Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Tudományos Diákköri Konferencia
dolgozat

Mozgásemuláció és -vizualizáció V2X járműkommunikációs alkalmazások valós hardveren történő fejlesztéséhez és tesztelséhez

Készítette
Szendrei Zsolt

Konzulens
Dr. Bokor László

2016

TARTALOMJEGYZÉK

Kivonat

Napjainkban már természetes, hogy szinte valamennyi autógyártó rendelkezik önvezető járművel és komoly kutatási-fejlesztési projektekkel a témában. A jelenlegi megoldások közös jellemzője, hogy az autonóm vagy részlegesen autonóm járművek nem tudnak egymással közvetlenül kommunikálni, működésük során csak saját szenzorjaikra, érzékelőikre támaszkodnak, és információcseréhez maximum a gyártó felhőjét veszik igénybe. Ez komoly korlátozás, mivel a jelenlegi hozzáférések nagy késleltetést okozhatnak, ami egy olyan környezetben, ahol pár tizedmásodperc is sokat számít a balesetek elkerülésében, nem megengedhető. A kooperatív jármű-kommunikáció dedikált frekvenciatartomány felhasználásával oldja meg a közvetlen kommunikációt a járművek és a közlekedési infrastruktúra között, és segítségével olyan információkat oszthatnak meg az infrastrukturális elemek, autók vagy egyéb a közlekedésben résztvevő eszközök, mint például sebesség, pozíció, irány és szenzorok, érzékelők által szerzett adatok [1] . A kooperatív járműkommunikációs (V2X, vehicle-to-everything) alkalmazások képesek felhasználni ezt az együttműködést arra, hogy figyelmeztessenek veszélyes helyzetekre, úthibákra, megváltozott útviszonyokra, megkülönböztető jelzést használó járművekre, stb. [2]. A jövőben ennek a technológiának a használata csökkentheti a közúti balesetek, halálesetek számát, a károsanyag-kibocsátást, és jelentősen javíthatja városaink közlekedését, azaz egy olyan alapvető eszközzé válhat, mint manapság a biztonsági öv.

A járműkommunikáció alapvető technológiái már jelenleg is nemzetközi szabványokban rögzítettek, a széleskörű telepítés első szakasza napjainkban zajlik, amit az egyre nagyobb számú tesztrendszer (pl. Connected Vehicles [3] , Cooperative ITS Corridor [4]), bizonyít. Azonban ezeknek a kooperatív intelligens közlekedési rendszereknek (C-ITS, Cooperative Intelligent Transport System) a fejlesztése, konkrét V2X alkalmazások implementálása és tesztelése nehéz, mert laboratóriumi körülmények között kellene vizsgálni olyan folyamatokat, melyek járművek/járműrendszerek mozgását igénylik. Léteznek ugyan komplex szimulációs keretrendszerek a járműkommunikációs alkalmazások teszteléséhez és új megoldások kifejlesztésének támogatásához, de ezekben a valós hardverek, valós C-ITS protokoll-stack-ek használata nem megoldott. Ilyen pl. a Veins [5], VSimRTI [6] , TraNS [7] , szimulációs rendszer is: minden szimulálva van, nincs lehetőség, vagy nagyon bonyolult lenne valós hardverrel, fizikai implementációval együtt használni, tesztelni a megoldásokat. Éppen ezért a dolgozatom egy olyan mozgásemulációs és -vizualizációs keretrendszer megalkotására fókuszál, mely valós hardveren működő, valós C-ITS stack használatát teszi lehetővé a fejlesztés/tesztelés összetett folyamatai során. A létrehozott keretrendszer hatékonyságát általam implementált V2X alkalmazások segítségével alá is támasztom.

Abstract

Nowadays nearly every automobile manufacturers have self-driving vehicle, research and development projects in this topic. All of the existing solutions have a common property: the autonomous or partially autonomous vehicles are not able directly communicate with each other. They rely only on their own sensors to self-driving or to assist for the driving decisions; if they need to exchange information, they usually use a manufacturer-specific cloud infrastructure. This is a tremendous restriction, because it can cause serious delay and in this environment, few milliseconds gain can help to prevent an accident. Counter to this, cooperative vehicle communication uses a dedicated frequency range to provide direct communication between vehicles and infrastructures. This dedicated short-range communication guarantees to share information such as velocity, position, direction and data from any sensors or sensor networks [1]. Cooperative vehicle communication (V2X, vehicle-to-everything) applications can use this cooperation to warn for hazardous situations, road warnings, changed road conditions and emergency vehicles, etc. [2]. In the future, this technology shall save millions of lives by avoiding or minimizing the effects of an accident, and will decrease emission and traffic congestion in urban environments.

There are already well defined international standards about the basics of cooperative intelligent transportation system (C-ITS) technologies; furthermore, the deployment of the first wide-scale test systems have already started, for example within the frameworks of the Connected Vehicles [3], or Cooperative ITS Corridor [4]. However, the development of C-ITS and the implementation of V2X applications is difficult, because these processes require moving cars and/or traffic systems. There are a lot of complex simulation frameworks to help vehicle communication application test and development and also to assist to deploy new systems, but these software products do not support real-life hardware devices and real C-ITS protocol stack implementations. Veins [5], VSimRTI [6], TraNS [7] are such simulation software products, but in these systems everything is simulated, we are not able to use these software solutions with real hardware and C-ITS software or if it is possible this would be so complex and difficult to implement. Therefore, my work focuses on a general framework for vehicle motion emulation and visualisation interoperable with real V2X hardware/software systems. My solution supports real C-ITS protocol stack and such it helps to develop, deploy and test complex cooperative vehicle communication applications. Moreover, I present the efficiency of the proposed framework with real V2X applications.

1. Bevezetés

A jármű-kommunikáció egy olyan új technológia, ami az autóipar és a hálózati területek kooperációjának köszönhetően jöhetett létre. Jelenleg is több kutatás és szabványosítási munka zajlik annak érdekében, hogy ezt az új technológiát minél hamarabb, minél szélesebb körbe el lehessen terjeszteni. A dolgozatom első felében magának a jármű-kommunikációs technológia-családnak az elméleti hátterébe nyújtok betekintést. A 2. fejezetben bemutatom, hogy mi is az a járműkommunikáció, vázolom a használt architektúrát. A fejezet második részében bemutatom a különböző használati lehetőségeit, az alkalmazásokat. A 3. fejezetben ismertetem azokat az eszközöket, melyekkel lehetőség van járműkommunikációs szimulációkat készíteni, abból a célból, hogy hálózati protokollokat, alkalmazásokat fejlesszünk, teszteljünk.

Dolgozatom második részében (4.- 6. fejezet) bemutatom az általam elkészített jármű-kommunikációs szimulációs keretrendszert, részletezem az egyes komponenseit. Az 4. fejezetben egyrészt ismertetem a keretrendszer központi elemét, ami a megjelenítést és az OBU-t (On- Board Unit) köti össze. Másrészt a járművek megjelenítését, követését megvalósító eszközt, ami böngészőben, a Google Map használatával teszi lehetővé, hogy az egyes járműkommunikációs alkalmazásokat működés közben megvizsgáljuk, teszteljük. A 5. fejezetben egy olyan létező eszköznek (gpsfake [8]) az általam végrehajtott kiegészítését/módosítását részletezem, aminek a segítségével mozgást lehet emulálni, így laborkörülmények között is úgy érzékelheti az OBU, hogy mozog. A megfelelő használathoz szükség volt egy olyan interfész létrehozására, mely a dinamikus műveleteket (megállás- elindulás, újratervezés, stb.) meg tudja valósítani. A 6. fejezetben a jármű-kommunikációs alkalmazásokhoz létrehozott interfészt ismertetem, mely lehetőséget teremt a koordinációt, adatgyűjtést és megjelenítést végző szerverrel történő kommunikáció végrehajtására. Továbbá bemutatok egy jármű-kommunikációs alkalmazási lehetőséget is (megkülönböztető jelzést használó jármű), mellyel a létrehozott keretrendszer működését demonstrálom.

2. Jármű-kommunikáció

A járműhálózatok (vehicular networks) egy olyan területe a vezeték nélküli hálózatoknak, ami a járműipari igények és a különböző vezeték nélküli technológiák egymásra találásával jöhetett létre. A járműhálózatokban spontán kapcsolódhatnak a mozgó járművek, infrastrukturális elemek (jelzőlámpák, táblák, út menti egységek, fizető kapuk, közös néven RSU-Road Side Unit) között. Az ilyen hálózatot VANET-nek (Vehicular Ad hoc Network- Ad-hoc jármű hálózat) hívják, mely lehetővé teszi a valós idejű hálózati alkalmazások használatát az egymáshoz közel elhelyezkedő egységek között. Ez az új technológia megkülönböztetett figyelmet kap a kutatóktól, a jármű ipartól, az államoktól és a különböző szabványosító testületektől is. Ennek köszönhetően jöhetett létre az a dedikált rövid hatótávú kommunikáció (DSRC-Dedicated Short Range Communications, [9]), mely az 5,9 GHz-s tartományban biztosít frekvencia használatot. Ezen felül létrejöttek különböző ipari/akadémiai együttműködési szövetségek is, mint Európában a Car to Car Communication Consortium (C2C-CC [10],), melynek a feladata a közút biztonságot- hatékonyságot növelő alkalmazások létrehozása. [11]

2.1. Architektúra [12]

Az ITS rendszerek megvalósulásának támogatásához több szabványügyi hivatal, mint például ISO [13] , ETSI [14], IEEE [15] továbbá konzorciumok C2C-CC, dolgoznak együtt, hogy egységes kommunikációs architektúrát hozhassanak létre a kooperatív járműkommunikáció számára. Ennek eredményeként két jelentősebb szabvány jöhetett létre IEEE Wireless Access in Vehicle Environments (WAVE [16],) és az ISO Communications Access for Land Mobiles (CALM [17]).

A WAVE architektúra felépítése megfelel az OSI hét rétegéből az első négynek. A fizikai réteg az IEEE 802.11 p protokollon alapszik és a DSRC sávot használja. A média hozzáférési és a logikai link kontrol az IEEE 802.11 és IEEE 802.2 szabványoknak megfelelően működik. A hálózati és transzport réteg támogatja az IPv6 és a TCP/UDP átviteli lehetőségeket. A biztonság kritikus alkalmazások számára a WAVE-ben létrehozták a Short Message Protocol (WSMP [16]) protokollt, hogy biztosítsa a magas prioritás és az alacsony késleltetéssel szemben támasztott követelményeket. A WAVE nem definiál alkalmazás réteget, ehelyett egy erőforrás menedzsert (Resource Manager) vezettek be, ami csatlakoztatja az alkalmazást az OBU-hoz (On-Board Unit) vagy az RSU-hoz, hogy engedélyezze az alkalmazások számára a rendszer erőforrásokhoz a hozzáférést.

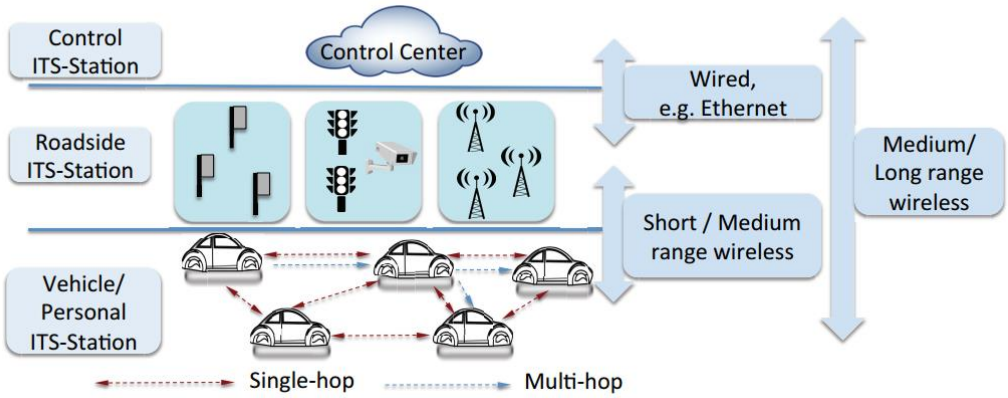
Az ISO CALM is négy protokoll blokkot definiál (hozzáférési, hálózati, alkalmazási és menedzsment), ami megfelel az OSI modellnek. A CALM többféle hozzáférési technológiát támogat, mint celluláris és WLAN, emellett támogatja az IEEE WAVE 802.11 p protokollt is (CALM-M5 [18]). A hálózati és transzport réteg főként az IPv6 alapú protokollokat támogatja. Az időkritikus ITS alkalmazások számára a CALM-FAST [19] lehetőséget biztosít egy ugrásos (az üzenet egyik járműtől egy má-

sik, közelében elhelyezkedőhöz kerül, az tovább nem küldheti) V2V kommunikációra is az alacsony késleltetés eléréséhez. A rétegek közötti menedzsment biztosított a CALM-ban, de nincs biztonsági réteg.

A szabványosító szervezetek mellett a gyártók által létrehozott szervezetek, mint a C2C-CC is létrehozta a maga architektúráját, hogy felgyorsítsa a fejlesztést. A C2C-CC protokoll stack nagyon hasonlít a CALM-ra, V2V irányú kommunikációt IEEE 802.11 p protokoll használatával biztosítja, míg a V2I (vehicle to infrastructure) IEEE 802.11 a/b/g/n segítségével biztosítja. A biztonságkritikus alkalmazások is támogatottak (C2C transzport protokoll [20]), míg az egyéb kommunikáció történhet TCP/UDP/IP használatával. A hálózati protokoll földrajzi címzést használ, együttműködve az ETSI C-ITS szabványaival.

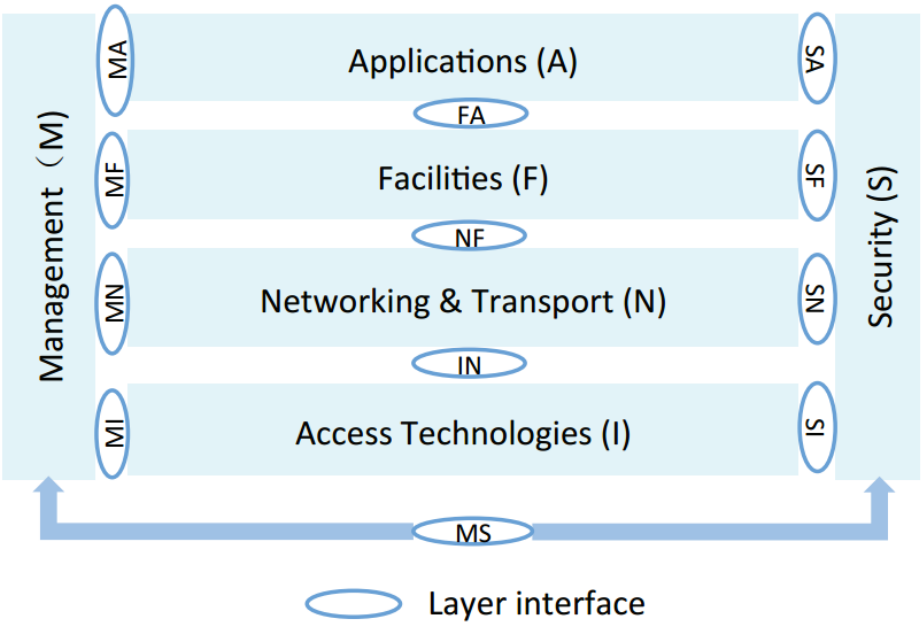
Az Európai Unió célja globálisan együttműködő járműkommunikációs rendszer fejlesztése, mely képes együttműködni különböző gyártók esetén is. Ennek elősegítésére olyan projekteket hozott létre, mint például a SAFESPOT [21], ami részt vesz a különböző kommunikációs architektúrák fejlesztésében, tesztelésében.

Egy álltalános architektúra vázlat látható 2.1. ábra-n, melyen egy C-ITS rendszer fontosabb résztvevői láthatók. Mobil résztvevők lehetnek járművek, gyalogosok, tömegközlekedési járművek. Közöttük a kommunikáció lehet egy vagy több ugrásos, ami függ a kommunikáció tartalmától, használt protokolltól. A fix infrastruktúra állhat egyrész már ma is létező elemekből, mint jelzőlámpa, tábla, stb., de már kooperációval rendelkező funkcióval, másrészt szükség lehet olyan állomásokra, mely a nagy terület lefedését megvalósító kommunikációt képes biztosítani. Ezen felül fontos, hogy a fix elemek képesek legyenek a központtal kommunikálni, különböző információkat megjelenteni, továbbítani.



2.1. ábra C-ITS koncepció architektúra [12]

Az architektúra minden eleme tartalmaz olyan általános ITS-Station (ITS-S, ITS állomás) funkciót, melyet minden résztvevőnek meg kell valósítani (pl. a protokoll stack, kommunikáció stb.). Ez az általános kommunikációs architektúra látható a 2.2. ábra-n is. Az architektúra úgy lett megvalósítva, hogy az ITS alkalmazásokkal minél jobban együtt tudjon működni, ugyanakkor az egyes rétegek igazodjanak az OSI modellhez.



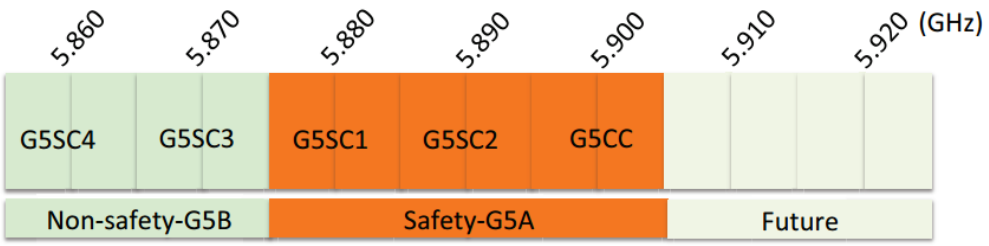
2.2. ábra ITS állomás kommunikációs architektúrája [12]

Az architektúra négy horizontális réteget (hozzáférési-, hálózati és transzport-, facilities- és alkalmazás réteg) és két vertikálist (menedzsment és biztonsági réteg) tartalmaz. Az utóbbi kettő rendelkezik rétegek közötti kommunikációs funkcióval, ami növeli az információ csere hatékonyságát és lehetőséget biztosít optimalizációra (decentralizált torlódásvezérlés (DCC- Decentralized Congestion Control [22])). Mindez a hagyományos TCP architektúrában nem lehetséges, mivel ott az egyes rétegek nem képesek közvetlenül kommunikálni, csak üzeneteken keresztül. Ennek a célja az volt, hogy ne legyen feleslegesen túlbonyolítva a struktúra, jól elkülöníthetők legyenek a funkciók. Mára már nyilvánvalóvá vált, hogy optimális használhatóság érdekében nélkülözhetetlen(ek) a vertikális réteg(ek), amivel lehetőség nyitnak az egyes rétegek működését adaptívan módosítani, ha arra szükség van. A következő alfejezetekben az ETSI által is szabványosított C-ITS kommunikációs rétegek lesznek részletesen bemutatva.

