CHALMERS

En kommunikationslösning för autonoma fordon

Examensarbete inom Data- och Informationsteknik

Nils Gangby
Arvid Wiklund

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

EXAMENSARBETE

En kommunikationslösning för autonoma fordon

Examensarbete inom Data- och Informationsteknik

NILS GANGBY

ARVID WIKLUND

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2018

En kommunikationslösning för autonoma fordon
Examensarbete inom Data- och Informationsteknik
NILS GANGBY
ARVID WIKLUND

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
417 56 Göteborg
Telefon: 031-772 10 00

Institutionen för Data- och Informationsteknik
Göteborg 2018

Sammandrag

Enligt Statens Väg- och Transportforskningsinstitut [1] är den mänskliga faktorn inblandad i 90-95 procent av alla dagens bilolyckor och 33-53 procent av fallen beror på förarens egna uppfattningar och normer. Dagens autonoma fordon reducerar inverkan av den mänskliga faktorn betydligt men eliminerar den inte. Teknologin är inte tillräckligt säker för att helt utesluta människans påverkan då det i dagsläget inte är säkert att låta bilen själv köra i alla miljöer och situationer med acceptabelt låg risk. För att reducera risken ytterligare kan det komma att krävas kommunikation mellan fordon. Examensarbetet har utförts i syfte att undersöka möjligheterna för en mjukvarulösning som hanterar kommunikation mellan fordon. Metoden omfattar förarbete i form av utvärdering av möjligheter, systemarkitektur och implementation. Av förarbetet framkom intressant data och många aspekter. Dessa har utvärderats och applicerats i en applikation. Slutresultatet är en mjukvarulösning för kommunikation mellan fordon där datasäkerhet beaktas. Alla fordon kommunicerar även med en server vars funktion är att underlätta kryptering och verifiering samtidigt som den ger upphov till goda möjligheter för vidareutveckling. Förslag på vidareutveckling kan vara positionsbaserad beslutsfattning och kommunikation med infrastruktur.

Nyckelord: Autonoma fordon, Kommunikation

Abstract

According to Statens Väg- och Transportforskningsinstitut [1], the human factor plays a part in up to 95 percent of today's car accidents where 33-53 percent of all recorded cases is due to the individual's own perceptions and standards. In today's car industry there are autonomous vehicles that reduce the human factor. These autonomous vehicles do not eliminate it though as the technology is not yet stable nor safe enough to be used in certain environments and situation. Communication between vehicles may need to play a role in the future to reduce the risk and increase the reliability. This thesis aims to examine the possibilities of a software solution that handles the communication across multiple vehicles. The methodology encompasses a pre-study to evaluate the possibilities, system architecture and an implementation. The result is a software solution that handles communication between multiple vehicles where data security is taken into consideration. The vehicles also communicate with a server which purpose is to facilitate the encryption of data and verification of units. Further development ideas are position based decision making and communication with infrastructure.

Keywords: Autonomous vehicles, Communication

Förord

Denna rapport är ett examensarbete inom mjukvaruutveckling på datateknikprogrammet vid Chalmers Tekniska Högskola (CTH) i Göteborg. Examensarbetet utfördes våren 2018 och beskriver utvecklingen av en mjukvarulösning för bil-till-bilkommunikation.

Särskilda tack riktas till:

Joachim von Hacht, handledare på CTH, för vägledning under examensarbetets genomförande.

Jonathan Gustavsson, handledare på Infotiv AB, för vägledning under examensarbetets genomförande.

INNEHÅLLSFÖRTECKNING

Ordlista

Agil - En iterativ arbetsmetodik som vanligtvis användst vid mjukvaruutveckling.

Scrum – En agil arbetsmetod.

Device to device – Direkt kommunikation mellan ett flertal enheter utan användandet av en mellanhand.

1. Inledning

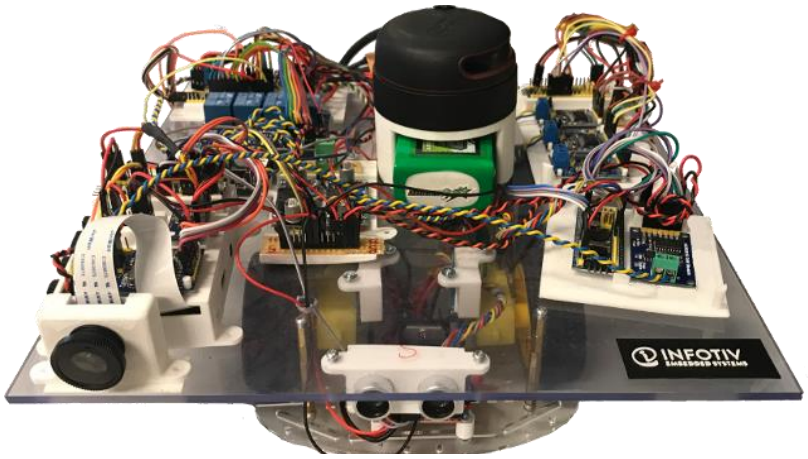
1.1 Bakgrund

Autonoma fordon, det vill säga självkörande fordon, är ett aktuellt område inom transportsektorn. Inom området finns det stor utvecklingspotential och många tänkbara fördelar. Övervägande delen av dagens trafikolyckor sker på grund av den mänskliga faktorn. Om fordon hade kunnat köra helt autonomt hade antalet olyckor på vägen kunnat minska. Man hade dessutom kunnat tänka sig att transportsektorns miljöpåverkan hade sjunkit.

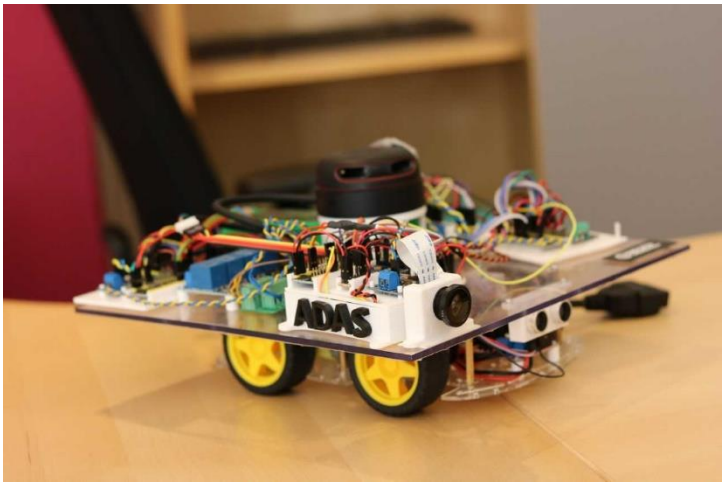
I dagsläget har man lyckats ta fram delvis autonoma fordon. Dessa tar hjälp av sensorer och kameror för att skapa sig en uppfattning om sin omgivning. Det finns dock många situationer där föremål eller händelser inte registreras av sensorer eller kameror. Exempelvis kan det bli problematiskt vid en vägkorsning där en häck eller mur blockerar synen för både kameror och sensorer. För att kompensera för dessa brister är uppfattningen, inom fordonsindustrin, att fordon kommer att behöva kommunicera med varandra.

Ett problem med kommunikation mellan fordon är hur informationens integritet ska bevaras. Om fordon ska kommunicera med varandra och kunna lita på att informationen som tas emot är korrekt måste även säkerheten av meddelandet garanteras.

Infotiv är ett konsultföretag som jobbar inom kärnkraft, medicinteknik och inbyggda system med fokus mot fordonsindustrin. Infotiv har under 2017 utvecklat en bilplattform, se figur 1.1 & figur 1.2, som är tänkt att spegla kommunikationsdelarna av en fullstor bil. Plattformen är även utrustad med liknande komponenter såsom kamera och sensorer. Bilplattformen har även stöd för nätverkskommunikation.



Figur 1.1: Bild på bilplattformen.



Figur 0:1.2: Bild på bilplattformen.

1.2 Syfte

För att åtgärda olika problem inom autonom körning vill Infotiv undersöka möjligheterna för kommunikation mellan autonoma fordon. Kommunikation mellan autonoma fordon kommer vidare benämnas bilkommunikation. Företaget har tagit fram ett examensarbete i syfte att utvärdera och implementera ett exempel på hur bilkommunikation skulle kunna hanteras. I examensarbetet ingår att avgöra vilken typ av data som är relevant att kommunicera, hur datan säkert kan överföras samt hur olika data kan tolkas och användas.

1.3 Mål

Målet är att implementera en säker mjukvarulösning för att hantera bilkommunikation. Lösningen ska kunna integreras med Infotivs befintliga bilplattform. Mjukvarulösningen ska av säkerhetsskäl kryptera data och verifiera användare.

1.4 Avgränsning

Examensarbetet kommer endast omfatta överföring av data.

1.5 Metod

Nedan är en punktlista på de steg som genomgicks i examensarbetet.

Instudering och förarbete:

- Positionering
- Kommunikation
- Kryptografi

Implementation:

- Läsa av meddelande som skickas på tidigare implementation av bilplattformen
- Undersöka och testa hur ZeroMQ fungerar
- Prioritering
- Skicka meddelande mellan två bilplattformar
- Identifiera ett meddelande från bilkommunikation
- Få bilplattformar att kommunicera med server

2. Teori

Här tas den teori som ligger till grund för examensarbetet upp.

2.1 Datornätverk

Ett datornätverk utgörs av sammankopplade tekniska enheter som kan överföra data till varandra.

