Intro til CANbus Hacking

Teoriafsnit af Adis Begic & Benjamin Fester Henningsen

Introduktion

For at kunne oprette et system hvori der skal samles data fra bestemte komponenter i bilen, skal der indgå visse enheder. Dette kan gøres ved at benytte protokoller som CANbus, samt OBD-II moduler eller scannere, der kan bruges til dette formål. Følgende afsnit omhandler teknikken bag kommunikationen mellem disse enheder.

Bus-systemer

Et bus-system er en måde at overføre mere data på en ledning end bare en enkelt bit af information per ledning[1]. En bit information kan eksempelvis tænde og slukke et bestemt lys i bilen ved at henholdsvis sende 5V igennem eller 0V gennem ledningen. Biler indeholder en masse enheder med hver deres chip der er forbundet af en masse ledninger. Formålet med at indføre et bus-system er at mindske antallet af ledninger og hermed både kompleksiteten og prisen. I bilverden er der typisk tale om CAN protokollen til kommunikation mellem enheder for at opnå dette.

CAN

CAN (Control Area Network) bus protokollen blev udviklet i midten af 1980 hovedsageligt til bilindustrien[2] af BOSCH. Den tillader kommunikation mellem microcontrollers og enheder uden en host computer. Teknikken er pålidelig og billig og bliver også benyttet i søværnet, militæret, elevatorer, proteser, læge tekniske enheder og flere. CAN findes som CAN 2.0 A (standard) og 2.0 B (forlænget), hvor BOSCH garanterer for følgende egenskaber ved brug af CAN protokollen[3]:

---

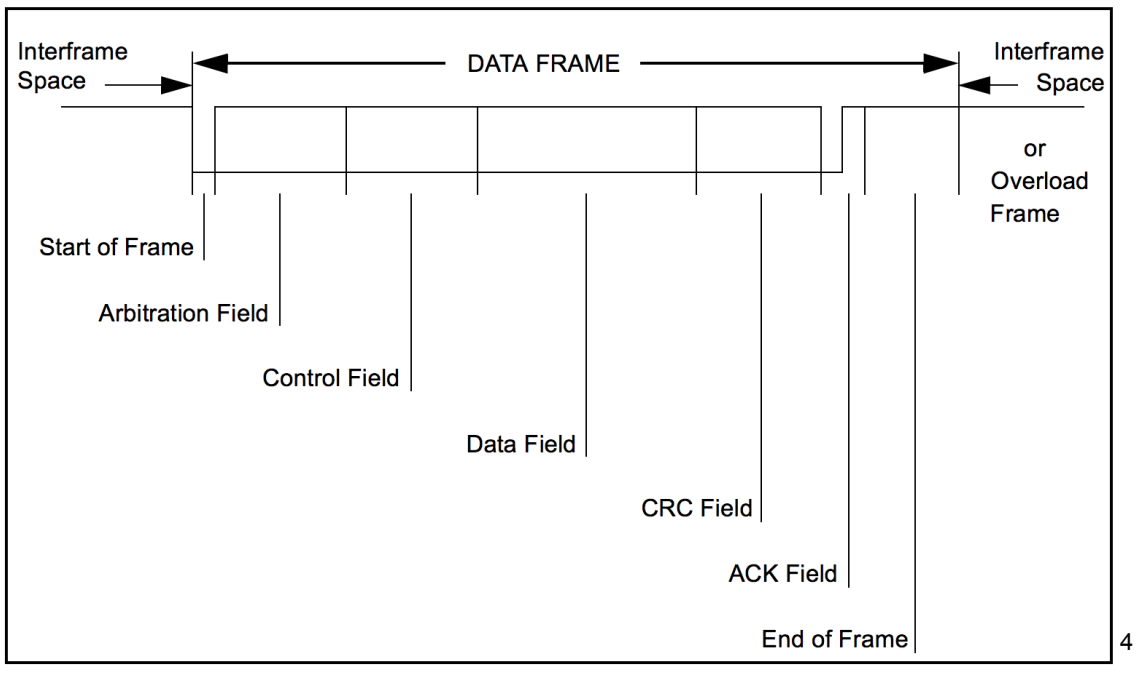
[1] https://mechanics.stackexchange.com/questions/25561/difference-between-obdii-and-can
[2] http://www.testandmeasurementtips.com/exploring-canbus-oscilloscope/
[3] *CAN specification 2.0*, sep 1991, BOSCH

- Prioritering af beskeder
- Garanti af latenstid.
- Konfigurationfleksibilitet.
- Multicast modtagning med tidssynkronisering
- System data konsisten
- Multimaster
- Fejlcheck og signalering
- Automatisk gentransmission af fejlbeskeder når bussen bliver ledig.
- Forskelsgenkendelse mellem midlertidige fejl og permanente fejl og automatisk slukning af defekte noder.

