**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**

FAKULTA PRÍRODNÝCH VIED

V2V komunikácia použitím minipočítača Raspberry Pi

DIPLOMOVÁ práca

**Nitra 2020 BC. DÁVID BOJNANSKÝ**

**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**

FAKULTA PRÍRODNÝCH VIED

V2V KOMUNIKÁCIA POUŽITÍM MINIPOČÍTAČA RASPBERRY PI

DIPLOMOVÁ práca

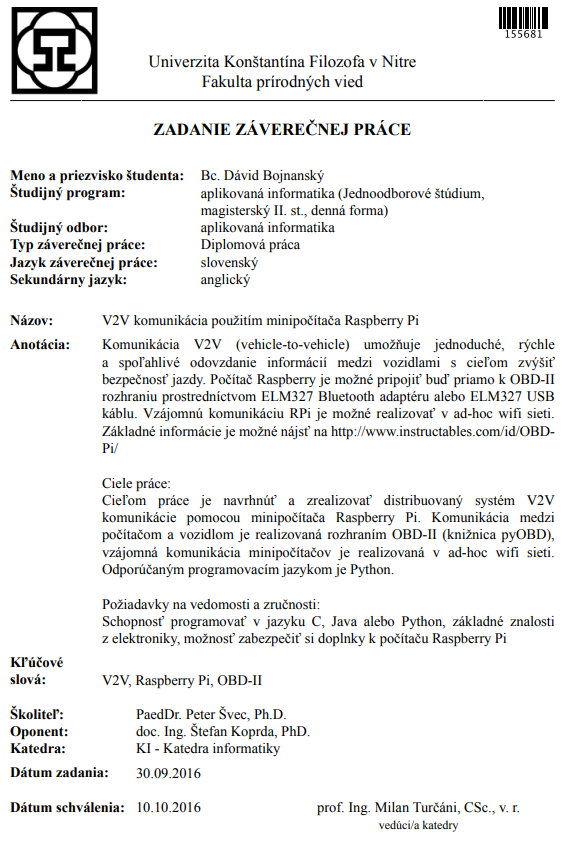
Študijný odbor: 9.2.9 Aplikovaná informatika

Študijný program: Aplikovaná informatika

Školiace pracovisko: Katedra informatiky

Školiteľ: PaedDr. Peter Švec, PhD.

Nitra 2020 Bc. Dávid Bojnanský



# Poďakovanie

Touto cestou vyslovujem poďakovanie pánovi PaedDr. Petrovi Švecovi, PhD. za odporúčania, rady a pripomienky počas písania tejto práce, ako aj nasmerovania a uvedenia do problematiky na začiatku.

# Abstrakt

BOJNANSKÝ, Dávid: V2V komunikácia použitím minipočítača Raspberry Pi. [Diplomová práca]. Univerzita Konštantína Filozofa v Nitre. Fakulta prírodných vied. Školiteľ: PaedDr. Peter Švec, PhD. Stupeň odbornej kvalifikácie: Magister odboru Aplikovaná informatika. Nitra: FPV, 2020. 81 s.

Diplomová práca sa venuje problematike komunikácie (t. j. výmene informácií) medzi viacerými vozidlami (tzv. V2V systém), vytvoreniu takéhoto systému pomocou minipočítača Raspberry Pi, programovacieho jazyka Python a rozhrania OBD-II, otestovaniu vytvoreného systému v praxi, a vyhodnoteniu dosiahnutých výsledkov. Diplomová práca je rozdelená na teoretickú a praktickú časť. Teoretická časť je zameraná na analýzu súčasného stavu a je rozdelená na 4 ďalšie časti. Prvá časť sa zaoberá V2V systémom v jeho rannom štádiu, kdežto druhá časť sa zaoberá V2V systémom v súčasnosti – v nej autor porovnáva minulosť a súčanosť, rozoberá dostupné bezdrôtové technológie vhodné pre V2V systém, rozoberá bezpečnosť tohto systému a prekážky brániace zavedeniu takéhoto systému do praxe. Tretia časť poskytuje stručnú charakteristiku minipočítačov Raspberry Pi a štvrtá časť zase rozhrania OBD-II. Praktická časť je rozdelená na 3 ďalšie časti. Prvá časť je zameraná na vytvorenie návrhu V2V systému po hardvérovej a softvérovej stránke. Druhá časť sa zaoberá realizáciou tohto návrhu krok za krokom. Tretia časť opisuje testovanie v praxi a vyhodnocuje dosiahnuté výsledky. V závere práce autor navrhuje možné vylepšenia vytvoreného V2V systému.

Kľúčové slová: V2V. Raspberry Pi. OBD-II.

# Abstract

BOJNANSKÝ, Dávid: V2V communication using minicomputer Raspberry Pi. [Diploma Thesis]. Constantine the Philosopher University in Nitra. Faculty of Natural Sciences. Supervisor: PaedDr. Peter Švec, PhD. Degree of Qualification: Magister of Applied Informatics. Nitra: FNS, 2020. 81 p.

