar flera stationer inom fabriken. Det kan även innebära kommunikation till Enterprise Resource Planning (ERP) system som kan övervaka saker som materialsaldo eller ställtider.[8]

Inom tillverkningsindustrin så är det typiskt **kategori B** som tillämpas och de flesta Ethernet-baserade fältbussprotokollen är designade för att uppfylla denna kategori av realtidskrav. Realtidsegenskaperna åstadkoms genom att hoppa över TCP/IP lagret och basera protokollet helt på Ethernet-ramen. Detta gör protokollen kompatibla med vanlig Ethernet hårdvara. Adressering av enheter på bussnätverket sker med MAC-adresser vilket ger mindre protokoll-overhead och leder till bättre responstider och högre determinism.[8]

För **kategori C** så modifierar man Ethernetegenskaperna för att uppnå ännu lägre responstider. Till exempel kan åtkomstkontrollen eller timingegenskaperna ändras. Man förlitar sig också på olika egenskaper hos hårdvaran för att uppfylla realtidskraven vilket kräver specialutvecklade Ethernetuttag och switchar. Denna kategori av realtidskrav tillämpas på applikationer som hanterar rörelse, till exempel styrning av motorer och servon. Eftersom det krävs hög precision på rörelsen av en mekanisk arm som styrs av en servomotor eller en robot så krävs det därmed en högre uppdateringsfrekvens. Detta för att styrsystemet ska ha en tillräckligt korrekt bild av verkligheten.[8]

6.5 Hierarkier inom industriell kommunikation

Fabrikens komponenter och nätverk delas upp i olika nivåer. Olika prestandakrav sätts på kommunikationen beroende på vilken nivå i hierarkin kommunikationen sker. Figuren 6.2 beskriver fem olika lager av nätverk och komponenter, L1 till L5, inom en fabrik. Lager L1 och L2 innehåller fabriks komponenter som givare, ställdon, elmotorer eller robotar. Inom lager L3 finns de styrsystem som styr en fabriksstation eller fabrikslinje. Lager L4 omfattar typiskt olika typer av *Human Machine Interface (HMI)* som exempelvis operatörspaneler eller överordnade system för att övervaka flera processer eller fabrikslinjer. Lager L5 innebär olika affärssystem som följer upp produktionsflödet som program för *ERP* eller *supply management*. Dessa kan användas för kontroll av materiallager, inköpspriser eller kartläggning av flaskhalsar i produktionen.[9]



Figur 6.2: Hierarki inom processautomation. Figuren är tagen från boken *Fieldbus and Networking in Process Automation* [9].

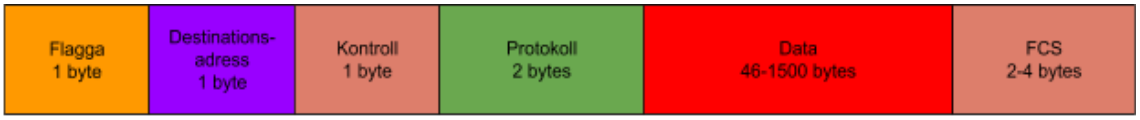
I en fabriksstation så innefattas lager L1 till L3. De Ethernetbaserade fältbussprotokoll som används i dessa lager tillämpar oftast realtidskrav från kategori B med en cykeltid på 10ms. Se lista i figur 6.1. Denna cykeltid gäller främst inom tillverkningsindustrin medan processindustrin, där det rör sig om långsammare processer, kan ha krav på cykeltider som ligger något högre. [9] Mellan lager L3 och L4 tillämpas typiskt realtidskrav från kategori A. På kommunikationen från L3 och L4 till L5 appliceras antingen kategori A eller inga realtidskrav alls. Se lista i figur 6.1.

6.6 LTE

Long Term Evolution (LTE) är en standard inom telekommunikationsteknologi som utvecklades för att ersätta 3G-standarderna.[10]

6.7 PPP

Point-to-Point Protocol (PPP) är ett protokoll som ligger i datalänkskiktet i OSI-modellen och används för att upprätta en länk mellan två ändpunkter. PPP innehåller bland annat en metod för att kapsla in andra nätverksprotokoll som då kan skickas över den upprättade länken. Detta gör det möjligt att skicka nätverkspaket från ett främmande protokoll över ett nätverk som egentligen inte stöder det främmande protokollet.[11]



Figur 6.3: PPP-ramen

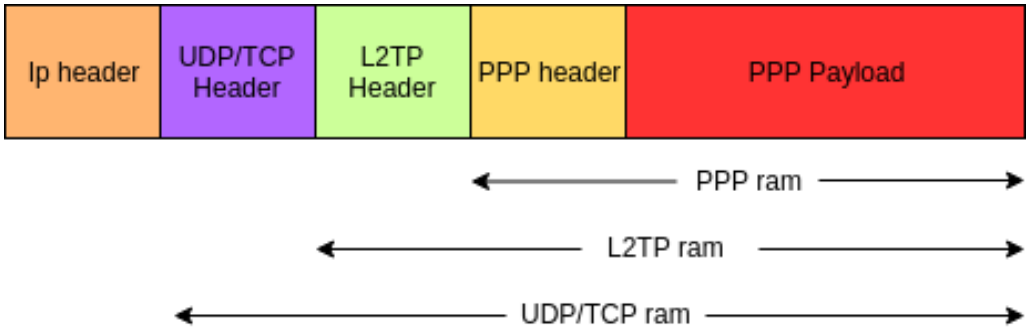
PPP-ramen består av sex olika sektioner. **Flagga**, avser när ramen börjar och slutar. **Destinationsadress**, IP-adressen för mottagaren. **Kontroll**, sätts till ett statiskt värde (00000011) och används av High-Level Data Link Control (HDLC) för att kontrollera ramen. **Protokoll**, Avser vilket nätverksprotokoll som är inkapslat i ramen. **Data**, här lagras informationen och data som skickas. **FCS**, Innehåller en kontrollsumma som används av mottagaren för felhantering av paketet, ifall något har hänt under överföringen. Se figur 6.3.[11]

6.8 Layer two

Med layer two syftas det på det andra lagret i OSI-modellen även kallat data link layer eller datalänkskiktet.[12]

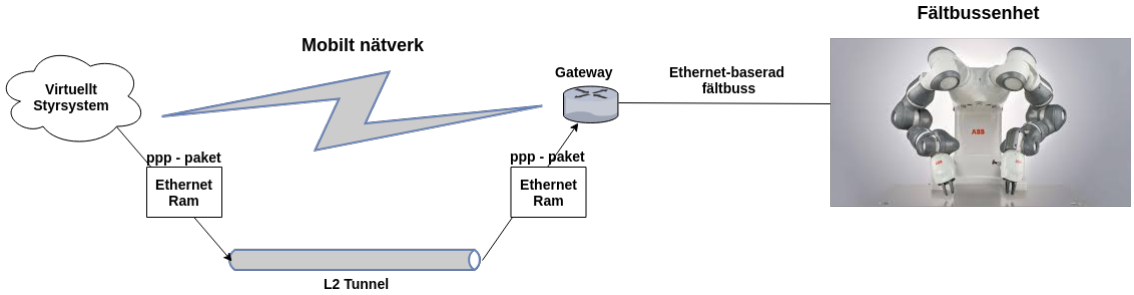
6.9 L2TP

Layer Two Transfer Protocol (L2TP) är en utökning av PPP-protokollet. L2TP använder sig av den generella inkapslingsmetoden hos PPP för att skicka paket från ett främmande protokoll över ett IP-nätverk. Detta innebär att man kan kapsla in ett paket från det främmande protokollet i ett UDP- eller TCP-paket.[11] Se figur 6.4.



Figur 6.4: Figuren beskriver ett IP-paket då man använder sig utav L2TP. PPP Payload innehåller paketet från det främmande protokollet.

Användaren av en L2TP-tunnel upprättar en länk i datalänksskiktet och använder den länken för att tunnla ramar från det främmande protokollet. Eftersom att Ethernet är en IEEE802.2 standard och LTE är en 3 Generation Partnership Project (3GPP) standard går det inte att skicka Ethernet-ramar över ett LTE-nätverk. Med L2TP-tunneln blir detta möjligt. Se figur 6.5.



Figur 6.5: Beskrivning av hur en Ethernet-ram kan transporteras via en L2TP-tunnel över ett LTE-nätverk

7
Teknisk bakgrund

7.1 PROFINET

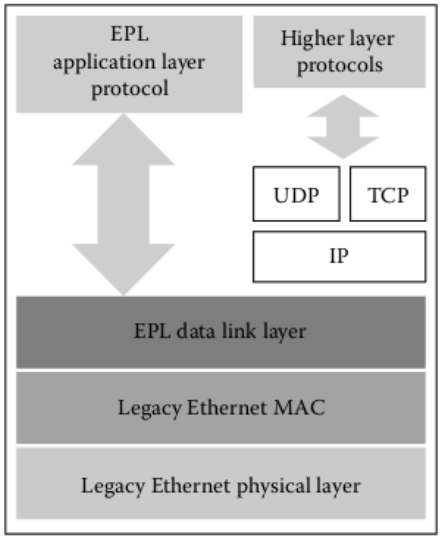
PROFINET är ett Ethernet-baserat fältbussprotokoll utvecklat av PROFIBUS International.[13] PROFINET är det protokoll som används i ”Smarta Fabrikers” fabriksstation för kommunikation mellan PLC och ABBs industrirobotar.

