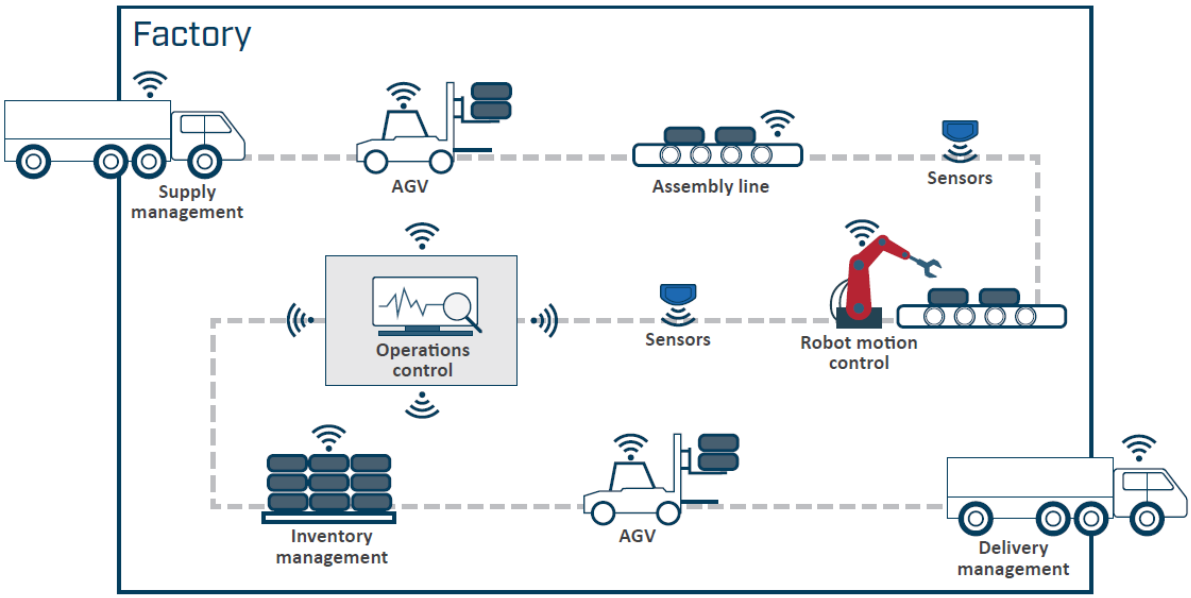
CHALMERS



Trådlös styrning av industrirobot via 5G

Examensarbete i högskoleingenjörsprogrammet Mekatronik

TIMOTEI VASIU
ANDRÉ DA SILVA GONÇALVES

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2019

EXAMENSARBETE

Trådlös styrning av industrirobot via 5G

TIMOTEI VASIU
ANDRÉ DA SILVA GONÇALVES

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2019

Trådlös styrning av industrirobot via 5G
TIMOTEI VASIU
ANDRÉ DA SILVA GONÇALVES

Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

Omslag:
Framtidens fabrik (5G-ACAI, 2018)

Chalmers Reproservice
Göteborg, Sverige 2019

Trådlös styrning av industrirobot via 5G

TIMOTEI VASIU
ANDRÉ DA SILVA GONÇALVES
Institutionen för Elektroteknik, Chalmers tekniska högskola

Examensarbete

Sammanfattning

På senare år har framsteg inom nätverksteknik, robotteknik och molntjänster öppnat upp nya möjligheter för lättare uppbyggnad och förbättrad funktionalitet av så kallade 'smarta' fabrikstationer. Ytterligare en viktig utveckling inom detta område vore att kunna "klippa kablarna", det vill säga ersätta de traditionella fältbussarna inom industrin med ett trådlöst kommunikationssystem. Effekterna blir då en smartare och mer flexibel fabrik som gör det enklare att installera nya enheter och att bevaka samtliga processer.

Utgångspunkten för detta examensarbete är tidigare studier inom ämnet där studenter vid Chalmers skapade ett underlag för kommunikation mellan ett virtuellt *programmerbart styrsystem* (PLC) och en industriell robot via det mobila nätverket, LTE. Föregående examensarbete redovisade ett delvis lyckat koncepttest, men sedan dess har teknologierna utvecklats en hel del. Den nya generationen av mobila nätverk, 5G, förväntas öka prestationer genom att medföra högre bandbredd samt avsevärt lägre fördröjningstider. Detta möjliggör bland annat "realtidskontroll" av industriella robotar, vilket är ett område väl värt att utforska. I detta sammanhang är en maximal *cykeltid* av 10 ms ett viktigt krav.

En mjukvaruapplikation skapades för att kunna skicka styrsignaler fram och tillbaka från den virtuella PLCn *VAC500*, till en simulerad robot där programvaran *RobotStudio* från ABB användes. *WebSockets* användes som metod för koppling från ena änden till den andra, istället för att använda proprietära fält-bussprotokoller. Den här metoden erbjuder 'full-duplex' tvåvägskommunikation utan behov av konstant efterfrågning. Testerna som genomfördes bestod i att skicka en styrsignal via 5G till en robot samt registrera kvittens. En genomsnittlig cykeltid på 238 ms kunde uppnås mellan klient och server. Genomsnittlig cykeltid på hela systemet blev 563 ms. Dessa resultat är begränsade specifikt till systemet som testades och tillägg i form av ökande antal givare och don kommer ha en negativ påverkan på den slutgiltiga cykeltiden.

Nyckelord: 5G, IIoT,Industri 4.0, TCP/IP-sockets, Trådlös kommunikation

Abstract

Recent advances in networking technology, robotics and cloud computing have the potential to facilitate easier implementation of a smart factory station as well as offer improved functionality. This is achieved by "cutting the cables" and replacing the traditional fieldbus arrangements with a wireless communication system, making the factory smarter and more flexible, making it easier to install new units and with the added benefit of being able to keep metrics on all the processes.

The background for this thesis is the previous research done on this subject where a proof of concept was created to allow communication between a virtual *Programmable Logic Controller* (PLC) and an industrial robot wirelessly over an LTE mobile network. This research showed only moderate success but the technological landscape has subsequently improved and the development of 5G communication brings with it performance gains, both in the high bandwith and the low response times it offers making "real-time" control of an industrial robot a possibility worth investigating. A maximum *cycle-time* of 10 ms is required for any such processes.

A software application was created to be able to send control signals from the virtual PLC *VAC500* to a simulation software by the name of *RobotStudio* and vice versa. *Websockets* were chosen as the method of connecting from endpoint to endpoint instead of any of the available fieldbus protocols. This method offers full duplex communication without the need for polling. Tests were carried out via 5G between a client and server as well as where an input was sent to the robot and an output was received by the PLC. An average cycle-time of 238 ms and 563 ms was achieved for the client/server connection and the whole system respectively. These results are only applicable to a system of a similar size, increasing the number of sensors and actuators will make synchronization more difficult, negatively impacting the recorded cycle-time.

Förord

Vi vill tacka ABB och Ericsson för att fått möjligheten att genomföra detta intressanta examensarbete. Ett särskilt tack till ”SmartaFabriker” som gav oss möjlighet att resa till Hannovermässan 2019, vilket verkligen var en givande upplevelse. Ett tack till Lars Egeland och Fredrik Flyrin från Ericsson som ordnat möten och hjälpt oss att komma i kontakt med rätt personer respektive handlett oss. Ett tack till Magnus Seger och Oskar Henriksson från ABB som har gett oss feedback under arbetetsgången. Ett stort tack till Jesper Pedersen som handlett och gett oss värdefull feedback under rapportskrivandet samt även vår examinator Tommy Svensson. Vi vill också tacka Rasmus Lindy och Oscar Hall som gav oss insiktsfulla råd i början av arbetet.

André Da Silva Gonçalves, Timotei Vasiu, Göteborg, 2019

I would also like to thank my beloved wife Maria who, with her constant love and support throughout this journey of mine, has given me the strength to follow my dream.

André Da Silva Gonçalves, Göteborg, 2019

I wish to express my sincere thanks to my precious wife, Camelia and my family for their support and encouragement over the past three years. Without them I would have never reached this far.

Timotei Vasiu, Göteborg, 2019

Förkortningar

3GPP	Generation Partnership Project.
AGV	Automated Guided Vehicle.
IIoT	Industrial Internet of Things.
PLC	Programmable Logic Controller.
REST	Representational State Transfer
RWS	Robot Web Sercives
NR	New Radio
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Figurer

Innehåll

1

Inledning

1.1 Bakgrund

Sedan industriella revolutionen har vissa milstolpar ansetts vara så viktiga att de ansetts vara en 'ny' industriell revolution. Efter tre iterationer ligger tekniken vid tröskeln till en fjärde iteration av den industriella revolutionen, Industri 4.0 eller Industrial Internet of Things (IIoT). Detta innebär att de flesta industrienheterna kommer att ha en ständig anslutning till nätverksinfrastruktur, vilket kommer att möjliggöra kommunikation mellan olika enheter i industrin på högre grad. Detta är speciellt relevant i samband med industriella processer som är kända för att vara komplexa och de mest krävande där hög precision, exakt synkroniseringen av enheter och storskalig koordination behövs. Det är också tänkt att de flesta industrienheter kommer kunna ge ständig återkoppling som kan leda till mer effektivt förebyggande underhåll. Detta anses leda till mer flexibla, robusta och effektiva fabriker som i högre grad styr sig självt[1].

Projektet Smarta Fabriker är ett exempel på en smart och uppkopplad fabrik som är för tillfället byggd hos Ericsson - ett av de största företag i kommunikationsteknik - på Lindholmen i Göteborg. Fabriken har som mål att öka attraktiviteten för tekniken och sprida kunskap om Industri 4.0. ABB är en stor aktör inom automationsteknik och har flera robotar i fabriken. Detta examensarbete utförs i samarbete med ABB och Ericsson samt "SmartaFabriker".

Ett område där möjligheter för vidare utveckling av fabriken har identifierats är där molnbaserad virtuella styrenheter kan användas istället för det traditionella Programmable Logic Controller (PLC). Med ett virtuellt PLC i molnet kommer den fysiska trådbundna kommunikationen via fältbussar ersättas av trådlös teknik så att styrsystemet på det sättet kan centraliseras, dvs 'ett' PLC är tillräckligt för att styra i princip alla enheter i en anläggning. Tidigare examensarbete inom molnbaserad virtuell PLC som utfördes under 2018 visade att det är möjligt att upprätta en anslutning via vanligt mobilt LTE-nätverk genom att översätta styrsignalerna till något industriellt fältbussprotokoll. Det resulterade dock i relativt långa överföringstider som inte riktigt uppnår de realtidskrav som krävs i en fabrikstation. Med 5Gs uppkomst kan däremot överföringstiden minskas avsevärt till följd av höga överföringshastigheter i kombination med låga svarstider[2].

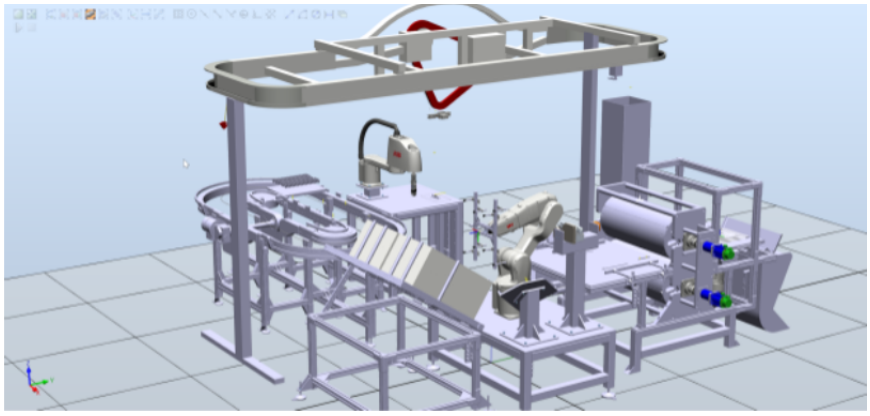
I detta examensarbete kommer möjligheten att ersätta den trådade kommunikationen i fabriken med trådlös kommunikation analyseras genom att låta ett virtuellt PLC styra en virtuell fabriksstation via 5G. Ett koncepttest ska även visas upp i "SmartaFabrikers" fabriksstation.

Fabrikstation

För att kunna automatisera en fabrikstation behövs ett styrsystem som ansvarar för styrningen av alla underordnade enheter, en fabrik består av flera av dessa styrsystem. Dessa styrsystem kan kommunicera såväl med varandra som med underordnade enheter såsom ställdon,

industrirobotar och enheter med tillkopplade givare. Kommunikationen mellan alla dessa enheter utförs ofta via en industriell kommunikationsbuss, en så kallad fältbuss, baserad på Ethernet. Överföringen av data utförs med hjälp av speciellt utformade transportprotokoll. Industriella fältbussprotokoll garanterar att data kommer fram eller skickas vid bestämda tider, med den höga noggrannhet som krävs för att uppfylla de strikta realtidskraven som ställs på kommunikationen i moderna industriella processer.

En fabrik består av flera styrsystem. Varje styrsystem behöver flera kablar för att hantera ingångar och utgångar från olika enheter samt även kommunikation mellan olika styrsystem. Detta innebär att en anläggning består av en oerhörd mängd kablar som behöver installeras, underhållas och eventuellt ersättas, vilket medför stora kostnader för industrin. Detta upplägg gör visserligen att datakommunikationen inom en fabrik blir stabil och pålitlig, men begränsar samtidigt möjligheterna att göra produktionslinjen mer flexibel. Dessutom blir det svårt att införa nya mobila tekniklösningar som t.ex. Automated Guided Vehicles (AGV), vilka förlitar sig på en trådlös koppling för att kunna navigera i en fabrik. En trådlös lösning skulle kunna göra samma fabrik mer dynamisk, där produktionslinjen snabbt kan omformas för att anpassas till dagens industrikrav. Utmaningen med att använda trådlös kommunikation är att realtidskraven i kommunikationssystemet behöver uppfyllas med hög pålitlighet. Det finns idag trådlösa lösningar med t.ex. WIFI som ofta drabbas av störningar och hög latens då flera enheter ansluts, vilket gör dem opassande för hög-precision, realtidsbaserade processer som kräver flera I/O enheter eller flera styrsystem.



Figur 1.1: Simulering av fabrikstation

1.2 Syfte

Vidare undersöka möjligheten att ersätta den trådade kommunikationen d.v.s. fältbussar som för nuvarande används i industriella styrsystem, i synnerhet på tidskritiska system med trådlös kommunikation över 5G.

Även utveckla en prototyp för att kunna testa kommunikationen mellan ett virtuellt PLC och en industrirobot via 5G-nätet. För att kunna bekräfta att realtidskravet uppfylls ska cykeltiden i systemet mätas.

1.3 Avgränsningar

- Kommunikation mellan en dator och en robot ska testas.
- Endast en ingång samt en utgång används för att mäta cykeltid genom 5G.
- Protokoll som analyseras ska vara möjliga att implementeras i den riktiga YuMi-roboten från ABB.
- Hänsyn till real-tidskraven i industrin behöver inte uppfyllas i första hand utan en upprättning av kommunikationen mellan en robot och industridator prioriteras.
- Ett virtuellt PLC kommer att användas men det ska endast köras 'lokalt' då konfigureringen av molntjänsterna innebär stora kostnader i form av tid.

1.4 Precisering av frågeställningen

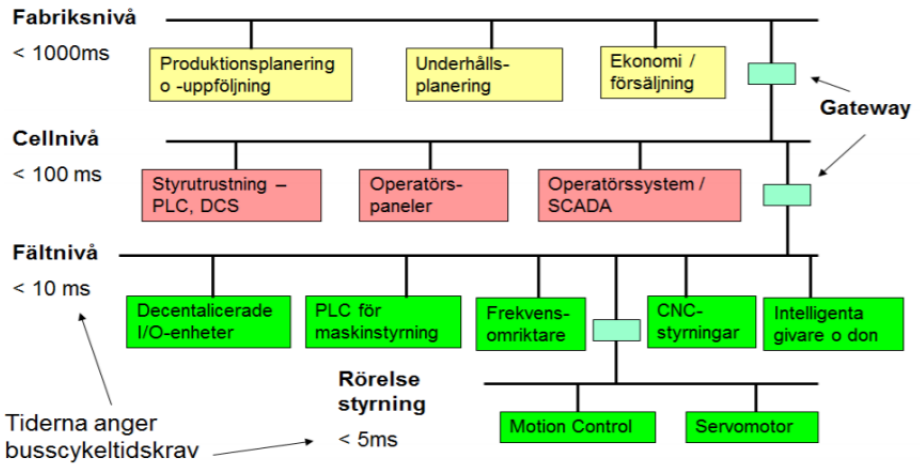
- Vilket kommunikationsprotokoll är lämpligt att använda för mobila nätverk inom industrin?
- Vilken cykeltid kan uppnås för enkla signaler då 5G används som transportmedium?
- Vilken hårdvara och mjukvara behövs för att uppnå tidskraven för kommunikation inom industrin?