Diploma thesis deals with Lua programming language and creation of sample applications. Bachelor thesis is divided into theoretical and practical part. The theoretical part is focused on analyzing of the current situation and is divided into 3 parts. The first part deals with programming paradigms used by Lua programming language. The second part deals with the characteristics of the programming language Lua. The third part provides a practical insight into the Lua programming language. The practical part is focused on analysis of the characteristics, concepts and procedures specific for the programming in the programming language Lua. This analysis is complemented by the sample applications that demonstrates the Lua programming language in practice. In the sample applications the focus is mainly on their complexity and simplicity. Bachelor thesis also deals with the embedding into the host application, which we demonstrate on the PHP programming language and trading platform *FXCM Trading Station*.

Keywords: V2V. Raspberry Pi. OBD-II.

# Obsah

[UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE 1](#_Toc33983436)

[UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE 2](#_Toc33983437)

[Poďakovanie 4](#_Toc33983438)

[Abstrakt 5](#_Toc33983439)

[Abstract 6](#_Toc33983440)

[Obsah 7](#_Toc33983441)

[Zoznam ilustrácií 9](#_Toc33983442)

[Zoznam zdrojových kódov 10](#_Toc33983443)

[Úvod 11](#_Toc33983444)

[1 Analýza súčasného stavu V2V systému 12](#_Toc33983445)

[1.1 Prvé náznaky už v roku 2007 12](#_Toc33983446)

[1.1.1 Skúšanie v praxi 13](#_Toc33983447)

[1.1.2 Uvedenie do praxe 14](#_Toc33983448)

[1.2 V2V systém v súčasnosti 15](#_Toc33983449)

[1.2.1 Porovnanie rozdielov minulosť/súčanosť 16](#_Toc33983450)

[1.2.2 Bezdrôtové technológie 18](#_Toc33983451)

[1.2.3 Bezpečnosť V2V systému 23](#_Toc33983452)

[1.2.4 Prekážky brániace zavedeniu V2V systému do praxe 24](#_Toc33983453)

[1.3 Minipočítač Raspberry Pi 25](#_Toc33983454)

[1.3.1 Raspberry Pi 3 25](#_Toc33983455)

[1.3.2 Raspberry Pi 4 26](#_Toc33983456)

[1.4 OBD (II) 26](#_Toc33983457)

[2 Ciele DIPLOMOVEJ práce 30](#_Toc33983458)

[3 Návrh, realizácia a testovanie V2V systému 31](#_Toc33983459)

[3.1 Návrh 31](#_Toc33983460)

[3.2 Realizácia 36](#_Toc33983461)

[3.2.1 Inštalácia operačného systému Raspbian 36](#_Toc33983462)

[3.2.2 Inštalácia softvérového balíčku BlueZ 37](#_Toc33983463)

[3.2.3 Informácie o našom vozidle (súbor vehicleinfo.txt) 37](#_Toc33983464)

[3.2.4 Párovanie OBD-II skenera s minipočítačom (súbor obd2rpi/pairwithobd) 38](#_Toc33983465)

[3.2.5 Inštalácia a konfigurácia BATMAN-ADV 40](#_Toc33983466)

[3.2.7 Príprava vývojového prostredia 42](#_Toc33983467)

[3.2.8 Inštalácia programovacieho jazyka Python a balíčkov 42](#_Toc33983468)

[3.2.9 Klientská aplikácia (priečinok vehicleclient) 42](#_Toc33983469)

[3.2.10 Serverová aplikácia (priečinok vehicleserver) 54](#_Toc33983470)

[3.3 Praktické testovanie a zhodnotenie výsledkov 72](#_Toc33983471)

[Záver 78](#_Toc33983472)

[Zoznam bibliografických odkazov 79](#_Toc33983473)

[Zoznam príloh 82](#_Toc33983474)

# Zoznam ilustrácií

Obrázok 1 Blížiace sa záchranné vozidlo signalizované vo V2V systéme 14

Obrázok 2 Čiastočne a plne prepojená Mesh sieťová topológia 18

Obrázok 3 Anglický názov Bluetooth pri doslovnom preklade predstavuje modrý zub 19