2.1.1. Horizontális rétegek felépítése

2.1.1.1. Hozzáférési réteg (Access layer)

A hozzáférési réteg az OSI modell első két rétegét, azaz a fizikai és az adatkapcsolati réteget valósítja meg. Mivel az ITS-S számtalan területen használható, így megköveteli, hogy a hozzáférési technológiák széles spektrumát támogassa (heterogén hozzáférési rendszerek). A vezeték nélküli lehetőségek közül a DSRC, WiFi, IR továbbá a celluláris hozzáférések is, mint a GSM/GPRS, UMTS, 4G, beyond 4G is elérhetők. Ezek mellett vezetékes technológiák is támogatottak, például az Ethernet, amit a fix infrastruktúrák és a Central-ITS állomás használhat.



2.3. ábra C-ITS spektrum felbontás [12]

A járműkommunikáció számára az Európai Unióban az 5,9 GHz-es sávot tartják fent, mely az 5,855 GHz-től a 5,925 GHz-es frekvenciáig használható (2.3. ábra). A tartomány három részre lett bontva. Az elsőt (ITS-G5B) az általános alkalmazások használhatják, mint például a közlekedés hatékonyságát célzók. A sáv közepén helyezkedik el a biztonságkritikus alkalmazások számára fenntartott tartomány (ITS-G5A). Ezeknek a jellemzője, hogy nélkülözhetetlen a magas prioritás és az alacsony késleltetés. A harmadik sáv (ITS-G5D) további, jövőbeli alkalmazások számára van fen tartva.

A DCC segítségével a többi réteggel együttműködve olyan teljesítményoptimalizációs lehetőségek valósíthatok meg, mint például csatornaterhelés csökkentés, optimális rádiós erőforrás kihasználás stb.

2.1.1.2. Hálózati és transzport réteg (Networking and Transport layer)

A hálózati és transzport réteg megfelel az OSI modell harmadik és negyedik, azaz hálózati és szállítási rétegének. A C-ITS architektúra lehetőséget biztosít a járművek közti adatkommunikációra anélkül, hogy fix infrastruktúrát be kelljen vonni (pont- pont, ad hoc információcsere).

A C-ITS hálózati és transzport protokollja GeoNetworking [23]alapú. Ez azt jelenti, hogy ahelyett, hogy a járművek egy IP címet kapnának, azok a pozíciójuk szerint lesznek azonosítva. A GeoNetworking segítségével hatékonyabban meg lehet valósítani a periodikusan küldött CAM (Cooerative Awerness Message) és a DENM (Decentralized Environment Notification Message) vagy egyéb üzenetek küldését, továbbá olyan alkalmazásokat is, mint a közlekedés hatékonyságát növelő vagy az információt szolgáltató V2X lehetőségek.

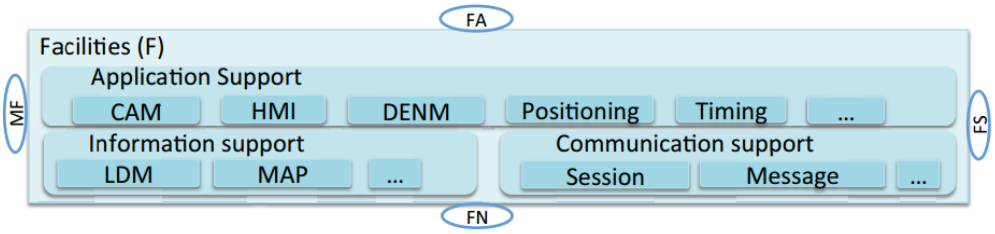
A GeoNetworking két fő funkciót definiál: Geo-addressing és Geo-forwarding. A Geo-addressing a címzési séma, melyhez a földrajzi koordinátát használja, ez alapján lehet az egyes csomópontokat azonosítani a hálózatban. A Geo-Forwarding az üzenetterjesztést valósítja meg az ad-hoc járműhálózatban, addig továbbítja azokat, míg el nem érik a céljukat.

A C-ITS ezen kívül megoldást biztosít az IP és a GeoNetworking egyesítésére (GN6SL [24]). Ezzel lehetőség nyílik IPv6 csomagokat továbbítani ad-hoc járműhálózatban és mivel a GN6ASL teljes mértékben kompatibilis az IPv6 szabvánnyal semmilyen módosításra nincs szükség.

A C-ITS architektúrában a GeoNetworking transzport protokollja a BTP (Basic Transport Protocol [25]). A BTP végpont- végpont kommunikációt tesz lehetővé ad-hoc ITS hálózatban. Az UDP-hez hasonlóan a BTP is best-effor jellegű, azaz nem vállal garanciát az elküldött csomagra. A C-ITS támogatja az általános transzport protokollok használatát is, mint a TCP, UDP és az IP, de csak nem biztonságkritikus ITS alkalmazás esetén.

2.1.1.3. Facilities réteg (Facilities layer)

A C-ITS architektúra Facilities rétege az OSI modell viszony-, megjelenítési és alkalmazás rétegébe pozicionálható. A réteg legfontosabb célja olyan szolgáltatások nyújtása, mint például a helyzet- és az idő-információk az ITS alkalmazások számára. Az általa nyújtott szolgáltatásokat három fő részbe lehet sorolni: alkalmazás-, információs- és kommunikációs támogatás (2.4. ábra).



2.4. ábra Facilities réteg az ITS-Station kommunikációs architektúrában [12]

Az alkalmazástámogatás olyan szolgálatásokat képes nyújtani, mint az ember- gép interakció LDM (Local Dynamic Map [26]), pozíció és idő. Az információstámogatás képes begyűjteni a járműre vonatkozó adatokat, monitorozni azt, továbbá értelmezni a különböző forgalmi üzeneteket. A kommunikációs támogatás az üzenetek összeállítását, értelmezését, menedzsmentjét képes ellátni, például a CAM, DENM üzeneteknek, emellett felel a címzésért, információterítésért (data dissemination), rádiós hozzáférési módválasztásért (ITS G5A, cellular, WLAN, stb.).

A CA (Cooperative Awareness [27]) az ITS-Station egyik nélkülözhetetlen szolgáltatása ahhoz, hogy felderítse a környékbeli ITS állomásokat. A CA heartbeat üzeneteket küld (CAM) időnként (1 - 10 Hz), mely tartalmazza a jármű legfontosabb tulajdonságait, például a pozíció, jármű dinamikája, jármű típusa, stb. Ezt geobroadcast üzenetben minden környékbeli ITS állomásnak eljuttatja.

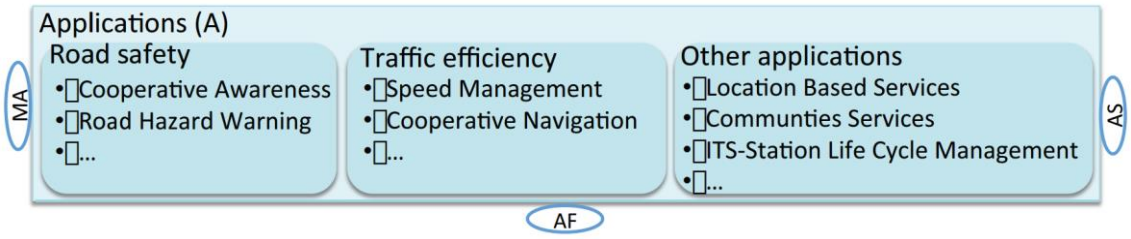
Másik fontos szolgáltatás a DEN (Decentralized Environment Notification [28]), mely az alkalmazások számára juttat el különböző környezeti eseményeket (például a forgalmat befolyásoló útfelújítási munkálatok), amik hatással lehetnek a közlekedés biztonságra, hatékonyságra. A DEN szolgáltatás készíti, menedzseli ezeket az üzeneteket, melyek tartalmazzák az eseménnyel kapcsolatos legfontosabb információkat. Ezeknek az üzeneteknek a segítségével a többi járművet időben lehet figyelmeztetni egy nem várt eseményre, így fel tudnak készülni rá. A DENM üzenetek küldését a CAM-el ellentétben lehet több ugrással is továbbítani, sőt nagy kiterjedésű terület informálása esetén el is várt ez a működés.

A CAM és DENM üzenetekből szerzett adatokat az ITS-Station az LDM adatbázisban tárolja, amihez a különböző ITS alkalmazások biztonságosan hozzáférhetnek, használhatnak. LDM-ben tárolható a környékbeli járművek és az infrastruktúra helyzete, de mivel az adatok nagy része idő és helyfüggő nélkülözhetetlen az adatbázis karbantartása, a lejárt üzenete figyelmen kívül hagyása, elvetése.

Ezeken kívül még sok más facility réteg beli protokoll létezik, mint az út topológiáját és geometriáját tartalmazó MAP [29], az útjelzések küldésére használt IVI [30], a SPaT [29], amely jelzőlámpák jelzési frekvenciáját, az egyes állapotok idejét tudja továbbítani, stb.

2.1.1.4. Alkalmazás réteg (Application layer)

Az ITS alkalmazások az alkalmazás rétegben vannak megvalósítva, ezek nélkülözhetetlenek egy biztonságosabb, hatékonyabb közlekedési rendszer létrehozásához. Ahogyan a 2.5. ábra-n is látható, az alkalmazásokat három nagy csoportba lehet bontani: közlekedésbiztonsági (Road safety), -közlekedéshatékonysági (Traffic efficiency) és egyéb információs alkalmazásokra (Other application).



2.5. ábra Alkalmazás réteg az ITS-Station kommunikációs architektúrában [12]

A közlekedésbiztonsági alkalmazások célja, a közlekedésben résztvevők biztonságának a növelése, akár életének a megmentése, azáltal, hogy veszélyes helyzetekre előre próbál figyelmeztetni. Használható például ütközések elkerülésére, megkülönböztető jelzést használó járműre- és útfelújításra figyelmeztetésre, stb.

Közlekedéshatékonyságot növelő alkalmazások célja az utak kihasználtságát optimalizálni. Emellett megoldást kíván nyújtani a környezetvédelmi problémák egy részére azáltal, hogy az üzemanyaghasználatot képes csökkenteni, optimális sebességet ajánl, képes olyan útvonalat keresni, mellyel elkerülhető a forgalmi dugó, torlódás, stb.

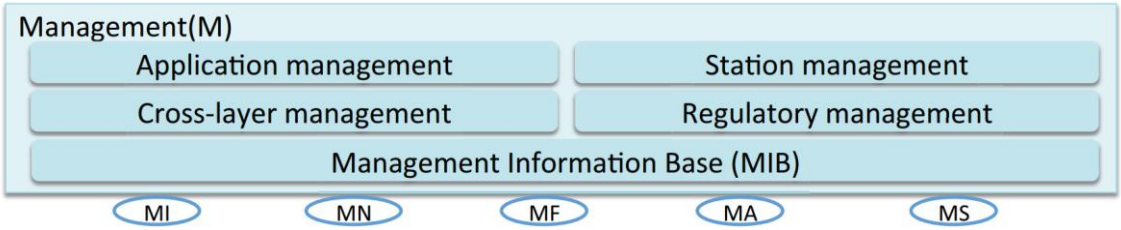
Az utolsó csoportba olyan információs szolgáltatások tartoznak, melyek főként internet alapúak, azaz inkább kényelmi szolgálatásnak tekinthetjük. Ilyen lehet egy érdekes helyre figyelmeztetés (POI), elektronikus fizetés.

A különböző alkalmazások használati lehetőségéről a 2.2. fejezetben írok részletesebben.

2.1.2. Vertikális rétegek felépítése

2.1.2.1. Menedzsment réteg (Management layer)

A rétegek közötti információ csere (cross-layer optimization) egy hatékony megoldást nyújt a teljes rendszer teljesítményének növelésében. A C-ITS kommunikációs architektúrában a menedzsment réteg rendelkezik ilyen funkcióval. A különböző rétegektől információt gyűjt, amik alapján optimális döntést tud hozni. A teljesítmény növelés céljából a gyűjtött információkat meg tudja osztani a többi réteggel is. A fő menedzsment funkciói a különböző rétegek jogosultságának kezelése, dinamikus rádiós interfészválasztás és adó teljesítményallokálás. A C-ITS öt modult definiál a menedzsment rétegben, melyek a 2.6. ábra-n is láthatók.

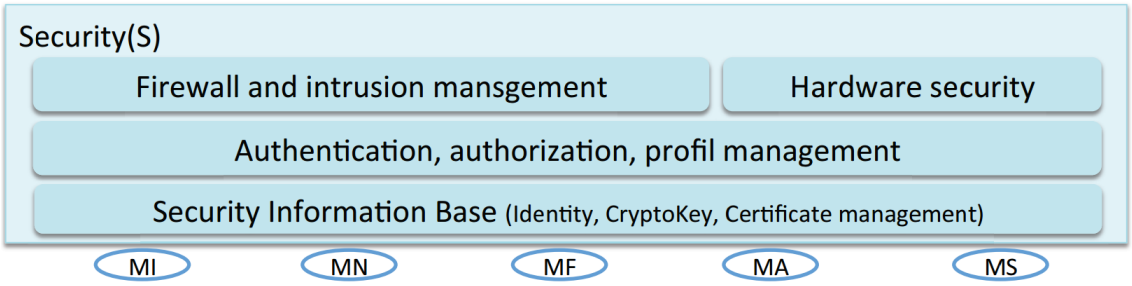


2.6. ábra A menedzsment réteg az ITS-Station kommunikációs architektúrában [12]

Az alkalmazás menedzsment (Application management) felelős az ITS állomáson futó alkalmazás telepítéséért, frissítéséért konfigurálásáért és a felügyeletéért. Az állomás menedzsment (Station management) menedzseli, konfigurálja az adott eszközt, vezéreli a kommunikációt a többi állomással, és felügyeli az LDM-et. A rétegek közötti menedzsment (Cross-layer management) a különböző rétegek közötti információcserét végzi, hogy a teljesítményt maximalizálni lehessen. A szabály menedzsment (Regulatory management) gondoskodik arról, hogy a beállított követelményeknek, szabályoknak az állomás eleget tegyen (például a spektrum használat esetén). Az utolsó modul a menedzsment információs adatbázis (Management information database – MIB), ez tárolja az állomással kapcsolatos összes adatot, profilt, szabályt, gyártóspecifikus tulajdonságokat.

2.1.2.2. Biztonsági réteg (Security layer)

A biztonsági rétegnek is meg van az az előnyös tulajdonsága, hogy rendelkezik a különböző rétegekhez kommunikációs interfésszel, ezáltal információt tud kinyerni tőlük, optimalizálni tudja a működésüket. Ez a réteg elengedhetetlen a biztonságos V2X kommunikáció biztosításához. A segítségével implementálható biztonsági mechanizmusok elengedhetetlenek, hogy a járművek közötti kommunikáció megbízható legyen, továbbá, hogy biztosítható legyen a járművek privátszféra védelme (privacy). Habár járművek közötti kommunikációt úgy tervezték meg, hogy hibatűrő és meghibásodásmentes legyen, még is ez a legkritikusabb pont. A C-ITS négy fő modult definiál ebben a rétegben, melyet a 2.7. ábra szemléltet.



2.7. ábra Biztonsági réteg az ITS-Station kommunikációs architektúrában [12]

A tűzfal és behatolás megelőző modul (Firewall and intrusion management) tartalmazza az átjárót és a tűzfalat, hogy monitorozható, ellenőrizhető legyen az adatforgalom a jármű és a kommunikációban résztvevők között, így megakadályozva egy támadást. A behatolás megelőző rendszer segít abban, hogy illetéktelenek ne férhessenek hozzá a jármű rendszereihez. A hitelesítési, hozzáférési és profil menedzsment modul (Authentication, authorization, confidentiality, profile management) biztosítja a biztonságos kommunikációt az egyes ITS állomások között, míg az információ biztonsági központ (Security Information Base) tárolja a kriptográfiai kulcsokat, és az egyéb védelmi célú adatot.

2.2. Alkalmazási lehetőségek

Járműkommunikációs alkalmazások széles választéka létezik már a közlekedésbiztonsági alkalmazásoktól, melyek a sofőrt tudják segíteni vezetés közben, egészen az utasok számára nyújtható különböző multimédiás alkalmazásokig.

Elsődleges célja a jármű-kommunikációnak olyan közlekedésbiztonsági alkalmazások támogatása, melyek a vezető és az utasok életét védi. Ennek módja a balesetek valószínűségének minimalizálása azáltal, hogy hasznos információkkal látjuk el a sofőrt vezetés közben. Emellett a közlekedés hatékonyságát is tudjuk növelni, mivel jelezni lehet a különböző forgalmi állapotokat (dugó, baleset, stb.), és kerülőutat ajánlva ezek negatív hatása elkerülhető.

A következőkben a jármű-kommunikációs alkalmazások három csoportjából mutatok be érdekesebb alkalmazási lehetőséget, de egy részletesebb leírást, megvalósítási lehetőségeket ajánló oldal elérhető itt [31].

2.2.1. Közútbiztonsági alkalmazások [32]

A világban évente 1,3 millió ember hal meg autóbalesetben és további 20- 50 millióan sérülnek meg, válnak mozgásképtelenné [33]. Fontos cél ennek a számnak a csökkentése, de ezt nehezíti, hogy a járművek száma folyamatosan növekszik. Az utasok élete attól függ, hogy a sofőr milyen gyorsan tud reagálni egy hirtelen bekövetkezett eseményre, forgalmi dugóra. Emellett nagyobb eséllyel következhet be baleset rossz időjárási körülmények, gyenge látótávolság miatt. A vezetés időtartamától függően a sofőr reakcióideje is jelentősen csökkenhet, aminek következtében hibákat követhet el vezetés közben. Jelenleg nincs olyan automata rendszer, mely gyorsabban tudna válaszolni ilyen megváltozott helyzetre, így kutatók, fejlesztők dolgoznak azon, hogy a ve-

zetést segítő rendszerek minél megbízhatóbban tudják előre jelezni a szenzorjaik segítségével a különböző forgalmi szituációkat, és segítség napjaink sofőrjeit, valamint a jövő szemi-autonóm/autonóm járműveit. Napjaink megoldásainak (pl. Google Car) az a hátránya, hogy csak lokális adatokkal tud dolgozni, ami sokszor nem elég egy döntés meghozásakor.

