



**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

**Monitorizarea de la distanță a datelor din monopostul ART  
TU E17 pentru depanare și optimizare, utilizând tehnologia  
LoRa**

LUCRARE DE LICENȚĂ

Absolvent: **Tompea Radu-Gabriel**

Coordonator **Conf. Dr. Ing. Bogdan Iancu**  
științific:

**2025**

# Cuprins

<b>Capitolul 1 Introducere</b>	<b>1</b>
<b>Capitolul 2 Obiectivele proiectului</b>	<b>3</b>
2.1 Analiza cerințelor specifice monopostului ART TU E17 . . . . .	3
2.2 Obiectiv principal . . . . .	3
2.3 Obiective specifice . . . . .	3
2.4 Cerințe funcționale . . . . .	4
2.5 Cerințe non-funcționale . . . . .	4
<b>Capitolul 3 Studiu bibliografic</b>	<b>6</b>
3.1 Sisteme de monitorizare a datelor în timp real . . . . .	6
3.1.1 Introducere . . . . .	6
3.1.2 Scopul unui sistem de telemetrie . . . . .	6
3.1.3 Sistem de telemetrie în domeniul industrial sau militar . . . . .	6
3.1.4 Schema unui sistem de telemetrie în automotive . . . . .	7
3.1.5 Sisteme de telemetrie în istorie . . . . .	8
3.2 Starea actuală a sistemelor de telemetrie în automotive . . . . .	10
3.2.1 Caracteristici . . . . .	10
3.2.2 Avantaje . . . . .	10
3.2.3 Scenarii de utilizare . . . . .	11
<b>Capitolul 4 Analiză și fundamentare teoretică</b>	<b>14</b>
4.1 Cazuri de utilizare . . . . .	14
4.1.1 Cazul 1: Monitorizarea datelor care vin de la toți senzorii monopostului . . . . .	14
4.1.2 Cazul 2: Monitorizarea datelor care vin de la senzorii specifici unui departament . . . . .	15
4.1.3 Cazul 3: Monitorizarea evoluției unui senzor specific în timp real .	16
4.1.4 Cazul 4: Preluarea datelor din Cloud și vizualizarea lor . . . . .	17
4.2 Componentele sistemului de telemetrie wireless ART TU E17 . . . . .	18
4.2.1 Protocole de comunicare . . . . .	18
4.2.2 Componete software . . . . .	23
4.2.3 Componete electrice . . . . .	25
4.2.4 Tehnici de programare . . . . .	29
4.3 Soluții existente la alte echipe de Formula Student . . . . .	30
4.4 Arhitectura monopostului ART TU E17 . . . . .	30
<b>Capitolul 5 Proiectare de detaliu și implementare</b>	<b>32</b>
5.0.1 Arhitectura sistemului de telemetrie ART TU E17 . . . . .	32
5.0.2 Diagrama de flux . . . . .	33
5.0.3 Diagrama de secvență . . . . .	35
5.0.4 Modulul 1: Partea relevantă a monopostului ART TU E17 . . . . .	36
5.0.5 Modulul 2: Transmițătorul LoRa . . . . .	38
5.0.6 Modulul 3: Receptorul LoRa . . . . .	42
5.0.7 Modulul 4: Baza de date locală din Road-Side Unit (Server) . . . . .	48

5.0.8	Modulul 5: Interfața grafică a utilizatorului . . . . .	50
5.0.9	Modulul 6: Baza de date din Cloud . . . . .	51
<b>Capitolul 6</b>	<b>Testare și validare</b>	<b>56</b>
6.1	Testarea și rezultatele cazurilor de utilizare . . . . .	56
6.1.1	Testarea și validarea cazului 1 . . . . .	56
6.1.2	Testarea și validarea cazului 2 . . . . .	57
6.1.3	Testarea și validarea cazului 3 . . . . .	58
6.1.4	Testarea și validarea cazului 4 . . . . .	58
6.2	Testarea și validarea distanței efective a sistemului . . . . .	59
6.3	Testarea și validarea vitezei de actualizare a interfeței grafice . . . . .	60
6.4	Testarea și validarea sistemului pe monopost . . . . .	61
<b>Capitolul 7</b>	<b>Manual de instalare și utilizare</b>	<b>64</b>
7.1	Cerințe ale sistemului și resurse . . . . .	64
7.1.1	Cerințele sistemului . . . . .	64
7.1.2	Resurse Hardware . . . . .	64
7.1.3	Resurse Software . . . . .	65
7.2	Instalarea resurselor software . . . . .	65
7.2.1	Instalare Docker . . . . .	66
7.2.2	Instalare Python . . . . .	66
7.3	Instalarea și utilizarea aplicației . . . . .	67
<b>Capitolul 8</b>	<b>Concluzii</b>	<b>71</b>
<b>Bibliografie</b>		<b>73</b>
<b>Anexa A</b>	<b>Secțiuni relevante din cod</b>	<b>76</b>
A.1	Transmițător LoRa cu AES-128 CTR . . . . .	76
A.2	Receptor LoRa cu AES-128 CTR. . . . .	81
A.3	Scriptul de transfer de la receptor la InfluxDB . . . . .	91
<b>Anexa B</b>	<b>Lucrări științifice care susțin rezultatele prezentate</b>	<b>93</b>
B.1	Computer Science Student Conference 2025 (CSSC) . . . . .	93

# Capitolul 1. Introducere

Am intrat în era digitală a informației, iar nevoia de acces instantaneu la informații devine esențială, atât pentru oamenii care vor să folosească aplicații pentru management urban sau navigarea în rutina zilnică, cât și pentru sisteme informatiche, pentru a lua o decizie cât mai precisă, cu date actualizate la minut și interacțiuni fără întârzieri sau blocaje.

Pentru a putea realiza aceste instrumente precise, monitorizarea datelor joacă un rol esențial în dezvoltarea noilor tehnologii și infrastructuri moderne, îmbunătățind calitatea vieții oricărui om. În același timp, noile descoperiri în tehnologie se aplică și în industria automotive. Aceste integrări ajută ingineriei să dezvolte autovehicule mai sigure, mai eficiente și mai inteligente, îmbunătățind industria transporturilor, atât în domeniul privat, cât și în cel public.

Generația modernă de autovehicule este echipată cu o rețea complexă de senzori și sisteme de siguranță care achiziționează în timp real informații critice, de la performanța motorului și consumul de combustibil, până la parametrii de frânare sau unghiul volanului. Aceste date sunt constant analizate, oferind șoferilor o experiență mult mai sigură și plăcută a drumului, prin sistemul de frânare de urgență sau ajustarea dinamică a suspensiei.

Pe lângă transportul sigur și eficient al șoferului, astfel de sisteme ajută înzecit inginerul care proiectează autovehiculul respectiv, oferindu-i informații pentru a putea vedea care sunt problemele și unde excelează produsul finit sau în dezvoltare.

De-a lungul timpului, verificarea performanței oricărei tehnologii la nivel de date a fost un proces anevoieios, arătând nevoia unui sistem de transmitere a datelor la distanță, cu vizualizare intuitivă, pentru a putea observa mult mai ușor dacă un sistem dintr-un autovehicul funcționează în parametrii optimi. Acest fel de sisteme s-a dovedit a fi crucial în procesul de dezvoltare, iar fiecare companie prezintă o soluție unică pentru a rezolva această problemă. În industrie se preferă un mod de transmitere a datelor wireless pentru a ușura achiziționarea de date, în special în domeniul unde un sistem cablat ar îngreuna semnificativ lucrurile.

Scopul acestui proiect este de a oferi date reale care să tină de eficiență și performanța monopostului electric (mașina de curse) **ART TU E17** din cadrul echipei de Formula Student ART TU. Datele vor fi **transmise în timp real, folosind tehnologia LoRa**, la o distanță posibilă de peste un kilometru cât timp monopostul funcționează. Acest proiect permite monitorizarea și analiza acestor date în scop de dezvoltare ulterioară, dar și găsirea problemelor apărute pe circuit, ușurând astfel procesul de identificare a problemelor des întâlnite în autovehicule. Folosind o interfață grafică intuitivă și un proces de stocare a datelor, proiectul permite echipei să vizualizeze datele atât în timp real (cât timp monopostul este pe circuit), dar și după ce acesta a terminat momentul de testare. Proiectul este bazat pe o arhitectură care nu necesită accesul constat la internet și permite conectarea mai multor utilizatori la interfața clientului, având și o conexiune rezistentă la un număr mare de factori perturbatori (zgomot electromagnetic, zone de testare foarte populate, zgomot generat de componente mecanice).

În ceea ce privește componența tehnică, proiectul își propune organizarea datelor

generate de monopost într-un format care îi permite să fie transmis la distante mari, cu tehnologie care este rezilientă zgomotelor produse din varii surse, și vizualizarea acestor date într-o interfață grafică, accesibila tuturor membrilor echipei. Prin urmare, lucrarea mea se va concentra asupra diferitelor tehnologii folosite pentru trimiterea datelor la distanță într-un mod sigur, precum și asupra factorilor care au condus la această decizie, respectiv asupra integrării acestei tehnologii în arhitectura existentă a monopostului. Această lucrare oferă un sistem complet de gestionare și trimitere a datelor de la monopost la un server, care agreghează și filtrează datele și ajută echipa de depanare și dezvoltare în găsirea problemelor monopostului print-o vizualizare intuitivă și prin accesarea datelor ulterior testării dinamice.

Pentru orice proiect care ține de automotive, **Formula Student** este un loc unde se pot testa proiectele în condiții cât mai apropiate de realitate. Este o competiție globală care acționează ca un magnet pentru tinerii ingineri care gravitează spre lumea automotive. Scopul unei echipe de Formula Student este de a proiecta, construi și testa un monopost construit în totalitate de ei, într-un mediu similar cu cel real în ceea ce privește reglementările și provocările. Este un mediu atractiv pentru mulți tineri, devenind din ce în ce mai popular pe zi ce trece.

Aici, atât profesorii, companiile cât și studenții pot încerca diferite soluții la probleme reale care intervin în dezvoltarea și fabricarea unui monopost de curse, inclusând oameni din diferite domenii precum proiectare, testare, marketing, bugetare, management și multe altele, oferind oportunitatea de a aplica cunoștințele învățate din facultate într-un context real, cu consecințe reale dar mult mai mici comparativ cu un mediu de producție, oferind experiență foarte valoroasă.

Formula Student caută să dezvolte abilitatea unui student de a inova, eficientiza și de a rezolva probleme care nu țin doar de sfera tehnică. Ca și în lumea reală a dezvoltării automobilelor, există un regulament care trebuie respectat de fiecare echipă care participă.

Dezvoltarea rapidă a tehnologiei ajută studenții în găsirea rezolvărilor, folosind materiale ușoare, rezistente, sisteme de comunicare extrem de rapide, și cele mai eficiente moduri de a atinge cea mai mare viteză sau eficiență, punându-se un accent foarte mare în cadrul Formula Student pe **mașinile electrice** și pe impactul lor în creearea și menținerea unui mediu sustenabil și plăcut.

Această competiție a devenit un mod foarte valoros pentru intrarea în piața muncii, oferind tinerilor absolvenți o experiență inegalabilă în ceea ce privește abilitățile lor tehnice, de leadership, gestionare a timpului și cunoștințe generale din lumea ingineriei, iar cu privirea orientată spre soluții de mobilitate diferite de cele tradiționale arată cât de contribuie un astfel de mediu pentru dezvoltarea societății.

**Universitatea Tehnică din Cluj-Napoca** participă anual la competițiile Formula Student cu echipa **ART TU Cluj-Napoca**, care a fost fondată la începutul anului 2019 cu 20 de membri inițial, care au ajuns dealungul anilor să fie aproximativ 120 de membri activi, care participă în diferitele departamente ale echipei, precum dinamica vehiculului, sistemul de voltaj mic, sistemul de tracțiune și voltaj mare, iar în final departamentul care se ocupă de mecanica vehiculului. Începând din 2022, echipa ART TU a fost prezentă la competiția **Alpe Adria** din Croația, **Formula Student Czech** din Cehia în 2023 și 2024, iar în final, **Formula Student Balkans** în 2024, care a fost organizat în România, în orașul Dej.

## **Capitolul 2. Obiectivele proiectului**

În acest capitol, se vor discuta obiectivele proiectului și pașii necesari pentru a le realiza. Voi începe de la un obiectiv principal și apoi o serie de obiective specifice proiectului.

### **2.1. Analiza cerințelor specifice monopostului ART TU E17**

Ca obiectiv teoretic trebuie identificate și analizate condițiile la care este supus monopostul în timp ce este pe circuit, găsind sursele care pot adăuga impiedimente la funcționarea optimă a transmiterii, recepționării și interpretării datelor. Următorul pas este de a mitiga aceste surse sau găsirea unei soluții care evită aceste probleme identificate, cu schimbări minime la monopost și cu un cost redus. Când aceste surse perturbatoare au fost reduse, trebuie integrat sistemul de telemetrie pentru a transmite datele la distanțe mari și de a instala resursele necesare pentru a le putea interpreta. Ultimul pas este de a integra o metodă pentru a realiza persistența datelor după ce acestea au fost transmise.

### **2.2. Obiectiv principal**

Obiectivul principal al proiectului “Monitorizarea de la distanță a datelor din monopostul ART TU E17 pentru depanare și optimizare, utilizând tehnologia LoRa” este de a proiecta, dezvolta și implementa un sistem de monitorizare a datelor pentru monopostul de Formula Student care este rezistent la o multitudine de factori, descriși ulterior. Proiectul urmărește să dezvolte o arhitectură care va rezolva problemele referitoare la transmiterea de date la distanță și persistența datelor în sistem, folosind diferite tehnici și tehnologii actuale. Produsul final este un sistem eficient, rezilient și scalabil care permite persistența și acuratețea datelor provenite de la monopost într-o interfață intuitivă cu pași de instalare minimali.

### **2.3. Obiective specifice**

Pentru a putea realiza un sistem de transmitere a datelor la distanță mare se vor defini următoarele obiective:

1. **Proiectarea arhitecturii sistemului.** Pentru realizarea acestui obiectiv trebuie identificate și analizate condițiile în care este supus monopostul în timp ce este pe circuit. Bazat pe acestea, se pot deduce cerințele sistemului cu privire la impiedimentele ce pot apărea la nivelul hardware și cum se pot contracara, pe lângă proiectarea unei arhitecturi care permite un flux de date mare și reziliență la condiții dificile.
2. **Identificarea datelor relevante.** Acest obiectiv se axează pe selectia datelor relevante pentru restul echipei în ceea ce privește identificarea problemelor pe monopost și alegerea unei tehnologii care suportă volumul de date ridicat.
3. **Definirea structurilor de date și implementarea acestora la nivel hardware.** Pentru acest obiectiv este nevoie de proiectarea structurilor de date care să stocheze și să organizeze datele într-un mod eficient din punct de vedere al accesibilității și al implementării pe hardware-ul disponibil.

4. **Implementarea funcționalităților la nivel de server.** Acest obiectiv privește funcționalitățile de bază pe care server-ul trebuie să le îndeplinească, dar și dezvoltarea, respectiv implementarea acestora. Trebuie luate în calcul testarea și validarea acestor funcționalități, precum și integrarea acestora în sistemul final pentru a putea realiza o soluție viabilă din punct de vedere tehnic.
5. **Persistența datelor în condiții dificile.** Prin acest obiectiv îmi propun să dezvolt o strategie și un mediu care priorizează corectitudinea datelor în ciuda factorilor perturbatori care pot apărea într-un monopost electric, pe lângă stocarea datelor pe partea de server.
6. **Accesarea și vizualizarea datelor după sesiunea de testare.** Scopul acestui obiectiv este de a oferi o soluție care permite utilizatorului să acceseze datele și după ce monopostul nu mai rulează pentru a putea oferi echipei posibilitatea de a analiza datele cât timp monopostul este static sau în proces de modificare.

## 2.4. Cerințe funcționale

În cadrul acestei lucrări au fost definite cerințele funcționale ale sistemului propus:

- **Colectarea și înregistrarea datelor relevante din monopost.** Sistemul trebuie să achiziționeze date de la toți senzorii relevanți ai monopostului și să recunoască tipul lor de date.
- **Stocarea acestora într-o structură de date eficientă pentru transmiterea acestora.** Sistemul ar trebui să stocheze temporar datele într-o structură de date adecvată pentru transmisia la distanță.
- **Configurarea și gestionarea mediului de transmitere la distanță.** Sistemul ar trebui să fie configurat în concordanță cu hardware-ul disponibil pentru a acoperi cea mai mare distanță posibilă.
- **Comunicarea între monopost și server.** Sistemul trebuie să furnizeze o comunicare dintre monopost și server constantă pe durata rulării, fără întreruperi și fără să compromită integritatea datelor, pe o distanță de **0,8 către 1 km non-line-of-sight**.
- **Prelucrarea și stocarea datelor provenite de la monopost.** Sistemul trebuie să aibă capacitatea de a agrega datele transmise de la distanță într-o formă procesabilă și să le stocheze într-un mod avantajos din punct de vedere al memoriei și al timpului de procesare.
- **Monitorizarea în timp real a datelor.** Sistemul ar trebui să ofere un mod de vizionare în timp real al modificării datelor provenite din funcționarea monopostului și a sistemelor sale de siguranță, cu o actualizare a datelor la secundă.
- **Stocarea persistentă a datelor pentru accesare ulterioară.** Sistemul ar trebui să stocheze datele pentru a putea fi accesate după ce monopostul nu mai rulează pentru a fi analizate ulterior.
- **Vizualizarea datelor după ce transmisia s-a încheiat.** Sistemul ar trebui să furnizeze un mod de vizualizare grafică a datelor după ce transmisia s-a încheiat pentru a putea analiza datele cu ușurință și accesarea lor ulterioara pentru comparații.

## 2.5. Cerințe non-funcționale

În cadrul acestei lucrări au fost definite și cerințele non-funcționale ale sistemului propus:

- **Performanță.** Sistemul trebuie să ofere o responsivitate ridicată pentru a putea monitoriza datele la un timp de actualizare minim.

- **Scalabilitate.** Sistemul trebuie să ofere posibilitatea, fie din arhitectură fie din implementare, de a fi adaptat unui număr mai mare de date sau o transmitere pe o distanță mai ridicată.
- **Securitate.** Sistemul trebuie să incorporeze o modalitate de a securiza datele transmise și stocate, persoanele neautorizate neavând acces la datele monopostului.
- **Disponibilitate și Accesibilitate.** Sistemul trebuie să ofere stabilitate dar și disponibilitate pentru a putea accesa constant serviciile acestuia, ce pe orice dispozitiv ar avea utilizatorul.
- **Flexibilitate.** Sistemul trebuie să fie “plug-and-play”, fiind ușor de instalat și modificat pentru diferitele cerințe ale monopostului în sezoanele ulterioare.
- **Design intuitiv.** Sistemul trebuie să fie ușor de folosit de un utilizator la prima vedere și ușor de navigat de un utilizator care nu este familiar cu zona IT.

În figura 2.1 se poate observa principul de funcționare al sistemului de telemetrie al ART TU E17.

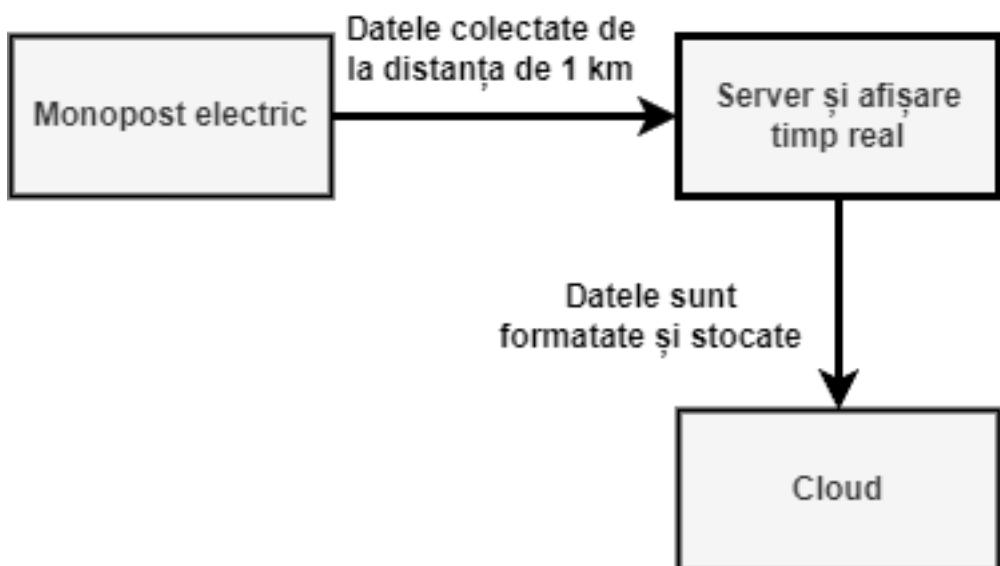


Figura 2.1: Colectarea, formatarea și stocarea datelor dintr-un sistem de timp real.

## Capitolul 3. Studiu bibliografic

### 3.1. Sisteme de monitorizare a datelor în timp real

#### 3.1.1. Introducere

În orice proces de dezvoltare sau menenanță trebuie să existe o formă de monitorizare a datelor, fie aceasta în timp real, stocată pe memorie și apoi vizualizată sau pur și simplu trasarea unui flux de date, este o componentă crucială în orice parte a unui produs digital. Pe parcursul acestei lucrări, o să mă refer la monitorizarea datelor prin termenul **telemetrie**.

Telemetria reprezintă măsurarea automată și transmiterea datelor de la surse distante wireless. Senzorii măsoară fie date electrice, precum voltaj și curent, cât și date fizice, precum temperatură sau presiune, care sunt trimise de dispozitive electronice în locații aflate la distanță pentru observarea parametrilor acestora. [1]

Developerii software și administratorii IT folosesc telemetria wireless pentru a urmări starea, securitatea și performanța aplicațiilor și a componentelor în timp real, observând timpi de procesare și folosirea resurselor sau determinarea stării unui sistem.

#### 3.1.2. Scopul unui sistem de telemetrie

Scopul unui sistem de telemetrie este de a colecta date dintr-un loc care este depărtat sau incomod și să trimită datele la un loc unde acele date vor fi evaluate. În mod normal, sistemele de telemetrie sunt folosite în testarea vehiculelor în mișcare precum automobile, avioane sau rachete, fiind un set special de comunicații. Când un sistem de telemetrie este folosit pentru control și achiziționare de date, termenul “controlul supravegherii și achiziția de date” este folosit. [2]

#### 3.1.3. Sistem de telemetrie în domeniul industrial sau militar

Un sistem de telemetrie în domeniul industrial sau militar este compus din 7 componente, ale căror funcționare se poate observa în figura 3.1:

- Un sistem de colectare a datelor.
- Un sistem de multiplexare.
  - Un sistem de multiplexare a diviziunilor de frecvențe.
  - Un sistem de multiplexare a diviziunilor de timp.
  - Un sistem hibrid, care este o combinație a multiplexării a diviziunilor de frecvență și a diviziunilor de timp.
- Un modulator, transmițător și antenă.
- Un mod de transmitere a datelor.
- O antenă, un receptor de frecvențe radio, o secțiune de frecvență intermediară, demodulator al datelor.
- Un sistem de demultiplexare a diviziunilor de timp sau de frecvență.
- Un sistem de procesare a datelor. [2]

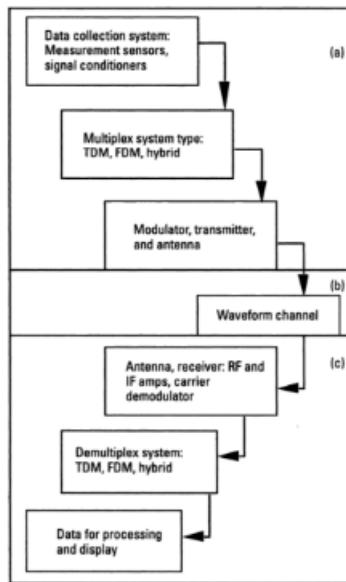


Figura 3.1: Schema unui sistem de telemetrie în domeniul industrial sau militar [2]

Acest tip de sisteme de telemetrie au fost folosite în testarea de zbor (flight testing) a unei rachete, fiind un sistem care trebuie să fie rezilient la multe tipuri de zgromot și perturbări, pe când un mediu cablat ar fi dificil de implementat.

#### 3.1.4. Schema unui sistem de telemetrie in automotive

Un astfel de sistem de telemetrie de obicei este mare și greu care necesită un invertor de putere CA (current alternativ), iar pentru un monopost folosit pentru Formula Student (care poate avea lungimea maximă de 1,525 metri [3]) nu este o soluție posibilă. Prin urmare, un astfel de monopost necesită o soluție care are o alimentare de 12V, care este mic și ușor. [4]

Un sistem de telemetrie care este folosit în automotive are șapte componente, iar principiul de funcționare se poate observa în figura 3.2: [5]

- Un sistem de alimentare
- Un modul “Real-Time-Clock”
- Un microcontrolor
- Un modul de transmisie
- EEPROM (Electrically Erasable Programmable Read-Only Memory)
- Datele de la senzori
- Un card de memorie flash detașabil

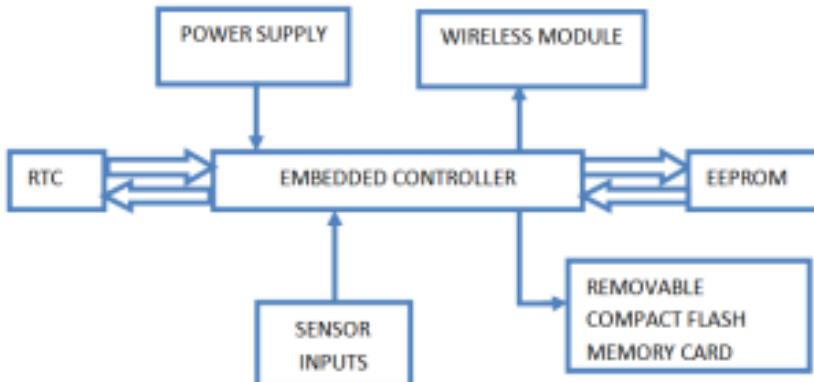


Figura 3.2: Schema unui sistem de telemetrie în domeniul automotive [5]

### 3.1.5. Sisteme de telemetrie în istorie

De-a lungul istoriei cercetării, sistemele de telemetrie au fost cruciale, fiind un concept necesar pentru a investiga cum funcționează diferite domenii sau arii de expertiză. În 1992, autorii definesc telemetria sau biotelemetria ca determinarea de la distanță a stării unui animal, incluzând nivelul curent de activitate, măsurători fiziologice (temperatura corporală, ritmul inimii, etc.) și locația lui fizică.

De aproape 40 de ani, biologi ai vietii sălbatice au folosit radiotelemetria pentru a studia animalele în libertate în mediul lor natural, și, de-a lungul timpului, cercetătorii au adaptat tehnologii noi de telemetrie în diversele lor arii de studii.

O abordare comună în folosirea urmăririi prin intermediul radio presupune atașarea unui radiotransmițător de frecvență mare (Very High Frequency VHF) unui animal sub forma unei zgarde, care include o sursă de alimentare și o antenă. Semnale radio pe bază de puls sunt folosite pentru a estima locația animalului, fiind identificate prin frecvență unică și rata de puls a transmițătorului pe care îl poartă. [6]

Pe lângă alte limitări, cea mai mare barieră era distanța semnalului transmițătorului. Pe când transmițătoarele VHF pot avea o distanță de 3-20km, multe animale mari și specii migratoare se deplasează la distanțe de sute și chiar mii de kilometri. Aceasta problemă s-a rezolvat prin folosirea avioanelor pentru urmărire, dar aceasta implica costuri foarte mari cu cât aria de căutare devinea mai mare.

Cu nevoie tot mai mare de telemetrie, s-au făcut progrese mari de-a lungul anilor 1990, variind de la gestionarea curentului (power management), componente electronice și microcontroloare, dar și a sistemelor bazate pe noi tehnologii, punându-se accent pe mărimea redusă a sistemelor, acuratețe sporită și o automatizare a sistemelor de telemetrie.

Printre componente care au ajutat exponențial la dezvoltarea sistemelor de telemetrie se numără: [6]

- **Microcontroloare**

- La sfârșitul anilor 1980 și începutul anilor 1990 microcontroloarele single-chip au devenit populare și aveau consum redus (20,-3,0V, 0,015-0,020mAh). Acestea erau mult mai complexe decât ce s-a folosit în trecut, incorporând EPROM, RAM, porturi de comunicare I/E și cel mai important, temporizare bazată pe cristale de quartz, fiind și mult mai ușor de controlat de la distanță
- Acestea au devenit cruciale și în automatizarea sistemelor de telemetrie, receptorii permitându-le utilizatorilor să itereze printr-un set programat de frecvențe,

ascultând pe fiecare pentru o perioadă scurtă de timp

- **Transmițători codati (coded transmitters)**

- Prin codarea semnalelor transmise, mai mult decât un utilizator poate fi asignat unei frecvențe și aceleiași coduri pot fi folosite pe mai multe frecvențe
- Dezvoltarea circuitelor de controlul al pulsului prin CMOS în anii 1980 permite ca un singur puls să fie înlocuit cu un grup de două sau mai multe pulsuri distanțate egal unde numărul de pulsuri, timpul dintre pulsuri și rata de repetiție pot identifica transmițători individuali, fiind prezentat cum funcționează în figura 3.3.

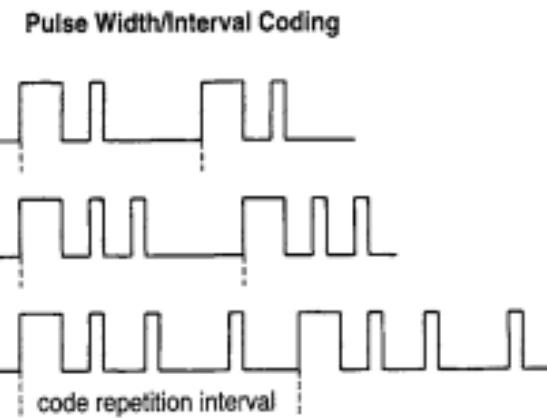


Figura 3.3: Un exemplu de transmițător codat [6]

- **Senzori**

- O mare parte din tehnologia modernă care a îmbunătățit sistemele de telemetrie codate a beneficiat de performanța senzorilor care sunt incorporate în dispozitivele de urmărire.

- **Data Loggers**

- Etichetele arhivale (data loggers) au fost inițial propuse de Holster în 1961 ca un mod de a înregistra pulsul inimii pacenților în aplicări medicale unde monitorizarea constantă nu este tot timpul posibilă.

Cu incrementale evoluții a diferitelor sisteme, s-a ajuns la diferite moduri de a implementa telemetrie în diverse domenii: [2]

- **Sisteme de telemetrie bazate pe satelit**

- Au fost inițial dezvoltate pentru a monitoriza date despre mediu precum presiune atmosferică și temperatură, pe lângă furnizarea locațiilor geografice pentru meteorologi sau oceanografi. Primii receptori ai unui astfel de sistem au fost montați pe sateliții de vreme Nimbus (între 4,5-16 kg), iar implementări ulterioare fiind sistemul Argos.

- **Sisteme de telemetrie bazate pe GPS**

- Deși sistemele bazate pe satelit au fost o descoperire imensă, acestea puteau să monitorizeze mișcările animalelor pe un spectru larg, acuratețea lor în ceea ce privește locația era insuficientă pentru unele studii. Această limitare a pornit dezvoltarea sistemului NAVSTAR Global Positioning System (GPS) în 1992, iar de atunci multe configurații au fost dezvoltate și testate și un număr mare de îmbunătățiri a fost adus acestor sisteme.
- Acestea puteau oferi utilizatorilor să obțină estimări ale locației (fixes) 24 de ore din zi, având 3 componente: un segment de spațiu de 24 de sateliți, care

tot timpul difuzau semnale radio cu spectru extins, o rețea de monitorizare pe sol și stații de control care mențin timpul standard al sistemului și calculează informația orbitală

- **Sisteme de telemetrie hiperbolice**

- Sistemele acestea se bazează pe luarea măsurătorilor a diferențelor de timp în care semnalele de la transmițători să ajungă la trei sau mai multe stații de recepție, formându-se funcții hiperbolice din diferența dintre timpul în care a fost recepționat semnalul între două stații, iar punctul de intersecție dintre aceste hiperbole este folosit pentru a determina locația transmițătorului.