Obrázok 4 Frekvenčné pásma kanálov štandardu IEEE 802.11p 22

Obrázok 5 Minipočítač Raspberry Pi 4 25

Obrázok 6 Schéma OBD-II konektora a rozloženie PIN-ov podľa protokolov 29

Obrázok 7 Komponenty nášho V2V systému 32

Obrázok 8 Návrh riešenia z pohľadu softvérovej stránky 34

Obrázok 9 Grafické rozhranie aplikácie servera 58

Obrázok 10 Zobrazované informácie a ich funkčnosť 73

Obrázok 11 Testovanie dosahu signálu 74

Obrázok 12 Testovanie nadviazania spojenia míňajúcich sa vozidiel 75

Obrázok 13 Celková rýchlosť míňajúcich sa vozidiel je 80 km/h 75

# Zoznam zdrojových kódov

Zdrojový kód 1 Súbor vehicleinfo.txt 36

Zdrojový kód 2 Súbor obd2rpi/pairwithobd 38

Zdrojový kód 3 Súbor /etc-network/interfaces 39

Zdrojový kód 4 Súbor /etc-network-interfaces.d/wlan0 40

Zdrojový kód 5 Súbor /etc-network-interfaces.d/bat0 40

Zdrojový kód 6 Súbor batman/batman 40

Zdrojový kód 7 Súbor vehicleclient/mycommands.py 43

Zdrojový kód 8 Súbor vehicleclient/myvehicle.py 47

Zdrojový kód 9 Súbor vehicleclient/obdmonitor.py 49

Zdrojový kód 10 Súbor vehicleclient/vehicleclient.py 52

Zdrojový kód 11 Súbor vehicleserver/vehicle.py 55

Zdrojový kód 12 Súbor vehicleserver/foreignvehicles.py 56

Zdrojový kód 13 Súbor vehicleserver/objectmaker.py 57

Zdrojový kód 14 Súbor vehicleserver/gui.py 60

Zdrojový kód 15 vehicleserver/vehicleserver.py 64

Zdrojový kód 16 Súbor lib-systemd-system/v2v-vehicleclient.service 66

Zdrojový kód 17 Súbor lib-systemd-system/v2v-vehicleclient.py 67

Zdrojový kód 18 Súbor lib-systemd-system/v2v-vehicleserver.service 68

Zdrojový kód 19 Súbor install 70

# Úvod

V dnešnej dobe začína byť pre ľudí hlavnou prioritou ich bezpečnosť a začali si to uvedomovať aj výrobcovia vozidiel. Vozidlá disponujú airbagmi, systémom ABS, ktoré kontroluje prudké brzdenie na povrchoch s rôznou priľnavosťou, rôznymi asistenčnými systémami, ktoré vedia pomocou sady senzorov predvídať rôzne situácie a zlepšujú tak zážitok z zjazdy, a pododobne. Výrobcovia vozidiel chcú ísť v dohľadnej budúcnosti ešte ďalej. Chcú umožňiť vozidlám, aby sa medzi sebou „rozprávali.“ Nemyslíme tým, že teraz každé vozidlo bude mať v jeho prednej časti ústa a bude vydávať zvuky. Myslíme tým, neviditeľnú a nepočuteľnú bezdrôtovú komunikáciu. Komunikácia v tomto prípade predstavuje výmenu užitočných informácií ako napríklad smer jazdy vozidla, rýchlosť vozidla, či vozidlo brzdí a podobne. Takáto komunikácia sa začala označovať ako systém V2V. Je to naozaj skvelá myšlienka, už len pri predstave, že sa neskutočne zredukuje množstvo dopravných nehôd a zachráni sa tak množstvo nie len ľudských životov.

My sme si vybrali tému diplomovej práce, ktorá s takouto komunikáciou medzi vozidlami súvisí. Jej hlavný cieľ je vytvoriť funkčný V2V systém, otestovať ho v praxi a vyhodnotiť dosiahnuté výsledky. Diplomová práca je rozdelená do 3 kapitol.

V prvej kapitole sa zaoberáme analýzou súčasného stavu V2V systému, kde sme sa snažili pozrieť aj do minulosti, porovnať minulosť a súčasnosť, rozobrať dostupné bezdrôtové technológie vhodné pre V2V systém, rozobrať bezpečnosť tohto systému a prekážky brániace zavedeniu takéhoto systému do praxe. Ďalej sme sa pozreli aj na stručnú charakteristiku minipočítačov Raspberry Pi a rozhrania OBD-II, ktoré sú neoddeliteľnou súčasťou nami vytvoreného V2V systému.

V druhej kapitole definujeme ciele, ktoré realizujeme v rámci tretej kapitoly.

V tretej kapitole sa zaoberáme praktickou časťou, kde sme sa snažili vytvoriť návrh nášho V2V systému po hardvérovej a softvérovej stránke, zrealizovať tento návrh krok za krokom, otestovať ho v praxi a nakoniec vyhodnotiť dosiahnuté výsledky.

# 1 Analýza súčasného stavu V2V systému

Prvé náznaky a ambície na realizáciu V2V systému boli už okolo roku 2007. Označenie V2V predstavuje skratku pre anglický výraz Vehicle-to-Vehicle, teda po našom Vozidlo-Vozidlo. Keď k tejto skratke pridáme slovo systém, tak spolu to predstavuje najmä systém, v ktorom komunikujú viaceré vozidlá medzi sebou.

V rámci tejto kapitoly (1) sme v prvej podkapitole rozobrali pohľad na V2V systém v jeho začiatkoch – čo viedlo k jeho vytvoreniu, aká bola predstava jeho fungovania, z akých technológií mal byť poskladaný. Pozreli sme sa aj na skúšku vtedajšieho prototypu v praxi (podkapitola 1.1.1). V závere sme uviedli vtedajší plán očakávaného zavedenia tohto systému do praxe (podkapitola 1.1.2). V druhej podkapitole sa venujeme jeho súčasnému stavu. V tretej podkapitole spomíname minipočítač Raspberry Pi, na ktorom je založená naša diplomová práca. V štvrtej podkapitole sa zaoberáme systémom OBD, ktorý je taktiež dôležitou súčasťou našej diplomovej práce.

## 1.1 Prvé náznaky už v roku 2007

Podľa článku „Komunikácia medzi autami - vyskúšali sme technológiu budúcnosti“ publikovaného v roku 2007 autorom (Karpat, 2007), dôvodom na vôbec začatie rozmýšľania o komunikácii medzi vozidlami bola nemecká štatistika o nehodovosti na cestách z roku 2003. Tá hovorila jasnými číslami, a teda že nehodovosť má rastúcu tendenciu. Autor v článku uviedol: „26,1 % havárií bolo spôsobených jazdou neprimeranou rýchlosťou a nedodržiavaním bezpečných odstupov medzi vozidlami, 22,8 % príčin tvorili prekážky na ceste, v zákrute, kľučkovania medzi autami a otáčania, a 11,1 % pripadlo na jazdu v zlom pruhu a nesprávne predbiehanie.“ Autor článku ďalej uviedol, že jednou z možností ako znížiť počet nehôd a úmrtí na cestách je nehodám predchádzať.