2.1.1 TCP

Transmission Control Protocol (TCP) är ett dataöverföringsprotokoll som garanterar att den data som skickas kommer fram till mottagaren. [2] För att säkerställa att inga meddelanden försvinner går varje överföring igenom tre faser; anslutning, dataöverföring och nedkoppling. Dessa tre faser upprättar en så kallad session som gör att TCP klassas som ett förbindelseprotokoll.

Anslutningsfasen innebär att klienten skickar ett så kallat SYN-segment till servern. Om servern godtar anslutningen svarar den med ett SYNACK-segment. Klienten svarar återigen med ett ACK för att avsluta anslutningsfasen.

I dataöverföringsfasen görs ingen skillnad på server och klient, dock måste mottagaren skicka tillbaka ett ACK-segment för varje mottaget paket. Får inte avsändaren tillbaka ett ACK-segment implicerar det att paketet måste skickas om.

I avslutningsfasen skickas ett FIN-segment som berättar för den andra parten att all data är överförd. FIN-segmentet bekräftas med ett ACK-segment och efter det stängs kommunikationen ner.

Nackdelen med TCP är att det tar lång tid att sätta upp och avsluta kommunikationen.

2.1.2 UDP

User Datagram Protocol (UDP) [3] är ett dataöverföringsprotokoll som är snabbare än TCP. UDP är till skillnad från TCP förbindelselöst vilket innebär att det inte skapas någon session. Detta medför att avsändaren inte kan garantera att mottagaren tagit emot paketet. Mottagaren kan inte heller garantera att den fått alla paket eller att paketen kommit fram i rätt ordning. Om mottagaren får fram korrupta paket ber den inte att paketet skall skickas om utan slänger istället paketet, alternativt skickar det vidare med en flagga satt.

2.1.3 DNS

Domain Name System (DNS) [4] förenklar IP-baserad kommunikation genom att koppla ihop IP-adresser med domännamn. Detta gör det möjligt att skicka meddelande utanför ditt lokala nätverk. I så kallade domännamnsservrar finns kopplingar mellan domännamn och adresser.

2.1.4 mDNS

Multicast Domain Name System (mDNS) [5] har många likheter med DNS. Paket som skickas är nästan identiska med de DNS använder sig av. Skillnaden är att mDNS dessutom har samma funktionalitet i ett mindre nätverk där en domännamnsserver inte nödvändigtvis finns tillgänglig för att lokalisera olika enheter.

2.1.5 Latens

Latens är ett mått på hur lång tid det tar för en mängd data att ta sig från punkt A till punkt B. [6]

2.1.6 Genomströmning

Genomströmning är ett mått på hur mycket data som kan skickas mellan punkt A och B inom en begränsad tidsperiod. [6]

2.2 Avståndsmätning

2.2.1 Radiofyr

En radiofyr är en radiosändare med kända, konstanta koordinater som kan användas för navigation. [7] Den skickar ut en radiovåg med en tidsstämpel på. När mottagaren får in en signal från radiofyren kan den med hjälp av den medskickade tidsstämpeln avgöra distansen mellan sig själv och fyren. Får mottagaren in signaler från minst tre olika radiofyrar samtidigt kan den göra en exakt estimering av sin egen position.

2.2.2 GPS

Global Positioning System (GPS) [8] är ett navigationssystem som använder sig av satelliter och radiovågor för att avgöra en mottagares position. GPS kräver flertal anslutningar samtidigt för att kunna avgöra en precis position. Som användare av GPS har man alltid kontakt med minst fyra satelliter samtidigt [9]. Till skillnad från radiofyrar, som kan ge en exakt estimering inom små områden, har GPS en felmarginal på 20 - 75 meter. [10]

2.2.3 Ultraljud

Ultraljud är ljudvågor vars frekvens är högre än 20 kHz. [11] Ultraljud används primärt för avståndsmätning. För att avgöra distans med hjälp av ultraljud skickas en ljudvåg med tidsstämpel ut, studsar på en yta och återvänder till avsändaren. När signalen kommit tillbaka jämförs tidsstämpeln mot nuvarande tid och distans till närmaste föremål kan bestämmas. Då ultraljud skickar signaler i ljudets hastighet istället för ljusets hastighet är det långsammare än radiofyrar och GPS. Dock är ultraljud en billigare lösning som används på kortare distanser vilket kompenserar för den lägre hastigheten.

2.3 Kryptografi

Kryptografi innebär att göra data svårläslig genom kryptering. [12] Med dekryptering kan man sedan återfå originaldatan. För att kryptera och dekryptera krävs att båda sidorna av kommunikationen känner till en hemlighet som används för att kryptera data. Detta är känt inom kryptografi som nyckel. Modern kryptering använder sig ofta av avancerade algoritmer för att garantera säkerhet.

Inom kryptografi finns även autentisering. Autentisiering innebär att endast de delar som måste vara privata för att motverka intrång är krypterade. Ett vanligt exempel på detta är vid inloggning på en webbplats. Som användare har man ofta ett användarnamn och lösenord. Användarnamnet är publikt och behöver inte krypteras. Lösenordet däremot är det som i kombination med användarnamnet identifierar ägaren av profilen. Lösenordet måste därför krypteras.

2.3.1 Symmetrisk kryptering

Vid symmetrisk kryptering används samma nyckel för att kryptera och dekryptera meddelanden. Detta medför att båda sidor vid ett informationsutbyte måste känna till nyckeln som används för att kunna läsa av meddelandet. Den vanligaste typen av symmetrisk kryptering är Advanced Encryption Standard(AES) [12].

2.3.2 Asymmetrisk kryptering

Den stora skillnaden mellan symmetrisk och asymmetrisk kryptering är att vid asymmetrisk kryptering används två olika nycklar, en privat nyckel och en publik nyckel. Den publika nyckeln används för att kryptera meddelanden medan den privata används för att dekryptera meddelanden. Genom att hålla den privata nyckeln gömd hos ägaren kan endast ägaren dekryptera meddelanden. Den vanligaste typen av asymmetrisk kryptering är RSA, som är namngett av dess upphovsmän R. Rivest, A. Shamir och L. Adleman [12].

3. Teknisk bakgrund

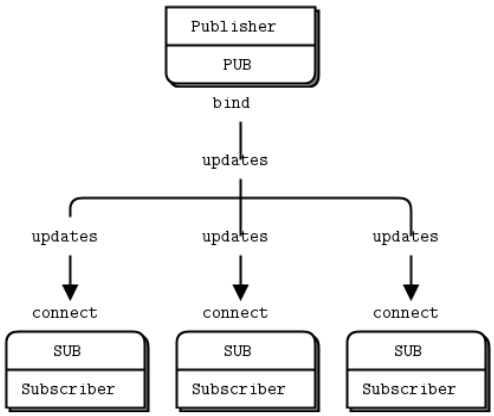
I detta kapitel förklaras tekniska begrepp som är relevanta för rapporten.

3.1 ZeroMQ

ZeroMQ är ett kommunikationsbibliotek vars syfte är att underlätta utvecklingen av nätverkskommunikation mellan enheter [13]. Biblioteket är byggt ovanpå TCP men använder sig av bättre doseringsalgoritmer för att öka genomströmningen. Det finns API:er för de vanligaste programmeringsspråken. ZeroMQ har möjlighet att skapa olika typer av sockets.

3.1.1 PUB/SUB

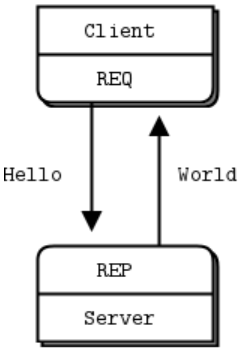
ZeroMQ kan kommunicera med hjälp av publisher och subscriber-sockets, se figur 3.1 [13]. Publishern skickar ut data på sin lokal port, som sedan en subscriber kan hämta information från. De som hämtar data gör det genom att på sin egen sida ha en lokal subscriber socket vars enda jobb är att hämta data och omdirigera den.



Figur 3.1: Beskriver PUB/SUB -protokollet i ZeroMQ.

3.1.2 REQ/REP

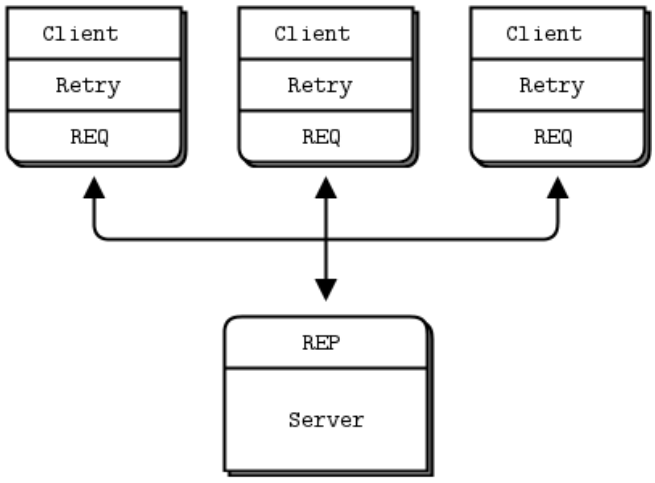
ZeroMQ kan även kommunicera med hjälp av request och reply-sockets, se figur 3.2 [13]. Request-sockets ansluter till en annan enhets port och skickar ut ett paket. Den skickar inga nya paket innan den har fått tillbaka ett meddelande som säger att paketet tagits emot. Paketet tas emot av en reply-socket som skickar tillbaka ett ok-meddelande om meddelandet tagits emot korrekt. Fördelen med denna struktur är att användaren enkelt kan avgöra om mottagaren tagit emot paketet och om det skulle uppstå komplikationer, skicka om samma paket.