7.2 EtherNet/IP

EtherNet/IP är ett Ethernet-baserat fältbussprotokoll.[14] Det implementeras av ABBs industrirobot som används i ”Smarta Fabrikers” fabriksstation.

7.3 Powerlink

Powerlinkprotokollet är ett Ethernet-baserat fältbussprotokoll utvecklat av B&R Automation. Det är definierat i datalänksskiktet ovanpå Ethernets metod för åtkomstkontroll. Se figur 7.1. Protokollet är helt mjukvarubaserat vilket innebär att det inte förlitar sig på specialdesignad hårdvara för att uppnå realtidsegenskaper. Man kan använda samma Ethernet-kontakt som sitter på en vanlig PC.[15]



Figur 7.1: Powerlink-protokollets arkitektur. EPL står för Ethernet Powerlink. Figuren är tagen från boken *Industrial Communication Systems* [15].

7.3.1 Kollisionshantering i Powerlink

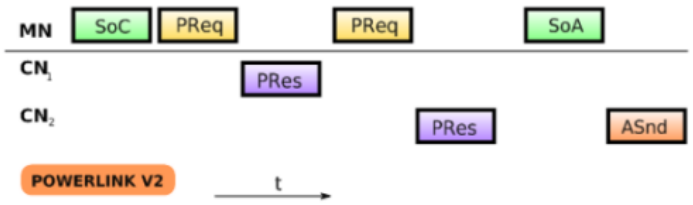
I Powerlink har Ethernets ursprungliga kollisionshantering CSMA/CD bytts ut. Anledningen till detta är att CSMA/CD är baserad på att alla noder i ett nätverk först tittar om bussen är fri, innan den skickar data. Om den inte är fri, genereras en slumpmässig tid som noden väntar innan den tittar igen. Detta medför ett ickedeterministisk beteende där en nod, teoretiskt sett, kan systemet hamna i ett scenario då den aldrig får skicka data på bussen.[16] Istället för CSMA/CD använder sig Powerlink av en master (Managing Node) och slav (Controlled Node) princip där varje slav i nätverket blir tilldelad ett eget tidsfönster där den får skicka data. På det sättet kan man garantera att data inte kolliderar och blir korrupt, samt att data inte skickas ut på bussen från flera noder samtidigt.[15]

7.3.2 Kommunikationscykel i Powerlink

Längden av en kommunkationscykel i Powerlinkprotokollet är baserad på antalet noder som finns på nätverket, samt hur mycket data som skickas från och till varje nod. Varje cykel innehåller följande meddelanden:

- En Start of Cycle (SoC) ram med en storlek på 64 bytes.
- n antal Poll Request (PReq)/Poll Respone (PRes) ramar med storlek mellan 64 - 1518 bytes.
- En Start of Asynchronous phase (SoA) ram med storlek på 64 bytes.
- Asynchronous Send (ASnd) ram med storlek mellan 318 - 1518 bytes.

I början av en cykel skickar Managing Node (MN) ett SoC för att synkronisera nätverket. Sedan sker en dialog mellan MN och varje Controlled Node (CN) där ett PReq och PRes utbytes. Därefter reserveras en fast tid av Powerlink för asynkron dataöverföring. Starten av den asynkrona fasen markeras med SoA. och ASnd. Under denna sista fas kan paket som inte kräver någon determinism skickas i en ASnd, till exempel eventuella TCP/IP-paket. Se figur 7.2.



Figur 7.2: En kommunikationscykel i fältbussprotokollet Powerlink. Figuren är tagen från *POWERLINK and Real-Time Linux* [17].

7.3.3 OpenPowerlink

OpenPowerlink är en open-source implementation av Powerlinkprotokollet skrivet i C. Implementationen användes i examensarbetets mjukvarulösning.[18]

7.4 Gateway

En gateway är en nätverksnod som gör det möjligt för nätverk med olika protokoll att kommunicera med varandra.[19]

7.5 Programmable Logic Controller (PLC)

Industriella styrsystem Programmable Logic Controller (PLC) är hjärnan i ett produktionsflöde eller i en fabriksstation. Ett PLC kan beskrivas som en dator med en processor och ett stort antal ingångs- och utgångsanslutningar. Till dessa anslutningar kopplas givare och ställdon in. Till skillnad från en vanlig dator är ett PLC speciellt utvecklat för att kunna motstå tuffa miljöer. Det ska exempelvis kunna klara extrema temperaturer, fukt, vibrationer och dammiga miljöer.[7] Se figur 7.3.

Figur 7.3: ABBs modulära styrsystem AC500 med bland annat digitala ingångs- och utgångsanslutningar. Bilden är tagen ifrån ABBs hemsida.

Ett PLC-program består av logiska funktioner som beror på en eller flera ingångsvärden. Varje logisk funktion påverkar värdet på en eller flera utgångsanslutningar. PLC:t arbetar cykliskt genom att läsa av ingångsvärden, exekvera programlogiken och därefter uppdatera med nya utgångsvärden.

Ett PLC kan också vara anslutet till en fältbuss. Via fältbussen kan ett PLC hantera ingångs- och utgångsvärden från externa moduler med ingångs- och utgångsanslutningar. PLC:t kan också styra andra enheter som motorstyrningar eller industrirobotar som är kopplade till fältbussen.

7.6 CodeSys

Controller Development Systems (CodeSys) och är en utvecklingsmiljö för att utveckla PLC program inom den internationella industristandarden IEC 61131-3. [20] CodeSys har använts under examensarbetet för att programmera ABBs virtuella PLC.

7.7 PLCHandler

PLCHandler är ett mjukvaru API för styrsystem som tillhandahålls av företaget 3S som även utvecklar CodeSys. API:t kan användas för att få tillgång till värden på PLC:ts symboler och signaler från ett externt system.[21] ABBs virtuella PLC som används i examensarbetet implementerar detta API.

7.8 RobotStudio

RobotStudio är ett program från ABB som används för att simulera industrirobotar i en 3D miljö för att kunna testköra programmeringen av robotar i ett tidigt skede under ett projekt.

7.9 Expericom

Expericom är Ericssons egna mobila nätverk, vilket kommer att användas under utvecklingsarbetet av koncepttestet. Nätet är baserat på LTE standarden samt vidareutvecklingen LTE-Advanced.

8

Kravspecifikation

För att kommunicera med fabrikens fältbuss via LTE-nätverket behöver signalerna från styrsystemet översättas till rätt protokoll för motsvarande fältbuss. Konceptestet skall kunna ta emot signaler från enheterna på fältbussen och skicka vidare dessa signaler till ett virtuellt styrsystem i serverklustret.

Koncepttestet behöver bestå av:

- En hårdvarulösning som har stöd för både mobil kommunikation och anslutningar för den trådade fältbussen.
- En mjukvarulösning som kommunicerar med styrsystemet och vidarebefordrar signalerna till fältbussen.

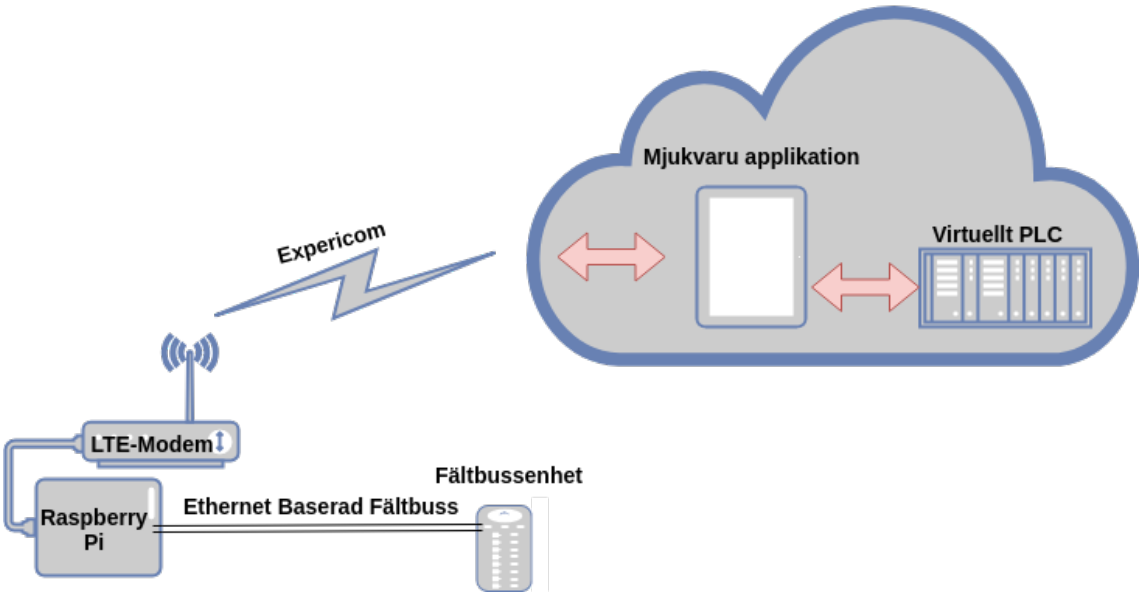
9

Genomförande

9.1 Systemöversikt

Systemet som utvecklades för koncepttestet består av en gateway baserad på en Raspberry Pi samt mjukvara som körs tillsammans med det virtuella PLC:t i ett serverkluster. Denna mjukvara implementerar både ett fältbussprotokoll samt ett API mot PLC:t. Mjukvaran har också till uppgift att mappa signalernas symbolnamn i PLC:t till motsvarande symbolnamn på fältbussen. Det virtuella PLC som används i koncepttestet är ett mjukvarubaserat PLC från ABB.