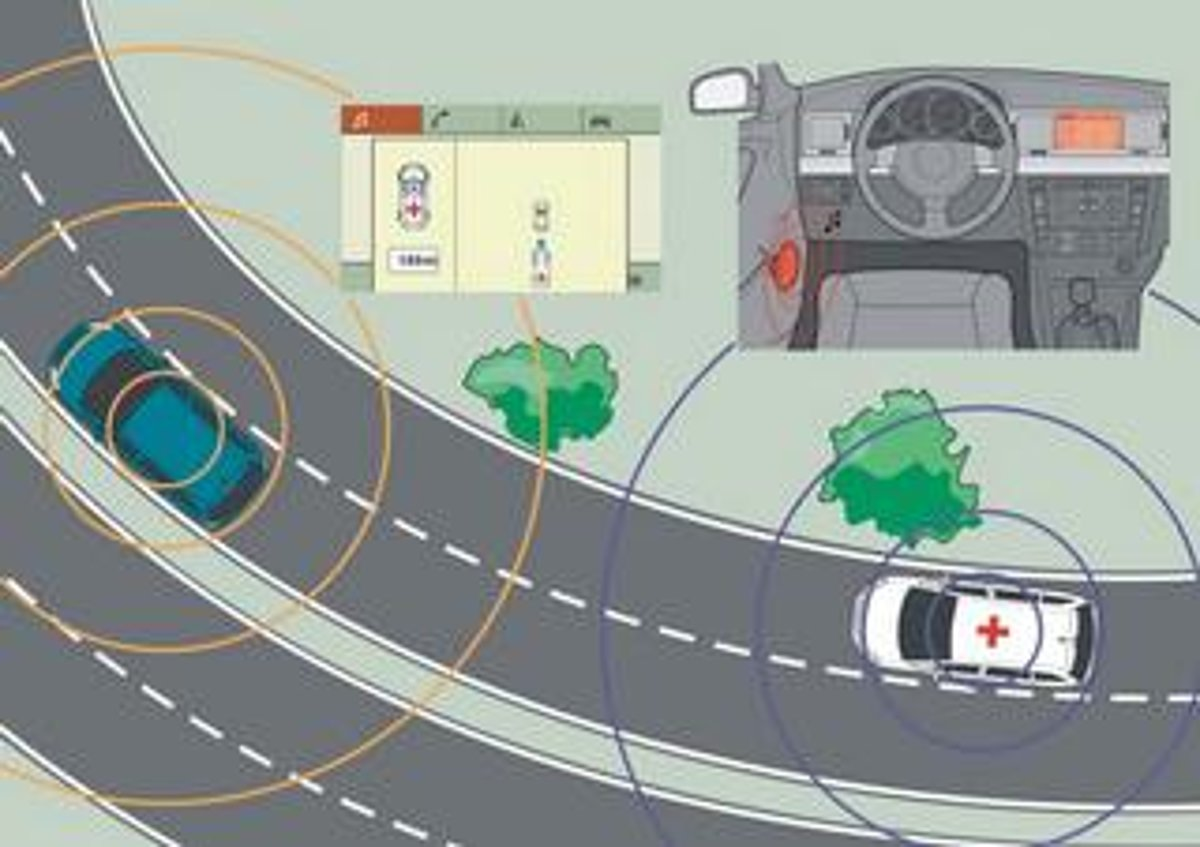
Ďalšou možnosťou by bolo zníženie počtu vozidiel. Keď sa však zamyslíme nad dnešnou dobou, tak klesanie, dokonca ani stagnácia tohto počtu sa nejaví ako reálna. V dnešnej dobe nie je nič nezvyčajné, ak napríklad 4 členná rodina vlastní aj 5 áut. Z toho nám vyplýva, že počet vozidiel na cestách bude stále len narastať. S tým však súvisia aj ďalšie problémy ako napríklad nadmerná produkcia CO2 alebo kolaps dopravy v každom väčšom meste. To je však nad rámec tejto témy.

V roku 2007 predstavila automobilka General Motors telematický systém prenosu informácií medzi vozidlami. V tých časoch bolo cieľom systému V2V včas upozorniť vodiča vozidla na nepredvídané nebezpečenstvo a v predstihu sa naň správne pripraviť a zareagovať. Medzi takéto nebezpečenstvá sa zaraďovalo riskantné konanie vodičov (odstavenie vozidla v zákrute, otáčanie sa za zákrutou), autá zoraďujúce sa za sebou pri spájaní viacerých pruhov do jedného, práce na ceste alebo zlé poveternostné podmienky (námraza, sneh, hmla). Sústava systému V2V mala pozostávať z mikroprocesora, GPS prijímača a modulov realizujúcich bezdrôtovú komunikáciu. Táto sústava mala dokázať sledovať polohu vozidla, jeho rýchlosť a smer. Tieto údaje si potom mala vymieňať s okoloidúcimi vozidlami do vzdialenosti 2 až 5 kilometrov. Ak by systém zistil neštandardnú dopravnú situáciu, mal odoslať informácie cez bezdrôtovú WiFi sieť. Ostatné vozidlá v dosahu mali tieto informácie prijať a akusticky (cez reproduktory v aute) aj opticky (cez displej v aute) na ne vodiča upozorniť. Nie všetky informácie zachytené vozidlom, mali byť aj vodičovi interpretované. Dôvodom malo byť, že nie všetky informácie sú v danom čase pre vodiča dôležité. Systém mal upozorniť len na nebezpečné situácie na trase vozidla. Výhodou použitia bezdrôtovej WiFi siete malo byť to, že táto technológia už existovala a nevyžadovala si žiadne náklady na prevádzku, čo platí dodnes. V roku 2009 mala byť Európskou úniou na účel komunikácie medzi vozidlami schválená jedna spoločná frekvencia (Karpat, 2007).