Figur 3.2: Beskriver REQ/REP protokollet.

3.1.3 The Lazy Pirate Pattern

Då ZeroMQ är byggt ovanpå TCP kan man garantera att de individuella paket som skickas kommer fram. Det kan dock uppstå komplikationer om det är problem med socketen. Om en socket tappar koppling till mottagarsocketen kan anslutningen behöva göras om. Denna metod kallas The Lazy Pirate Pattern, se figur 3.3 [14] . Det går ut på att man kräver ett svar inom en förbestämd tidperiod. Får man inte svar från mottagaren inom den angivna tiden så stänger man ner förbindelsen, återskapar socketen och försöker igen (Retry). Man försöker ett fast antal gånger. Funkar det inte efter försöken tolkas det som att det är fel på mottagarsidan och socketen stängs ner.



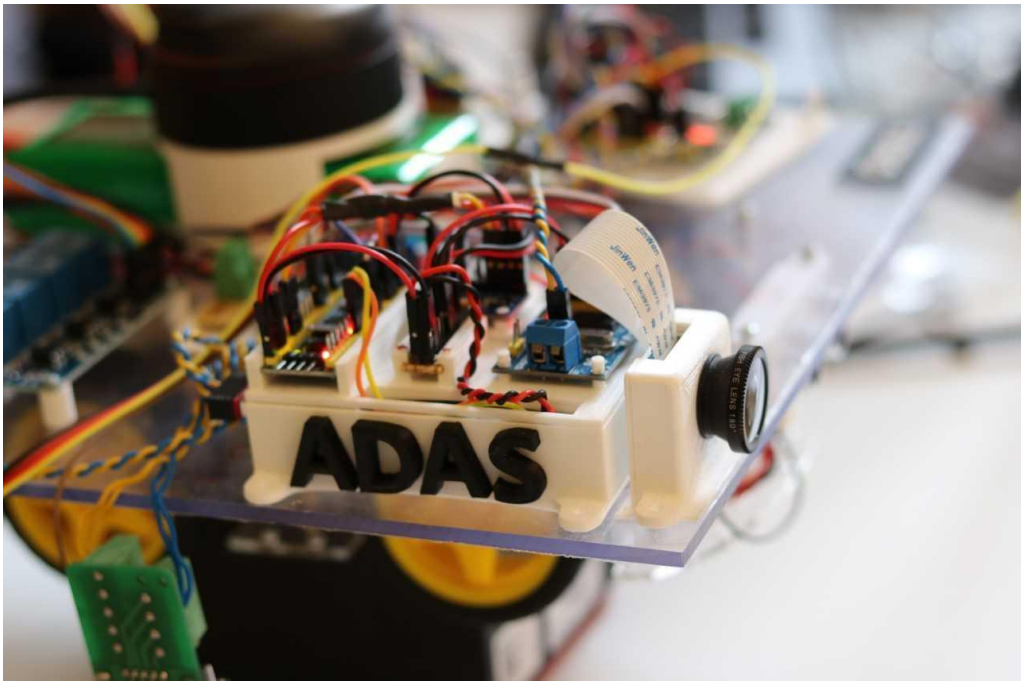
Figur 3.3: Beskriver The Lazy Pirate Pattern i ZeroMQ.

3.2 CAN

Controller Area Network (CAN) är ett kommunikationsprotokoll som används för kommunikation i fordon. [15] CAN skickar ut meddelanden på en databuss som sedan intresserade enheter läser av. Detta gör att samma meddelande kan skickas till flera noder samtidigt. CAN används ofta när det är höga krav på snabbhet och säkerhet.

3.3 ADAS

Advanced Driver Assistance System (ADAS), se figur 3.4, är en enhet på den använda bilplattformen som innehåller en Raspberry Pi, en CAN-transceiver och en mikroprocessor. Alla signaler som kommer till bilen från utomstående källor kommer till ADAS:en och hanteras där. Raspberry Pi:n är ansvarig för att hantera inkommande data från exempelvis en styrande dator eller en annan bilplattform. Den tolkar datan och skickar vidare den till CAN-transceivern som gör om datan till ett CAN-meddelande. CAN-meddelandet skickas sedan ut på CAN-bussen där bilplattformens komponenter kan läsa informationen. Signaler som inte kommer från utomstående källor, såsom sensordata skickas istället till mikroprocessorn. Detta eftersom denna data är realtidsbaserad och Raspberry Pi:n inte kan processa data tillräckligt snabbt.



Figur 3.4: ADAS-enheten på Infotivs bilplattform.

3.4 Python

Mjukvarulösningen kommer implementeras i Pyhton. Python stödjer både objektorienterad och procedurell programmering [16]vilket gör det till ett kraftfullt alternativ för olika typer av program. Python har en inbyggd kompilator. Den kompilerar inte koden till standard maskinkod utan istället till en byte-kod som körs via en tolk.

3.4.1 PyCrypto

PyCrpyto är Pythons krypteringsbibliotek [17]. Här finns moduler för de vanligaste krypteringsmetoderna, till exempel AES och RSA.

4. Genomförande

I detta kapitel kommer det redogöras för hur examensarbetet genomförts.

4.1 Scrum

Infotivs verksamhet använde sedan tidigare ett agilt arbetsflöde enligt metoden Scrum. Det var därför naturligt att jobba med Scrum även i detta examensarbete.

Sprintlängden bestämdes till en vecka. Under veckan hölls möten med produktägare och scrummästare vid två olika tillfällen. Det hölls även dagliga sprintmöten inom utvecklingsgruppen där mål för dagen bestäms.

Under produktägarmötena deltog alla examensarbetsgrupper samt projektledare. Här beskrevs vad som hade gjorts under den gångna veckan och vad som ska göras under nästkommande vecka. Det fanns även möjlighet att diskutera eventuella avvikelser tillhörande examensarbetet. Då flera examensarbeten utfördes på samma bilplattform diskuterades även frågor som berörde hela plattformen.

Under sprintmötena gicks den gångna sprinten igenom där man kartlade vad som gått bra respektive dåligt. Uppnåddes inte alla mål från förra sprinten diskuterades varför detta inträffat samt vad som behöver göras i framtiden. Efter detta planerades nästa sprint där mål för nästa vecka bestämdes.

4.2 Utvecklingsmiljö

För att utveckla mjukvara till bilplattformen krävs en dator med Python 3 installerat. När mjukvara är uppdaterad med ny funktionalitet skickas den över till bilplattformen med hjälp av Linux-kommandot secure copy (scp) [18] som för över filer med hjälp av en krypterad SSH-anslutning. [19] För att köra koden behöver utvecklaren, med hjälp av SSH, ta sig in på plattformen och manuellt starta programmet.

Att testa mjukvara på bilplattformen tar relativt lång tid på grund av överföringen. Det kan därför vara fördelaktigt att ha ytterligare en dator för att slippa föra över mjukvaran till bilplattformarna för att testa mer grundläggande funktioner. Kommunikationen kan testas mellan datorer lika väl som mellan bilar.

4.3 Instudering och förarbete

Här beskrivs de val som gjorts under förarbete och varför andra alternativ inte var optimala för examensarbetet.

4.3.1 Positionering

Initialt var tanken att med hjälp av en central server specificera bilens position. Positionen kunde sedan användas för att identifiera och meddela andra bilar i närområdet. För att kunna uppnå detta krävs någon typ av teknologi som kan spåra ett fordons position. Det som används i bilindustrin är mestadels GPS. GPS var inte en bra lösning då examensarbetet testades i ett kontorslandskap där det inte var acceptabelt med 20 - 75 meters felmarginal.

Infotiv hade sedan tidigare tänkt sätta upp radiofyrar för att kunna mäta avstånd på mindre skala. Fördelen med att använda radiofyrar hade varit att de fungerar liknande GPS vilket hade gjort att övergången till fullstor bil inte påverkat mjukvaran. Tyvärr hade Infotiv inte färdigställt konfigurationen av radiofyrarna så det var inte användbart för examensarbetet.

Det insågs senare att utan positioner blev det tämligen tama demonstrationer. Man kunde visa att bilarna fick in kommandon som kom från andra bilar, men avsakanden av position påverkade inte mottagarbilens handlingar. För att kunna skapa en bra demonstration implementerades mjukvarustöd för att avläsa och handla baserat på ultraljudssensorer. Det konstaterades att ultraljud inte är bra för att avgöra distans i bilkommunikation. Sensorerna som satt på plattformen hade enligt specifikation ett maximalt mätningsasvstånd på fyra meter men de flesta av dem kunde inte läsa längre än två meter.

4.3.2 Kommunikation

För en acceptabel kommunikationslösning krävs att fordon kan lokalisera varandra om de befinner sig i samma område. De behöver även kunna garantera att de fordon som försöker ta kontakt faktiskt är fordon.

För att kunna verifiera sig som fordon krävs någon typ av unikt ID. Vilka fordon som ska känna till vilka ID:n kan dock vara ett problem. För att kunna verifiera sig krävs att ett fordon har fått ett annat fordons ID samt att den känner till de ID:n som tillhör registrerade fordon. Om detta är publik information finns inget som hindrar utomstående från att skicka felaktig information.