En message transfer er kontrolleret af fire forskellige frame typer:

- Data frame som leverer data
- Remote frame som requester data fra data frame med samme identifier
- Error frame til at opfange fejlbeskeder
- En Overload frame til at give delay mellem Data og Remote frames. Data og remote frames bliver separeret af en interframe space

Data frame for CANbus protokollen ser således ud:



4

Den første **Start of Frame** betegner start of frame transmissionen . **Arbitration field**, 11 bits, er det felt der indikerer vigtigheden af beskeder fra de forskellige devices. Der menes her at hvis eksempelvis lys og motor går ud samtidigt, så er det beskeden fra motoren der bliver prioriteret højere end beskeden fra lyset. Arbitration field kaldes også *identifier*, og indikerer altså hvilke enheder i bilen der prioriteres højest. **Control field** består af 6 bits, Data Length Code(4 bits) og 2 bits til

[4] *CAN specification 2.0*, sep 1991, BOSCH

tilpasselige egenskaber. Data Length Code bestemmer antallet af bytes der skal være i **Data field**. Efterfølgende er **CRC Field**, feltet der checker for fejl. Til slut er **ACK Field** og **End of Frame** disse skal være recessive bits da data frames og remote frames skal være afgrænset af nuller.
Inde i disse data frames indeholder der informationer vedrørende de forskellige enheder installeret i bilen. Denne information vil vi i vores projekt benytte ved brug af et OBD-II modul. Et modul der blandt andet understøtter CAN protokollen.

OBD-II

OBD-II (On-board diagnostics) er en protokol på et højere niveau der bliver brugt som diagnoseværktøj. OBD-II kan bruge flere forskellige bus systemer, heriblandt CAN bussen. OBD-II modulet bliver indsat i OBD porten i bilen, typisk omkring styret og kobles til den ønskede enhed. OBD-II moduler opererer typisk via Wi-Fi eller Bluetooth til ens mobile enheder. OBD-II modulet er altså i vores tilfælde mellemleddet mellem bil og telefon. Den oversætter data information fra bilens enheder til et sprog der kan fortolkes. Denne transmission sker på vores OBD modul via Bluetooth.

Opsætning i Android

Det er en forudsætning at man har opsat Bluetooth iht. Bluetooth ‘aktivering’ samt opdagelse og parring af enheder, inden man begynder at arbejde med hvad dette teoriafsnit har at tilbyde. Derudover skal du selv kunne oprette variabler, objekter o.l. da vi hovedsageligt vil fokusere på anvendelige metoder. Det anbefales at man kigger på tidligere teoriafsnit udarbejdet af forrige elever, eller eventuelt kigger i projektkoden tilhørende dette projekt, for hjælp til ovenstående.

For at kunne begynder at arbejde med data fra bilen, kræver det at vi laver en BluetoothSocket. Dette gøres for at kunne igangsætte en forbindelse som er afgørende for at kunne starte en dataudveksling.

Et UUID skal defineres før man laver en BluetoothSocket eftersom enheder bruger SDP (Service Discovery Protocols) til at finde ud af hvilke services (UUID’s) der er understøttet på den tilgående enhed.

Nedenstående er angivet det UUID som bliver brugt når man arbejder med serial port grænseflader. Som det ses, er SPP (Serial Port Profile) i dette tilfælde 0x1101.

Sidstnævnte tal definere en Bluetooth profil som tillader kommunikation imellem en Bluetooth enhed og en host/slave enhed.
Dette tal variere alt efter hvad man har med at gøre. Det vigtigste ved brug af UUID er at begge enheder man ønsker forbundet, bruger det samme UUID.

```
UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
```

For at lave en BluetoothSocket til at forbinde til et kendt device bruges nedenstående linje. I parentesen angiver man identifikationen på sit UUID.

```
 .createInsecureRfcommSocketToServiceRecord()
```

Herefter kan man forsøge at tilslutte en forbindelse ved at kalde nedenstående.

```
.connect()
```

Hvis man har success med at tilslutte forbindelse, kan man nu åbne for input/output kanalerne ved at kalde nedenstående.

```
.getInputStream()
.getOutputStream()
```

Du har hermed nu hentet objekterne der står for at data udveksle mellem enheder.