### 1.1.1 Skúšanie v praxi

Autor (Karpat, 2007) v spomínanom článku vysvetľuje, ako to v tom čase malo fungovať v praxi. Vysvetlil to na príklade hmly: „Ak vodič vojde do hmly, prvé, čo urobí zníži rýchlosť a zapne hmlové svetlá. Keď to zopakuje desať áut po sebe, dá sa táto informácia považovať za dôveryhodnú v porovnaní s tým, keď mladí zapnú hmlovky bezdôvodne len tak pre frajerinu.“ Inžinieri automobilky General Motors im umožnili si celý systém názorne vyskúšať. K dispozícii mali 4 vozidlá.

Prvá ukážka spočívala v tom, ako vie systém V2V upozorniť na blížiace sa policajné alebo záchranné vozidlo. Problém, ktorý nastal a platí dodnes je, že sirénu síce počujeme, ale nemusíme vždy tušiť odkiaľ dané vozidlo prichádza, čo môže spôsobiť vodičovi stres. Systém ukázal z ktorej strany sa dané vozidlo blíži a ako je od nich ďaleko. Myšlienku znázorňuje obrázok ?.



Obrázok Blížiace sa záchranné vozidlo signalizované vo V2V systéme

Druhá ukážka bolo stojace vozidlo v neprehľadnej zákrute alebo dopravné obmedzenie na neprehľadnom mieste. Aj keď prekážka nebola v ich jazdnom pruhu a nehrozila kolízia, tak systém V2V aj tak včas upozornil, akusticky aj vizuálne, že na ceste je prítomná prekážka.

Ďalšia ukážka spočívala v tom, ako systém funguje pri vznikajúcej zápche na diaľnici. Systém na pomaly idúce alebo stojace vozidlo pred nimi upozrnil najprv akusticky a vizuálne na displeji. Ak vodič nezareagoval na žiadne upozornenie, napríklad kvôli telefonovaniu, tak ich vozidlo poslalo informáciu vozidlu tomu pred ním, že sa blíži, a to zaplo brzdové svetlá.

Autor v článku ďalej vysvetľuje, že okrem upozorňovania mal systém V2V vedieť zasiahnuť aj v reálnom čase. Popísal to na príklade, kde blížiace vozidlo prichádzajúce do križovatky z bočnej cesty si Vás nevšíma a nespomaľuje, tak naše vozidlo to zaregistruje a ešte pred vjazdom do križovatky prudko zabrzdí. Vďaka vzájomnej komunikácii medzi sebou by však mohli obe vozidlá zastať súčasne.

### 1.1.2 Uvedenie do praxe

Predstavitelia automobilky General Motors chceli tento systém uviesť do praxe v priebehu 5 až 10 rokov. Aby tento systém fungoval v praxi, tak predpokladali, že minimálne 15 % jazdiacich vozidiel by muselo byť vybavených týmto systémom. Jedným zo základných predpokladov úspechu tohto systému bolo aj to, že by mal byť dostupný aj v lacnejších verziách vozidiel. Viceprezident pre vývoj automobilky General Motors v EÚ v tom čase povedal: „Čím viac vozidiel bude vybavených týmto systémom, tým efektívnejšia bude jeho účinnosť. Preto je dôležité, aby systém V2V bol cenovo dostupný (Karpat, 2007).“

## 1.2 V2V systém v súčasnosti

„Automobilový priemysel kladie na dnešné bezdrôtové komunikačné systémy súbor nových požiadaviek. Komunikačné aplikácie vozidiel zabezpečujúce bezpečnosť nemôžu tolerovať dlho trvajúce nadväzovanie spojenia predtým ako sa vozidlá stretnú na ceste a začnú medzi sebou komunikovať. Podobne aj komunikačné aplikácie, ktoré nesúvisia s bezpečnosťou, taktiež závisia na efektívnom nadviazaní spojenia so zariadeniami poskytujúcimi služby, ako napríklad aktualizácia mapy, pretože čas, počas ktorého prejde vozidlo cez oblasť pokrytú signálom, je limitovaný. Navyše, rýchlo sa pohybujúce vozidlá a zložité okolie cesty predstavujú výzvy na úrovni fyzickej vrstvy OSI modelu,“ uvádzajú autori (Jiang, a iní, 2008) v konferenčnom príspevku „IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments.“

National Highway Traffic Safety Administration (NHTSA) je americký úrad zodpovedený za bezpečnosť ľudí na amerických cestách. Prostredníctvom presádzania štandardov a partnerstiev so štátom a miestnymi vládnymi organizáciami, redukuje úmrtia, zranenia a ekonomické straty pri nehodách motorových vozidiel (nhtsa.gov, 2019).