I detta examensarbetet övervägdes två möjligheter för att lokalisera fordon. Ett sätt var att låta fordon själva söka efter andra fordon. Detta hade kunnat uppnås med mDNS.

Med mDNS kan fordon hitta varandra i trafiken utan hjälp från externa tjänster. Det fanns dock en avgörande begränsning med mDNS; det går inte garantera att ett fordon har hittat alla andra fordon i närheten. Utöver detta finns ingen skalbar metod för att verifiera de fordon som hittats. Det är inte hållbart att låta alla fordon ha information om alla andra fordon.

En annan lösning hade varit att ha en central server som känner till alla fordon och deras position. Serverns ansvar blir då att verifiera fordon och se till att de vet vilka andra fordon som de ska kommunicera med. Om fordonet kan lita på att servern är vad den utger sig för att vara så kan fordonet även lita på att den information som fås från servern är korrekt.

Om det ställs krav på att alla fordon måste vara uppkopplade till en server för att få vara aktiva i trafiken kan man garantera att alla närliggande fordon kommunicerar med varandra.

4.3.3 Kryptering

Som bestämdes i föregående kapitel så ska varje fordon ha ett unikt ID. För att kunna skicka ID:t säkert mellan fordonet och servern så måste den vara krypterad. Om ID:t är okrypterat kan alla som lyssnar på datatrafiken använda ID:t för att maskera sig som ett fodon.

Vid bilkommunikation är det fördelaktigt att kryptera all data då mycket av den data som skickas, exempelvis position, är privat information. Pågrund av detta var enbart autentisiering inte ett bra alternativ och resultatet blev att all data krypteras.

4.3.3.1 Biblioteksval

Eftersom all data ska vara krypterad föll ZeroMQ:s bibliotek bort. I ZeroMQ finns det stöd för autentisering av användare men för det här examensarbetet var det inte tillräckligt. Istället användes PyCrypto. Innan ett meddelande skickas via ZeroMQ krypteras meddelandet först, se figur 4.1.

Figur 4.1: När ett meddelande skickas via ZeroMQ krypteras det först.

4.3.3.2 Val av kryptering

Den lösning som används för att servern ska verifiera fordon är asymmetrisk kryptering. Eftersom asymmetrisk kryptering tillåter alla med den publika nyckeln att kryptera meddelanden, så kan alla skicka meddelanden till servern.

Det är dock bara ägaren av den privata nyckeln som kan dekryptera meddelanden, i det här fallet servern. Servern kan då verifiera innehållet av meddelandet och om det är korrekt accepteras avsändaren som ett fordon. Detta innebär även säkerhet för fordonet då svaret från servern måste innehålla rätt information. Är svaret oförståeligt, alltså att dekrypteringen gick fel, så vet fordonet att det inte är en verifierad server den kommunicerar med. I svaret från servern bör det finnas en symmetrisk nyckel som används för kommunikation mellan fordon.

En klar nackdel med asymmetriska kryptering är skalbarhet. Om varje fordon ska använda sin egen asymmetriska kryptering innebär det att varje fordon som den ska kommunicera med måste få motsvarande publika nyckeln. Att skicka nycklar fram och tillbaka medför stora mängder dataöverföring vilket inte är hållbart vid bilkommunikation. Av denna anledning togs beslutet att en gemensam symmetrisk nyckel ska användas för kommunikation mellan fordon. Den symmetriska nyckeln finns på servern och ska med jämna mellanrum bytas ut för att garantera säkerhet.

4.3.3.3 AES och RSA

Det finns flera olika stöd för symmetrisk kryptering i PyCrypto-biblioteket, där AES är ett av dem. Tyvärr är majoriteten av dem antingen inte klassade som säkra längre eller så är de långsammare än AES. Enligt PyCrypto är AES de facto-standard för symmetrisk kryptering vilket ledde till att AES är det som används i detta examensarbete. [17]

PyCrypto har stöd för tre olika asymmetriska metoder; DSA, ElGamal och RSA. DSA kan endast användas för signering, vilket inte uppfyller de krav på kryptering som ställts. ElGamal hade fungerat men datalängden på ett krypterat meddelande blir dubbelt så långt som motsvarande meddelande i RSA. På grund av skalbarhetskrav bör storleken på varje meddelande vara så litet som möjligt, vilket gjorde RSA fördelaktigt. RSA är det som används i detta examensarbete. [17]

4.4 Kravspecifikation

Ingen på företaget hade tidigare jobbat inom området bilkommunikation och det fanns få exempel på externa projekt. Till följd av detta allokerades mycket tid till efterforskning. Det var först senare, när implementationen påbörjats som en ordentlig kravspecifikation togs fram.

Följande är önskemål på kommunikationslösningen från Infotivs sida:

- Servern ska kunnas skicka meddelande till en eller flera bilar.
- Servern ska hålla koll på vart bilarna befinner sig och hur de går att nås.
- En bil ska kunna skicka meddelande till en central server.
- En bil ska kunna skicka till en eller flera bilar.

Kommunikationslösningen bör ha följande egenskaper:

- Multicast - En bil ska kunna skicka samma meddelande till flera bilar.
- Skalbart - Protokollet ska vara skalbart och kunna hantera ett stort antal bilar i nätverket. Detta kommer inte kunna testas under examensarbetet men ska tas i åtanke vid alla beslut som fattas angående konstruktion av mjukvaran.
- Kryptering - All kommunikation ska vara krypterad efter erkänd standard.
- Autentisering - En lista på aktiva fordon i nätverket ska finnas på en central server. Fordon som inte finns på listan ska inte få några meddelanden. Vilka fordon som tillhör nätverket bestäms i fabrik.
- Time To Live - Alla meddelanden ska ha en Time To Live (TTL). Ifall meddelandet inte skickas innan TTL utgår ska meddelandet kastas bort. Detta är för att fordon inte ska skicka utgången data som inte är aktuell längre.
- Priority – Olika typer av data ska ha olika prioritet. Detta är för att garantera att viktigare trafik skickas först.
- Delivery guarantees - Alla meddelanden ska komma fram till mottagaren. Om meddelandet inte kommer fram, ska det skickas igen.

4.5 Implementation

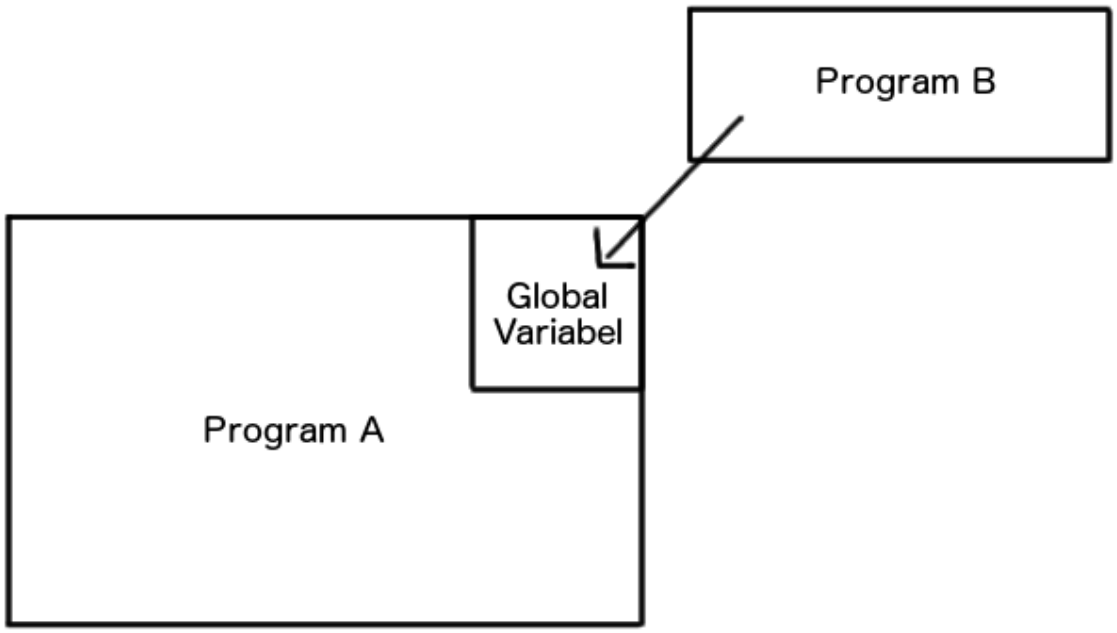
I det här kapitlet kommer all implementation att redovisas steg för steg.

4.5.1 PUB/SUB & REQ/REP

Den initiala tanken var att använda sig av ZeroMQ:s PUB/SUB. Ett problem uppstod dock i och med att publishern inte kunde få reda på hur många som tagit del av informationen eller identifiera de som gjort det. Detta medför att man inte kan kräva ett ACK från de som fått informationen och kan därför inte garantera att meddelandet har levererats till alla berörda. Lösningen blev att istället använda REQ/REP. Det negativa med REQ/REP är att varje socket fryser när den inte får tillbaka ett ACK. Detta gör att man måste skapa en ny tråd för varje meddelande som ska skickas ut och på det sättet uppnå viss mängd parallellitet.