## 3.2. Starea actuală a sistemelor de telemetrie în automotive

### 3.2.1. Caracteristici

Cunoscând acum ce este și cum funcționează un sistem de telemetrie în automotive, voi prezenta principalele caracteristici ale acestuia, pentru a putea vedea pe ce se bazează: [7]

1. **Timă de răspuns rapid.** Un sistem de telemetrie are ideal o latență redusă, notificând utilizatorul cât mai repede din momentul în care se detectează o valoare anormală până aceasta este recepționată și procesată.
2. **Ușor de monitorizat date în medii ostile sau dificile.** Sistemele de telemetrie sunt proiectate să funcționeze în medii dure și ostile, unde factorii de mediu pot afecta negativ transmisiile de date, folosindu-se senzori industriali sau cu protecție electromagnetică (IP67+)
3. **Realocare ușoară.** În majoritatea aplicațiilor, sistemele de telemetrie pot fi ușor mutate datorită modularității arhitecturii hardware și software și folosirii comunicațiilor wireless și domeniul hardware.
4. **Funcțional într-o gama largă de condiții de operare.** Aceste sisteme pot să funcționeze la variații mari de umiditate, intervale mari de temperatură, sau în mobilitate, fiind aplicate în multe domenii, dar în special automotive.
5. **Interoperabilitate.** Deseori, acestea respectă standarde deschise precum IEEE 802.11p sau C-V2X, permitând comunicarea între vehicule și infrastructuri de la producători diferiți. [8]

### 3.2.2. Avantaje

În nevoie de un astfel de sistem în echipa ART TU, am văzut rapid care sunt beneficiile acestuia, mai ales o implementare wireless:

- **Optimizarea consumului de combustibil / energie.** Prin evitarea frânerilor și acelerărilor inutile (ex. Semafoare coordonate), consumul de combustibil sau energie electrică este redus.
- **Reducerea emisiilor poluante.** Prin optimizarea traseelor și reducerea staționarilor inutile, scade consumul și implicit emisiile.
- **Suport pentru infrastructuri inteligente (smart cities).** Permite interacțiuni eficiente între vehicule, semafoare, camere de trafic și centre de control.
- **Actualizări și diagnosticare de la distanță.** Vehiculele pot primi informații, software sau alerte fără a merge în service.

### 3.2.3. Scenarii de utilizare

1. **Vehicle-To-Everything (V2X).** Vehicle-To-X este un termen generic care se referă la un sistem de comunicație wireless care permite schimbul de date în timp real între un vehicul și orice entitate din mediul său, inclusiv alte vehicule (V2V), infrastructură rutieră (V2I), rețele de comunicație (V2N) sau rețea electrică (V2G). Principalul scop al acestei tehnologii este de a îmbunătăți siguranța pe drum și a traficului. [9]
2. **Vehicle-To-Building (V2B).** Această formă de comunicație se bazează pe interacțiunea dintre un vehicul și sistemele inteligente ale unei clădiri, precum sistemul energetic și de securitate pentru a permite schimbul de date sau de energie electrică, fie de la vehicul spre clădire sau invers, fiind un factor important în optimizarea consumului energetic. [10]
3. **Vehicle-To-Network (V2N).** Prin această formă de comunicație se realizează comunicarea unui vehicul cu infrastructuri de rețea largi, precum servere cloud, centre de date sau rețele mobile (4G/5G), oferind informații externe precum condiții de trafic și vreme permitând vehiculului să ia decizii mai bine informate și să îmbunătățească experiența utilizatorului. [10]
4. **Vehicle-To-Vehicle (V2V).** Această tehnologie permite vehiculelor să schimbe direct informații între ele, în timp real, fără a fi nevoie de o infrastructură intermediară. Prin aceasta, se transmit date precum viteza, direcția, frânarea și poziția pentru a crește siguranța rutieră. [10]

Evoluția tehnologiilor noi ajută la definirea conceptului de sistem conectat. În acest deceniu, orice poate fi conectat, cu sau fără cablu.

O mare parte din cercetare a venit din partea industriei și a altor organizații pentru a aborda metodele și tehnologiile de comunicare în vehicule și infrastructuri de transport, incluzând în principal tehnologii de tip V2V, V2I și V2N. Acestea pot îmbunătăți siguranța tuturor și pot eficientiza sistemele de transport, fie cele în comun, fie cele de marfă.

Multe lucrări de specialitate au prezentat posibile arhitecturi care definesc conceptul de vehicule conectate. Eficiența acestei comunicări bi-directionale între sistemele de transport poate fi realizată doar prin utilizarea senzorilor inteligenți și a tehnologiilor avansate de comunicare. Termenul „vehicul conectat” este valabil doar dacă vehiculul poate comunica folosind platforma internetului. [11]

Totuși, termenul de vehicul conectat poate fi aplicat și pentru două sau mai multe vehicule aflate în aceeași zonă care se pot detecta reciproc, care pot detecta pietoni, biciclete, animale și indicatoare rutiere. Acest lucru este posibil chiar și în absența conexiunii la internet, iar vehiculele pot fi conectate folosind alte protocoale de comunicare precum XBee, Bluetooth și sisteme radio. Totuși, această comunicare nu poate fi eficientă fără senzori și camere instalate în orașe, infrastructuri sau vehicule. Chiar dacă aceste condiții sunt îndeplinite, eficiența sistemului în ansamblu nu poate fi garantată dacă nu există rețea de internet.

În general, V2B și V2I reprezintă o etapă intermediară între vehicul și conexiunea directă la rețea (V2N). Acest lucru prezintă mai multe avantaje în ceea ce privește pierderile de informații și problemele de economisire a resurselor.

În figura 3.4 se observă cum funcționează V2V, V2N, și V2B/I prezentând fiecare tehnologie ce mijloace are nevoie pentru a funcționa:

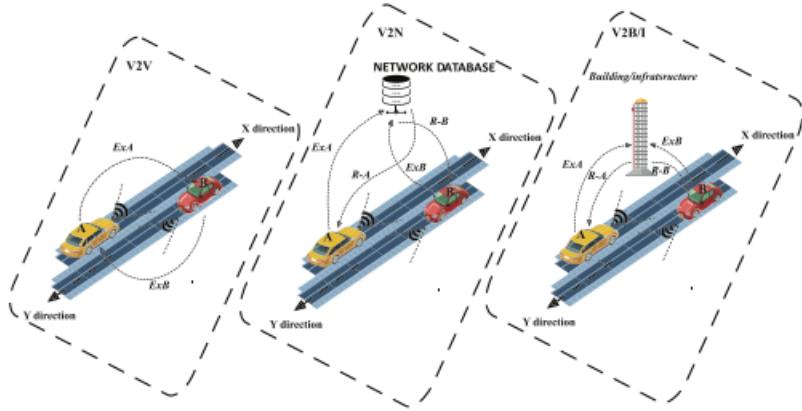


Figura 3.4: Exemple de comunicare V2V, V2N, V2B/I, respectiv. [11]

Spre exemplu, un sistem de transport poate partaja și schimba diverse informații legate de poziția mașinii, starea vehiculului și a șoferului, precum și situația energetică a vehiculului folosind V2B/I, V2N și V2V într-un oraș unde există clădiri și infrastructură, vehiculul și clădirile sau infrastructura pot face schimb de informații despre starea drumului, performanțele vehiculului și situația energetică reală. Aici, este posibil ca infrastructura sau clădirile să ofere vehiculului informații utile despre consumul energetic al altor vehicule din anumite zone, utilizând informații salvate anterior.

În afara orașului, clădirile și infrastructura nu au o prezență asa de mare, dar comunicarea în rețea este mai ușoară prin sistemele de transmisie wireless, precum 4G și, recent, 5G. Aceasta permite vehiculelor să fie conectate la rețea în majoritatea locurilor de pe șosea. În acest caz, vehiculele vor încărca și descărca informații din bazele de date cloud.

Bazându-ne pe cele mai noi tehnologii utilizate în vehiculele electrice, cum ar fi senzori inteligenți, antene GPS, camere inteligente, tehnologii avansate de conectivitate precum Bluetooth și tehnologii wireless, precum și pe puterea mare de procesare, se poate ajunge la o soluție realizabilă.

O astfel de soluție se bazează pe informațiile partajate între vehicule și entitățile din jur. Dacă fiecare vehicul își partajează propria experiență energetică și stil de condus, celelalte mașini pot folosi aceste date pentru a-și ajusta propriii parametri. Astfel, fiecarui vehicul își poate atribui cea mai potrivită metodă de condus. Această metodă de condus optimă este echivalentă cu un consum energetic eficient într-o anumită zonă de drum. Fiecare vehicul poate fi conectat la alt vehicul apropiat, la o clădire sau infrastructură, și la sistemul general de rețea. [11]

În figura 3.5 este prezentat cum fiecare din aceste tehnologii funcționează într-un oraș, arătând că este nevoie de o infrastructură complexă pentru a susține o tehnologie V2X:

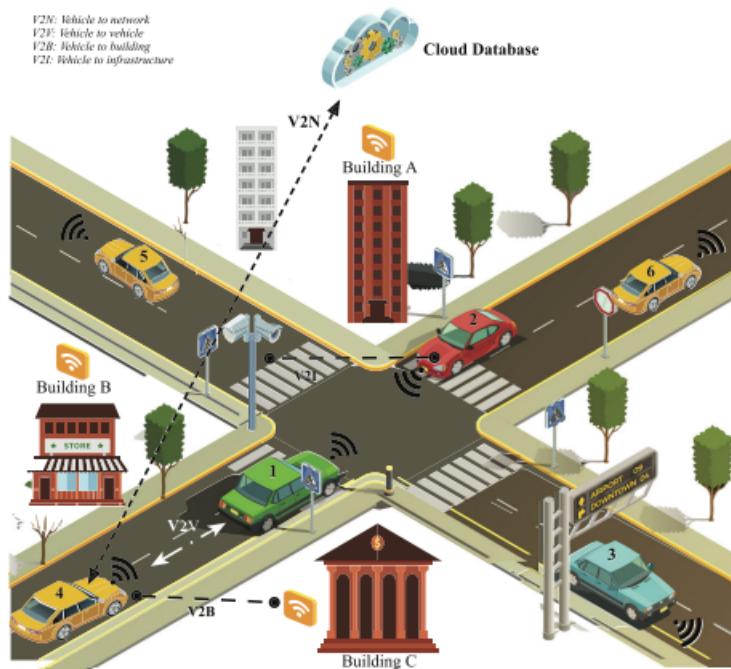


Figura 3.5: Interconectarea tehnologiilor descrise mai sus. [11]

## Capitolul 4. Analiză și fundamentare teoretică

### 4.1. Cazuri de utilizare

În această secțiune voi descrie câteva cazuri de utilizare a aplicației, observate în diagrama 4.1:

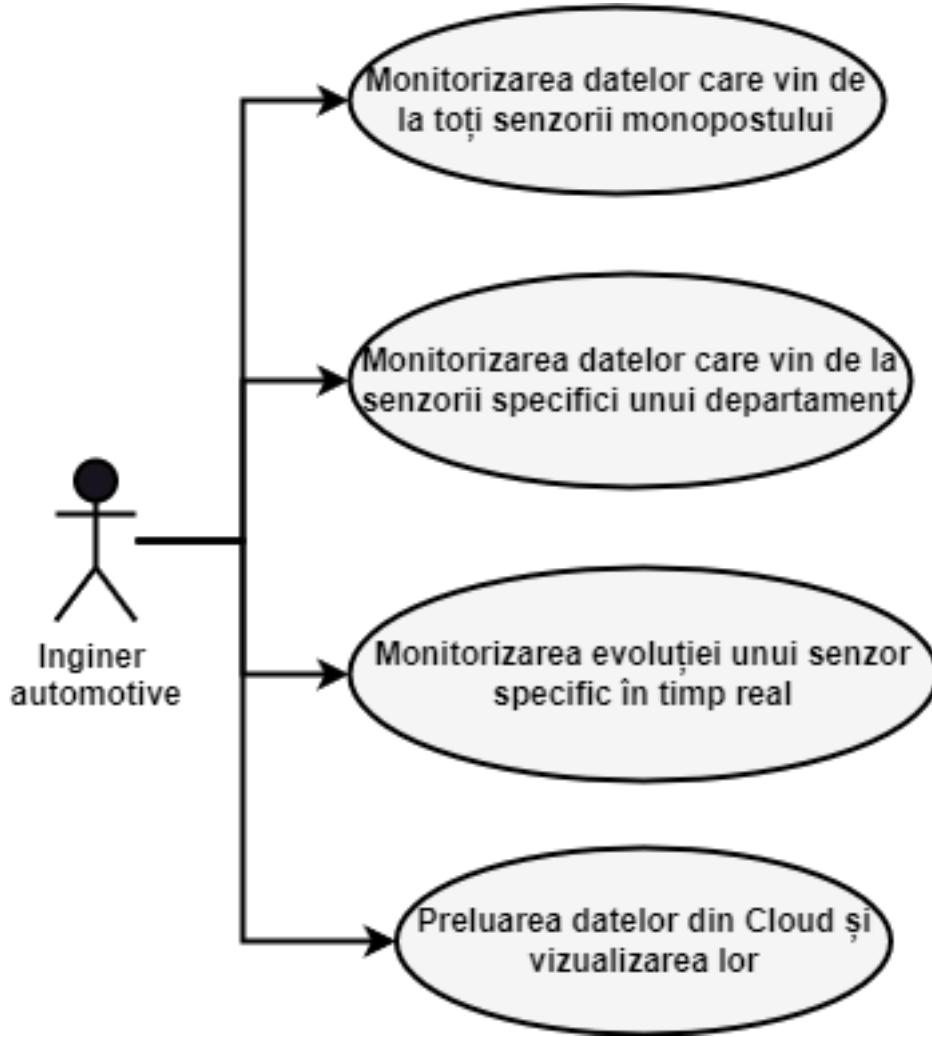


Figura 4.1: Diagrama de use case a sistemului.

#### 4.1.1. Cazul 1: Monitorizarea datelor care vin de la toți senzorii monopostului

- **Use case:** Monitorizarea datelor care vin de la toți senzorii monopostului.
- **Descriere:** Principalul scop al acestui use case este de a oferi inginerului automotive o privire de ansamblu asupra tuturor senzorilor și sistemelor din monopost. Aceste use case urmărește să obțină toate datele din monopost, să agregheze datele și să le afișeze sub formă diferitelor vizualizări inginerului de sistem, luând parte la toți senzorii și toate sistemele neținând cont de departamentul de care aparțin.

- **Actor principal:** Inginerul automotive.
- **Precondiții:** Transmițătorul, receptorul, și serverul să fie operaționale și conectate.
- **Flux de evenimente:**
  1. Pasul 1: Inginerul conectează la server receptorul prin portul USB.
  2. Pasul 2: Inginerul se asigură că monopostul este operațional.
  3. Pasul 3: Inginerul automotive rulează comanda de a pune în funcțiune serverul.
  4. Pasul 4: Monopostul trimite pachete la fiecare 3 secunde cu datele relevante ale tuturor sistemelor.
  5. Pasul 5: Receptorul interceptează aceste pachete și le decryptează și deschetează.
  6. Pasul 6: Serverul preia datele primite și le stochează în baza de date locală.
  7. Pasul 7: Aplicația preia datele din baza de date și le pune într-o formă vizuală.
- **Postcondiții:** Inginerul automotive vede toate datele senzorilor și sistemelor, indiferent de departamentul care le-a instalat, într-o formă vizuală care se actualizează la fiecare secundă cu date noi. Figura 4.2 arată cum poate arăta o astfel de interfață grafică pentru utilizator.



Figura 4.2: Date de la toți senzorii monopostului.

#### 4.1.2. Cazul 2: Monitorizarea datelor care vin de la senzorii specifici unui departament

- **Use case:** Monitorizarea datelor care vin de la senzorii specifici unui departament.
- **Descriere:** Principalul scop al acestui use case este de a oferi inginerului automotive o privire mai specifică asupra senzorilor și sistemelor unui departament (ex. mecanică, dinamica vehiculului). Acest use case urmărește să obțină datele din monopost care sunt relevante doar unui departament specific, agregându-le și afișându-le sub forma diferitelor vizualizări inginerului de sistem, luând parte la acesta doar sistemele și senzorii departamentului.
- **Actor principal:** Inginerul automotive.
- **Precondiții:** Transmițătorul, receptorul, și serverul să fie operaționale și conectate.
- **Flux de evenimente:**
  1. Pasul 1: Inginerul conectează la server receptorul prin portul USB.
  2. Pasul 2: Inginerul se asigură că monopostul este operațional.
  3. Pasul 3: Inginerul automotive rulează comanda de a pune în funcțiune serverul.
  4. Pasul 4: Monopostul trimite pachete la fiecare 3 secunde cu datele relevante ale tuturor sistemelor.
  5. Pasul 5: Receptorul interceptează aceste pachete și le decryptează și deschetează.

6. Pasul 6: Serverul preia datele primite și le stochează în baza de date locală.
  7. Pasul 7: Aplicația preia datele relevante din baza de date (doar cele aparținând de departamentul ales) și le pune într-o formă vizuală.
- **Postcondiții:** Inginerul automotive vede doar datele senzorilor și sistemelor departamentului care le-a instalat, într-o formă vizuală care se actualizează la fiecare secundă cu date noi. Figura 4.3 arată posibilitatea utilizatorului de a alege între interfețele pentru diferite departamente.

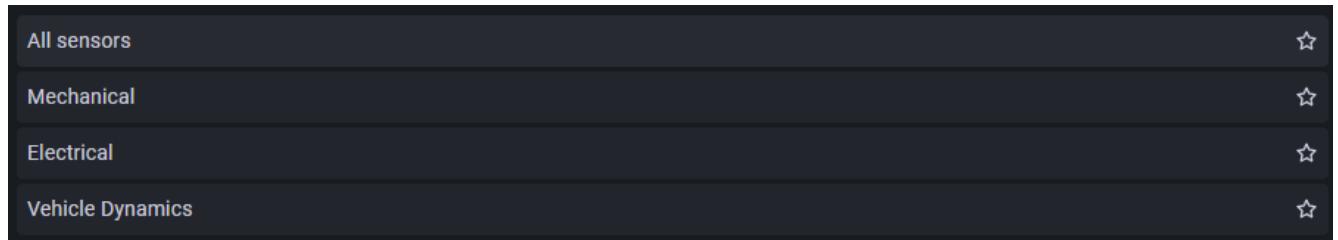


Figura 4.3: Selectarea datelor specifice unui departament.

#### 4.1.3. Cazul 3: Monitorizarea evoluției unui senzor specific în timp real

- **Use case:** Monitorizarea evoluției unui senzor specific în timp real.
- **Descriere:** Principalul scop al acestui use case este de a oferi inginerului automotive o actualizare evolutivă a unui senzor specific pentru a putea analiza comportamentul acestuia în timp real. Acestea urmărește să obțină datele exact a unui senzor anume, agregându-le și afișându-le sub forma unui grafic inginerului de sistem, luând parte la acesta doar un senzor.
- **Actor principal:** Inginerul automotive.
- **Precondiții:** Transmițătorul, receptorul, și serverul să fie operaționale și conectate.
- **Flux de evenimente:**
  1. Pasul 1: Inginerul conectează la server receptorul prin portul USB.
  2. Pasul 2: Inginerul se asigură că monopostul este operațional.
  3. Pasul 3: Inginerul automotive rulează comanda de a pune în funcțiune serverul.
  4. Pasul 4: Monopostul trimite pachete la fiecare 3 secunde cu datele relevante ale tuturor sistemelor.
  5. Pasul 5: Receptorul interceptează aceste pachete și le decryptează și deschidează.
  6. Pasul 6: Serverul preia datele primite și le stochează în baza de date locală.
  7. Pasul 7: Aplicația preia datele senzorului specific din baza de date și le afișează într-un grafic care se actualizează la o secundă.
- **Postcondiții:** Inginerul automotive vede doar datele unui senzor ales de el sub forma unui grafic actualizat în timp real. În figura 4.4 se observă cum se poate analiza evoluția unui senzor specific în timp.

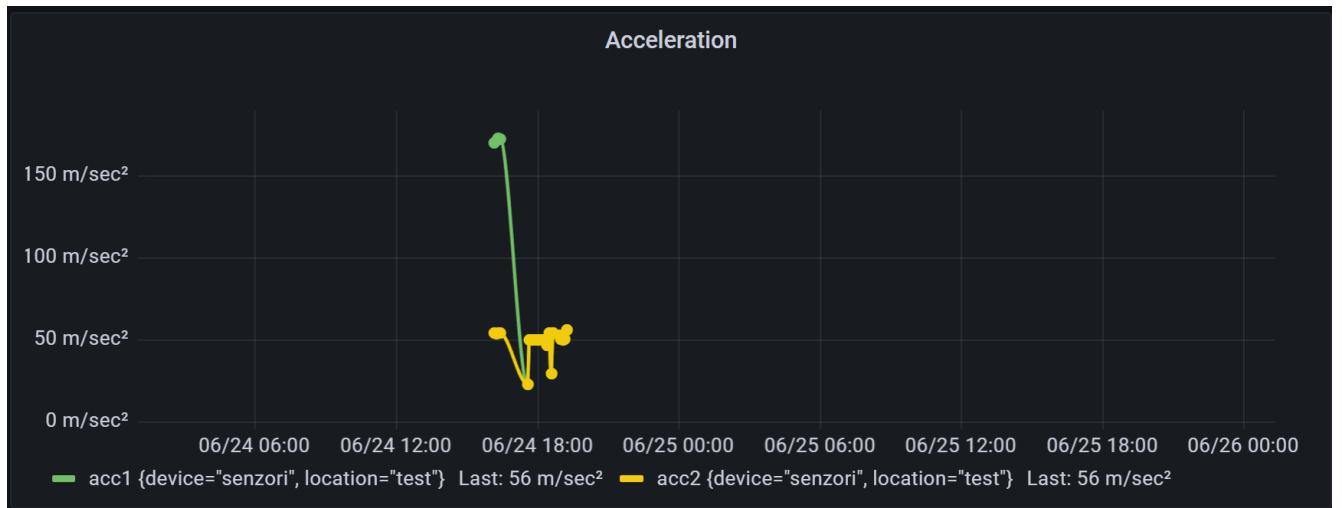


Figura 4.4: Selectarea unui senzor specific.

#### 4.1.4. Cazul 4: Preluarea datelor din Cloud și vizualizarea lor

- **Use case:** Preluarea datelor din Cloud și vizualizarea lor.
- **Descriere:** Principalul scop al acestui use case este de a oferi inginerului automotive un mod de a accesa datele și după ce monopostul nu mai este operațional pentru a le putea analiza. Acest use case urmărește să obțină o metodă persistentă de a salva datele și un mod intuitiv de a le accesa, prezentându-i-le sub o formă grafică, ușor de citit.
- **Actor principal:** Inginerul automotive.
- **Precondiții:** Transmițătorul, receptorul, și serverul să fie operaționale și conectate, și o conexiune la internet.
- **Flux de evenimente:**
  1. Pasul 1: Inginerul automotive rulează aplicația de colectare a datelor.
  2. Pasul 2: Inginerul descarcă datele de pe Cloud.
  3. Pasul 3: Inginerul alege datele care vrea să le vizualizeze.
  4. Pasul 4: Aplicația de vizualizare a datelor le preia și le afișează sub o formă grafică, secționată pe fiecare senzor în parte.
- **Postcondiții:** Inginerul automotive vede datele care au fost transmise în timpul rulării monopostului după acesta nu mai este operațional. Figura 4.5 prezintă accesarea unui senzor specific după ce monopostul nu mai este operațional.

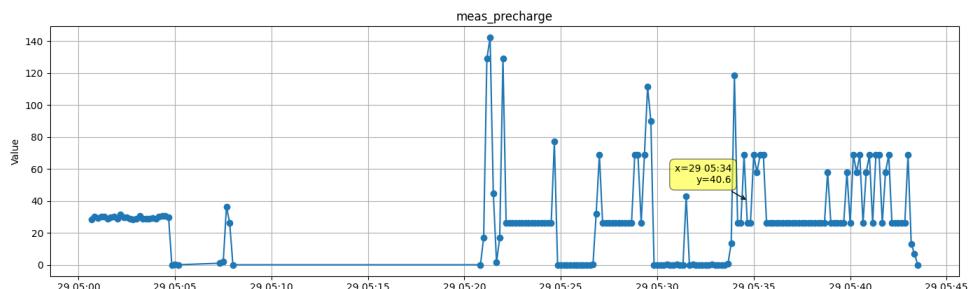


Figura 4.5: Vizualizarea datelor unui senzor după ce monopostul nu mai este operațional.

## 4.2. Componentele sistemului de telemetrie wireless ART TU E17

### 4.2.1. Protocole de comunicare

**LoRa si LoRaWAN** LoRa (Long Range) este o tehnologie de comunicație radio wireless, dezvoltată pentru a permite transmisii pe distanțe foarte lungi, cu un consum de energie extrem de redus. Este folosită în principal pentru Internet of Things (IoT), unde dispozitivele trebuie să transmită cantități mici de date pe distanțe mari, fără să consume multă baterie.

LoRa nu este un protocol complet, ci o tehnologie fizică (modulație) care operează pe benzi de frecvențe radio neautorizate, cum ar fi 868 MHz în Europa sau 915 MHz în America.

Nivelul fizic al LoRa poate fi folosit cu orice nivel de MAC, însă LoRaWAN este nivelul MAC propus pentru a menține o rețea simplă. Acesta folosește CSS (Chirp Spread Spectrum Modulation) cu Forward Error Correction (FEC), iar transmisile folosesc o bandă lată pentru a contracara interferența și pentru a gestiona decalajele de frecvență cauzate de folosirea cristalelor de slabă calitate. Astfel, LoRa poate ajunge la distanțe foarte mari. Proprietățile principale ale LoRa sunt raza lungă de acțiune (long range), robustețe mare, rezistență multipath, rezistență Doppler, consum foarte puțină putere și operează în frecvențe radio neautorizate. [12]

Un radio LoRa are patru parametrii de configurare: carrier frequency, spreading factor, lățimea de bandă și rata de codare. Selectia acestor parametri determină consumul de energie, distanța de transmisie și rezistența la zgomot.

**Carrier Frequency** este centrul frecvenței folosită pentru banda de transmisie, care poate avea diferiti parametri de configurare, alegându-se între 137 MHz spre 1020 MHz.

**Spreading Factor** este raportul dintre rata de simbol și rata de chirp. Un Spreading factor mai mare crește Signal-To-Noise Ratio, afectând astfel sensibilitatea și distanța, dar crește astfel și timpul în care packetul stă în aer. Un Spreading Factor poate să aibă o valoare între 6 și 12, iar SF6 are cea mai mare rată de transmisie, și este un caz special care necesită operații speciale, iar fiecare increment în SF înjumătățește rata de transmisie dar dublează durata transmisiei și consumul de energie.

**Lățimea de bandă** este o gamă de frecvențe în banda de transmisie, iar pe când o lățime de bandă oferă o rată mai mare de date dar o sensibilitate mai redusă (se introduce mai mult zgomot), o lățime de bandă mai subțire oferă o sensibilitate mai ridicată dar o rată de date mai scăzută.

**Coding Rate** este rata FEC folosită de modem-ul LoRa și oferă protecție împotriva impulsurilor de interferență. Pe când un CR ridicat oferă mai multă protecție, crește și timpul în care packetul stă în aer. Radio-urile cu CR diferite pot comunica între ele. [12]

**Printre diferențele între diferite moduri de transmisie includ** rețeaua celulară tradițională formată din GSM, 2G, 3G, 4G sunt foarte populare și răspândite. Acestea sunt printre rețelele de bază. Dar acestea au fost construite în mod tradițional pentru un debit ridicat de date și, prin urmare, acestea nu optimizează consumul de energie. Aceste tehnologii consumă prea multă putere și nu sunt o opțiune bună atunci când sunt rate mici de date să fie transmise mai rar. Costul total al proprietății este, de asemenea, foarte mare. Odată cu apariția tehnologiei 5G pe care mulți dintre furnizorii de telefonie mobilă îl întrerup serviciile 2G și fac ca dispozitivele IoT să nu mai funcționeze. Tehnologia LoRa, pe de altă parte, vine cu o putere mai mică rata de consum și este foarte potrivită pentru cantități mici de date care urmează să fie transmise pe distanțe lungi. [13]

**Limitările acestei tehnologii sunt** rata acesteia de a transmite date (până la 27

KBps), limitări în ceea ce privește Duty Cycles în rețelele LoRa limitează numărul de mesaje ce pot să fie trimise într-un anumit timp. [13]

**Duty cycle-ul** reprezintă proporția de timp în care un dispozitiv de comunicație radio are voie să transmită semnal într-un interval de timp definit. În contextul tehnologiei LoRa, care operează în benzi de frecvență neautorizate, această restricție este impusă de reglementările autorităților de comunicații din diverse regiuni, pentru a preveni congestionarea spectrului de radio.

De exemplu, în Europa, dispozitivele care operează în banda 868 MHz trebuie să respecte un duty cycle maxim de 1 la sută pe anumite sub-benzi. Astă înseamnă ca dacă eu transmit timp de o secundă, 99 de secunde nu mai am voie să trimit nici un mesaj. Această reglementare necesită rezolvări inteligente pentru a trimit un volum mai mare de date, una dintre acestea fiind **Frequency Hopping**.

Implementarea duty cycle-ului în LoRaWAN este gestionată automat de stack-ul LoRaWAN, care poate amâna sau fragmenta transmisia în funcție de încărcarea canalului și timpul disponibil, dar aceasta trebuie implementată manual în cazul în care nu se folosește **The Things Network**.

În figura 4.6 este o formă tabelară cu diferențele acestora, iar în figura 4.7 este prezentat principiul de funcționare al unei rețele LoRa: [14]

Technology	Wireless Communication	Range	Tx Power
Bluetooth	Short range	10 m	2.5 mW
Wifi	Short range	50 m	80 mW
3G/4G	Cellular	5 km	5000 mW
LoRa	LPWAN	<ul style="list-style-type: none"> <li>• 2-5 km (urban)</li> <li>• 5-15 km (rural)</li> <li>• &gt; 15 km (LOS)</li> </ul>	20 mW

Figura 4.6: Diferențe între LoRa și alte tehnologii de transmisie a datelor [14]

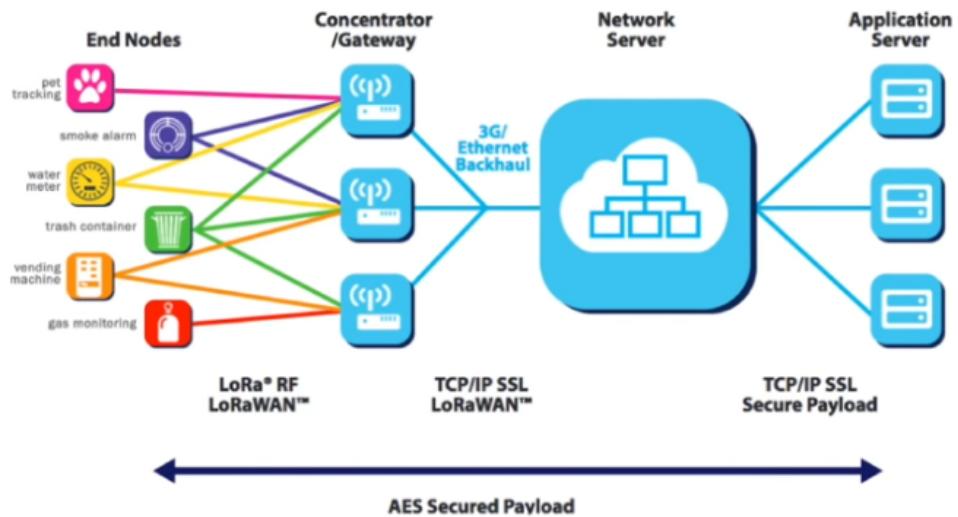


Figura 4.7: O rețea tipică LoRa [14]

În ceea ce privește decizia mea pentru această tehnologie, m-am uitat și la alte alternative:

- **Rețele Celulare.** 5G pare o opțiune bună, dar cu numărul de persoane de la o competiție tipică de Formula Student (aproximativ 2400), turnul de comunicații din apropiere s-ar putea aglomera și ar putea interfera cu comunicația noastră.
- **Sateliț.** Aceasta poate fi interfațat cu LoRa, are acoperire globală care este excesivă și nu e necesară pentru acest proiect, stația terestră se poate aglomera din cauza altor echipe care folosesc această tehnologie iar latența, interferențele și costul ridicat cauzează alte probleme, mai mari.
- **Microunde.** Rată de transfer ridicată, consum redus de energie, precizie mare și robustețe dar necesită line-of-sight, are rază de acțiune limitată, iar absorbția atmosferică reprezintă obstacole majore pentru proiectul nostru
- **WiMAX.** Distanță de până la 50 km, 70 Mbps, funcționează atât în condiții cu, cât și fără linie de vizibilitate dar costul ridicat elimină această opțiune.

Pe când LoRa oferă impulsuri de energie redusă de unde radio pentru a transmite date, nu interferează cu Wi-Fi, Bluetooth, durabilitate ridicată împotriva obstacolelor care blochează vizibilitatea directă dar pentru distanțe mari trebuie folosită o antenă omnidirectională sau o antenă direcțională planară și rate de transfer între 0.3 kbps și 27 kbps, deci rată de transmisie redusă din cauza limitării de duty cycle.

#### CAN Bus și ISO-TP

Controller Area Network (CAN Bus) este un sistem multi-master de trimitere a mesajelor care specifică rata maximă de transmisie de 1 Mbps. Spre deosebire de o rețea tradițională, cum ai fi USB sau Ethernet, CAN nu trimite blocuri mari de date care pleacă din punctul A în punctul B sub supravegherea unui master central. Într-o rețea CAN, multe mesaje scurte precum RPM sau temperatură sunt trimise către întreaga rețea, ceea ce asigură consistența datelor în fiecare nod al sistemului.