Na stránke (nhtsa.gov, 2019) sa ďalej uvádza, že NHTSA spolupracuje s automobilovým priemyslom a akademickými inštitúciami už vyše dekádu, aby sa V2V systém stal realitou kvôli jeho potenciálu zachraňovať životy.

Od prvých náznakov a ambícii z roku 2007 na komunikáciu vozidiel medzi sebou ubehlo už viac ako desaťročie. V rámci tejto kapitoly (1.2) sme v prvej podkapitole stručne porovnali rozdiely súčasnosti a v krátkosti rozobratej minulosti (kapitola 1.1), a síce čo pribudlo, od čoho sa upustilo, čo sa zmenilo, vylepšilo alebo jednoducho ostalo rovnaké. V druhej podkapitole sme sa zamerali na bezdrôtové technológie, pomocou ktorých sa dnes dá, pri niektorých aspoň teoreticky, V2V systém realizovať. V tretej podkapitole sa zameriavame na samotnú bezpečnosť V2V systému. V poslednej podkapitole sa venujeme prekážkam, ktoré v súčasnosti bránia zavedeniu V2V systému do praxe.

### 1.2.1 Porovnanie rozdielov minulosť/súčanosť

Oblasť systému V2V sa rozšírila do viacero smerov. V článku „Talking Cars: A Survey of Protocols for Connected Vehicle Communication“ autorka (Jain, 2018) vysvetľuje podobné systémy ako V2V systém: „Vozidlá sa nepochybne stávajú chytrejšími a technológia vo vozidle sa zlepšuje. Jedným z aspektov, ktorým sa venuje veľká pozornosť, je schopnosť inteligentného vozidla komunikovať s cestujúcimi, ako aj s jeho okolím.“ Tieto systémy môžu byť kategorizované nasledovne:

* V2E (Vehicle-to-Environment) – Vozidlo-Prostredie – predstavuje komunikáciu medzi vozidlom a dynamicky sa meniacim prostredím (V2V, V2P) ako aj komunikáciu medzi vozidlom a statickým prostredím (V2I, V2G). Patrí sem:
* V2V,
* V2I (Vehicle-to-Infrastructure) – Vozidlo-Infraštruktúra – semafory, mýtne brány, cesty, a iné,
* V2P (Vehicle-to-Pedestrian) – Vozidlo-Chodec – chodci, cyklisti, a iné,
* V2G (Vehicle-to-Grid) – Vozidlo-Sieť – elektrická sieť,
* V2U (Vehicle-to-User) – Vozidlo-Používateľ – predstavuje komunikáciu medzi vozidlom a jeho posádkou vrátane vodiča. Jedná sa o prispôsobenie vozidla a spríjemnenie tak cestovania,
* V2N (Vehicle-to-Network) – Vozidlo-Sieť – predstavuje komunikáciu vozidla a sieťového operátora, ktorý poskytuje užitočné informácie ako dočasné uzavretie cesty alebo tvoriaca sa kolóna na našej trase. Môže predstavovať aj prístup k internetovej sieti,
* V2D (Vehicle-to-Device) – Vozidlo-Zariadenie – predstavuje komunikáciu vozidla s elektronickým zariadením pripojeným k vozidlu (telefóny a iné),
* V2X (Vehicle-to-Everything) – Vozidlo-Všetko – predstavuje všetky vyššie spomenuté systémy v jednom.

Koncept bezdrôtovej siete sa samozrejme zachoval. Tak isto, hlavné informácie, ktoré si vozidlá bezdrôtovo vymieňajú sú rýchlosť, GPS lokácia a smer jazdy. V minulosti sa hovorilo o dosahu 2 až 5 kilometrov. V súčasnosti sa však hovorí približne o dosahu viac ako 300 metrov (nhtsa.gov, 2019). Niektoré zdroje uvádzajú 1 kilometer, no všetko je závislé od použitej technológie.

Podľa článku „Will Vehicle-to-Vehicle Communication Ever Take Off?“ od autora (Koon, 2019) sú v kurze dve konkurenčné a vzájomne sa vylučujúce technológie na implementáciu V2V systému. Prvou z nich je Dedicated Short-Range Communication (DSRC) založenej na známom protokole WiFi siete a druhou je 5G celulárna sieť, ktorú autor opisuje ako rýchlejšiu, s nižšou odozvou a väčším dosahom (500 metrov). Najväčšou prekážkou je, že obe tieto technológie používajú rovnaké frekvenčné pásma a nevedia medzi sebou spolupracovať. Výhodou technológie DSRC je, že nie je závislá na žiadnej externej celulárnej sieti. Autor uvádza aj to, že podľa primyselného analytika sa automobilový priemysel prikloní skôr k 5G sieti kvôli jej lepším vlastnostiam, hoci niektoré automobilky sú naopak za DSRC technológiu, aby sa vyhli poplatkom za jej používanie. NHTSA explicitne nepresadzuje ani jedno, ani druhé, avšak častejšie sa odkazuje na technológiu DSRC.