2

Teoretisk Bakgrund

2.1 Industriellt kommunikationsnätverk

I en modern uppkopplad fabrikstation sker informationsutbytet på flera nivåer med olika snabbhetskrav för överföringen. Hastighetskraven är höga när det gäller t.ex. överföring av signaltillstånd hos en givare i processen till styrsystemet men för att t.ex. planera underhåll krävs inte lika snabba besked. Det finns alltså olika tidskrav för olika typer av information och därmed olika krav på medium som transporterar informationen [3]. Figur 2.1 visar olika krav på överföringshastighet för de olika nivåer i en fabrik.



Figur 2.1: Olika krav på överföringshastighet för de olika nivåerna i en fabrik.

Vid fältnivån, där överföring av insignaler från processens olika givare till styrsystemen och utsignaler till olika ställdon sker, är kommunikationsbussens transportprotokoll utformat på ett sådant sätt att den uppfyller hårda realtidskrav genom att all data skickas och mottas inom ett visst tidsintervall. Vid fältnivån krävs att detta tidsintervall ligger under 10 millisekunder.

Med detta i tanke var det extremt viktigt för författarna att kunna klassificera olika fältbussar enligt hastighet och funktionalitet. För detta ändamål användes OSI-modellen.

2.2 OSI-modellen

För att kommunikationen mellan datorer ska kunna ske behöver reglerna för kommunikationen fastställas - detta kallas för protokoll. Olika protokoll används av olika tillverkare och operatörer och utan väl definierade standarder kan kompatibilitetsproblem uppstå. Därför skapades OSI-modellen - Open System Interconnection Reference Model. Den modellen är ett ramverk som definierar standarder för kommunikation mellan olika datorer och datanät.

Figur 2.2: OSI-modellen [3].

Modellen består av sju lager men här behandlas främst lager två till fyra då de är mest maskinnära. Lager ett, det fysiska lagret är i detta fall irrelevant:

2.Länkskiktet
Svarar för dataöverföringen mellan två intilliggande noder dvs upprättar en länk med de reglerna som är definierat för just det protokollet så att mottagaren kan förstå meddelandet. Det kan också finnas funktioner för flödeskontroll och felhantering för ökad tillförlitlighet.

3.Nätskiktet
Detta skikt ansvarar för kommunikation över ett nätverk som kan bestå av flera noder. Därför är det viktigt att protokollet i detta skikt ser till att rätt paket kommer fram till rätt mottagare. Det finns även funktioner för att hitta den snabbaste vägen för data som skickas.

4.Transportskiktet
Ser till att data kommer fram till mottagaren i rätt ordning. Den har även adresseringsfunktioner så att data kommer fram till rätt applikation - detta medför att flera applikationer kan användas samtidigt av mottagaren.

Skiktet fem, sex och sju ansvarar för hur dialog mellan datorer synkroniseras, hur data framställs respektive kopplingen av nätverksprocess till applikationer[4]. För att kunna uppfylla hårda reatidskrav ligger de flesta fältbussprotokoller i datalänkskiktet. Genom att använda denna modell kunde olika lösningar systematiskt jämföras och sållas bort i genomförandet.

2.2.1 Fältbussar

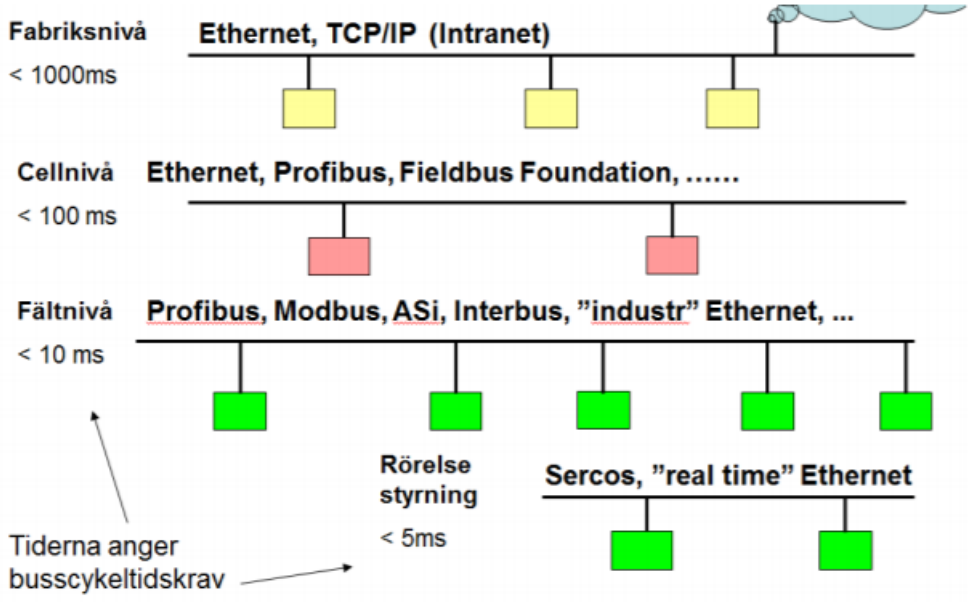
Eftersom kraven är olika har också seriella bussar av olika typer utvecklats. Några exempel av de olika fältbussarna finns i figur 2.3. Några fördelar respektive nackdelar med fältbussar jämfört med traditionell överföring via en ledare per in/-utsignal är minskad kablage och flexibel installation respektive fördröjd dataöverföring och utökad komplexitet. Fördröjningen beror på bland annat hur många enheter är kopplade till en styrenhet, kabellängd samt på fältbussens *access metoder*. Den tiden det tar för data som skickas från styrenheten till en annan enhet att skickas tillbaks till sändaren kallas för *cykeltid*.

2.2.2 Accessmetoder - master/slave

Det finns olika principer för hur en nod (enhet i nätverket) får åtkomst till nätet - vanligast förekommande metod är s.k. Polling, dvs en master-slave uppsättning: en masterenhet och ett antal slavarenheter i nätverket. Mastern lämnar och får data från sina slavar genom att kontakta dem i tur och ordning - genom att ha exakt mängd datautbyte med varje slav blir denna åtkomsttyp tidsdeterministisk.

2.2.3 Cykeltid och realtidskrav

Med cykeltid, i samband med industriella fältbussprotokoller, menas den tiden det tar för masternoden att skicka och få tillbaka från samtliga slavnoder. Cykeltiden är ett fast tidsintervall och sätts då busssystemet konfigureras. Faktorer som kan påverka cykeltiden är antal uppkopplade slavnoder samt fördröjning i nätverket.



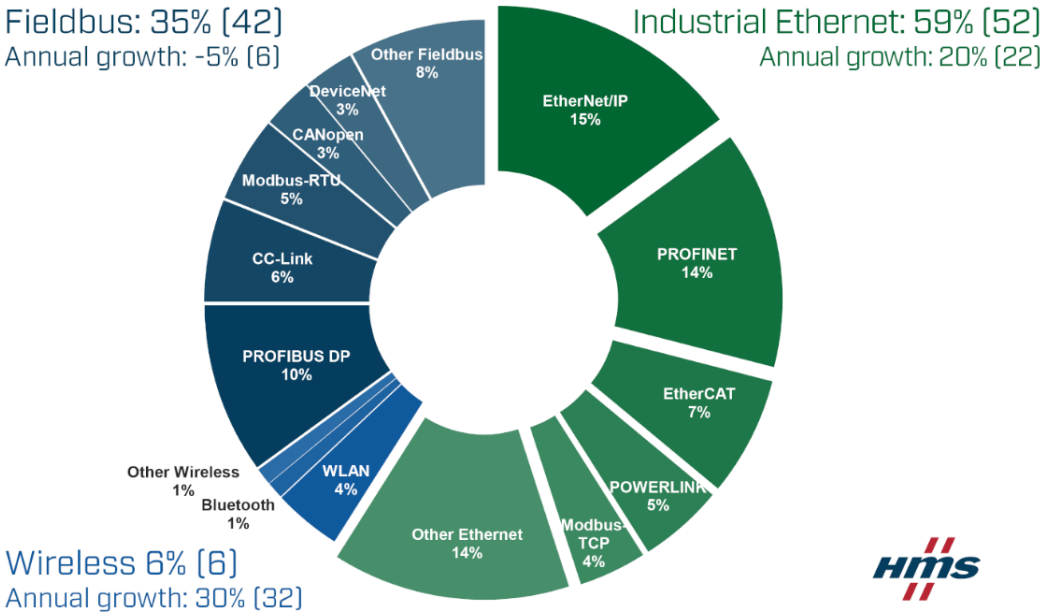
Figur 2.3: Olika fältbussar.

2.3 Protokoll

Ett protokoll som tidigt i förstudiet identifierades som ett flexibelt och väldigt användbart protokoll - i alla nivåer i en fabrik - var Ethernet-protokollet. Detta protokoll har fördelen att det ligger vid skikt två dvs lägsta förutom det fysiska skiktet.

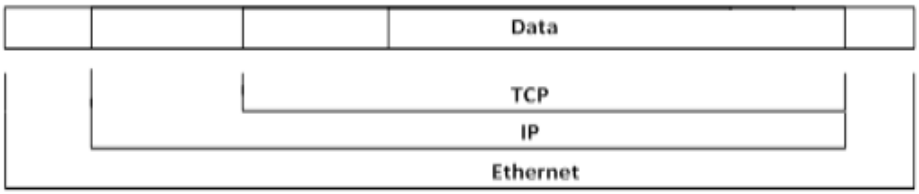
2.3.1 Ethernet

Ethernet-kommunikation blir allt vanligare som kommunikationssätt även inom automationsanläggningar. Det visar sig att industriell Ethernet har gått förbi traditionella fältbussar då antalet nya installerade noder i industrin räknades under 2018, enligt en studie gjort av HMS Networks [5]. Detta kan ses i figur 2.4. Från samma figur kan en ökning av trådlösa överföringsmetoder i industrin identifieras.



Figur 2.4: Industiell Ethernet är nu större än traditionella fältbussar! [5]

Största fördelen med Ethernet är att kunna ansluta till Internet dvs att enkapsulera flera skikt i OSI-modellen. Data som överförs från sändare till mottagare kan tänkas vara ett meddelande av en viss längd som kan transporteras i paket. Via Ethernet kapslas paket in i en Ethernetram tillsammans med annat viktig data som information om IP, TCP/UDP och själva datan som skickas. Figur 2.5 visar en typisk Ethernet-ram.



Figur 2.5: Ethernet TCP/IP-ram.

Ethernetprotokollet har tre grundläggande beståndsdelar: hur data överförs fysiskt och villkor som gäller för varje media; regler för hur varje nod har rätt att kommunicera, s.k. acessmetoder; Ethernetramens innehåll - MAC-adresser till sändare och mottagare, vilken typ av data som skickas, data och slutligen en checksumma för att kunna upptäcka avvikelser i överföringen på grund av störningar mm. Nackdelen med Ethernet är att det inte går att skicka Ethernet-ramar över ett 5G-nätverk utan ett annat protokoll.

2.3.2 PPP och L2TP

Point-to-Point Protocol (PPP) är ett protokoll som ligger i datalänkskiktet, skikt ett i OSI-modellen och som används för att upprätta en länk mellan två ändpunkter.
Layer Two Transfer Protocol (L2TP) är en utökning av PPP-protokollet. L2TPanvänder sig av den generella inkapslingsmetoden hos PPP för att skicka paket från ett främmande protokoll över ett IP-nätverk. Detta innebär att paket från det främmande protokollet kan kapslas in i ett UDP- eller TCP-paket. Detta är väldigt relevant eftersom mobila nätverk inte stödjer Ethernet.

Figur 2.6: Figuren visar ett IP-paket då man använder sig utav en L2TP-tunnel.

Med L2TP-tunneln blir det möjligt att skicka Ethernet-ramar genom 5G. Användaren av en L2TP-tunnel upprättar en länk i datalänksskiktet och använder den länken för att skicka ramar från det främmande protokollet. Eftersom att Ethernet är en IEEE 802.2 standard och 5G är en 3 Generation Partnership Project(3GPP) standard går det inte att skicka Ethernet-ramar över ett 5G-nätverk ännu men möjligheten kommer att inkluderas i senare versioner.

2.3.3 TCP - Transmission Control Protocol

Protokollet är ett sätt att skicka trafik mellan olika datorer. Tillsammans med UDP-protokollet kallas dessa två för transportprotokoll. Detta eftersom de styr över hur data skickas över nätverket [6]. TCP har inbyggda funktioner som säkertställer att data kommer till mottagaren i rätt ordning och om den inte kommer fram skickas den igen automatiskt. Detta gör att ingen data förloras när den skickas via TCP-protokollet över ett nätverk. Nackdelen med detta är att det blir långsammare trafik via nätverket på grund av alla extrafunktioner som finns implementerade i protokollet.

2.3.4 UDP - User Datagram Protocol

Protokollet är betydligt enklare än TCP utan den kontroll av felfri överföring som erbjuds av TCP. Här ligger fokus på att det ska gå snabbt vilket innebär så lite overhead som möjligt och inga omsändningar. Användningsområdet är därför realtidsapplikationer där dataöverföringen är tidsrelaterad. I automationssammanhang är operatörssystem med realtidsövervakning av processer över Ethernet en typisk UDP/IP-applikation [3].

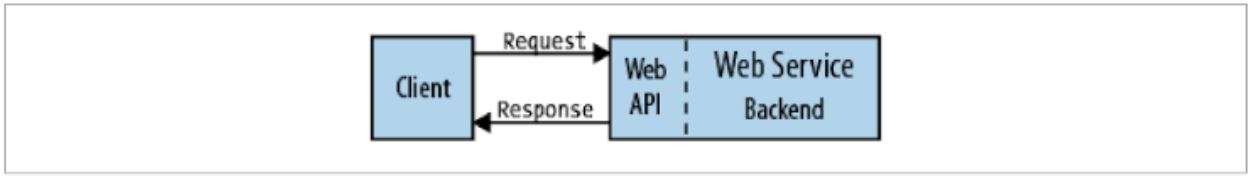
2.3.5 Websockets

En enkel beskrivning av vad Websockets-protokollen innebär ges av Mike Lara [7] i artikeln som han skriver om Websockets för IoT och realtidswebb: Websockets är ett protokoll som möjliggör låga svarstider, bidirektionell samt permanent kommunikation via en enda TCP kanal mellan en klient och en server.
Några exempel på var Websockets används i verkligheten är: realtidsuppdatering av data på en hemsida, realtidssamtal, videokonferenser, IoT övervakning samt många andra tillämpningar där data fås in och behöver ständigt uppdateras. Websockets gör det möjligt att uppdatera data på en webbsida till exempel, utan att behöva ladda om sidan varje gång ny data uppdateras.

En av skillnaderna mot till exempel HTTP- eller AJAX-protokoll som har används tidigare i webbsammanhang är att med Websockets behöver inte klienten initiera kommunikationen genom att göra en begäran till servern först sedan kan klienten begära en gång till för att hämta data från servern. Dessutom när HTTP används så kan inte servern skicka data till klienten utan att klienten först har begärt att data ska skickas till den. Detta skapar onödig fördröjning i realtidsapplikationer och samtidigt behövs mer bandbredd i jämförelse med Websockets. Varje enstaka byte som kan optimeras bort är viktig när en applikation där tusentals givare kopplas upp och data måste uppdateras i realtid.