Figuren 9.1 beskriver systemets utformning. Molnet representerar en server som kördes i Ericssons serverkluster. Fältbussenheten är en enhet som kan kommunicera via det implementerade fältbussprotokollet tex en modul med ingångs- och utgångan-slutningar eller en industrirobot.



Figur 9.1: Översikt över det system som utvecklades för koncepttestet.

9.1.1 Val av fältbussprotkoll

Det fältbussprotokoll som i huvudsak används i ”Smarta Fabrikers” fabriksstation är PROFINET. PROFINET är också det fältbussprotokoll som är mest utbrett inom Europeisk industri. Mycket energi lades åt att få tag på en implementation av PROFINET men då det är ett propietärt protokoll var det svårt och innebar höga licensavgifter. Även fältbussprotokollet EtherNet/IP utvärderades men då ingen licensfri implementation av protokollet, som kunde agera som masternod, hittades valdes detta bort. Det fältbussprotokoll som istället valdes för koncepttestet var Powerlinkprotokollet. Powerlink valdes då både slav- och masterimplementationen är öppen källkod samt väldokumenterade vilket gjorde det smidigt att komma igång med.

9.1.2 Val av API för kommunikation med PLC

Det API som användes för att hämta ut värden på PLC:ts utgångssignaler samt ange värden för ingångssignaler var PLCHandler. PLCHandler är ett C/C++ API som tillhandahålls av företaget 3S och implementeras av ABBs virtuella PLC. Ett annat API som implementeras av ABBs virtuella PLC, och bygger på Windows Communication Foundation, utvärderades också. Detta API valdes bort till fördel för PLCHandler då det var implementerat i C#. Implementationerna av fältbussprotokollet och PLC:ts API behövde gå att användas i en och samma applikation. Eftersom implementationen av fältbussprotokollet som valdes ut var implementerad i C var PLCHandler det mest lämpliga alternativet att gå vidare med.

9.2 Hårdvarulösning

Som gateway mellan LTE-nätverket och fältbussen användes en Raspberry Pi model 3B med operativsystemet Arch Linux. Linuxkärnan som användes var version 4.4.29. Denna version användes för att kunna använda det LTE-modem som tillhandahölls av Ericsson då drivrutinerna till detta modem inte var kompatibelt med den senaste versionen av Linuxkärnan.

9.2.1 Kompilering av Linux kärnan

Linux kärnan som användes kompilerades i Ubuntu 17.10 med hjälp av korskompilatorn arm-linux-gnueabihf. Korskompilatorn hämtades hem med kommandot

```
git clone https://github.com/raspberrypi/tools din_filvag/tools.
```

Kod 9.1: Bashkommando för att hämta hem korskompilatorn till Raspberry Pi.

Sedan lades *din_filväg/tools/arm-bcm2708/arm-linux-gnueabihf/bin* till i systemets PATH-variabel. För att hämta hem källkodsfilerna för Linuxkärnan patchad för Raspberry Pi samt patchad med linux-rt patchen kunde följande kommando användas

```
git clone --depth=1 https://github.com/raspberrypi/linux -b rpi-4.4.y
```

Kod 9.2: Bashkommando för att hämta hem version 4.4 av Linuxkärnan.

För att kompilera Linuxkärnan så utfördes följande kommandon i mappen ”linux” som laddades ner i föregående steg:

```
KERNEL=kernel7
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
make -j 4 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

Kod 9.3: Bashkommandon för att kompilera Linuxkärnan

För att köra den kompilerade Linuxkärnan på en Raspberry Pi behövde filerna som kompileringen resulterade i kopieras över till det SD-kort som ska används utav Raspberry Pi:en för att starta upp. Efter att SD-kortet förts in i kortläsaren utfördes följande kommandon i Ubuntu 17.10 för att montera SD-kortet:

```
mkdir mnt
mkdir mnt/fat32
mkdir mnt/ext4
sudo mount /dev/sdb1 mnt/fat32
sudo mount /dev/sdb2 mnt/ext4
```

Kod 9.4: Bashkommandon för att montera SD-kort.

För att installera moduler samt kopiera över nödvändiga filer utfördes följande kommandon [22]:

```
sudo make ARCH=arm
    CROSS_COMPILE=arm-linux-gnueabihf-INSTALL_MOD_PATH=mnt/ext4
    modules_install
sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
sudo cp arch/arm/boot/dts/*.dtb mnt/fat32/
sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/
```

Kod 9.5: Bashkommandon för att kopiera den kompilerade Linuxkärnan till SD-kort.

9.2.2 Konfigurering av L2TP tunnel i Linux

Linux kommer med L2TP inbyggt och man behöver därför inte installera någon extra mjukvara för att kunna sätta upp en L2TP tunnel. Genom att utföra kommandon i ett terminalfönster så sattes det upp ett virtuellt Ethernet-interface som fungerar som L2TP tunnelns ändpunkt, se kod 9.6. Ethernet-ramar som skickas till detta interface kapslas in i ett UDP datagram som sedan skickas vidare till andra änden av tunneln. [23]

```
ip l2tp add tunnel tunnel_id 3000 peer_tunnel_id 4000 encap udp local
    <lokal IP-adress> remote <remote IP-adress> udp_sport 5000 udp_dport
    6000
ip l2tp add session tunnel_id 3000 session_id 1000 peer_session_id 2000
ip link set l2tpeth0 up mtu 1500
```

Kod 9.6: Bashkommandon för att sätta upp en L2TP-tunnel.

9.2.3 Brygga Ethernet-interface i Linux

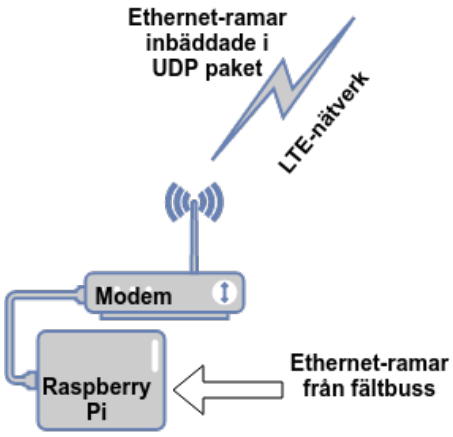
Med hjälp av en virtuell nätverksbrygga kunde de Ethernet-ramar som togs emot på det fysiska Ethernet-interfacet bryggas vidare till L2TP-tunnelns ändpunkt.

Kod 9.7 vissar de kommandon som utfördes i Linux för att skapa en brygga, br0, mellan två Ethernet-interface. Det fysiska Ethernet-interfacet motsvaras av eth0 medan det virtuella Ethernet-interfacet som fungerar som L2TP-tunnelns ändpunkt motsvaras av l2tpeth0. [23]

```
ip link add br0 type bridge
ip link set l2tpeth0 master br0
ip link set eth0 master br0
ip link set br0 up
```

Kod 9.7: Bashkommando för att sätta upp en Ethernetbrygga.

Detta innebar att de Ethernet-ramar som kom in på eth0-interfacet skickades vidare till l2tpeth0-interfacet. Och omvänt skickades de ramar som kom in på l2tpeth0 vidare till eth0. På så sätt kunde det upprättas en kommunikationskanal över LTE-nätverket där det var möjligt att kommunicera via vilket Ethernet-baserat protokoll som helst. Se figur 9.2.



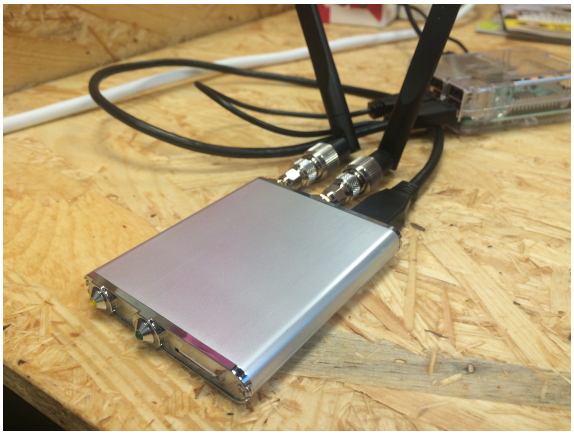
Figur 9.2: Skiss över bryggning av Ethernet-ramar mellan LTE-nätverket och fältbussen

9.2.4 LTE-modem

För att ansluta Raspberry Pi:en till LTE-nätverket så användes ett USB-modem som tillhandahölls av Ericsson. Modemet var ett LTE-Advanced modem EM7565 från Sierra Wireless, se figur 9.3. Det installerades genom att kompilera medföljande drivrutiner. Modemet behövde också läggas till i NetworkManager. I samband med det ställdes även specifika operatörsinställningar som Access Point Name (APN) in, se kod 9.8.