Tvrdenie, že sa na prenos informácií použije technológia bezdrôtovej WiFi siete sa stalo čiastočne pravdivé. Ako sme spomenuli vyššie, v hre je aj ďalšia technológia. V roku 2010 bol schválený štandard IEEE 802.11p, ktorý predstavuje vylepšenú bezdrôtovú WiFi sieť oproti jej predchodcom, pridáva bezdrôtový prístup v dopravnom prostredí (Wireless Access in Vehicular Environment – WAVE) vyžadovaný aplikáciami ITS (Intelligent Transportation Systems), a ktorý je základom pre DSRC (standards.ieee.org, 2010). Bližšie tento štandard rozoberieme v podkapitole (1.2.2.4).

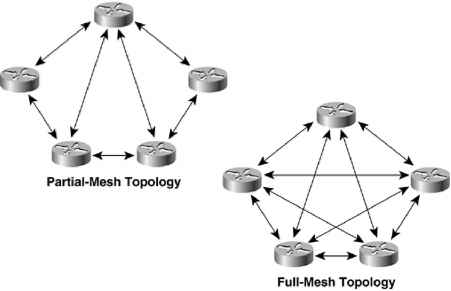
Koncept upozornenia vodiča V2V systémom na nebezpečenstvo ostal. Okrem toho vie tento systém aj sám zareagovať. Dnes sa však začalo hlasnejšie hovoriť aj o plne autonómnych vozidlách, čo je vyšší level V2V systému. Pri týchto vozidlách si len zadáme cieľ kam chceme ísť. Pre zaujímavosť, ako prvá spoločnosť takéto auto začala predávať spoločnosť Tesla v roku 2014, konkrétne vozidlo Tesla Model S. Systém sa volá Autopilot a dnes už je reálne inštalovaný do všetkých vozidiel Tesla. Prvenstvo si však drží spoločnosť Google, ktorá ako vôbec prvá začala na takomto vozidle pracovať, a to v roku 2009 (vozidlo Waymo) (Dormehl, a iní, 2019).

Automobilka General Motors mala v pláne zaviesť technológiu V2V systému v najbližších 5 až 10 rokoch v tej dobe. Dnes však vieme, že sa to realitou nestalo. Pre viaceré automobilky sa v súčasnosti ako zlomový rok javí 2023 (Hall-Geisler, 2017).

Pri analýze týchto rozdielov sme dospeli k tomu, že za toto desaťročie sa V2V systém posunul určite vpred, no vyskytli sa nové problémy a výzvy, pričom veľa z nich nie je jednoduché vyriešiť.

### 1.2.2 Bezdrôtové technológie

V článku „An Overview of Vehicular Communications“ autori (Arena, a iní, 2019) uvádzajú, že V2V systém používa na prenos informácií medzi vozidlami bezdrôtovú Ad-Hoc Mesh sieť. Jedná sa o decentralizovaný systém spojenia medzi zariadeniami. Mesh topológia môže byť plne prepojená medzi uzlami alebo len čiastočne. Na obrázku ? môžeme vidieť rozdiel medzi týmito topológiami.



Obrázok Čiastočne a plne prepojená Mesh sieťová topológia

Pri čiastočne prepojenej Mesh topológii sú len niektoré uzly priamo pospájané medzi sebou, a teda môžu medzi sebou priamo komunikovať bez sprostredkovateľa (single-hop). Ak priame spojenie neexistuje, komunikácia môže ísť prostredníctvom uzla, s ktorým vedia komunikovať oba koncové uzly (multi-hop). Spojenia medzi uzlami môžu vznikať aj zanikať dynamicky podľa toho, ako často a v akom objeme si dáta vymieňajú. Pri plne prepojenej Mesh topológii sú priamo pospájané všetky uzly (Arena, a iní, 2019).

#### 1.2.2.1 Bluetooth

V dnešnej dobe existuje viacero technológií a protokolov na bezdrôtovú komunikáciu medzi zariadeniami vo všeobecnosti. Bluetooth je jednou z týchto technológií. Slúži na výmenu informácií medzi zariadeniami na krátku vzdialenosť. Slúži na vytvorenie tzv. Personal Area Network (PAN), čiže na osobnú sieť dostupnú na malom rozsahu v okolí. Pracuje na UHF (Ultra High Frequency) rádiovej frekvencii v rozsahu od 2.402 do 2.480 GHz. Bluetooth má proprietárny otvorený štandard označený IEEE 802.15.1, avšak už nie je udržiavaný. Vývoj zastrešuje zoskupenie Bluetooth SIG (Special Interest Group), ktoré má 35 000 členských organizácií. (Singh, a iní, 2015).

Bluetooth bol predstavený v roku 1994 ako náhrada pre RS-232 káble. Využiť sa dá napríklad na prehrávanie hudby v bezdrôtových slúchadlách púšťanej cez telefón, prenos súborov medzi dvoma telefónmi a podobne (techopedia.com, 2019).

V dobe písania tejto práce je najnovšou verziou Bluetooth 5.1. Dosah signálu je až do 200 metrov (niekde uvádzajú aj 400, resp. 600 metrov) v exteriéri a 50 metrov v interiéri. Rýchlosť prenosu dát je 2 Mb/s. Je spätne kompatiblný s predchádzajúcimi verziami. Veľkosť paketu je 255 bajtov. Dokáže eliminovať rušivé vplyvy a tým pádom aj znížiť počet neočakávaných odpojení (10najs.sk, 2018).