2.3.6 RWS och REST-API

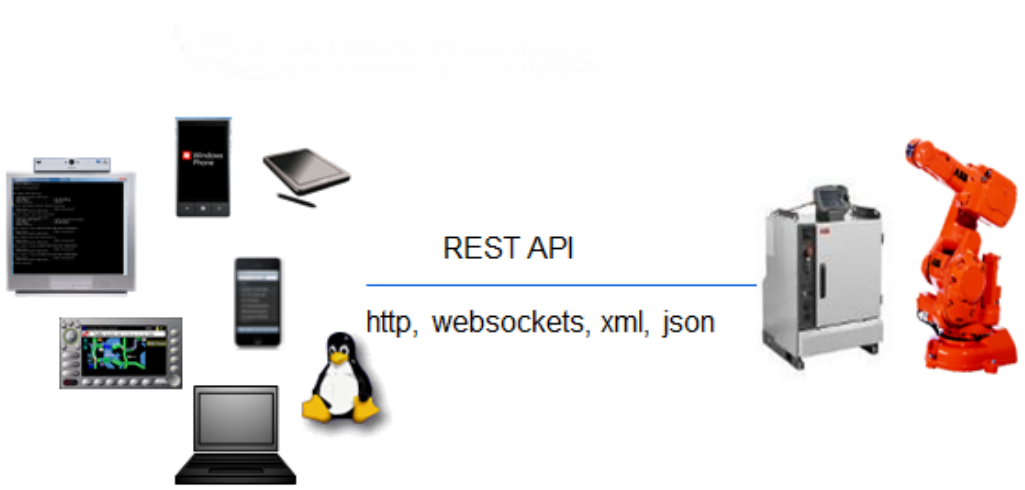
REST är ett IT-arkiktektur begrepp som beskriver hur maskin-till-maskin kommunikation utförs med nuvarande internetinfrastruktur. Webbtjänster är egentligen speciella servrar som stödjer t.ex. webbsidor eller applikationer. Klientprogram använder API (Application Programming Interface) för att kommunicera med webbtjänsterna. En API exponerar data och funktioner för att underlätta kommunikationen mellan olika datorprogram. En webb-API är det som lyssnar och svarar på klienternas begäran. I figur 2.7 kan vi se hur en enkel förfråga/svar kommnunikation sker mellan en klient och en server.



Figur 2.7: Illustration på dataöverföringen med REST-API.

RWS(Robot Web Services) använder HTTP som drivande protokoll vilket i sin tur använder URL och olika kommandon för att skriva och läsa data. En URL, som känd, identifierar en hemsida till exempel: www.google.com, men den kan även identifiera en IO-signal som: '127.0.0.1/rw/iosystem/signals/DO1'. Figur 2.8 visar de olika sätt som finns tillgängliga för att kunna hämta och skriva data till robot-kontrollern.

Klienterna behöver inte fråga efter data från servern utan när någon ändring sker i en variabel så skickas detta automatiskt till lyssnande klienter som en ny händelse. Även Websockets protokollet kan användas för att till exempel prenumerera på olika händelseändringar i roboten.



Figur 2.8: Bild som visar olika protokoll för att kommunicera med robot-kontrollern.

2.3.7 5G New Radio (NR)

Nästa generation av mobila nätverk kallas 5G, eller NR som står för New Radio, och det är en uppgradering av befintlig 4G (LTE) nätverk som i skrivande stund (sommaren 2019) har börjat lanseras i ett tidigt stadie för allmänheten. Det som gör 5G speciellt är bland annat hundra gånger snabbare överföring av datatrafik jämfört med 4G samt mycket mindre fördröjning i nätverket. Ericsson nämner i en rapport från juni 2019 att antalet 5G prenumerationer kommer nå 1.9 miljarder, 35 procent av trafiken kommer använda 5G som transportmedium och upp till 65 procent av den globala populationen kommer ha 5G täckning tills slutet av 2024 [8].

Nedan kommer en lista med viktiga egenskaper som 5G stödjer eller kommer stödja i framtiden [9]:

- Låg energiförbrukning
- IP-baserat paketnät
- Maskin-till-maskin-kommunikation (M2M)
- Network slicing - semi-privata nätverk
- Ultra-reliable-low-latency communication (URLLC)
- TSN (Time Sensitive Networking)
- Skicka Ethernet-ramar

De viktigaste egenskaperna som behövs för vårt ändamål är M2M, URLLC som innehåller TSN kompabiliteter och att kunna skicka Ethernet-ramar. Dessa egenskaperna kommer tidigast släppas under Release 16 och Release 17 enligt 3GPP [9].

Den stora förändringen av 5G inom industrin är att automatisera det som tidigare inte kunde automatiseras på grund av kablarna. Högre flexibilitet och tillförlitlighet kommer möjliggöra för högre grad av automation i framtidens industrier.

Men en viktig aspekt som måste tas hänsyn till är övergången av dagens PLC kommunikation från industriellt Ethernet till 5G. Enligt en studie gjort av Dr. Jens Jakobsen så kan 5G ersätta PLC kommunikationen i stora drag men inte ersätta rörelsestyrningen. Figur 2.9 visar de olika cykeltiderna för de största Ethernetbaserade protokollen. På samma bild visas också att 5G måste kunna skicka Ethernet-ramar eftersom endast två stora protokoll är baserade på TCP/IP eller UDP/IP stacken likt 5G [10].

Network	Market Share	Layer	Cycle time	Real-time extensions
EtherNet/IP	33%	UDP/IP	~ 1 ms	No
PROFINET IO	26%	Ethernet	~ 1 ms	No
PROFINET IRT		Ethernet	250 µs (31,25 µs)	Yes
EtherCAT	15%	Ethernet	12.5 µs	Yes
Modbus/TCP	9%	TCP/IP	~ 10 ms	No
POWERLINK	9%	Ethernet	100 µs	Yes
SERCOS III	< 9%	Ethernet	62 µs	Yes
CC-Link IE Field	< 9%	Ethernet	~ 20 µs	Yes

Figur 2.9: Dagens Ethernetbaserade protokoll [10]

2.4 Mjuk- och hårdvara

2.4.1 Programmable Logic Controller (PLC)

PLC är speciellt utvecklade, robusta datorer som används för att styra olika tekniska processer. De består av en processor och decentraliserade I/O moduler, vilket kan vara såväl digitala som analoga ingångs- och utgångsmoduler. PLC arbetar cykliskt dvs den läser av ingångarnas värden, utför programmet och lägger ut uppdaterade värden på utgångsportarna.

PLC som valdes för detta projekt var ett av ABBs PLC:er, nämligen PLC:t VAC500 - som finns både i fysiskt- och mjukvaruform (SoftPLC). VAC500 kan köras virtuellt dvs en process eller robot kan styras genom att exekvera PLC-programmet i en vanlig Windows miljö.

2.4.2 Automation Builder

Denna mjukvara från ABB används som gränsnitt för att kunna konfigurera, programmera och felsöka VAC500. Denna mjukvara använder Controller Development Systems (CodeSys) - ett utvecklingsmiljö inom den internationella industristandarden IEC 61131-3 - som programmeringsgränssnitt.

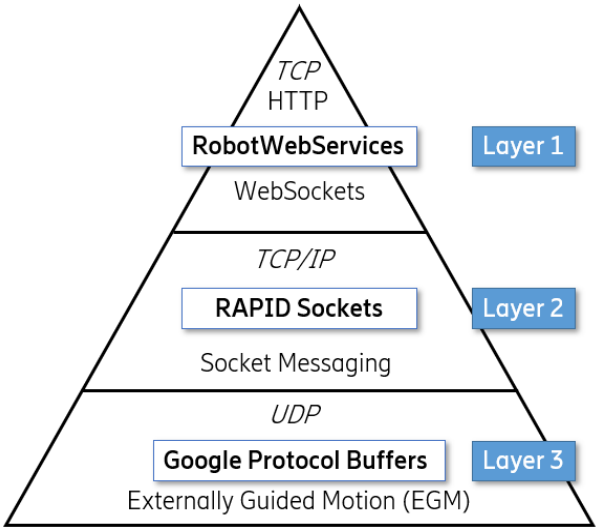
2.4.3 Codesys' PLCHandler

PLCHandler är en API och bibliotek från Codesys utvecklare, som används för att externt kunna komma åt variabler för en kontroller eller symboler från PLC:t. Dessutom tar PLCHandler hand om interprocesskommunikation mellan VAC500 och andra delar av systemet.

2.4.4 RobotStudio

För att kunna simulera en robot används ABB:s egna programvara RobotStudio. Den har en inbyggd styrenhet som beter sig som en verklig robotstyrenhet. Detta gör att om själva programmet fungerar i simuleringen så kommer den i slutskedet även att fungera i verkligheten.

Det finns olika sätt att kommunicera med ABB:s robotar och de tre vanligaste metoderna visas i figur 2.10. De olika lagren visar även bakomliggande protokoll som används för att etablera kommunikationen.



Figur 2.10: Bild som visar olika metoder för att kommunicera med ABB:S robotstyrenhet.

Genom att använda Robot Web Services(RWS)[11] API (RobotWare version 6.08.01) som finns tillgänglig på ABB:s hemsida kan följande processer utföras:

- Läsning/Skrivning till I/O-signaler
- Läsning/Skrivning av data från RAPID-moduler
- Starta/Stoppa/Starta om RAPID-moduler
- Prenumerationer, till exempel när någon variabel ändras (WebSocket)
- Starta eller stoppa motorerna

Dessa och några andra funktioner kan åstadkommas via HTTP och WebSockets som använder TCP protokollet för kommunikation mellan server och klient. Användaren behöver endast programmera klienten som ansluter sig till den inbyggda servern i styrenheten. Att kommunicera med roboten blir då betydligt enklare men åt andra sidan kan det förekomma högre svarstider genom att använda den här uppsättningen. Mer information om RWS finns beskrivet i detalj på ABBs hemsida [11].

Nästa lager i figur 2.10 visar RAPID socket kommunikation. Här skickas meddelanden mellan en klient och server. Dessa meddelanden kan sedan omvandlas till RAPID-instruktioner för att styra roboten. En hantering av inkommande meddelanden samt programmering av både klient och server till styrenheten ska utvecklas vilket medför ytterligare tid. Mer kunskaper om ABB:s egna programmeringsspråk RAPID behövs för att sätta upp kommunikationen mellan en server och en klient. Svarstiderna här är lägre jämfört med RWS eftersom autentiseringen görs endast en gång då en ny socket upprättas, sedan kan meddelanden skickas mellan klient och server utan extra tillägg.

Vidare i lager 3 så kommer 'Google Protocol Buffers' som används mest till direkt rörelsekontroll av roboten. Här används UDP som bakomliggande protokoll eftersom kommunikationen ska ske så snabbt som möjligt. För att komma åt funktionen behöver EGM (Externally Guided Motion) valet finnas tillgängligt i styrenheten. Detta kräver en extra licens.

2.4.5 Machina

Machina är ett öppen källkod program skrivet i .NET, dock kan även andra programmeringsspråk användas för att kommunicera med en robot via ramverket. Machina är ett ramverk som underlättar människa till robot uppkopplingen genom att bidra med ett visuellt gränssnitt för användaren och på så sätt mycket lättare koppla sig till robotkontrollern och utföra rörelser samt läsa/skriva till I/Os. Projektet är dock i start-up fasen och en hel del funktioner är inte testade ännu.

Om ramverket fungerar som det ska kommer den implementeras för att kontrollera en ABB-industrirobot via 5G i realtid med hjälp av en server-klient TCP-kommunikation via 5G. Enligt utvecklaren har de testat att kontrollera en stor industrirobot från ABB med hjälp av en vanlig handkontroll via ramverket. Dock har de kopplat roboten med vanliga Ethernetkablar, en video på detta finns tilllagt på YouTube som visar detta [12].

3

Metod

Detta projekt kan indelas i två stadier: första stadiet utgör en förstudie där förra årets hårdvarulösning ska försöka återskapas och cykel-tidsmätningar utföras och andra stadiet, som kan indelas vidare i tre ytterligare steg där alternativa lösningar kommer att undersökas och utvecklas.

3.1 Första stadiet

Första stadiet utgör kärnan i detta projekt, då en fungerande hårdvarulösning med en medellåg cykeltid anses vara av ytterst vikt att fastställa i ett tidigt skede av projektet. Därmed möjliggörs även en eventuell vidareutveckling av den befintliga hårdvaran. Beroende på kompatibiliteten mellan 4G och 5G samt huruvida ett 5G modem finns tillgänglig, kan lösningen snabbt testköras med 5G då uppsättningen väl är i drift.

Första stadiets delkomponenter:

- Raspberry Pi med Linux, används som nätverksbrygga
- 4G eller 5G Modem
- Laptop kopplad till ett virtuellt styrsystem
- Kommunikation med RobotStudio.

En L2TP tunnel kommer att konfigureras med avsikt att skicka Ethernet-ramar via mobila nätverket. En fungerande mjukvarulösning med PLCHandler (tidigare mjukvarulösning som skickar signaler från styrsystemet) och det mjukvarubaserade fältbussprotokolet OpenPowerlink kommer att implementeras.

3.2 Andra stadiet

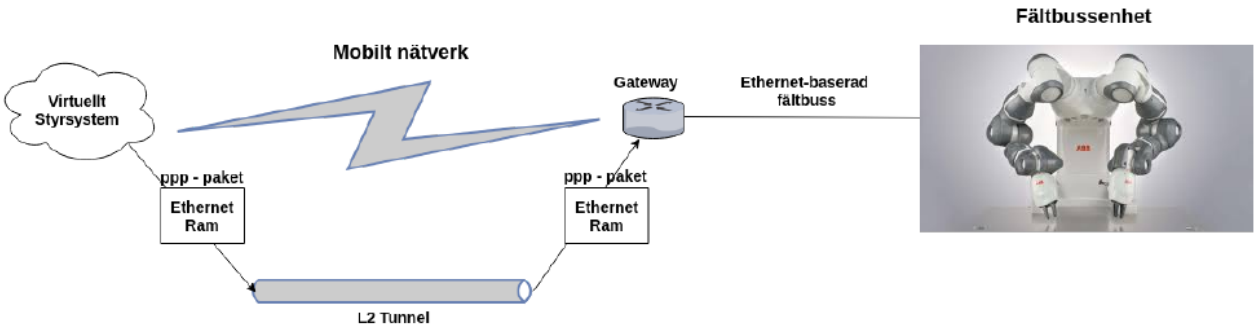
Implementera en annan kommunikationprotokoll mellan ABBs industrirobot och hårdvarulösningen. Detta görs via Robot Web Services (RWS) som är ett platformoberoende protokoll baserat på Representational State Transfer (REST) vilket är en Application Programming Interface (API). Detta kan underlätta och möjliggöra lättare och snabbare kommunikation till och från industriroboten. RobotStudio kommer användas för att simulera ABB roboten som kommer användas vid senare testning i verkligheten. Programvaran samt licens fås från ABB.

Överföringen via 5G/NR till och från hårdvarulösningen kommer implementeras antingen via en tunnel som tidigare examensarbete eller via TCP/IP. Dock så kommer även andra överföringsmetoder att analyseras så att den bästa metoden identifieras och implementeras.

Signaler från styrsystemet kommer testas och programmeras så att de kan tas emot från hårdvarulösningen med tillhörande protokoll. Här kommer antingen OpenPowerlink användas eller annan protokoll för överföring beroende på analysen som utförs och möjligheterna som finns. Alla kommunikationstester kommer utföras individuellt för att sedan se om det finns någon flaskhals i systemet. Genom att dela på systemet i flera olika delar kan flaskhalsen lättare identifieras.

3.3 Tredje stadiet

Här kommer hela systemet testas i en simuleringsmiljö och om hela implementationen fungerar felfritt kommer den även att testas i verkligheten med alla ingående delar samt en industrirobot från ABB.