Acesta a fost dezvoltat inițial pentru industria automotive pentru a simplifica cablajul complicat a magistralelor bidirectionale.

CAN este un protocol de tip CSMA/CD+AMP, adică Carrier-Sense Multiple Access with Collision Detection and Arbitration on Message Priority (Acces Multiplex cu Detectarea Transportului și a Coliziunii și Arbitraj pe Baza Priorității Mesajului).

CSMA înseamnă că fiecare nod de pe magistrală trebuie să aștepte o perioadă de prestabilită de inactivitate înainte de a încerca să transmită un mesaj nou. [15]

CD+AMP înseamnă că eventualele coliziuni sunt rezolvate printr-un arbitraj la nivel de biți, bazat pe prioritatea preprogramată a fiecarui mesaj din câmpul identificator al mesajului. Identificatorul cu prioritate mai mare câștigă întotdeauna accesul la magistrală. Cu alte cuvinte, ultimul bit de tip logic-high din identificator continuă să fie transmis, deoarece are prioritatea cea mai mare.

Standardul ISO 11898:2003, cu identificatorul standard pe 11 biți, permite rate de semnalizare între 125 kbps și 1 Mbps. Ulterior, standardul a fost completat cu un identificator „extins” pe 29 de biți. Câmpul identificatorului standard pe 11 biți permite  $2^{11} = 2048$  identificatori de mesaje diferite, în timp ce identificatorul extins pe 29 de biți permite  $2^{29} = 537$  milioane de identificatori. [15]

Structura identificatorilor pentru CAN standard este prezentată în figura 4.8, în figura 4.9 este prezentată structura identificatorului pentru CAN extins iar în figura 4.10 este prezentat principiul de funcționare al protocolului CAN Bus.

SOF	<b>11-bit Identifier</b>	RTR	IDE	r0	DLC	0...8 Bytes Data	CRC	ACK	EOP	IIF
-----	--------------------------	-----	-----	----	-----	------------------	-----	-----	-----	-----

Figura 4.8: Identificator CAN Standard pentru 11 biți [15]

SOF	<b>11-bit Identifier</b>	RRR	IDE	<b>18-bit Identifier</b>	RTR	r1	r0	DLC	0...8 Bytes Data	CRC	ACK	EOP	IIF
-----	--------------------------	-----	-----	--------------------------	-----	----	----	-----	------------------	-----	-----	-----	-----

Figura 4.9: Identificator CAN Extins pentru 29 biți [15]

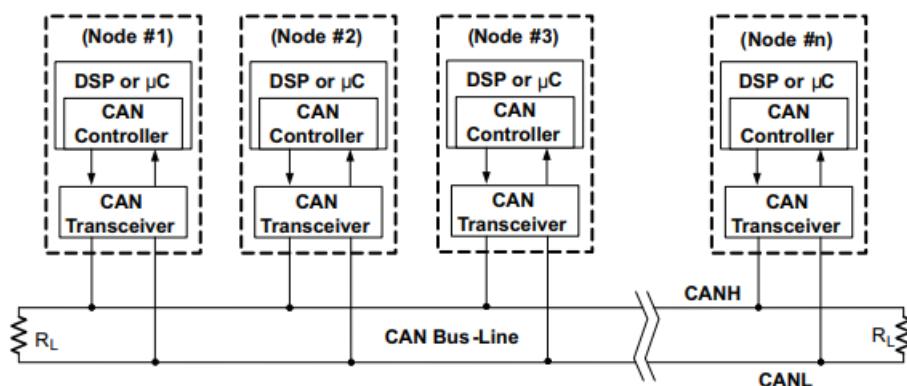


Figura 4.10: Principiul de funcționare al magistralei CAN [15]

**ISO-TP 15765-2**, cunoscut și sub numele de ISO-TP, este un protocol de transport al datelor definit special pentru comunicația pentru diagnosticare pe magistrala CAN. Este utilizat pe scară largă în industria auto ca protocol pentru serviciile de diagnosticare legate de emisii (ISO 15031-5), pe lângă multe altele.

ISO-TP poate fi utilizat atât pe rețele bazate pe CAN CC (CAN Clasic), cât și pe CAN FD (CAN Flexible Datarate).

În forma sa cea mai simplă, ISO-TP se bazează pe două tipuri de moduri de adresare pentru nodurile conectate la aceeași rețea:

- Adresarea fizică este implementată prin două adrese specifice de nod și este folosită pentru comunicarea de tip 1-la-1.
- Adresarea funcțională este implementată printr-o adresă specifică de nod și este folosită pentru comunicarea de tip 1-la-N.

Când se transmit date folosind protocolul ISO-TP, este discutabil dacă merită să fie folosit în unele aplicații luând în considerare costul suplimentar generat de protocol și, optional, adresarea extinsă. În primul caz, datele sunt transmise deodată folosind un Single Frame. În al doilea caz, ISO-TP definește un protocol cu mai multe cadre, în care expeditorul oferă (prin intermediul First Frame) lungimea datelor care trebuie transmise și solicită un cadru de Flow Control, care oferă dimensiunea maximă a unui bloc macro de date (blocksize) și timpul minim între mesajele CAN individuale care compun acel bloc (stmin). Odată ce aceste informații au fost primite, expeditorul începe să trimită cadre conținând fragmente ale sarcinii utile (Consecutive Frames), oprindu-se după fiecare bloc de dimensiune blocksize pentru a aștepta confirmarea de la receptor, care ar trebui să

trimite un nou cadru FC pentru a informa expeditorul despre abilitarea lui de a primi mai multe date.

Motivul pentru care am ales să folosesc CAN Bus împreună cu ISO-TP este pentru rezistența sa împotriva zgromotelor și integrarea sa foarte bine cu arhitectura existentă relevantă a ART TU E17. ISO-TP îmi permite să trimit mesaje mult mai mari decât CAN Standard, motiv pentru care eu pot trimite toate datele necesare într-un mesaj, care îl pot trimite mai des. De asemenea, acesta este și un standard în automotive, iar planul echipei ART TU în sezoanele viitoare este de a schimba tot protocolul de comunicare în CAN, și mă bucur foarte tare că am putut ajuta la acest proces, iar componentele pentru acesta sunt foarte ieftine, un alt motiv care m-a facut să aleg CAN Bus. [16]

### **AES-128-CTR**

AES (Advanced Encryption Standard) este un algoritm simetric de criptare pentru protejarea datelor, fiind standardul de criptare stabilit de Institutul Național de Standarde și Tehnologie al Statelor Unite (NIST) în anul 2001. AES criptează și decriptează blocuri de date de 128 biți folosind chei de 128, 192 sau 256 biți, oferind o securitate ridicată și performanță eficientă, fiind folosit pe scară largă în aplicații guvernamentale, comerciale și personale. [17]

Modul Counter (CTR) este un mod de confidențialitate care constă în aplicarea forward cipher asupra unui set de blocuri de intrare, numite contoare, pentru a produce o secvență de blocuri de ieșire care sunt combinate prin operația OR exclusiv cu textul clar pentru a produce textul cifrat și invers. Secvența de contoare trebuie să aibă proprietatea ca fiecare bloc din secvență să fie diferit de toate celelalte blocuri. Această condiție nu este limitată la un singur mesaj: pentru toate mesajele criptate cu cheia dată, toate contoarele trebuie să fie distințe.

În criptarea CTR, funcția forward cipher este aplicată fiecărui bloc de contor, iar blocurile de ieșire rezultate sunt combinate prin OR exclusiv cu blocurile corespunzătoare de text clar pentru a produce blocurile de text cifrat. Pentru ultimul bloc, care poate fi incomplet, cei mai semnificativi biți sunt folosiți pentru OR exclusiv iar restul biților ai ultimului bloc de ieșire sunt eliminați.

În decriptarea CTR, funcția forward cipher este aplicată fiecărui bloc de contor, iar blocurile de ieșire rezultate sunt combinate prin OR exclusiv cu blocurile corespunzătoare de text cifrat pentru a recupera blocurile de text clar.

Atât în criptarea CTR, cât și în decriptarea CTR, funcțiile cîrfului înainte pot fi executate în paralel; similar, blocul de text clar care corespunde oricărui bloc de text cifrat poate fi recuperat independent de celelalte blocuri de text clar dacă blocul de contor corespunzător poate fi determinat. Mai mult, funcțiile cîrfului înainte pot fi aplicate contoarelor înainte ca datele de text clar sau text cifrat să fie disponibile. Modul în care funcționează AES-128 CTR este prezentat în figura 4.11:

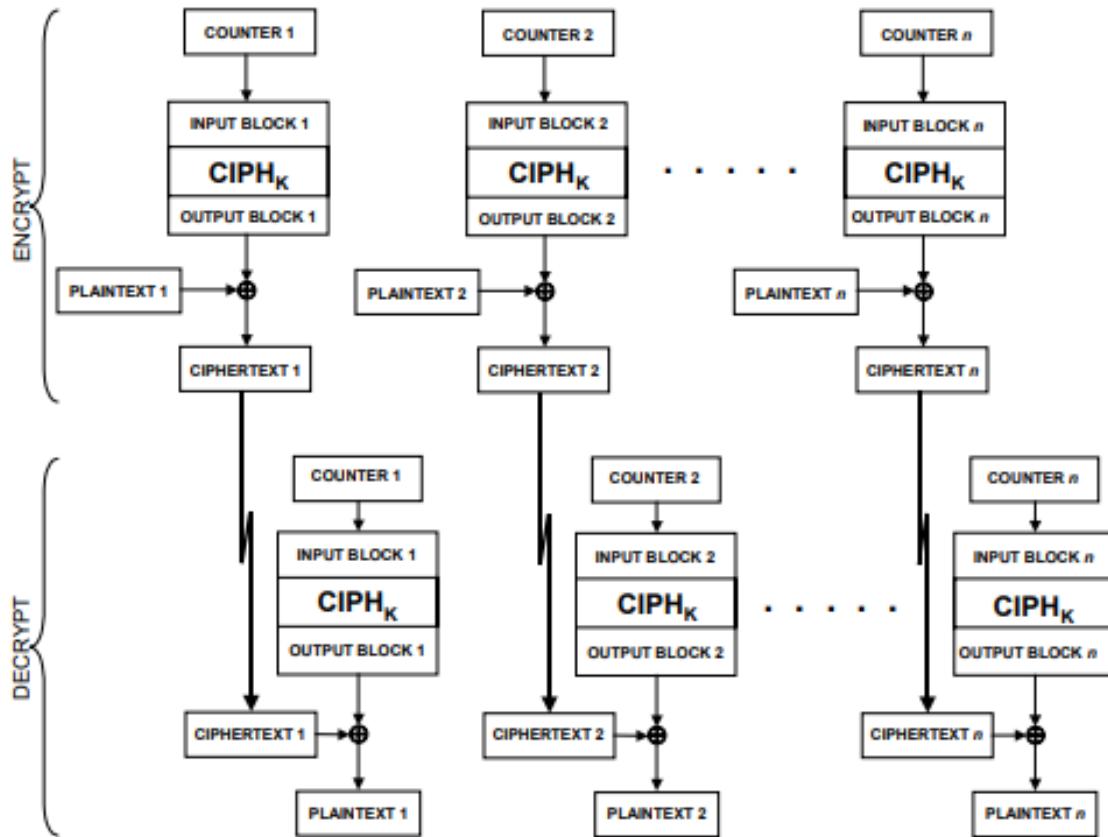


Figura 4.11: Modul de funcționare al AES-128 CTR [18]

Motivul pentru care am ales să folosesc metoda de criptare AES-128-CTR este pentru că acesta nu are nevoie de o mărime fixă a mesajului pentru a-l putea cripta, folosește resurse puține, ideal pentru aplicațiile în hardware, și are foarte mult support.

#### 4.2.2. Componete software

##### InfluxDB

InfluxDB este o bază de date specializată pentru stocarea și gestionarea datelor de tip time series, fiind colectate secvențial în funcție de timp, precum date de telemetrie, metrici de performanță, date financiare, sau date de monitorizare a infrastructurii. Este optimizat pentru scenarii unde este necesar timp de răspuns aproape instant, având răspunsuri la query-uri de sub 10 ms. [19]

InfluxDB este optimizată pentru:

- Insertii rapide și volum mare de date.
- Stocarea eficientă a datelor marcate cu timestamp.
- Interrogări puternice și specifice, oferind un limbaj dedicat (Flux) pentru a extrage, transforma și analiza.
- Funcții încorporate pentru date temporale precum agregări pe intervale de timp, downsampling, interpolări, și alerte.
- Integrare ușoară cu sisteme IoT, platforme de monitorizare și instrumente de vizualizare precum Grafana. [19]

Acesta foloseste Line Protocol, fiind un format bazat pe text pentru inserări, unde

fiecare linie conține câmpurile:

1. Measurement: Conține numele seriei sau a tabelului.
2. Tag set: perechi cheie-valoare care sunt separate prin virgulă.
3. Field set: perechi cheie-valoare pentru date, fiind separate prin virgulă.
4. Timestamp: timp UNIX care e măsurat în nanosecunde, folosind timpul serverului.

În figura 4.12 este prezentată structura unei comenzi din Line Protocol:

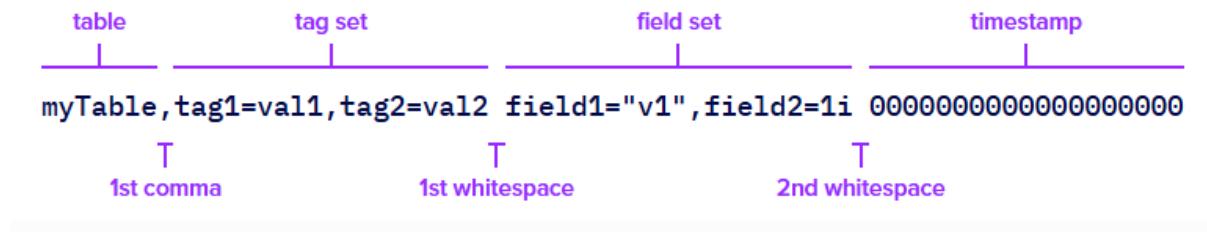


Figura 4.12: Structura unei comenzi de Line Protocol [19]

Am ales să folosesc InfluxDB datorită capacitatea sale native de a gestiona date marcate cu timestamp, ceea ce permite stocarea precisă și analiza eficientă a informațiilor colectate în timp real de la vehiculul electric. Performanța în scrierea rapidă a unui volum mare de date face ca InfluxDB să fie potrivită pentru aplicația de telemetrie unde datele sunt generate continuu și frecvent. Funcțiile integrate de agregare și downsampling ajută la reducerea volumului de date fără pierderea informațiilor esențiale.

### Grafana

Grafana este o platformă open-source de vizualizare și monitorizare a datelor, concepută pentru a transforma datele complexe în grafice și dashboard-uri interactive, ușor de interpretat. Aceasta suportă o gamă largă de surse de date, inclusiv baze de date time-series precum InfluxDB, Prometheus, Graphite, Elasticsearch, dar și baze SQL. Grafana oferă suport pentru filtrarea și ajustarea în timp real a datelor afișate, fără a fi nevoie de reconstrucția dashboard-ului.

Variabilele template permit creearea de dashboard-uri reutilizabile pentru mai multe scopuri, valorile nefiind codate direct în aceste template-uri, ci fiind actualizate în timp real în funcție de sursele de date. Aceasta oferă posibilitatea de a crea dashboard-uri personalizate și adaptabile la nevoile specifiche a unei echipe.

Am ales să folosesc Grafana pentru că este o platformă flexibilă pentru vizualizarea datelor, oferind numeroase moduri de afișare, cum ar fi grafice, diagrame, tabele și alerte personalizate, ceea ce permite interpretarea clară a informațiilor colectate. Integrarea sa nativă cu InfluxDB facilitează preluarea rapidă a datelor temporale, oferind actualizări în timp real și o experiență fluidă în monitorizarea sistemelor complexe, iar variabilele template mă ajută la adaptarea aplicației de vizualizare după nevoile exacte ale echipei.

### Firebase

Firebase este o platformă dezvoltată de Google care oferă o suite completă de servicii backend pentru dezvoltarea rapidă a aplicațiilor web. Printre componentele sale principale se numără baze de date NoSQL în timp real (Realtime Database și Firestore), stocarea fișierelor, hostare, printre multe altele. Firebase facilitează sincronizarea datelor în timp real între clienți și server, ceea ce îl face ideal pentru aplicații care necesită actualizări instantanee și colaborare în timp real.

Platforma este construită pentru a se integra ușor cu diferite limbi și framework-uri, oferind SDK-uri pentru Android, iOS, Web, C++ și Unity, precum și API-uri REST,

ceea ce permite dezvoltarea rapidă și flexibilă a aplicațiilor. Firebase asigură și scalabilitate automată, gestionând infrastructura backend fără ca dezvoltatorii să se preocupe de administrarea serverelor, având și suport nativ pentru Python. [21]

Motivul pentru care folosesc Firebase este planul gratuit generos, ceea ce reduce costurile inițiale și facilitează testarea rapidă. În plus, Firebase beneficiază de o documentație extensivă și bine structurată, cu numeroase tutoriale și exemple, ceea ce accelerează procesul de dezvoltare, reducând timpul necesar pentru implementarea funcționalităților complexe.

### Docker

Docker este o platformă de containerizare care permite dezvoltatorilor să împacheteze aplicațiile și toate dependențele acestora într-un mediu izolat, referit ca și un container. Acest container rulează în mod consistent pe orice sistem care suportă Docker, indiferent de diferențele hardware sau software ale gazdei. Prin utilizarea containerelor, Docker asigură o izolare completă a mediului de execuție, eliminând problemele cauzate de incompatibilități între versiuni de librării, sisteme de operare sau configurații hardware.

Un avantaj major al Docker este că facilitează testarea rapidă și reproductibilă a mediilor de dezvoltare și producție, oferind posibilitatea de a rula exact același cod în aceeași condiții, indiferent de sistemul utilizatorului. Astfel, pot evita problema de „funcționează pe calculatorul meu” și poate garanta consistența rezultatelor.

Docker folosește namespace-uri pentru a izola resursele pentru un set de procese, fiind o funcționalitate a nucleului de Linux. Acestea oferă o vedere distinctă asupra resurselor sistemului, creând un mediu controlat pentru procese. Izolarea aceasta este esențială pentru containere, garantând separarea acestora de celălalte containere și de sistemul gazdă.

Există mai multe tipuri de namespace-uri:

- PID Namespace (Process ID) gestionează identificatorii de proces, asigurând că fiecare container primește propriul namespace PID, ceea ce înseamnă că procesele dintr-un container nu pot vedea sau interacționa cu procesele din alt container sau de pe sistemul gazdă.
- Network Namespace oferă un network stack propriu fiecărui container. Fiecare container are propriile interfețe de rețea, tabele de rutare și reguli de firewall, permitând configurații de rețea independente.
- Mount Namespace controlează punctele de montare ale sistemului de fișiere pentru un container. Astfel, fiecare container poate avea propria vedere asupra sistemului de fișiere, inclusiv directoare și fișiere diferite.
- User Namespace permite separarea ID-urilor de utilizator și grup. Aceasta înseamnă că un container poate rula cu privilegii diferite față de sistemul gazdă.
- IPC Namespace (Inter-Process Communication): gestionează mecanismele de comunicare între procese. Se asigură că procesele dintr-un container nu pot comunica sau interferă cu cele din alte containere. [22]

Am ales Docker pentru că se potrivește cu paradigma plug and play care am vrut să o implementez, minimzând diferențele de hardware și software dintre utilizatorii cu configurații diferite de laptopuri sau stații de lucru.

#### 4.2.3. Componete electrice

##### **Arduino MKR WAN 1310**

Placa Arduino MKR WAN 1310 oferă o soluție practică și eficientă din punctul de

vedere al costurilor pentru a adăuga conectivitate LoRa proiectelor care necesită consum redus de energie, precum acest proiect.

Arhitectura sa cu consum scăzut de energie a îmbunătățit considerabil autonomia bateriei pe MKR WAN 1310. Atunci când este configurată corect, placa poate consuma 104 µA. Se poate alimenta placa prin portul USB (5V), funcționând cu sau fără baterii.

Data logging și alte funcții OTA (Over The Air) sunt posibile datorită memoriei Flash de 2MB integrată pe placă. Această funcționalitate permite transferul fișierelor de configurare din infrastructură pe placă, dar și crearea propriilor comenzi scriptate sau stocarea locală a datelor, care pot fi trimise ulterior, atunci când conectivitatea este optimă. [23]

Figura 4.13 arată cum arată o placă Arduino MKR WAN 1310, pe față și pe verso:

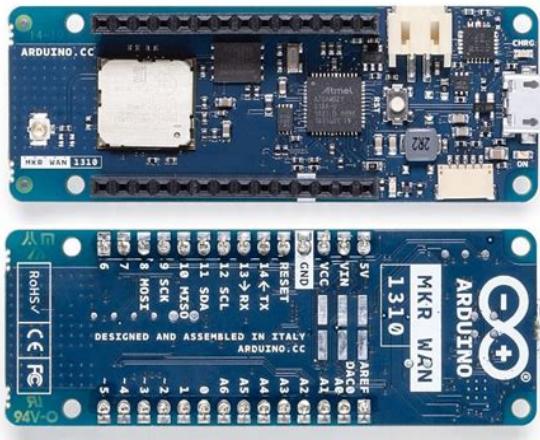


Figura 4.13: Plăcuța Arduino MKR WAN 1310, față-verso [23]

Această placă folosește procesorul **Atmel SAMD21** care este o serie de microcontrolere cu consum redus de energie, bazate pe procesorul ARM Cortex M0+ pe 32 de biți, și sunt disponibile în variante de la 32 la 64 de pini, cu până la 256 KB de memorie Flash și 32 KB de SRAM. Aceste microcontrolere funcționează la o frecvență maximă de 48 MHz iar toate modelele din serie includ periferice inteligente și flexibile, precum și un Event System pentru comunicare între periferice fără implicarea CPU-ului. Ele oferă și suport pentru interfețe tactile capacitive: butoane, slider și rotite.

Functile esențiale ale acestuia sunt: [24]

- Memorie Flash programabilă in-system
  - Direct Memory Access Controller cu 12 canale
  - Event System cu 12 canale
  - Controler de întreruperi programabil
  - Până la 52 de pini I/O programabili
  - Ceas și calendar RTC pe 32 de biți
  - 5 temporizatoare/numărătoare (TC) pe 16 biți
  - 4 temporizatoare/numărătoare pentru control (TCC) pe 24 de biți
  - USB 2.0 Full-Speed integrat (host + device)
  - Până la 6 module SERCOM configurabile ca USART, UART, SPI, I2C (până la 3.4 MHz), SMBus, PMBus sau LIN slave
  - I2S pe 2 canale

- ADC pe 12 biți, până la 20 de canale, 350 ksps, cu amplificare programabilă, oversampling și decimare (până la rezoluție de 16 biți)
- DAC pe 10 biți, 350 ksps
- Până la 4 comparatoare analogice cu mod „Window”
- Peripheral Touch Controller (PTC) pentru până la 256 de butoane, slidere, roți și detectie de proximitate
- Watchdog Timer (WDT) programabil
- Detector brown-out și reset la pornire
- Interfață de depanare și programare Serial Wire Debug (SWD) cu doi pini

Figura 4.14 prezintă imaginea unui controller SAMD21:

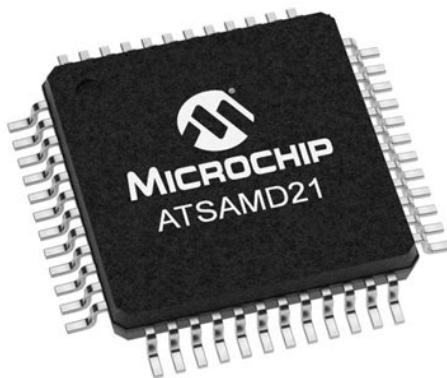


Figura 4.14: Integratul SAMD21 [24]

Motivul principal pentru care am ales arhitectura Arduino pentru proiectul meu este costul său redus, fiind un factor decizional decizional și familiaritatea în rândul inginerilor este un plus mare.

#### **MCP 2515**

Microchip Technology MCP2515 este un controler CAN (Controller Area Network) independent, care implementează CAN versiunea 2.0B. Este capabil să transmită și să primească cadre standard, cât și extinse, de date și de control de la distanță. MCP2515 dispune de două măști de acceptare și șase filtre de acceptare, care sunt folosite pentru a filtra mesajele nedorite, reducând astfel sarcina procesorului principal (MCU). MCP2515 comunică cu microcontrolerele (MCU) printr-o interfață SPI (Serial Peripheral Interface) standard din industrie. [25]

Funcțiile esențiale ale acestuia sunt: [25]

- Implementează CAN V2.0B la 1 Mb/s
- Buffere de receptie, măști și filtre, două măști de 29 de biți, șase filtre de 29 de biți
- Filtrarea octetilor de date, primii doi octeți din cadru
- Trei buffere de transmisie cu funcții de prioritizare și anulare
- Interfață SPI de mare viteză (10 MHz)
- Mod One-Shot: asigură că transmiterea mesajului este încercată o singură dată
- Pin de ieșire a ceasului (Clock Out) cu divizor programabil
- Semnal Start-of-Frame (SOF) disponibil pentru monitorizare
- Pin de ieșire pentru întreruperi cu activare selectabilă
- Pini de ieșire Buffer Full configurabili
- Pini de intrare Request-to-Send (RTS) configurabili individual
- Tehnologie CMOS cu consum redus

- Intervale de temperatură acceptate, industrial: -40°C până la +85°C, Extins: -40°C până la +125°C
  - Certificat AEC-Q100 (standard de calitate pentru industria auto)

Figura 4.15 prezintă modulul de MCP2515 care este folosit în proiect:



Figura 4.15: Modulul MCP2515 [25]

## **868/915 MHz ISM Adhesive Mount Flexible Polymer Embedded Antenna (2JF0415P)**

Este o antenă ISM Dual Band foarte compactă, flexibilă și cu eficiență ridicată cu o grosime de doar 0,2 mm. Modelul 2JF0415P este o antenă ultra-subțire, eficientă și flexibilă, concepută special pentru aplicații NB-IoT în care spațiul disponibil în dispozitiv este foarte limitat. Datorită designului său omnidirectional, antena oferă o acoperire de 360°, fiind ideală pentru toate implementările care necesită o frecvență dedicată.

Utilizează frecvența ISM de 868 MHz pentru standardele europene și 915 MHz pentru piața din SUA. Această antenă flexibilă integrabilă este ideală pentru aplicații precum: comunicații M2M (machine-to-machine), contoare inteligente (smart metering), sisteme de acces de la distanță (remote keyless), senzori subacvatici, monitorizarea activității fizice. [26] Figura 4.16 arată antena care este folosită în proiect:



Figura 4.16: Antena 2JF0415P. [26]

#### 4.2.4. Tehnici de programare

##### **Fog Computing**

Din perspectiva Cisco, fog computing este considerat o extensie a paradigmii cloud computing, de la centrul rețelei către marginea acesteia. Este o platformă puternic virtualizată care oferă servicii de calcul, stocare și rețelistică între dispozitivele finale și serverele tradiționale din cloud. [27]

Pe de altă parte, conform unei alte abordări, fog computing este definit ca „un scenariu în care un număr foarte mare de dispozitive eterogene (fără fir și uneori autonome), ubique și decentralizate, comunică și, potențial, cooperează între ele și cu rețeaua pentru a îndeplini sarcini de stocare și procesare, fără intervenția unor terți. Aceste sarcini pot sprijini funcții de bază ale rețelei sau pot susține noi servicii și aplicații care rulează într-un mediu izolat (sandbox). Utilizatorii care pun la dispoziție o parte din resursele dispozitivelor lor pentru găzduirea acestor servicii primesc stimulente pentru acest lucru.”

Deși această definiție este încă dezbatută, este necesară o definiție clară pentru a distingere fog computing-ul de alte tehnologii înrudite, deoarece oricare dintre acestea poate părea similară la nivel de bază. [27]

Motivația principală pentru folosirea fog computing în acest proiect este legată de necesitatea unui răspuns rapid și continuu într-o aplicație în timp real. Prin procesarea locală a datelor, fog computing minimizează numărul de cereri trimise către serverele cloud, reducând astfel latența și traficul pe rețea și îmbunătățește timpul de actualizare a aplicației. Acest lucru este esențial în cazul în care timpul de reacție trebuie să fie foarte scurt, iar conexiunea la internet poate fi intermitentă sau cu viteză redusă. Astfel, datele critice pot fi analizate și acționate imediat, fără a aștepta un răspuns din cloud, asigurând performanța și stabilitatea necesară în condiții de funcționare în timp real.

##### **Frequency Hopping**

Spectrul extins (Spread Spectrum) este o tehnică de modulație folosită în comunicațiile militare. În acest tip de modulație, lățimea de bandă a semnalului transmis este mult mai mare decât lățimea de bandă a mesajului original și este determinată de codul de extindere care este un semnal digital independent de mesaj, dar cunoscut atât de emițător, cât și de receptor. Există două metode frecvent utilizate pentru realizarea extinderii spectrului, Frequency Hopping (FH-SS) și Direct Sequence (DS-SS). Modulațiile cu spectru extins oferă: [28]

- Protecție ridicată împotriva interferențelor,
- Dificultate în interceptarea și spionajul comunicațiilor,
- Posibilitatea de multiplexare prin cod (toți utilizatorii pot comunica simultan pe aceeași bandă de frecvență),
- Adresare selectivă a utilizatorilor individuali (Todorović, 2021).

În tehnica Frequency Hopping Spread Spectrum, emițătorul și receptorul își schimbă frecvența în funcție de o secvență prestabilită, cunoscută doar de ei. Această schimbare are loc în cadrul benzii de transmisie, fiind controlată de un generator de secvență de cod. Tehnica poate fi folosită pentru a transmite mesaje atât în format analogic, cât și digital. [28]

FH-SS este intens utilizată în transmisii de date foarte securizate, atât în comunicațiile militare, cât și în cele comerciale, în radiouri HF, VHF, UHF, control și transmisie de date pentru drone (UAV) precum și telefoane mobile GSM, tehnologie Bluetooth

Avantajele FH-SS: [28]

- Rezistență ridicată la bruijaj, intenționat sau nu.
- Performanță bună în condiții de semnale apropriate (near-far).

- Rezistență eficientă la: interceptare, localizarea sursei (direction finding), blocaj electronic (jamming).

Acestea fiind spuse, FH-SS este considerată o tehnologie potrivită pentru aplicații în medii ostile și pentru comunicații sigure, fiind o alegere potrivită pentru proiectul meu, existând multe interferențe electromagnetice de la motoare și invertoare iar folosind FH-SS evit riscul ca o altă echipă să intercepteze datele monopostului de telemetrie.

### 4.3. Solutii existente la alte echipe de Formula Student

În ceea ce privește inspirația de la alte echipe de Formula Student, m-am uitat la TUL Racing din Cehia și AMZ Motorsport din Elveția.

În ceea ce privește TUL Racing, sistemul lor de telemetrie wireless are o distanță limitată din cauza folosirii tehnologiei Wireless, care este varianta lor simplă, dar prin folosirea ESP-32 ca un relay-node aceștia pot extinde distanța până la 1 kilometru, dar vine la costul transferului de date, care ajunge la 250kb/s. Această soluție mi-a dat idei bune, arătându-mi că viziunea pentru proiectul meu este posibilă, dar folosirea de hardware suplimentar se contrazicea cu parada "plug-and-play" propusă, fiind incomod ca un membru al echipei să instaleze mai multe puncte de releu pentru date, creând mai multe puncte de eșec, astă însemnând un sistem mai puțin reliable la distanțe mari. [29]

Pentru AMZ Motorsport, ei folosesc WLAN ca mediu de transmisie, care este o soluție bună în ceea ce privește throughput-ul de date și documentația pentru această stivă, dar pentru a putea integra această tehnologie în monopostul lor a fost construit un PCB special pentru asta, iar pentru proiectul prezentat, nu este o soluție optimă în ceea ce privește timpul care ar presupune dezvoltarea acestei plăcuțe și bugetul alocat pentru acesta. [30]

### 4.4. Arhitectura monopostului ART TU E17

Pentru a înțelege întâi cum funcționează sistemul de telemetrie, trebuie întâi să ne uităm la tot sistemul electronic al monopostului ART TU E17

**Electronic Control Unit (ECU)** este centrul de control vehiculului electric și este responsabil pentru colectarea și transmiterea tuturor datelor de la senzori, temperaturi, dashboard și multe alte informații importante care mențin vehiculul eficient și sigur în funcționare. Dezvoltat intern, acesta se bazează pe placa Teensy 4.1 și are integrate multe protocoale de comunicație (SPI, UART, RS232, CAN Bus, I2C) pentru toate tipurile de senzori. ECU controlează vehiculul cu ajutorul codului Arduino, de la pragurile la care anumite sisteme se activează până la controlul motorului.