```
nmcli c add type gsm ifname ttyUSB2 con-name Expericom apn
    internet.erinet.se
```

Kod 9.8: Bashkommando för att lägga till modem hos NetworkManager samt ställa in APN-inställningar.



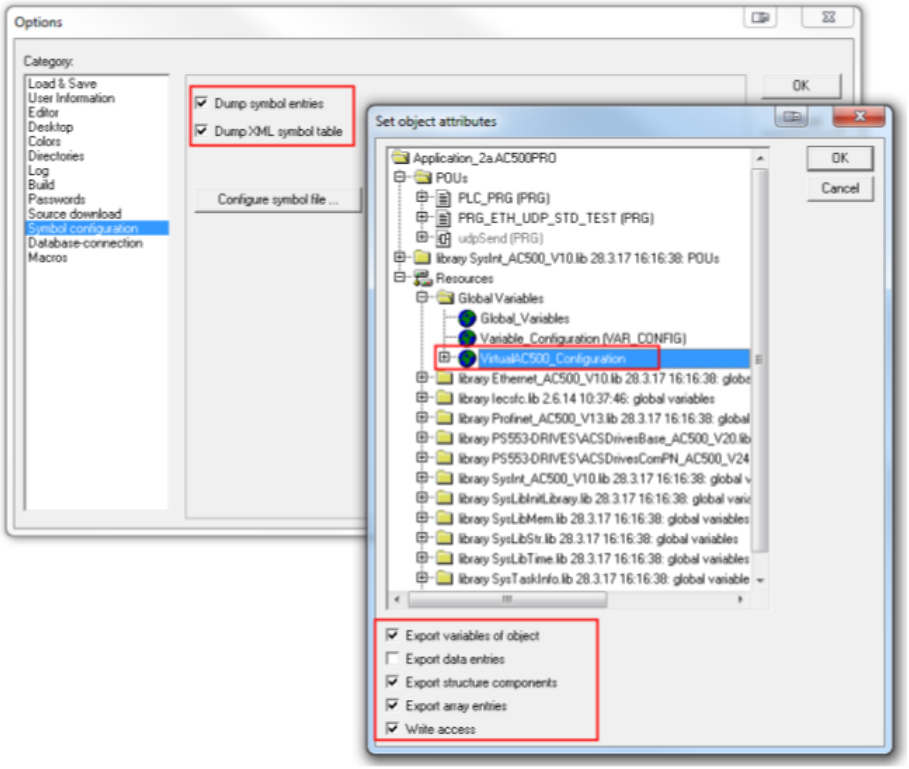
Figur 9.3: Figuren visar det LTE-modem som användes under examensarbetet.

9.3 Mjukvarulösning

Mjukvarulösningen som utvecklades till koncepttestet var en C-applikation som utvecklades i Microsoft Visual Studio. Den kördes tillsammans med det virtuella PLC:t i ett serverkluster och använde två stycken färdiga bibliotek för att underlätta hanteringen av Powerlink protokollet och kommunikationen med PLC:t.

9.3.1 CodeSys PLCHandler

PLCHandler är ett bibliotek och API från CodeSys som användes i mjukvaran för att kommunicera med PLC:t. PLCHandler gör det enkelt att kunna läsa och skriva till PLC-programmets ingångs- och utgångssymboler. Första problemet som stöttes på var att få kontakt med det virtuella PLC:t och extrahera symboler. Lösningen var att exponera de signaler som PLC:t använde. Detta behöver göras i CodeSys innan programmet laddas in i PLC:t. Signalerna exponerades genom att gå in på menyalternativet ”Project” -> ”Options” -> ”Symbol configuration” och välja inställningar enligt figur 9.4. Detta gjordes för varje symbol eller grupp av symboler som behövde exponeras för det externa systemet.



Figur 9.4: Figuren visar inställningarna som gjordes i CodeSys för att exponera PLC:ts signaler.

9.3.2 PLCCommunication

För att kunna få tillgång till fler datastrukturer som till exempel listor så skrevs ett funktionsbibliotek i C++ som i sin tur använde sig utav PLCHandler API:t. Detta funktionsbibliotek kallades PLCCommunication och exponerade en uppsättning funktioner som kunde användas i C-applikationen som knöt ihop kommunikationen med PLC:t och implementeringen av Powerlinkprotokollet.

9.3.3 I/O mappning

För att kommunikationen mellan PLCHandler och OpenPowerlink skulle fungera, implementerades funktionalitet för att mappa mellan symbolnamnen hos PLC:t och Powerlink. Detta för att rätt värde skulle skrivas till rätt symbol i PLC:t och att rätt värde skulle läsas av i andra riktningen. Detta löstes genom att använda en konfigurationsfil som innehöll information om hur symbolnamnen skulle mappas mot varandra. Konfigurationsfilen var skriven i XML och lästes in när programmet startade upp. I/O mappningen implementerades i ett funktionsbibliotek kallat IOMapping. Biblioteket skrevs i C++ för att få tillgång till datastrukturer som strängar och hashtabeller.

9.3.4 OpenPowerlink

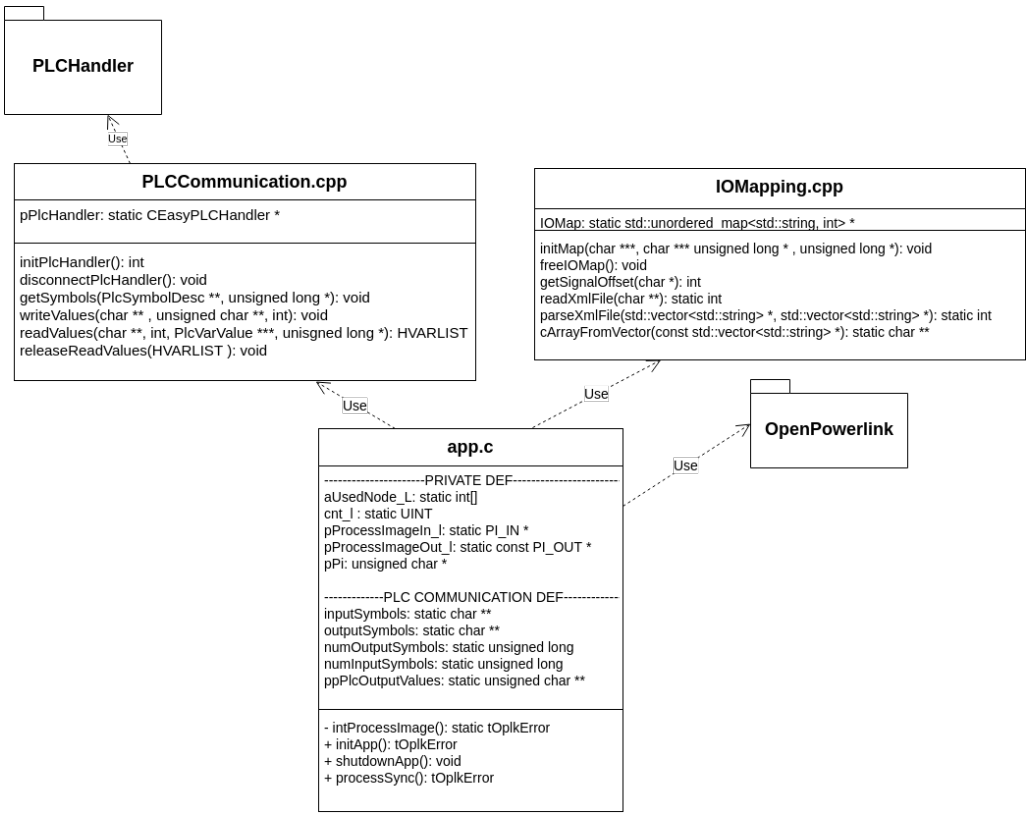
OpenPowerlinkimplementationen användes för att kunna agera som Powerlink masternod och hantera kommunikationen med slavnoderna i nätverket. Med OpenPowerlink biblioteket följde det ett antal demoapplikationer som exempel på hur det kunde användas. Dessa undersöktes för att förstå hur protokollet implementerats och hur biblioteket skulle användas. Demoapplikationerna fungerade som mall för att utveckla en C-applikation som implementerade Powerlink. Därefter vidareutvecklades C-applikationen med PLCCommunication biblioteket för att kunna kommunicera med det virtuella PLC:t. Innan powerlinkstacken kunde användas behövde biblioteket byggas från källkoden som hade laddats ner.

9.3.4.1 Powerlinkstacken

Under examensarbetet kompilerades powerlinkstacken i Windows. Konfigureringar och steg för installationen varierar beroende på operativsystem. Cmake behövdes för att kunna konfigurera och generera byggfilerna, och hämtades från deras officiella hemsida. [24] Powerlinkbiblioteket hämtades hem från OpenPowerlinks officiella centralkatalog på Sourceforge. [18]

9.3.5 Powerlink applikationen

PLCHandler var skriven i C++ och openPowerlink i C, vilket underlättade implementationen av APIerna i samma mjukvara då det är enkelt att blanda dom två språken. Under implementeringen av de två API:erna i huvudmjukvaran så uppstod ett antal länkningsproblem som löstes genom att titta på felmeddelanden från kompilatorn. Filen app.c var en av de ordinarie filerna från demoapplikationen från openPowerlink som vidareutvecklades med hjälp av funktionaliteten som skrevs i PLCCommunication och IOMapping. Figuren 9.5 beskriver hur de olika komponenterna är integrerade med varandra i mjukvaran.



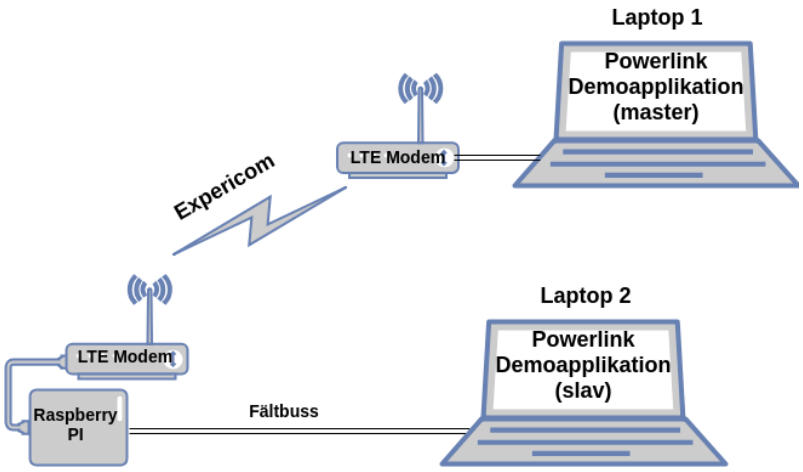
Figur 9.5: UML-Diagramet beskriver mjukvaruapplikationens struktur.

9.4 Genomförande av tester

Ett par tester gjordes för att undersöka vilken lägsta cykeltid som gick att uppnå med koncepttestet. ABBs mjukvarubaserade PLC användes inte under de här testerna. Detta då testerna syftade huvudsakligen på att mäta fältbussprotokollets cykeltid över LTE-nätverket.

9.4.1 Test mellan två laptops