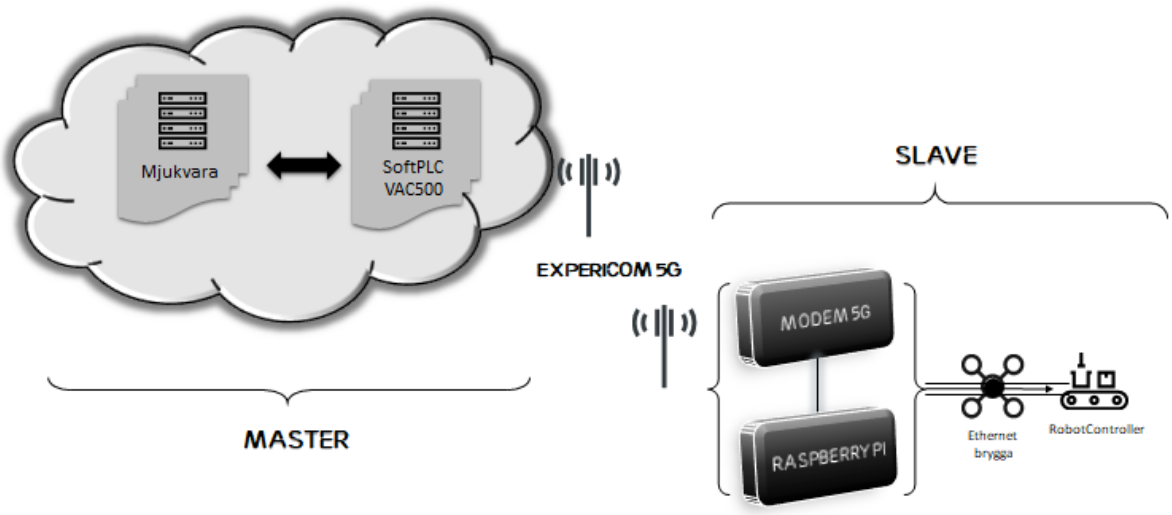
Figur 3.1: Överföring av Ethernet-ram via L2TP-tunnel. [2]

4

Genomförande

4.1 Förstudie - Systemöversikt

Det tänkta systemet består av två delsystem: molnet representerar en server i Ericssons serverkluster som kör både en speciellt utvecklad mjukvara och det virtuella PLC:t – detta delsystem utgör fältbussens 'Master node' och får benämning *master*; delsystem 2 består av ett modem och en Raspberry Pi som är konfigurerat på så sätt att den agerar som en nätverksbrygga mellan 5G-nätverket och fältbussen - detta delsystem utgör fältbussens 'Slave node' och får benämning *slave*. Detta kan ses i figur 4.1.



Figur 4.1: Systemöversikt för förra årets examensarbete som blev utgångpunkten för detta arbete.

4.1.1 Master

Såväl det virtuella PLC:t som den utvecklade mjukvaran körs på en 'virtual machine' – i detta fall både Windows och Linux. Det virtuella PLC som används i koncepttestet är ett mjukvarubaserat PLC från ABB. Mjukvaran implementerar både fältbussprotokollet OpenPowerlink samt ett API mot PLC:t. Mjukvaran har också till uppgift att mappa signalernas symbol-namn i PLC:t till motsvarande symbolnamn på fältbussen (PLC-handler). Styrsignalerna kapslas sedan in i en ethernetram och skickas in till L2TP-tunneln för överföring över 5G-nätet.

4.1.2 Slave

Ett modem installerat på en RaspberryPi tar emot Ethernetramen från 5G-nätet som sedan skickas vidare till rätt port för att kunna läsas av fältbussenheten - en enhet som kan kommunicera via det implementerade fältbussprotokollet, tex en modul med ingångs- och utgång-anslutningar eller en industrirobot. Därefter kan enheten (slave-nod) skicka svar till master-nod i omvänt ordning.

4.2 Procedur

En RaspberryPi införskaffats med Linux och ett 4G-modem installerades. Vidare utvecklades mjukvara i C++ som gjorde det möjligt att koppla sig mot det virtuella PLC:t m.h.a. PLCHandler-biblioteket. Det färdiga biblioteket användes som mall i utveckling av mjukvaran och enkel testfunktionalitet åstadkoms så att variabler kunde skrivas och läsas.

Även om förstudien var givande för senare stadier i detta projekt valdes att avbryta då det insågs att det skulle bli svårt att implementera denna lösning på grund av att OpenPowerlink-protokollet inte stödjs av ABB-robotar och därför inte går att implementera i Smarta Fabrikers robotarna.

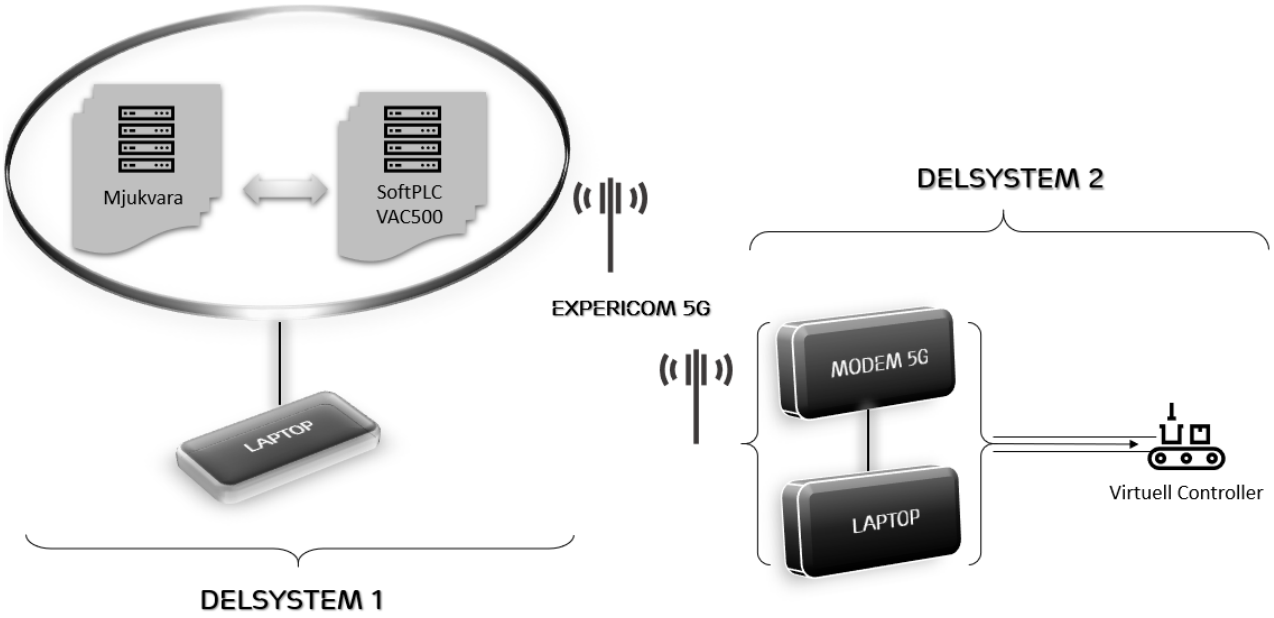
4.3 Syntes - Systemöversikt

Även om OpenPowerlink-protokollet valdes bort, var det ändå rimligt att behålla vissa delar av systemet för vidare utveckling. Det virtuella PLC:t VAC500 ansågs vara ändamålsenlig, PLCHandlers C++ bibliotek likaså. Nästa steg blev då att välja ett kompatibelt fältbussprotokoll som skulle kunna integreras i den hittills utvecklade mjukvaran. För en översikt av det tänkta slutgiltiga systemet se figur 4.2.

4.4 Mjukvarulösning

4.4.1 Val av fältbussprotokoll

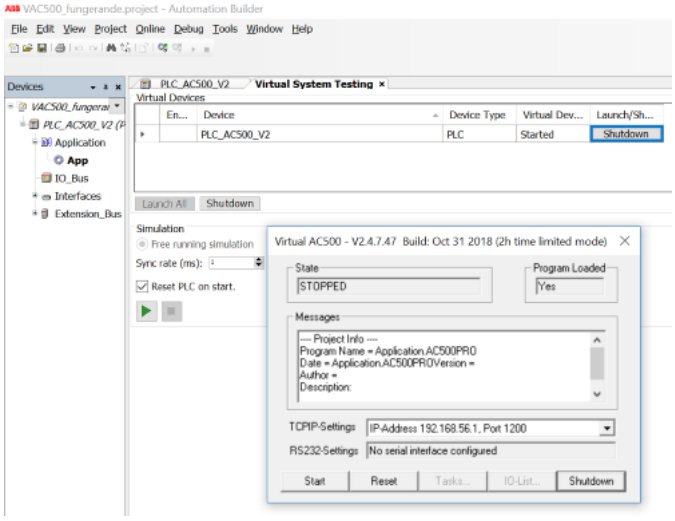
Det finns flera fältbussprotokoll riktat mot realtidsprocesser, de flesta är proprietära protokoll och kan innebära höga licensavgifter och expert-hårdvara för att upprätthålla snabba synkroniserade cykler. Valet av protokoll var begränsat till de som ABB-robotar stödjer vilket utesluter även 'öppna' protokoll såsom OPC-UA och OpenPowerlink. I Smarta Fabriker används huvudsakligen PROFINET som övervägdes tyngst då det var viktigt för författarna att kunna ta fram en fungerande implementation. Att välja fältbussprotokoll blev en mer komplex och tidskrävande process än tidigare tänkt eftersom det råder låg kompatibilitet och lite standardisering i industrin. Därför övervägdes istället att använda befintlig internetsinfrastruktur som ett alternativ till befintliga fältbussprotokoller, i detta fall *Websockets*. Eftersom ABB är en intresserad aktör, bestämdes det, i diskussion med vår handledare, att ABB:s egna Robot Web Services skulle kunna användas för att ansluta oss till RobotStudios server samt att skicka och ta emot data till/från RobotStudio. Även om Socketsprotokollet ligger på skikt fem i OSI-modellen ansågs detta protokoll - i samband med de förväntade låga svarstider som 5G kommer att innebära - värt att undersöka vidare.



Figur 4.2: Översikt av hela systemet

4.4.2 Kommunikation med PLC:t

För att kunna kommunicera med det virtuella PLC:t och därmed komma åt PLC:ts variabler användes API:t/biblioteket PLCHandler. Automation Builder utgör gränsnittet till VAC500, CodeSys används för att skapa styrprogram och körs in i Automation Builder. Två globala bool I/O variabler skapades i Codesys: *DO1 och DI1* och exponerades genom att ändra inställningar in i CodeSys meny ”Project” -> ”Options” -> ”Symbol configuration”. Ett enkelt testprogram skapades där en utgångssignal sätts till TRUE och en tidsräknare sätts igång. Programmet inväntar sedan en ingångssignal för att då stanna räknaren om en sådan registreras. Detta program kan sedan laddas in till VAC500 genom att köra Virtual System Testing från menyn. Programmet behöver köras i Free running simulation, detta visas i figur 4.3.

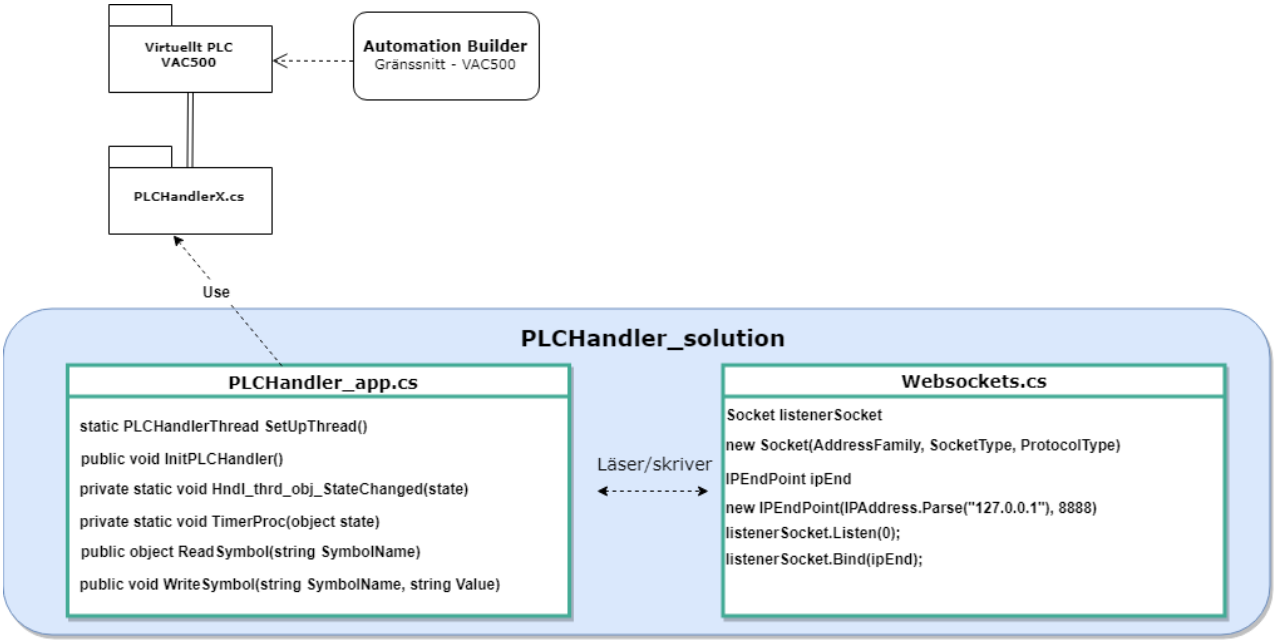


Figur 4.3: VAC500s gränsnitt Automation builder inställt på Virtual System Testing

Som tidigare nämnt var det smidigt att använda PLC:t VAC500 för styrning av roboten. Istället för det ursprungliga C++ program, användes istället en annan version av PLCHandler som skulle vara mer kompatibel med de olika Websockets implementationerna. Denna version bestod av ett C# bibliotek med inbyggd *Interprocess-kommunikation* och utökad funktionalitet t.ex. *events*, som är väldigt användbara för att uppmärksamma användaren på en förändring i någon av variablerna. En synkron 'polling' av variablerna sker var millisekund med hjälp av en *Timer*-metod där de booleska variablerna i PLC:t läses och sparas så att nya värden jämförs cykliskt med tidigare värden och att förändringar på deras värden upptäcks. Då en förändring registreras skapas en event och den variabel vars värde ändrats skickas vidare till nästa del av programmet.

4.4.3 Websocketsservern

Ett enkelt Websocketsserver med låg funktionalitet skapades både i C++ och sedan i C# för att kunna skicka och ta emot data till en klient.



Figur 4.4: UML-diagram för kommunikation mellan PLC:t och Websockets

4.4.4 RobotStudio

RAPID koden som användes för att påverka och läsa I/O är följande (Koden är lite förenklad):

```
MOVEL Target_10; !Hem-läget
WaitDI Start_seq,1; !Vänta tills Start_seq går från 0 till 1
SetDO RobotReady,1; !Sätt RobotReady till 1
MoveL Target_20; !Gå till nästa target med robotarmen
Reset RobotReady; !Sätt RobotReady till 0
MoveL Target_10; !Gå tillbaka till Hem-läget
```

'!' är tecknet för kommentarer i RAPID-språket.

Nästa steg för att kunna koppla upp sig till en virtuell robotkontroller ska följande egenskaper kontrolleras i förhand:

1. Kontrollera att brandväggen på datorn tillåter kommunikation från andra enheter. Stäng av brandvägen om du har den möjligheten.
2. En virtuell kontroller tillåter endast kommunikation från den lokala datorn som standard. För att ändra detta måste du lägga in IP-adressen till datorn som du försöker fjärrstyra roboten ifrån. Detta görs i en konfigurationsfil som finns under:

   "C:\Users\{user}\AppData\Roaming\ABB Industrial IT\Robotics IT\RobVC\vcconf.xml"

   Den kan se ut som nedan:

   <?xml version="1.0"encoding="ISO-8859-1"?>

   <hosts>

   <host ip="10.11.125.6"/>

   <host ip="10.11.125.7"/>

   </hosts>

   Om filen inte finns så skapa en fil 'vcconf.xml' och lägg till informationen ovan.
3. Skapa en ny station med robotkontroller i RobotStudio och välj en robotmodell. I vårt fall användes YuMi roboten(IRB 14000) och RobotWare versionen 6.08.01.00 eftersom den stationen kunde sedan testas på en riktig YuMi robot som finns tillgänglig hos Ericsson.
4. Kontrollera att den nya stationen har alternativet '616-1 PC Interface' installerad under *Virtual Controller>Change Options>Communication*.
5. Skapa globala I/Os under rubriken. *Controller>Configuration>I/O System>Signal* och namnge en digital ingång (Start_seq kallas den i vårt fall, se bilaga A för fullständig klientkod) samt en digital utgång (RobotReady). Se till att 'Access Level' på variablerna är 'All'.
6. Starta om kontrollern och nu kan de nya variablerna som skapades i föregående steg övervakas och deras värden kan läsas samt ändras från RWS-klienten.