4.5.2 Parallell utveckling

Det som för examensarbetet klassas som relevant information är de styrsignaler som hanterats på plattformen, till exempel hastighetsändringar eller riktningsändringar. Sådan data hade kunnat skickas vidare från programmet som kördes på bilplattformen innan. Detta hade dock inneburit att applikationen framtagen i examensarbetet potentiellt hade kunnat påverka den befintliga funktionaliteten. För att kunna utveckla helt fristående från tidigare programvara skapades en variabel. Variabeln uppdaterades varje gång det senaste relevanta kommandot ändras. Detta öppnade för möjligheten att dela en variabel mellan de två programmen och på så sätt utveckla helt fristående från tidigare mjukvara, se figur 4.2.



Figur 4.2: Program A är det program som fanns på bilplattformen innna detta examensarbete började. Program B är den mjukvarulösning som examensarbetet resulterat i. Program B läser av en variabel från program A.

Ett problem med att en variabel uppdateras varje gång någonting hanteras av ett fordon är att alla kommandon som kom från en annan bil även klassades som relevant data. Om man inte filtrerar bort den data som kommer från andra fordon kommer samma meddelande skickas tillbaka igen. Detta då det klassas som relevant information även på mottagarbilen. Lösningen blev att modifiera strängen av bytes som skickas till den befintliga koden genom att lägga till byten 0xff på plats 0. På mottagarsidan filtreras sedan alla meddelanden som börjar med 0xff bort. Lösningen öppnade även för att modifiera fler bytes, förutsatt att samma bytes modifierades varje gång.

4.5.3 Starta ZeroMQ socket

All kommunikation i examensarbetet sköts av kommunikationsbiblioteket ZeroMQ. För att skapa en anslutning på en port behöver följande steg göras.

- Skapa ett context. Samma context-objekt går att använda i hela programmet då det är trådsäkert:

  ```
  context = zmq.Context()
  ```

- Skapa en socket. Följande är exempel på en REP och REQ socket respektive:

  ```
  rep_socket = context.socket(zmq.REP)
  rep_socket.bind("tcp://0.0.0.0:5560")
  ```

  Ovan är en reply-socket (REP) som används för att lyssna på inkommande requests på porten 5560. 0.0.0.0 vilket är localhost fungerar för reply då socketen lyssnar på meddelanden på sin egna port. En reply-socket kan hantera flera olika användare.

  ```
  req_socket = context.socket(zmq.REQ)
  req_socket.bind("tcp://0.0.0.0:5560")
  ```

  Ovan är en request-socket (REQ) som används för att skicka REQ meddelanden till andra enheter. I detta exempel är socketen bunden på port 5560 och på localhost. Om man vill skicka till andra enheter hade en extern IP adress behövt anges. Exempelvis 192.168.x.x inom ett lokalt nätverk.

- När sockets är skapade kan man ta emot eller skicka meddelanden:

  ```
  message = rep_socket.recv()
  ```

  En reply-socket behöver köra receive som låser programmet tills ett meddelande tas emot.

  ```
  req_socket.send('message')
  ```

  Request-socketen kan skicka meddelande med hjälp av metoden send.

Genom att starta trådar för både REP och REQ -sockets kan man löpande ta emot och skicka meddelanden.

4.5.4 Prioritet på meddelanden

Då meddelanden varierar i hur tidskritiska de är det inte tillräckligt att implementera en kö. Python har stöd för så kallade prioritetsköer där varje element har en prioritet och hanteras olika snabbt av processorn. Prioritetsköer har dock inte stöd för att fasa ut gamla meddelanden. Om det konstant läggs till nya, högprioriterade meddelanden i prioritetskön kan det ligga gamla, lägre prioriterade meddelanden som aldrig omhändertas. Lösningen blev att implementera en Time To Live (TTL) -variabel som avgör om meddelandet är tidsrelevant eller inte. TTL-variabeln hade sedan minskats varje gång data hade hämtas från kön.

Prioritetsköer tar in en tuple som argument där första värdet är prioritet och andra värdet är datan som ska omhändertas. Tanken var att göra om databiten till en tuple, där första värdet var TTL-variabeln och andra värdet var datan. Det hade med denna struktur blivit en nästlad tuple. Ett problem är dock att tuples i Python inte är muterbara. Detta betyder att originalvärdet i en tuple inte kan ändras och det finns därför inget sätt att minska TTL-variabeln. Lösningen på detta var att byta ut den senare tuplen mot en lista. Prioritetskön hade då tagit in en tuple som i sin tur hade haft en lista i sig. Denna lösningen fungerade, men ytterligare problem uppstod då prioritetsköer inte var itererbara. Det var alltså inte möjligt att minska TTL-variabeln varje gång data hämtades från kön.

Den sista och fungerande lösningen blev att istället behålla originalidén om att ha en nästlad tuple men att endast undersöka TTL-variabeln när objektet hämtades ut. Om TTL-variabeln inte är relevant när objektet hämtas slängs objektet. Kön kan i detta fall fortfarande fyllas upp så det sattes en övre gräns på 200 värden i kön. Om antal kö-objekt överskrider 200 klonas kön, varje objekt uppdaterar sin TTL-variabel och gamla värden rensas.

4.6 Kommunikationsflöde

Följande är de steg ett fordon behöver ta, i form av kommunikation med server, för att kunna kommunicera med andra fordon.

När ett fordon startas skickas ett initieringsmeddelande till servern, krypterat med serverns publika nyckel, se figur 4.3. Denna nyckel finns på fordonet från produktion. För att skicka initieringsmeddelandet krävs vetskapen om hur servern kontaktas i form av IP eller domännamn.

Figur 4.3: Fordonet skickar ett initieringsmeddelande som är krypterat med serverns publika nyckel.

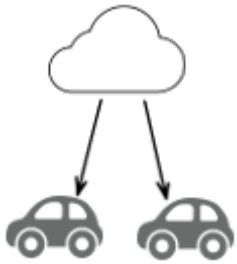
Servern dekrypterar meddelandet med sin privata nyckel och tolkar det. Meddelandet ska bestå av fordonets identifikation, dess publika nyckel samt annan intressant information som servern behöver känna till såsom IP eller positionering. Misslyckas servern med att läsa av meddelandet beror det antingen på att det inte var korrekt krypterat, vilket i så fall kommer

ignoreras, eller att datan inte var korrekt. Tredje fallet är om din unika identifikation inte finns med som ett registrerat fordon.

Om dessa tre kontroller passerar kommer servern se fordonet som aktiv och lägga till den i en lista med andra aktiva fordon. Därefter skickas ett svar till fordonet som är krypterat med den publika nyckel servern tog emot. I detta meddelande finns en symmetrisk nyckel, se figur 4.4, som används för kommunikation mellan fordon samt för uppdateringar från servern.

Figur 4.4: Servern skickar ett svarsmeddelande som är krypterat med den publika nyckel som fordonet skickade med i initieringsmeddelandet.

Servern kommer sedan starta en uppdateringsfas där alla fordon, inklusive det nya, får den uppdaterade listan, se figur 4.5. Denna lista är krypterad med den symmetriska nyckeln.



Figur 4.5: När en förändring sker bland de uppkopplade fordonen skickar servern ut ett uppdateringsmeddelande med den nya informationen. Detta meddelande är krypterat med den symmetriska nyckeln som används.

När alla fordon tagit emot listan börjar de kommunicera med varandra, se figur 4.6. Varje fordon startar en REQ-tråd för alla nya element i den uppdaterade listan.

Figur 4.6: Fordon kan skicka REQ/REP -meddelande mellan varandra.

Den symmetriska nyckeln uppdateras med jämna mellanrum. Varje bil som är aktiv måste hålla koll på hur länge den symmetriska nyckeln gäller. När det återstår 5 sekunder, en tid som anses tillräckligt långt att skicka och få svar, ber den servern om den nya nyckeln, se

figur 4.7. När den gamla nyckeln går ut byts den mot den nya. Detta gör att alla nycklar byts samtidigt oberoende av latens.

Figur 4.7: Varje fordon frågar servern efter en ny symmetrisk nyckel när den nuvarande nyckelns TTL är påväg att ta slut.

5. Resultat

Detta kapitel beskriver det slutgiltiga resultatet som kommit av examensarbetet. En mjukvarulösning har tagits fram där de flesta mål i kravspecifikationen har uppnåtts.

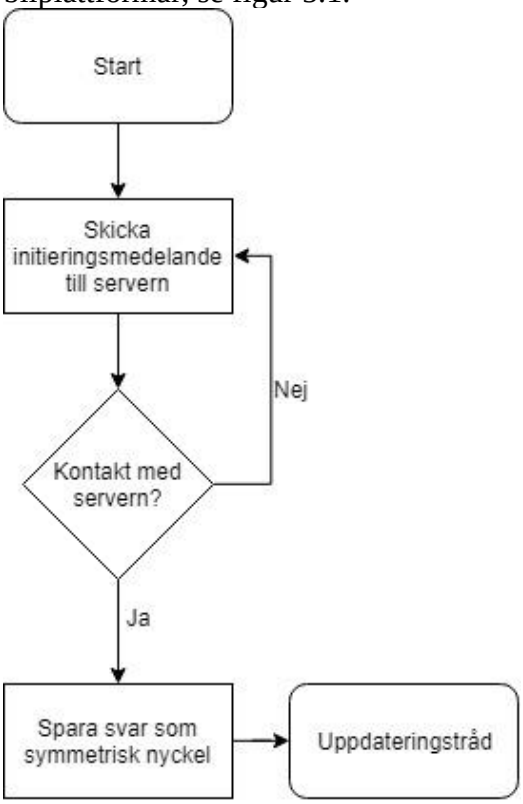
5.1 Bilplattformen

Sedan examensarbetet började har bilplattformen utrustats med ytterligare sensorer för att underlätta arbetet. Plattformens mjukvara har uppdaterats och kompletterats med examensarbetets kod. Plattformen har möjlighet att läsa av kommandon via en buffer, avgöra om de är viktiga och skicka dem vidare. Skulle meddelandet inte komma fram till mottagaren kommer det göras försök att skicka om meddelandet enligt The Lazy Pirate Pattern.