Under ett första test av vilken lägsta cykeltid som går att uppnå med Powerlink så klarade systemet av en cykeltid på 75 ms innan kommunikationen blev ostabil. Testet utfördes med två stycken laptops samt en Raspberry Pi. Den första laptopen körde en Powerlink master demoapplikation och var uppkopplad till LTE-nätverket. Den andra laptopen körde en Powerlink slav demoapplikation och var kopplad till Raspberry Pi:en via ett Ethernet-nätverk. Raspberry Pi:en var kopplad till både LTE-nätverket via ett LTE-modem samt det trådade Ethernet-nätverket och fungerade som en brygga mellan de två nätverken. Mellan Raspberry Pi:en och den första laptopen användes en L2TP-tunnel för att transportera Powerlink-ramen över LTE-nätverket. Se figur 9.6.

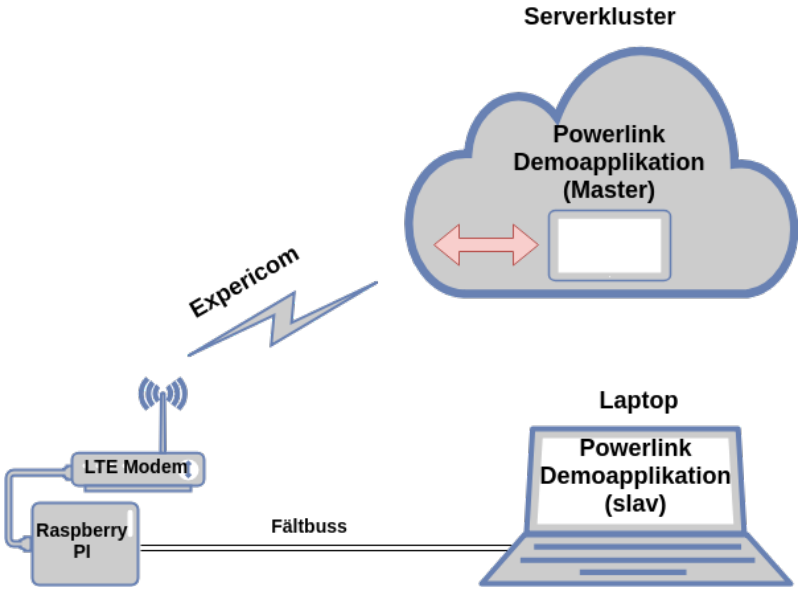


Figur 9.6: Skiss över systemuppsättning som användes under första testet.

Att kommunikationen till slut blev ostabil berodde på att responsen från slavnoden inte nådde fram i tid till masternoden. Anledningen till detta var att vi hade en fördröjning i kommunikationen på ca 30 ms. Detta uppmättes genom att utföra en ping från den ena noden till den andra.

9.4.2 Test med virtuell maskin

Andra testet utfördes med en laptop, en Raspberry Pi och en virtuell maskin placerad i ett serverkluster. Den virtuella maskinen agerade master med en Powerlink master demoapplikation. På laptopen kördes en Powerlink slav demoapplikation. På samma sätt som i första testet så var Raspberry Pi:en kopplad till både LTE-nätverket via ett LTE-modem samt det trådade Ethernet-nätverket. En L2TP-tunnel användes mellan den virtuella maskinen i serverklustret och Raspberry Pi:en för att transportera Powerlink-ramen över LTE-nätverket, se figur 9.7.



Figur 9.7: Skiss över systemuppsättning som användes under andra testet

Fördröjningen över LTE-nätverket uppmättes genom att utföra en ping vilket resulterade i ett medelvärde på 30 ms fram och tillbaka. Det vill säga cirka 15 ms fördöjning enkel väg.

Modemet som användes vid andra testet var ett Sierra Wireless EM7565 som är ett LTE-Advanced modem. Det innebär att det klarar av högre frekvensband än modemet vid första testet och därmed är kapabelt till högre överföringshastigheter. Modemet hade också två stycken antenner monterade på utsidan av modemet vilket ska påverka fördröjningen positivt. Lägsta stabila cykeltid som testades fram var 40 ms.

10
Resultat

Målet var att skapa ett koncepttest där ett virtuellt styrsystem kommunicerar med en industrirobot via mobilnätet. Med det system som utvecklats har vi skapat ett koncepttest som möjliggör kommunikation mellan ett virtuellt PLC från ABB och en godtycklig enhet på en Powerlinkfältbuss. Koncepttestet kan i sin nuvarande form inte kopplas ihop med en industrirobot från ABB. För att kunna detta behövs någon översättning mellan Powerlink och PROFINET eller Ethernet/IP utföras. Dessa två protokoll är de Ethernet-baserade fältbussprotokoll som ABBs industrirobotar för närvarande stödjer. Alternativt kan man integrera en Ethernet/IP eller PROFINET implementation i mjukvaran istället för nuvarande Powerlinkimplementation.

10.1 Ethernet-baserat fältbussprotokoll över LTE-nätverk

För att göra det möjligt att skicka Ethernet-ramar från ett Ethernet-baserat fältbussprotkoll över LTE-nätverket så användes en metod för tunnling. Tunnelprotokollet som användes var L2TP vilket paketerade Ethernet-ramarna i ett UDP-paket som i sin tur är möjligt att skicka via LTE.

10.1.1 Testresultat från tester mellan två laptops

Uppmätt fördröjning över LTE-nätverket under testerna var i genomsnitt ca 30 ms. Se testresultat i tabell 10.1, den fetstilade raden markerar lägsta uppnådda cykeltid. PRes timeout står för poll response timeout. Detta är en timeout i Powerlinkprotokollet som sätts på slavnodernas poll respons (PRes).

Cykeltid (ms)	PRes timeout (ms)	Fungerade (J/N)
120	40	J
100	40	J
50	10	N
50	10	N
80	30	N
80	35	N
90	30	J
90	20	J
85	20	J
80	20	J
70	20	N
75	**20**	**J**
75	15	N

Tabell 10.1: Testresultat från tester mellan två laptops.

10.1.2 Testresultat från tester med virtuell maskin

Uppmätt fördröjning över LTE-nätverket under testerna var i genomsnitt ca 15 ms. Se testresultat i tabell 10.2, den fetstilade raden markerar lägsta uppnådda cykeltid.

Cykeltid (ms)	PRes timeout (ms)	Fungerade (J/N)
40	20	N
40	15	N
30	15	N
50	15	J
50	10	J
50	5	J
45	10	N
35	5	N
45	5	J
50	2.5	J
50	1	J
40	5	J
39	5	N
38	5	N
40	**1**	**J**

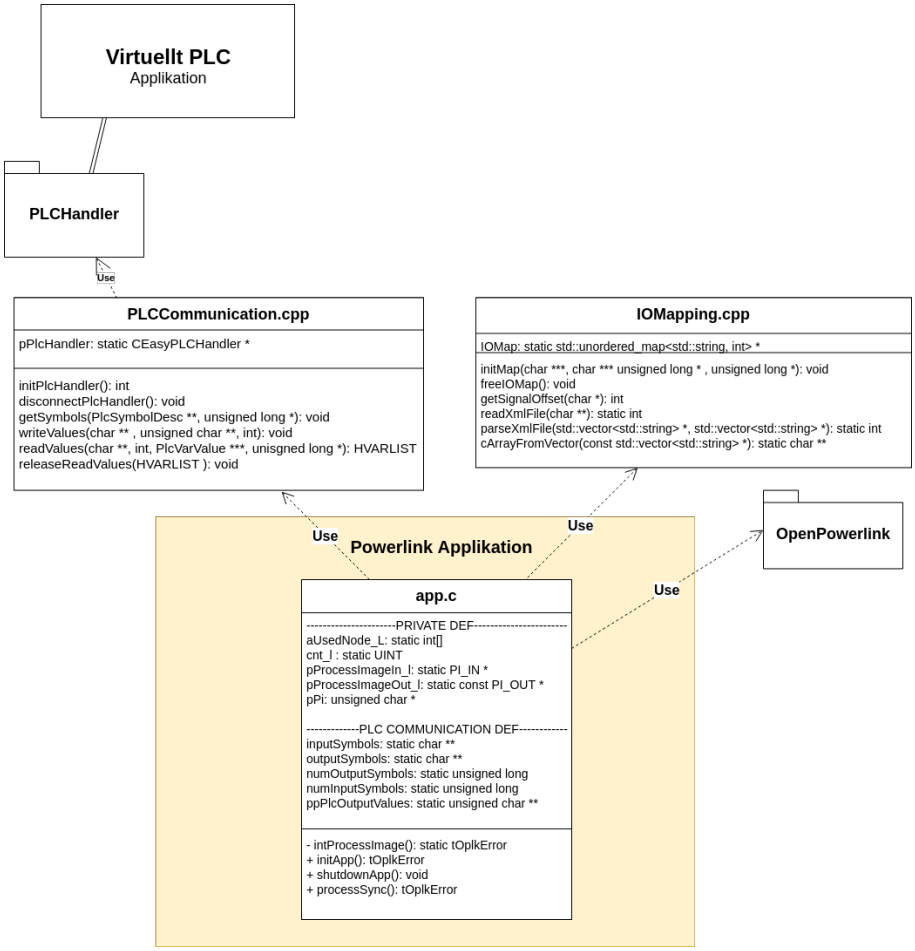
Tabell 10.2: Testresultat från tester med virtuell maskin.

10.2 Lägsta uppnådda cykeltid

Den lägsta cykeltid som uppnåddes under testerna uppgick till 40ms vilket kan ses i tabell 10.2.

10.3 Mjukvara

Mjukvaran resulterade i en Windowsbaserad applikation skriven i C och C++. Denna applikation kommunicerade med ett virtuellt PLC och Powerlink fältbussen via LTE nätverket. Figur 10.1 är en överblick över strukturen av mjukvaruapplikationen.



Figur 10.1: UML-Diagram och struktur över utvecklad mjukvara

10.3.1 PLCCommunication.cpp

Funktionen **getSymbols**, se kod 10.1, hämtar de exponerade symbolerna från PLC och lägger de i en lista, **symbolList**. Därigenom behövde inte symbolnamnen hårdkodas statiskt i programmet.