**Tractive System Active Light (TSAL Main)** semnalizează, prin utilizarea mai multor LED-uri, starea sistemului de voltaj mare, cu lumină verde și roșie. Verde indică faptul că tensiunea din vehicul este sub 60 VDC și contactorii sunt toți deconectați, în principal fiecare releu din mașină este deschis, ceea ce înseamnă că nu există pericol din partea celor 200 VDC ai bateriei vehiculului. Roșu indică faptul că tensiunea depășește 60 VDC și vehiculul nu este sigur pentru condus și manipulare, în special dacă este detectată o defecțiune. Toată circuitele TSAL trebuie să fie electronice cablate (hard-wired), fără control software, iar circuitele pentru roșu și verde trebuie să funcționeze independent.

**TSAL Indicators** servesc aceeași funcție ca placa TSAL Main, dar extind semnalul către cockpit și main hoop (unde este situat volanul), fiind vizibile pentru șofer în lumina directă a soarelui. Aceleași reguli se aplică și acestor plăci TSAL, fără componente software, doar componente cablate (hard-wired). La fel ca TSAL Main, toate semnalele

trebuie să fie Semnale Critice de Siguranță (SCS), între 0.5V și 4.5V. Indicatoarele de pe dashboard arată șoferului starea dispozitivului de măsurare a izolației (IMD), a sistemului de management al acumulatorului (AMS) și starea verde sau roșie a TSAL-ului.

**TSAL Measure** separă sistemul de voltaj mic de sistemul de voltaj mare printr-un spațiu de 6 mm, conectat printr-un optocuplător. Detectează dacă o tensiune mai mare de 60 VDC este prezentă la punctele de măsurare și dacă contactorii din secvența de pornire sunt închisi. Aceleași reguli ca pentru TSAL Main se aplică: doar componente cablate (hard-wired) și toate semnalele trebuie să fie SCS.

**Telemetria Wireless** trebuie să comunice datele vehiculului în timpul funcționării (viteză, temperaturi, starea de încărcare, starea contactorilor) către o echipă la sol, folosind un sistem wireless către un server. De asemenea, are rolul de a stoca măsurările anterioare pentru analiză prin grafice.

În figura 4.17 este ilustrată arhitectura sistemului de voltaj mic al monopostului ART TU E17 și cum fiecare componentă interacționează una cu cealaltă.

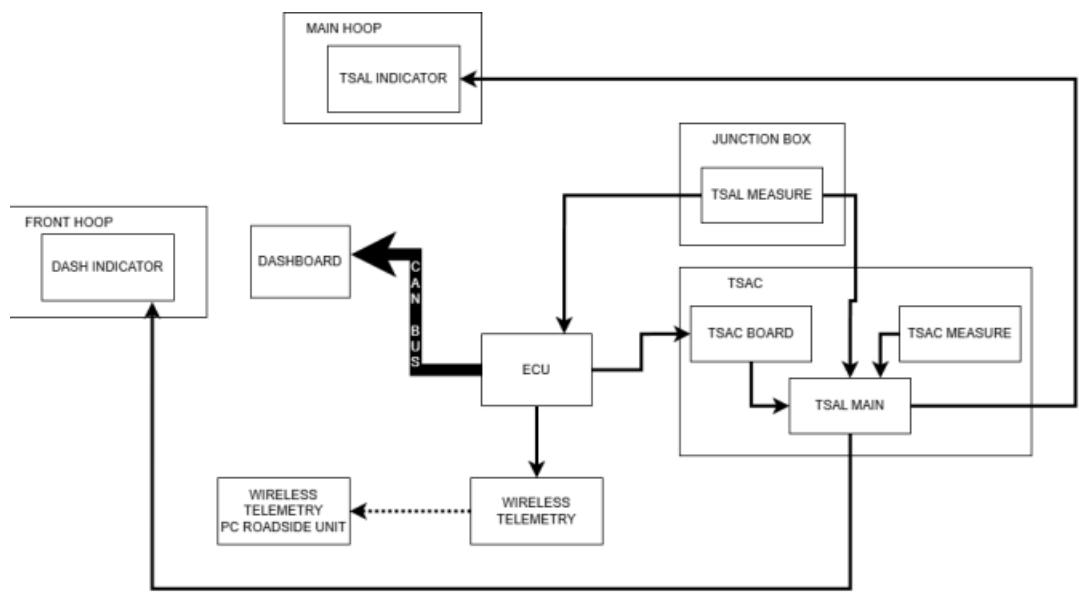


Figura 4.17: Diagrama Conceptuală a arhitecturii sistemului de voltaj mic al ART TU E17

## Capitolul 5. Proiectare de detaliu și implementare

### 5.0.1. Arhitectura sistemului de telemetrie ART TU E17

În cadrul acestei secțiuni voi prezenta arhitectura aplicației și componentele sale, acesta având 5 componente:

- Partea relevantă a monopostului ART TU E17
- Transmițătorul LoRa
- Receptorul LoRa
- Road-Side Unit Server, incluzând baza de date locală și interfața pentru utilizator
- Baza de date din Cloud

În figura 5.1 se poate observa arhitectura propusă sistemului de telemetrie ART TU E17 și legătura dintre toate componentele descrise.

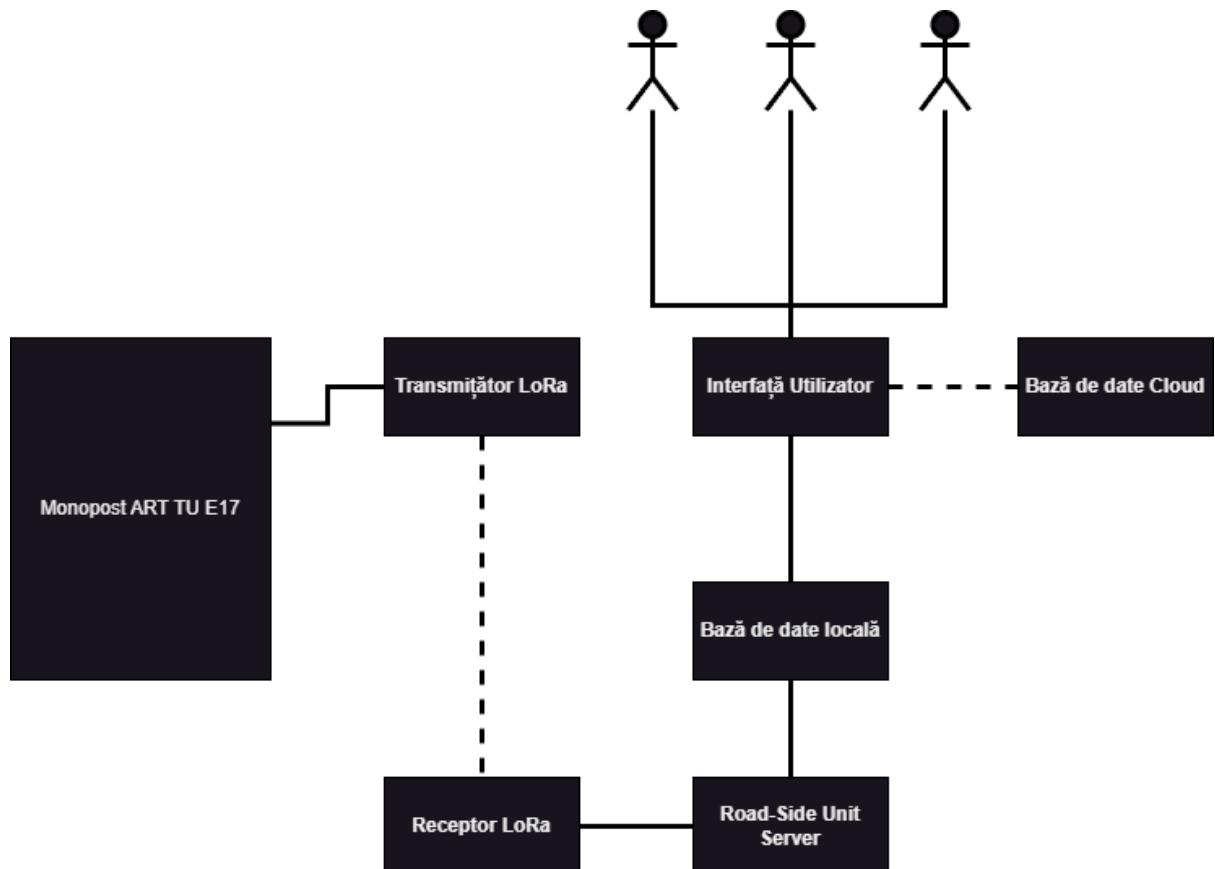


Figura 5.1: Arhitectura conceptuală a sistemului de telemetrie wireless

Pentru prima componentă am ales să includ monopostul ART TU E17 pentru că în acesta se regăsesc senzori și sisteme care trebuie monitorizate și care, prin urmare, generează date. Acesta este compus din sistemele descrise la 4.4 și o multitudine de senzori, inverteoare și motoare ale căror date sunt centralizate și procesate în ECU. Prin CAN Bus se leagă transmițătorul LoRa, care primește mesajele de la ECU conținând datele care se doresc să fie transmise, iar mai apoi le stochează temporar în structura de date,

---

formează un pachet, îl criptează și îl transmite receptorului LoRa. Acesta recepționează pachetul, îl decriptează, stochează datele în structura de date și le trimit bazei de date locale. De acolo, li se adaugă un timestamp și sunt stocate pe o perioadă de 2 zile. De acolo, datele sunt preluate de interfața grafică a utilizatorului, unde datele ce au fost transmise sunt afișate în timp real. După ce monopostul nu mai este operațional, utilizatorul va da upload la date către baza de date din Cloud unde sunt stocate indefinit din interfața sa grafică, iar de acolo utilizatorul poate descărca datele și le poate vizualiza interfața grafică.

### 5.0.2. Diagrama de flux

Prezint mai departe diagrama de flux a sistemului în figura 5.2, unde de la punctul de start, primul pas este preluarea datelor de la monopost de la ECU, unde sunt centralizate datele și compuse într-un mesaj ISO-TP CAN Bus care trimis spre transmîtătorul LoRa prin magistralele de CAN. Transmîtătorul preia mesajul, extrage datele din acesta și le stochează în structura de date aferentă ca un buffer. Apoi, le compune într-un pachet LoRa, care are o structură predefinită, criptează datele și le transmite prin LoRa.

În cazul în care pachetul nu a fost recepționat cu succes, acesta este ignorat și se așteaptă următorul pachet. Dacă transmisia are succes, atunci receptorul LoRa preia pachetul, îl decriprează, descompune pachetul și stochează datele în strucaturile de date aferente, ca un buffer. După aceea, receptorul trimite datele spre baza de date locală, unde sunt preluate de interfața grafică a utilizatorului și sunt vizualizate.

Când monopostul nu mai este operațional și nu mai trimit date, utilizatorul poate alege să nu uploadeze datele către Cloud, iar în acest caz datele sunt stocate timp de 2 zile iar apoi șterse. Dacă utilizatorul alege să uploadeze datele către cloud, atunci acestea sunt uploadate și mai poi utilizatorul poate descărca aceste date și să le vizualizeze ulterior.

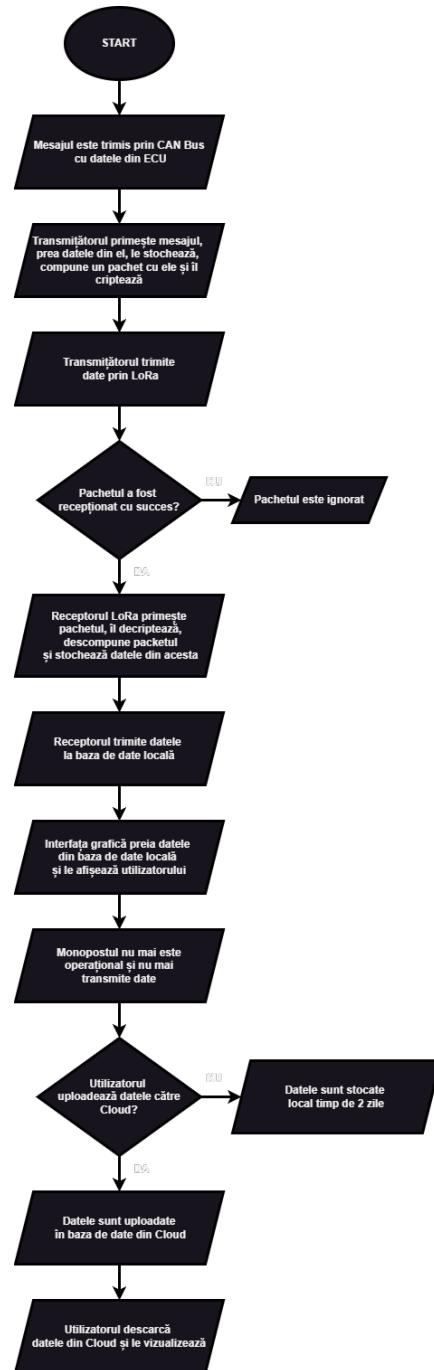


Figura 5.2: Diagrama de flux a sistemului de telemetrie.

O să descriu **fluxul principal**, unde totul merge fără probleme, iar primul pas este primirea mesajului CAN de la monopost către transmițător, fără nici un fel de problemă, cu datele integre și primite de receptor.

- **Use-Case Starts.** Transmițătorul LoRa primește mesajul de la ECU prin liniile de CAN.
  - <Step 1> Transmițătorul primește mesajul, preia datele din el, le stochează în structura de date, compune un pachet din ele și îl cripteză.
  - <Step 2> Transmițătorul trimite datele prin LoRa.
  - <Step 3> Pachetul a fost recepționat cu succes de receptorul LoRa
  - <Step 4> Receptorul LoRa decriptează pachetul, descompune pachetul și

---

stochează datele din acesta în structura de date.

- <Step 5> Receptorul trimite datele la baza de date locală.
  - <Step 6> Interfața grafică preia datele din baza de date locală și le afișează utilizatorului.
  - <Step 7> Monopostul nu mai este operațional și nu mai transmite date.
  - <Step 8> Utilizatorul uploadazează datele în baza de date din Cloud.
  - <Step 9> Utilizatorul descarcă datele din Cloud și le vizualizează.
- **Use-Case Ends.** Datele au fost preluate de la monopost, transmise, recepționate, stochate local, afișate și stochate persistent in Cloud cu succes.

**Fluxul alternativ 1**, unde pachetul este pierdut la legătura dintr transmițător și receptor.

- **Alternative Flow Starts.** Transmițătorul LoRa primește mesajul de la ECU prin liniile de CAN.
  - <Step 1> Transmițătorul primește mesajul, preia datele din el, le stochează în structura de date, compune un pachet din ele și îl criptează.
  - <Step 2> Transmițătorul trimite datele prin LoRa.
  - <Step 3> Pachetul **nu** a fost recepționat cu succes de receptorul LoRa.
  - <Step 4> Pachetul este ignorat.
  - <Step 5> Se reia transmisia cu date noi.
- **Alternative Flow Ends.** Datele au fost preluate de la monopost, transmise, dar nu au fost recepționate.

**Fluxul alternativ 2**, unde use-case-ul funcționează cum trebuie dar utilizatorul alege să nu uploadaze datele către Cloud.

- **Alternative Flow Starts.** Transmițătorul LoRa primește mesajul de la ECU prin liniile de CAN.
  - <Step 1> Transmițătorul primește mesajul, preia datele din el, le stochează în structura de date, compune un pachet din ele și îl criptează.
  - <Step 2> Transmițătorul trimite datele prin LoRa.
  - <Step 3> Pachetul a fost recepționat cu succes de receptorul LoRa
  - <Step 4> Receptorul LoRa decriptează pachetul, descompune pachetul și stochează datele din acesta în structura de date.
  - <Step 5> Receptorul trimite datele la baza de date locală.
  - <Step 6> Interfața grafică preia datele din baza de date locală și le afișează utilizatorului.
  - <Step 7> Monopostul nu mai este operațional și nu mai transmite date.
  - <Step 8> Datele sunt stochate local timp de 2 zile.
- **Alternative Flow Ends.** Datele au fost preluate de la monopost, transmise, recepționate, și afișate dar nu au fost uploadate către Cloud.

#### 5.0.3. Diagrama de secvență

Următoarea secvență va prezenta diagrama de secvență a sistemului atunci când acesta primește date noi de la monopost.

În figura 5.3 se poate observa că punctul de start este atunci când monopostul primește datele de la senzori și le centralizează în ECU. De acolo, ECU trimite datele către transmițător printr-un mesaj de CAN Bus. Apoi, transmițătorul formează pachetul și cripează datele înainte de a le trimite spre receptor folosind LoRa. După ce date au fost recepționate, în receptor se decriptează și descompune pachetul pentru a salva datele temporat în baza de date locală. De acolo, interfața grafică a utilizatorului afișează datele.

Tot din interfață, se salvează datele în Cloud și de acolo prin interfață se pot vizualiza datele din Cloud. În această diagramă, împreună cu figura 5.2, se poate observa cum funcționează sistemul de telemetrie al ART TU E17.

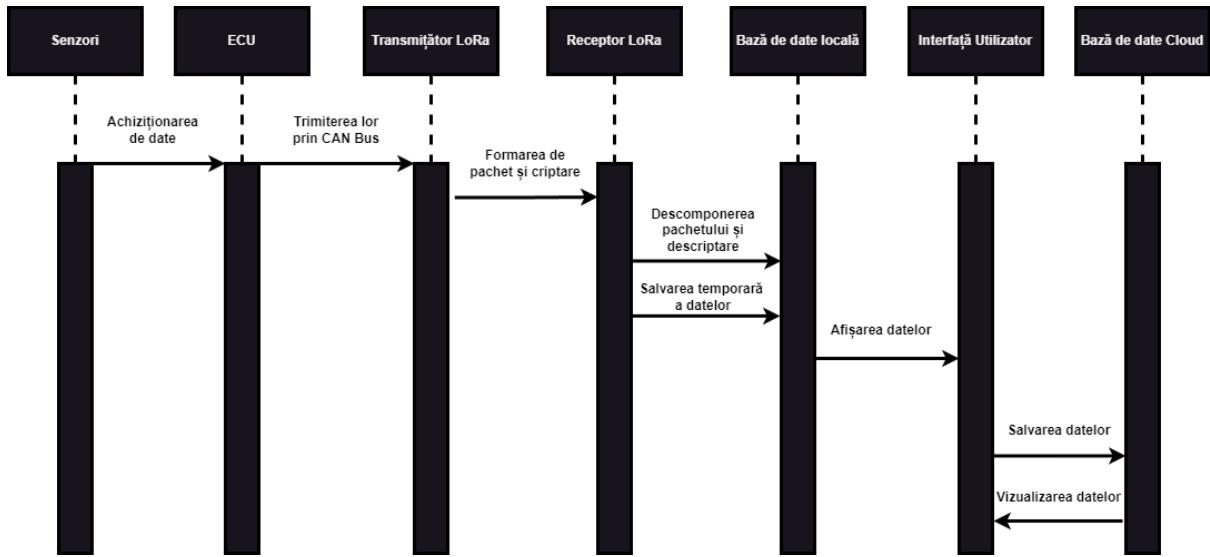


Figura 5.3: Diagrama de secvență a sistemului de telemetrie.

#### 5.0.4. Modulul 1: Partea relevantă a monopostului ART TU E17

În cadrul acestei secțiuni voi discuta despre primul modul al aplicației, despre partea relevantă a monopostului ART TU E17, mai exact de partea de trimitere a mesajului CAN către transmițător. Colectarea și centralizarea datelor care se întâmplă în ECU nu este în scopul lucrării.

Cum a fost descris mai sus, am ales ISO-TP CAN Bus pentru a putea trimite mesaje mari, des. Acet lucru nu se poate întâmpla fără o structură clară a datelor, iar după câteva ședințe cu echipa de dezvoltare, s-a ajuns la următoarele date, observate în figurile 5.4, 5.5, 5.6 și 5.7:

Department	Parameter	Importance	Bytes	ECU variable
Mechanical	Ride-height	downloaded		
	Suspension arms tensions	downloaded		
	Cooling parameters	downloaded	1	valueFlow, PUMP, VENT
	Brakes	downloaded		valueBrake
	Amortizoare	downloaded		
	Steering angle	Most Important	1	valueSteering
	Acceleration x2	Most Important	2	valueAcceleration
	Brake pressure	Most Important	1	valueBrake
	Lateral acceleration (from IMU)	Medium importance	3 values	
	Tire temperature	Medium importance		T1 T2
	Wheel slip	Medium importance		
	Suspension compression x4	Low importance		
	Pitch and roll	Low importance		

Figura 5.4: Datele necesare pentru departamentul Mechanical.

Vehicle Dynamics	Acceleration (longitudinal and lateral)	Most Important		valueAcceleration
	Suspension displacement	Most Important		
	Brake pressure	Most Important		
	Wheel spin x2	Most Important	4	RPM
	Battery monitoring (voltage, current, temp)	Most Important		LV_SOC, HV_SOC, valueCurrent
	Motor winding and inverter temperature	Most Important		INV_V
	Spinning wheel	Medium importance		
	Torque sensors	Medium importance		
	Brake disc/pad/tire surface temperature	Medium importance		
Deformation under stress	Deformation under stress	Low importance		PUMP1, PUMP2, VENT1
	Cooling system	Low importance		VENT2

Figura 5.5: Datele necesare pentru departamentul Vehicle Dynamics.

Tractive System	RPM Controller temperatures BMS (voltage, current, temp monitoring) Pack current, voltage, SoC Pack amperage Current limit status Highest temperature Total pack capacity Total kW power Motor controller temperature Motor temperature Input Voltage x2 Phase Current x2 Serial throttle value x2	All important	RPM LV_SOC, HV_SOC, valueCurrent LV_SOC, HV_SOC, valueCurrent
			2 PWM_C1, PWM_C2

Figura 5.6: Datele necesare pentru departamentul Tractive System.

Electrical	ECU events Data bytes for transmission Write parameters Pre-charge Imp. class All sensors not named by the other departments	All important	
	Door lock R2D speaker R2D Button AIR AIR- Precharge SOC PreCharge_Vin_COMP Vinf_PrefCharge Hump_charge Current sensor LV_SOC Ventilator	Low importance	
			1 bit on/off

Figura 5.7: Datele necesare pentru departamentul Electrical.

Datele sunt clasate în funcție de importanță lor în coloana de "Importance", având 4 valori, Low Importance, Medium Importance, High Importance, și Downloaded, care spune că datele respective nu sunt necesare în timp real, ci mai degrabă accesibile echipei după ce monopostul nu mai este operațional. Datele care sunt marcate cu roșu sunt date care nu pot fi preluate de ECU și nu pot fi transmise, dar dacă într-un final se pot adăuga, atunci sănii unde trebuie adăugate.

Pe lângă aceste coloane, avem și coloana de Bytes, care spune câți bytes ocupă data respectivă, fiind mapată după valoarea maxima care o poate avea aceasta (0 - 255 este 1 byte, 256 - 65536 2 bytes). Ultima coloană este variabila ECU care se ocupa de această măsurătoare, făcând mai facil accesul la acea dată. Datele care sunt doar de 1 bit (la departamentul Electrical) sunt comasate într-un singur byte (ECU-Control).

Așezarea acestora în mesajul de CAN este observată în figura 5.8, fiind mai ușoară descompunerea când acestea sunt trimise.

Byte	Name
0	Cooling
1	Brake Sensor
2	Steering Sensor
3	Accel. Sensor 1
4	Accel. Sensor 2
5	Wheel Spin 1
6	Wheel Spin 2
7	Motor Temp 1
8	Motor Temp 2
9	Controller Temp 1
10	Controller Temp 2
11	Serial throttle 1
12	Serial throttle 2
13	Input V1 PI
14	Input V1 PF
15	Input V2 PI
16	Input V2 PF
17	Phase Current 1 PI
18	Phase Cur 2 PI
19	Phase Cur 2 PF
20	Susp. travel FL
21	Susp. travel FR
22	Susp. travel BL
23	Susp. travel BR
24	ECU Control
25	VRef_PreCharge
26	MeasPrecharge
27	Current sensor
28	LV SOC

Figura 5.8: Datele aranjate în mesajul de CAN Bus.

Acste date sunt apoi trimise la transmîtător prin CAN BUS.

### 5.0.5. Modulul 2: Transmîtătorul LoRa

Modulul TxLoRa (Transmitter LoRa) reprezintă componenta esențială responsabilă cu colectarea datelor de telemetrie de la ECU și transmiterea acestora prin legătura radio LoRa către receptorul LoRa. Acesta este conectat direct la magistrala CAN a vehiculului, prin cele două fire CAN High (CAN H) și CAN Low (CAN L), utilizând protocolul ISO-TP pentru a transfera mesaje către Arduino MKR WAN 1310.

Acste mesaje CAN sunt interceptate și procesate de codul rulând pe placă Arduino MKR WAN 1310, unde sunt mapate într-o structură de date dedicată, TelemetryData. Această clasă conține câmpuri definite în concordanță cu parametrii de interes, și a fost proiectată astfel încât datele să poată fi serializate rapid într-un format binar compact, potrivit pentru transmisie LoRa.

```
struct TelemetryData
{
    uint8_t cooling;
    uint8_t brake_sens;
    uint8_t steer_sens;
    uint8_t acc1;
```

---

```
    uint8_t acc2;
    uint16_t wheel_spin_1;
    uint16_t wheel_spin_2;
    uint8_t motor_temp_1;
    uint8_t motor_temp_2;
    uint8_t control_temp_1;
    uint8_t control_temp_2;
    uint8_t serial_throttle_1;
    uint8_t serial_throttle_2;
    float inputV1_p;
    float inputV2_p;
    float phasecurrent1;
    float phasecurrent2;
    uint8_t susp_travel_FL;
    uint8_t susp_travel_FR;
    uint8_t susp_travel_BL;
    uint8_t susp_travel_BR;
    uint8_t ECU_control;
    uint16_t VRef_purge;
    uint16_t meas_purge;
    uint16_t current_sensor;
    uint16_t LV_state_of_charge;

    //for packed bytes
    uint8_t ventValue;
    uint8_t pumpValue;
    float flowValue;

    uint8_t brakeEngaged;
    uint8_t soundPlaying;
    uint8_t R2D_Button_State;
    uint8_t airPlusValue;
    uint8_t airMinusValue;
    uint8_t prechgValue;
    uint8_t SDC_END;
    uint8_t Measure_Digital;

    uint8_t IMU_Speed;
    float accelX_IMU;
    float accelY_IMU;
    float pitch_IMU;
    float roll_IMU;
    float yaw_IMU;

    uint16_t rawTemp1;
    uint16_t rawTemp2;
};
```

Pentru transmisia de date prin intermediul LoRa, am decis să folosesc libăria **Ra-**

**dioLib** care se ocupă de toate protocolele de nivel 1 și 2, iar tot ce trebuie să fac eu este ajustarea valorilor pentru comunicarea radio (Frecvență, Spreading Factor, Bandwidth, Coding Rate, Output Power, Preamble Length, CRC și cuvântul de sincronizare). Aici este partea relevantă de cod care se ocupă de aceste setări, încercând să atingă echilibrul dintre distanță și integritatea datelor:

```
void setSettings2(){
    radio.setFrequency(868.1); //FRECVENȚĂ
    radio.setSpreadingFactor(8); //SPREADING FACTOR
    radio.setBandwidth(125.0); //BANDWIDTH
    radio.setCodingRate(5); //CODING RATE
    radio.setOutputPower(14); // OUTPUT POWER
    radio.setPreambleLength(8); //PREAMBLE LENGTH
    radio.setCRC(true); //CRC
    radio.setSyncWord(0x45); // CUVÂNTUL DE SINCRONIZARE
    Serial.println("radio settings init done");
}
```

Pentru a putea realiza **Frequency Hopping**, am ales să folosesc frecvențele **868.1, 867.1, 868.5, 868.3, 867.3, 867.5, 867.9, 867.7** pentru a putea trimite cât mai multe date, secționându-le și distribuindu-le pe toate cele 7 frecvențe. Aici este partea relevantă din header-ul LoraRadio.h, care se ocupă de transmiterea de date:

```
void sendData(uint8_t *str, size_t size) {
    Serial.print("sizeof(str) = "); Serial.println(sizeof(str));
    float str_size = size;
    int subSize = ceil(str_size / 7);
    freq_cnt = 0;

    Serial.print("str_Size: ");
    Serial.println(str_size);
    Serial.print("subSize: ");
    Serial.println(subSize);

    //cycle through freq and divide the message into subSize chunks
    uint8_t* sizedChunk = (uint8_t*)calloc(subSize, sizeof(uint8_t));
    int cnt = 0;
    for (int i = 0; i < 7; ++i) {
        Serial.print("Cnt: \t\t");
        Serial.println(cnt);

        Serial.print("Chunk:\t\t");
        for(int j=0;j<subSize; j++){
            sizedChunk[j] = str[j+cnt];
            Serial.print(sizedChunk[j]);
            Serial.print(" ");
        }
        Serial.println();
    }
}
```

---

```

        Serial.print("Freq:\t\t");
        Serial.println(freq_list[freq_cnt]);

        radio.setFrequency(freq_list[freq_cnt]);
        transmissionState = radio.startTransmit(sizedChunk, subSize);
        if (transmissionState == RADIOLIB_ERR_NONE)
            Serial.println("Chunk transmit OK.\n");
        else {
            Serial.print(F("failed chunk transmit, code "));
            Serial.println(transmissionState);
        }

        // delay transmission of each chunk by some value
        // to make sure we can receive every packet on the receiver side
        // if we want to also acknowledge then we need to change this
        delay(100);

        freq_cnt++;
        cnt += subSize;
        if (freq_cnt > 6) freq_cnt = 0;
    }
    free(sizedChunk);
    transmitFlag = true;
}

```

După cum se poate observa, mesajul primit sub formă de String este împărțit egal între cele 7 frecvențe, numite **chunk-uri**. Aceste chunk-uri sunt trimise secvențial pe fiecare frecvență, reușind astfel să se transmită un număr mai mare de date cu un timp de așteptare mai mic, reușind să trecem peste limitarea de duty cycle impusă de LoRa. Transmisia de date începe de la frecvența 868.1, iar așezarea datelor în pachet este identică cu așezarea datelor din mesajul de CAN.

Apoi, datele sunt criptate folosind **AES-128 CNT**, care aduc un nivel de securitate aplicației. În primă fază, se face o structură care ține starea operațiilor AES în **AES-ctx** și apoi prin **AES-init-ctv-iv** se initializează folosind initialization vector-ul de 128 de biți. Apoi se face o copie a datelor și **AES-CRT-xcrypt-buffer** cripteză aceste date folosind CTR și cheia de 128 de biți, date care mai apoi sunt trimise prin LoRa.

```

can.isotp.receive(&can.receivedMsg);

if(can.receivedMsg.tp_state == ISOTP_FINISHED) {
    Serial.println();
    Serial.print("Received msg: ");
    for(int i=0;i<can.receivedMsg.len;i++) {
        Serial.print(can.receivedMsg.Buffer[i]);
        Serial.print(" ");
    }
    Serial.println();

    // AES CTR ENCRYPTION pentru date CAN
}

```

```

    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, aes_key, aes_iv);
    uint8_t encryptedCAN[can.receivedMsg.len];
    memcpy(encryptedCAN, can.receivedMsg.Buffer, can.receivedMsg.len);
    AES_CTR_xcrypt_buffer(&ctx, encryptedCAN, can.receivedMsg.len);

    lora.sendData(encryptedCAN, can.receivedMsg.len);
    transmitFlag = true;

    can.receivedMsg.tp_state = ISOTP_IDLE;
}

```

### 5.0.6. Modulul 3: Receptorul LoRa

RxLoRa (Receiver LoRa) este componenta responsabilă cu recepționarea datelor provenite din Transmițătorul LoRa. Acest modul este conectat la un Road-Side Unit, în cazul nostru acționând ca un server un laptop normal. Legătura dintre cele două se realizează cu un cablu **MicroUSB** și comunicarea dintre cele două se face prin **Serial**.

În ceea ce privește structura de date, este identică, doar că aici avem logica de descompunere a pachetului LoRa și decriptarea sa.