4.4.5 Machina.NET

Machina-biblioteket användes under projektet för att upprätta en kommunikation med robotkontrollern. För att komma igång med Machina.NET följdes instruktionerna från den öppna GitHub-koden [13]. Ett exempel från biblioteket provades och RobotStudio konfigurerades så att en YuMi robot kunde kontrolleras via kommandon från tangentbordet. Detta användes under en presentation/workshop hos Ericsson för att visa möjligheterna av 5G. YuMi-roboten som fanns på datorn simulerades och dess armar kontrollerades från en annan dator via tangentbordet. Detta projekt använde TCP/IP sockets för kommunikation men krävde att RAPID-funktioner skulle läggas in i robotkontrollern för att fungera. Eftersom tiden var begränsad valde vi att inte fortsätta med biblioteket eftersom vi endast skulle skapa en funktion som beräknar tiden en cykeltid tar. Istället ansågs det vara smidigare att skapa en Python-klient för att kunna ansluta till RobotStudio.

4.4.6 Konfiguration av Python-script

Filen 'RWPanelSubscriber.py' som finns i bilaga A ska startas men innan den kan köras behöver dessa steg följas (Python 3.7):

1. Installera senaste *requests* biblioteket genom att använda antingen **'easy_install requests'** eller **'pip install requests'**
2. Installera *websocket ws4py* biblioteket genom att använda antingen **'easy_install ws4py'** eller **'pip install ws4py'**
3. Ändra den globala variabeln 'ip_adress' till IP-adressen för robotkontrollern.

4.4.7 Kommunikation mellan Python och RobotStudio

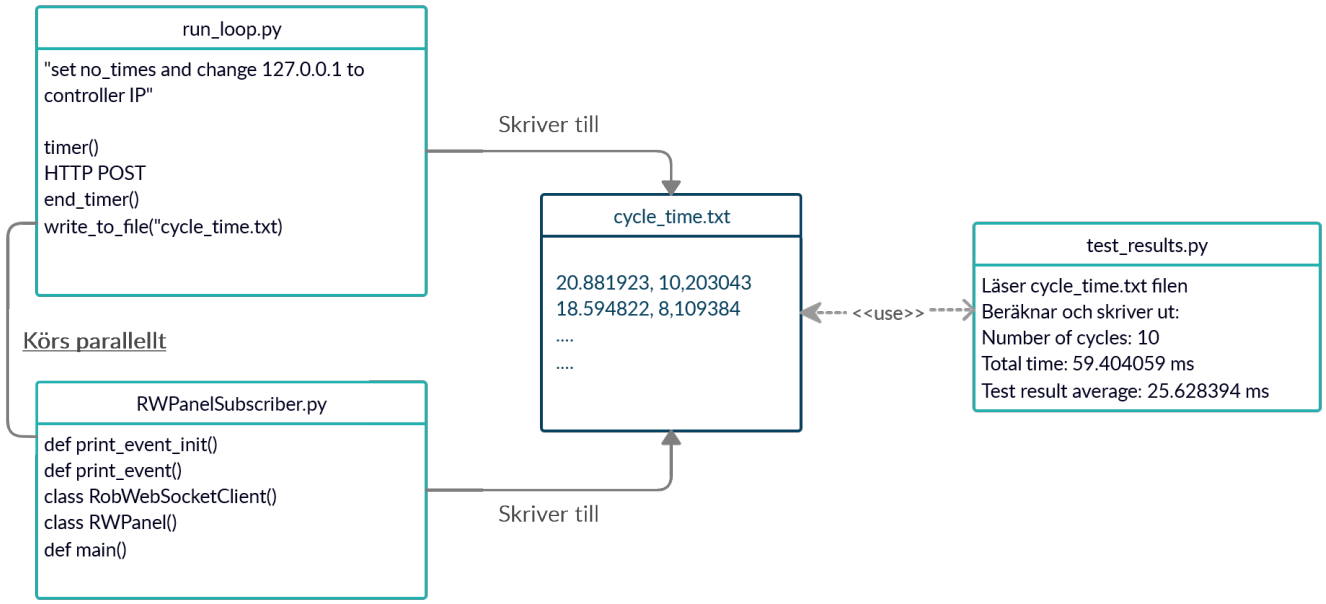
Figur 4.5 visar hur Python-scripen är kopplade till varandra samt de viktigaste funktionerna som användes. Nedan kommer även en beskrivning av hur en koppling till RobotStudio utförs med hjälp av Python.

En typisk utskrift när RWPanelSubscriber.py startas och en kommunikation med roboten upprättas ser ut som nedan:

Initial Events:
    RobotReady state: 0
    Start_seq state: 0
Web Socket connection established.

Sedan om dessa variabler ändras kommer den nya värdet skrivas ut löpande i realtid.

Filen 'run_loop.py' som finns i bilaga B ska köras parallellt efter en lyckad uppkoppling. Genom att ändra variabeln *no_runs* kan antalet cykeltester bestämmas. Därefter utförs automatiskt en sekvens där 'Start_seq' variablen sätts till '1' och roboten gör en enkel uppgift som ska programmeras av användaren. Efteråt ska roboten skicka tillbaka en kvittens genom att sätta variabeln 'RobotReady' till '1'. En cykeltid har då utförts och tiden det tog för en HTTP POST request sparas i filen 'cycle_time.txt'.



Figur 4.5: Diagram som visar Python kod som användes för kommunikation mellan RWS klient och RWS server.

Observera att tiden det tar för roboten att svara skrivs också till samma fil men i en annan kolumn. Dessa adderas sedan för att få den fullständiga cykeltiden. För att se ett exempel på hur filen 'cycle_time.txt' ser ut och de olika kolumnerna separerade av ett kommatecken, se bilaga D.

Filen 'test_results.py' bearbetar informationen som finns i file 'cycle_time.txt'. Den adderar de två olika kolumnerna och sedan summerar varje rad. Därefter divideras den totala tiden med antalet cykeltester, alltså antalet rader, så att ett medelvärde fås. Detta resultat skrivs ut i terminalen.

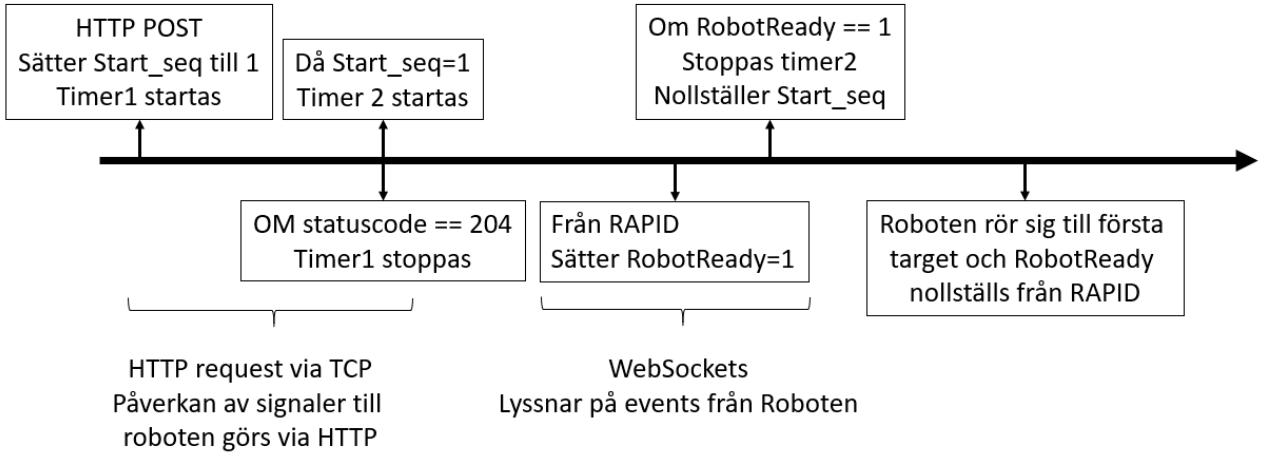
4.4.8 Genomförande av test mellan klient och server

För att kunna testa 5G-nätet så tidigt som möjligt så skapades en uppsättning likt den i figur 4.7. Observera att den cykeltid som var viktigast för oss att mäta i första hand, är mellan RWS klienten från ena datorn till RWS servern som finns på en annan dator. Den andra datorn kör RoboStudio och simulerar en riktig robotkontroller från ABB. Båda dessa datorerna är kopplade till 5G-nätverket via Sierra Wireless modemen.

Som figur 4.7 visar utfördes cykeltidtesterna mellan RWS KLIENTEN och RWS SERVERN som finns inbyggd i robotkontrollern. Efter att en virtuell robotkontroller har startats i RobotStudio så kan cykeltesterna utföras.

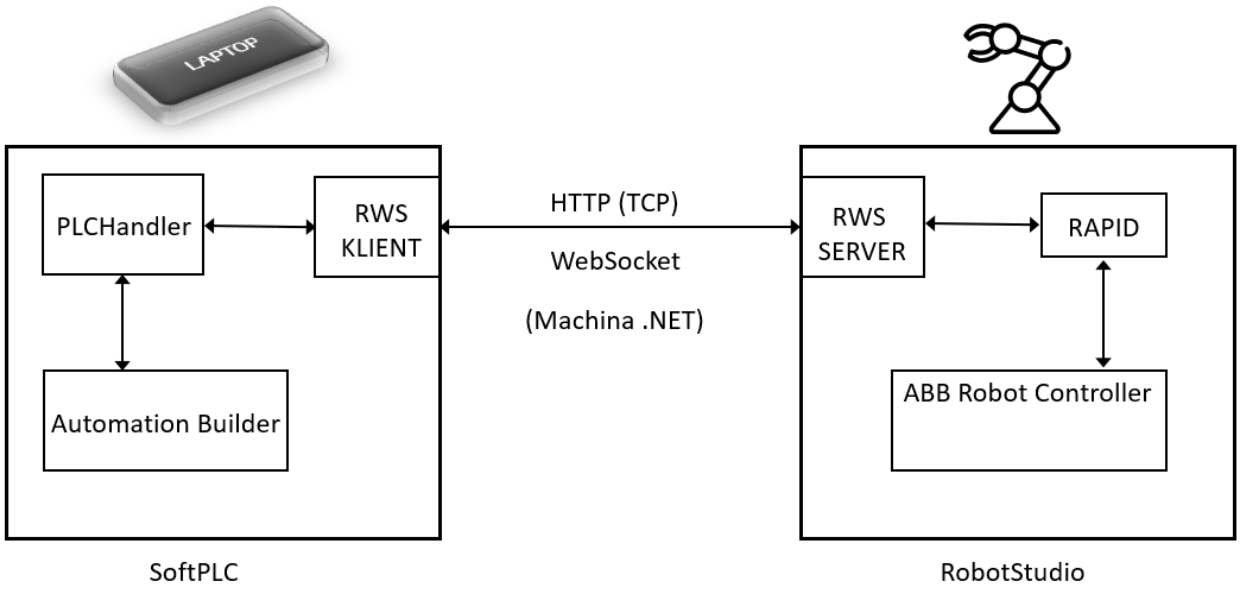
Sekvensen som är beskrivet i figur 4.6 visar hur I/O signalerna påverkas från Python-scripten samt även hur de nollställs för att kunna skapa en oändlig loop.

För att kunna få ett tillförlitlig resultat utfördes hundra cykeltester och resultatet av dessa kan ses i en tabell under *Resultat*. Genomförande av cykeltester gjordes även via WiFi, LAN och att köra simuleringarna lokalt på en och samma dator utöver det viktiga testet via 5G-nätet.



Figur 4.6: Sekvens för att utföra automatiska tester

4.4.9 Test av hela systemet



Figur 4.7: Testuppsättning

5

Resultat

Målet var att utveckla en prototyp för att kunna testa kommunikationen mellan ett virtuellt PLC och en industrirobot via 5G-nätet samt att mäta dess cykeltid. Med systemet som utvecklades är det möjligt att styra de flesta ABB-robotar om motsvarande *controller* (RobotWare 6.08.01) används. Detta på grund av att den virtuella controller beter sig på samma sätt som den verkliga. Koncepttestet testades virtuellt i RobotStudio och cykeltidstester utfördes. Koncepttestet kopplades även till en verklig YuMi-robot men inga cykeltidstester utfördes då 5G-täckning i lokalen ansågs vara otillräcklig.

Tabell 5.1 visar resultatet från de slutgiltiga testerna som utfördes under projektets gång. Det är viktigt att poängtera att 5G inte är färdigimplementerad än och 5G-täckningen var därför inte konsekvent under projektets gång. Den minsta fördröjningen som mättes via 5G var i genomsnitt 70 ms. Fördröjningen mättes genom att utföra ett antal ping komandon från ena datorn till den andra. Observera också att eftersom datorerna inte hade samma gateway så kunde inte kommunikationen ske direkt mellan datorerna.

Då nedladdnings- och uppladdningshastigheten testades via tjänster som 'fast.com' eller 'bredbandskollen.se' upptäcktes att - då vi använde de givna 5G modemen - medelvärdet av testerna låg runt 100 Mbit/s ned och 5 Mbit/s upp. Observera den låga uppladdningshastigheten. Viktigt att ha i åtanke är även att modemen, vilka var av typen 'Sierra Wireless EM6575', egentligen är '4G LTE Advanced' modem som har låsts till framtida 5G-frekvenser (Band 42 / 3.5 Ghz).

5.1 Tabeller

	Antal cykeltester	Medelvärde (ms)	Fördröjning/ping(ms)
Lokalt på samma dator	100	15.0208	0
LAN	100	21.0237	5
Trådlös nätverk (WiFi)	100	38.3158	15
Via 5G (NR)	100	238.25	70

Tabell 5.1: Resultat från alla utförda tester då kommunikation mellan robotcontrollern och en klient testades

	Antal cykeltester	Medelvärde (ms)	Fördröjning/ping(ms)
Via 5G (NR)	100	563	70

Tabell 5.2: Resultat från kommunikation mellan PLC och RobotStudio via 5G

5.2 Mjukvara

Mjukvaran som utvecklades är skriven i C# och Python-script. Detta program upprättar kommunikation mellan ett virtuellt PLC och en Websockets-server via mobila nätverk. Kommunikation in till RobotStudios server hanteras med hjälp av Robot Web Services. Websockets används som alternativ till de befintliga fältbussprotokollen.

5.2.1 Kommunikation med PLC:t

Först behöver en *thread* sättas upp genom att skapa en *instance* av *PLCHandlerthread* och att knyta den *StateChanged* event till denna instance. Om PLC:ts *state* ändras så körs en metod som skriver ut PLC:ts nya *state*.

```
static PLCHandlerThread SetUpThread()
{
    var hndl_thrd_obj = new PLCHandlerThread();
    hndl_thrd_obj.StateChanged += Hndl_thrd_obj_StateChanged;
        return hndl_thrd_obj;
}

private static void Hndl_thrd_obj_StateChanged(PLCHandlerThread.PLCHANDLER_STATE state)
{
    Console.WriteLine("State Changed!" + state.ToString());
}
```

Listing 5.1: PLC:ts nuvarande läge

Sedan kan PLCHandler ansluta till VAC500 genom att använda metoden *InitPLCHandler* med rätt konfigurationsfil.

```
public void InitPLCHandler()
{
    PLCIniFile iniFile = new PLCIniFile("PLC_Id0", "127.0.0.1", true);
    iniFile.PLCIpAddress = new KeyValuePair<string, int>("localhost", 1200);
    InitPLCHandler(iniFile);
}
```

Listing 5.2: Anslutning till PLC:t

Därefter körs en *Timer* som läser variablernas värden cykliskt - varje millisekund - för att hålla koll på förändringar i variablernas värden. När en förändring registreras startas tidtagningen och om värdet på ut-variabeln är '1' körs Python-scriptet. Då in-variabel blir '1' skrivs det värdet till PLC:t.

```
public static void TimerProc(object state)
{
    Stopwatch stopWatch = new Stopwatch();
    object o = hndl_thrd_obj.ReadSymbol(".DO1"); string DO1 = o.ToString();
    if (DO1 == "TRUE") DO1 = "1";
    else DO1 = "0";

    if (DO1 != oldValue){
        oldValue = DO1; stopWatch.Start();
        if (DO1 == "1"){
            string DI1 = ExecPython(DO1);
            if (DI1 != "0"){
                DI1 = "TRUE";
                hndl_thrd_obj.WriteSymbol(".DI1", DI1);
            }}}
    stopWatch.Stop();
```

Listing 5.3: Räknare som läser av variablernas värden

Metoden *ReadSymbol* används för att kunna läsa variablerna. På liknande sätt fast med metoden *WriteSymbols* istället, kan variablerna tilldelas värden.