Bilplattformen har möjlighet att ta emot kommandon från andra fordon. När den tar emot kommandon tolkas meddelandet och det tilldelas en prioritet. Prioriteten bestämmer hur viktigt meddelandet är i förhållande till andra meddelanden som väntar på att skickas ut på CAN-bussen. Biplattformen har även möjlighet att, om det är ett hastighetskommando som kommer in, avgöra distans till framförvarande objekt och ändra värdet på det inskickade kommandot.

5.1.1 Bilapplikation

Med bilapplikation menas den mjukvarulösning för bilkommunikation som körs på plattformarna. Bilapplikationen har tre uppgifter. Skicka data, ta emot data och analysera data. Varje bilplattform måste ta kontakt med servern innan den kan skicka data till andra bilplattformar, se figur 5.1.



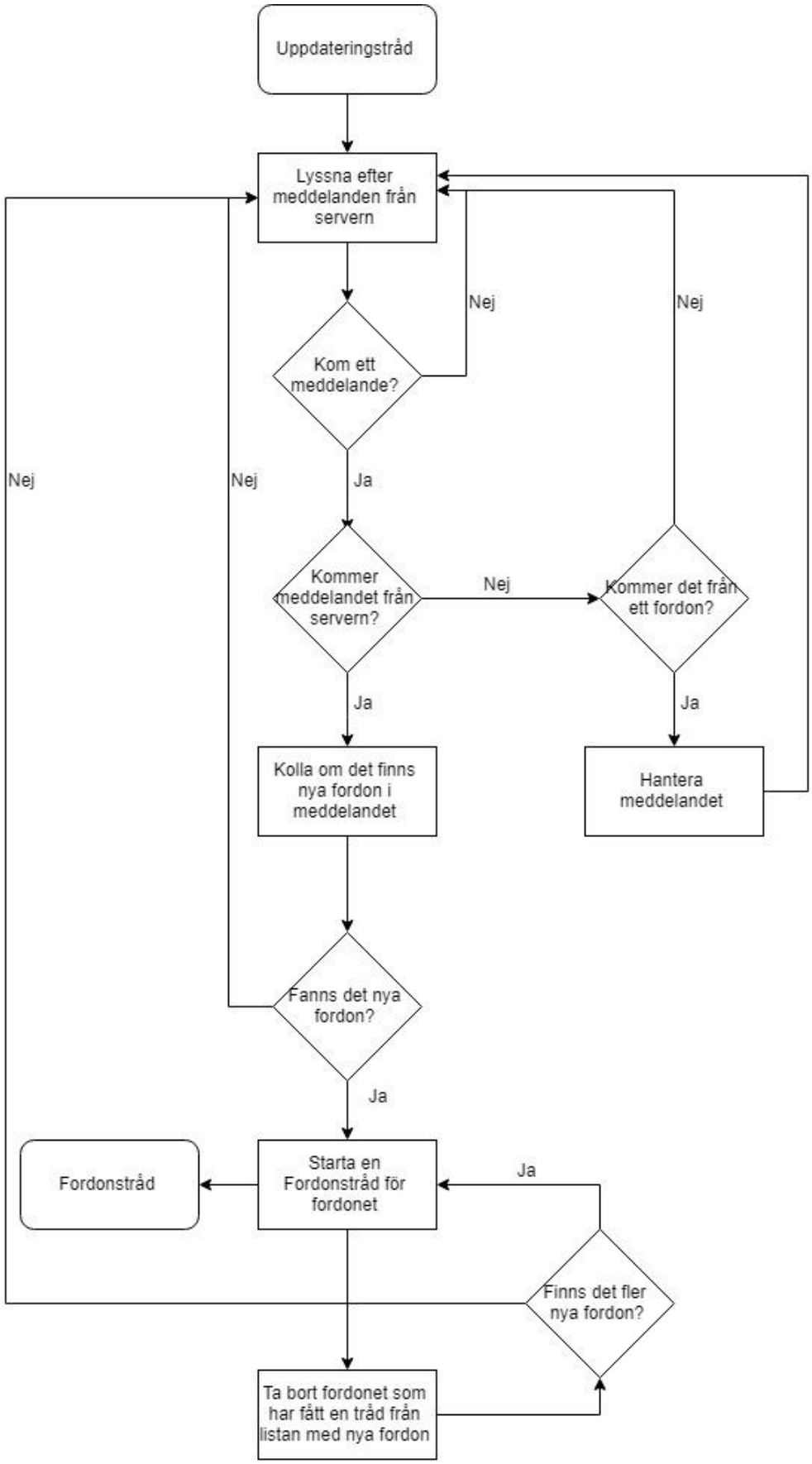
Figur 5.1: Beskriver det som händer vid start av applikation på bilplattformen.

```
def contact_server(self):
    return_value = send_message_to_server()
    if return_value = False:
        contact_server()
    else:
        aes = create_aes_object(return_value)
        starting_rep_thread()
        starting_aes_thread()
```

Figur 5.2: Förenklad kod för hur ett fordon tar kontakt med servern. Om fordonet inte får kontakt kallas metoden rekursivt tills kontakt uppstått.

Som det går att se i figur 5.2 skapas, vid svar från servern, ett AES-objekt för kryptering av meddelande samt två olika trådar, en AES-tråd och en svarstråd.

AES-tråden ser till att den nuvarande AES-nyckelns tidstämpel aldrig går ut. Detta görs genom att, 5 sekunder innan nyckeln går ut, frågar den servern efter en ny nyckel. Detta garanterar att den symmetriska nyckeln byts på alla plattformar samtidigt.



Figur 5.3: En REP-tråd som lyssnar på alla meddelanden som kommer till ett fordon. Antingen kommer det från servern och då kommer listan med bilar att uppdateras eller så kommer det från ett av det fordon det redan kommuniceras med och då hanteras och tolkas datan på lämpligt sätt.

I svarstråden, se figur 5.3**,** lyssnas det efter uppdateringsmeddelanden från servern. Uppdateringsmeddelandet består av information gällande alla fordon som det bör kommuniceras med. Det ett fordon behöver veta för att börja kommunicera med ett annat fordon är dess position och IP-adress, se figur 5.4.

POS	POS	IP	IP	IP	IP

Figur 5.4: Varje ruta representar en byte. Första och andra byten är X och Y -possition. Resterande fyra bytes representerar en IP-adress till ett annat fordon.

Varje bilplattform, se figur 5.3**,** lyssnar även på meddelanden från andra fordon som den ska kommunicera med i en REP-tråd. Beroende på vilken typ av meddelande det är görs olika saker. Exempelvis vid ett hastighetskommando hämtas data från den framåtriktade sensorn och hastigheten regleras baserat på avståndet till framförvarande objekt. Meddelandet görs om och skickas ut på CAN-nätverket.

För varje fordon som tas emot i svarstråden skapas en egen tråd för kommunikation med det fordonet, se figur 5.5. Varje tråd läser av en delad variabel som representerar det senaste kommandot som skickats på CAN-nätverket. Denna data tolkas och klassas det som relevant information för andra fordon att veta om skickas datan till alla fordon som ska kommuniceras med, se figur 5.6.

```
def update_carlist(self, verified_cars):
    oldlist = carlist
    carlist = verified_cars

    for car in carlist:
        ip = car[2:]
        tup = (ip, str(self.port))
        if car not in oldlist:
                start_new_car_thread()
```

Figur 5.5: Förenklad kod som visar hur en REQ-tråd skapas för varje fordon en bilplattform ska kommunicera med.

Figur 5.6: En REQ-tråd för varje fordon som det kommuniceras med.

5.1.2 Bytearray

Alla meddelanden som skickas mellan bilplattformar är en array av bytes. Med hjälp av att alla bytes har en fördefinierad betydelse kan man skapa komplexa meddelanden med relativt liten informationsstorlek, se tabell 5.7.

0x02	0x10	0x1c	0x20

Figur 5.7:Exempel på hur ett meddelande kan se ut. Varje ruta motsvarar en byte. Försa byten representerar vilken typ av meddelande det är. Andra byten bestämmer vad som ska hända och resterande bytes är värdet som ska appliceras.

Detta skulle, på mottagarbilen, tolkas som ett kommando där man vill ändra parametrar (0x02), se figur 5.8. Parametern man vill ändra är hastigheten (0x10), se figur 5.9, och hastigheten man vill sätta är 450 (0x1c/0x20).

```
CMD_SET_PARAMS = 0x02
SET_MOT_THR = 0x01
DO_CALIB_GYRO = 0x02
ARM_MOTORS = 0x03
DISARM_MOTORS = 0x04
```

Figur 5.8: Exempel på några av de command.bytes som finns tillgängliga. Dessa är generellt sätt första byten i arrayen.

```
CMD_SPEED = 0x10
WHEEL_SPD = 0x01
CAR_SPD = 0x02
TURN_SPD = 0x03
```

Figur 5.9: Exempel på några av de command bytes som finns tillgängliga. Dessa är generellt sätt andra byten i arrayen.