După recepționarea unui pachet LoRa, primul pas este decriptarea acestuia folosind algoritmul AES-128 în modul CTR, utilizând aceeași cheie de criptare ca la transmisie. Această etapă restabilește conținutul original al mesajului, păstrând confidențialitatea și integritatea datelor în fața interceptărilor, observându-se în codul de mai jos. Aceeași pași ca și la transmițător, doar că aici **AES-CTR-xcrypt-buffer** decriptează în loc să cripteze datele.

```

if (receiveValid)
{
    size_t received_len = 32; // or however many bytes you actually receive
    struct AES_ctx ctx;
    uint8_t ctr_iv[16];
    memcpy(ctr_iv, aes_iv, 16); // must match the IV/nonce used by transmitter

    AES_init_ctx_iv(&ctx, aes_key, ctr_iv);

    // Decrypt in-place (CTR mode)
    AES_CTR_xcrypt_buffer(&ctx, receivedMsg, received_len);

    // now receivedMsg contains the original struct
    TelemetryData* receivedData = (TelemetryData*)receivedMsg;
    memcpy(&dataToSend, receivedData, sizeof(TelemetryData));
    receiveValid = false;
    Serial.println("\nStarting data processing.");

    // process data
    sendToInfluxDB();
    memset(receivedMsg, 0, sizeof(receivedMsg));
}

```

---

Odată decriptat, conținutul pachetului este mapat în structura TelemetryData. Funcția convertToInfluxDB despachetează pachetul și mapează datele. Un mic truc pentru a reuși să transmit datele care sunt de tip Float este să împart cele două valori, ce este înainte de virgulă și ce este după, în parte integrală și parte fracționară, care mai apoi le unesc împreună. Pentru datele care sunt doar un bit ON / OFF, aici se descompun folosind operații logice și mapate la valorile corespunzătoare din structura de date.

```

void convertToInfluxDB(void){
    dataToSend.cooling = receivedMsg[0];                                // vent/pump/flow packed
    dataToSend.brake_sens = receivedMsg[1];                               // valueBrake
    dataToSend.steer_sens = receivedMsg[2];                               // valueSteering
    dataToSend.acc1 = receivedMsg[3];                                     // pos1
    dataToSend.acc2 = receivedMsg[4];                                     // pos2

    // Wheel RPMs
    dataToSend.wheel_spin_1 = (receivedMsg[5]) | (receivedMsg[6] << 8); // controllerDataL.RPMController
    dataToSend.wheel_spin_2 = (receivedMsg[7]) | (receivedMsg[8] << 8); // controllerDataR.RPMController

    // Temperatures
    dataToSend.motor_temp_1 = receivedMsg[9]; // controllerDataL.tempMotor
    dataToSend.motor_temp_2 = receivedMsg[10]; // controllerDataR.tempMotor
    dataToSend.control_temp_1 = receivedMsg[11]; // controllerDataL.tempController
    dataToSend.control_temp_2 = receivedMsg[12]; // controllerDataR.tempController

    // Throttle voltages (truncated integer values)
    dataToSend.serial_throttle_1 = (uint8_t)(receivedMsg[13] | (receivedMsg[14] << 8)); // controllerDataL.throttle
    dataToSend.serial_throttle_2 = (uint8_t)(receivedMsg[15] | (receivedMsg[16] << 8)); // controllerDataR.throttle

    // Supply voltages (scaled int16_t, divide by 100)
    int16_t inputV1_raw = (int16_t)(receivedMsg[17] | (receivedMsg[18] << 8));
    int16_t inputV2_raw = (int16_t)(receivedMsg[19] | (receivedMsg[20] << 8));
    dataToSend.inputV1_p = inputV1_raw / 100.0f;
    dataToSend.inputV2_p = inputV2_raw / 100.0f;

    // Phase currents (scaled int16_t, divide by 100)
    int16_t phase1_raw = (int16_t)(receivedMsg[21] | (receivedMsg[22] << 8));
    int16_t phase2_raw = (int16_t)(receivedMsg[23] | (receivedMsg[24] << 8));
    dataToSend.phasecurrent1 = phase1_raw / 100.0f;
    dataToSend.phasecurrent2 = phase2_raw / 100.0f;

    // Suspension travel
    dataToSend.susp_travel_FL = receivedMsg[25];
    dataToSend.susp_travel_FR = receivedMsg[26];
    dataToSend.susp_travel_BL = receivedMsg[27];
    dataToSend.susp_travel_BR = receivedMsg[28];
}

```

```

// Packed ECU control bits
dataToSend.ECU_control = receivedMsg[29];
dataToSend.VRef_purge = (uint16_t)(receivedMsg[30] | (receivedMsg[31] << 8));
dataToSend.current_sensor = (uint16_t)(receivedMsg[32] | (receivedMsg[33] << 8));
dataToSend.meas_purge = (uint16_t)(receivedMsg[49] | (receivedMsg[50] << 8));
dataToSend.LV_state_of_charge = receivedMsg[51]; // state of charge in %

// Wheel speed (km/h)
dataToSend.IMU_Speed = receivedMsg[34];

// ----- IMU Data (signed int16_t)
dataToSend.accelX_IMU = (int16_t)(receivedMsg[35] | (receivedMsg[36] << 8)) /
    / 32768.0f * 16.0f; // accelX in g
dataToSend.accelY_IMU = (int16_t)(receivedMsg[37] | (receivedMsg[38] << 8)) /
    / 32768.0f * 16.0f; // accelY in g
dataToSend.roll_IMU = (int16_t)(receivedMsg[39] | (receivedMsg[40] << 8)) /
    / 32768.0f * 180.0f; // roll in degrees
dataToSend.pitch_IMU = (int16_t)(receivedMsg[41] | (receivedMsg[42] << 8)) /
    / 32768.0f * 180.0f; // pitch in degrees

dataToSend.rawTemp1 = (uint16_t)(receivedMsg[43] | (receivedMsg[44] << 8));
dataToSend.rawTemp2 = (uint16_t)(receivedMsg[45] | (receivedMsg[46] << 8));

dataToSend.yaw_IMU = (int16_t)(receivedMsg[47] | (receivedMsg[48] << 8)) /
    / 32768.0f * 180.0f; // yaw in degrees

// ----- Bit-unpack logic -----
dataToSend.ventValue      = (receivedMsg[0] >> 7) & 0x01;
dataToSend.pumpValue      = (receivedMsg[0] >> 6) & 0x01;
dataToSend.flowValue      = unpackFlowLps((receivedMsg[0] & 0x3F)<<2);
// unpacked flow in L/s
dataToSend.brakeEngaged   = (receivedMsg[29] >> 7) & 0x01;
dataToSend.soundPlaying    = (receivedMsg[29] >> 6) & 0x01;
dataToSend.R2D_Button_State = (receivedMsg[29] >> 5) & 0x01;
dataToSend.airPlusValue    = (receivedMsg[29] >> 4) & 0x01;
dataToSend.airMinusValue   = (receivedMsg[29] >> 3) & 0x01;
dataToSend.prechgValue     = (receivedMsg[29] >> 2) & 0x01;
dataToSend.SDC_END         = (receivedMsg[29] >> 1) & 0x01;
dataToSend.Measure_Digital = (receivedMsg[29] >> 0) & 0x01;
}

```

În continuare, TelemetryData este trecută printr-o funcție de conversie numită sendToInfluxDB(), care transformă valorile într-un sir text conform cu Line Protocol, formatul utilizat de InfluxDB pentru scrierea de date în bazele sale de timp. Line Protocol este un format text simplu, dar eficient, în care fiecare linie reprezintă un punct de măsurare, cu un nume de serie (measurement), tag-uri optionale, câmpuri și un timestamp care sunt trimise prin Serial la server.

---

```

void sendToInfluxDB()
{
    // telemetry received array to struct
    convertToInfluxDB();

    // formatting the data in line protocol for InfluxDB
    Serial.println("START INFLUX");
    Serial.print("telemetry_data,location=test,device=senzori ");

    Serial.print("cooling="); Serial.print(dataToSend.cooling);
    Serial.print(",");
    Serial.print("brake_sens="); Serial.print(dataToSend.brake_sens);
    Serial.print(",");
    Serial.print("steer_sens="); Serial.print(dataToSend.steer_sens);
    Serial.print(",");
    Serial.print("acc1="); Serial.print(dataToSend.acc1); Serial.print(",");
    Serial.print("acc2="); Serial.print(dataToSend.acc2); Serial.print(",");
    Serial.print("wheel_spin_1="); Serial.print(dataToSend.wheel_spin_1);
    Serial.print(",");
    Serial.print("wheel_spin_2="); Serial.print(dataToSend.wheel_spin_2);
    Serial.print(",");

    // Add the extracted boolean/packed fields
    Serial.print("ventValue="); Serial.print(dataToSend.ventValue);
    Serial.print(",");
    Serial.print("pumpValue="); Serial.print(dataToSend.pumpValue);
    Serial.print(",");
    Serial.print("flowValue="); Serial.print(dataToSend.flowValue);
    Serial.print(",");
    Serial.print("brakeEngaged="); Serial.print(dataToSend.brakeEngaged);
    Serial.print(",");
    Serial.print("soundPlaying="); Serial.print(dataToSend.soundPlaying);
    Serial.print(",");

    Serial.print("accelX="); Serial.print(dataToSend.accelX_IMU); Serial.print(",");
    Serial.print("accelY="); Serial.print(dataToSend.accelY_IMU); Serial.print(",");
    Serial.print("roll="); Serial.print(dataToSend.roll_IMU);
    Serial.println(" ");
}

```

Acum toți acești pași se întâmplă doar după ce datele au fost reconstruite după transmisie. Acest lucru se întâmplă în loop, iar principiul de bază este următorul: când se primesc date pe o frecvență, se salvează într-un buffer și se așteaptă până când s-au primit cele 7 pachete de pe cele 7 frecvențe. Receiver-ul așteaptă până când primește datele pe prima frecvență, iar dacă nu primește date pe alte frecvențe timp de 100 milisecunde, atunci ignoră tot pachetul. Dacă datele de la o frecvență nu sunt primite, atunci datele de pe celălalte frecvențe sunt ignorate și se reîncearcă transmisia cu date noi. După ce s-au primit datele de pe toate frecvențele, atunci se construiesc într-un mesaj final cu datele corecte. Aici este partea relevantă a codului care se ocupă de acești pași:

```

// when flag is set, that means data was sent/received
if (operationDone)
{
    // reset flag
    operationDone = false;
    Serial.println("Entered operation done initial main.");
    freq_cnt = 0;
    int cnt = 0, cntM = 0;
    // module received data on channel 1
    uint8_t *str = (uint8_t *)calloc(20, sizeof(uint8_t));

    int packetLength = radio.getPacketLength();
    int state = radio.readData(str, packetLength);
    for (int i = 0; i < packetLength; i++)
    {
        receivedMsg[i + cnt] = str[i];
        cntM++;
    }
    cnt += packetLength;

    // if message seen on first channel, cycle through the rest
    if (state == RADIOLIB_ERR_NONE)
    {
        // display first chunk
        lora.displayData(str, packetLength);
        receiveValid = true;
        /*
        chunk cycler
        goes through all channels and waits for the
        respective chunk to arrive
        */
        for (int i = 1; i < 7; ++i)
        {
            freq_cnt++;
            radio.setFrequency(freq_list[freq_cnt]);
            radio.startReceive();

            // reset str array
            memset(str, 0, 20 * sizeof(*str));

            // wait until next message is received on next channel
            //OR timeout of 100ms
            // opDone = false when entering, interrupt will make it true
            unsigned long timeStart = millis();
            Serial.print("waiting for next chunk");
            while (!operationDone && (millis() - timeStart) < 300)
            {}
            Serial.print("\n");
        }
    }
}

```

---

```
        if (!operationDone)
            Serial.println("ERR!! Message missed.");
        operationDone = false;

        packetLength = radio.getPacketLength();
        state = radio.readData(str, packetLength);
        for (int i = 0; i < packetLength; i++)
        {
            receivedMsg[i + cnt] = str[i];
            cntM++;
        }
        cnt += packetLength;

        // display received chunk if everything is ok
        if (state == RADIOLIB_ERR_NONE)
        {
            lora.displayData(str, packetLength);
        }
        else
        {
            Serial.print(F("failed chunk read, code "));
            Serial.println(state);
            receiveValid = false;
        }
    }
else
{
    Serial.print(F("failed chunk 1 read, code "));
    Serial.println(state);
    Serial.print(F("Freq listening: "));
    Serial.println(freq_list[freq_cnt]);
}

// display whole message and reset string if all chunks are received
if (receiveValid)
{
    Serial.println("\nFull message: ");
    for (int i = 0; i < cntM; i++)
    {
        Serial.print(receivedMsg[i]);
        Serial.print(" ");
    }
    Serial.println();
}
else
{
    Serial.println("Message chunks invalid. Skipping this packet.");
}
```

```

        memset(receivedMsg, 0, sizeof(receivedMsg));
    }

    // set freq back to channel 1 and listen again for next packet
    freq_cnt = 0;
    radio.setFrequency(freq_list[freq_cnt]);
    radio.startReceive();

}

```

#### 5.0.7. Modulul 4: Baza de date locală din Road-Side Unit (Server)

Road-Side Unit (RSU) reprezintă componenta intermediaрă dintre receptorul LoRa și infrastructura software de tip cloud. RSU este implementat, în cazul nostru, ca un laptop, având rolul de gateway local pentru receptia și stocarea datelor transmise de vehicul.

Înainte de a putea introduce date în InfluxDB, containerul pentru acesta trebuie pornit, deschizând un terminal în directorul unde se află fișierul "docker-compose" și se introduce comanda "docker-compose up -d" și se așteaptă până când status-ul containteurului InfluxDB este "Started"

Pentru a automatiza procesul de preluare și încărcare a datelor în baza de date, RSU rulează un script Python transferScript.py. Acest script are mai multe funcționalități, precum detectarea portului Serial și așteptând input de la utilizator pentru confirmarea că acesta este receptorul LoRa, citirea datelor sub forma Line Protocol și scrierea în InfluxDB.

Pentru a putea scana și detecta porturile de Serial, am folosit librăria Serial.tools.list\_ports, care doar afișează toate dispozitivele conectate la porturile Serial ale serverului. Pe lângă acesta, am adăugat și o fereastră pop-up care arată utilizatorului care sunt porturile Serial și care vrea să se monitorizeze, așteptând confirmarea sa. Mai jos este codul aferent:

```

#list ports
ports = list(serial.tools.list_ports.comports())
port_list = "\n".join([f"{p.device}: {p.description}" for p in ports]) +
+ or "No serial ports found."

# pop up with available serial ports and ask for COM port
def get_com_port():
    while True:
        def on_ok():
            nonlocal com_port
            com_port = entry.get()
            root.destroy()
        def on_exit():
            root.destroy()
            os._exit(0)

        com_port = None
        root = tk.Tk()

```

---

```

root.title("Select COM Port")
tk.Label(root, text=f"Detected serial ports:\n\n{port_list}\n\nEnter COM port (e.g., COM6):").pack(padx=10, pady=10)
entry = tk.Entry(root)
entry.pack(padx=10, pady=5)
entry.focus()
button_frame = tk.Frame(root)
button_frame.pack(pady=10)
tk.Button(button_frame, text="OK", command=on_ok).pack(side=tk.LEFT, padx=5)
tk.Button(button_frame, text="Exit", command=on_exit).
    .pack(side=tk.LEFT, padx=5)
root.mainloop()
# validate COM port from user
available_ports = [p.device for p in ports]
if com_port in available_ports:
    return com_port
elif com_port is not None:
    tk.messagebox.showerror("Invalid Port", f'{com_port} is not a valid port. Please try again.')
selected_com_port = get_com_port()
if not selected_com_port:
    messagebox.showerror("Error", "No COM port selected. Exiting.")
    exit(1)

```

Datele de autentificare pentru InfluxDB sunt stocate într-un fișier .env și acest script de Python preia datele de acolo (username, parola, URL, bucket, organization), eliminând nevoie de un pas de login. Apoi, acesta deschide portul Serial selectat de utilizator și anunță utilizatorul că s-a conectat cu succes. Următorul pas este preluarea datelor de pe Serial și citirea acestora, ele sunt deja în Line Protocol după cum se observă în receptor, iar când scriptul recunoaște linia "START INFLUX", acesta va introduce datele în baza de date locală. Acest pas este necesar pentru a delimita datele de debugging de cele reale. Aici este codul aferent:

```

#load .env file
load_dotenv()

# read values from .env
INFLUXDB_URL = f"http://{{os.getenv('DOCKER_INFLUXDB_INIT_HOST')}}:{os.getenv('DOCKER_INFLUXDB_INIT_PORT')}}"
INFLUXDB_TOKEN = os.getenv("DOCKER_INFLUXDB_INIT_ADMIN_TOKEN")
INFLUXDB_ORG = os.getenv("DOCKER_INFLUXDB_INIT_ORG")
INFLUXDB_BUCKET = os.getenv("DOCKER_INFLUXDB_INIT_BUCKET")

# serial port configuration (Windows)
ser = serial.Serial(selected_com_port, 9600)
print(f"Connected to {selected_com_port} at 9600 baud.")
# initialize InfluxDB client
client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)

```

```

write_api = client.write_api()

start_influx = False

try:
    while True:
        line = ser.readline().decode('utf-8').strip()
        #print(line)
        if line:
            if line == "START INFLUX":
                start_influx = True
                print("Received START INFLUX")
            elif start_influx:
                write_api.write(bucket=INFLUXDB_BUCKET,
                               ,org=INFLUXDB_ORG, record=line)
                print(f"Sent to InfluxDB: {line}")
                print(f"URL: {INFLUXDB_URL} | Org: {INFLUXDB_ORG} |
| Bucket: {INFLUXDB_BUCKET}")
                start_influx = False
except KeyboardInterrupt:
    print("Exiting...")
finally:
    ser.close()

```

În cadrul acestei arhitecturi, RSU joacă un rol esențial în paradigma de fog computing, în care o parte din procesarea și filtrarea datelor este realizată local, chiar înainte ca datele să fie trimise către cloud. Această abordare reduce latența în transmiterea și procesarea datelor critice, traficul de rețea și dependența de conexiunea la internet, oferind un timp de răspuns mai scurt pentru aplicații în timp real.

Astfel, RSU acționează ca un nod de calcul de margine (edge node), menținând sistemul eficient, chiar și în condiții de conectivitate limitată.

#### 5.0.8. Modulul 5: Interfața grafică a utilizatorului

Pentru vizualizarea datelor, pe RSU este instalat și configurat Grafana, o platformă de monitorizare și analiză a datelor. Dashboard-ul Grafana a fost configurat cu un refresh rate de 1 secundă, astfel încât graficele și valorile se actualizează aproape instantaneu pe măsură ce datele noi sunt scrise în InfluxDB. Grafana interoghează direct baza de date și afișează valori gata prelucrate de ECU, exact așa cum vin din sistem, fără niciun delay sau procesare intermediară, pentru a oferi utilizatorului o vedere în timp real asupra stării monopostului.

Pentru a putea lua date din InfluxDB, este nevoie de putin cod Flux, care este limbajul de preluat date nativ InfluxDB, similar cu SQL. În figura 5.9, se poate observa meniul de configurare al unui panou din Grafana și folosirea Flux, care apare în partea de jos a figurii:

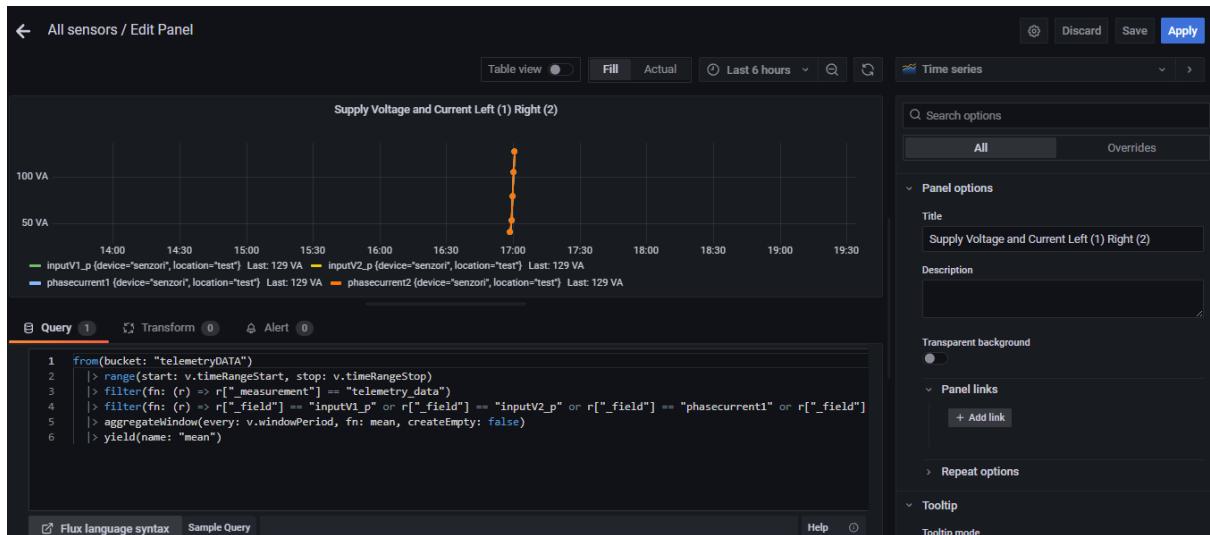


Figura 5.9: Meniul de configurare al unui panou din Grafana cu folosirea Flux.

În figura 5.10 se poate observa cum arată interfața grafică pentru utilizator finală, cu date reale, care se actualizează la secundă:



Figura 5.10: Configurarea finală a interfeței grafice a utilizatorului.

### 5.0.9. Modulul 6: Baza de date din Cloud

Componenta de Cloud Computing joacă un rol esențial în extinderea accesului la datele de telemetrie dincolo de limita serverului (RSU), permitând atât stocarea centralizată cât și analizarea datelor la scară mai largă.

Pentru a realiza această funcționalitate, sistemul utilizează un script Python uploadScript.py care are rolul de a extrage în timp real datele introduse în InfluxDB (baza de date locală) și de a le urca automat în Firebase, o platformă cloud oferită de Google. Astfel, datele de telemetrie devin disponibile pentru a fi accesate după ce monopostul nu mai este operational. Același script descarcă automat datele din ultima oră. Acesta întâi preia certificatele pentru Firebase (sunt cenzurate pentru această lucrare) și se conectează la acesta. Apoi, se face conexiunea cu InfluxDB cu tokenul pentru admin și bucket, iar funcția wait-for-user fie preia datele din InfluxDB din ultima oră prin variabilele query

și result, le grupează în funcție de tipul senzorului și timestamp și le uploadează când utilizatorul apasă pe buton. Pentru descărcarea datelor, procesul este similar, doar că în query se adună toate datele, iar mai apoi acestea se scriu sub format JSON în directorul Cloud-Data.

```
# initialize firebase
cred = credentials.Certificate('REDACTED')
FIRESTORE_APP = firebase_admin.initialize_app(cred)
FIRESTORE_DB = firestore.client()
DEFAULT_COLLECTION_NAME = "sensor-data"
def upload_to_firebase(data: dict):
    FIRESTORE_DB.collection(DEFAULT_COLLECTION_NAME).document().set(data)

# influxDB config
influx_url = "http://localhost:8086"
influx_bucket = "telemetryDATA"

# connect to influxDB
influx_client = InfluxDBClient(url=influx_url, token=INFLUXDB_INIT_ADMIN_TOKEN,
org=INFLUXDB_INIT_ORG)
influx_query_api = influx_client.query_api()

def wait_for_user():
    def on_upload():
        print("Querying InfluxDB for data...")
        query = f'from(bucket: "{influx_bucket}") |> range(start: -1h)'
        print(f"Executing query: {query}")
        result = influx_query_api.query(org=INFLUXDB_INIT_ORG, query=query)
        print(f"result: {result}")

        # group by (timestamp, measurement)
        grouped = defaultdict(dict)
        for table in result:
            for record in table.records:
                ts = str(record.get_time())
                measurement = record.get_measurement()
                key = (ts, measurement)
                grouped[key]['timestamp'] = ts
                grouped[key]['measurement'] = measurement
                grouped[key][record.get_field()] = record.get_value()

        upload_count = 0
        for doc in grouped.values():
            print(f"Uploading grouped data: {doc}")
            upload_to_firebase(doc)
            upload_count += 1

    messagebox.showinfo("Upload", f"Uploaded {upload_count} grouped records to Firestore!")
```

---

```

def on_download():
    # Download and aggregate all data from Firestore
    docs = FIRESTORE_DB.collection(DEFAULT_COLLECTION_NAME).stream()
    all_data = [doc.to_dict() for doc in docs]

    # Ensure output directory exists
    output_dir = "./Cloud_Data"
    os.makedirs(output_dir, exist_ok=True)

    # Save aggregated data to a CSV file with timestamp
    filename = f"cloud_data_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
    filepath = os.path.join(output_dir, filename)

    if all_data:
        # Save aggregated data to a JSON file with timestamp
        json_filename = f"cloud_data_{datetime.now().strftime('%Y%m%d_%H%M%S')}.json"
        json_filepath = os.path.join(output_dir, json_filename)
        with open(json_filepath, "w", encoding="utf-8") as f:
            json.dump(all_data, f, ensure_ascii=False, indent=2, default=str)
        messagebox.showinfo("Download Complete",
                            f"Data downloaded and saved to:\n{json_filepath}")
    else:
        messagebox.showinfo("No Data",
                            "No data found in Firestore to save.")

root = tk.Tk()
root.title("Cloud Data Actions")
tk.Label(root, text="Press a button to download or upload data from cloud.").pack(padx=20, pady=10)
tk.Button(root, text="Download Data", command=on_download).pack(pady=5)
tk.Button(root, text="Upload Data", command=on_upload).pack(pady=5)
root.mainloop()

wait_for_user()

```

În paralel, sistemul include un alt script Python csv-plotter.py plotează grafic datele descărcate din Firebase, la alegerea utilizatorului. Această componentă este utilă atât pentru vizualizare, cât și pentru depanare sau analiză după ce monopostul nu mai este pe circuit. Datele sunt tratate în format CSV și procesate în funcție de tipul senzorului (de exemplu: temperatură, tensiune, turație motor etc.), iar rezultatele sunt afișate direct într-o fereastră interactivă. Aici se poate observa meniul de selectare al fișierului cu date, care poate fi CSV sau JSON:

```

root = tk.Tk()
root.withdraw()
file_path = filedialog.askopenfilename(
    title="Select JSON or CSV file",

```

```

filetypes=[("JSON/CSV files", "*.json;*.csv"), ("All files", "*.*")]
)
if not file_path:
    print("No file selected.")
    exit()

# detect file type and load data
ext = os.path.splitext(file_path)[1].lower()
if ext == ".json":
    with open(file_path, "r", encoding="utf-8") as f:
        data = json.load(f)
    df = pd.DataFrame(data)
elif ext == ".csv":
    # skip comment lines starting with '#'
    df = pd.read_csv(file_path, comment="#")
    # rename columns to match expected names
    rename_map = {
        "_time": "timestamp",
        "_field": "metric_name",
        "_value": "value"
    }
    df = df.rename(columns=rename_map)
    # only keep required columns if they exist
    required_csv_cols = ["timestamp", "metric_name", "value"]
    if not all(col in df.columns for col in required_csv_cols):
        print(f"CSV file must contain columns: {required_csv_cols} (after renaming)")
        exit()
    df = df[required_csv_cols]
else:
    print("Unsupported file type.")
    exit()

```

Mai apoi, se preiau datele din fișier și scriptul plotează datele, 2 grafice pe pagină și se poate comuta între ele prin două butoane, Next și Previous:

```

def plot_page(page, fig, axes):
    start = page * plots_per_page
    end = min(start + plots_per_page, n_metrics)
    for i, ax in enumerate(axes):
        ax.clear()
        idx = start + i
        if idx < end:
            metric = metrics[idx]
            group = df[df['metric_name'] == metric]
            line, = ax.plot(group['timestamp'], group['value'], marker='o')
            ax.set_title(metric)
            ax.set_xlabel("Timestamp")
            ax.set_ylabel("Value")
            ax.grid(True)

```

---

```

        ax.set_visible(True)
        mplcursors.cursor(line, hover=True)
    else:
        ax.set_visible(False)
    fig.suptitle(f"Page {page+1} of {n_pages}")
    fig.canvas.draw_idle()

# fig, axes = plt.subplots(2, 2, figsize=(12, 8))
# axes = axes.flatten()

axprev = plt.axes([0.3, 0.01, 0.1, 0.05])
axnext = plt.axes([0.6, 0.01, 0.1, 0.05])
bnext = Button(axnext, 'Next')
bprev = Button(axprev, 'Previous')

def next_page(event):
    if current_page[0] < n_pages - 1:
        current_page[0] += 1
        plot_page(current_page[0], fig, axes)

def prev_page(event):
    if current_page[0] > 0:
        current_page[0] -= 1
        plot_page(current_page[0], fig, axes)

bnext.on_clicked(next_page)
bprev.on_clicked(prev_page)

plot_page(current_page[0], fig, axes)
plt.tight_layout(rect=[0, 0.08, 1, 1])
plt.show()

```

În figura 5.11 se poate observa cum arată vizualizarea datelor preluate din Cloud, folosind scriptul csv-plotter.py:

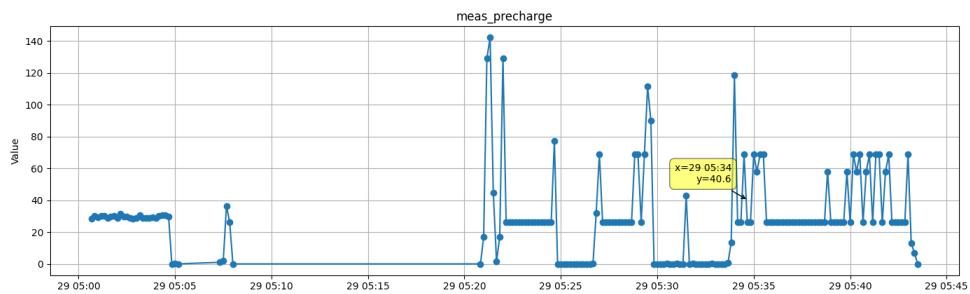


Figura 5.11: Vizualizarea datelor preluate din Cloud.

# Capitolul 6. Testare și validare

În acest capitol voi prezenta cum am testat sistemul de telemetrie prezentat. Voi prezenta testarea și rezultatele a:

- Cazurilor de utilizare.
- Distanței la care date se pot transmite fără a fi compromise.
- Viteza de actualizare a interfeței grafice.

## 6.1. Testarea și rezultatele cazurilor de utilizare

În cadrul capitolului 4, am prezentat cazurile de utilizare a sistemului, în figura 4.1. Am testat fiecare caz în parte și am verificat că sistemul îndeplinește aceste cazuri, cât și câteva cerințe, privind tot sistemul. Am decis, în această secțiune, să nu mă axeze pe componente individuale ale acestui sistem pentru că acestea nu se pot testa individual.

### 6.1.1. Testarea și validarea cazului 1

Pentru acest caz, s-a verificat funcționarea centralizării tuturor senzorilor și sistemelor care sunt prezente în monopost și dacă aceste date se pot trimite prin intermediul CAN Bus și LoRa. Acest caz prezintă funcționalitatea finală a sistemului, testând capacitatea acestuia de a gestiona un volum mare de date la o distanță rezonabilă. Testarea acestuia constă în instalarea transmîtătorului pe monopost și monitorizând datele acestuia când este static. Rezultatele se pot observa în figura 6.1, fiind prezente mai multe valori care nu sunt prezente din cauza sistemului de voltaj înalt care este deconectat. Aceasta este doar un test pentru a vedea dacă monopostul este capabil de a trimite date și receptorul de a recepționa date.

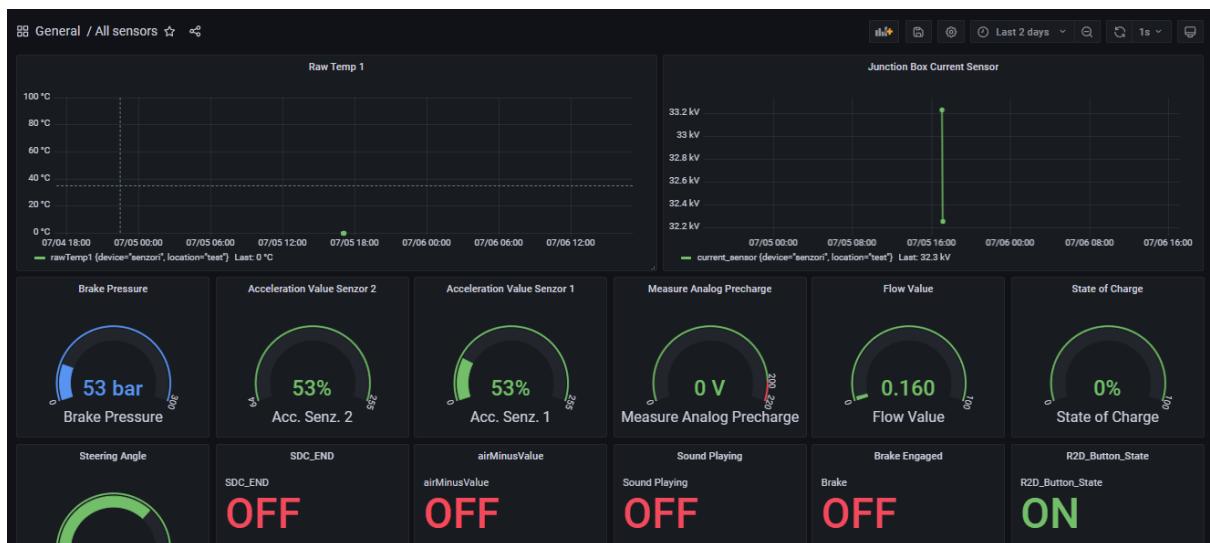


Figura 6.1: Testarea sistemului de telemetrie cât timp monopostul este static.