```
int getSymbols(PlcSymbolDesc ** symbolList, unsigned long * numOfSymbols)
}
    if (pPLCHandler->GetAllItems(symbolList, numOfSymbols) == RESULT_OK)
       return 0;
    return -1;
}
```

Kod 10.1: Funktionen getSymbols hämtar symbolnamn från PLC.

Funktionen **writeValues**, se kod 10.2, skriver nya värden till PLC:t med de symbolnamn som finns angivna i listan **symbols**.

```
void writeValues(char **symbols, unsigned char **values, int numOfVars)
{
   unsigned long ulStart = CAL_SysTimeGetMs();

   if (pPLCHandler->SyncWriteVarsToPlc(symbols, numOfVars, values) ==
       RESULT_OK) {};
      printf("Writing value successful: \%ld ms\\n", CAL_SysTimeGetMs() -
          ulStart);
   else
      printf("*** Writing value failed ***\\n");

}
```

Kod 10.2: Funktionen writeValues skriver värden till PLC.

readValues, se kod 10.3, läser av de värden som är angivna i **symbols** från PLC:t.

```
HVARLIST readValues(char **symbols, int numOfVars, PlcVarValue ***values,
    unsigned long * numReadValues)
{
   return pPLCHandler->SyncReadVarsFromPlc(symbols, numOfVars, values,
       numReadValues);
}
```

Kod 10.3: Funktionen readValues läser av värden från PLC.

10.3.2 IOMapping.cpp

IOMapping ansvarar för mappningen mellan symbolnamnen hos PLC:t och symbolnamnen hos Powerlinkbussen. Mappningen förkonfigurerades i en XML-fil, som lästes in vid uppstart, och sparades i en hashtabell. För varje symbol som ska mappas anges dess symbolnamn hos PLC:t respektive fältbussen samt en offset i Powerlinks ProcessImage. Powerlinks ProcessImage är en datastruktur som delas mellan Powerlinknoderna där symbolernas värden sparas. Den är specifik för Powerlink och skapas när fältbussen konfigureras upp i samband med att man installerar bussystemet. Genom att använda denna offset kan tilldelning av värden ske till symbolerna på Powerlink på ett dynamiskt vis utan att behöva känna till Powerlinkbussens symbolnamn i programmet.

Kod 10.4 beskriver hur en XML-fil med I/O mappningen kan se ut. Man kan exempelvis utläsa från första raden att PLC-symbolen **.DI0** har mappats mot **DigitalInput0** på Powerlinkbussen. Det framgår också att **DigitalInput0** har offset 0 i Powerlinkbussens ProcessImage.

```
<input plcSymbol=".DI0" fieldbusSymbol="DigitalInput0" PIOffset="0x00"/>
<input plcSymbol=".DI1" fieldbusSymbol="DigitalInput1" PIOffset="0x01"/>
<output plcSymbol=".DO0" fieldbusSymbol="DigitalOutput0" PIOffset="0x00"/>
<output plcSymbol=".DO1" fieldbusSymbol="DigitalOutput1" PIOffset="0x01"/>
```

Kod 10.4: Exempel på utformning av konfiguration av symbolmappning i XML-fil.

IOMapping hade en privat mappningstabell där mappningen mellan PLC:ts symbolnamn och offseten Powerlinkbussens ProcessImage låg sparad. Tabellen implementerades som en hashtabell med datastrukturen unordered_map från C++ STL-bibliotek, se kod 10.5. För att parsa XML-filerna användes tredjepartsbiblioteket rapidXML.

```
static std::unordered_map<std::string, int> *IOMap;
.
.
int getSignalOffset(char * IOName) {
   std::unordered_map<std::string, int>::iterator got =
       IOMap->find(IOName);
   if (got != IOMap->end()) {
      //return offset for found String value
      return got->second;
   }
   return -1;
}
```

Kod 10.5: Funktionen getSignalOffset hämtar offset för en specificerad symbol.

10.3.3 Powerlinkapplikationen

App.c från openPowerlinks demo applikation vidareutvecklades för att använda sig utav PLCCommunication och IOMapping. Syftet var att sy ihop kommunikationen mellan det virtuella PLC:t och slavnoder på Powerlinkbussen.

Kod 10.6 visar initieringsfasen av applikationen. Här sker bland annat minnesallokering för datastrukturer och initieringsfunktionerna för biblioteken.

```
 .
 .
// Init digital outputs in process image to 0
 pPi = (unsigned char *)pProcessImageIn_1;
 for(unsigned int i = 0; i < COMPUTED_PI_IN_SIZE; i++)
     *(pPi+i) = 0;

// INIT PLC COMMUNICATION
initPlcHandler();

initIOMap(&inputSymbols, &outputSymbols, &numInputSymbols,
    &numOutputSymbols);

ppPlcOutputValues = (unsigned char **)malloc(numInputSymbols *
    sizeof(unsigned char *));
for (unsigned int i = 0; i < numInputSymbols; i++) {
   ppPlcOutputValues[i] = (unsigned char *)malloc(sizeof(unsigned
       long));
}
 .
 .
```

Kod 10.6: Initieringsfasen av programmet. Utdrag ur kodfil app.c.

Kod 10.7 visar exekveringsfasen som upprepas varje powerlinkcykel. Processen läser av powerlinkbussen och skriver värden till PLC:t. Därefter läses de nya utgångsvärdena av från PLC och skrivs till powerlinkbussen.

```
 .
 .
 /* Update values to PLC from Processimage */
pPi = (unsigned char *)pProcessImageOut_l;

for (unsigned int i = 0; (i < numInputSymbols) && (i <
    COMPUTED_PI_OUT_SIZE); i++)
{
   int offset = getSignalOffset(inputSymbols[i]);
   if (offset == -1) {
      log_error("Couldn't find symbol \%s in I/O map", inputSymbols[i]);
   }
   else {

      * (unsigned long *) ppPlcOutputValues[i] = *(pPi + offset);
   }
}

writeValues(inputSymbols, ppPlcOutputValues, numInputSymbols);

/* Read output values from PLC */
PlcVarValue **ppValues;
unsigned long ulNumOfValues = 0;
HVARLIST hList = readValues(outputSymbols, numOutputSymbols,
    &ppValues, &ulNumOfValues);