Emiatt vált szükségessé olyan proaktív biztonsági megoldások létrehozása, melyek lehetővé teszik a járművek közötti kommunikációt, és így egyre több baleset válhat elkerülhetővé. Annak érdekében, hogy ez a megoldás hatékony legyen nem elég csak a járművek közötti kommunikáció, nélkülözhetetlen, hogy a járművek a különböző infrastrukturális elemekkel is kapcsolatba léphessenek. Ilyen kooperáció történhet például jelzőlámpák között is, hogy a jelzési szekvenciák különböző forgalmi igényekhez alkalmazkodjanak, így például csúcsforgalom idején növelhető a zöld jelzés időtartama, annak érdekében, hogy több jármű áthaladhasson. Nyilván ahhoz, hogy ezek az alkalmazások megfelelően használhatok legyenek, széles körben el kell, hogy terjedjenek, melyhez még sok idő szükséges.

A kooperáció során a járművek, közlekedési eszközök, infrastrukturális elemek megoszthatják egymással különböző tulajdonságaikat, mint pozíció, sebesség, irány, szenzorjaikkal szerzett egyéb adatok, stb. Ezen felül képesek lehetnek a megszerzett információkat továbbküldeni, így nagyobb távolságból már fel lehet készülni egy hirtelen fékezésre, megkülönböztető jelzést használó járműre, veszélyes útvonalakra. Ezeket az információkat használva a közlekedésbiztonsági alkalmazások tájékoztatni, lehetőség szerint segíteni tudják a sofőröket egy-egy veszélyes helyzet elkerülésében. A következőkben néhány érdekesebb ilyen alkalmazási lehetőséget fogok bemutatni.

2.2.1.1. Kooperatív keresztező ütközésre figyelmeztetés (Cooperative Intersection Collision Warning) [32]

Ez egy olyan rendszer, mely figyelmeztetni tudja a sofőrt, ha a kereszteződésben ütközés történhet. Ehhez a RSU-ok is játszhatnak közvetítő szerepet. Részletes térkép szükséges a kereszteződésről, illetve ismerni kell a sávok számát, záróvonalak- sávfestések elhelyezkedését. Mivel a járművek periodikusan üzenetet küldenek saját magukról, így az RSU a dinamikájuk alapján ki tudja számolni, ha kereszteznék egymás útját.

2.2.1.2. Megkülönböztető járműre figyelmeztetés (Approaching Emergency Vehicle Warning) [32]

A megkülönböztető jelzést használó jármű figyelemeztető üzenetet küld a többi járműnek, akik tudni fogják, hogy merre van a szirénázó jármű, merre szeretne tovább menni, így hatékonyan tudnak utat engedi neki.

2.2.1.3. Veszélyes helyzetre figyelmeztetés (Hazardous Location Notification) [32]

Veszélyes helyzetekre képes figyelmeztetni, amely észlelhető egy jármű érzékelője segítségével, vagy akár egy RSU is detektálhatja az általa lefedett területen. Ilyen információ lehet a vizes, csúszós út egy bizonyos sávban/irányban. Ezt a jármű érzékelheti ESP (Electronic Stability Program - menetstabilizáló) segítségével, majd üzenetet

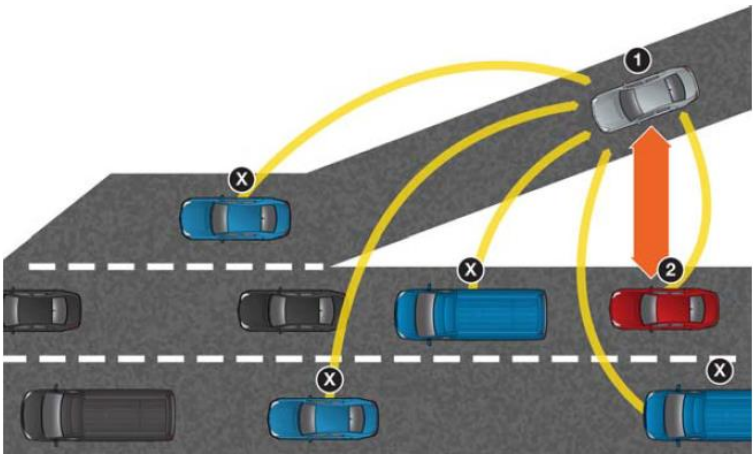
küld a többi járműnek, melyben a helyzetét veszélyesnek jelöli meg, szükség esetén sebességet is képes ajánlani a többi járműnek. Az üzenetet továbbíthatják az RSU-k, így a járművek időben felkészülhetnek a veszélyes helyzetre.

2.2.1.4. Hirtelen fékezésre figyelmeztetés (Emergency Electronic Brake Lights) [32]

Az a jármű, amely hirtelen nagy fékezésre kényszerül, üzenetet küld a mögötte lévő járműveknek, hogy időben megkezdhessék a fékezést. Szükséges, hogy a járművek időnként üzenetet küldjenek egymásnak, ismerjék egymás helyzetét, és hogyha szükséges, tudják, hogy kinek kell küldeni az üzenetet.

2.2.1.5. Kooperatív besorolási segéd (Cooperative Merging Assistance) [32]

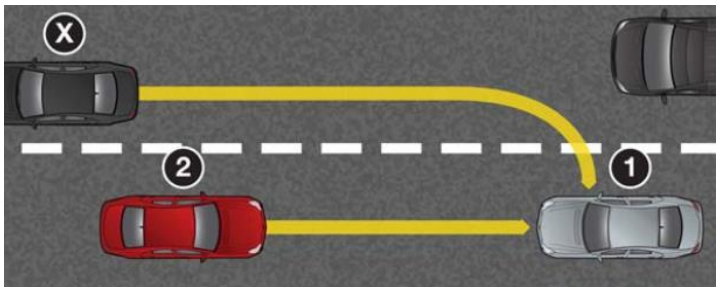
Ez az alkalmazás lehetőséget biztosít a járműnek, hogy biztonságosan becsatlakozzon/besoroljon a forgalomba anélkül, hogy megszakítaná annak dinamikáját. A rendszer segít a sofőrnek eldönteni, hogy hova soroljon be. Ahogyan a 2.8. ábra is mutatja, V1 jármű be akar sorolni, ehhez üzenetet küld minden egy ugrás távolságban lévő a szomszédos járműnek. A sebesség, irány, pozíció alapján V2 látja, hogy kétirányú kommunikációt szükséges kezdeményeznie V1-el, melyben megtárgyalhatják a besorolást. Ez az alkalmazás amellett, hogy a közlekedés biztonságot segíti, a hatékonyságáért is tesz, mivel a becsatlakozást dinamikussá teszi, megelőzve a feltorlódást a gyorsítósávban.



2.8. ábra Kooperatív sávegyesítési segéd [32]

2.2.1.6. Rossz útirány figyelmeztetés (Wrong Way Driving Warning) [32]

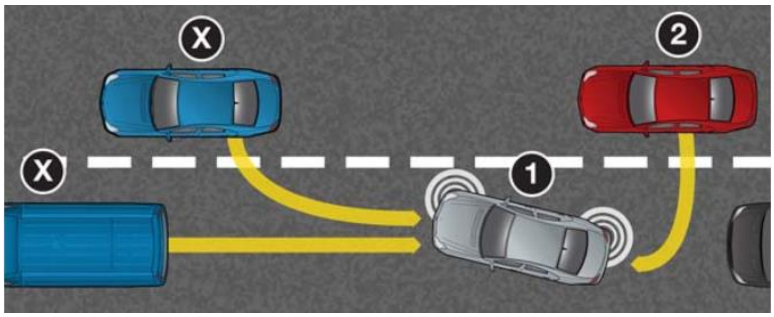
Az alkalmazás észleli, hogy egy másik jármű rossz irányban közlekedik, nem abban a sávban halad, amelyben menetirány szerint neki lennie kellene, majd figyelmezteti a sofőrt, hogy ütközés lehetséges.



2.9. ábra Szembe vezetésre figyelmeztetés [32]

2.2.1.7. Sávváltási segéd (Lane Change Assistance) [32]

A rendszer figyelmezteti a sofőrt, hogy megváltozik a sávja, és jelzi, hogy a szomszédos járművek hogyan helyezkednek el. Továbbá együttműködik a holttér figyelő alkalmazással is. A rendszer segítségével elkerülhetők a oldalsó ütközések, segíti a sávváltást rossz látási viszonyok között.



2.10. ábra Sávváltási segéd [32]

2.2.2. Közlekedéshatékonyságot növelő alkalmazások [32]

Ez az alkalmazási terület a forgalmi dugók csökkentését és az üzemanyag hatékonyabb használatát segíti elő. Manapság a sofőrök a világ összes nagyvárosában, autópályákon, forgalmas utakon ugyanazzal a problémával szembesülnek: a közlekedési dugókkal. Amellett, hogy ez komoly ingerültséget vált ki a résztvevőkben a környezetszennyezést is növeli és komoly pénzbe kerül. Emiatt ezeknek az alkalmazásokoknak kulcspontja a forgalom gördülékenyebbé tétele azáltal, hogy képesek észlelni a forgalmi dugókat, torlódásokat, így lehetőséget adva arra, hogy ezeket elkerüljük. Ennek következményeként az üzemanyag fogyasztást csökkenteni lehet, ami költséghatékonyság mellett a károsanyag kibocsátásra is jó hatással van. A következőkben néhány példát mutatok az ilyen típusú V2X alkalmazásokra.

2.2.2.1. Zöld hullám (Green Light Wave) [32]

A jelzőlámpa vagy az út menti infrastruktúra elküldi a következő kereszteződés helyét, geometriáját és azt, hogy mennyi időnként, milyen hosszan van zöld jelzés. Ezek ismeretében a jármű ki tudja számolni azt a sebességet, ami szükséges ahhoz, hogy ne kelljen megállnia. Ennek segítségével a forgalom áteresztőképessége gördülékennyé tehető, csökkenthető az üzemanyag felhasználás, kevesebb megállást, elindulást ered-

ményez. Emellett a kommunikáció lehetőséget teremt arra is, hogy megnövekedett forgalom hatására a zöld jelzés ideje dinamikusan változzon.

2.2.2.2. Továbbfejlesztett útvonal választó és navigáló rendszer (Enhanced Route Guidance and Navigation) [32]

Az alkalmazás feladata dinamikus útvonal újratervezés friss közlekedés információk alapján. Tájékoztatja a sofőrt a várható késésről, illetve a forgalmi állapot alapján jobb utat ajánl. Jelenleg is léteznek megoldások, melyek a dinamikus újratervezésben segítenek, mint a TMC (Traffic Message Channel- forgalmi információ [34]), melyhez a GPS alapú navigáció számításba tudja venni az esetleges forgalmi dugókat, elkerülheti azokat. Emellett létezik olyan lehetőség is, mint a TomTom és valós idejű szolgáltatása [35], mely internet segítségével nyújt hasonló megoldást. Ezenkívül manapság az okostelefonok terjedése miatt, a Waze [36] is nagyon népszerű. Ez egy közösségi alapú navigációs eszköz, mely a felhasználóktól gyűjtött és szerkesztett adatok alapján képes útvonalat tervezni. Ezeknek a hátránya, hogy a mobilinternetért és TomTom esetén rendszerint a szolgáltatásért is fizetni kell, illetve a valós idejű adat rendszerint 10-15 percenként frissül. Ez a hátrány a TMC esetén is adott, általában az üzenet a központból 30 másodperc alatt ér el a járműhöz.

A jármű-kommunikáció ezzel szemben sokkal gyorsabban juttatja el az információkat a járművek között, emellett sokkal gyakrabban is frissíti a többi közlekedésben résztvevő eszköz ismerete alapján. A legnagyobb előnye, melyet a TMC sem tud, az adott területre küldött, geografikus pozícionáláson alapuló üzenetek (geocast). Ez segíthet elkerülni azt a hibát, hogy mindenkit egy új, jobb útvonalra terel, így eldugítva azt is. A geocast használatával lehetőség van csak egy bizonyos területen tartózkodó járműveket elterelni. Az alkalmazás előnyét az is növeli, hogy az RSU-k gyűjtik a járművek telemetria adatait (ezt a rendszeresen küldött beacon üzenetekből nyeri ki), melyet aztán egy központi egység analizál, hogy optimális útvonalat ajánlhasson.

2.2.3. Egyéb információs alkalmazások [32]

Ez az alkalmazási csoport olyan szolgálatásokat tartalmaz, mint a járművön belüli szórakoztatás (zene, film) vagy ad-hoc szolgáltatások, amelyek csak egy rövid időre jönnek létre (útdíjfizetés), illetve egyéb olyan szolgáltatás, mely internet hozzáférést igényel (web böngészés az utas számára). Néhány példát gyűjtöttem össze az alábbiakban.

2.2.3.1. Érdekes helyekre figyelmeztetés (Point of Interest (PoI) Notification) [32]

Az infrastruktúra a környékbeli érdekes helyekről értesítéseket küld. Ezek a POI-k lehetnek látványosságok, szálláshelyek boltok, benzinkutak és egyéb a sofőr számára érdekes helyek. Ezen felül az RSU küldhet olyan információkat is, mint az étterem nyitva tartása, árak és napi menü, ajánlatok, akciók, stb. Maga a POI nem új találmány, ma is használnak ilyeneket GPS alapú navigációs eszközök, de ezek statikusak, nem tudnak másodpercre friss információt szolgáltatni a közeli helyekről.

2.2.3.2. Áthajtásos fizető kapu (Drive-Through Toll) [32]

Ez az alkalmazás lehetővé teszi az autópálya fizetőkapuban a fizetést anélkül, hogy meg kellene állni. Az elektronikus útdíjrendszer érzékeli a járművet és végrehajtja a fizetési tranzakciót. Ezáltal szükségtelenné válik, hogy egy ember/gép beszedje a díjat, megálljon a jármű: folyamatos haladás válik lehetővé, elkerülve a torlódást, dugót.

2.2.3.3. Flotta menedzsment (Fleet management) [32]

A flotta menedzsment lehetővé teszi a járművek állapotának távoli elérését, helyzetének nyomon követését. Így lehetővé válik a távoli diagnosztika, a sofőr és az üzemanyag felügyelete, menedzsmentje.,

2.2.4. V2X alkalmazások összefoglalása

A bemutatott néhány alkalmazásból látszik, hogy széles körben képesek segíteni a sofőrt, megmenteni az utasok, közlekedésben résztvevők életét. Mivel egy olyan rendszerben, ahol a mozgás elengedhetetlen a működéshez, és a fejlesztés folyamatában nem lehet megoldani, hogy minden tesztelést, fejlesztési lépést valósan mozgó járművökön hajtsanak végre, ebből adódóan nélkülözhetetlen a szimulációs eszközök alkalmazása. Ilyenek nélkül az alkalmazások, a hozzájuk szükséges protokollok nem lehetnének megbízhatók, vagy nagyon költségessé tenné a fejlesztésüket. Ezért a következő (3.) fejezetben jármű-kommunikációs szimulációs eszközöket mutatok be.

3. Létező szimulációs rendszerek

A vezeték nélküli kommunikációs technológiák nagy hatást gyakorolnak a mindennapi életünkre, kezdve a különböző beltéri megoldásoktól a cellás hálózatokig. Manapság az ad-hoc járműhálózatok (VANET) egyre nagyobb figyelmet kapnak, fejlesztésük jelenleg is zajlik. Több kutató/fejlesztő csapat készített már VANET szimulációs szoftvert, hogy tanulmányozzák, vizsgálják a különböző média hozzáférési módokat, forgalomirányítást és az egyes kommunikációs protokollokat. A VANET szimulációk különböznek a MANET (mobile adhoc network - mobil ad-hoc hálózat) szimulációktól, mivel a járműhálózatokban néhány speciális kihívással is meg kell küzdeni. Ilyen megkötés pl. az úthálózat, az út menti akadályok okozta többutas terjedés, a közútforgalmi modellek, változó járműsebesség, jelzőlámpák, forgalmi dugók, vezető viselkedése stb. Léteznek már forgalomgeneráló, hálózatszimuláló és VANET tesztalkalmazások is. Ebben a fejezetben a két legfontosabb VANET szimulációs szoftvert mutatom be, szemléltetve használati lehetőségeiket, előnyeiket és hiányosságaikat [37].

3.1. Veins [38]

A Veins (Vehicles in Network Simulator- Járművek hálózati szimulátorban) egy hibrid szimulációs keretrendszer. Két részből áll: egy a hálózati – melynek a szimulációját az OMNET++ valósítja meg – továbbá egy járműforgalmat generáló eszközből, a SUMO-ból. A program alkalmazásának fő célja járművek közötti kommunikációs (IVC - Inter Vehicle Communication) protokollok valóságosabb modellezése a VANET hálózatban.

3.1.1. Hálózati szimulátor [38]

A hálózati szimulátorokat főként arra használják, hogy mielőtt a hálózatot valóságban elkészítik, azelőtt modellezzék, teszteljék az architektúrát és a protokollokat. Így különböző beállítások mellett összehasonlíthatják a teljesítményt, észrevehetik az esetleges gyenge pontokat anélkül, hogy a valóságban is implementálni kellene. Emellett a hálózati szimulációs eszközöket széles körben használják kutatásokban is, hogy megvizsgálják az új protokollok, megoldások viselkedését, teljesítményét. Ilyen célú szimulációs eszközből több elérhető, mint például a nyílt forrású ns-2 [39], OMNeT++ [40], J-SIM [41] és JiST/ SWANS [42], továbbá léteznek kereskedelmi eszközök is, mint a OPNET [43] . Ezeknek a programoknak az alapelve hasonló, a különbség az elérhető modellek számában van, azaz tipikusan a használt MAC és forgalom irányítási protokollokban.

A Veins a hálózati szimulációs eszköznek az OMNeT++-t használja az INET keretrendszerrel [44], hogy különböző VANET protokollokat egyszerűbb legyen szimulálni. OMNeT++-ban a hálózatot hierarchikus modulokból lehet összeállítani, amiket C++-ban lehet elkészíteni. A modulok közötti kapcsolatot, kommunikációs lehetőségeket a NED (Network Description - hálózat leíró) fájlban kell definiálni. A szimulációs rend-

szer lehetőséget biztosít mind grafikus, mind parancssoros fejlesztésre is. Az INET keretrendszer biztosít az OMNeT++-ban olyan modulokat, amely a valódi internet protokoll használatát teszik lehetővé (például: TCP, UDP, IPv4, ARP). Ezen kívül lehetőséget biztosít mobil csomópontok és IEEE 802.11 alapú átvitelt használó modulok közötti kommunikációra is. Mobilitási lehetőségre általában csak korlátozottabb minták elérhetőek. A legtöbb hálózat szimulációs keretrendszer a Random Waypoint vagy a Manhattam mobilitási modellt támogatja. Sajnos ezek a modellek VANET hálózatokban forgalom szimulálására nem alkalmasak.