Nedenstående ses et eksempel på en metode til opsættelse.

```
public void connectToDevice(BluetoothDevice device)
{

   //Kald for stopning af skan for at sikre at der ikke er noget der skanner længere da vi allerede har valgt en enhed.
   stopScan();

   // Hvis der allerede er et forsøg der prøver at forbinde, så stop nuværende handling, og lad forsøget gøres færdigt først.
   if (connecting)
   {
      return;
   }

   connecting = true;
```

```
try
{
    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");


    //Prøv at forbinde til Bluetooth enheden.
    currentSocket = device.createInsecureRfcommSocketToServiceRecord(uuid);


    //Hvis det lykkedes, så lad enheden være nuværende enhed.
    currentDevice = device;

    //Åben op for socket forbindelsen til enheden.
    currentSocket.connect();

    //Modtag input stream fra socket til at lytte for besked.
    inputStream = currentSocket.getInputStream();
    //Modtag output stream fra socket til at sende besked.
    outputStream = currentSocket.getOutputStream();

    //Gem nuværende enhed for at gøre det lettere i fremtiden at forbinde igennem
automatisk forbindelse.
        app.getSessionManager().saveODB(currentDevice.getAddress());

        // Forbindelsen lykkedes, start med at lytte efter data.
        startListening();
        // Spørg om data først.
        askForData();
        // Planlæg næste spørgsel med en forsinkelse på 0.5 sekunder.
        reAskDataIfGotNoData();
    } catch (IOException e)
    {
        // Notificer aktivitet hvis noget gik galt.
        EventBus.getDefault().post(e.getMessage());
    }
    connecting = false;
}
```

Data manipulation i Android

Gennem data manipulation kan du skrive tilbage til OBD'en og på den måde, i princippet, forme datastrømmen som ønsket. Grunden til at der skrives at du kan skrive tilbage til OBD'en er fordi at OBD-II aldrig vil sende data på egen vis.
Den 'lytter' efter en kommando som du sender og baseret på den kommando, så giver den et svar (agere som en slave enhed).

OBD'en fungere i praksis som et modem, hvorfor der sendes AT (Hayes command set) kommandoer afsted for at påvirke datastrømmen.
Det anbefales at nedenstående kommandoer sendes afsted for at få en ren datastrøm.

atsp6: Vælger CAN protokol ISO15765-4 11/500.

ate0: slår ekko fra.

ath1: Sætter header på.

atcra: Sætter PID adresse

atcaf0: Slår auto-formatering fra

ats0: Ingen mellemrum i can besked

atma: Starter læsning af alle valgte PIDs.

I android kan du sende kommandoer til OBD'en ved at bruge nedenstående linje. Husk at udskifte kommando feltet med den ønskede kommando.

```
sendData(outputStream, "kommando");
```

Herunder er et eksempel angivet hvor en metode sender de tidligere nævnte kommandoer afsted til en OBD.

```
public void askForData()
{
 try
   {
      sendData(outputStream, "atsp6");
      sendData(outputStream, "ate0");
      sendData(outputStream, "ath1");
      sendData(outputStream, "atcra");
      sendData(outputStream, "atcaf0");
      sendData(outputStream, "atS0");
      sendData(outputStream, "atma");
   } catch (IOException e)
   {
      e.printStackTrace();
   }
```

Ovenstående ses et eksempel på en metode ved navn askForData. Metoden indeholder try handleren der forsøger at sende kommandoer til OBD'en mens catch handleren agere ved mislykket afsending og står for at printe stakken til konsollen.

Data mining i Android

Når man skal til at lave data mining altså finde den relevante information, bruges InputStream. Følgende kald vil prøve at læse datavekslingen, finde længden samt gemme indholdet til bufferen.

```
.read(buffer);
```

Hvad man efterfølgende vil gøre er at konvertere bufferen til en string da det er mere overskueligt at arbejde med bogstaver og tal.

```
new String(buffer, "ASCII");
```

Herefter vil man gerne læse PID'et af beskeden.

```
.substring(0, 3);
```

Når man så har fundet det PID man søgte, kan man lave en Kontrol erklæring så man kun modtager relevant data for sit fundene PID.
Det demonstreres i slutningen af afsnittet.