5.2.2 Websockets

Funktionalitet där det gäller Websockets åstadkoms genom att exekvera python-scriptet ifrån C#-programmet. Då en variabel uppdateras, skickas en HTTP-request och en Websockets-anslutning kan då upprättas där scriptet lyssnar på förändringar i variablernas värden.

```
set_Start_seq = requests.post('http://{0}/rw/iosystem/signals/Start_seq?action=set'.format(
    ip_adress),
                                  data=data,
                                  auth=HTTPDigestAuth('Default User', 'robotics'))
    if set_Start_seq.status_code == 204:

    def subscribe(self):
        payload = {'resources': ['1', '2'],
                   '1': '/rw/iosystem/signals/Start_seq;state',
                   '1-p': '2',
                   '2': '/rw/iosystem/signals/RobotReady;state',
                   '2-p': '2'
                  }
        resp = self.session.post(self.subscription_url, auth=self.digest_auth, data=payload)
        print ("Initial Events : ")

        print_event_init(resp.text)
        if resp.status_code == 201:
            self.location = resp.headers['Location']
            self.cookie = '-http-session-={0}; ABBCX={1}'.format(resp.cookies['-http-session-'],
                resp.cookies['ABBCX'])
            return True
        else:
            print ('Error subscribing ' + str(resp.status_code))
            return False
    def start_recv_events(self):
        self.header = [('Cookie', self.cookie)]
        self.ws = RobWebSocketClient(self.location,
                                     protocols=['robapi2_subscription'],
                                     headers=self.header)
        self.ws.connect()
        self.ws.run_forever()

    def close(self):
        self.ws.close()
```

Listing 5.4: Kod för att prenumerera på RWS

För att kunna anropa Python-scriptet används funktionen *ExecPython()*

```
public static string ExecPython(string fromPLC)
        {
            string DO1 = fromPLC;

            //create process info
            var psi = new ProcessStartInfo();
            psi.FileName = @"C:\Users\edslndr\AppData\Local\Programs\Python\Python37-32\python.exe"
                ;

            //provide script and arguments
            var script = @"C:\Users\edslndr\PycharmProjects\RS_sub_write\src\run_loop_2.py";
            //var script = @"C:\Users\edslndr\PycharmProjects\RS_sub_write\src\RWPanelSubscriber.py
                ";

            psi.Arguments = $"\"{script}\" \"{DO1}\"";

            //Process configuration
            psi.UseShellExecute = false;
            psi.CreateNoWindow = true;
            psi.RedirectStandardOutput = true;
            psi.RedirectStandardError = true;

            //Execute process and get output
            var errors = "";
            var results = "";

            using (var process = Process.Start(psi))
            {

                errors = process.StandardError.ReadToEnd();
                results = process.StandardOutput.ReadToEnd();
                Console.WriteLine("ERRORS");
                Console.WriteLine(errors);
            }

            //Display to output
            return results;
        }
```

Listing 5.5: Metoden som används för att exekvera Python-scriptet

De olika metoderna kan sedan anropas i Main-funktionen.

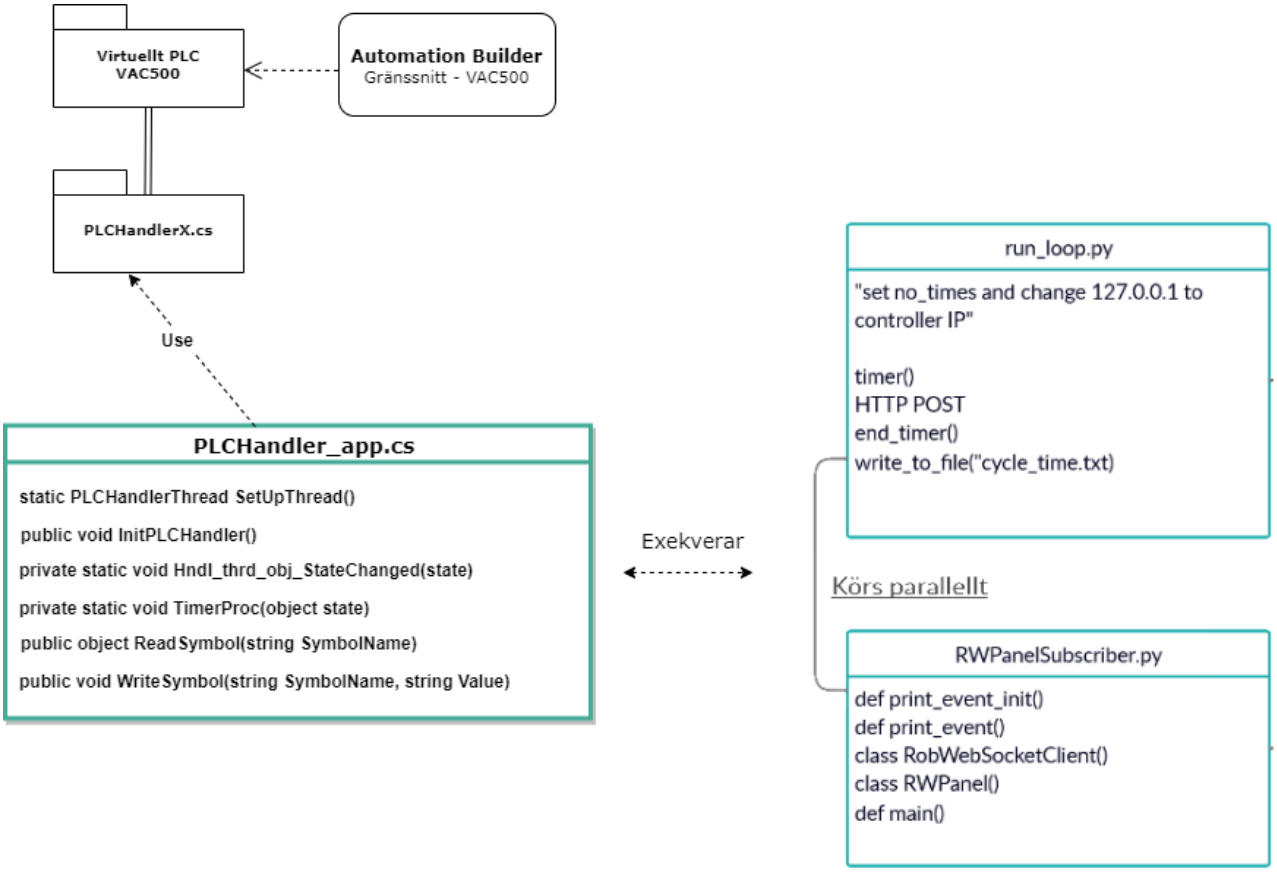
```
static void Main(string[] args)
        {
            hndl_thrd_obj = SetUpThread();
            hndl_thrd_obj.InitPLCHandler();


            Timer t = new Timer(new TimerCallback(TimerProc));
            t.Change(1, 0);
            Console.ReadKey();
        }
```

Listing 5.6: Main-funktionen

5.2.3 Diagram för hela systemet

Figur 5.1 visar det slutgiltiga systemet. Angivna är de viktiga funktionerna som användes för att upprätta en kommunikation mellan PLC:t och RobotStudio samt att skriva/läsa till/från PLC:t. Fullständig kod finns i bilagorna A, B och C. Filen 'cycle_time.txt' finns i bilaga D.



Figur 5.1: UML-diagram över hela systemet

5.2.4 Besvarande av frågeställning

Nedan besvaras studiens frågeställningar systematiskt, utifrån erhållna resultat:

Vilket kommunikationsprotokoll är lämpligt att använda för mobila nätverk inom industrin?

Inom tidsramen och med den utrustning som fanns tillgänglig inom projektet, kunde en lösning med RestAPI / Websockets tas fram för att kommunicera med ABB:s robotar. Dock uppfyller inte denna lösning de hårda realtidskrav som ställs inom industrin.

Vilken cykeltid kan uppnås för enkla signaler då 5G används som transportmedium?

Tabell 5.1 och tabell 5.2 visar cykeltiderna mellan robotcontrollern och python-scriptet (238 ms) respektive PLC:t (563 ms).

Vilken hårdvara och mjukvara behövs för att uppnå tidskraven för kommunikation inom industrin?

Proprietär hårdvara samt befintliga protokoll anpassade för just detta ändamål är vanligast. Om kommunikationen dock sker vid en tillräckligt låg nivå i mjukvaran, t.ex. om socketsservern skrivs i RAPID-kod inuti robotcontrollern, kan tidskraven möjligtvis uppfyllas. Detta för att fördröjningen i 5G-nätverket kommer ligga runt 1 ms och då kan de besparingar som görs i nätet kompensera för den relativt höga cykeltiden, med viss marginal.

6

Slutsats och diskussion

Vi lyckades skapa ett system där styrsignaler skickades till en industriell robot via 5G. Observera att inga riktiga 5G-modem användes. Istället användes de nyaste 4G-LTE Advanced. Värt att nämna också är att modemen ställdes in på att endast använda band 42 (3.5 GHz) frekvensen. Cykeltiden för hela systemet uppfyllde inte kravet för realtids-processer, men om de utlovade svarstider som 5G skulle medföra faktiskt uppnås de kommande åren - kan detta system utgöra en realistisk lösning. Dock tror vi inte att just Robot Web Services kommer vara användbart i detta sammanhang. Dessutom kan vi räkna med en halvering på fördröjningen om ett PLC i molnet används, vilket bevisades i ett tidigare examensarbete [2]. Däremot kan en ökning av anslutna enheter försämra vårt systems prestanda avsevärt eftersom ett högre antal enheter kräver mer synkronisering, så att data från de olika enheterna inte krockar med varandra.

Eftersom 5G är en ny teknik som fortfarande håller på att utvecklas försvårade det väldigt mycket för oss t.ex. då vi skulle installera modemen eller när vi skulle utföra tester. Detta på grund av inkonsekvent täckning och att förutsättningarna kunde ändras plötsligt när nätverket t.ex. uppdaterades. När 5G är lite mer etablerat eller åtminstone konfigurerat för just det användningsområde som det är avsett för, kan man med stor sannolikhet få ytterligare en minskning i svarstiderna.

6.1 Tillämpningsområden för 5G

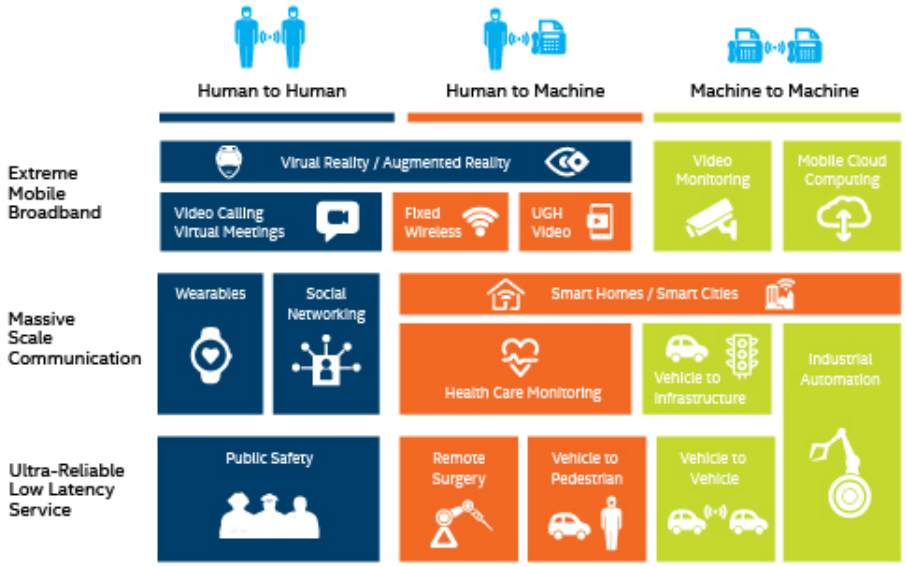
Även om realtidskravet för cykeltiden inte uppnåtts i detta arbete är det fortfarande rimligt att använda 5G på *cellnivå* där cykeltider på 100 ms accepteras, se figur 2.1.

6.1.1 Övervakning av processflöde och komponenter

IIoT och industri 4.0 kommer att kräva ständig uppkoppling till Internet, det talas om att de flesta vardagliga föremål kommer att kunna hålla *metrics* om dess användning så att de kan anpassa sig till användaren. Ett vanligt föremål skulle t.ex. kunna mäta hur ofta och hur länge det används, när det behöver underhållas och dylikt. Dessutom skulle de olika föremålen kunna kommunicera med varandra. Detta uttrycker sig i industriella sammanhang på ett liknande sätt där olika givare och don kan kommunicera inte bara med varandra utan även centralt. Kommunikation mellan dessa kommer att underlättas avsevärt med en trådlös uppkoppling.

6.1.2 Självkörande enheter (AGV)

Ett annat användningsområde som kommer att tjäna på trådlös uppkoppling är självkörande enheter som t.ex. transporterar material i en fabrik och eventuellt även människor. I dagsläget är det svårt att upprätthålla en konstant uppkoppling på grund av störningarna som uppstår



Figur 6.1: Olika användningområden med 5G [14]

i industriella miljöer, men 5G lovar såväl högre bandbredd som privata frekvensband för att minska störningarnas påverkan.

6.1.3 Virtuellt PLC i molnet

Mjukvarubaserade PLC:er visar hög potential och även om de inte kan användas för närvarande i processer med hårda realtidskrav, representerar de en viktig utveckling i att kunna centralisera flera PLC:er och därmed eliminera behovet av specialiserad hårdvara.

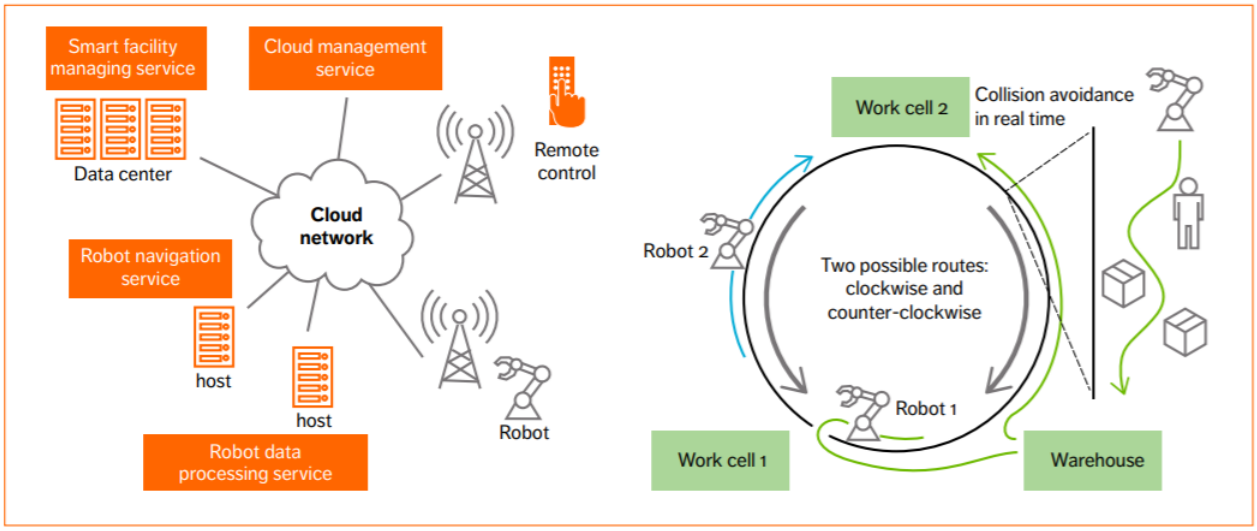
6.1.4 Fjärrstyrning

Ett tillämpningsområde som vi inte hade tänkt på alls var i fjärrstyrning av robotar. Nya *haptic* enheter som ger återkoppling på t.ex. hur hårt eller mjukt ett föremål är litar sig på en stabil och fördröjningsfri anslutning för att kunna motverka användarens rörelse om denne stöter på något hårt föremål i programmet.