3.1.2. Forgalomszimulátor [38]

A legtöbbször a realisztikus mozgásszimulációhoz a csomópontok olyan útvonalat követnek, amelyet előre rögzítettek, így a mozgáshoz csak ki kell olvasni azt egy fájlból. Az ilyen esetekben sokszor csak egy adott forgatókönyvet lehet leszimulálni, így minden egyes esethez egyedi útvonalakat kell létrehozni. Amennyiben csak egy paramétert szeretnék változtatni, például a forgalom sűrűségét, de minden mást változatlanul hagynánk, akkor az ezzel a módszerrel megvalósíthatatlan. A szcenárió minden részletre kiterjedő paraméterezhetőségét csak úgy lehet megvalósítani, ha mozgást forgalomszimulációs eszközzel valósítjuk meg.

Hagyományosan a forgalomszimulációs modelleket három részre tudjuk bontani: makroszkopikus, mezoszkopikus és mikroszkopikus osztályra aszerint, hogy milyen részletességgel akarjuk a forgalmakat vizsgálni. A makroszkopikus modellek, mint a METACOR [45] a forgalmat nagy felbontásban kezelik (mint egy folyam), így gyakran alkalmaznak hidrodinamikai elveket a járművek viselkedésének a szimulálására. A mezoszkopikus modellek, mint a CONTRAM [46], egy adott szakasznak kezelik a mozgását, aggregált sebesség sűrűség függvénnyel modellezik a viselkedést. Ezzel szemben a VANET szimulációkban az egyes mobil csomópontokhoz rádiós átvitel van hozzárendelve, így nélkülözhetetlen ismerni a szimulált csomópontok pontos helyét. Mind a makroszkopikus, mind a mezoszkopikus modell ezt nem teszi lehetővé, csak a mikroszkopikus nyújt arra megoldást, hogy egyedi járműveket és interakciójukat szimuláljuk VANET csomópontként.

Szállítási és közlekedési szcenáriókhoz számtalan mikroszintű szimulációs modellt fejlesztettek, szinte mindegyik más-más megközelítésben működik, ennek eredményeként ezek különböző komplexitásúak. Ilyen modellek például a Cellular Automaton (CA) [47], SK car-following [48], IDM/MOBIL [49] [50]. Minden modellnek megvan a maga előnye és hátránya is. A modellek pontossága minden esetben megfelelő, de sok esetben hatékonyabb lehet az egyszerűbben megvalósított modellt választani, mivel egy komplexebb nem fog feltétlen jobb megoldást nyújtani. Lényegében ez azt jelenti, hogy addig, amíg a hálózat szimulációra koncentrálunk, minden mikroszintű szimuláció megegyezik a mobilitási modellel.

A forgalomszimuláció a Veinsben szintén mikroszkopikus modell szerint van megvalósítva a SUMO [51] segítségével, mely a SK mobilitási modellt használja. Az eszköz lehetőséget biztosít mind GUI-val, mind nélküle történő a használatra és a város térképek importálására is sok féle fájlformátumot ajánl. A SUMO lehetővé teszi a kiterjedt

hálózatok szimulációját teljesítménybeli korlátozások nélkül, támogatja a több sáv használatát, ezeken keresztező forgalmat biztosít, melyek között interakciót jobb kéz szabály vagy jelzőlámpa használatával oldja meg. A járművek típusa szabadon változtatható, a járművek követhetnek előre elkészített statikus vagy dinamikusan létrejövő utakat is.

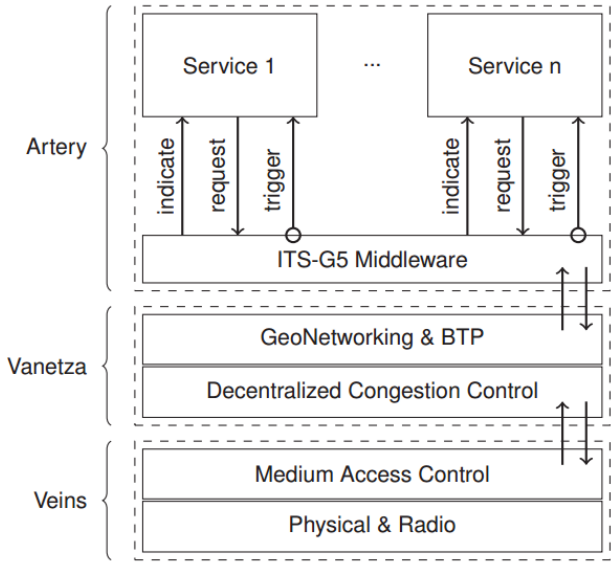
A mikroszkopikus forgalom szimulációk és az IVC protokoll együttes használata, mély betekintést tud nyújtani a VANET protokollok működésébe. Tehát ezeknek a szimulációs komponensek nélkülözhetetlen, hogy együttműködjenek, közöttük létrejöjjön a kétirányú kapcsolat.

3.1.3. Kétirányú kapcsolat az eszközök között [38]

Kétirányú kapcsolat létrehozása az OMNeT++ és a SUMO között elengedhetetlen, mivel szimuláció alatt egymásnak parancsokat, mobilitási útvonalakat küldenek. OMNeT++ esemény alapú szimulációs eszköz, tehát úgy kezeli le a mobilitást, hogy a csomópont lépéseit szabályos intervallumba helyezi. Ez a megoldás teljesen illeszkedik a SUMO működéséhez, mivel a szimulációs időt diszkrét lépésekként kezeli. Az OMNet++ minden lépésben elküldi a pufferében található parancsokat a SUMO-nak és az adott lépést triggereli a forgalomszimulátortól. A forgalomszimulátor minden lépés befejezéseként visszaküldi parancsok sorozatát és az összes járműpéldánynak a pozícióját az OMNet++-nak. A parancsváltásnak köszönhetően a SUMO képes megállítani, újraindítani, elterelni a járműveket, hogy elkerüljenek bizonyos útszakaszokat, forgalmi helyzeteket.

3.1.4. Artery [52]

Az Artery a Veins szimulációs eszköz egy bővítménye, mely az ITS kommunikációs lehetőségeit egészíti ki az ETSI ITS G5 protokoll stackkel (GeoNetworking, BTP). Így az Artery segítségével még realisztikusabban lehet a VANET alkalmazásokat szimulációs környezetben tesztelni.



3.1. ábra Artery architektúra [52]

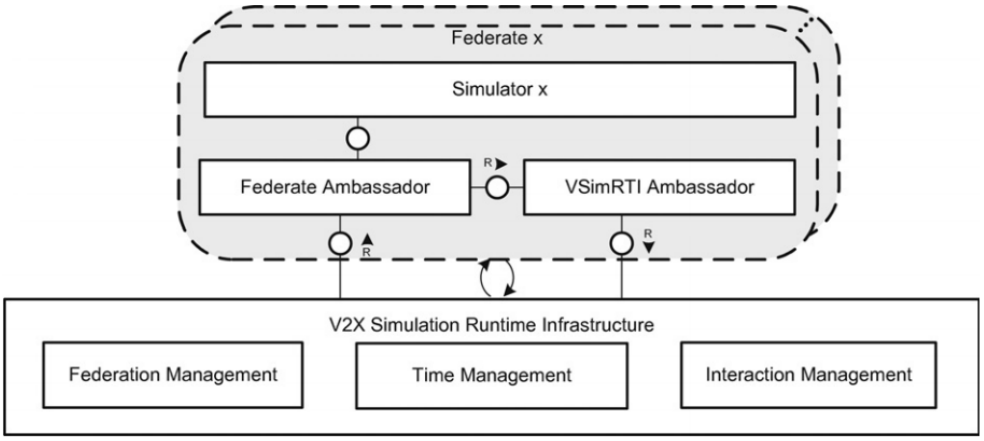
Ahogyan a 3.1. ábra-n is látható a Veins két blokkal lett kiegészítve, az így létrejött architektúrában ő biztosítja a hozzáférési és a MAC réteget. A köztes blokk a Vanetza, ami megvalósítja az ETSI ITS G5 hálózati és transzport protokollokat továbbá támogatja a DCC [22] mechanizmust is. Az Artery egy flexibilis keretrendszert biztosít a különböző alkalmazásoknak (szolgálatásoknak), ezzel elősegítve az új VANET alkalmazások fejlesztését és tesztelését. Az Arteryben a kulcs elem az ITS-G5 Middleware, mivel gerincét jelenti a különböző szolgáltatások számára azáltal, hogy információs központként viselkedik, továbbá interfészt is biztosít a komponensek számára, forgalom irányítást végez közöttük.

3.2. VSimRTI [6]

A V2X Simulation Runtime Infrastructure (VSimRTI- V2X Futásidejű Szimulációs Infrastruktúra) a jármű mozgását, V2X kommunikációs technológiákat és celluláris hálózatokat képes modellezni. VSimRTI az egyik legflexibilis rendszer a járműipari kutatásban, mivel dinamikusan szimulálja a különböző alkalmazásokat és értékeli azok hatását és előnyeit. A VSimRTI számos szimulációs eszköz segítségével képes ITS rendszereket megvizsgálni különböző szempontok szerint. A könnyű integráció és szimulációs eszközök cserélhetősége lehetővé teszi, hogy a még realisztikusabban lehessen megvizsgálni a járműforgalmat, a vezeték nélküli kommunikációt, a sofőrviselkedést és modellezni az alkalmazásokat.

3.2.1. VSimRTI koncepciója [53]

A szimuláció legnagyobb nehézsége, kihívása, hogy különböző szimulációs eszközöknek, területeknek kell együtt dolgoznia: járműforgalom, vezeték nélküli kommunikáció és alkalmazás modellezés együttesen kezelendő. A probléma megoldásaként a VSimRTI több szimulációs eszközzel működik együtt, és a különböző kihívásoknak megfelelően képes akár lecserélni is azokat.



3.2. ábra VSimRTI architektúrája [53]

A VSimRTI szimulációs rendszerek szövetségét készíti el, melynek a menedzseléséért a futásidejű infrastruktúra felel. Ahogyan a 3.2. ábra mutatja, egy létező szimulációs eszköz egy szövetségbe (Federate) van integrálva, úgy, hogy két nagykövet (Ambassador) biztosítja az interfészt, melynek segítségével a szimulációs eszközt be lehet csatlakoztatni a keretrendszerbe. Ez a nagyköveti architektúra lehetővé teszi a különböző interfészek egyszerű hozzáadását, továbbá olyan interfészt biztosít, mely lehetővé teszi a távoli vezérlést. A módszer előnye, hogy egy új eszköz csatolása esetén csak a két nagykövet interfészt kell definiálni, ami képes a két rendszer közötti kommunikációt fordítani.

A szövetség menedzser (Federation Management) felelős az egyes résztvevők életciklusáért. Az idő menedzser (Time Management) nélkülözhetetlen a szimulációs folyamat koordinálásában és a különböző eszközök közötti szinkronizációban. A különböző szövetségek között az információ cserét az interakció menedzser (Interaction Management) vezéreli le.

Azonnali használhatóság céljából a következő szimulációs eszközökkel működik már együtt alapból a VSimRTI: forgalom szimulátornak a VSSIM és a SUMO, kommunikációs szimulátornak JiST/SWNAS és OMNeT++, alkalmazás szimulátornak a VSimRTI_APP van előkészítve.

3.3. Szimulációs rendszerek összefoglalása

A bemutatott, napjainkban leginkább elterjedt jármű-kommunikáció- és járműforgalom-szimulációs keretrendszerek alapján látható, hogy ezeket a rendszereket nem lehet egy komponenst használva összeállítani. Mivel egy új hálózattípusról lehet beszélni (VANET), ahol a MANET elvek mellett megjelennek a járművek mozgását igénylő feladatok is, ezért a már létező komponensek újrafelhasználása sokat segít a szimulációban. Ezért nyújt hatalmas segítséget az olyan létező szimulációs eszköz, mint az OMNeT++ és a SUMO. Ezeket összefogó szimulációs keretrendszerek jó hatásfokkal tudnak szimulálni valós körülményekre hasonlító szituációkat, nélkülözhetetlenek a IVC protokollok és V2X alkalmazások fejlesztésében és tesztelésében.

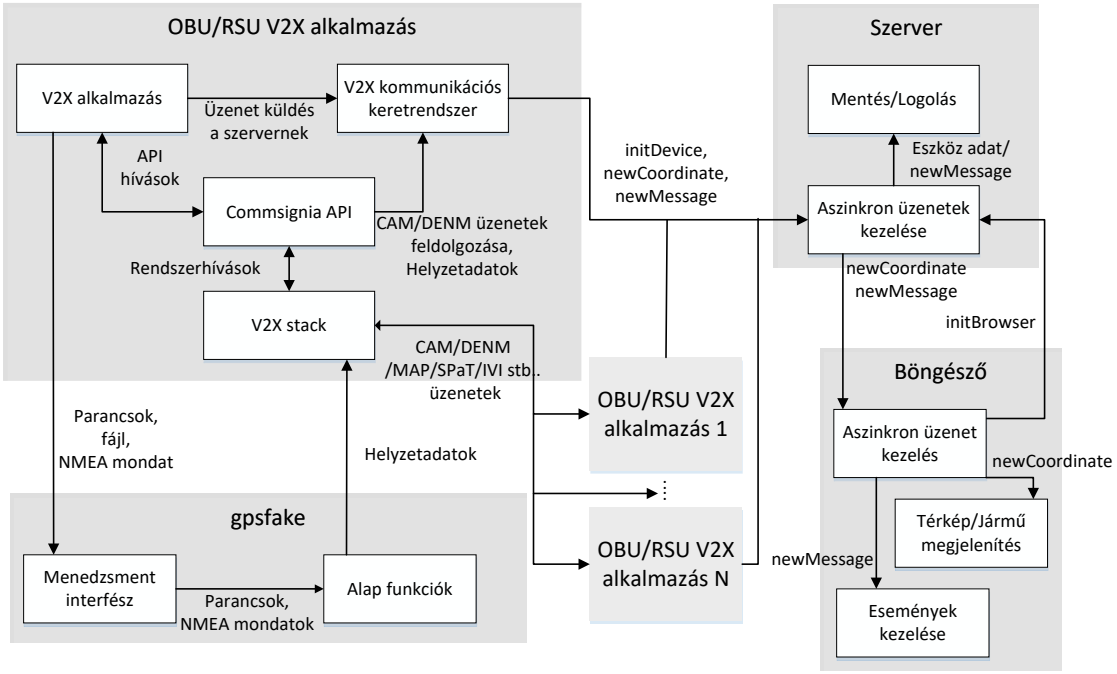
A legnagyobb kihívást egy ilyen rendszernek az elkészítésében az jelenti, hogy a különböző eszközöket összehangolják, felépüljön közöttük egy kommunikációs híd, melynek segítségével a szimulációs rendszer hatékony lehet. Erre példa a SUMO és OMNeT++ között létrehozott TraCl (Traffic Command Interface) interfész. További példa a VSimRTI által alkalmazott nagyköveti architektúra, mely bármely olyan szimulációs eszköz becsatlakoztatását lehetővé teszi, amihez elkészült az interfész, ami a képes a VSimRTI magja és a becsatolt eszköz közötti hívásokat átfordítani.

A dolgozatomban bemutatott szimulációs keretrendszert (4. fejezet) hosszútávon én is hasonló képességekkel szeretném felfejleszteni, de egy ilyen komplex rendszert összeállítani sok idő, munka. Mélyrehatóan meg kell ismerni azt adott eszközök által használt parancsokat, elvárt bemeneteket és ennek megfelelően egy interfészt készíteni, ami képes a valós V2X eszközökkel és az egyéb modulokkal kapcsolatba lépni. Mivel munkám célja az volt, hogy valós eszközökkel végezzem a szimulációt, és ehhez rapid prototyping módszerrel minél hamarabb, egy működő rendszert összeállítani, ezért ez nem tette lehetővé, hogy létező rendszereket beimplementáljak a keretrendszerbe. Számomra a legnagyobb előnyt az jelentette, hogy a protokoll stack és a V2X alkalmazás szimulálására nem kell külön eszközöket igénybe venni, ezt az OBU/RSU implementáció megoldja. Ennek következményeként jelenleg nincsen forgalomszimulációs eszköz bekötve az általam létrehozott rendszerbe, csak egy ezt lehetővé tevő megoldás (lásd később). Az általam elvárt funkció -valós ITS stackkel tesztelni szimulációs eszközökben- a már létező szimulációs eszközöknek is hiányossága, ezért mindenképpen fontos ezt megvalósítani, hogy szimulált környezetben minél jobban lehessen tesztelni a valós V2X alkalmazásokat.

4. Megvalósított keretrendszer

4.1. Architektúra

A létrehozott keretrendszer célja, hogy valós eszközökön futó valós V2X alkalmazásokat lehessen laborkörülmények között fejleszteni, tesztelni, analizálni. Ehhez mozgásemulációs és megjelenítési eszközöket készítettem a hatékonyabb vizsgálat miatt. Az létrehozott keretrendszer három fő komponensből áll: a mozgásemulációért felelős gpsfake és általam végrehajtott kiegészítései (5. fejezet), a V2X alkalmazás és a keretrendszere (6. fejezet), valamint és a szerver és a járművek megjelenítéséért felelős böngésző, melyeknek a működését ebben a fejezetben fejtem ki.



4.1. ábra A létrehozott keretrendszer felépítése

Az elkészült keretrendszer általános felépítését az 4.1. ábra mutatja. Az egyes nyilakon az általam elkészített üzenetek típusa, vagy a két modul közötti információfolyam van jelölve. Két továbbítási módszert használok az üzenetek küldésére: az egyszerű TCP socketet [54] és a websocketet [55]. Mindkét esetben port - IP cím párossal vannak azonosítva a kommunikációs felek, de a második típust a böngésző is tudja használni. Ennek az a nagy előnye van, hogy miután kiépült a szerver és a böngésző közötti kapcsolat, a szerver anélkül tud üzenetet küldeni, hogy korábban kérést kellett volna neki küldeni. Ez nagymértékben képes az üzenetek mennyiségét csökkenteni, mivel nem kell a böngészőnek folyamatosan kérésekkel fordulnia (pollingozni) a szerverhez, amikor új adat érkezik, azt a szerver egyből el tudja küldeni. Mivel a szerver és a böngésző websocketen kommunikál, ezért egyszerűbbnek láttam, hogy a szerveren nem implementálok más kommunikációs módszert, ezt használom a V2X alkalmazásokkal történő üze-

netváltáshoz is. Ez az alkalmazás esetén semmilyen előnnyel nem jár, csak a szerver megvalósítását egyszerűsítette le. A websocketben JSON (JavaScript Object Notation) [56] objektumokat küldenek egymásnak a résztvevők, mivel egy üzenetben egyszerre több adatot is el kell küldeni, a feldolgozása sokkal egyszerűbb, mintha csak egyszerű szövegben kerülne sor a kommunikációra.