/* Update Processimage with new values from PLC */
pPi = (unsigned char *)pProcessImageIn_l;

for (unsigned int i = 0; (i < ulNumOfValues) && (i <
    COMPUTED_PI_IN_SIZE); i++)
{
    int offset = getSignalOffset(outputSymbols[i]);
   if (offset == -1) {
      log_error("Couldn't find symbol \%s in I/O map",
          outputSymbols[i]);
   }
   else {
      *(pPi + offset) = ppValues[i]->byData[0];
   }
}
.
.
```

Kod 10.7: Exekveringsfasen under en kommunikationscykel. Utdrag ur kodfil app.c.

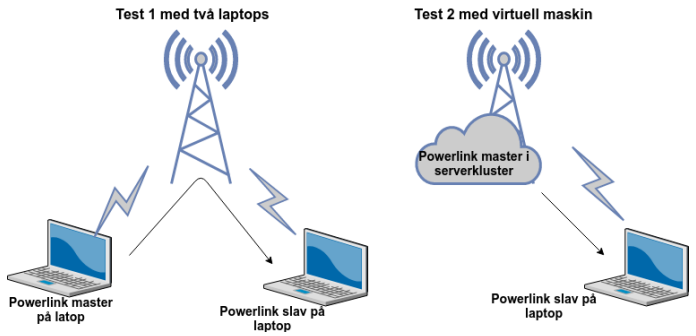
11

Slutsats

I brist på tid lyckades vi inte att skapa ett koncepttest där vi kunde styra en av ABBs industrirobotar via LTE-nätverket. Vi gjorde ett val att gå vidare med Powerlink istället för PROFINET då det var svårt att få tag på en implementation av PROFINET eller Ethernet/IP utan att betala dyra licensavgifter. Genom att använda Powerlink så kunde vi snabbare få upp ett koncepttest med ett virtuellt PLC som kommunicerar över en fältbuss. Genom att simulera en fältbussenhet med en laptop kunde vi verifiera att systemkonstruktionen fungerade. För att kommunicera med en industrirobot hade det behövts mer tid till att få fram en implementation av antingen PROFINET eller Ethernet/IP.

Lösningen för att transportera paket från fältbussprotkollet över LTE-nätverket inkapslade i UDP-paket fungerade bra. Tills dess att det finns ett fältbussprotokoll helt baserat på IP-trafik eller på något annat sätt kompatibelt med LTE eller framtidens telekommunikationstekniker så är detta en tillsfredställande lösning. En nackdel med L2TP var att det var komplext att sätta upp i Windowsmiljö.

Det var stor skillnad i cykeltid mellan första och andra testet. Under det andra testet hade vi ett bättre modem, men skillnaden berodde främst på att vi flyttade upp det virtuella PLC:t i serverklustret. Serverklustret var inkopplat på en central plats på LTE-nätverket vilket gjorde att en Powerlink-ram enbart behövde färdas en väg över LTE-nätverket för att komma fram. Under det första testet behövde varje Powerlink-ram färdas två gånger över LTE-nätverket för att komma fram till sin destination. Se figur 11.1 för att se en jämförelse över hur Powerlink-ramarna färdades över LTE-nätverket beroende på testuppsättning.



Figur 11.1: Figuren visar skillnaden i vilken väg över LTE-nätverket som Powerlink-ramarna färdades beroende på vilken testuppsättning som användes.

Att Powerlink-ramarna bara behövde färdas en gång över LTE-nätverket i den andra testuppsättningen förklarar varför fördröjningen halverades. Utifrån detta drar vi slutsatsen att att det virtuella PLC:t alltid måste vara placerat på en central plats i LTE-nätverket. Annars kommer cykeltiden att mer eller mindre dubbleras.

Den bästa cykeltid som uppnåddes under testerna på 40 ms uppnåddes enbart med en slavnod på fältbussen. Varje slavnod som läggs till i Powerlink innebär ytterligare en uppsättning med poll-request samt poll-response meddelanden som ska utbytas. Man kan dra slutsatsen att den lägsta möjliga cykeltiden ökar snabbt i ett verkligt scenario med flera slavnoder tillkopplade på fältbussen. Detta är långt ifrån de 10ms som man vill ha inom en fabriksstation i tillverkningsindustrin.

I ”Smarta Fabrikers” fabriksstation finns det 20 stycken slavnoder på PROFINET-bussen. [25] För att hålla en cykeltid under 10ms skulle en mycket lägre fördröjning per skickat paket behövas om man skulle använda sig utav vår lösning. Men det behövs vidare testning och analys för att komma fram till vilken fördröjning som är tillräcklig. Vi drar slutsatsen att den lösning vi konstruerat är otillräcklig i förhållande till de realtidskrav som ställs av tillverkningsindustrin på kommunikationen inom en fabriksstation. Mobila nätverk lämpar sig bättre för kommunikation med högre cykeltid eller där det inte finns några hårda realtidskrav på samma sätt som i ett fältbussprotokoll. Exempel på sådan kommunikation benämns som kategori A i figur 6.1.

12
Diskussion

Den lägsta cykeltiden som uppnåddes var högre än vad vi hade väntat oss. Det är möjligt att vi hade kunnat få till en något bättre cykeltid om det funnits tid till att konfigurera LTE-nätverket bättre. Då hade vi kanske kunnat få ner fördröjningen ytterligare. I slutsatsen kom vi fram till att det troligtvis krävs en fördröjning under 1ms för att åstadkomma de cykeltider som krävs inom en fabriksstation i tillverkningsindustrin. Dagens LTE-nätverk kan inte tillgodose den typen av fördröjningar men framtidens 5G-nätverk kommer att tillhandahålla mycket lägre fördröjningstider på ned till 1ms. Det återstår att se exakt vilka fördröjningstider 5G kommer att leverera men det ser ut som att det kan bli svårt att använda även 5G i syfte att styra exempelvis fabriksstationer. Trots detta tror vi att det finns stor potential för 5G inom industrin. Det finns flera nivåer inom fabrikskommunikationen och användningsområden där 5G-nätverk kan lämpa sig väl.

12.1 Tillämpningsområden för 5G inom industrin

Vi drog slutsatsen utifrån testresultaten i examensarbetet att kommunikation med realtidskrav som benämns som kategori C och B, se avsnitt 6.4, kommer att bli väldigt svårt med mobila nätverk. Däremot tror vi att kommunikation inom kategori A, där man accepterar cykeltider upp till 100 ms, skulle kunna ersättas med mobil kommunikation. Detta kan inkludera kommunikation mellan olika fabriksstationer inom en fabrik eller kommunikation mellan fabriksstationer och överordnade system. Det kan också innebära kommunikation mot operatörspaneler för övervakning. Det finns även andra typer av industri än tillverkningsindustrin, exempelvis processindustrin, där kommunikationen ofta inte behöver vara lika snabb.

12.1.1 Processautomation

Processautomation syftar på styrning av processer inom till exempel den petrokemiska industrin. Den typen av processer är oftast långsammare än processer inom tillverkningsindustrin och ställer inte lika höga krav på korta cykeltider. Detta gör att ett mobilt nätverk kan lämpa sig bättre inom processindustrin än inom tillverkningsindustrin.

12.1.2 Uppkopplade komponenter för övervakning

Inom industri 4.0 så pratas det bland annat om ”predictive maintenance” vilket innebär att givare för övervakning av komponenter är uppkopplade till överordnade system. Då kan man samla in data från komponenter i fabriken i syfte att diagnostisera och försöka förutse när det är dags för underhåll. Ett exempel är uppkopplade elmotorer som övervakar frekvensen hos vibrationerna i motorn för att varna när det är dags att byta lager.

Eftersom det rör sig om ett stort antal givare om varje komponent ska vara övervakad kommer det underlätta om dessa kan kopplas upp trådlöst. Dessutom kommer inga hårda realtidskrav att ställas på den här typen av kommunikation då det enbart handlar om diagnostisering. Därför tror vi att detta kan vara ett framtida användningområde för 5G inom industrin.

12.1.3 Automated guided vehicles

Inom industri 4.0 så spås det att så kallade Automated Guided Vehicle (AGV) kommer att ersätta transportband i större grad än idag. En AGV är en självkörande truck eller transportplattform som kan transportera material och produkter inom en fabrik, se figur 12.1. Eftersom de är mobila och trådlösa i sin natur finns det ett stort behov av ett stabilt trådlöst kommunikationssystem.

Figur 12.1: Ett exempel på AGV:er som transporterar halvfärdiga traktorer till nästa fabriksstation för vidare bearbetning. Bilden är tagen ifrån AGV-tillverkaren Red Vikings officiella hemsida.

Då AGV:er hela tiden måste kommunicera med omgivande fabriksstationer så tror vi att uppkopplade AGV:er kommer att vara ett av de första användningsområdena för 5G inom tillverkningsindustrin.

12.2 Trådlöst fältbussprotokoll

Det blev tydligt under examensarbetet att det inte finns något standardiserat trådlöst fältbussprotokoll som kan användas med LTE-nätverk. Detta löstes i examensarbetet med en L2TP-tunnel men för att mobilt nätverk ska vara ett verkligt alternativ

behövs ett fältbussprotokoll som är kompatibelt med de mobila nätverken. Det hade varit intressant att utforska om det går att skapa ett protokoll ett IP-baserat fältbussprotokoll. Det vill säga ett protokoll baserat på IP-stacken som kan användas för att överföra sensordata och kan tillhandahålla determinism. Detta protokoll skulle kunna vara helt kompatibelt med alla nätverk där man kan skicka IP-trafik, som exempelvis ett LTE-nätverk eller framtida 5G-nätverk. Alternativt skulle man kunna utforska om det går att utveckla ett fältbussprotokoll som är baserat på 5G.