Hvis du gerne vil arbejde med Havariblink er det fundene PID for bilens lamper 423. Herefter skal man så eksperimentere med venstre blinklys, højre blinklys, samt havari for at se hvilken byte der ændrer sig. I tilfældet med havari, er det position 6 der ændrer sig. For at være helt sikre, konverteres de modtagende hex værdier til binære tal.

Hex værdierne for lamperne er følgende:

C konverteret til binære tal = **1100**
D konverteret til binære tal = **1101**
E konverteret til binære tal = **1110**
F konverteret til binære tal = **1111**

hvad du kan aflæse fra ovenstående er at position 3 og 4 af den binære data er det eneste der ændre sig. Derudover kan man se et mønster. Når alle lygter blinker, så repræsenteres det som C eller 1100, position 3 og 4 er altså 00. Når man tænder højre blinklys, repræsenteres det som D eller 1101, altså er position 3 nu 0 og position 4 nu 1. Når man tænder venstre blinklys repræsenteres det som E eller 1110, altså er position 3 nu 1 og position 4 nu 0. Dette giver et klart mønster om at havariblink så må være 1111 og det er det selvfølgelig, repræsenteret som F.

Nedenstående vises en metode for at lytte samt finde data for havariblink.

```
protected void startListening()
{
   // Forbereder byte array med 20 data, eftersom vores data som vi lytter efter er 20 i
længde.

   byte[] buffer = new byte[20];

   // Forbereder besked bufferen.
   String data;

   /**
* lytter efter data så længe der ikke er nogen afbrydelse fra tråd
<kode>!Thread.currentThread().isInterrupted()</kode>
* og BluetoothSocket stadig forbindes <kode>currentSocket != null</kode>
* samt at der ikke er nogle kommando der afbryder
lytningen<kode>!stopReceiving<kode>     */
```

```
   while (!Thread.currentThread().isInterrupted() && currentSocket != null &&
!stopReceiving)
   {
       try {

// Prøver at læse fra din Bluetooth stream, gemme data i bufferen samt finde
længden.
            int bytesRead = inputStream.read(buffer);

// Hvis længden af data er 20, så er det muligvis den rigtige data vi har med at gøre,
// ellers ignorere vi dataen.
            if (bytesRead == 20)
            {

// Konverter bufferen til en string.
                String message = new String(buffer, "ASCII");

// Læs PID af beskeden fra Bluetooth enheden.
                String pid = message.substring(0, 3);

 // Hvis PID er 423, den rigtige, så fortsæt, ellers ignorer det.
                if (pid.equals("423"))
                {
                    data = "";

 // Læs position 6 fra den fulde besked (C,D,E,F).
                    String dataPart = message.substring(6, 7);

 // Konverter beskeden til binære tal.
                    String bin = new BigInteger(dataPart, 16).toString(2);

// Hvis binære data har 4 i længde og position 3 og 4 er 11, så er havari tændt.
                    if (bin.length() > 3 && bin.substring(2, 4).equals("11"))
                    {
                        data = "accident on";
                    }
//  Send data til aktiviteten hvor data skal bruges.
                    EventBus.getDefault().post(data);

 // Fjern tidligere beskeder og forbered på en ny besked igen.
                    buffer = new byte[20];
                }
```

```
            }
 //Fang undtagelse hvis mislykket.
        } catch (IOException e)
        {
//Print stakken til konsollen.
            e.printStackTrace();
        }
    }
}
```

Anvendelsesmuligheder i Android

Der er en masse data der kan benyttes til at optimere sikkerheden for bilister på vejene. Nogle ting vi har taget i betragtning og der evt. kunne indføres gennem OBD eller arbejdes videre på i fremtiden generelt:

- Grader rattet bliver drejet i forhold til tiden kan indikere når bilister drejer skarpt på vejene for at nødværge. Disse målinger kan deles med andre bilister.
- Airbag. Hvis airbaggen bliver udløst ved et uheld, eventuelt ved en eneulykke, hvor der er begrænset mulighed for hjælp andre steder og der ikke er muligvis fra bilistens side at aktivere havariblinket.
- Bremser. På samme måde som rattet kan bilens bremser blive målt. Hvis der bliver bremset hårdt (nok) kan det være en potentiel fare.
- Is på vejene eller andet glat føre kan deles mellem bilister.
- Ambulancer, politibiler og brandbiler der er i udrykning kan give advarsel om deres ankomst på vejene.

Nogle af disse punkter kræver beregninger hvori fysiske omstændigheder tages nøjes i betragtning.