Az architektúra egyik nagy komponense a járműkommunikációs eszköz (OBU/RSU V2X alkalmazás), ami két részből áll. Először is a V2X alkalmazás, amely képes például a 2.2. fejezetben található használati esetek, szolgáltatások közül egyet vagy többet megvalósítani. Jelen dolgozatomban a vészhelyzeti járműre figyelmeztető alkalmazással szemléltetem a működést, aminek a leírása a 6.2. alfejezetben található. Az eszköz másik komponense a szerverrel történő kommunikációt menedzselő keretrendszer. Mivel a szerver és az eszköz között egy meghatározott felépítésű JSON objektumban történik a kommunikáció, sokat segít a használatán, ha a V2X alkalmazást író fejlesztőnek nem kell tisztában lennie az üzenet felépítésével, használatával. A keretrendszer figyeli az eseményeket (új koordináta érkezése, CAM/DENM/MAP/SPAT/IVI/stb. üzenet), azokat lekezeli, elküldi a szervernek. Ezenkívül üzenet küldésre megfelelő függvényeket biztosít, hogy lehetőség legyen manuálisan is kommunikálni.

A következő komponens a gpsfake, ami a mozgás emulációját valósítja meg. Két részből áll: (1) az alap, GPS adatokat szolgáltató modul, (2) az általam fejlesztett menedzsment interfész. Az utóbbi TCP socket segítségével tartja a kapcsolatot a V2X alkalmazással, melyben a megadott parancsokat figyeli (5.4. alfejezet). Ezeket feldolgozza, majd továbbítja az alap szolgáltatásnak, ami ennek hatására útvonalat válthat, megállhat.

A szerver az a része a keretrendszernek, ami összekapcsolja a megjelenítést és az adatszolgáltató komponenseket. Websocketben aszinkron módon várja mind a böngészőtől, mind az eszközöktől érkező üzeneteket. Nincs maximalizálva a hozzá csatlakozó kliensek száma, csak a hardver teljesítménye szabhat felső határt. Ezáltal nagy mennyiségű OBU egymással történő interakcióját is megfigyelhetjük, akár több böngészőből is, mivel mindegyiken ugyan azt látjuk egyszerre. Az eszköztől érkező JSON objektumokat a szerver először feldolgozza, a koordináta, eszköz adatokat (`newCoordinate`) elmenti, az eseményekről kapott üzeneteket (`newMessage`) fájlba logolja.

A böngésző jeleníti meg Google térkép segítségével az adott eszköz helyzetét, fontosabb eseményekről információkat. Aszinkron figyeli a szervertől érkező üzeneteket, majd a típusának megfelelően megjeleníti azt. Minden böngésző, mikor felcsatlakozik a szerverhez, egy szinkronizációs (initBrowser) üzenetet küld, melynek hatására megkapja az összes eszköz helyzetét, tulajdonságait. Ennek hatására minden becsatlakozó böngésző egyből konzisztens állapotba kerül.

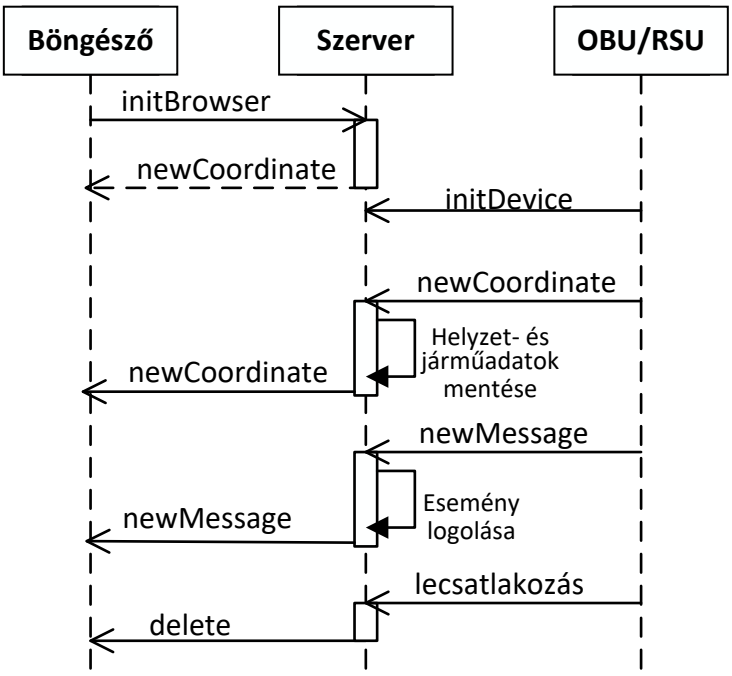
4.2. Vizualizáció

Mivel szöveges kimenet alapján sokszor nehéz megítélni, követni, hogy egy-egy helyzetben hogyan dönt egy V2X alkalmazás, mit csinál a jármű vagy egy adott C-ITS infrastruktúra elem, ezért nélkülözhetetlen volt egy olyan rendszer elkészítése, mely

meg tudja jeleníteni a járművek mozgását, az üzenetek áramlását, az egyes ITS állomások helyzetét. Mivel valós járműkommunikációs eszközök álltak rendelkezésemre, ezért szükséges volt, hogy létező koordinátákkal dolgozzak. A járművek a megfelelő pozícióban történő megjelenítését a böngészőben a térképen a Google Maps API segítségével valósítottam meg, melyet egy olyan szerver lát el adatokkal, melyhez az OBU-k/RSU-k hozzácsatlakoztak.

4.3. Üzenet típusok

A szerver mind a böngésző, mind a OBU/RSU eszközök között JSON üzenetben kommunikál. A JSON formátum tömör módon képes információt tárolni (ellentétben az XML-el), használata megkönnyíti az adatok feldolgozását, továbbá különböző eszközök között biztosít egységes kommunikációs lehetőséget. A különböző típusú kliensek és feladatok miatt négy típusú üzenetet hoztam létre, melynek a szekvenciája a 4.2. ábra-n látható.



4.2. ábra Üzenet szekvencia

Az üzenet fajtája a `type` kulcsban van megadva, ennek a beolvasása után dönti el az adott függvény, hogy hogyan fogja feldolgozni. Az első típust a böngésző küldi a szervernek, ez az `initBrowser`. Ennek hatására a szerver a sessiont elmenti a böngészők listájába, így egyszerűbbé válik az eszközöktől kapott koordinátákat kiküldeni a böngészőknek. Ezen felül válaszban a szerver visszaküldi az összes, még aktív kapcsolattal rendelkező jármű helyzetét, ezáltal minden új felcsatlakozott, vagy frissített böngésző azt az állapotot fogja látni, mint a többi, tehát konzisztens állapotba kerül egyből. Egy ilyen típusú JSON objektum a 4.3. ábra szerint néz ki.

```
{
    "type": "initBrowser"
}
```

4.3. ábra Böngésző csatlakozása

Mivel fontos külön kezelni a böngészőket és a járműveket a szükséges csoportos üzenetkezelés miatt, ezért nélkülözhetetlen volt az eszköz inicializálására (`initDevice` 4.4. ábra) is egy külön üzenetfajtát konstruálni. Itt a típusán kívül a JSON-nek tartalmaznia kell az OBU/RSU állomás azonosítóját (`id`) is, mely alapján azok egyedien megkülönböztethetőek.

```
{
    "type": "initDevice",
    "id": 43863
}
```

4.4. ábra OBU csatlakozása

A következő üzenettípus a koordináták küldésére használható, neve a `newCoordinate`. A kötelező típuson kívül rendelkezik egy `vehicles` tömbbel is, melybe az egyes OBU-k/RSU-k adatai kerülnek. A feldolgozás miatt minden esetben kötelező megadni az `id`-t, hogy lehessen tudni melyik eszköztől jött az adat. Az eszköz típusa (`type`) megadása nem kötelező, de ez alapján történik az ikon hozzárendelése a böngészőben. A dolgozat megírásakor a következő típusokhoz van ikon: mentő (ambulance), autó (car, 10 féle), RSU és jelzőlámpa (trafficlight) [57]. Amennyiben ezektől eltérő, nem létező típus érkezik az üzenetben, egy kérdőjel ikonnal jelenik meg a marker a térképen. A Google Térképen a marker egy kis ikon, melyet arra használnak, hogy jelezze a térképen a pozíciót. Ezeken kívül szükséges még megadni a szélességi (lat) és a hosszúsági (lng) adatokat is, melyek alapján a marker mozoghat. A 4.5. ábra egy olyan JSON objektumot szemléltet, melyben egy mentőnek és egy autónak a helyzete van leírva. Mivel a járművek adata tömbként van megadva, így egyszerűen biztosít lehetőséget arra, hogy nagyobb mennyiségű eszközadatot küldjünk el. Így lehetővé teszi olyan szimulációs rendszer elkészítését is, ahol az egyes járművekből észlelhető más eszközöket akarjuk megfigyelni.

```
{
    "type": "newCoordinate",
    "vehicles": [{
          "id": 16414,
          "type": "ambulance",
          "lat": 47.714182,
          "lng": 17.681969
    }, {
          "id": 33670,
          "type": "car",
          "lat": 47.714182,
          "lng": 17.681969
    }]
}
```

4.5. ábra Új koordináta

A különböző események jelzésére is egy külön üzenettípus lett létrehozva, melynek a típusa a `newMessage`. Ennek az az oka, hogy az OBU külön eseményben kap CAM/DENM/stb. üzenetet és koordináta frissülésről jelzést. Mivel nem akartam, hogy valamelyiknek is be kelljen várnia a másikat – nem is láttam ennek szükségességét –, így a külön üzenettípus választása mellett döntöttem. Emellett a feldolgozásban is jobban szét lehet választani a különböző funkciókat. A koordinátaküldéstől ez annyiban tér el, hogy a `vehicles` tömbben nincs helyadat, helyette egy `message` tömb van, amiben az esemény tulajdonságai vannak. Ebben az esemény részleteit kell megadni, amely a típusa (például: CAM, DENM, MAP, SPAT, IVI, stb.), a forrás és a cél OBU/RSU állomás azonosítója, az esemény leírása, továbbá az észlelés ideje. Az OBU/RSU típusának elküldése nem kötelező, de ha előbb lenne elküldve a `newMessage`, mint a `newCoordinate`, akkor ez szükséges, hogy meg lehessen jeleníteni a JSON objektum küldőjének az ikonját. Ennek elmulasztása esetén csak egy kérdőjel ikon jelenhet meg, a használható típusok megegyeznek a `newCoordinate`-nél bemutatottokkal. Mivel az esemény leírása itt is tömbben van megvalósítva, ez is egyszerűen biztosít lehetőséget arra, hogy nagy mennyiséget küldjünk egyszerre.

```
{
    "type": "newMessage",
    "vehicles": [{
          "id": 43863,
          "type": "car",
          "message": [{
                "type": "DENM",
                "src": 44863,
                "dst": 43863,
                "description": "Emergency warning",
                "time": "2016.10.15 21:07:05"
          }]
    }]
}
```

4.6. ábra Új esemény típusú üzenet

Az utolsó üzenet típus (`delete`) csak a szerver és a böngésző közötti kommunikációban használja a szerver, hogy értesítse, ha egy OBU megszakítja a kapcsolatot, lecsatlakozik. Az azonosító alapján ez lehetőséget ad a böngészőnek, hogy a térképről letörölje az adott eszközt, így nem maradnak nem használt eszközök a térképen.

```
{
    "type": "delete",
    "id": 43863
}
```

4.7. ábra Delete típusú JSON objektum

4.4. Szerver

A szerver WildFly 10.0-n, 1.8 verziójú Java felhasználásával lett elkészítve [58]. A böngésző és az OBU-k/RSU-k felől érkező aszinkron hívások kezelést websocket segítségével valósítottam meg. Ez egy független, TCP alapú protokoll, mely olyan előnyöket

biztosít, hogy a szerver a nélkül tud üzenetet küldeni egy böngészőnek, hogy az előzetesen kéréssel fordul volna a szerverhez. Ez abban segít ebben a környezetben, hogy az új koordináta érkezésekor azt a szerver egyből tovább küldheti, nem kell a böngészőnek ezt folyamatosan kérdezgetnie. Amint az OBU új helyzetadatot küld, azt eltárolás után egyből eljuttathatja a csatlakozott böngészőkhöz. Ezáltal nincs szükség a böngészőnek folyamatosan frissítési kérelemmel fordulni a szerverhez, nem kell fölösleges üzenetekkel terhelni a hálózatot. A szerveren külön mentésre kerül a böngésző és az eszköz munkamenete, így egyszerűen értesíthető az összes csatlakozott kliens csoport típustól függően. A szerver emellett eltárolja az egyes eszközök tulajdonságait, mint a típusa (autó, jelzőlámpa, RSU…), egyedi azonosítója és az utoljára kapott hosszúsági és szélességi adat, hogy egy később becsatlakozott, vagy egy frissített böngésző is egyből megkaphassa ugyanazokat az adatokat, így a böngésző egyből konzisztens állapotba kerül. Emellett ez azért is fontos, mert így azok az eszközök is meg fognak jelenni a térképen, amelyek már régebben küldtek magukról helyzet adatot, ez olyan eszköz esetén lehet fontos, mint a fix infrastruktúra (RSU, jelzőlámpa), ami nem változtatja a helyét.

4.4.1. Üzenetfogadás

Mikor a szerver aszinkron módon új üzenetet kap a böngészőktől vagy az eszközöktől, akkor a `MyMap` osztály `open(…)` függvénye kerül meghívásra. Ez a metódus megkapja az üzenet tartalmát és azt a sessiont, ami az üzenetet küldte. Ezután a JSON objektumból kiolvassa az üzenet típusát, mely alapján el tudja dönteni, mit tegyen az üzenettel, hogyan dolgozza fel. A kapott JSON átalakítására, készítésére a Javax JSON apit használom [59].

Amennyiben `initBrowser` típusú az üzenet, akkor a sessiont elmenti egy listába, hogy egyszerűbben lehessen üzenetet küldeni a böngészőknek. Válaszként készít egy olyan JSON objektumot, melyben a felcsatlakozott OBU-k adatait vannak, a legutolsó helyadattal (üzenet felépítése: 4.3. alfejezet).

Az `initDevice` üzenet célja, hogy elmentse az OBU-val létrehozott kapcsolatot egy `mapbe`, melyben a kulcs a session, a hozzá tartozó érték egy `Device` típusú objektum (tartalmazza az eszközök legfontosabb adatait, mint az azonosító, típus, hosszúsági és a szélességi adatok). Erre azért volt szükség, hogy amikor egy OBU megszünteti a kapcsolatát, lecsatlakozik, akkor csak a session áll rendelkezésre, mely alapján azonosítani lehet. Ennek segítségével kinyerhető az állomásazonosító mellyel a törlés már végrehajtható, értesítenie lehet a böngészőket.

Egy új koordinátát az OBU, egy `newCoordinate` típusú üzenetben tud küldeni. Az üzenetben található `vehicles` tömb elemei mentén az adatok feldolgozását, tartalmának elmentését a `storeVehicle(…)` függvény végzi. Miután kiolvasta az objektum értékeit, azaz az állomás azonosító (`id`), hosszúsági (`lng`) és szélességi(`lat`) adatot, megvizsgálja, hogy az adott azonosítóval rendelkező eszköz el lett e már mentve (ezt a `devices` map ellenőrzésével lehet megtenni). Ezt azért fontos ellenőrizni, ha esetleg egy `initDevice` üzenetet nem kapna meg a szerver, akkor is el tudja menteni az ada-

tokat, beregisztrálja az OBU-t/RSU-t. Amennyiben a munkamenet már szerepel a mapben, akkor a kapott adatokkal frissítve lesznek a változók. Mentés végeztével a JSON objektumot a szerver tovább küldi, a böngészőknek a `SendtoBrowser(…)` függvény segítségével. A metódus böngészők listájának minden tagjának elküldi az üzenetet, miután megbizonyosodott róla, hogy a kapcsolat még fennáll, nem szakadt meg.

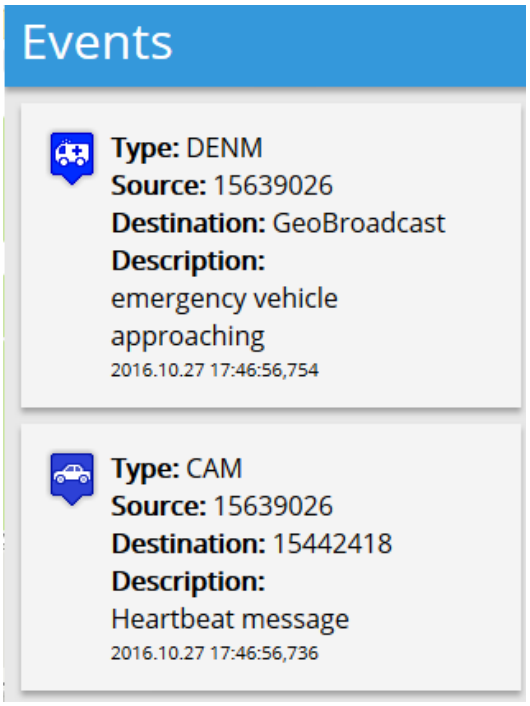
Az új események jelzését a `newMessage` típusú üzenet segítségével lehet küldeni. Ennek az üzenetnek a részleteit a szerver nem menti el magának, egyből tovább küldi a böngészőknek a `SendtoBrowser(…)` függvény segítségével. A koordinátával ellentétben erre azért nincs szükség, mivel egy később becsatlakozó böngészőnek, már nem biztos, hogy aktuális lenne az az információtartalom. A szimulációban ez egyes események utólagos ellenőrzése érdekében az üzenetek egy fájlba kerülnek logolásra. A fájlokat elérni a WildFly könyvtárában a log mappában lehet, amelyen belül a szerver elindításának ideje tartalmazza az egyes logokat. Minden eszköz saját fájlba dolgozik (hogy egyszerűbb legyen az utólagos feldolgozás), melynek a neve megegyezik az állomás azonosítójával. Az elérési út elkészítése az első OBU felcsatlakozásakor történik meg.

4.4.2. Kapcsolat lezárása

Mikor valamelyik kliens (OBU, böngésző) bontja a kapcsolatot a `close(session)` függvény hívódik meg. Böngésző esetén egyszerűen töröli a `browser` listából az adott sessiont. OBU esetén ennél többet kell tenni, mivel erről informálni kell a böngészőt is, hogy az el tudja távolítani a térképről a markert, törölhesse az adatokat az eszközről. Ahhoz, hogy értesítenie lehessen a klienst szükséges az állomásazonosító, melyet a `device` mapből ki lehet nyerni. Ezután a szerverről eltávolítható a hozzá tartozó adatok, és az 4.3 fejezetnek megfelelően egy `delete` üzenet készíthető, ami minden csatlakozott böngészőnek el is küld.