6.2 Framtidsutveckling av 5G

6.2.1 Frekvensband

5G kommer att kunna erbjuda ett brett frekvensspektrum och förväntas fungera mellan 700 MHz och 100 Ghz. Band 3 är benämningen för det vanliga LTE-nätverket och Band 42 anser det nya höga frekvensbandet. De höga frekvenserna har som nackdel att vågorna har svårt att ta sig igenom hårda medier t.ex. väggar. Fördelen är den förväntade höga bandbredden samt låga responstiderna. 5G har två olika typer av frekvensband, licensierad och olicensierad.



Figur 6.2: Hur det ser ut med logistik i en fabrik [8]

Det olicensierade spektrumet kan användas av vem som helst, på ett liknande sätt som WIFI i dagsläget, det lisensierade sprektrumet å andra sidan är reserverat för implementation i valda anläggningar för att skapa en s.k. 'private network' - detta förväntas minska störningar i industriella miljö avsevärt [15].

6.2.2 Integration med annat teknik

Det som är mest spännande är den utveckling som pågår just nu inom 5G just där det gäller realtidsaspekten. Version 17 av 5G specifikationen kommer att ha *native*-stöd för Ethernet utan att behöva tunnla sig fram. Dessutom kommer begreppet Time-Sensitive Networking (TSN) integreras, vilket kommer att föra industrin ett steg närmare det s.k. ultra-reliable low-latency communication (URLLC) [16].

6.3 Vidare utveckling av det befintliga systemet

6.3.1 Framtidsutveckling av fältbussprotokoller

Nya protokoll utvecklas löpande och även om de inte är helt nya, kan de återigen bli användbara för olika ändamål. Om man tänker på protokoll som ryms inom den befintliga internetinfrastrukturen, är ett exempel s.k. WebRTC (Web Real-Time Communication) som använder sig av Real-time Transport Protocol (RTP) för att hantera realtids-aspekten, vilket i sin tur använder sig av UDP för att kunna uppehålla den snabba kommunikationen som krävs [17]. Detta utvecklades ursprungligen för webb-baserade applikationer men kommer eventuellt kunna användas även för detta ändamål, även om den är skriven i C++ och JavaScript.

Dessutom pågår förändringar i industrin. Vid det laget har rt-Labs släppt öppen källkod för ProfiNet-stacken som kan skicka Ethernet-ramar i andra skiktet i OSI-modellen och därmed uppfylla realtids-kravet. Den är skriven i C och kan snabbt bli en banbrytare i strävan mot standardisering inom industrin [18].

6.3.2 Machina-biblioteket

Innehåller mycket som är väldigt användbart och är dessutom skriven i C#, vilket gör det kompatibelt med vårt utvecklade program. Den använder sig av Sockets för kommunikation med RobotStudio men i detta fall skapas en Socket-server in i RobotStudio genom att skriva RAPID-kod. Jämfört med programmet som utvecklades i detta arbete så kan en Sockets-server användas för att upprätta kommunikation mellan roboten och klienten och därmed slipper man använda RWS, som verkar vara för långsamt för att kunna används i realtidsapplikationer. Något som är intressant att analysera i framtiden är huruvida detta bibliotek, eller åtminstone ett liknande upplägg, eventuellt kan sänka cykeltiden ytterligare. En fördel med detta bibliotek ät att styrningen av en robot kan göras med handkontroller via 5G och detta kan vara en intressant demo att visa upp.

6.3.3 Säkerhet och kryptering

En viktig aspekt som behöver tas i anspråk är säkerhet. Upplägget som har utvecklas under detta arbetes gång innehåller inte någon kryptering av meddelanden som skickas över nätverket. Detta kan dock tilläggas utan större ändringar i systemet eftersom det går att använda HTTPS (Hypertext Transfer Protocol Secure) och WSS (WebSocket Secure). Detta innebär i praktiken att ett extra lager, TLS (Transport Layer Security), adderas och möjliggör säkrare transport över nätverket.

6.3.4 Miljö och hållbarhet

Förhoppningen är att uppkopplade fabriker kan leda till en så pass stor effektivisering att det kan leda till en minskad miljöpåverkan - även om den totala produktionen eventuellt kan komma att öka. Som tidigare nämnt är övervakning av processflöden och komponenter en slags effektivisering där begreppet ”predictive maintenance” - som förutser när komponenter behöver service, vilket medför att fel kan åtgärdas direkt - eventuellt kan leda till återanvändning av vissa komponenter och maskindelar och på det sättet ge mindre spill i produktionen. Detta uttrycker sig också i att plötsliga stopp i produktionen därmed kan undvikas och på så sätt kan oplanerade stopptider minskas rejält. Dessutom är det virtuella PLC:t en utveckling som kommer att kunna ta över styrningen av processer som i dagsläget hade krävt flera PLC:er för att kunna styras. Detta har flera fördelar: det leder till minskad produktion av PLC:er, mindre resurser kan tilldelas till installationen av de olika PLC:er och slutligen kommer behovet av kablar i industriella miljöer minskas.

En möjlig nackdel med trådlös kommunikation är att det medför konstanta radiovågor omkring människan, och i synnerhet med 5G:s ankomst kommer vågtätheten i samhället bli ännu högre. Det finns ingen fastställd process som tyder på att människans hälsa påverkas negativt av detta, men eftersom 5G kommer använda nya frekvensområden, behöver dess påverkan ändå undersökas. Myndigheter rekommenderar såväl framtida epidemiologiska studier som forskning för att karlägga påverkan av andra faktorer i samband med elektromagnetiska fält [19].

Referenser

[1] F. C. Mattern F. *From the Internet of Computers to the Internet of Things*. Springer, 2010. ISBN: 978-3-642-17226-7.

[2] O. Hall och R. Lindy. "Trådlös kommunikation över mobilt nätverk i industriell miljö". Bachelor's Thesis. Chalmers Tekniska Högskola, 2018.

[3] O. Oesbeck. "Styr- och övervakningssystem". Kompendium. 2019.

[4] A. Ho. "Kommunikationen mellan två olika PLC:er med hjälp av trådlösa fältbussar". Bachelor's Thesis. Högskolan i Gävle, 2017.

[5] H. Networks. *Industrial Ethernet is now bigger than fieldbuses*. URL: https://www.hms-networks.com/press/2018/02/27/industrial-ethernet-is-now-bigger-than-fieldbuses. (accessed: 01.06.2019).

[6] V. Beal. *TCP-Transmission Control Protocol*. URL: https://www.webopedia.com/TERM/T/TCP.html. (accessed: 28.05.2019).

[7] M. Lara. *Websockets for real time web and IoT applications*. URL: https://www.netburner.com/learn/websockets-for-real-time-web-and-iot-applications/. (accessed: 22.05.2019).

[8] E. M. Report. *The power of 5G is here and will continue to spread across the globe in the coming years*. URL: https://www.ericsson.com/en/mobility-report/reports/june-2019. (accessed: 01.09.2019).

[9] T. 3. G. P. P. (3GPP). *5G Releases*. URL: https://www.3gpp.org/release-15. (accessed: 04.09.2019).

[10] D. J. Jakobsen. *The benefits of 5G in the factory"*. Whitepaper. Development Manager - HMS Labs, 2019.

[11] *Robot Web Services manual. REST API för kommunikation med kontrollern*. ABB Group, 2016. URL: http://developercenter.robotstudio.com/webservice/api_reference.

[12] G. D. Castillo. *Machina - Real Time Robot Control*. Youtube. 2017. URL: https://www.youtube.com/watch?v=-8rCfHdKsPY. (accessed: 01.05.2019).

[13] G. D. Castillo. *Machina.NET*. URL: https://github.com/RobotExMachina/Machina.NET. (accessed: 05.04.2019).

[14] R. Yallapragada. *5G Services and use cases*. URL: https://blogs.intel.com/technology/2017/12/5g-americas-white-paper-explores-5g-services-use-cases-and-market-implications/#gs.7qvs64. (accessed: 09.05.2019).

[15] S. P. Erik Dahlman och J. Sköld. *5G NR: The Next Generation Wireless Access Technology*. Mara Conner, 2018. ISBN: 978-0-12-814323-0.

[16] 3GPP. *Release 17 - key milestones*. URL: https://www.3gpp.org/release-17. (accessed: 05.10.2019).

[17] G. repository. *Websockets for real time web and IoT applications*. URL: https://en.wikipedia.org/wiki/WebRTC. (accessed: 05.10.2019).

[18] rt-labs. *Profinet open source stack*. URL: https://github.com/rtlabs-com/p-net. (accessed: 05.10.2019).

[19] S. S. C. on Electromagnetic Fields. *Recent Research on EMF and Health Risk"*. Report. Strålsäkerhetsmyndigheten, 2019.

Bilagor

A

Pythonkod för koppling till virtuell robotkontroller

Innehåll i filen RWPanelSubscriber.py”

```
#IMPORTS
import argparse
import sys
import xml.etree.ElementTree as ET
from timeit import default_timer as timer

import requests
from requests.auth import HTTPDigestAuth
from ws4py.client.threadedclient import WebSocketClient

#GLOBALS
ip_adress = '127.0.0.1'
start_timer = 0
end_timer = 0
RobotReady = 0
Start_seq = 0
namespace = '{http://www.w3.org/1999/xhtml}'

def print_event_init(evt):
    # Parsing XML-file to get the needed information
    root = ET.fromstring(evt)

    if root.findall(".//{0}li[@title='RobotReady']".format(namespace)):
        print ("\tRobotReady state : " + root.find(".//{0}li[@title='RobotReady']/{0}span".format(
            namespace)).text)
    if root.findall(".//{0}li[@title='Start_seq']".format(namespace)):
        print ("\tStart_seq state : " + root.find(".//{0}li[@title='Start_seq']/{0}span".format(
            namespace)).text)


def print_event(evt):
    global start_timer, end_timer
    # Parsing XML-file to get the needed information
    root = ET.fromstring(evt)

    if root.findall(".//{0}li[@title='Start_seq']".format(namespace)):
        global Start_seq
        print ("\tStart_seq state : " + root.find(".//{0}li[@title='Start_seq']/{0}span".format(
            namespace)).text)
        Start_seq = root.find(".//{0}li[@title='Start_seq']/{0}span".format(namespace)).text
        if Start_seq == '1':
            start_timer = timer()
            # print ("Start time: {}".format(start_timer)) #FOR DEBUG

    if root.findall(".//{0}li[@title='RobotReady']".format(namespace)):
        global RobotReady
        print ("\tRobotReady state : " + root.find(".//{0}li[@title='RobotReady']/{0}span".format(
            namespace)).text)
        RobotReady = root.find(".//{0}li[@title='RobotReady']/{0}span".format(namespace)).text

        if RobotReady == '1':
            end_timer = timer()
            # print ("End time: {}".format(end_timer))  #FOR DEBUG
```

```
            # print "Time elapsed = ", end_timer - start_timer, "seconds" #FOR DEBUG
            data = {'lvalue': '0'}
            reset_Start_seq = requests.post('http://{0}/rw/iosystem/signals/Start_seq?action=set'.
                format('ip_adress'),
                                            data=data,
                                            auth=HTTPDigestAuth('Default User',
                                                                'robotics'))


# This class encapsulates the Web Socket Callbacks functions.
class RobWebSocketClient(WebSocketClient):
    def opened(self):
        print ("Web Socket connection established")

    def closed(self, code, reason=None):
        print ("Closed down", code, reason)

    def received_message(self, event_xml):
        global end_timer, start_timer
        if event_xml.is_text:
            print ("Events : ")
            print_event(event_xml.data.decode("utf-8"))
            if RobotReady == '1' and Start_seq == '1':
                elapsed_time = end_timer - start_timer
                print ("\tElapsed time =  {} ms".format(elapsed_time * 1000))
                f = open("cycle_time.txt", "a")
                f.write("{}\n".format(elapsed_time * 1000))
                f.close()
        else:
            print ("Received Illegal Event " + str(event_xml))


# The main RobotWare Panel class
class RWPanel:
    def __init__(self, host, username, password):
        self.host = ip_adress  # Set this to the IP of the robot or the computer running
            RobotStudio
        self.username = username
        self.password = password
        self.digest_auth = HTTPDigestAuth(self.username, self.password)
        self.subscription_url = 'http://{0}/subscription'.format(self.host)
        self.session = requests.Session()

    def subscribe(self):
        # Create a payload to subscribe on RobotWare Panel Resources with high priority
        payload = {'resources': ['1', '2'],
                   '1': '/rw/iosystem/signals/Start_seq;state',
                   '1-p': '2',
                   '2': '/rw/iosystem/signals/RobotReady;state',
                   '2-p': '2'
                   }

        # Print events
        resp = self.session.post(self.subscription_url, auth=self.digest_auth, data=payload)
        print ("Initial Events : ")
        print_event_init(resp.text)
        if resp.status_code == 201:
            self.location = resp.headers['Location']
            self.cookie = '-http-session-={0}; ABBCX={1}'.format(resp.cookies['-http-session-'],
                resp.cookies['ABBCX'])
            return True
        else:
            print ('Error subscribing ' + str(resp.status_code))
            return False

    def start_recv_events(self):
        self.header = [('Cookie', self.cookie)]
        self.ws = RobWebSocketClient(self.location,
                                     protocols=['robapi2_subscription'],
                                     headers=self.header)
```

```
        self.ws.connect()
        self.ws.run_forever()

    def close(self):
        self.ws.close()


def enable_http_debug():
    import logging
    # import httplib
    # httplib.HTTPConnection.debuglevel = 1
    logging.basicConfig()  # Initialize logging
    logging.getLogger().setLevel(logging.DEBUG)
    requests_log = logging.getLogger("requests.packages.urllib3")
    requests_log.setLevel(logging.DEBUG)
    requests_log.propagate = True


def main(argv):
    try:

        parser = argparse.ArgumentParser()
        parser.add_argument("-host", help="The host to connect. Defaults to localhost on port 80",
                            default='localhost:8080')
        parser.add_argument("-user", help="The login user name. Defaults to default user name",
            default='Default User')
        parser.add_argument("-passcode", help="The login password. Defaults to default password",
            default='robotics')
        parser.add_argument("-debug", help="Enable HTTP level debugging.", action='store_true')
        args = parser.parse_args()
        if args.debug:
            enable_http_debug()
        rwpanel = RWPanel(args.host, args.user, args.passcode)
        if rwpanel.subscribe():
            rwpanel.start_recv_events()

    except KeyboardInterrupt:

        rwpanel.close()


if __name__ == "__main__":
    main(sys.argv[1:])
```

B

Automatisk påverkan av insignal för att utföra cykeltester

Innehåll i filen run_loop.py”

```
#IMPORTS
from time import sleep

import requests
from requests.auth import HTTPDigestAuth
from timeit import default_timer as timer
#GLOBALS
ip_adress = '127.0.0.1'
no_runs = 10
i = 1

while i <= no_runs:

    data = {'lvalue': '1'}
    before = timer()
    set_Start_seq = requests.post('http://{0}/rw/iosystem/signals/Start_seq?action=set'.format('
        127.0.0.1'),
                                  data=data,
                                  auth=HTTPDigestAuth('Default User', 'robotics'))
    if set_Start_seq.status_code == 204:
        after = timer()
        req_time = after - before
        print("\tRequest time =  {} ms".format(req_time * 1000))

    i += 1
    f = open("cycle_time.txt", "a")

    f.write("{},".format(req_time * 1000))
    f.close()
    sleep(5)
```

C

Läsning av resultatet och skriv ut medelvärdet från antalet cykeltester

Innehåll i filen 'test_results.py'

```
sum_number = 0
openthefile = open("cycle_time.txt", "r")

for line in openthefile:
    for num in line.split(','):
        sum_number = sum_number + float(num.strip())

lines = 0
with open('cycle_time.txt') as f:
    for line in f:
        lines = lines + 1
#print("The sum of your numbers is %.1f" %(sum_number))
print("Number of cycles: {}".format(lines))
print("Total time: {} ms".format(sum_number))
print("Test result average: {} ms".format(sum_number / lines))
```