5.2 Server

Servern har tre uppgifter; autentisera bilar, se till att en uppdaterad lista med bilar finns på varje fordon och uppdatera den symmetriska nyckeln som används för kommunikation mellan bilar. På förfrågan kan den också skicka den symmetriska nyckeln till alla som vill ha den. För detta krävs att bilen är autentiserad och finns med bland aktiva bilar.

5.2.1 Mottagartråd

Serverns reply-socket är alltid aktiv och väntar på att fordon ska skicka förfrågningar. Den kan hantera tre olika typer av meddelanden. Vilken typ av meddelande det är representeras av den första byten, se figur 5.10.

```
if msg[0] == 0x01:
    #Do stuff
elif msg[0] == 0x02:
    #Do stuff
elif msg[0] == 0x03:
    #Do stuff
else:
    #Unknown message
```

Figur 5.10: Visar olika typer av meddelanden som servern kan ta emot.

Alla typer av meddelanden till servern kräver 11 bytes. Alla resterande bytes är typspecifika bytes.

TYPE	PID	PID	PID	PID	POS	POS	IP	IP	IP	IP

- TYPE består av en byte som bestämmer vilken typ av meddelande det är (1, 2 eller 3).
- PID är 4 bytes lång vilket medför ett värde mellan 0 och 4 294 967 295.
- POS är 2 bytes där den första är x-positionen och den andra är y-positionen. Detta ger en grid på 255 x 255 med meter precision. Befinner sig ett fordon på ett större avstånd är det inte intressant att kommunicera med.
- IP består av fyra bytes.

Första typen av meddelande, 0x01, är ett initieringsmeddelande. För att ett fordon ska vara med i nätverket krävs att den blir godkänd av servern och läggs till bland de aktiva fordonen. Utöver de 10 bytes som representerar fordonet måste den publika RSA nyckeln skickas med.

För meddelande nummer två, 0x02, hanteras uppdateringsmeddelande. Detta meddelande är uppbyggt på samma sätt som initeringsmeddelandet exklusive RSA nyckeln. Servern kontrollerar att detta fordon finns med bland aktiva bilar och om så är fallet uppdateras informationen om just den bilen. Exempelvis vid ny position.

Tredje meddelandet, 0x03, består av 10 bytes som representerar ett fordon. I detta meddelande frågar ett fordon efter en ny, uppdaterad symmetrisk nyckel som servern returnerar om bilen är inlagd bland aktiva bilar

5.2.2 Avsändartråd

Servern skickar REQ-meddelanden när listan av aktiva bilar har ändrats. Detta kan ske vid tre tillfällen. Antingen när en ny bil läggs till som inte var med tidigare (reply typ 1). Det andra tillfället är när en bil uppdaterar sin information (reply typ 2). Det tredje tillfället är när servern tar bort en bil från listan. Detta sker när antingen en bil har varit inaktiv för länge eller när det inte går att kontakta fordonet.

När något av detta sker anropas metoden update_active_cars vars uppgift är att se till att alla fordon får den uppdaterade versionen av aktiva bilar, se figur 5.11. Om fordonet har varit aktiv, det vill säga att den skickat ett meddelande inom de senaste 30 sekundrarna, skapas en REQ-tråd med fordonets IP på port 5561. Uppdateringstråden krypterar listan med bilar med den aktiva symmetriska nyckeln och försöker skicka meddelandet. Om den misslyckas tre gånger anses fordonet ha försvunnit och tas bort från listan med aktiva fordon.

```
def update_active_cars(self):
    port = 5561
    for key in self.activeCars:
        if reqHandler.check_last_active(self, key) is True:
            self.update_active_cars(i)
            break
        else:
            socket = cr.init_req(self, (self.activeCars[key], str(port))
                if reqHandler.check_within_range(self, self.activeCars[key]):
                    threading.Thread(target=self.update_thread, args = [socket, tup]).start()
```

Figur 5.11: Kodbit som beskriver hur servern skickar meddelande till fordon.

5.2.3 Uppdatering av symmetrisk nyckel

Den symmetriska nyckeln finns och kontrolleras på servern. Det finns en aktiv symmetrisk nyckel som har en TTL. När denna tiden går ut byts den nuvarande nyckeln ut mot en ny. Den nya nyckeln generas i förväg för att fordon ska kunna hämta den innan nyckelns TTL tar slut. På så sätt går det att garantera att alla bilar byter nyckeln samtidigt oavsett latens och vart de befinner sig. För att åstadkomma detta används två trådar. Se figurerna 5.12 och 5.13.

```
def change_aes_key(self):
    while True:
        self.aes = self.new_aes
        time.sleep(self.aes.ttl - int(time.time()))
```

Figur 5.12: Denna tråd ser till att nyckeln byts ut när tiden är slut.

```
def update_aes_key(self):
    while True:
        if self.new_aes.ttl - int(time.time()) <= 5:
            self.new_symmetric_key()
        time.sleep(0.2)
```

Figur 5.13: Denna tråd kontrollerar med jämna mellanrum hur lång livsläng den symmetriska nyckeln har kvar. Om den är mindre än fem sekunder skapas en ny symmetrisk nyckel.

5.3 Demonstration

Examensarbetet har resulterat i en prototypapplikation för att visa att bilplattformarna kan kommunicera med varandra på ett säkert sätt. Demonstrationen börjar med att en av plattformarna startas. Den skickar en signal till en bakomvarande bilplattform som tolkar kommandot och kör efter. Den mottagande plattformen får in sensordata och ändrar hastigheten beroende på avstånd till framförvarande plattform. När den främre plattformen stannar, stannar den bakomvarande också och gör en mjuk inbromsning.

6. Slutsats

Baserat på resultatet kan det konstateras att målet och syftet är uppnått. Slutresultatet är ett program som förmedlar information automatiskt mellan fordon samtidigt som det tar hänsyn till datasäkerhet och verifiering. Det har gjorts försök till att utvärdera teknologin men tyvärr har det inte funnits tillräckligt med tid för att göra detta mer utförligt. Det som kan konstateras är att kommunikationen fungerar acceptabelt men att bilplattformarna är för instabila.

6.1 Kravspec

Av den givna kravspecifikationen är sex av de sju givna kraven uppfyllda. Det krav som inte är uppnåtts är kravet om multicast. Ett fordon kan idag inte skicka meddelanden till fler än ett fordon åt gången. Det finns mjukvarustöd implementerat för att tillåta detta men det uppstår kritiska fel när man försöker. Felen beror dock inte på den nya mjukvaran utan på en känd bugg i ZeroMQ. Det finns uppdaterade versioner av ZeroMQ som sägs lösa problemet men det var inte helt trivialt att uppdatera då det inte är den senaste stabila versionen. På grund av detta bortprioriterades kravet och fokus blev att färdigställa resterande krav.

6.2 Vidare utveckling

Här preseneteras idéer för vidare utveckling som kommit fram under examensarbetet.

- I dagsläget utvärderar mottagarbilen datan den får in och har möjlighet att ändra värden på parametrar baserat på avstånd till framförvarande objekt. Detta görs med ultraljud och är tämligen ostabilt. För att kunna förbättra beslutsfattandet hos bilarna är de rimligt att veta var andra fordon befinner sig i förhållande till den. Med hjälp av GPS och bilar i större skala kan man förbättra bilarnas beslutsfattande markant.
- Servern kan i framtiden användas för mer än vad den gör idag. Dagens användning är att underlätta kryptering samt verifiering. En utvecklingsidé, förutsatt att GPS fungerar, är att använda molnet för att kommunicera saker som halt väglag till fordon först när de kommer inom farozonen.
- Fordon kan kommunicera med andra fordon och server. En utvecklingsidé är att låta dem kommunicera med infrastruktur såsom stoppljus eller parkeringshus. En tanke hade då varit att rödljus kommunicerade med fordon när de slår om och på så sätt ge bilen ytterligare information för att göra bränslesnålare beslut. De hade även kunnat få information om hur många platser det finns i ett parkeringshus direkt i bilen och var optimala parkeringsplatsen finns.
- Förutsatt att fordon är helt autonoma hade en utveckling varit att integrera information från andra fordon in i befintligt autonom kod för att göra mer intelligenta beslut.

7. Diskussion

I det här kapitlet kommer det diskuteras vilka brister och framtida utvecklingsmöjligheter som finns.

7.1 Kritisk diskussion

Under genomförandet av detta examensarbete har mycket gått bra men det finns saker som kunde gjorts annorlunda.

Det största problemet visade sig vara positionering vilket tas upp i flera delar av rapporten. Trots att snabbt blev klart att positionering inte skulle kunna vara applicerbart lades mycket tid på att hitta en alternativ lösning. Detta medförde att andra delar av projektet fick mindre fokus än det förtjänade. Vad som borde gjorts var att släppa positionering och fokusera mer på andra delar av projektet.

En annan del som medförde mycket frustration för gruppmedlemmarna blev nya efterfrågningar från Infotivs sida under examensarbetets gång. Exmepelvis fanns inte kryptering med i den initiala projektbeskrivningen men visade sig bli en stor del av examensarbetet. Optimalt bör en tydlig kravspecifikation tagits fram innan projektets början. Som lösning på problemet hade man kunnat säga nej till de ytterligare efterfrågningarna då det inte var en del av det projektet vi valde att utföra.