4.5. Webböngésző kliens

Járművek mozgását egy webböngésző segítségével lehet nyomon követni. Ehhez a megjelenítésében Google Map JavaScript API-t [60] használom fel. Az oldal funkcionális része is JavaScriptben lett megvalósítva, ami az app.js fájlban lett megírva. A weboldal váza a map.html állományban lett elkészítve. Itt történik a szükséges JavaScript könyvtárak meghívása és a weboldal vázának összeállítása. Két nagy részre bontható az oldal. Baloldalon található a Google térkép, melyen a csatlakozott eszközök egy marker segítségével megjelenhetnek, míg a jobb oldalra (4.8. ábra) olyan események kerülnek, melyeket ezek az OBU-k/RSU-k fogadnak, vagy küldenek. Ilyen lehet például egy DENM üzenet, melyet mondjuk egy mentő küldhet, hogy figyelmeztesse a többi járművet a megkülönböztető jelzés használatára, és hogy ezért adjanak neki elsőbbséget. Továbbiakban a weboldal működését, azaz szerverrel történő kommunikációt és a markerek megjelenítését, az eseménykezelést mutatom meg.



4.8. ábra Kommunikációkor keletkező események

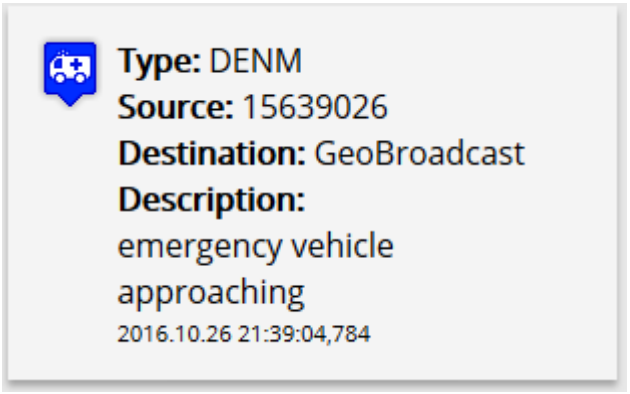
Mikor a weboldal betölt, az `initialize()` függvény fut le, melynek több feladata van. Egyrészt websocket segítségével csatlakozik a szerverhez, mint egy böngésző (4.3. ábra), létrehozza a térképet, majd a megfelelő helyre (`id = googleMap`) berakja az oldalra. Ennek a kezdeti középpontja előre meg van adva (`defaultlocation`), de amennyiben elérhető a helyadat a böngészőből, akkor azt adatot fogja használni, mint a térkép közepe. Emellett létrehoz egy eseménykezelőt is, mely figyeli a térképre kattintást. Ha egy ilyen esemény megtörténik, akkor az adott helyet állítja be középpontnak, megszünteti az esetleges járműkövetést.

A websocket létrehozása az `initWebSocket()` függvényben történik, ahol a megadott elérési út megadásával csatlakozik a szerverhez, beállítja, hogy a különböző események hatására, azaz kapcsolat kezdeményezése (`onopen`), üzenet érkezése (`onMessage`), hiba esetén (`onerror`) és a kapcsolat zárásakor (`onclose`) milyen függvények fussanak le. Üzenet küldést a `dosend()` függvény hajtja végre, mely a globális `message` értékét küldi el meghívása után, de mielőtt ezt megtenné, megvizsgálja, hogy van-e felépült kapcsolat, ha nincs `initWebSocket()` függvénnyel újra kapcsolódni próbál a szerverrel. Ellenkező esetben a szervernek elküldi az üzenetet, mely böngésző esetében is JSON formátumban van.

Új aszinkron üzenet érkezésekor azokat a már említett `onMessage(evt)` függvény fogja lekezelni. A paraméterben kapott változóból kiolvassa a kapott JSON üzenet értékét. Először az üzenet típusának meghatározása történik, ami három féle lehet `delete`, `newMessage` vagy `newCoordinate`. Ezeknek a részletes felépítése a 4.3. alfejezet tartalmazza.

A `delete` típus azt jelenti, hogy az adott OBU lecsatlakozott a szerverről, a böngésző törölheti a hozzá tartozó adatokat. Ilyen üzenet esetén a `vehicle` tömb elemei között megkeresi azt az azonosítót, ami az üzenetben található. Mikor megtalálta eltávolítja a hozzá tartozó markert, és a tömbből kiveszi azt az elemet.

Amikor `newMessage` típusú JSON objektum érkezik akkor, abban az OBU valamilyen küldött vagy fogadott CAM/DENM üzenet tartalmát osztja meg. Az objektum olvasásakor a függvény először megvizsgálja, hogy melyik eszközhöz tartozik az üzenet, ki küldte (JSON objektum részletes felépítése a 4.3 fejezetben található). Erre azért van szükség, hogy a megfelelő ikont lehessen hozzárendelni az üzenethez. Amennyiben nincs olyan azonosító, mint amivel az üzenetben szerepel, akkor létrehoz egy új markert és eltárolja az OBU adatait a `vehicle` tömbbe, de pozíciót nem ad, mivel ebben az üzenet típusban ilyen adat nincs, ezt a funkciót a `storeNewDevice(…)` függvény látja el. Mikor létrejön egy marker, akkor egy olyan esemény kezelő is készül hozzá, amely figyeli, ha rá kattintanak, utána azt adott markert állítja be a térkép közepének, mozgás esetén követi így a helyváltoztatást. Egy markerhez az ikon hozzárendelése a `getVehicleIcon(…)` segítségével történik. A függvény a paraméterben kapott típus alapján (car, rsu, trafficlight, ambulance) visszaadja a hozzá tartozó kép elérési útvonalát. Autó esetén mindig a soron következő képet választja ki a tíz darabból, amennyiben elfogy, újra az elejétől kezdve osztja ki. Amennyiben nincs olyan eszköz, mint ami meg van adva a típusban, akkor egy kérdőjel lesz megadva az ikonban. Az azonosító és az ikon megszerzése után, már létrehozható az új üzenetdoboz, melyért a `newEventMessage(…)` függvény felel.



4.9. ábra Üzenet boksz felépítése

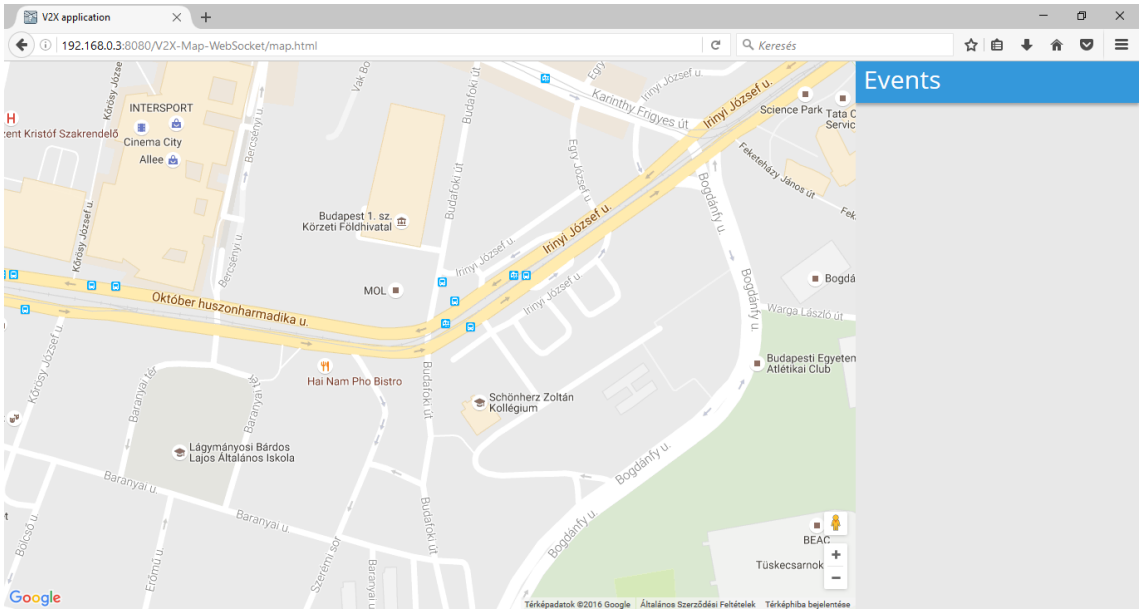
Az üzenetdoboz a weboldal jobb oldalán jelenik meg, az Events panel alatt. A legfrissebb kerül mindig a legtetejére Ahogyan a 4.9. ábra-n látható a doboz két részből áll. Bal oldalon található egy ikon, ami megegyezik a térképen megjelenő markerével. A jobb oldalon az üzenet van részletezve, melynek az előállításában a `buildMessage(…)` függvény is részt vesz. A paraméterben megkapott értékekből létrehoz egy HTML taget, mely az adott sort tartalmazza. Részletezésben először az üzenet típusa (CAM/DENM/stb.) szerepel, majd a forrás és a cél állomás azonosítója, majd végül egy rövid leírás az üzenet tartalmáról, a doboz alján az észlelés ideje látható. A `newEventMessage(…)` függvényben emellett egy buborékot (info window) is helyez a

markerre, mely jelzi, hogy tőle származik az üzenet és egy rövid leírásban ismerteti is azt.

Új koordináta érkezésekor (`newCoordinate`) amennyiben még nincs más eszköz a térképen, akkor megvizsgálja, hogy a térkép közepe és az eszköz milyen távol vannak egymástól [61]. Ha ez a mérték egy beállítottnál nagyobb, akkor az OBU aktuális helyére beállítja a térkép közepét. Ez akkor lehet hasznos, ha a helyadatok nem elérhetőek, nem tudja az alapján beállítani a térkép közepét, vagy egy teljesen másik helyről végzünk szimulációt. Ezután a JSON objektum `vehicles` tömb elemein iterál végig, hogy frissítse a markerek helyzetét, továbbá amennyiben egy markerre be van állítva a követés (`folllowID` egyenlő az eszköz azonosítójával) akkor a térkép közepe is átkerül a kapott jármű pozíciójára. Amennyiben az eszköz azonosítója nem szerepel a `vehicle` tömbben akkor a már említett `storeNewDevice(…)` függvény segítségével elmenti azt, létrehoz egy új markert.

4.6. Architektúra használata

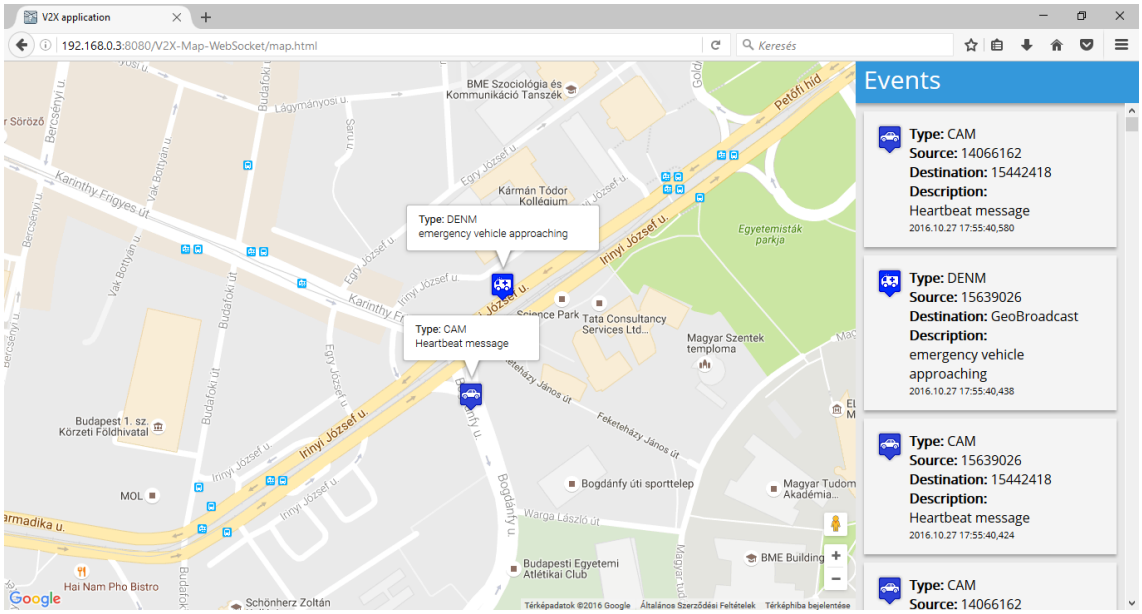
Miután a szerver elindult a böngészőn megnyitható a webszolgáltatás, melyet a `www.szerver_IP_címe:8080/V2X-Map-WebSocket/map.html` címen lehet elérni. Betöltése után a 4.10. ábra-n látható weboldal lesz elérhető, mivel amíg a járműkommunikációs alkalmazások nincsenek elindítva, a térkép és a jobb oldali esemény panelen nincs semmi.



4.10. ábra Megvalósított szimulációs rendszer a böngészőben

A böngésző beállításától függően egy kérést jelenhet meg betöltés közben, melyben azt kérdezi a böngésző, hogy hozzájárulunk-e ahhoz, hogy a helyzetünket felhasználhassa a weboldal. Ennek az a célja, hogy oda kerüljön a térkép közepe, ahol a felhasználó lehet. Mihelyst elindulnak a V2X alkalmazások, a markerek megjelennek a térképen, alkalmazási beállítástól függően üzenetek érkezhetnek az esemény panelre. Amennyiben a marker helyzete sokkal messzebb van, mint a térkép aktuális közepe, akkor módo-

sítja is azt, hogy egyből látható legyen az OBU pozíciója a térképen. A weboldal jelenlegi megvalósításában nincs lehetőség semmilyen interakcióra az eszközökkel, csak az adott szimulációnak az eredményét lehet megfigyelni. Paraméterezésre és módosításokra csak a V2X alkalmazásban van lehetőség. Tovább fejlesztési lehetőségként, egy menedzsment eszköz létrehozása nélkülözhetetlen lenne a weboldalon is, mellyel akár paraméterezhetővé lehetne tenni az alkalmazásokat, módosítani a szimuláció kimenetet, több aspektusból – akár az egyes járművek szemszögéből látott adatok alapján – lehessen megfigyelni a történéseket.



4.11. ábra A működő szimulációs rendszer

5. Mozgásemuláció

Dolgozatom egyik legfőbb célja volt egy olyan mozgásemulációs rendszer létrehozása, melynek a segítségével a V2X alkalmazások valóshoz közeli helyzetekben fejleszthetők, tesztelhetők és vizsgálhatók. A dinamikus útvonalmódosítás egy másik kielégítendő igény, mely azoknál az alkalmazásoknál merült fel, melyeknél valamilyen döntési helyzet után a mozgást meg kell változtatni. Ez lehet csak egy megállás, vagy egy teljesen más úti cél választása, azaz az eddigi út lecserélése egy másikra. Ebben a fejezetben bemutatom a gpsfake nevű eszközt és az ebben végzett változtatásaimat, melyek lehetővé teszik, hogy az igényeinknek megfelelő segédprogram jöhessen létre.

5.1. NMEA [62]

A Nemzeti Hajózási Elektronikai társaság (National Marine Electronics Association-NMEA) fejlesztette, hogy egységes interfészt biztosítson a különböző elektronikus hajózási eszközökhöz. A legtöbb mai program, ami GPS adatokkal dolgozik, ismeri és el is várja, hogy NMEA formátumba kapja az adatokat. Az elküldött adatok tartalmazzák a legfontosabb információkat: PVT (position, velocity, time- pozíció, sebesség és idő). NMEA ötlete, hogy az adatokat sorokba rendezi, melyeket mondatoknak nevez. Minden mondatnak van egy típusa, mely definiálja, hogy milyen típusú adatokat tartalmazhat. Ezeknek a típusoknak az első két betűje jelöli, hogy milyen eszköz használja azt a mondatot (a GPS vevők: GP (GPS reciver)). Az előtagot három további követ, mely megadja, hogy a mondat milyen típusú üzenetek közlésére használható. Az NMEA szabvány támogatja, hogy a gyártók saját mondatokat hozzanak létre. Ebben az esetben a mondat típusa ’P’ betűvel kezdődik, és ugyan úgy három betű követi.

Az NMEA mondatok egy ’$’ jellel kezdődnek, kocsi vissza/soremeléssel érnek véget és maximum nyolcvan karakter hosszúak lehetnek. Az egyes adatok vesszővel vannak egymástól elválasztva, melyek csak ASCII karakterek lehetnek. Minden mondat végét egy ellenőrző összeg zár a ’*’ jel után. Ez a két számjegy hexadecimális, az egyes karaktereken vett XOR művelettel számolja (’$’ és ’*’ jel közöttit mondaton).

5.1.1. Használt NMEA mondatok [62]

A mozgásemulációhoz kétfajta NMEA mondatot használok. Az első a GGA, melynek a felépítése a 5.1. ábra-n látható. A GGA az egyik legfontosabb NMEA mondat, mivel nélkülözhetetlen fix adatokat tartalmaz (3D-s helyzet).

```
$GPGGA,000100.00,4728.5334,N,01903.3143,E,1,,,103,M,,,,*0A
```

5.1. ábra GGA típus NMEA mondat

A típus utáni első adat az idő, amelyet ilyen formátumban lehet megadni: hhmmss.ss. Ezt a szélességi és a hosszúsági adat követ az égtájjal kiegészítve. Az első két számjegye a szögperc, melyet a szögmásodperc követ. Jelen példában ez 47 fok és 28.5334 szögperc, azaz 47,47556 fok. A helyzet adatok után azok minőségét lehet megadni, egy

a fix GPS-t jelöli. Ezt követi az észlelhető műholdak száma és a horizontális pozíció pontossága. A magassági értéket kétféleképpen lehet megadni, az első a tengerszint feletti, második a geoid magassága a WGS84-es szabvány szerint. Az ellenőrző összeg előtti többi mező már üres.

Az RMC az elvárt minimumot jelöli (Recommended minimum), mivel tartalmazza az alapvető PVT adatokat.

```
$GPRMC,000100.00,A,4728.5334,N,01903.3143,E,36.2,229,190516,,*1E
```

5.2. ábra RMC típusú NMEA mondat

Az 5.2. ábra-n látható, hogy az első öt adat megegyezik a GMC mondattal, azzal a kivétellel, hogy a második helyen a státusz található, 'A' azt mutatja, hogy rendben van az adat, 'V' figyelmeztetést jelentene. A helyzet adatokat a sebesség követi csomóban, melynek a váltószáma az 1,852, a példa esetben ez 67 km/h. Ezek után kiolvasható még az irány fokban és a dátum.