12.3 Cybersäkerhet

Ur ett cybersäkerhetsperspektiv är LTE-nätverk och framtida 5G-nätverk ett bra val för trådlös industriell kommunikation då det finns inbyggda mekanismer i protokollen som skyddar från att obehöriga får tillgång till den överförda datan. Dessutom är den överföda datan i sig krypterad om någon ändå skulle lyckas få tillgång till datan.[26]

Cybersäkerheten är ett viktigt perspektiv eftersom att många av de företag som vill använda trådlös kommunikation inom fabriksprocessen också kan vara utsatta för industriellt spionage. En annan risk är intrång där förövaren manipulerar delar av fabriksprocessen. Ett scenario man kan tänka sig är att förövaren ändrar parametrar eller inställningar på ett subtilt sätt utan att det först märks men påverkar kvaliteten på de tillverkade produkterna. Ett annat scenario är där förövaren är ute efter att sabotera eller skada fabriken. Förutom ekonomisk skada kan ett sådant angrepp även innebära en säkerhetsrisk för operatörer och montörer som rör sig runt fabriksstationerna. Ett verkligt exempel är viruset Stuxnet som skapades specifikt för att attackera ett Siemens PLC och användes för att slå ut och förstöra en av Irans anrikningsanläggningar av uran.

12.4 Mjukvarubaserade PLC

Under examensarbetet använde vi oss utav ett virtuellt PLC från ABB och vi tror att mjukvarubaserade PLC:er har en stor potential för framtiden. Även om fabrikskommunikationen inte skulle vara trådlös skulle man kunna centralisera fabrikens olika PLC till ett serverkluster centralt placerat i fabriken. Detta serverkluster skulle kunna tillhandahålla mycket mer beräkningskraft än dagens PLC:er. Man kan tänka sig ett serverrum med flera terraflopp i beräkningskapacitet tillgängliga för att exekvera programlogiken eller göra mer avancerade reglertekniska beräkningar.

Till fordonsindustrin har företag som Nvidia utvecklat specialdesignade kretsar för självkörande bilar med 8 terraflopp för att hantera stora mängder sensordata och applicera deep-learning algoritmer på denna. ABBs PLC AC500 PM595 är snarlikt det som används i ”Smarta Fabrikers” fabriksstation och har en klockhastighet på 1,3GHz. Med tillgång till bara en terraflopp i beräkningskapacitet skulle det finnas tillräckligt för att ersätta åtskilliga PLC:er av den typen. Med en specialde-

signad krets med 8 terraflopp som är utvecklad för att göra snabba exekveringar utav grindlogik samt flyttalsberäkningar för reglertekniska beräkningar skulle man kunna centralisera ett stort antal PLC.

Eftersom det inte kommer att krävas någon specialdesignad hårdvara som är fukt-, vibrations- och dammtålig eller explosionsklassad kan det finnas stora pengar att spara i utvecklings- och tillverkningskostnader. Det kan även bli enklare att underhålla och övervaka styrsystemen när de är centraliserade till en och samma plats.

12.5 Miljö och hållbarhet

Uppkopplade fabriker kan innebära en minskad miljöpåverkan. Med till exempel ”predictive maintenance” som förutser när komponenter behöver service kan fel åtgärdas redan innan komponenten går sönder och behöver bytas ut helt. Genom att kunna analysera underhållsbehovet bättre så kan företag undvika plötsliga stopp i produktionen. På så sätt kan man minska oplanerade stopptider och få en högre produktivitet vilket leder till en högre resurseffektivitet.

Genom att koppla upp och samla in mer information från givare i fabriken kan miljöpåverkan kartläggas och undvikas på ett mer effektivt sätt.

12.6 Vidare utveckling

Om det hade funnits mer tid hade vi implementerat PROFINET-protokollet istället för Powerlink-protokollet. Då hade vi kunnat få en lösning som är kompatibel i större grad med existerande fabrikskomponenter. Men i förlängingen hade det varit intressantare att undersöka om det går att ta fram ett IP-baserat fältbussprotokoll som kan tillhandahålla determinism. Då skulle man kunna få ett protokoll som är helt kompatibelt med LTE- och 5G-nätverk utan att göra sig beroende av ett specifikt medium.

Det skulle även kunna vara intressant att undersöka om det går att mjuka upp vissa av de hårda realtidskrav som finns. Då skulle man exempelvis kunna titta på att skapa en industrirobot som kan hantera att deadlines i kommunikationen med styrsystemet missas ibland.

Referenser

[1] H. Lasi, P. Fettke, H. G. Kemper, T. Feld och M. Hoffmann, ”Industry 4.0”, *Business and Information Systems Engineering*, årg. 6, nr 4, s. 239–242, 2014. DOI: 10.1007/s12599-014-0334-4.

[2] *Smarta Fabriker*. URL: http://www.smartafabriker.se/ (hämtad 2018-06-06).

[3] S. K. Sen, ”I/O BUS NETWORKS”, i *Fieldbus and Networking in Process Automation*, 1. utg., sen2014: CRC Press, 2014, kap. 4.3, s. 71–72, ISBN: 978-1-4665-8677-2.

[4] R. Johansson och R. Snedsbøl, *Realtidssystem för högskolans ingenjörsutbildningar*, 8. utg. 2004, s. 146, ISBN: 91-89280-21-0.

[5] S. K. Sen, ”Fieldbuses”, i *Fieldbus and Networking in Process Automation*, 1. utg., CRC Press, 2014, kap. 5, s. 81, ISBN: 978-1-4665-8677-2.

[6] B. M. Wilamowski och J. D. Irwin, ”Industrial Ethernet as the “Silver Bullet” for Future Process Automation Communication Needs”, i *Industrial Communication Systems*, 2. utg., CRC Press, 2011, kap. 25.5, ISBN: 9781315218434.

[7] F. Lamb, ”Programmable Logic Controllers (PLCs)”, i *Industrial Automation: Hands-On*, 1. utg., McGraw-Hill Education, 2013, kap. 3.1.3, ISBN: 9780071816458.

[8] F. Klasen, V. Oestreich och M. Volz, *Industrial Communication with Fieldbus and Ethernet*. Vde Verlag Gmbh, 2011, s. 252–255, ISBN: 978-3-8007-3358-3.

[9] S. K. Sen, ”Communication hierarchy in factory automation”, i *Fieldbus and Networking in Process Automation*, 1. utg., CRC Press, 2014, kap. 4.2, s. 69–71, ISBN: 978-1-4665-8677-2.

[10] *LTE*. URL: http://www.3gpp.org/technologies/keywords-acronyms/98-lte (hämtad 2018-06-15).

[11] C. Metz, ”A Pointed Look at the Point-to-Point Protocol”, *IEEE Internet Computing*, årg. 3, nr 4, s. 85–88, juli 1999. URL: http://dx.doi.org/10.1109/4236.780964.

[12] W. Stallings, *Data and computer communications*, English, 8. utg. Upper Saddle River, N.J: Pearson Prentice Hall, 2007, s. 42–43, ISBN: 9780132433105;

[13] B. M. Wilamowski och J. D. Irwin, ”PROFINET Introduction”, i *Industrial Communication Systems*, 2. utg., CRC Press, 2011, kap. 40.1, ISBN: 9781315218434.

[14] *ODVA - EtherNet/IP Overview*. URL: https://www.odva.org/Technology-Standards/EtherNet-IP/Overview (hämtad 2018-06-06).

[15] B. M. Wilamowski och J. D. Irwin, ”Ethernet POWERLINK”, i *Industrial Communication Systems*, 2. utg., CRC Press, 2011, kap. 39, ISBN: 9781315218434.

[16] *Carrier Sense Multi-Access/Collision Detection (CSMA/CD) - Cisco Systems*. URL: https://www.cisco.com/en/US/tech/tk389/tk214/tk125/tsd_technology_support_sub-protocol_home.html (hämtad 2018-06-06).

[17] J. Baumgartner och S. Schoenegger, ”POWERLINK and Real-Time Linux: A Perfect Match for Highest Performance in Real Applications”, s. 1–7, 2010.

[18] *openPOWERLINK*. URL: http://openpowerlink.sourceforge.net/ (hämtad 2018-06-06).

[19] *Svenska datatermgruppen - Gateway, bridge*. URL: http://www.termado.com/DatatermSearch/?ss=Gateway (hämtad 2018-06-06).

[20] *CODESYS Development System - CODESYS*. URL: https://www.codesys.com/products/codesys-engineering/development-system.html (hämtad 2018-06-06).

[21] *CODESYS PLCHandler – Compact software interface (API) - CODESYS*. URL: https://www.codesys.com/products/codesys-runtime/plchandler.html (hämtad 2018-06-06).

[22] *Kernel building - Raspberry Pi Documentation*. URL: https://www.raspberrypi.org/documentation/linux/kernel/building.md (hämtad 2018-06-06).

[23] *ip-l2tp linux command man page*. URL: https://www.commandlinux.com/man-page/man8/ip-l2tp.8.html (hämtad 2018-06-06).

[24] *CMake*. URL: https://cmake.org/ (hämtad 2018-06-06).

[25] W. Petersson och D. Zöögling, *Elkonstruktion för projekt Smarta Fabriker*, 2017.

[26] D. Fang, Y. Qian och R. Q. Hu, ”Security for 5G Mobile Wireless Networks”, *IEEE Access*, årg. PP, nr 99, s. 1, 2017, ISSN: VO - PP. DOI: 10.1109/ACCESS.2017.2779146.