7.2 Device to Device Communication

I det här examensarbetet har en serverlösning använts för att verifiera fordon som befinner sig i ens närhet. På så sätt ska bilar kunna kommunicera med varandra. Med dagens teknologi finns vissa begränsningar på andra alternativ. Device to Device (D2D) kommunikation tros vara en av de stora utvecklingsområdena inom kommande 5G nätverk [20] . D2D innebär att en enhet kan hitta och kommunicera med andra enheter i sin närhet utan en mellanhand, som exempelvis en telefonmast.

Det undersöktes ett liknande alternativ tidigt i examensarbetet men beslutet togs att det hade varit för komplicerat att implementera. Däremot hade det varit en intressant lösning då molnlösningen, även om den fungerar, resulterar i stora mängder data som måste kommuniceras och över större sträckor vilket ökar latensen för hur snabbt ett meddelande når slutdestination. Ett stort problem med D2D är huruvida det går att garantera att alla enheter i närheten har hittats. Fördelen med en molnlösning är att alla enheter är uppkopplade vilket gör att det inte ska kunna hända.

Hur lång räckvidd D2D skulle komma att ha är också intressant. D2D fungerar så att varje enhet har liknande funktionalitet som en router. Exempelvis kan en enhet fungera som ett hopp för två andra enheter vilket möjliggör kommunikation över långa sträckor trots att de två enheterna som kommunicerar med varandra är inom räckvidd. Detta är dock endast möjligt om det finns enheter i närheten. Ute på landsväg där inga andra fordon finns i närheten kommer bilar bara kunna kommunicera med varandra när de är inom räckvidd av varandra vilket ställer krav på teknologin om den ska vara användbar i trafiken. Vi tror dock att D2D kommer spela en stor roll inom bilkommunikation i framtiden trots de svårigheter som fortfarande måste lösas.

7.3 Positionering och riktning

Som det har nämnts tidigare i rapporten fanns det förhoppningar om att positionering skulle vara möjligt. En av de frågeställningar som försökte besvaras var: Vem ska kommunicera med vem? Det är svårt att avgöra vem ett fordon behöver kommunicera med om det inte finns en uppfattning om vart den befinner sig. En bil i Stockholm är inte intresserad av att veta bilen i Göteborgs sensordata. Detta är dock inte möjligt i nuvarande lösning. Dessutom är det viktigt att veta om det är ett fordon man möter eller kör efter. Därför hade även riktning på bilen, exempelvis med hjälp av kompass, varit intressant data att jobba med. En bil bör inte hantera ett fordon likadant beroende på om de kör i samma riktning eller är på väg mot varandra. Tyvärr har inte detta varit möjligt vilket har tvingat oss att jobba runt det problemet.

7.4 Datatrafiksavläsning

Vid frågan om hur data ska krypteras insåg vi att det fanns en del problem som måste övervägas. Ett av de stora problemen inom den tekniska utvecklingen är hur privat information, framförallt inom sociala medier, hålls gömd från tredje parter. Nyligen har Facebook anklagats för sälja personlig information till företag för att de ska kunna rikta reklam till rätt målgrupp.

Liknande information om människors privatliv kan avläsas om deras fordon skickar positionering till en molntjänst. Vart de befinner sig när på dagen, arbetsplats etc. är exempel på information som teoretiskt kan utläsas från den lösning detta examensarbete har tagit fram. Detta är en av anledningarna till att stor vikt lades på datasäkerhet och kryptering.

7.5 Nackdelar med ZeroMQ

Valet av ZeroMQ som kommunkationsbibliotek gjordes åt oss av Infotiv då tidigare utveckling hade använt sig av detta. Nu i efterhand går det att ifrågasätta huruvida är ett bra alternativ för bil-till-bil kommunikation. ZeroMQ är byggt ovanpå TCP protokollet vilket garantera leverans av meddelande men är betydligt långsammare än UDP protokollet.

Vid bilkommunikation finns två viktiga aspekter vid själva leveransen av meddelandet. Det ska gå snabbt och det måste komma fram. ZeroMQ garanterar inte något av dem ursprungligen. Det finns sätt att garantera att ett meddelande levereras men det är inte inbyggt i grunden. Liknande funktionalitet hade kunnat byggas på ett UDP protokoll och det hade dessutom varit snabbare. Tyvärr insåg vi detta för sent i utvecklingen vilket innebär att ZeroMQ fortfarande används.

7.6 Etik

När frågan om etik i samband med bilar kommer upp finns det inga lätta svar. Vem är ansvarig om en bil, vars förare inte har kontroll, kör över ett oskyldigt barn? Vad prioriteras om valet står mellan att rädda föraren i bilen eller en flerbarnsfamilj? Om resultatet av detta examensarbete skulle användas i ett verkligt scenario och det hade skett en miss i kommunikationen, hade programmerarna då varit ansvariga, eller föraren, som valde att släppa kontrollen? Dessa är frågor som är värda att överväga när man utvecklar mjukvara som någon dag kan påverka vem som får leva. Även om det så kallade svaret inte har någon påverkan på examensarbetet är det fortfarande någonting som har funnits i åtanke.

7.7 Miljö

Enligt naturvårdsverket står vägtransporter för cirka 30 procent av Sveriges koldioxidutsläpp [21] . Detta beror till stor del på dålig hantering av bränsle och ofördelaktiga förbränningsmetoder. Den mänskliga faktorn spelar dock också roll. Det finns sätt som kan hjälpa föraren minska sina utsläpp med 10 till 20 procent [22]. Detta sker genom att exempelvis hålla jämn hastighet, köra på rätt växel och göra hastiga accelerationer. I autonoma fordon kan detta ske automatiskt. Dock är teknologin inte perfekt. Om fordon hade kunnat kommunicera med varandra och en bil kunnat berätta för andra fordon att de skulle bromsa långt innan en sensor hade fått reda på det, kan en mjukare motorbromsning göra. Om man ser utanför bil-till-bilkommunikation och tänker sig att fordon kunde kommunicera med infrastruktur hade samma princip kunnat appliceras vid stoppljus och korsningar.

Referenser

[1] S. Forward.

[2] ”Wikipedia,” 21 April 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Transmission_Control_Protocol. [Använd April 2018].

[3] ”Wikipedia,” 27 August 2017. [Online]. Available: https://sv.wikipedia.org/wiki/User_Datagram_Protocol. [Använd April 2018].

[4] ”Wikipedia,” 2018 April 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Domännamnssystemet. [Använd April 2018].

[5] ”Wikipedia,” 18 May 2018. [Online]. Available: https://en.wikipedia.org/wiki/Multicast_DNS. [Använd April 2018].

[6] M. Guardigli, ”Quora,” 28 February 2018. [Online]. Available: https://www.quora.com/What-is-the-difference-between-latency-and-throughput. [Använd May 2018].

[7] ”Wikipedia,” 30 December 2016. [Online]. Available: https://sv.wikipedia.org/wiki/Radiofyr. [Använd April 2018].

[8] ”Wikipedia,” 17 April 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Global_Positioning_System. [Använd April 2018].

[9] ”gps-navigation,” [Online]. Available: http://www.gps-navigation.se/. [Använd April 2018].

[10] ”Illustrerad Vetenskap,” Illustrered Vetenskap, 30 January 2018. [Online]. Available: http://illvet.se/teknologi/hur-fungerar-gps-systemet. [Använd April 2018].

[11] ”Wikipedia,” 16 November 2017. [Online]. Available: https://sv.wikipedia.org/wiki/Ultraljud. [Använd April 2018].

[12] ”Wikipedia,” 18 May 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Kryptering. [Använd April 2018].

[13] P. Hintjens, ”zguide,” [Online]. Available: http://zguide.zeromq.org/page:all#toc0. [Använd Mars 2018].

[14] P. Hintjens, ”zguide,” [Online]. Available: http://zguide.zeromq.org/php:chapter4. [Använd April 2018].

[15] ”Wikipedia,” 24 Mars 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Controller_Area_Network. [Använd May 2018].

[16] ”Wikipedia,” 26 April 2018. [Online]. Available: https://sv.wikipedia.org/wiki/Python_(programspråk). [Använd April 2018].

[17] ”dlitz,” 24 May 2012. [Online]. Available: https://www.dlitz.net/software/pycrypto/api/2.6/. [Använd May 2018].

[18] ”Wikipedia,” 17 Mars 2015. [Online]. Available: https://sv.wikipedia.org/wiki/SCP. [Använd May 2018].

[19] ”Wikipedia,” 15 November 2017. [Online]. Available: https://sv.wikipedia.org/wiki/Secure_Shell. [Använd May 2018].

[20] X. Shen, ”Device-to-device communication in 5G cellular networks,” *IEEE Network,* vol. 29, nr 2, pp. 2 - 3, 24 March 2015.

[21] ”Naturvårdsverket,” 4 May 2017. [Online]. Available: https://www.naturvardsverket.se/Miljoarbete-i-samhallet/Miljoarbete-i-Sverige/Uppdelat-efter-omrade/Transporter-och-trafik/Vagtrafik/Vagtrafikens-miljopaverkan/ . [Använd April 2018].

[22] ”Wikipedia,” 17 September 2017. [Online]. Available: https://sv.wikipedia.org/wiki/Sparsam_k%C3%B6rning. [Använd April 2018].