5.2. GPSD

A GPSD egy olyan szolgálatás démon, mely lehetőséget biztosít arra, hogy a csatlakoztatott GPS eszközöket monitorozzuk, az adatokat (helyzet, sebesség, irány, stb) TCP kapcsolaton keresztül elérhetővé tegyük. Ezzel a GPSD-vel megoldható, hogy több szoftver férjen hozzá ugyan ahhoz a szenzorhoz, anélkül, hogy versengés, adat vesztés történne. [63]

5.3. Az alap gpsfake

A gpsfake a GPSD rendszeren belül lehetőséget biztosít arra, hogy előre elkészített útvonalat játsszunk le, ezzel valódi GPS eszköz kimenetét, és így valódi mozgást szimulálva. Egy pseudo terminál (pseudo-TTY) kapcsolatot hoz létre a GPSD-vel, mely azt hiszi róla, hogy egy valós GPS eszköz. A log fájlból folyamatosan olvassa ki a GPS adatokat, majd küldi is tovább a GPSD-nek. A fájlban a következő formátumokban lehetnek: NMEA, SiRF, TSIP vagy Zodiac. A programok, hardver eszközök a GPSD-hez kapcsolódva tudnak GPS adatokat lekérni, számukra teljesen rejtve marad az információ forrása. [8]

A gpsfake jelenleg elérhető megvalósítása nem ad arra lehetőséget, hogy valós járműkommunikációs eszközre olyan alkalmazást készítsünk, melynek módosítania kellene a jármű dinamikáját. Tehát nincs lehetőség arra, hogy a járművet megállítsuk (azaz folyamatosan ugyan azt helyzetet adja vissza), igény szerint újra elindítsuk, a rögzített GPS útvonaltól eltérjünk, azt módosítsuk, stb. A publikus változatban csak azt lehet tenni, hogy az alkalmazást leállítjuk, az OBU-n átváltunk statikus GPS beállításra, melyben a megállás helyére mutat. Újra elinduláskor új példányt kell indítani a gpsfake programból, melyhez az útvonal fájl átrendezése szükséges, hogy ugyan arról a helyről lehessen folytatni a tesztet. Nyilvánvalóan látható, hogy ez a megoldás nagyon körülményes, emellett a leállítás- újraindítás, útvonal fájl rendezése időigényes, képes befo-

lyásolni a teszt időbeli kimenetét, nem használható. Hasonlóan más, dinamikát igénylő helyzetben, például útvonal újra tervezésnél ugyan így kellene eljárni.

Ezen okok miatt vált szükségessé a gpsfkake átírása, hogy ezeket az igényeket ki lehessen elégíteni, laborkörülmények között is lehetővé váljon a mozgásváltoztatást igénylő járműkommunikációs alkalmazásokat fejleszteni valós hardver/szoftver implementációkat használva.

5.4. A gpsfake módosítása

A 5.3. fejezetben vázolt hiányosságok miatt nélkülözhetetlen volt a gpsfake kibővítése, melyet a legfrissebb (2016.10) GPSD 3.16-os verzióban valósítottam meg. Három új funkciót készítettem el melyek a következők:

- Megállás és újra indulás támogatása
- NMEA mondatok távolról küldése
- Új útvonal fájl küldése

Az új funkciók megköveteltek egy menedzsment interfészt létrehozását is, mely lehetőséget ad arra, hogy az alkalmazást távolról elérve az új funkciókat el lehessen indítani.

A módosítás két állomány átírását igényelte, mely a gpsfake.py és a gps/fake.py. Az első, a program elindítását és új gpsfake példányosítását (fake.py) végzi. A fake.py feladata a GPS adatok folyamatos beolvasása és kiküldése a csatlakozott klienseknek.

A program indulását követően az argumentumlista feldolgozása történik. Mivel a menedzsmentinterfész a programmal párhuzamosan fut és egy portot figyel, így szükségessé vált, hogy ezt argumentum listában meg lehessen adni. Ezért létrehoztam egy új argumentumot a -M-et, mely után egy port számot kell megadni, melyen figyelni fog a menedzsment szál. Amennyiben a kapcsoló nélkül indítjuk el az alkalmazást, akkor „csak” az alap funkciót kapjuk meg, nem lehet távolról konfigurálni.

A menedzsment szál kezeléséért a `TCPThread` osztály felel, melyet a port számmal és egy `Queue`-val kell inicializálni. A Queue a Pythonban a szálak közötti kommunikációt, információ cserét teszi lehetővé. Az egyik szál (menedzsment) beteszi a parancsokat ebbe a várakozási sorba és FIFO elven a fő szál, amely a gpsfake-t kezeli, kiveszi és megvizsgálja, hogy mit kell vele tennie. Ez a módszer egy szál biztos kommunikációt tesz lehetővé.

A szál elindításához a run(…) függvényt kell meghívni, mely egy TCP socket elindítását végzi. A menedzsment interfész egyszerre csak egy kliensnek ad lehetőséget csatlakozni, mivel, ha többen csatlakozhatnának, akkor egymással ellentmondásosan adhatnának ki parancsokat. A szálban egy TCP socket szerver van megvalósítva, amely végtelen ciklusban figyeli, hogy csatlakozik-e új kliens. Ha jön egy új kapcsolat, akkor clientthread(…) függvényt hívja meg, mivel ez felelős a klienssel történő kommunikációért. Ez a metódus végtelen ciklusban figyeli a kapott portot és feldolgozza, ha jött parancs TCP socketen keresztül. Mind a három funkcióhoz három parancs tartozik, melyek az indítását és lezárását jelölik. A három parancspár a következő:

- **`file-begin:`** Ez a parancs jelzi, hogy utána NMEA mondatok fognak következni. Ezután a mondatok egy ideiglenes állományba kerülnek, melyet majd a gpsfake ki tud olvasni.
- **`file-end:`** A parancs után már nincs több NMEA mondat, lezárható a fájl. Ezt jelzem a gpsfake-nek is, a queue-ba egy file paranccsal, mely pluszban megkapja még a fájl elérési útját és a nevét is a feldolgozás céljából.
- **`manual-begin:`** Ez a parancs arra szolgál, hogy jelezze: mostantól TCP socketben NMEA mondatok fognak érkezni, amíg erről a másik fél nem jelez. Manuális módban az aktuális útvonal (ha van) leállításra kerül, az új útvonaladatok egyesével kerülnek a gpsfake-hez, amit egyesével ki is ad a feliratkozott klienseknek. Tehát ha elküldünk egy mondatot, akkor az bekerül a gpsfake útvonal tömbjébe, ilyenkor amíg nem zárjuk a manuális módot és nem küldenénk többet mondatot ugyanazt az utolsót ismételné, mint megálláskor. Az NMEA mondatok feldolgozását a `readNMEA(…)` függvény végzi. Minden mondatot a sor vége jel mentén felbont, berakja a feldolgozási sorba, átadva a gpsfake példánynak. Mivel egy fix méretű bufferből történik a kiolvasás, ezért figyeli, ha egy mondat félbe lett vágva. Ekkor azt elmenti a következő adag feldolgozásáig, majd mikor újabb üzenetet küld a távolról vezérlő, akkor beszúrja a kapott üzenet blokk elejére, hogy újra egész sor jöjjön létre.
- **`manual-end:`** Ez a parancs jelzi, hogy nem jön több NMEA mondat, azaz elindítható a gpsfake útvonal tömbjének a ciklikus olvasása.
- **`stop:`** Ennek az üzenetnek a hatására a gpsfake példány megállítja az új mondatok küldését, a parancs érkezésekor éppen aktuálisat fogja ismételgetni, mintegy állást szimulálva.
- **`go:`** Stop üzenet után újra elindulni az útvonal lejátszását a go parancs segítségével lehet. Hatására gpsfake példány újra folytatja az útvonal tömb ciklikus lejátszását.

Mikor a kliens bontja a kapcsolatot a menedzsment interfésszel, akkor a program kilép a clientthread(…) függvényből, újra figyeli a kliensek csatlakozási kérelmét.

Menedzsment interfész elindítása után elindulhat a gpsfake példány is, mely paraméterben megkapja a feldolgozási sor (`Queue`) egy példányát is. Inicializálás után a fake.py `TestSession` osztály `run()` metódusához kerül a futás, mely végtelen ciklusban dolgozik, amíg a daemont le nem állítják. Ennek a függvénynek a feladata, hogy ellássa a felcsatlakozott klienseket GPS adatokkal, illetve itt történik a várakozási sor figyelése is. Amennyiben érkezett új parancs vagy NMEA mondat manuális mód esetén, akkor a `processingcommand()` függvénynek a feladata azt feldolgozni. `Stop` illetve `go` parancs hatására a `stop` változót igaz- hamis értékét változtatja. Mikor a `run()` függvény egy eszköznek kiküldi a helyadatot, akkor ahhoz a FakeGPS osztály `feed(…)` metódusát használja, mely paraméterben megkapja, ha meg kell állítani az koordináták kiadását. Ez a függvény nyilvántartja, hogy melyik sort küldte ki korábban az útvonal tömbből, ezt az indexet növelve képes a kapott útvonalon végig iterálni.

Megállás esetén az index értékét nem növeli, így ugyanazt az NMEA mondatot fogja ismételni, így állás szimulálhat.

File parancs hatására készül egy új `TestSession` osztály, mely az útvonal tömb előállítását, ellenőrzését végzi. Miután ez a tömb elkészül, lecserélem az eredeti tömböt erre, beállítva az indexet nullára. Így anélkül lehet új útvonalat kérni, hogy a démont le kelljen állítani.

Harmadik mód, azaz a manuális esetén, a gpsfake az aktuális pozíciónál leáll, majd mikor megérkezik az első NMEA mondat a tömb tartalmát le is töröli, az újak már egy üres tömbbe kerülnek. Ilyenkor az index értéke mindig az utolsó elemre mutat és minden NMEA mondat hatására eggyel előre lép. Így megelőzhető, hogy a gpsfake egy fél útvonalon végig iteráljon, amíg nem ér végére a manuális mód. Ez lehetőséget biztosít arra, hogy a küldő igénye szerint változzanak a koordináták, haladjon a jármű. Manuális mód végével az útvonalon a lépegetés újraindul a kapott új útvonallal.

6. V2X alkalmazás

Az 4.1. alfejezetben bemutatott architektúrán látható (4.1. ábra), a V2X alkalmazás két nagy részből áll. Először is az adott funkciót megvalósító járműkommunikációs alkalmazásból, továbbá egy olyan interfészből, ami a webszerverrel websocketben fenntartja a kapcsolatot. Elküldi a megváltozott koordinátát és a kapott CAM/DENM/stb. üzenet tartalmát, jellemzőit.

Egy járműkommunikációs alkalmazás bemutatására, az elkészített keretrendszer használatára, funkcióinak bemutatására készítettem a vészhelyzeti járműre figyelmeztető alkalmazást (2.2.1.2. alfejezet). Két szerepkört mutatok meg benne: a mentőt, aki DENM üzenetben tudatja, hogy megkülönböztető jelzést használ, és egy járművet, aki figyeli a kapott üzeneteket és ha úgy ítéli, megáll a kereszteződésben, hogy elengedje a mentőt, így biztosítva szabad utat a megkülönböztető jelzést használó jármű számára.

6.1. Kommunikációs interfész

A létrehozott interfész célja egy olyan kommunikációs rendszer biztosítása, mely lehetővé teszi, hogy a V2X alkalmazás írójának ne kelljen a szerverrel történő kommunikációt, az üzenetek felépítését pontosan ismernie, továbbá az alkalmazáshoz ne kelljen minden alkalommal egy websocket kezelő osztályt létrehozni.

A kommunikációs felületet a V2X alkalmazással a `MapCommunicator` osztály biztosítja. Példányosításakor paraméterben meg kell adni a szerverhez csatlakozás szükséges adatait (cím, port és elérési út). A websocket kliens funkcióit (kapcsolat nyitás, -zárás, üzenet küldés, -fogadás) a MyClientEndpoint osztály végzi, melyet felhasználtam a megadott weboldalról [64]. Csatlakozás után az alkalmazás beregisztrálja az eseménykezelőit a Commsignia járműkommunikációs API-ba, hogy a változáskor azonnal tudjon üzenetet küldeni a szervernek. Ennek a nagy előnye, hogy nem kell a V2X alkalmazásokban már ezt megvalósítani, a háttérben ez megtörténik.

Az új koordinátát figyelő eseménykezelő beregisztrálás automatikusan megtörténik az osztály példányosításakor. Mikor az OBU-nak frissül a helyzete, akkor a `sendCoordinate(…)` függvény a paraméterben kapott értékek segítségével létrehozza a `newCoordinate` típusú JSON üzenetet, majd el is küldi azt a websocketen keresztül a szervernek. Ez a függvény publikus, az osztály példányosítása után szabadon elérhető, ha szükség lenne más koordináták küldésére is. Az elvárt paraméterek – mely alapján az üzenet elkészíthető – az eszköz azonosítója, szélességi és hosszúsági adatok és az eszköz típusa. Az utóbbi egy enumerációval lehet megadni, ez tartalmazza az összes elérhető típust (car, ambulance, trafficlight, RSU). Ezen kívül ezeket az adatokat egy tömbben is meg lehet adni, ha több koordinátát kellene elküldeni.

A CAM/DENM/stb. üzenetek figyelése külön eseménykezelőben történik, mindkettő esetén már az osztály példányosításakor meg kell adni, hogy szükséges-e ezeket az üzeneteket automatikusan elküldeni a szervernek. JSON objektum előállítását – melynek a

típusa `newMessage` – és elküldését a szervernek a `sendEventMessage(…)` függvény végzi, amely az előzőhöz hasonlóan publikus, hozzáférhető az osztály példányosítása után. Üzenet konstruálásához meg kell adni az eszköz azonosítóját és típusát, a kapott/küldött üzenet típusát, a küldő és a fogadó állomás azonosítóját és az elküldendő üzenet tartalmát, illetve itt is lehetőség van listában több üzenetet megadni.

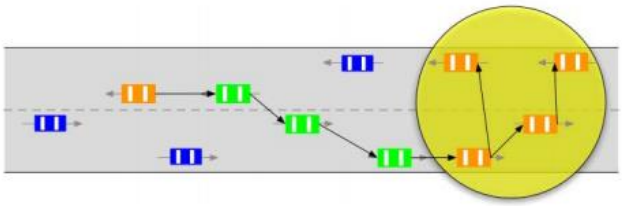
Természetesen az eszköz webszerverhez történő regisztrálását is megcsinálja automatikusan ez az interfész, ehhez az `initDevice(…)` függvény létrehozza az elvárt `initDevice` típusú JSON üzenetet, amit el is küld.

6.2. Vészhelyzeti járműre figyelmeztetés

Az elkészült jármű-kommunikációs keretrendszer demonstrálása céljából készítettem el a 2.2.1.2. fejezetben ismertetett, megkülönbözető jelzést használó járműre figyelmeztető V2X alkalmazást. Két szerepkört készítettem: egyrészt a mentőt, ami DENM üzenetben tudatja a környezetével, hogy szirénázik, míg a másik egy autó, ami veszi ezt az üzenetet, a saját helyzetével összehasonlítja a mentőautó pozícióját, irányvektorát, sebességég, és amennyiben úgy ítéli, megáll a kereszteződésben, elsőbbséget adva a mentőnek. A jármű-kommunikációs eszköz, amit használok egy Commsignia Kft. által gyártott OBU [65], melyek között a kommunikáció az IEEE 802.11 p protokoll segítségével történik. Az alkalmazás eredetileg a szakdolgozatomhoz készült, de megfelelő módosításokkal és kiegészítésekkel használhatóvá tettem a szimulációs rendszereben, hogy a létrehozott mozgásemulációs és -vizualizációs keretrendszer működését tesztelhessem.

6.2.1. Mentő

A mentő alkalmazás egy egyszerű szabályt követ, elindulás után elkezdi küldeni a DENM üzenetet, mely megfelel annak, hogy bekapcsolja a megkülönböztető jelzést. Ez az üzenet tartalmazza a helyzetét, sebességét, irányát és az üzenet tartalmát (Emergency Vehicle Approaching- Megkülönböztető jelzést használó jármű), azaz azt az eseményt, ami a DENM üzenet küldését kiváltotta. A DENM üzenetet GeoBroadcast segítségével kapják meg a környékbeli járművek (6.1. ábra). Ez azt jelenti, hogy az üzenet készítésekor az alkalmazás beállítja a cél pozíciót és hozzá egy átmérőt, így kijelölve a tájékoztatni kívánt/szükséges területet. A GeoNetwoking-nek köszönhetően ennek a megvalósítása nem okoz gondot a hálózatban. Addig lesz tovább küldve a csomag a cél pozíció irányába, amíg el nem éri azt a célterületet. Mikor ez megtörtént, broadcast formában eljuttatja minden eszköznek az üzenetet az adott zónában.



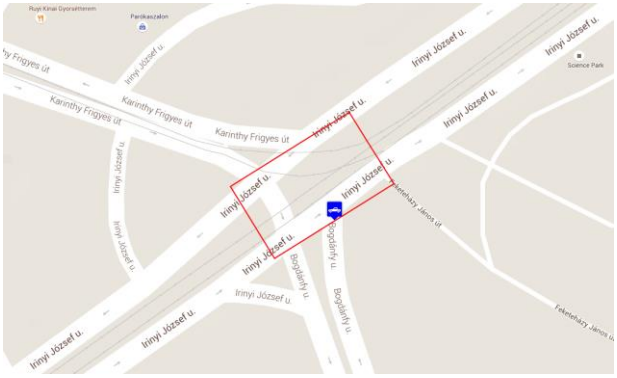
6.1. ábra GeoBroadcast [66]

Az alkalmazás indulását követően a minden pozíció változásakor frissíti a DENM üzenetet az új helyadataival, újabb változási ezt továbbítja. Ezen kívül példányosítja a 6.1. alfejezetben ismertetett kommunikációs interfészt is, melynek segítségével az új koordinátát automatikusan elküldi a szervernek, míg a DENM frissítésekor az interfész által biztosított függvény segítségével manuálisan teszi ugyan ezt, mivel az üzenet generálását az eseménykezelő nem tudja elkapni.

6.2.2. Jármű

Az alkalmazás hivatott bemutatni, hogy a járművek egymással történő kommunikációjának következtében észlelhető, ha a mentő más járművekkel kereszteznék egymást, így a normál járművek időben megállhatnak, elengedve a mentőt.

Az alkalmazás elindulását követően a kommunikációs interfészre beregisztrál a V2X alkalmazás, hogy a jármű koordinátája és a kapott DENM üzenet automatikusan el legyen küldve a szervernek. Ezután egy külön szálban csatlakozik a gpsfake menedzsment interfészéhez TCP socketen keresztül, hogy amikor megállás szükséges, akkor azt a gpsfake végre tudja hajtani. Továbbá két eseménykezelőt is létrehoztam a koordináta frissítésülésének és a DENM üzenet érkezésének a figyelésére. Az első eseménykezelő feladata, hogy amikor megváltozik a jármű helyzete, akkor azt elmenti, hogy a DENM-ben kapott értékekkel össze tudja hasonlítani.