D

Testresultat cykeltester för olika transportmedium

Innehåll i filen 'cycle_time.txt' via samma datorn, medelvärde 15.0208 ms

HTTP request | Svar från roboten | Tre kolumner

11.0603851989,8.5527740617
8.76718750591,5.01738261134
9.6102590082,5.58231072883
8.46852091989,5.34197324099
8.91138999861,6.37245397537
10.0185626954,5.28904498524
11.0776678946,6.19260592266
9.09825914646,5.73299423245
9.92242770027,5.86801529302
9.61782018759,5.7259731373
9.4800987058,5.74649633849
8.77636893803,5.60391409851
8.84117904711,6.01869879664
9.30781183249,5.68816724032
9.63510288334,5.21289310709
8.42531418051,4.91476660531
9.08961779859,5.43648798339
12.3522667066,5.69950900942
8.75044489439,5.63523898458
8.33566019628,5.34791416763
9.70261341364,5.67196471306
9.32293419127,6.10025151724
9.20735616342,5.78322206698
8.62838585563,5.12755979676
9.03830979556,5.63739932156
8.42855468596,5.51480019854
9.23003970159,5.55854702216
9.23922113371,5.68978749305
8.95027606406,5.27608296343
8.95027606406,8.8806051968
9.09069796707,5.21181293857
9.09069796707,9.09069796707
8.78068961197,5.24475807737
8.74396388349,5.54180441065
8.7174997556,5.33387197731
11.5264779,6.1942261754
9.15604816039,5.51804070399
8.59220021139,5.69572841972
9.32941520218,5.08813364709
9.74906065848,5.75297734941
8.79581197075,5.64766092215
8.73478245137,5.22531504465
9.23490045977,5.81994779546
10.6699042917,5.4364879834
8.64674871986,5.288504901
8.79257146529,5.8982600106
9.2505629028,5.72759339002
9.22733928038,5.20317159071
8.61704408653,5.43918840461
9.83331380029,5.51912087248
12.0195748133,5.54450483187
10.9107818638,5.83128956455
9.5681324373,6.31142445599
9.18737304645,5.75675793911
8.68023394289,5.30308717555
9.69667248697,5.61525586761
8.93083303134,5.90960177971
8.46636058292,5.50075800822
9.06423383919,5.7940237518
9.33589621309,5.49211666035
8.55007364049,6.87041164682
8.72722127197,5.76971996091
9.6599467585,5.62875797365
8.83307778347,5.5553065167
9.37208185733,5.33441206161
10.9188831274,5.45755126885
9.49090039064,5.34737408341
8.71695967137,5.66278328094
9.51412401306,5.21451335979
8.94757564285,6.17370297419
10.2119128542,5.36033610523
8.98052078163,6.02355955482
9.49360081185,5.81400686879
10.4646722796,5.91446253787
10.7179717892,5.61741620458
9.41960927065,5.64982125912
8.72020017682,5.2912053222
9.90082433058,5.68708707184
8.49876563746,5.33333189311
9.40394682763,5.54558500035
8.39830996838,5.61039510943
9.11824226343,6.92279981832
8.81741534045,5.56448794883
10.5192207881,6.29036117054
9.37586244702,5.65792252276
10.4581912687,5.46025169007
9.00104398284,5.95388868756
8.94649547436,5.1648256095
8.83901871014,5.77566088758
9.38396371066,5.71301111547
9.33805655006,5.91716295908
8.70021705986,5.24205765615
9.27864728341,5.34305340948
8.68995545925,5.235036561
8.66133099441,5.49157657611
9.13552495918,5.37059770582
8.69751663864,5.41434452947
9.13984563312,5.26096060463
9.29322955794,5.22531504464
8.84982039499,5.19020956889

Innehåll i filen 'cycle_time.txt' via LAN, medelvärde 21,9237 ms

HTTP request | Svar från roboten | Tre kolumner

17.95659900000002,5.271893999974964 15.415415999996185,5.564740000004065 16.43079400000147,5.327999999998667
16.47253099999979,5.486053999959495 16.370583000004046,5.076208000005522 15.71578699999776,5.607845000000111
16.82627199999942,5.5962129999898025 16.31105599999927,5.575686000000246 15.869051999999328,5.214420000001496
15.654206999998976,5.06868200000099 16.131790999999396,5.337578999998982 15.806787999999017,5.510686000000931
15.638470000002513,4.907890999959363 16.17968600000097,5.60373999999797 15.981947999996748,5.1952620000008665
15.728787000000466,5.6762660000231335 16.529321999996682,5.210998999999106 16.229634000005433,5.371105999998349
15.827998999998982,5.340316000001621 15.558416999994051,5.39436900000112 15.723998000005679,5.238367000004018
16.913850999998203,5.335526000010304 18.604552999999967,5.425844000001234 17.54675299999997,5.331420999993952
16.420530000004874,5.351947999997719 16.376055999999473,5.2992629999977225 16.389740999999347,5.670793000007279
16.346636000001524,5.034470999987661 16.283686999999603,5.356738000003247 17.439330000000197,5.14941899999144
15.684997000001033,5.71663499999886 16.184475999999393,5.070734999989668 16.317214000000746,6.110745000000861
15.793787999996312,5.268473000000995 15.64873399999911,4.869573999997101 15.83141999999782,5.547633999995583
18.363708000000088,5.330052999966028 16.100316999999364,5.371105999998349 16.091422999998883,5.16857700000628
16.57790100000156,5.4272119999723145 17.819754999997883,5.18910400001004 75 15.942946999999208,5.079628999993702
15.593312000000026,4.53978199999483 17.880650000002163,5.453212000006147 17.67196399999804,10.786686000017198
15.44757399999952,5.646845999990546 16.161897000003478,5.286947999991298 15.793788000003417,5.4737380000062785
16.20705499999886,7.716602999948918 16.926167000001158,6.502116999996588 15.765050999995367,5.247946999986652
15.634365000000372,5.330735999962144 16.155054999998697,5.573634000000993 16.761954999999773,5.327999999991562
16.46569000000042,5.295157999967159 16.478005000003293,5.88084800000388 15.88889399999971,5.263683000009678
18.480025000002342,6.483644000013555 16.164633999999012,5.035155000001623 16.118791000003796,5.734424999985777
15.539942999993173,8.364556999993056 16.23100299999436,5.446369999987155 16.27342399999776,5.214420000015707
15.481784999998638,12.382279999997081 16.722954999998763,5.38410600000816 16.749640000000454,5.135050000006913
15.895051999997634,5.264367999984643 18.153654000000017,5.49357999997823 75 15.69457600000046,5.702266999975336
16.516322000001082,5.451158999960626 16.48963700000028,5.259577999993326 16.045580000000115,5.818583999996463
15.63436499999682,5.348525999977483 16.307636000000514,5.423105999994959 16.249477000000567,5.217157000004136
15.667892000003292,5.308841999976721 19.2908230000004,5.462106000010181 15.731523999999553,5.595529000004262
18.32949699999997,5.47168599999992 15.838946999998882,5.232210000002624 18.281601999999953,5.38068500000044
16.218002999999648,5.2356299999996025 16.291213999999776,5.220577999999421 15.78010299999999,5.246577999999502
16.291213999998888,5.093997999999544 16.623743999998553,5.320473000001158 16.1133179999986,5.277368000001559
15.796524000000645,4.979049999999319 15.681576000000419,5.685161999998911 16.628533000000445,5.319788999997854
15.559101000000908,5.2458940000015275 16.151634000003412,5.2623159999996005 16.640847999997987,5.401211999998878
15.694576000001348,5.679688000000738 15.48110100000244,5.0652609999986 16.595691000002688,5.203473000001679
16.45474199999697,5.111103000000838 16.433531000000556,5.469632999997032
16.675744000000492,5.924637999996207 15.943631999999042,5.198683000003257

Innehåll i filen 'cycle_time.txt' via lokal trådlös nätverk, medelvärde 38.3158 ms

HTTP request | Svar från roboten | Tre kolumner

26.59910600000001,4.7624549999832
24.323734000001096,0.72965300004964
35.297687999999994,4.646337000005
23.98294199999995,5.31495999999758
29.058644999999217,5.87556700003912
105.72942000000296,30.82309800006155
50.61769299999952,7.93004399991199
27.46539900000755,5.76106900001839
27.15971199999956,2.057716999995704
22.303282999999396,0.23115500869815
26.090887000000006,5.26959300008399
28.90526200000032,9.89108699999743
23.32079900000039,6.05649499995721
33.26049400000031,5.27013399999871
21.365158000001827,5.428917999784665
36.194766999997796,6.17639299975633
21.66706399999896,8.87087000009745
28.281464999992068,14.28736500001273
26.164878000003,4.93906300023225
23.37048700000821,5.07084200009179
53.73613499999408,5.73622500021022
34.36874499999476,6.70729600001861
28.128081999994947,7.96515000018184
23.53629300000648,5.20262300002404
23.039415999988933,7.47097300010872
33.550519000016266,3.72765500342468
22.404818000012483,6.2017770000113
36.60144900001683,3.36795999991399
29.006257999981244,4.32390800003485
39.394760999982736,1.162800000314387
26.07522499999959,4.476210999965105
25.628576000002568,4.27799999999338
24.770384000021295,4.82888600041166
40.27995799998507,2.699336999672723
30.22522500000946,5.61578699953798
24.951312000013104,5.65953399960331
28.258241999992606,5.73784499998742
30.277612999952908,5.95928000013223
36.43510500000957,4.84832800036872
24.58189499998298,3.036888999863313
23.379669000007652,0.23115505133072
29.18502500000386,9.83545799997743
160.81090299996958,18.379577000188
24.784966000026998,0.264100999706512
34.31203700000651,9.22084399969475
34.31203700000651,11.1365190046974
24.42148899996255,6.67597100049685
24.14820800004236,2.121447000359142
23.355904999959876,3.34743700021736
24.292949999960456,6.44967599988358
24.436611999988145,3.98365499956343
21.42564799999036,3.330153999683882
24.490620999984003,7.365657000036663
26.783273999967605,6.102941999984068
28.094596000016736,6.03219100031162
24.31077300002471,2.837598000419077
36.50423499999533,16.2430080000604
28.042749000007916,4.50429500060665
27.279610999983106,1.984806999852669
37.78261200000002,14.68000599999857
23.159315000000902,8.51333399999873
27.447035999998093,7.41156399998094
32.29968600000177,9.00264999999335
25.544863000000362,7.376999000016605
26.590464000001646,6.434012999996241
27.61932300000325,7.45855099999147
33.58940500000074,8.283798999997316
31.585154999987708,8.92541800004015
26.686060000002954,9.03127400008693
36.37515500000177,9.59134000000006
28.41378600000155,7.71617099999844
88.96685300000229,0.2435780000984753
26.234549000008656,7.70266800007407
28.174529000011717,7.4893360002033
26.33554499999491,8.33186600020227
26.524575000024697,8.40207699999508
29.51609600000893,6.968696000001273
39.29214500001876,4.284480999984908
23.75502699999288,9.40717200010178
33.929116999985354,0.2516790000155896
29.36555500000759,6.866079999980457
25.49949599998058,3.725494999997636
29.52527799999416,12.489428000009184
25.485994000007395,7.065371000101455
30.14259300002209,9.517888999994284
29.197986999975,6.976257000019359
34.718178999980864,9.037214999978005
27.532909999990807,9.167915999967136
40.66881800002875,6.018689000029553
31.273527999985617,8.081267000022763
28.324132000022928,8.435021999957826
25.99475199997414,6.445355000039399
29.220130999988214,7.226314999968508
27.826175000029707,9.453618999998525
24.403126999970937,6.889304000026186
26.481366999973943,10.460874999978387
23.671853999999993,8.620270999999846
36.27902000000027,2.4049909999988017
25.84028900000135,8.948640999999924
32.738774000002024,7.008122000002004

Innehåll i filen 'cycle_time.txt' via 5G, medelvärde 238.25 ms

HTTP request | Svar från roboten | Tre kolumner

241.19275499999992,1.400592999999617 229.482376,9.70493800027141 244.5741569999882,2.3636449999694378
245.72227399999846,9.442198000002122 242.8745610000007,1.3410660000090502 228.39378599999804,5.411540000491718
250.249740000001,1.3841709999979912 245.71337899999435,1.607227000222488 203.1205510000001,259.0768269999999
242.2129230000003,1.3342230000006339 237.41929700000242,9.110352999982751 237.54929800000113,4.238726000000526
227.18135299999886,1.774859999976022 203.574871000003,1.328066000041872 225.5461010000017,1.540857000019495
229.39411200000137,1.358171999996216 222.1270679999981,1.3198550000197429 264.7079350000006,14.704514000001723
227.19161600000604,1.373908999944554 206.39245099999528,1.482699000074147 278.41691300000093,4.094062999964813
229.1211090000047,1.336960999992698 213.8670320000074,9.7179379999432 246.79786299999762,3.871692999934564
226.4293979999934,1.3807509999992362 247.81529399999158,1.45464600007017 213.6277479999987,1.5093829999983654
214.3887339999975,1.3410660000090502 204.71370999999294,1.262380999467994 238.88899499999638,1.312327999973647
231.55350099999993,1.344486999983019 228.21588899998346,9.392935000050784 247.41707999999107,1.335592000037713
241.53349500000053,1.676331999881787 246.33806899998945,1.344486999983019 214.33777300000116,3.20291399999996
255.07894500000106,1.406751000188577 264.88309499998763,1.315750000033906 202.050695,2.042062999999871
240.84380400000072,9.690569000014193 227.01235099998485,2.18059999997422 213.8310290000011,14.37540600002755
204.8656330000034,1.3225919999797497 207.1939969999903,1.3164340000457742 211.2484659999992,1.297276000025119
214.89779099999964,1.317118000028926 240.0405330000126,1.3862240000435122 215.6548920000006,14.1831410000373
226.94940400000263,1.336960999992698 239.19962999999998,1.371855999991567 229.7343589999994,4.858626999997284
229.20458400000143,1.343802999974788 239.68884499999987,1.39785600003961 217.49406799999775,5.61400200001506
227.72804299999905,10.11478499998475 241.4164939999992,1.0235890000558356 234.63954100000024,1.38690800006306
200.37922099999776,1.313697000005022 231.308551999998,1.39785600003961 220.7636139999994,1.362275999994722
228.4437340000096,1.3561189999791168 226.44718799999453,1.340381999666666 256.12563300000147,0.858692000084319
246.50296499999058,1.351328999877998 225.85465600000276,1.497752000911682 222.55505900000344,15.35862599994047
228.15909999999917,1.820017999894317 226.57034699999912,1.334224000611127 236.25341799999603,13.7445570000965
223.9990589999934,10.223574000008284 226.27408099999968,9.66935899985889 220.41515599999343,1.464224999781143
246.30933200000982,1.729700999879265 240.791803999997,1.4553309999882913 237.0046610000145,1.5873840000040218
243.64156900000467,1.325328000071754 245.09005599999512,1.336960999992698 228.5405920000039,9.644042999980229
211.53349499999877,1.965756000041485 225.51459899999315,10.57389400003747 220.05079599999135,1.380066999852744
242.2218179999902,1.343118000022514 228.43552400000533,1.037956999994094 236.5081110000074,1.3882769999895572
227.2778280000125,1.3212230000476666 245.69764100000668,1.385540000011286 240.8363049999798,1.3184870000202409
224.65932900001917,1.469698999933932 245.630589000001,1.3006969999196372 292.508368,1.4183830000149555
259.198617,1.350645000022596 243.71067499998844,1.232275999960908 239.145932999985,2.366023999996969
237.4603499999992,1.3349079999898095 227.66441100000634,5.348519999999371224.03570499999995,4.43988600000011553
228.34999600000216,1.603121000016472 219.7658069999875,1.3246450000679033
246.6295460000012,1.383486999982402 246.95933800001058,3.302748999933101

E

PLC-kommunikation

```
using PLCHandlerX;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Net.Sockets;
using System.Net;
using System.Diagnostics;
using AxPLCHANDLERXLib;
using System.IO;

namespace PLCHandler_app
{
    class Program

    {
        //Setting up the PLC
        private static PLCHandlerThread hndl_thrd_obj;
        static PLCHandlerThread SetUpThread()

        private static void Hndl_thrd_obj_StateChanged(PLCHandlerThread.PLCHANDLER_STATE state)

        private static string oldValue = "";

        //Polls the PLC synchronously
        public static void TimerProc(object state)

        public static string ExecPython(string fromPLC)

        static void Main(string[] args)
        {
            hndl_thrd_obj = SetUpThread();
            hndl_thrd_obj.InitPLCHandler();


            Timer t = new Timer(new TimerCallback(TimerProc));
            t.Change(1, 0);
            Console.ReadKey();
        }
    }
}
```