După cum se poate observa, monopostul trimite datele ”în repaus” a senzorilor de accelerație și frână a monopostului, de acceași valoare de 53, iar pentru State of

Charge, care indică starea de încărcare a bateriei este 0 pentru ca sistemul de voltaj ridicat este deconectat, pentru a testa sistemul de telemetrie. Toate valorile de ECU Control sunt off, iar Ready-To-Drive Button State este on pentru că am pornit sistemul de voltaj mic al monopostului. Pentru Junction Box Current Sensor, aici arată când a fost deconectat sistemul de voltaj mare. Aici este dashboard-ul cu toți senzorii, pentru a putea monitoriza datele care vin de la toți senzorii pentru a verifica funcționalitatea sistemului, chiar dacă aceste trimiteau aceleși date tot timpul. Pentru a testa această funcționalitate în întregime, am verificat acest dashboard și cât timp monopostul era operațional, observat în figura 6.2:



Figura 6.2: Testarea sistemului de telemetrie cât timp monopostul este operațional și în mișcare.

După cum se poate observa, în cadrul acestei sesiuni de test, senzorul de unghi de viraj este deconectat intenționat, iar folosind acest sistem am constat că senzorul de acceleratie Y și cel de Roll nu funcționează bine, găsind o problemă importantă.

### 6.1.2. Testarea și validarea cazului 2

Pentru cazul 2, am ales să testeze funcționalitatea sistemului pentru departamentul Electrical cât timp monopostul este în mișcare. A ajutat foarte mult filtrarea tuturor senzorilor fiind rămași doar cei relevanți pentru departamentul Electrical pentru că astfel am reușit să identificăm numeroase probleme, precum datele eronate citite de la controllerul de motoare (Supply Voltage and Current Left and Right), fiind o problemă care trebuie adresată de acest departament, datele de la controller fiind cruciale pentru cât curent consumă motoarele cât timp monopostul este operațional. De asemenea, folosind acest use case am văzut că valoarea pentru viteza este calculată greșit, pilot spunându-ne ulterior că viteza maximă atinsă era de 70 km/h. Aceste date se pot observa în figura 6.3, validând cazul de utilizare prezentat.

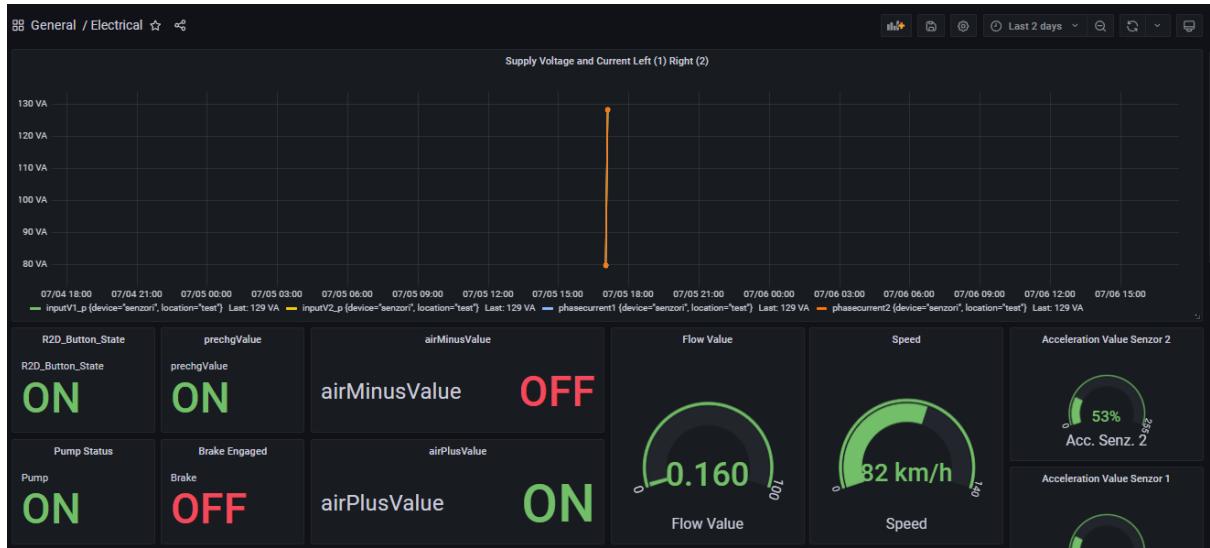


Figura 6.3: Testarea sistemului de telemetrie cât timp monopostul este operațional și în mișcare pentru departamentul Electrical.

#### 6.1.3. Testarea și validarea cazului 3

În cadrul acestei secțiuni am ales să monitorizez datele de la senzorul IMU (Inertial measurement unit) pentru a verifica funcționalitatea sa, împreună cu departamentul Vehicle Dynamics și Mechanical. Folosind modul de a monitoriza doar un senzor specific, am observat că acesta nu funcționează corect, validând și acest caz de utilizare prin găsirea unei probleme. În figura 6.4 se observă această problemă, semnalată de faptul că datele pentru accelerare în X sunt în formă de grafic, iar cele pentru Y sunt în linie dreaptă.

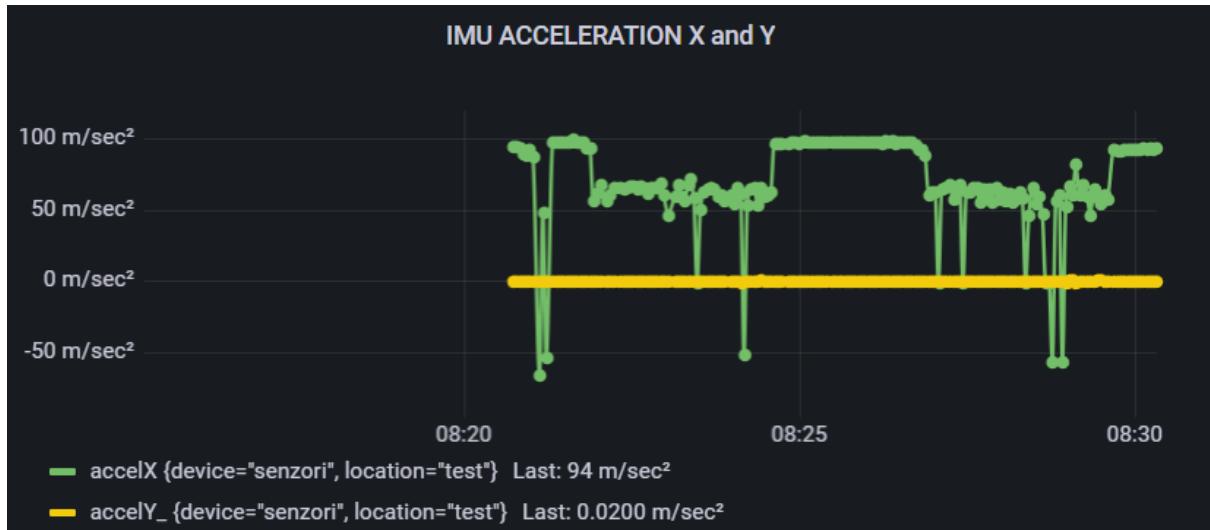


Figura 6.4: Testarea sistemului de telemetrie cât timp monopostul este operațional și în mișcare pentru senzorul IMU.

#### 6.1.4. Testarea și validarea cazului 4

Acest caz de utilizare a fost testat imediat după ce monopostul nu a mai fost operațional, testând funcționalitatea de Upload a sistemului din partea serverului, aceasta

funcționând ca atare, observabilă în figura 6.5.

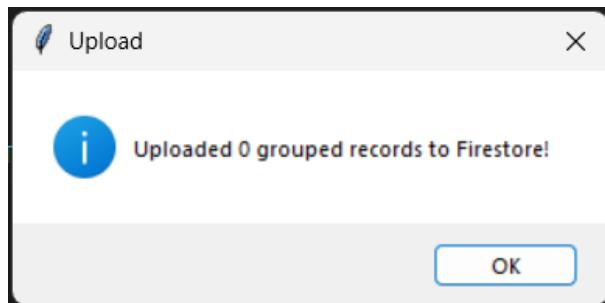


Figura 6.5: Pop-up care spune că uploadarea datelor în Cloud a avut succes.

Folosind alt laptop, am descărcat datele de pe Cloud (Figura 6.6) și am testat și scriptul de plotare a datelor, iar în figura 6.7 se pot observa rezultatele acestuia, fiind unele favorabile.

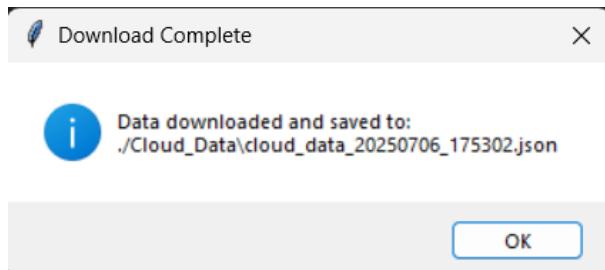


Figura 6.6: Pop-up care spune că descarcarea datelor din Cloud a avut succes.

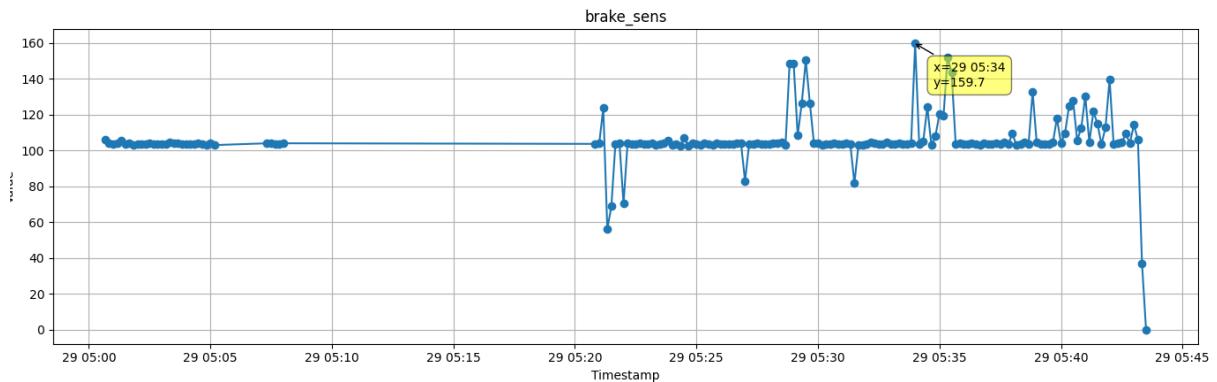


Figura 6.7: Datele grafice care funcționează corect.

## 6.2. Testarea și validarea distanței efective a sistemului

Pentru a testa distanța la care sistemul de telemetrie poate funcționa fără a compromite integritatea datelor, am folosit două laptop-uri, unul fiind transmîtătorul și unul fiind receptorul. Am mărit distanța gradual între noi până când datele nu mai erau integre și am ajuns la o distanță de aproximativ 300 de metri, observat în figura 6.9.

În figura 6.9 se observă cum a fost testată această funcționalitate, folosind Arduino Serial Monitor și funcțiile de debugging ale transmîtătorului și receptorului fiind

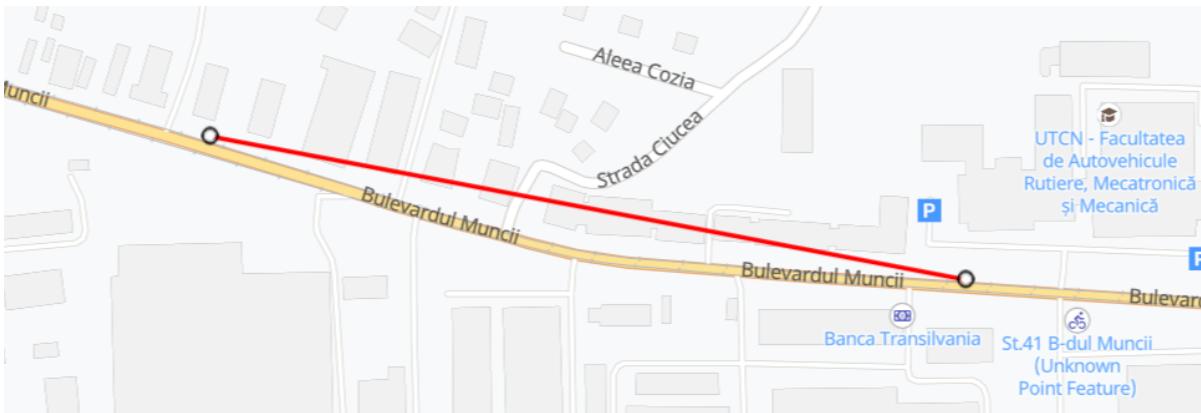


Figura 6.8: Distanța la care sistemul de telemetrie nu compromite datele.

activate pentru a vedea parametri ca SNR și RSSI, cât și faptul că encriptia și decriptia funcționează corect. Partea din stânga este transmițătorul, iar partea din dreapta este receptorul.

Figura 6.9: Modalitatea de testare a distanței sistemului de telemetrie.

### 6.3. Testarea și validarea vitezei de actualizare a interfeței grafice

Pentru a testa viteza de actualizare a interfeței grafice a utilizatorului, cât timp monopostul este static, am apăsat frâna și am cronometrat cât timp durează să se actualizeze. Rezultatul acestui este de aproximativ 2 secunde, care nu este ideal, dar trebuie luate și limitările legislative ale tehnologiei LoRa. Cât timp monopostul era pe circuit, această problemă nu a intervenit, fiind un timp de actualizare acceptabil.

Pentru a testa viteza de actualizare a interfeței grafice propriu-zise, în cadrul codului transmițătorului am integrat niște funcții de test, care dau fiecărui senzori valori de la 0-100 și biții fluctuează între 0 și 1 la fiecare secundă. Rezultatul este favorabil, interfața se actualizează la fiecare secundă, validând cerința.

Aici este codul de test:

```

//////////TEST VARIABLES
unsigned long lastSimUpdate = 0;
uint8_t simValue = 0;
bool simBit = false;
uint8_t msgTransmitSim[41];
//////////TEST VARIABLES

void simulateTelemetryData() {
    // all analog fields 0-100-0
    for (int i = 1; i < 27; ++i) {
        msgTransmitSim[i] = simValue;
    }

    // toggle bits ON or OFF every cycle
    if(simBit == 1){
        msgTransmitSim[0] = 255; // ventValue, pumpValue, flowValue (all bits)
        msgTransmitSim[27] = 255; // all ECU control bits
    } else {
        msgTransmitSim[0] = 0; // ventValue, pumpValue, flowValue (all bits)
        msgTransmitSim[27] = 0; // all ECU control bits
    }
    msgTransmitSim[28] = simValue; // VRef_purge
    msgTransmitSim[29] = simValue; // meas_purge
    msgTransmitSim[30] = simValue; // current_sensor
    msgTransmitSim[31] = simValue; // LV_state_of_charge

    msgTransmitSim[32] = simValue; // RPM_Speed (km/h)

    msgTransmitSim[33] = (int8_t)(simBit ? simValue : -simValue);
    msgTransmitSim[34] = simValue % 100;
    msgTransmitSim[35] = (int8_t)(simBit ? -simValue : simValue);
    msgTransmitSim[36] = (simValue * 2) % 100;
    msgTransmitSim[37] = (int8_t)(simValue / 2);
    msgTransmitSim[38] = (simValue * 3) % 100;
    msgTransmitSim[39] = (int8_t)(-simValue / 2);
    msgTransmitSim[40] = (simValue * 4) % 100;
}

```

#### 6.4. Testarea și validarea sistemului pe monopost

În timpul testelor monopostului, am testat și sistemul de telemetrie, care a funcționat din cele mai îndepărtate 2 puncte ale circuitului cu succes. Pentru a verifica integritatea datelor, am scris pe un card SD date înainte de transmitere și am stocat datele după transmisie, iar între aceste 2 nu s-au găsit diferențe, dovedind că integritatea datelor nu este compromisă la aproximativ 180 de metri, observat în figura 6.11. În figura 6.12, se poate vedea sistemul de telemetrie cât timp monopostul este pe circuit.

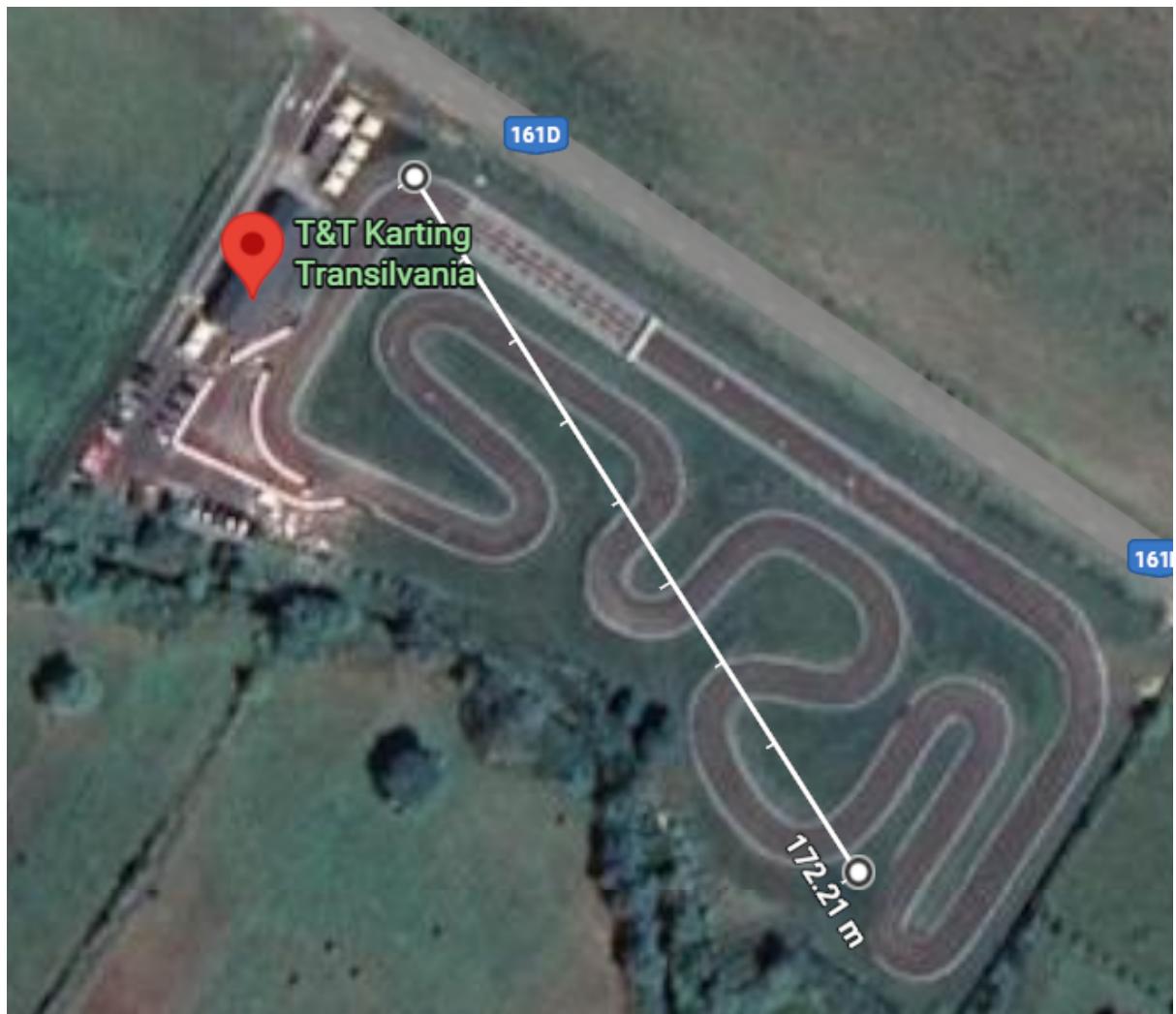


Figura 6.10: Distanța la care sistemul de telemetrie a mers fără probleme în timpul testării monopostului.



Figura 6.11: Sistemul de telemetrie funcțional.

## Capitolul 7. Manual de instalare și utilizare

### 7.1. Cerinte ale sistemului si resurse

### 7.1.1. Cerintele sistemului

Această parte a lucrării nu o să includă configurațiile necesare pentru transmițătorul de pe vehiculul electric, ci mai degrabă doar partea pentru a monitoriza datele pe partea utilizatorului. Specificațiile sistemului pe care a fost dezvoltat sistemul sunt următoarele, deși în procesul de proiectare una dintre paradigmile "plug-and-play" pe care am adoptat-o este că sistemul rulează pe orice sistem, indiferent de cerințele acestuia cu pași de instalare și configurare minimă:

- Procesor (CPU): AMD Ryzen 5 4600H, dar orice procesor care nu are aceeași putere poate rula acest sistem.
  - Memorie RAM: 8GB, dar deși sunt convins că sistemul nu are nevoie de atât, serverul beneficiază de orice resursă în plus a gazdei.
  - Tipul de sistem: Sistem de operare bazat pe 64 de biți, fiind găsit cu ușurință în zilele noastre.
  - Sistem de operare: Windows 11, iar limita inferioară este Windows 10 din moment ce multe aplicații nu mai sunt suportate de Microsoft după această versiune.

### 7.1.2. Resurse Hardware

Din moment ce acest proiect este embedded, vom avea nevoie de hardware suplimentar, acestea fiind un **Arduino MKR WAN 1310 cu antena aferentă** (Figura 7.1 și 7.2) și un **cablu MicroUSB** (Figura 7.3), ale căror poze le-am atașat mai jos.

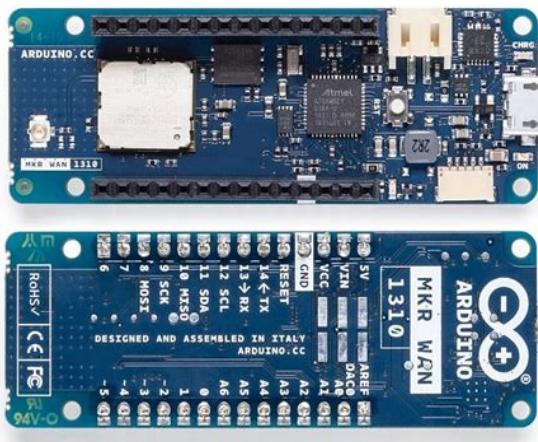


Figura 7.1: Plăcuța Arduino MKR WAN 1310 care este folosită [23]



Figura 7.2: Antena pentru plăcuța Arduino MKR WAN 1310 [26]



Figura 7.3: Cablul MicroUSB folosit pentru plăcuța Arduino MKR WAN 1310

### 7.1.3. Resurse Software

Pentru a putea avea un sistem de vizualizare, sunt folosite următoarele resurse software:

- Docker - Aceasta este aplicația care găzduiește aplicația, conținând baza de date și aplicația de vizualizare.
- Python - Este limbajul de programare și interpreterul care ajută optimă funcționare a aplicației, fiind nevoie de el pentru a o putea rula.
- Windows Powershell - Este programul care interpretează linile de comandă și permite utilizatorului să configureze aplicația dintr-un singur loc.

## 7.2. Instalarea resurselor software

Pentru partea hardware nu este nevoie de un proces de instalare, plăcuța Arduino MKR WAN 1310 fiind pre-încărcată cu codul necesar, iar pentru Windows Powershell nu

este nevoie de un proces de instalare pentru că acesta vine pre-instalat cu Windows. Pentru partea software, trebuie urmați următorii pași:

### 7.2.1. Instalare Docker

- Pasul 1: Se accesează linkul <https://www.docker.com/products/docker-desktop> și se alege sistemul de operare aferent. (Figura 7.4)

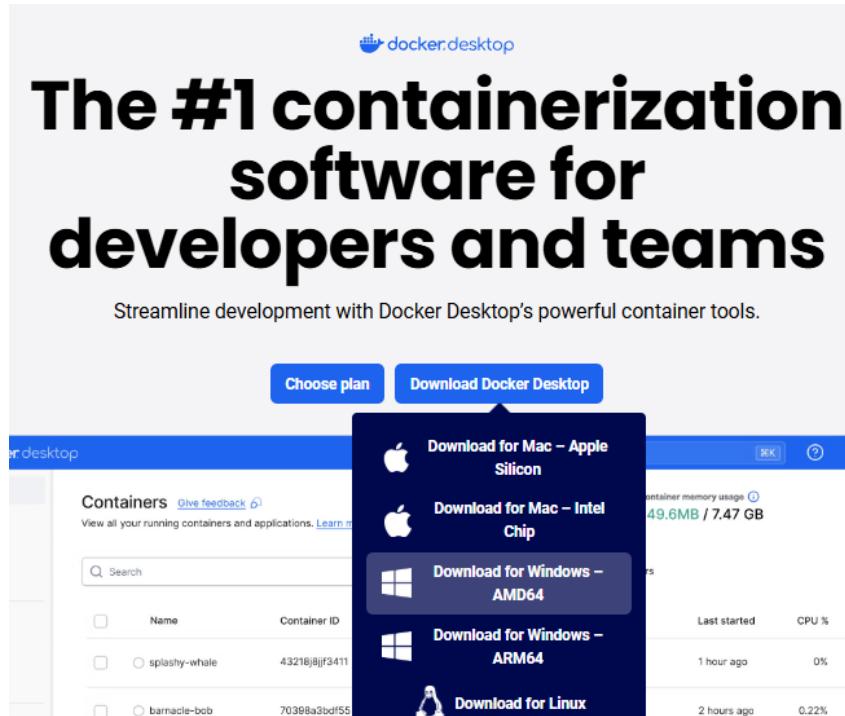


Figura 7.4: Descărcarea Docker pentru sistemul de operare Windows cu procesor AMD.

- Pasul 2: Se urmează pașii descriși în installer după ce acesta este accesat după descărcare, fiind un proces simplu.
- Pasul 3: După instalare, se rulează Docker Desktop și utilizatorul se asigură că funcționează în background. Pentru a verifica funcționalitatea Docker, se poate introduce comanda "docker ps" pentru a vedea acest răspuns, ilustrat în figura 7.5:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

Figura 7.5: Text afișat care arată că Docker funcționează.

### 7.2.2. Instalare Python

- Pasul 1: Se accesează linkul <https://www.python.org/downloads/> și se alege sistemul de operare aferent, la fel ca la Docker, ilustrat în figura 7.6.



Figura 7.6: Descărcare Python pentru sistemul de operare Windows.

2. Pasul 2: Se urmează pașii descriși în installer după ce acesta este accesat după descărcare, fiind un proces la fel de simplu ca la Docker.
3. Pasul 3: După instalare, utilizatorul va deschide un terminal Windows Powershell pentru a verifica funcționalitatea Python și va scrie comanda "python --version" observând următorul rezultat, prezentat în figura 7.7:

A black terminal window with white text displaying "Python 3.12.2".

Figura 7.7: Text afișat care arată ca Python a fost instalat corect.

### 7.3. Instalarea și utilizarea aplicației

Datorită design-ului "plug-and-play", utilizatorul nu trebuie să configureze nimic în ceea ce privește aplicația. Se navighează la folderul "Docker Components" al proiectului, apasă click dreapta și apoi "Open in Terminal".

Utilizatorul scrie comanda "docker compose up -d" și așteaptă până când sistemul își face configurările inițiale. La final, acest text trebuie să apară în linia de comandă, observat în figura 7.8:

[+] Running 4/4	
✓ Network tig-stack-clean_default	Created
✓ Container tig-stack-clean-influxdb-1	Started
✓ Container tig-stack-clean-upload_from_influx_to_firestore-1	Started
✓ Container tig-stack-clean-grafana-1	Started

Figura 7.8: Text afișat care arată că sistemul a fost inițializat corect.

Următorul pas care trebuie făcut de utilizator este să conecteze plăcuța Arduino MKR WAN 1310 la laptop-ul său printr-un port USB și apoi în aceeași linie de comandă scrie "python transferScript.py", ducându-l la acest meniu:

Și următorul pas este completarea căsuței cu "COM [Număr detectat]" pentru a realiza conexiunea, ca în pozele de mai jos (Figura 7.9 și 7.10):

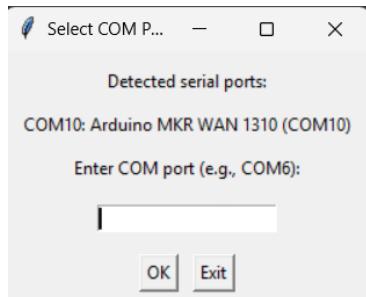


Figura 7.9: Pop-up TransferScript Necompletat.

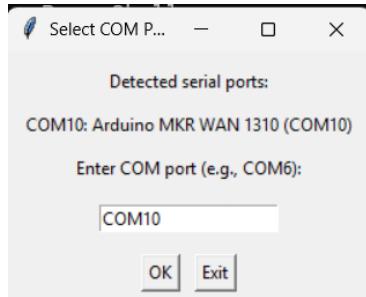


Figura 7.10: Pop-up TransferScript completat.

Când apare mesajul "Connected at 9600 baud", utilizatorul așteaptă până se primesc date, observat în figura 7.11.

```
Connected to COM10 at 9600 baud.
Received START INFLUX
Sent to InfluxDB: telemetry_data,location=test,device=senzori cooling=0,brake_sens=16,steer_sens=16,acc1=16,acc2=16,wheel_spin_1=4112,wheel_spin_2=4112,motor_temp_1=16,motor_temp_2=16,control_temp_1=16,control_temp_2=16,serial_throttle_1=16,serial_throttle_2=16,inputV1_p=41.12,inputV2_p=41.12,phasecurrent1=41.12,phasecurrent2=41.12,susp_travel_FL=16,susp_travel_FR=16,susp_travel_BL=0,susp_travel_BR=16,ECU_control=16,VRef_precharge=4112,meas_precharge=0,current_sensor=46260,LV_state_of_charge=0,ventValue=0,pumpValue=0,fFlowValue=0.00,brakeEngaged=0,soundPlaying=0,R2D_Button_State=0,airPlusValue=1,airMinusValue=0,prechgValue=0,SDC_END=0,Measure_Digital=0,accelX=11.29,accelY=-15.28,roll=102.69,pitch=0.01,yaw=0.00,IMU_Speed=119,rawTemp1=0,rawTemp2=0
URL: http://localhost:8086 | Org: telemetryARTTU | Bucket: telemetryDATA
```

Figura 7.11: Windows Powershell când conexiunea este realizată cu succes și se primesc date.

Apoi, utilizatorul deschide un browser la alegera lui (Ex. Google Chrome sau Mozilla Firefox) și introduce adresa "localhost:3000" pentru a accesa aplicația, care îl duce la acest meniu:

Apoi, utilizatorul va alege unul dintre dashboard-uri iar acesta este sistemul care funcționează corect (Figura 7.12 și 7.13):

### 7.3 Instalarea și utilizarea aplicației

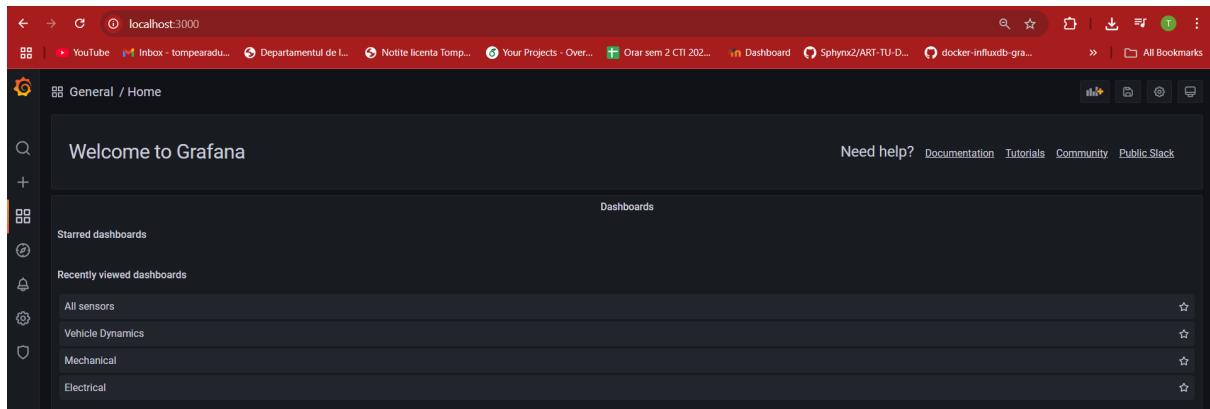


Figura 7.12: Pagina inițială a sistemului.



Figura 7.13: Sistemul funcționând corect, meniul All Sensors

După ce monopostul nu mai este operațional, utilizatorul trebuie să urce datele pe Cloud, asta realizându-se prin deschiderea unui Windows Powershell și scrierea comenzi "python uploadData.py", ducându-l la meniul ilustrat în figura 7.14.

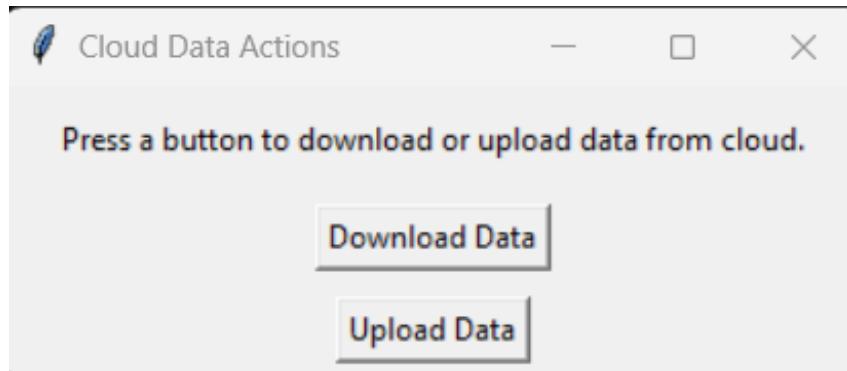


Figura 7.14: Meniul pentru urcarea datelor recepționate.