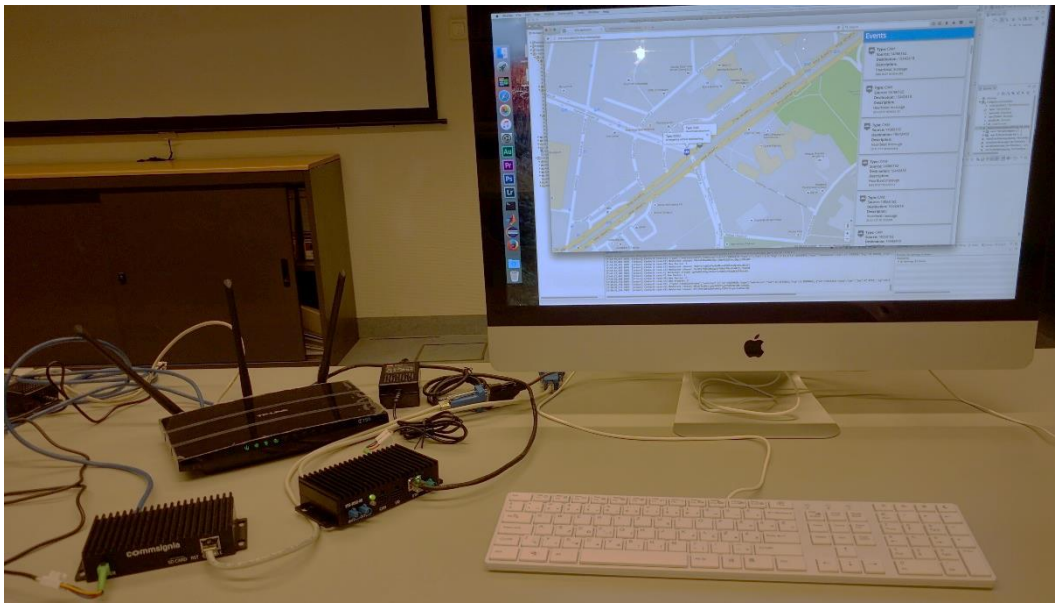


6.2. ábra Mentő figyelése a jármű elé vetített téglalapban

A DENM üzenetekre figyelő eseménykezelő felelős azért, hogy megállítsa a gpsfaket, ha úgy ítéli, hogy a mentő közel van és kereszteznék egymás útját. Ehhez egyszerű algoritmust használ (6.2. ábra): menet közben a jármű elé egy téglalapot vetít, és ha azt érzékeli, hogy ebben a téglalapban benne van a mentő, és közeledik, akkor `stop` üzenet küld a gpsfake management interfészének. A téglalapban tartózkodás vizsgálatát úgy oldottam meg, hogy négy pontot állítok elő a jármű elé és bármely kettő közé felirom az egyenes egyenletét. Ebbe behelyettesítem a mentő pozícióját, és aszerint, hogy az egyenes alatt, vagy felett van ez a pont, megállapítható, hogy a téglalapban van-e a mentő. Megállás után továbbra is figyeli a mentőtől kapott DENM üzeneteket a jármű. Amennyiben azt érzékeli, hogy a mentő távolodik, hagyja el a téglalapot, akkor újabb üzenetet küld a gpsfakenek, de ekkor már `go` üzenetet, hogy újra elindulhasson. Távolság számolását a földi koordináta rendszernek megfelelően az itt: [61] leírtaknak megfelelően történik.

6.3. Használt jármű-kommunikációs eszköz

A V2X alkalmazások szimulációjához a Commsignia Kft. által gyártott V2X OBU-kat használtam. Az eszköz támogatja az ETSI/IEEE/C2C-CC/ISO/SAE szoftver stackeeket. Egymás közötti kommunikációra az IEEE 802.11 p protokollt használják. Az V2X alkalmazás fejlesztéshez a Commsignia által biztosított Remote Java API-t használom, mely laboratóriumi körülmények között biztosítja az egyszerű fejlesztést. Azaz a V2X alkalmazás futtatásakor a számítógép az Ethernet porton adja a parancsokat a V2X stacknek. Az általam elkészített teszt környezet a 6.3. ábrarán látható. [65]



6.3. ábra A használt eszközök

7. Összefoglalás és továbbfejlesztési lehetőségek

A dolgozatomban egy rövid betekintést nyújtottam a járműhálózatok működésébe. Vázoltam legfontosabb előnyeit, melyek nélkülözhetetlenné teszik a jövőben a közlekedés biztonságban. Ennek céljából bemutattam néhány érdekesebb alkalmazási lehetőséget, ahol a jövőben ezt a technológiát használni lehet, hogy segítse a sofőrök munkáját, a jármű, közlekedés szereplőinek életet megmentse. A jármű-kommunikáció ismertetése után bemutattam néhány szimulációs eszközt is, melynek a segítségével laboratóriumi körülmények között lehet IVC protokollokat vizsgálni, fejleszteni tesztelni. A dolgozat második felében egy általam fejlesztett szimulációs keretrendszert ismertettem, amely valós jármű-kommunikációs hardver használatával biztosít lehetőséget különböző V2X alkalmazások fejlesztésére, teszteléséhez. A rendszer megvalósításához lehetővé kellett tennem, hogy a mozgás emuláció sokkal dinamikusabbá váljon, egy V2X alkalmazás igény szerint módosíthassa azt, amihez a gpsfake átírása nélkülözhetetlen volt. A mozgás követését, megjelenítését egy webböngészőből tettem lehetővé, melyekhez az adatokat egy szerver biztosítja, ezzel egy hidat kepézve OBU/RSU és a böngésző közé.

A szimulációs keretrendszer ezzel koránt sem készült el teljesen. Mindenképpen szeretném továbbfejleszteni a komponenseit. A böngészőben lehetőséget szeretnék biztosítani, hogy a V2X alkalmazást távolról lehessen konfigurálni, menedzselni, ezzel még több lehetőséget biztosítva az alkalmazások vizsgálatára. Emellett nélkülözhetetlennek érzem egy olyan forgalom szimulátor implementálását is a rendszerbe, mely a gpsfakkel együttműködve még több lehetőséget biztosítana a mozgás változtatására, nem kellene egy általunk létrehozott útvonalat mindig előre megadni. Távolabbi célok között szerepel az is, hogy a valós és a virtuális jármű-kommunikációs eszközöket egy keretrendszerbe építeni. A kettő közötti kommunikációjukat egy valós eszközön keresztül lehetne multiplexálni, ezzel komplexebb alkalmazásokat létrehozni, olyan eseteket vizsgálni, amikor a jármű-kommunikáció kezd elterjedni, de az utakon még sok a hagyományos jármű.

8. Köszönetnyilvánítás

A dolgozat elkészítéséhez szeretnék köszönetet mondani a Commsignia Kft.-nek a biztosított jármű-kommunikációs eszközökért (V2X OBU). Ezzel lehetőséget biztosítva, hogy a szimulációs rendszert valós hardver segítségével, valós V2X protokoll stack használatával tesztelhessem.

9. Ábrák jegyzéke

10. Rövidítésjegyzék

BTP	Basic Transport Protocol
C2C-CC	Car to Car Communication Consortium
CA	Cooperative Awareness
CAM	Cooperative Awareness Message
CALM	Communications Access for Land Mobiles
C-ITS	Cooperative Intelligent Transportation System
DCC	Decentralized Congestion Control
DEN	Decentralized Environmental Notification
DENM	Decentralized Environmental Notification Message
DSRC	Dedicated Short Range Communications
ESP	Electronic Stability Program
ETSI	European Telecommunications Standards Institute
HTML	HyperText Markup Language
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Organization for Standardization
ITS	Intelligent Transportation System
IVI	In-Vehicle Information Messages
IVC	Inter Vehicle Communication
JSON	JavaScript Object Notation
LDM	Local Dynamic Map
MANET	Mobile ad hoc network
MAP	Road Topology messages
NMEA	National Marine Electronics Association
OBU	On- Board Unit
OSI	Open System Interconnection
POI	Point of Interest
PVT	Position, Velocity, Time
RSU	Road Side Unit

SPaT	Signal Phase and Time messages
TCP	Transmission Control Protocol
TMC	Traffic Message Channel
UDP	User Datagram Protocol
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
V2X	Vehicular to everything
VANET	Vehicular ad hoc network
VEINS	Vehicles in Network Simulator
VSimRTI	V2X Simulation Runtime Infrastructure
WAVE	Wireless Access in Vehicle Environments

11. Hivatkozások

[1] K. Georgios, A. Onur, E. Eylem, H. Geert, J. Boangoat, L. Kenneth és W. Timothy, „Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions," *IEEE Communications Surveys & Tutorials,* pp. 584 - 616, 2011.

[2] A. Stevens és J. Hopkin, Benefits and deployment opportunities for vehicle/roadside cooperative ITS, Road Transport Information and Control (RTIC 2012), IET and ITS Conference on: IEEE, 2012.

[3] „Connected Vehicle," U.S Department of Transportation, [Online]. Available: http://its.dot.gov/cv_basics/index.htm. . [Hozzáférés dátuma: 20 09 2016].

[4] „Cooperative ITS Corridor," [Online]. Available: http://www.c-its-korridor.de/. [Hozzáférés dátuma: 20 09 2016].

[5] C. Sommer és F. Dressler, „Progressing Toward Realistic Mobility Models in VANET Simulations," *IEEE Communications Magazine,* pp. 132-137, November 2008.

[6] „VSimRTI - Smart Mobility Simulation," [Online]. Available: https://www.dcaiti.tu-berlin.de/research/simulation/.. [Hozzáférés dátuma: 20 09 2016].

[7] „Traffic and Network Simulation Environment," [Online]. Available: http://lca.epfl.ch/projects/trans. [Hozzáférés dátuma: 20 09 2016].

[8] E. S. Raymond , „gpsfake," [Online]. Available: http://catb.org/gpsd/gpsfake.html. [Hozzáférés dátuma: 20 10 2016].

[9] „Intelligent Transport Systems (ITS); Road Transport and Traffic Telematics (RTTT); Test specifications for Dedicated Short Range Communication (DSRC) transmission equipment; Part 1: DSRC data link layer: medium access and logical link control;". Szabadalom száma: ETSI TS 102 486-1-3 V1.2.2 , 05 2009.

[10] „Car 2 Car Communication Consortium," [Online]. Available: https://www.car-2-car.org.

[11] H. Moustafa és Y. Zhang, Vehicular Networks Techniques, Standards and Applications, Taylor & Francis Group, LLC, 2009.

[12] L. Chen és C. Englund, „Cooperative ITS - EU standards to accelerate," in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, 2014, pp. 681-686.

[13] „International Organization for Standardization," [Online]. Available: http://www.iso.org/.

[14] „European Telecommunications Standards Institute," [Online]. Available: http://www.etsi.org/.

[15] „Institute of Electrical and Electronics Engineers," [Online]. Available: https://www.ieee.org.

[16] „IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services". Szabadalom száma: IEEE Std 1609.3-2010, 2010 12 30.

[17] „Intelligent transport systems -- Communications access for land mobiles (CALM) -- Architecture". Szabadalom száma: ISO 21217:2014 , 04 2014.

[18] „Intelligent transport systems -- Communications access for land mobiles (CALM) -- M5". Szabadalom száma: ISO 21215:2010 , 11 2010.

[19] „Intelligent transport systems -- Communication access for land mobiles (CALM) -- Non-IP networking -- Part 1: Fast networking & transport layer protocol (FNTP)". Szabadalom száma: ISO 29281-1:2013 , 04 2013.

[20] CAR 2 CAR Communication Consortium Manifesto Overview of the C2C-CC System, CAR 2 CAR Communication Consortium , 2007.

[21] „SAFESPOT Integrated Project," [Online]. Available: http://www.safespot-eu.org/.

[22] „Intelligent Transport Systems (ITS); Decentralized Congestion Control Mechanisms for Intelligent Transport Systems operating in the 5 GHz range; Access layer part". Szabadalom száma: ETSI TS 102 687 V1.1.1, 07 2011.

[23] „Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality". Szabadalom száma: ETSI EN 302 636-4-1 V1.2.0, 10 2013.

[24] „Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 6: Internet Integration; Sub-part 1: Transmission of IPv6 Packets over GeoNetworking Protocols”. Szabadalom száma: ETSI EN 302 636-6-1 V1.2.0, 10 2013.

[25] „Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol”. Szabadalom száma: ETSI EN 302 636-5-1 V1.2.0 , 10 2013.

[26] „Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM)”. Szabadalom száma: ETSI EN 302 895 V1.0.0 , 01 2014.

[27] „Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service”. Szabadalom száma: ETSI EN 302 637-2 V1.3.1, 09 2014.

[28] „Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service”. Szabadalom száma: ETSI EN 302 637-3 V1.2.1, 09 2014.

[29] „Intelligent transport systems -- Cooperative ITS -- Using V2I and I2V communications for applications related to signalized intersections”. Szabadalom száma: ISO/PRF TS 19091 , 09 2016.

[30] „Intelligent transport systems -- Cooperative ITS -- Dictionary of in-vehicle information (IVI) data structures”. Szabadalom száma: ISO/TS 19321:2015 , 04 2015.

[31] „Connected Vehicle Reference Implementation Architecture,” [Online]. Available: http://www.iteris.com/cvria/html/applications/applications.html. [Hozzáférés dátuma: 26 10 2016].

[32] R. Popescu-Zeletin, I. Radusch és M. A. Rigani, Vehicular-2-X Communication: State-of-the-Art and Research in Mobile Vehicular Ad hoc Networks, 2010.

[33] „Annual Global Road Crash Statistics,” Association for Safe International Road Travel, [Online]. Available: http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics. [Hozzáférés dátuma: 27 09 2016].

[34] „TRAFFIC MESSAGE CHANNEL,” [Online]. Available: https://www.lta.gov.sg/content/ltaweb/en/industry-matters/traffic-info-service-providers/traffic-message-channel.html. [Hozzáférés dátuma: 26 10 2016].

[35] „Insanely Accurate Traffic Infommation," [Online]. Available: https://www.tomtom.com/en_hu/drive/tomtom-traffic/.

[36] „Waze," [Online]. Available: https://www.waze.com.

[37] F. J. Martinez, C. K. Toh, J.-C. Cano és P. Manzoni, „A survey and comparative study of simulators for vehicular," *Wireless Communications and Mobile Computing,* pp. 811-828, July 2011.

[38] C. Sommer, Z. Yao, R. German és F. Dressler, „On the Need for Bidirectional Coupling of Road Traffic Microsimulation and Network Simulation," in *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models*, ACM, 2008, pp. 41-48.

[39] „The Network Simulator - ns-2," [Online]. Available: http://www.isi.edu/nsnam/ns/.

[40] A. Varga, „The OMNeT++ Discrete Event Simulation System," in *Proceedings of European Simulation Multiconference* , Prague, Czech Republic, 2001.

[41] „JSim," [Online]. Available: http://www.physiome.org/jsim/.

[42] „Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator," [Online]. Available: http://jist.ece.cornell.edu/.

[43] „OPNET Technologies – Network Simulator | Riverbed," [Online]. Available: www.opnet.com/.

[44] „INET Framework," [Online]. Available: https://inet.omnetpp.org/.

[45] H. Haj-Salem, J. Chrisoulakis, M. Papageorgiou, . N. Elloumi és P. Papadakos, „The use of METACOR tool for integrated urban and interurban traffic control. Evaluation in corridor peripherique, Paris," in *Velucle Navlgaaon & Infomatton Systems Conference Proedmp* , IEEE, 1994, pp. 645-650.

[46] N. B. Taylor, „The CONTRAM dynamic traffic assignment model," in *Networks and Spatial Economics,* 2003, pp. 297-322.

[47] A. Schadschneider és M. Schreckenberg, „Cellular automation models and traffic flow," in *Journal of Physics A: Mathematical and General,* 1993.

[48] S. Krauß, Microscopic modeling of traffic flow: Investigation of collision Free Vehicle Dynamics, 1998.

[49] „Longitudinal Traffic model: The IDM,” [Online]. Available: http://traffic-simulation.de/IDM.html. [Hozzáférés dátuma: 26 10 2016].

[50] „The Lane-change Model MOBIL,” [Online]. Available: http://traffic-simulation.de/MOBIL.html. [Hozzáférés dátuma: 26 10 2016].

[51] M. Behrisch, L. Bieker, J. Erdmann és D. Krajzewicz, „SUMO - Simulation of Urban MObility: An Overview,” in *Proceedings of 3rd International Conference on Advances in System Simulation (SIMUL 2011)*, Barcelona, Spain, 2011.

[52] R. Riebl, H.-J. Günther, C. Facchi és L. Wolf, „Artery - Extending Veins for VANET Applications,” in *4th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS 2015)*, Budapest, Hungary, IEEE, 2015, pp. 450-456.

[53] B. Schünemann, „V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems,” in *Computer Networks*, New York, NY, USA, Elsevier North-Holland, Inc., 2011, pp. 3189-3198.

[54] B. W. David, „TCP Sockets,” [Online]. Available: http://users.cis.fiu.edu/~czhang/teaching/cen4500/project/TCP_Sockets.htm. [Hozzáférés dátuma: 20 10 2016].

[55] I. Fette, Google, Inc., A. Melnikov és Isode Ltd., „The WebSocket Protocol”. Szabadalom száma: IETF RFC 6455, 12 2011.

[56] The JSON Data Interchange Format, Ecma International , 2013.

[57] „Transportation map markers,” [Online]. Available: https://mapicons.mapsmarker.com/category/markers/transportation/. [Hozzáférés dátuma: 10 10 2016].

[58] „WildFly,” Red Hat, Inc. , [Online]. Available: http://wildfly.org/.

[59] „Package javax.json,” [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/json/package-summary.html.

[60] „Maps JavaScript API,” [Online]. Available: https://developers.google.com/maps/documentation/javascript/. [Hozzáférés dátuma: 9 10 2016].

[61] M. T. Scripts, „Calculate distance, bearing and more between Latitude/Longitude points,” [Online]. Available: http://www.movable-type.co.uk/scripts/latlong.html. [Hozzáférés dátuma: 22 10 2016].

[62] D. DePriest, „NMEA data,” [Online]. Available: http://www.gpsinformation.org/dale/nmea.htm. [Hozzáférés dátuma: 10 10 2016].

[63] „gpsd — a GPS service daemon,” [Online]. Available: http://catb.org/gpsd/.

[64] . J. Sasidharan, „JSR 356 - Java API for Websocket (Java Client),” [Online]. Available: http://www.programmingforliving.com/2013/08/jsr-356-java-api-for-websocket-client-api.html. [Hozzáférés dátuma: 15 10 2016].

[65] „V2X Hardware,” [Online]. Available: http://www.commsignia.com/hardware/.

[66] „Intelligent Transport System (ITS); Vehicular Communication; GeoNetworking; Part1: Requrements”. Szabadalom száma: Draft ETSI EN 302 636-1 V1.2.0, 08 2013.

12. Függelék

A módosított gpsfake, a teljes GPSD könyvtárral:

https://github.com/szzso/GPSD-with-management.git

A szerver és a weboldal:

https://github.com/szzso/V2X-simulation-tool.git