Butonul "Upload Data" urcă datele pe Cloud, iar butonul "Download Data" deschide un Windows Powershell și se va scrie comanda "python csvDataPlotter.py". Utilizatorul alege setul de date descărcat de pe Cloud și poate accesa aceste date vizualizate ca un grafic, observat în figura 7.15.

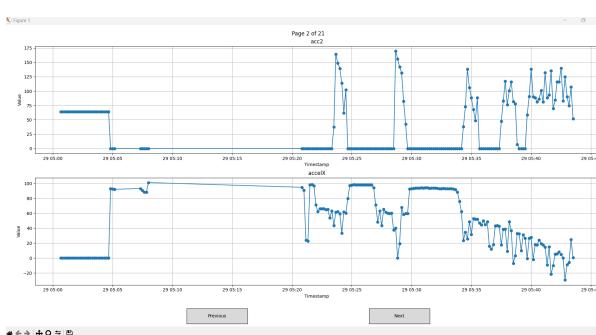


Figura 7.15: Vizualizarea datelor după ce sunt descărcate.

Pentru ca alți utilizatori să se poată conecta la server, acestea trebuie să fie în același rețea. Cel mai ușor mod de a realiza acest lucru e prin creezearea unui hotspot mobil și conectarea tuturor utilizatorilor la aceasta.

Următorul pas este de a da adresa IP locală, iar pentru aceasta utilizatorul care este server trebuie să deschidă Windows Powershell și să tasteze comanda "ipconfig" și să dezvăluie celorlalți utilizatori adresa IP, cea din figura 7.16:

```
Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix . . . .
Link-local IPv6 Address . . . . . : fe80::799b:3d9f:6843:2e69%6
IPv4 Address . . . . . : 192.168.0.127
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.0.1
```

Figura 7.16: Textul care arată unde se găsește adresa IP locală.

Ultimul pas, după ce utilizatorul este în același rețea ca și nodul server este să introducă în browser adresa IPV4 și să acceseze aplicația.

## **Capitolul 8. Concluzii**

Lucrarea prezentată arată doar una din multele de soluții care se pot aplica în lumea automotive pentru a monitoriza datele unui vehicul, iar soluția descrisă fiind una rar întâlnită, oferind un potențial mare de explorare. Scopul principal al acestei lucrări a fost să proiecteze, dezvolte și să testeze un sistem de monitorizare a datelor în monopostul ART TU E17 utilizând tehnologia LoRa pentru a transmite datele acestuia pe o rază posibilă de un kilometru.

În ceea ce privește atingerea obiectivelor enunțate în capitolul 2, am reușit în lucrarea mea să aduc un rezultat la următoarele:

- Principalul obiectiv al lucrării este de a putea transmite datele monopostului cât timp acesta este operațional, în ciuda impedimentelor la nivel de hardware sau din lipsa conexiunii la internet, dezvoltând un sistem care poate susține un flux ridicat de date la o distanță mare. La finalul acestei lucrări, am atins acest obiectiv crucial, având oportunitatea de a testa și de a valida sistemul pe monopost în timpul sesiunilor de testare.
- Printre obiectivele secundare ale acestei lucrări se numără identificarea datelor relevante, filtrând doar ce este necesar echipei de dezvoltare pentru a monitoriza starea monopostului în vederea eficienței și performanței acestuia. Am atins acest obiectiv, observându-se prin diagramele de clase care sunt prezente în lucrare și găsirea unei soluții pentru a transmite și stoca toate datele necesare folosind tehnologiile prezentate într-un mod eficient.
- Următorul obiectiv a fost de a defini și implementa o structură de date care este optimă la nivel de hardware și retine toate datele identificate la punctul anterior, fiind atins atât la legătura dintre monopost și sistemul de telemetrie cât și între transmițător și receptor, stabilind astfel un standard al datelor transmise.
- Pentru implementarea funcționalităților la nivel de server am reușit să identific și să implementez o soluție care este un echilibru între stocarea datelor cât și viteza de răspuns a sistemului, fiind una elegantă și eficientă.
- Persistența datelor în medii dificile a fost în sinea ei o sarcină dificilă, dar prin validarea sistemului în cadrul sesiunilor de testare a monopostului constat că am atins acest obiectiv.
- Cât despre accesarea și vizualizarea datelor după sesiunea de testare, folosind tehnologii de ora actuală am reușit să implementez o soluție care satisface nevoile echipei într-un mod intuitiv și eficient, fără costuri suplimentare aduse.

În ceea ce privește cerințele funcționale prezentate în capitolul 2, satisfacerea acestora a fost detaliată pe parcursul lucrării, fiind doar una singură care nu am reușit să o ating, fiind cea de monitorizare a datelor cu un timp de actualizare de o secundă, sistemul actualizându-se la 3 secunde. Această cerință nu este imposibil de realizat, doar sunt multe decizii care eu consider că le-am luat în cel mai bun interes al echipei ART TU. Pentru cerințele nefuncționale, am atins majoritatea, dar întotdeauna se poate îmbunătății, spre exemplu o securitate mai ridicata a datelor, atât în tranzit cât și staționare, iar pentru partea de design a interfeței de utilizator oricând se pot face modificări pentru oricine îl folosește, dar sistemul este "plug-and-play" iar orice modificare adusă acestuia

poate fi implementată cu ușurință.

Această lucrare abordează o soluție în realizarea unui sistem de transmitere a datelor pe o distanță mare, un lucru crucial în mai multe domenii de cercetare precum cel militar, automotive și în natură. Scopul acestei lucrări este de a arăta o abordare diferită asupra soluțiilor deja existente, folosind tehnologii actuale într-o lumină nouă. Progresele imense în industria automotive, atât pe vehicule bazate pe combustie internă, sisteme hybrid și mai ales cele electrice arată importanța unui astfel de sistem. Toată lumea beneficiază de un astfel de sistem, de la producător la consumator, iar pentru a avea cel mai bun sistem de telemetrie în mașini, la nivelul industriei, trebuie să se stabilească un standard de identificare și transport al datelor pentru a putea dezvolta un sistem flexibil, stabil, permanent și securizat pentru a putea depăși provocările domeniului automotive. Acestea fiind spuse, lucrarea mea prezintă o soluție care arată o perspectivă a cum poate arăta un sistem de telemetrie care funcționează la toți producătorii de autovehicule, în speranța unei soluții open-source, pentru toată lumea. Deși proiectul este proiectat și implementat pentru cazul specific al monopostului ART TU E17, tema este una foarte relevantă și are un potențial de explorare imens.

În ciuda celor spuse, proiectul meu nu este perfect, având nevoie de câteva modificări și dezvoltări pentru a putea fi folosit în producție:

- Principala modificare care i-aș aduce-o sistemului de telemetrie prezentat este găsirea unui mod de a transmite mai multe date prin intermediul LoRa cu un timp minim de așteptare, fiind un factor limitator în ceea ce privește câte date se pot trimite într-o anumită fază de timp.
- A doua modificare pe care aș aduce-o este dedicarea de hardware pentru server. În loc de orice laptop care funcționează să actioneze ca un server, aș dedica un laptop care singurul lui lor este de a ruta date și de a permite utilizatorilor să se conecteze la el, eliminând decizia de "al cui laptop vrea să fie server".
- În loc de o rețea locală care este hostată de server, aș vrea să implementez soluția mea pe un site cu un domeniu accesibil tuturor, făcând mai facil accesul la datele care sunt salvate în cloud.
- Pentru ca sistemul de telemetrie să beneficieze de o distanță cât mai mare a datelor transmise, mi-ar plăcea să găsesc o configurație a mediului de transmisie care atinge balansul între distanță și integritatea datelor.
- Fiind un proiect embedded, întotdeauna este o idee bună să folosești hardware mai bun, componente care se potrivesc cerințelor proiectului mai bine. Fiindcă costul a fost un factor limitator, acest sistem ar beneficia imens de hardware mai specializat pentru transmiterea și recepționarea de date.
- Spațiul este limitat pe monopostul ART TU E17, și un improvement ideal este scăderea dimensiunii și greutății sistemului, realizabilă prin desemnarea unui PCB pentru acest scop.

În concluzie, lucrarea mea prezintă o soluție la problema transmiterii datelor în domeniul automotive printr-o perspectivă care nu este aşa de des întâlnită în literatură. Pe viitor, această perspectivă merita studiată și testată mai mult decât este descris în această lucrare. Domeniul automotive este în continuă creștere, iar integrarea tehnologiilor noi în vehiculele de mâine este cel mai logic pas, dar fără niște standarde clar definite, acest lucru va îngreuna considerabil procesul de a ne adapta la noile probleme apărute.

## Bibliografie

- [1] TechTarget, “Telemetry definition,” <https://www.techtarget.com/whatis/definition/telemetry>.
- [2] F. Carden, R. P. Jedlicka, and R. Henry, *Telemetry Systems Engineering*, revised ed. Artech House, 2002.
- [3] Formula Student Germany, “FS-Rules 2025,” 2025.
- [4] D. Alford, “Lightweight, low cost, automotive data acquisition and telemetry system,” Master’s thesis, University of Cincinnati, Cincinnati, OH, USA, 2005, department of Mechanical, Industrial, and Nuclear Engineering.
- [5] J. R. Chandiramani, S. Bhandari, and H. S. A., “Vehicle data acquisition and telemetry,” in *Proceedings of the 2014 Fifth International Conference on Signal and Image Processing (ICSIP)*, 2014.
- [6] A. Rodgers, “Recent telemetry technology,” in *Radio Tracking and Animal Populations*, J. J. Millspaugh and J. M. Marzluff, Eds. Academic Press, 2001.
- [7] J. K. Roy, “An introduction to telemetry. part 1: Telemetry basics,” Online PDF, Self-published online, Tech. Rep., 2012, <http://dr-joyanta-kumar-roy.com/study-material/Telemetry%20systems/Telemetry%20basics.pdf>.
- [8] S. Gyawali, S. Xu, Y. Qian, and R. Hu, “Challenges and solutions for cellular based v2x communications,” *IEEE Communications Surveys & Tutorials*, 2021.
- [9] A. Ghosal and M. Conti, “Security issues and challenges in v2x: A survey,” *Computer Networks*, Mar. 2020.
- [10] H. Kraiem, M. Benrejeb, M. Benrejeb, and M. B. M. Benrejeb, “Improving electric vehicle autonomy in the smart city concept,” *International Journal of Energy Research*, 2019.
- [11] M. A. Hossain, M. R. Islam, and M. A. Rahman, “Design and analysis of a novel power management approach for energy-efficient vehicular networks,” *International Journal of Energy Research*, 2014.
- [12] M. C. Bor, J. E. Vidler, and U. Roedig, “Lora for the internet of things,” in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2016.
- [13] S. Devalal and A. Karthikeyan, “Lora technology, an overview,” in *2018 2nd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 2018.
- [14] LoRa Documentation Contributors, “LoRa Documentation – Range vs Power,” <https://lora.readthedocs.io/en/latest/#range-vs-power>, 2021.

- [15] S. Corrigan, “Introduction to the controller area network (can),” Texas Instruments, Application Report, SLOA101B, 2016. [Online]. Available: <https://www.ti.com/lit/an/sloa101b/sloa101b.pdf>
- [16] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, “Trends in automotive communication systems,” *Proceedings of the IEEE*, 2005.
- [17] M. Dworkin, “Advanced encryption standard (aes),” <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>, 2001.
- [18] ——, “Recommendation for block cipher modes of operation: Methods and techniques,” <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>, 2001.
- [19] InfluxData, *InfluxDB 3 Core Documentation*, 2025, available at <https://docs.influxdata.com/influxdb3/core/>.
- [20] *About Grafana*, <https://grafana.com/docs/grafana/latest/introduction/>, Grafana Labs, 2025.
- [21] *Firebase Admin Python SDK Documentation*, <https://firebase.google.com/docs/reference/admin/python>, Google LLC, 2025.
- [22] DockerPros, “What is a namespace in Docker?” <https://dockerpros.com/introduction-to-docker/what-is-a-namespace-in-docker/>, 2025.
- [23] Arduino, “Arduino mkr wan 1310 (without antenna) datasheet,” [https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/337/ABX00029\\_MKR\\_WAN\\_1310\\_wo\\_Antenna.pdf](https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/337/ABX00029_MKR_WAN_1310_wo_Antenna.pdf).
- [24] Microchip Technology Inc., *SAM D21 Family Data Sheet (DS40001882D) – Low-Power, 32-bit Cortex-M0+ MCU with Advanced Analog and PWM*, [https://content.arduino.cc/assets/mkr-microchip\\_samd21\\_family\\_full\\_datasheet-ds40001882d.pdf](https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf), 2018.
- [25] ——, *MCP2515 Family Data Sheet (DS20001801K)*, <https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MCP2515-Family-Data-Sheet-DS20001801K.pdf>.
- [26] 2J Antennas, *2JF0415P Dual-Band 868/915 MHz Flexible PCB (U.FL) Antenna Datasheet*, <https://www.2j-antennas.com/media/original/datasheets/2jf0415p.pdf>.
- [27] S. Yi, C. Li, and Q. Li, “A survey of fog computing: Concepts, applications and issues,” in *Proceedings of the 2015 Workshop on Mobile Big Data (Mobidata'15)*, Hangzhou, China, 2015.
- [28] V. B. Ristić, B. M. Todorović, and N. M. Stojanović, “Frequency hopping spread spectrum: History, principles and applications,” *Vojnotehnički Glasnik*, 2022.
- [29] O. Vacek, “Formula-student-telemetry,” <https://github.com/GKPr0/Formula-Student-Telemetry>, 2023.

- [30] N. Braune, “Telemetry unit for a formula student race car,” <https://pub.tik.ee.ethz.ch/students/2013-HS/SA-2013-67.pdf>, Institute of Technical Informatics and Communication Networks, ETH Zürich, Tech. Rep., 2013.

## Anexa A. Secțiuni relevante din cod

### A.1. Transmițător LoRa cu AES-128 CTR

```
#include <Arduino.h>
#include <RadioLib.h>
#include "can.h"
#include "LoraRadio.h"
#include "aes.h"

unsigned long startTime = millis(), currentTime = 0;

CAN can;
LoRaRadio lora;

byte aes_key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                  0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F }; // 16 bytes for AES-128
byte aes_iv[] = { 0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
                  0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF }; // 16 bytes IV for AES-128

//////////////////////////////TEST VARIABLES
unsigned long lastSimUpdate = 0;
uint8_t simValue = 0;
bool simBit = false;
uint8_t msgTransmitSim[41];
//////////////////////////////TEST VARIABLES

// struct the easy acces to data
struct TelemetryData
{
    uint8_t cooling;
    uint8_t brake_sens;
    uint8_t steer_sens;
    uint8_t acc1;
    uint8_t acc2;
    uint16_t wheel_spin_1;
    uint16_t wheel_spin_2;
    uint8_t motor_temp_1;
    uint8_t motor_temp_2;
    uint8_t control_temp_1;
    uint8_t control_temp_2;
    uint8_t serial_throttle_1;
```





```

// set settings for lora modulation
lora.setSettings2();

// set the function that will be called
// when new packet is received (both can and lora)
radio.setDio0Action(setFlag, RISING);
Serial.println("interrupt funcs set");

// init can module
can.init();

}

void loop()
{
    currentTime = millis();
    blink_led(1000);

    // // process ISO-TP receive only
    // can.isotp.receive(&can.receivedMsg);

    // if(can.receivedMsg.tp_state == ISOTP_FINISHED) {
    //     Serial.println();
    //     Serial.print("Received msg: ");
    //     for(int i=0;i<can.receivedMsg.len;i++) {
    //         Serial.print(can.receivedMsg.Buffer[i]);
    //         Serial.print(" ");
    //     }
    //     Serial.println();

    //     // AES CTR ENCRYPTION
    //     struct AES_ctx ctx;
    //     AES_init_ctx_iv(&ctx, aes_key, aes_iv);
    //     uint8_t encryptedCAN[can.receivedMsg.len];
    //     memcpy(encryptedCAN, can.receivedMsg.Buffer, can.receivedMsg.len);
    //     AES_CTR_xcrypt_buffer(&ctx, encryptedCAN, can.receivedMsg.len);

    //     lora.sendData(encryptedCAN, can.receivedMsg.len);
    //     transmitFlag = true;

    //     can.receivedMsg.tp_state = ISOTP_IDLE;
    // }
    // if(!digitalRead(INTCAN)) can.receive_normal_message();

    // send a message every 3s, with AES-128 CTR encryption
    if(currentTime - startTime >= 3000) {
        if (simValue >= 100) {

```

```

        simValue = 0;
    } else {
        simValue++;
        simBit = !simBit; // toggle all bits every cycle
    }
simulateTelemetryData();
//AES CTR ENCRYPTION
struct AES_ctx ctx;
AES_init_ctx_iv(&ctx, aes_key, aes_iv);

// uint8_t encryptedMsg[sizeof(msgTransmit)];
// memcpy(encryptedMsg, msgTransmit, sizeof(msgTransmit)); -----FOR CAN DATA

uint8_t encryptedMsg[sizeof(msgTransmitSim)];
memcpy(encryptedMsg, msgTransmitSim, sizeof(msgTransmitSim));

Serial.print("Original: ");
for (size_t i = 0; i < sizeof(msgTransmitSim); i++) {
    Serial.print(msgTransmitSim[i]);
    Serial.print(" ");
}
Serial.println();

AES_CTR_xcrypt_buffer(&ctx, encryptedMsg, sizeof(encryptedMsg));

Serial.print("Encrypted: ");
for (size_t i = 0; i < sizeof(encryptedMsg); i++) {
    Serial.print(encryptedMsg[i]);
    Serial.print(" ");
}
Serial.println();

lora.sendData(encryptedMsg, sizeof(encryptedMsg));
startTime = currentTime;
// radio.sleep();
}

// interrupt based handling
// when flag is set, that means data was sent/received
if (operationDone)
{
    // reset flag
    operationDone = false;

    if (transmitFlag)
    {
        // the previous operation was transmission
        if (transmissionState == RADIOLIB_ERR_NONE)

```

```

    {
        // packet was successfully sent
        Serial.println(F("package transmission finished!"));
    }
    else
    {
        Serial.print(F("failed, code "));
        Serial.println(transmissionState);
    }

    transmitFlag = false;
}
}
}

```

## A.2. Receptor LoRa cu AES-128 CTR.

```

#include <Arduino.h>
#include <RadioLib.h>
#include "can.h"
#include "LoraRadio.h"
#include "aes.h"

unsigned long startTime = millis(), currentTime = 0;

CAN can;
LoRaRadio lora;

byte aes_key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                  0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };
byte aes_iv[] = { 0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5, 0xA6, 0xA7,
                  0xA8, 0xA9, 0xAA, 0xAB, 0xAC, 0xAD, 0xAE, 0xAF };

//////////////////////////////TEST VARIABLES
unsigned long lastSimUpdate = 0;
uint8_t simValue = 0;
bool simBit = false;
uint8_t msgTransmitSim[41];
//////////////////////////////TEST VARIABLES

// struct the easy acces to data
struct TelemetryData
{
    uint8_t cooling;
    uint8_t brake_sens;
    uint8_t steer_sens;
    uint8_t acc1;

```

```
uint8_t acc2;
uint16_t wheel_spin_1;
uint16_t wheel_spin_2;
uint8_t motor_temp_1;
uint8_t motor_temp_2;
uint8_t control_temp_1;
uint8_t control_temp_2;
uint8_t serial_throttle_1;
uint8_t serial_throttle_2;
float inputV1_p;
float inputV2_p;
float phasecurrent1;
float phasecurrent2;
uint8_t susp_travel_FL;
uint8_t susp_travel_FR;
uint8_t susp_travel_BL;
uint8_t susp_travel_BR;
uint8_t ECU_control;
uint16_t VRef_purge;
uint16_t meas_purge;
uint16_t current_sensor;
uint16_t LV_state_of_charge;

//extracted boolean/packed fields
uint8_t ventValue;
uint8_t pumpValue;
float flowValue;

uint8_t brakeEngaged;
uint8_t soundPlaying;
uint8_t R2D_Button_State;
uint8_t airPlusValue;
uint8_t airMinusValue;
uint8_t prechgValue;
uint8_t SDC_END;
uint8_t Measure_Digital;

uint8_t IMU_Speed;
float accelX_IMU;
float accelY_IMU;
float pitch_IMU;
float roll_IMU;
float yaw_IMU;

uint16_t rawTemp1;
uint16_t rawTemp2;
};
```

```

TelemetryData dataToSend;

//////////////////////////////TEST FUNCTIONS

void simulateTelemetryData() {
    // analog-like fields 0-100-0
    for (int i = 1; i < 27; ++i) {
        msgTransmitSim[i] = simValue;
    }

    // toggle all bits ON or OFF every cycle
    if(simBit == 1){
        msgTransmitSim[0] = 255; // ventValue, pumpValue, flowValue (all bits)
        msgTransmitSim[27] = 255; // all ECU control bits
    } else {
        msgTransmitSim[0] = 0; // ventValue, pumpValue, flowValue (all bits)
        msgTransmitSim[27] = 0; // all ECU control bits
    }
    msgTransmitSim[28] = simValue; // VRef_purge
    msgTransmitSim[29] = simValue; // meas_purge
    msgTransmitSim[30] = simValue; // current_sensor
    msgTransmitSim[31] = simValue; // LV_state_of_charge

    msgTransmitSim[32] = simValue; // RPM_Speed (km/h)

    msgTransmitSim[33] = (int8_t)(simBit ? simValue : -simValue); // accelX_int
    msgTransmitSim[34] = simValue % 100; // accelX_frac

    msgTransmitSim[35] = (int8_t)(simBit ? -simValue : simValue); // accelY_int
    msgTransmitSim[36] = (simValue * 2) % 100; // accelY_frac

    msgTransmitSim[37] = (int8_t)(simValue / 2); // roll_int
    msgTransmitSim[38] = (simValue * 3) % 100; // roll_frac

    msgTransmitSim[39] = (int8_t)(-simValue / 2); // pitch_int
    msgTransmitSim[40] = (simValue * 4) % 100;
}

//////////////////////////////TEST FUNCTIONS

float unpackFlowLps(uint8_t packed) {
    float max_lps = 10.0f / 60.0f;
    return packed * (max_lps / 255.0f);
}

void convertToInfluxDB(void){
    dataToSend.cooling = receivedMsg[0]; // vent/pump/flow packed
    dataToSend.brake_sens = receivedMsg[1]; // valueBrake
}

```

```

dataToSend.steer_sens = receivedMsg[2];           // valueSteering
dataToSend.acc1 = receivedMsg[3];                 // pos1
dataToSend.acc2 = receivedMsg[4];                 // pos2

// Wheel RPMs
dataToSend.wheel_spin_1 = (receivedMsg[5]) | (receivedMsg[6] << 8); // controllerL
dataToSend.wheel_spin_2 = (receivedMsg[7]) | (receivedMsg[8] << 8); // controllerR

// Temperatures
dataToSend.motor_temp_1 = receivedMsg[9];         // controllerDataL.tempMotor
dataToSend.motor_temp_2 = receivedMsg[10];          // controllerDataR.tempMotor
dataToSend.control_temp_1 = receivedMsg[11];        // controllerDataL.tempControl
dataToSend.control_temp_2 = receivedMsg[12];        // controllerDataR.tempControl

// Throttle voltages (truncated integer values)
dataToSend.serial_throttle_1 = (uint8_t)(receivedMsg[13] | (receivedMsg[14] << 8));
dataToSend.serial_throttle_2 = (uint8_t)(receivedMsg[15] | (receivedMsg[16] << 8));

// Supply voltages (scaled int16_t, divide by 100)
int16_t inputV1_raw = (int16_t)(receivedMsg[17] | (receivedMsg[18] << 8));
int16_t inputV2_raw = (int16_t)(receivedMsg[19] | (receivedMsg[20] << 8));
dataToSend.inputV1_p = inputV1_raw / 100.0f;
dataToSend.inputV2_p = inputV2_raw / 100.0f;

// Phase currents (scaled int16_t, divide by 100)
int16_t phase1_raw = (int16_t)(receivedMsg[21] | (receivedMsg[22] << 8));
int16_t phase2_raw = (int16_t)(receivedMsg[23] | (receivedMsg[24] << 8));
dataToSend.phasecurrent1 = phase1_raw / 100.0f;
dataToSend.phasecurrent2 = phase2_raw / 100.0f;

// Suspension travel
dataToSend.susp_travel_FL = receivedMsg[25];
dataToSend.susp_travel_FR = receivedMsg[26];
dataToSend.susp_travel_BL = receivedMsg[27];
dataToSend.susp_travel_BR = receivedMsg[28];

// Packed ECU control bits
dataToSend.ECU_control = receivedMsg[29];
dataToSend.VRef_precharge = (uint16_t)(receivedMsg[30] | (receivedMsg[31] << 8));
dataToSend.current_sensor = (uint16_t)(receivedMsg[32] | (receivedMsg[33] << 8));
dataToSend.meas_precharge = (uint16_t)(receivedMsg[49] | (receivedMsg[50] << 8));
dataToSend.LV_state_of_charge = receivedMsg[51]; // state of charge in %

// Wheel speed (km/h)
dataToSend.IMU_Speed = receivedMsg[34];

dataToSend.accelX_IMU = (int16_t)(receivedMsg[35] | (receivedMsg[36] << 8)) / 32768;

```

```

dataToSend.accelY_IMU = (int16_t)(receivedMsg[37] | (receivedMsg[38] << 8)) / 32768

dataToSend.roll_IMU = (int16_t)(receivedMsg[39] | (receivedMsg[40] << 8)) / 32768

dataToSend.pitch_IMU = (int16_t)(receivedMsg[41] | (receivedMsg[42] << 8)) / 32768

dataToSend.rawTemp1 = (uint16_t)(receivedMsg[43] | (receivedMsg[44] << 8));
dataToSend.rawTemp2 = (uint16_t)(receivedMsg[45] | (receivedMsg[46] << 8));

dataToSend.yaw_IMU = (int16_t)(receivedMsg[47] | (receivedMsg[48] << 8)) / 32768.0

// ----- Bit-unpack logic -----
dataToSend.ventValue      = (receivedMsg[0] >> 7) & 0x01;
dataToSend.pumpValue      = (receivedMsg[0] >> 6) & 0x01;
dataToSend.flowValue      = unpackFlowLps((receivedMsg[0] & 0x3F)<<2); // unpacked

dataToSend.brakeEngaged   = (receivedMsg[29] >> 7) & 0x01;
dataToSend.soundPlaying    = (receivedMsg[29] >> 6) & 0x01;
dataToSend.R2D_Button_State = (receivedMsg[29] >> 5) & 0x01;
dataToSend.airPlusValue    = (receivedMsg[29] >> 4) & 0x01;
dataToSend.airMinusValue   = (receivedMsg[29] >> 3) & 0x01;
dataToSend.prechgValue     = (receivedMsg[29] >> 2) & 0x01;
dataToSend.SDC_END         = (receivedMsg[29] >> 1) & 0x01;
dataToSend.Measure_Digital = (receivedMsg[29] >> 0) & 0x01;
}

void sendToInfluxDB()
{
    // telemetry received array to struct
    convertToInfluxDB();

    // formatting the data in line protocol for InfluxDB
    Serial.println("START INFLUX");
    Serial.print("telemetry_data,location=test,device=senzori ");

    Serial.print("cooling="); Serial.print(dataToSend.cooling); Serial.print(",");
    Serial.print("brake_sens="); Serial.print(dataToSend.brake_sens); Serial.print(",");
    Serial.print("steer_sens="); Serial.print(dataToSend.steer_sens); Serial.print(",");
    Serial.print("acc1="); Serial.print(dataToSend.acc1); Serial.print(",");
    Serial.print("acc2="); Serial.print(dataToSend.acc2); Serial.print(",");
    Serial.print("wheel_spin_1="); Serial.print(dataToSend.wheel_spin_1); Serial.print(",");
    Serial.print("wheel_spin_2="); Serial.print(dataToSend.wheel_spin_2); Serial.print(",");
    Serial.print("motor_temp_1="); Serial.print(dataToSend.motor_temp_1); Serial.print(",");
    Serial.print("motor_temp_2="); Serial.print(dataToSend.motor_temp_2); Serial.print(",");
    Serial.print("control_temp_1="); Serial.print(dataToSend.control_temp_1); Serial.print(",");
    Serial.print("control_temp_2="); Serial.print(dataToSend.control_temp_2); Serial.print(",");
    Serial.print("serial_throttle_1="); Serial.print(dataToSend.serial_throttle_1); Serial.print(",")
}

```

```

Serial.print("serial_throttle_2="); Serial.print(dataToSend.serial_throttle_2); S
Serial.print("inputV1_p="); Serial.print(dataToSend.inputV1_p); Serial.print(",");
Serial.print("inputV2_p="); Serial.print(dataToSend.inputV2_p); Serial.print(",");
Serial.print("phasecurrent1="); Serial.print(dataToSend.phasecurrent1); Serial.pr
Serial.print("phasecurrent2="); Serial.print(dataToSend.phasecurrent2); Serial.pr
Serial.print("susp_travel_FL="); Serial.print(dataToSend.susp_travel_FL); Serial.p
Serial.print("susp_travel_FR="); Serial.print(dataToSend.susp_travel_FR); Serial.p
Serial.print("susp_travel_BL="); Serial.print(dataToSend.susp_travel_BL); Serial.p
Serial.print("susp_travel_BR="); Serial.print(dataToSend.susp_travel_BR); Serial.p
Serial.print("ECU_control="); Serial.print(dataToSend.ECU_control); Serial.print(
Serial.print("VRef_precharge="); Serial.print(dataToSend.VRef_precharge); Serial.p
Serial.print("meas_precharge="); Serial.print(dataToSend.meas_precharge); Serial.p
Serial.print("current_sensor="); Serial.print(dataToSend.current_sensor); Serial.p
Serial.print("LV_state_of_charge="); Serial.print(dataToSend.LV_state_of_charge);

Serial.print("ventValue="); Serial.print(dataToSend.ventValue); Serial.print(",");
Serial.print("pumpValue="); Serial.print(dataToSend.pumpValue); Serial.print(",");
Serial.print("flowValue="); Serial.print(dataToSend.flowValue); Serial.print(",");
Serial.print("brakeEngaged="); Serial.print(dataToSend.brakeEngaged); Serial.print(
Serial.print("soundPlaying="); Serial.print(dataToSend.soundPlaying); Serial.print(
Serial.print("R2D_Button_State="); Serial.print(dataToSend.R2D_Button_State); Ser
Serial.print("airPlusValue="); Serial.print(dataToSend.airPlusValue); Serial.print(
Serial.print("airMinusValue="); Serial.print(dataToSend.airMinusValue); Serial.pr
Serial.print("prechgValue="); Serial.print(dataToSend.prechgValue); Serial.print(
Serial.print("SDC_END="); Serial.print(dataToSend.SDC_END); Serial.print(",");
Serial.print("Measure_Digital="); Serial.print(dataToSend.Measure_Digital); Seria

Serial.print("accelX="); Serial.print(dataToSend.accelX_IMU); Serial.print(",");
Serial.print("accelY="); Serial.print(dataToSend.accelY_IMU); Serial.print(",");
Serial.print("roll="); Serial.print(dataToSend.roll_IMU); Serial.print(",");
Serial.print("pitch="); Serial.print(dataToSend.pitch_IMU); Serial.print(",");
Serial.print("yaw="); Serial.print(dataToSend.yaw_IMU); Serial.print(",");
Serial.print("IMU_Speed="); Serial.print(dataToSend.IMU_Speed); Serial.print(",");
Serial.print("rawTemp1="); Serial.print(dataToSend.rawTemp1); Serial.print(",");
Serial.print("rawTemp2="); Serial.print(dataToSend.rawTemp2);

Serial.println(" ");
}

void setup()
{
    Serial.begin(9600);
    while (!Serial)
        delay(50);
    Serial.println("serial initialized");

    // reset module for consistency
}

```

```

lora.resetModule();

// initialize the lora module
SPI1.begin();
lora.initModule();

// set settings for lora modulation
lora.setSettings2();

// set the function that will be called
// when new packet is received (both can and lora)
radio.setDio0Action(setFlag, RISING);
Serial.println("interrupt funcs set");

// init can module
can.init();

// start listening for LoRa packets on this node
Serial.println("starting to listen :)");
freq_cnt = 0;
radio.setFrequency(freq_list[freq_cnt]);
state = radio.startReceive();
if (state == RADIOLIB_ERR_NONE)
{
    Serial.println(F("success!"));
    Serial.print("Freq: \t\t");
    Serial.println(freq_list[freq_cnt]);
}
else
{
    Serial.print(F("failed, code "));
    Serial.println(state);
    while (true)
    {
        delay(10);
    }
}
}

void loop()
{

currentTime = millis();
blink_led(1000);

// receiver code here

```

```
// data has already been sent over serial to be displayed
if (receiveValid)
{
    size_t received_len = 32; // or however many bytes you actually receive

    Serial.print("Encrypted: ");
    for (size_t i = 0; i < received_len; i++) {
        Serial.print(receivedMsg[i]);
        Serial.print(" ");
    }
    Serial.println();

    struct AES_ctx ctx;
    uint8_t ctr_iv[16];
    memcpy(ctr_iv, aes_iv, 16); // must match the IV/nonce used by transmitter

    AES_init_ctx_iv(&ctx, aes_key, ctr_iv);

    AES_CTR_xcrypt_buffer(&ctx, receivedMsg, received_len);

    TelemetryData* receivedData = (TelemetryData*)receivedMsg;

    Serial.print("Decrypted: ");
    for (size_t i = 0; i < received_len; i++) {
        Serial.print(receivedMsg[i]);
        Serial.print(" ");
    }
    Serial.println();

    memcpy(&dataToSend, receivedData, sizeof(TelemetryData));
    receiveValid = false;
    Serial.println("\nStarting data processing.");

    // process data
    sendToInfluxDB();
    memset(receivedMsg, 0, sizeof(receivedMsg));
}

// interrupt based handling
// when flag is set, that means data was sent/received
if (operationDone)
{
    // reset flag
    operationDone = false;

    Serial.println("Entered operation done initial main.");
    freq_cnt = 0;
```

```

int cnt = 0, cntM = 0;
// module received data on channel 1
uint8_t *str = (uint8_t *)calloc(20, sizeof(uint8_t));

int packetLength = radio.getPacketLength();
int state = radio.readData(str, packetLength);
for (int i = 0; i < packetLength; i++)
{
    receivedMsg[i + cnt] = str[i];
    cntM++;
}
cnt += packetLength;

// if message seen on first channel, cycle through the rest
if (state == RADIOLIB_ERR_NONE)
{
    // display first chunk
    lora.displayData(str, packetLength);
    receiveValid = true;
    /*
    chunk cycler
    goes through all channels and waits for the respective chunk to arrive
    */
    for (int i = 1; i < 7; ++i)
    {
        freq_cnt++;
        radio.setFrequency(freq_list[freq_cnt]);
        radio.startReceive();

        // reset str array
        memset(str, 0, 20 * sizeof(*str));

        // wait until next message is received on next channel OR timeout
        // opDone = false when entering, interrupt will make it true
        unsigned long timeStart = millis();
        Serial.print("waiting for next chunk");
        while (!operationDone && (millis() - timeStart) < 300)
        {
        }
        Serial.print("\n");
        if (!operationDone)
            Serial.println("ERR!! Message missed.");
        operationDone = false;

        packetLength = radio.getPacketLength();
        state = radio.readData(str, packetLength);
        for (int i = 0; i < packetLength; i++)
        {

```

```

        receivedMsg[i + cnt] = str[i];
        cntM++;
    }
    cnt += packetLength;

    // display received chunk if everything is ok
    if (state == RADIOLIB_ERR_NONE)
    {
        lora.displayData(str, packetLength);
    }
    else
    {
        Serial.print(F("failed chunk read, code "));
        Serial.println(state);
        receiveValid = false;
    }
}
else
{
    Serial.print(F("failed chunk 1 read, code "));
    Serial.println(state);
    Serial.print(F("Freq listening: "));
    Serial.println(freq_list[freq_cnt]);
}

// display whole message and reset string if all chunks are received
if (receiveValid)
{
    Serial.println("\nFull message: ");
    for (int i = 0; i < cntM; i++)
    {
        Serial.print(receivedMsg[i]);
        Serial.print(" ");
    }
    Serial.println();
}
else
{
    Serial.println("Message chunks invalid. Skipping this packet.");
    memset(receivedMsg, 0, sizeof(receivedMsg));
}

// set freq back to channel 1 and listen again for next packet
freq_cnt = 0;
radio.setFrequency(freq_list[freq_cnt]);
radio.startReceive();

```

```
    }
}
```

### A.3. Scriptul de transfer de la receptor la InfluxDB

```
import serial
import os
from influxdb_client import InfluxDBClient, Point
from dotenv import load_dotenv
import tkinter as tk
from tkinter import messagebox
import serial
import serial.tools.list_ports

# List available serial ports
ports = list(serial.tools.list_ports.comports())
port_list = "\n".join([f"{p.device}: {p.description}" for p in ports]) or "No serial ports found"

# Pop up with available serial ports and ask for COM port
def get_com_port():
    while True:
        def on_ok():
            nonlocal com_port
            com_port = entry.get()
            root.destroy()
        def on_exit():
            root.destroy()
            os._exit(0) # Immediately exit the script

        com_port = None
        root = tk.Tk()
        root.title("Select COM Port")
        tk.Label(root, text=f"Detected serial ports:\n\n{port_list}\nEnter COM port").pack()
        entry = tk.Entry(root)
        entry.pack(padx=10, pady=5)
        entry.focus()
        button_frame = tk.Frame(root)
        button_frame.pack(pady=10)
        tk.Button(button_frame, text="OK", command=on_ok).pack(side=tk.LEFT, padx=5)
        tk.Button(button_frame, text="Exit", command=on_exit).pack(side=tk.LEFT, padx=5)
        root.mainloop()
        # Validate COM port
        available_ports = [p.device for p in ports]
        if com_port in available_ports:
            return com_port
        elif com_port is not None:
            tk.messagebox.showerror("Invalid Port", f"'{com_port}' is not a valid port")

selected_com_port = get_com_port()
```

```

if not selected_com_port:
    messagebox.showerror("Error", "No COM port selected. Exiting.")
    exit(1)

# Load .env file
load_dotenv()

# Read values from .env
INFLUXDB_URL = f"http://{{os.getenv('DOCKER_INFLUXDB_INIT_HOST')}}:{{os.getenv('DOCKER_INFLUXDB_INIT_PORT')}}"
INFLUXDB_TOKEN = os.getenv("DOCKER_INFLUXDB_INIT_ADMIN_TOKEN")
INFLUXDB_ORG = os.getenv("DOCKER_INFLUXDB_INIT_ORG")
INFLUXDB_BUCKET = os.getenv("DOCKER_INFLUXDB_INIT_BUCKET")

# Serial port configuration (Windows)
ser = serial.Serial(selected_com_port, 9600)
print(f"Connected to {selected_com_port} at 9600 baud.")
# Initialize InfluxDB client
client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN, org=INFLUXDB_ORG)
write_api = client.write_api()

start_influx = False

try:
    while True:
        line = ser.readline().decode('utf-8').strip()
        #print(line)
        if line:
            if line == "START INFLUX":
                start_influx = True
                print("Received START INFLUX")
            elif start_influx:
                write_api.write(bucket=INFLUXDB_BUCKET, org=INFLUXDB_ORG, record=line)
                print(f"Sent to InfluxDB: {line}")
                print(f"URL: {INFLUXDB_URL} | Org: {INFLUXDB_ORG} | Bucket: {INFLUXDB_BUCKET}")
                start_influx = False
except KeyboardInterrupt:
    print("Exiting...")
finally:
    ser.close()

```

## **Anexa B. Lucrări științifice care susțin rezultatele prezentate**

### **B.1. Computer Science Student Conference 2025 (CSSC)**

# Real-time Wireless Telemetry System for a Formula Student Electric Vehicle

Tompea Radu-Gabriel  
Computer Science Department  
Technical University of Cluj-Napoca  
Cluj-Napoca, Romania

*Coordinator:* Bogdan Iancu  
Computer Science Department  
Technical University of Cluj-Napoca  
Cluj-Napoca, Romania

**Abstract**— The modern generation of motor vehicles comes equipped with a complex network of sensors and safety systems, which need to gather critical information in real time, like engine performance and fuel economy, as well as breaking parameters and the steering angle. This data is constantly analyzed by engineers to offer the driver the safest and most efficient transportation option, uncovering faults and inefficiencies. This paper will offer insight into the design of such a system and the current development of vehicle communication structures, focusing on different state-of-the-art solutions and a novel approach to telemetry systems in Formula Student cars. The solution proposes using the radio technology LoRa in a different context than it was designed for, and combining different programming paradigms like fog computing and Cloud Storage, finally packaging it all in a plug-and-play enclosure and install system.

## I. INTRODUCTION

Over the years, checking the performance of any technology at the data level was a difficult task, thus underlying the need for a system that can transmit that data remotely and pairing it with intuitive visualization. In the automotive industry, such a system proves critical in the development and testing of automobiles, and every company has its own unique solution to this problem. When testing a vehicle, **wireless** data transmission is preferred, since a wired system would make things significantly harder.

The purpose of a telemetry system is to collect data in a place that is remote or inconvenient and to relay this data to a point where it can be evaluated. Typically, telemetry systems are used in the testing of moving vehicles such as cars, aircraft, and missiles. Telemetry systems are a special set of communication systems. When the telemetry system is used for both control and data collection, the term *supervisory control and data acquisition* is applied. [1]

The basic concept of telemetry has been in existence for centuries. Various mediums or methods of transmitting data from one site to another have been used. Dataradio provides a wireless method for transmitting information. Telemetry using radio waves or wireless offers several distinct advantages over other transmission methods. Some of these advantages are [2]:

- No transmission lines to be cut or broken
- Faster response time
- Lower cost compared to leased lines
- Ease of use in remote areas where it is not practical or possible to use wire or coaxial cables

- Easy relocation
- Functional over a wide range of operating conditions

Modern automobiles are becoming increasingly electromechanical and even the parts that remain mechanical are being tuned to work more efficiently by drawing data via sensors monitoring them. This phase of development of the vehicle is split into the initial static tests on a test bench and dynamic field tests.

For the initial static testing, where the main setup and tuning is done, the engineers require high resolution and high data rates as some of the more critical sensor data is being acquired, logged and analyzed here. For this, the DAQ (Data acquisition) system is highly essential, whereas the telemetry option is used only to monitor for safety. For the dynamic field testing, the engineers find the wireless telemetry option to be of higher importance, while the on board DAQ runs and only parts of the data are analyzed where an anomaly was noticed through the telemetry data. In both phases, due to heat from various components, vibrations of the moving vehicle, the inherent nature of each sensor and so on, the data being acquired by the sensors ends up being affected by noise. In order to filter out the noise, the use of filters is unavoidable and is highly essential.

Thus, it is seen that both systems play an extremely important role in the research and development phase of a vehicle, and having said so, there seems to be a scarcity of manufacturers who have products that cater directly to the automobile industry in this regard. Most companies have universal application type systems that work well but cause an unnecessary hike in the overall power consumption and cost. Furthermore, the end users are expected to develop their own filtering stages and so on to get the system to fit their application. [3]

**Vehicle-to-Everything (V2X)** is a generic term that refers to a wireless communication system that allows the exchange of data in real time between a vehicle and any entity in its environment, including other vehicles (V2V), infrastructure (V2I) and communication networks (V2N). The main purpose of this technology is to improve safety on the road and reduce traffic. [4]

**Vehicle-To-Network (V2N)** is a form of communication that allows the vehicle to transmit information to a large, infrastructural network like cloud servers, data centers and mobile networks to offer external information like traffic

conditions and weather, thus permitting the vehicle to take more informed decisions and to enhance driver experience. [5]

**Vehicle-To-Vehicle (V2V)** is a technology that makes real-time communication between two vehicles possible, without the need for an intermediary infrastructure. They can send information, like speed, direction, braking times and location to better road safety. [5]

In the case of V2V, V2N and V2B/I, as presented in Figure 1, these technologies can be integrated as separate solutions focusing on different problems (in the case of V2V to minimise accidents and V2N to accurately predict traffic data), and the ideal goal is to incorporate them into a unified, distributed, city-wide system.

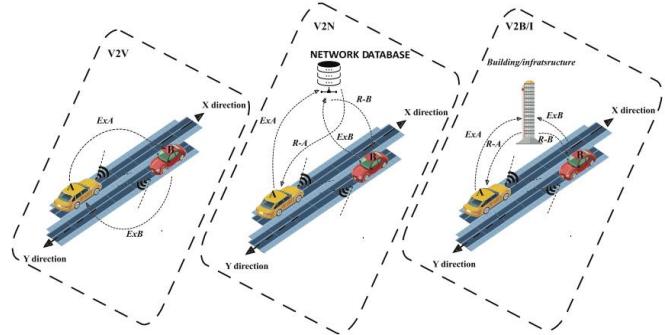


Fig 1. Different conceptual working principles for V2V, V2N, V2B/I, respectively

In this paper, we will first take a look at existing solutions under the “Relevant Work” section, and after that, we will analyze the architecture of the Formula Student Electric Vehicle ART TU E17, developed by team ART TU Cluj-Napoca that is part of the Technical University of Cluj-Napoca. Then, we will cover the requirements of the telemetry system that is being developed, along with the conceptual architecture and how the telemetry system will incorporate itself with the existing vehicle architecture under the section “Conceptual Architecture of the ART TU E17 Wireless Telemetry System”. In the final part of the paper, I will present the conclusions and future directions for research and development.

The focus of this paper is to give you an insight into the importance of a telemetry system in the automotive industry and the harsh conditions in which this system must operate, alongside a visualization concept where the data can be seen and analyzed.

## II. RELEVANT WORK

In [6], the authors present a solution to this difficult task, developing and testing an IoT-based telemetry system that uses OBD-II data and GPS, having 3 components:

1. **On-board hardware module.** It uses an ELM 327 OBD-II interface connected to an Arduino Mega 2560 to log engine parameters like RPM, speed coolant temperature and throttle position as well as the GPS position, storing the data to a microSD

and transmitting over Bluetooth (HC-05) and GSM/GPRS.

2. **Android application.** It provides the functionality of a user interface and GPS data provider, receiving the engine data from the server, displaying live statistics and uploading information for speed-of-transmission analysis.
3. **Web server and database.** Based on PostgreSQL, it handles the receiving, storage and visualization of the telemetry data.

As for the results, for 611 entries, 91.5% of them had transmission intervals less than 10s, and most common was approximately 5s, having lost around 2% of the data due to traffic or GPS loss. These tests were done on a Honda Stream 2010 and Toyota Rush 2017, monitored over 20 hours.

In terms of challenges and limitations, the GSM/GPRS connectivity posed a problem because of the data processing delays, alongside the variability of the data formats. Because each car had a different ECU, they require different commands to get the data and thus complicating the standardization. A unified design across varying vehicle architectures is needed for broader deployment and analysis.

In [7], the authors proposed a system for solar car races that integrate real-time video, GPS positioning, and velocity tracking to monitor and broadcast data.

It uses GPS for live location and speed, using a Raspberry Pi as an Onboard Unit (OBU), a camera system for live video capture and a wireless transceiver for transmitting all telemetry and video data.

A wireless point-to-point link is used to establish a connection between the solar car and the tracking vehicle, so there is not a very big distance between them. The transmission supports video and telemetry and is an interesting solution to monitor the status of the vehicle.

In [8], the authors propose a robust solution for secure remote vehicle diagnosis and maintenance over 5G Vehicle-to-Network (V2N), addressing critical safety and privacy issues with a dedicated authentication scheme. The system includes 3 primary components:

1. **Vehicle-side module (VMCS and UE):** Vehicles are equipped with a Vehicle Maintenance Control System (VMCS) and User Equipment (UE), enabling vehicle owners or drivers to initiate remote diagnostic services. Users authenticate via identity, password, and biometrics. The VMCS communicates securely with the Vehicle Networking Control Center (VNCC) over 5G, leveraging extended Chebyshev chaotic maps to protect session keys and data integrity.
2. **Vehicle Networking Control Center (VNCC):** Acting as a centralized trusted authority, the VNCC manages registration, authentication, and service coordination. It authenticates both vehicles and Vehicle Service Centers (VSCs), relays fault data, and establishes secure channels for maintenance actions. It also supports direct fault

- resolution from its database, bypassing the need for a VSC if a fix is available.
3. **Vehicle Service Center (VSC):** Each VSC includes a server and employees who must also authenticate using identity, password, and biometrics. The VSC can access vehicle data through a secure channel established by the VNCC, allowing diagnosis and online software/firmware repairs. Only if the fault is resolvable online is service performed remotely; otherwise, physical service is suggested.

The paper primarily focuses on the security and performance analysis rather than experimental transmission results. Using formal tools like the Tamarin Prover and a random oracle model, the authors demonstrate that their scheme meets essential security properties, including mutual authentication, perfect forward secrecy, identity anonymity, and resistance to known attacks. Efficiency is confirmed through comparisons to other recent protocols in terms of computational and communication overhead. The authors note that practical implementation faces challenges such as ensuring secure storage and processing of sensitive data on all devices (e.g., VMCS, UE, and VSC-S), protecting against device theft or side-channel attacks, and handling variability in user roles (e.g., owner vs. driver). They emphasize the importance of collaborative authentication when the driver is not the vehicle owner, and highlight the need for continued protection even if long-term keys are compromised.

### III. ART TU E17 ELECTRIC VEHICLE ARCHITECTURE

#### 3.1. Low Voltage System Overview

##### Electronic Control Unit (ECU)

The ECU is the center of the electric vehicle and is responsible for gathering and sending all the data, including sensors, temperatures, dashboard, and many more important information that keeps the vehicle running efficiently and safely.

Developed in-house, it is based on the Teensy 4.1, and has a lot of communication protocols in it (SPI, UART, RS232, CAN Bus, I2C) for all types of sensors.

The ECU controls the vehicle with Arduino code, from the threshold at which certain systems activate to motor control. Figure 2 presents a 3D render of the ECU, so as to better conceptualize how it looks and how it fits in the car, having standard automotive plugs and an easily interchangeable Teensy 4.1, if anything were to happen to it.



Fig 2. 3D render of the ECU printed circuit board

##### Tractive System Active Light (TSAL Main)

The TSAL Main board signals through the use of multiple LED's the status of the Tractive System, with green and red light. Green indicates that the voltage in the vehicle is below 60 VDC and the contactors are all disconnected, mainly every power relay in the car is opened, meaning that there is no hazard from the 200 VDC of the vehicle's battery.

Red indicates that the voltage exceeds 60 VDC and the vehicle is not safe to drive and handle, especially if a fault is detected.

All TSAL circuitry must be hard-wired electronics, excluding software control, with red and green circuitry operating independently.

##### TSAL and Dashboard Indicators

The indicators serve the same function as the TSAL Main board, but it extends the signal to the cockpit and main hoop, visible to the driver in direct sunlight. The same rules apply to these PCB's too, no software components, only hard-wired components.

As well as the Main board, all the signals must be Safety Critical Signals (SCS), between 0.5V to 4.5V.

The dashboard indicators show the driver the status of the Insulation Measurement Device (IMD), Accumulator Management System (AMS) and the TSAL green or red status.

##### TSAL Measurement

This board separates the Low Voltage System from the High Voltage system by a 6mm gap, connected by an optocoupler. It detects if a voltage above 60 VDC is present at the measurement points and that the contacts in the power-up sequence are closed. Same rules as the TSAL Main, only hard-wired components and all signals must be SCS.

##### Wireless Telemetry

This sub-system needs to communicate the data of the vehicle while operating (speed, temperatures, state of charge, state of contactors) to the on-ground team using a wireless system to a server.

It also serves the purpose of storing the past measurements for analysis through graphs.

Figure 3 presents a diagram which depicts the Low Voltage System of the ART TU E17 electric vehicle. The ECU is the component where all the data is stored and processed, and the connection with the wireless system should mainly be done with the ECU, in order to better gather and transmit the data after it has been aggregated in the ECU.

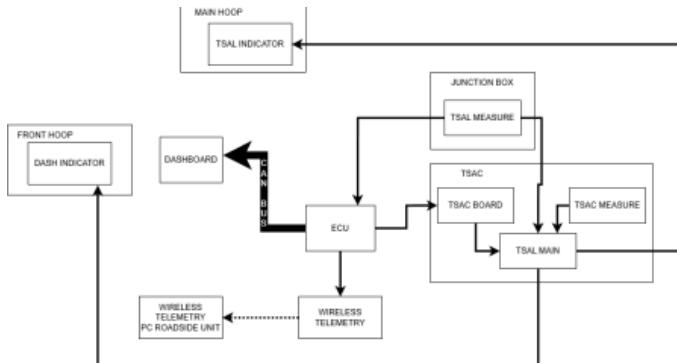


Fig 3. Graphical Depiction of ART TU E17 Low Voltage System Architecture

### 3.2. REQUIREMENTS OF THE TELEMETRY SYSTEM

For the scope of this telemetry system, these are the functional requirements it must achieve:

- 1. Collect and store the relevant data from the electric vehicle.** The system must acquire data from all the relevant sensors of the vehicle and to recognize their data type.
- 2. Store the data in a data structure that is efficient for transmission.** The system must store the data temporarily in a data structure that is adequate for transmission over long distances.
- 3. Configure and manage the transmission medium over the long distance.** The system must be configured in-tune with the available hardware to cover the most distance possible without compromising the data.
- 4. Communication between the electric vehicle and the server.** The system must provide a stable and reliable communication between the electric vehicle and the server over the course of the dynamic testing, without interruptions and without compromising the integrity of the data, over a distance of 0.8 to 1 kilometer non-line-of sight. Figure 4 presents the difference between line-of-sight transmission (left) and non-line-of-sight transmission (right) in order to better understand the importance of the resilience of the telemetry system. This is important because in a moving vehicle, different objects can (and will) interfere with the transmission medium.



Fig 4. Visualization of line-of-sight transmission and non-line-of-sight transmission [2]

- 5. Process and store the data produced by the vehicle.** The system must have the capacity to aggregate the data transmitted from the vehicle into a processable form and store them in an efficient manner in terms of memory and time of processing.
- 6. Real-time monitoring of the data.** The system must incorporate a real-time visualization of the data generated by the vehicle's sensors and safety systems, being updated every second.
- 7. Visualization of the data after the transmission is finished.** The system should provide the user with an easy-to-read visualization of the data after the transmission is finished so it can be analyzed with ease and comparing them for development purposes.

As for the non-functional requirements of the system:

- Performance.** The system should offer high responsiveness to monitor the data at the minimum refresh rate.
- Scalability.** The system must offer the possibility, be it from architecture or implementation, to be adapted to a large quantity of data or to a larger transmission distance.
- Security.** The system must incorporate a way to secure the data during the transmission and while stored so that unauthorized people cannot access the data of the electric vehicle.
- Availability and Accessibility.** The system must offer stability and availability to constantly access its services constantly, on whatever device might the user access it from.
- Flexibility.** The system must be “plug-and-play”, having easy installation, modification process and maintenance for the different needs of the development team in the coming seasons.
- Intuitive Design.** The system must easily be used by a user at first sight and easy to navigate for a user that is not familiar with IT.

### 3.3. CONCEPTUAL ARCHITECTURE OF THE ART TU E17 WIRELESS TELEMETRY SYSTEM

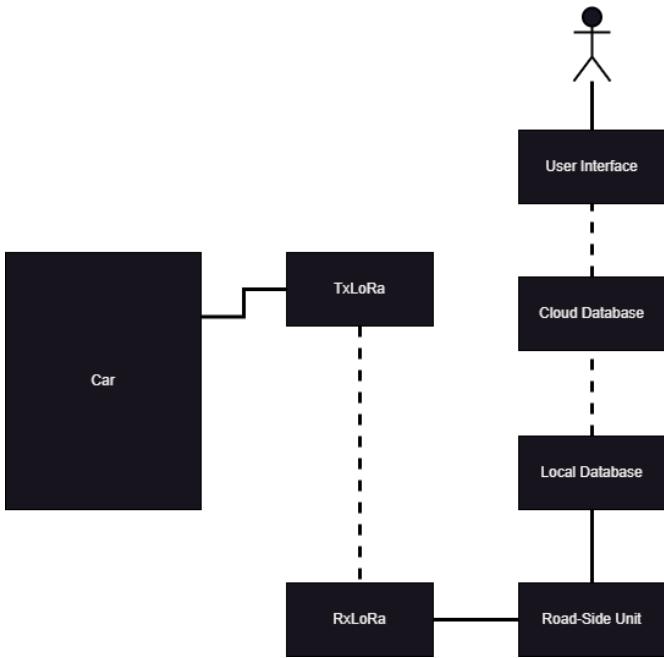


Fig 5. Conceptual Architecture of the Wireless Telemetry System

In figure 5, I am proposing the telemetry system's conceptual architecture , comprising of 4 main components:

- Transmitter LoRa (TxLoRa). Gathers the data from the ECU in a message that is transmitted every second, compresses the message and stores the data in the appropriate data structure to prepare it for transmission. Another step in the transmission is the encryption, adding it to the packet before the transmission. This approach provides the easy installation of the system with the current vehicle architecture, while also providing easy to maintain and read code and hardware, while also providing security and reliability, making this a very suitable option for incorporating the transmitter.
- Receiver LoRa (RxLoRa). Receives the packet from the transmission medium and decrypts the data, while also storing it in the adequate data structure. This provides a buffer for the data until it is stored in the local database, providing an extra level of redundancy, making the system more robust and resilient.
- Road-side Unit. It acts as the server, processing the data while providing a way for the users to connect to the User Interface, while also centralizing the data.
  - Local database. Its purpose is to store the data locally for easy access on the user interface. This approach is known as fog computing and is an ingenious way to provide very good response times for the interface with new data, while also having another layer which stores the data, adding

redundancy and making the system all the more effective and efficient.

- User Interface. This is what the user sees and is responsible for displaying the data in an easy to read format, usually visualisations and the like for debugging and fault-finding. Getting the data straight from the local database, its response time is very quick and can be easily modified.
- Cloud Database. The data is uploaded here after the transmission ended, providing a way to access the data while it is not in real-time, proving very useful to the development team. It adds redundancy and a much needed feature in the automotive industry.

When choosing a technology stack for this solution, I frequently referred to [2], giving some insight into what I need to keep in mind:

The main criteria for selection of the wireless protocol are power consumption and range. Due to the vehicle moving around a large track during testing or a race, Bluetooth and IR were immediately eliminated as options, as the range of Bluetooth is very limited and IR requires line of sight for communication, which would not be possible.

For the selection of the microcontroller, the multiplexing of the sensor signals through a single ADC seems like an option, it causes parts of the data to be lost, and this is especially important for signals that change within fractions of seconds in a vehicle. Hence the microcontroller was required to have a minimum of 8 on board ADCs to make the process of design and programming simpler.

And as for the implementation of GUI and filters, the user was required to select the data parameters to be viewed on each graph, then the filtering process the data was to go through and run the program. The telemetry interface required a different approach from the simple graph layout taken with the DAQ side. Since this data would be viewed live, and needed to be immediately interpretable, along with the raw numerical data being shown, a calibration option was provided so that a simple digital value of say, voltage could be converted to a throttle position percentage and so on.

The paper proposes a system that incorporates several components to achieve the requirements listed above:

- The connection between the vehicle's ECU and the telemetry system is done using CAN Bus (Controller-Area-Network) technology. While the CAN-Bus is specifically developed for automotive communication purposes, ISO-TP (ISO 15765-5) provides a much-needed feature in this system. By default, the CAN-Bus supports only 8 bytes per CAN frame, but with ISO-TP, we can send as many bytes as necessary because it splits the large messages into multiple CAN frames and reassembling them on the receiving end. The adoption of ISO-TP provides the necessary scalability and flexibility to transmit extensive telemetry data, such as sensor arrays without compromising CAN-Bus's real-time capabilities.

- The transmission medium proposed is LoRa, a low-power, long-range wireless communication technology designed for sending small amounts of data over large distances, achieving 2-5 kilometers in an urban setting, while consuming very little power and offering good resistance to interference and obstacles due to spread spectrum transmission. It achieves the favorable distances by operating in the sub-GHz ISM bands, extending further under line-of-sight conditions, making it perfect for test conditions. Figure 6 presents a typical LoRa network, comprised of the end nodes that gather data, gateways that centralize the data and send it to the network server, which sends the data to the different application servers, which display or use the data, all the while this system being encrypted with AES.

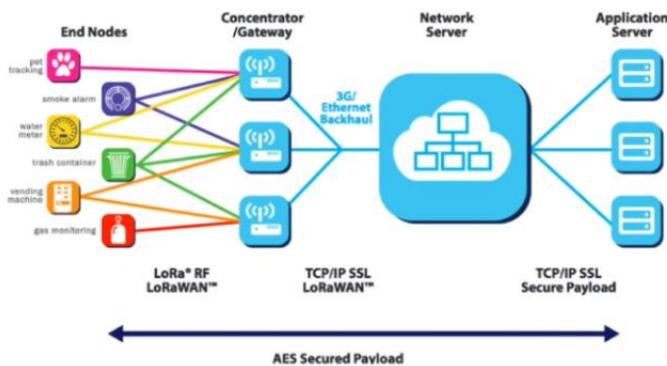


Fig 6. A typical LoRa network.

- For the software technology stack, the proposed solution uses Docker, making it easy to deploy and run applications inside containers, offering accessibility to anyone in the team intending to be the server node. It streamlines development and maintenance, encapsulating the application along with its dependencies, libraries and runtime environment into portable units that can run consistently across various computing environments, this minimizing the “works on my machine” problem.
- In terms of database selection, I propose InfluxDB, a time-series database that is designed to handle large volumes of time-stamped data, including measurements and sensor reading, making it perfect for this context. It will store and process the data locally, as to not impact the performance of the real-time data visualization. InfluxDB is optimized for handling high write and query loads of time-indexed data, supporting advanced functions such as aggregation, downsampling, and retention policies.
- For the visualization of the data, Grafana is the preferred option, offering a quick and elegant solution to create an interactive and responsive

dashboard. It has native integration with InfluxDB, making the connection seamless. Its support for various panel types and alerting mechanisms allows the team to monitor vehicle performance metrics in real time and receive notifications when parameters deviate from expected thresholds.

- For the persistence of the data, Firebase is used as a cloud service to store the data, and an in-house CSV plotting tool is then applied to visualize the data after the transmission ended. Firebase provides scalable data storage and synchronization capabilities, ensuring data durability and availability across distributed systems.
- The decision to use all these open-source solutions comes to cost. The components for this system are cheap, and the upkeep of the system is virtually free if we keep in mind the storage used by the Cloud service.

The system works in the following way:

- The ECU aggregates the data into a ISO-TP CAN-Bus message with all the relevant and necessary data and sends it to the LoRa transmitter.
- The LoRa transmitter then encrypts the data using AES-128 CTR encryption and forms it into a packet.
- The LoRa receiver receives the packet, decrypts the data and stores it into a chosen data structure.
- InfluxDB then receives the data through Influx Line Protocol and stores it locally for the duration of the testing session.
- Grafana then gathers the data from InfluxDB and creates a easy to read panel with all the relevant information.
- After the testing is completed, the system will upload the data to Firebase, where it will be stored for later use.
- When the data is needed for analysis, the team can download the data from the Cloud using the CSV plotter and visualize it.

Figure 8 displays a theoretical dashboard which gathers simulated data from the transmitter, showcasing how the system will look and behave.



Fig 7. The real-time data monitoring dashboard for the Mechanical department

#### 4. CONCLUSIONS

This paper outlined the design of a wireless telemetry system tailored to the ART TU E17 electric racecar. After reviewing existing solutions and defining key requirements, we proposed an architecture centered around long-range, low-power communication, reliable data storage, and accessible visualization tools.

Future work includes integrating the system with the ECU, field testing during dynamic events, and refining the web-based dashboard for improved usability and real-time performance analysis. This approach offers a scalable and efficient solution for student teams and other motorsport applications.

#### REFERENCES

- [1] Frank Carden, Russell P. Jedlicka, Robert Henry, "Telemetry systems engineering", 2002.
- [2] J. K. Roy, "An introduction to telemetry. part 1: Telemetry basics," Online PDF, Self-published online, Tech. Rep., 2012
- [3] J. R. Chandiramani, S. Bhandari, and H. S. A., "Vehicle data acquisition and telemetry," in Proceedings of the 2014 Fifth International Conference on Signal and Image Processing (ICSIP), 2014
- [4] A. Ghosal and M. Conti, "Security issues and challenges in v2x: A survey," Computer Networks, Mar. 2020
- [5] M. A. Hossain, M. R. Islam, and M. A. Rahman, "Design and analysis of a novel power management approach for energy-efficient vehicular networks," International Journal of Energy Research, 2014
- [6] F. Fahmi, E. Sutanto, M. Yazid, and M. Aziz, "Integrated car telemetry system based on Internet of Things: Application and challenges," *J. Eng. Sci. Technol.*, vol. 15, no. 6, pp. 3757–3771, Dec. 2020.
- [7] E. N. Mambou, T. G. Swart, A. R. Ndjiongue, and W. A. Clarke, "Design and implementation of a real-time tracking and telemetry system for a solar car," in *Proc. AFRICON 2015*, Addis Ababa, Ethiopia, 2015
- [8] R. Ma, J. Cao, D. Feng, H. Li, X. Li, and Y. Xu, "A robust authentication scheme for remote diagnosis and maintenance in 5G V2N," *Journal of Network and Computer Applications*, Jan. 2022.