Obrázok Anglický názov Bluetooth pri doslovnom preklade predstavuje modrý zub

#### 1.2.2.2 ZigBee

Ďalšiou bezdrôtovou technológiou je ZigBee. Stránka (techopedia.com, 2019) uvádza, že ZigBee má otvorený štandard označený IEEE 802.15.4, a teda môže byť implementovaný hocikým. Používa nízko energetické rádiové signály na vytváranie PAN sietí, ktoré pracujú s nižšou rýchlosťou prenosu dát, sú energeticky efektívne a bezpečné. Využitie si táto technológia našla v inteligentných domácnostiach, automatizovaných systémoch, medicínskych zariadeniach, atď. Na rozdiel od Bluetooth-u je navrhnutý jednoduchším spôsobom. Funguje na princípe Mesh sieťovej topológii. Poskytuje vysokú spoľahlivosť a rozumné rozpätie dostupnosti v okolí. Komunikácia je šifrovaná 128 bitovými kryptografickými kľúčmi. Obe strany používajú na šifrovanie a dešifrovanie rovnaké kľúče.

Stránka (electronics-notes.com, 2019) uvádza, že technológia ZigBee môže byť spoľahlivo použitá v prostredí, kde existuje viacero rádiových signálov. Vyvíjaný a udržiavaný je alianciou Zigbee Alliance, ktorá bola založená v roku 2002. Prenos údajov spoľahlivo funguje na vzdialenosť 70 metrov medzi dvoma uzlami, avšak pokiaľ sa komunikácia realizuje prostredníctvom viacerých medzi-uzlov, tak vzdialenosť môže byť oveľa väčšia. Technológia vysiela na frekvenčných pásmach 2.4 GHz, 915 MHz pre severnú Ameriku a 868 MHz pre Európu. Na frekvencii 2.4 GHz má k dispozícii celkovo 16 kanálov s maximálnou rýchlosťou prenosu 250 Kb/s. Frekvencia 915 MHz má k dispozícii celkovo 10 kanálov s rýchlosťou prenosu 40 Kb/s a frekvencia 868 MHz má k dispozícii 1 kanál s rýchlosťou prenosu 20 Kb/s. Dáta sú prenášané v paketoch s maximálnou veľkosťou 128 bajtov, pričom samotný obsah paketu (tzv. payload) môže mať maximálne 104 bajtov. Technológia podporuje až 64 bitové adresy. V dobe písania tejto práce je najnovšou verziou ZigBee 3.0.

Autor (Blank, 2016) uvádza, že technológia ZigBee umožňuje neobmedzené množstvo hop-ov medzi uzlami a podporuje až 65 536 zariadení v jednej sieti.

#### 1.2.2.3 Wireless Local Area Network

Ďalšiou bezdrôtovou technológiou je Wireless Local Area Network (WLAN), známejšia však ako WiFi sieť. WLAN sieť je štandardizovaná v skupine s označením IEEE 802.11.

„802.11 sa vzťahuje na skupinu špecifikácií vyvinutých asociáciou IEEE pre WLAN technológiu. 802.11 špecifikuje vzdušné rozhranie medzi bezdrôtovým klientom a základňou (napr. router) alebo dvoma bezdrôtovými klientmi. Asociácia IEEE schválila špecifikáciu 802.11 v roku 1997,“ uvádza autor (Beal, 2019) vo svojom článku „802.11 IEEE wireless LAN standards.“

Autor (Beal, 2019) ďalej uvádza, že v skupine 802.11 nájdeme tieto špecikácie:

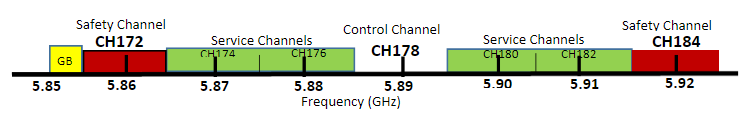
* 802.11 – poskytuje prenosovú rýchlosť 1 až 2 Mb/s na frekvencii 2.4 GHz podľa toho, či používa metódu vysielania rádiového signálu Frequency Hopping Spread Spectrum (FHSS) alebo Direct Sequence Spread Spectrum (DSSS),
* 802.11a – vylepšená 802.11, ktorá poskytuje prenosovú rýchlosť až do 54 Mb/s na frekvencii 5 GHz. Namiesto FHSS a DSSS používa metódu digitálneho kódovania dát na viacerých nosných frekvenciách Orthogonal Frequency-Division Multiplexing (OFDM),
* 802.11b – označovaná tiež ako 802.11 High Rate alebo WiFi. Založená je na 802.11 a poskytuje prenosovú rýchlosť 11 Mb/s na frekvencii 2.4 GHz. Používa iba metódu DSSS. Poskytuje funkcionalitu porovnateľnú s Ethernet protokolom,
* 802.11e – definuje koncept Quality of Service (QoS) a vylepšuje 802.11a a 802.11b. Pridáva podporu pre multimédiá a poskytuje spätnú kompatibilitu s uvedenými špecifikáciami,
* 802.11g – používa sa na prenos na krátke vzdialenosti s rýchlosťou do 54 Mb/s na frekvencii 2.4 GHz,
* 802.11n – stavia na predchádzajúcich špecifikáciach a pridáva princíp Multiple Input/Multiple Output (MIMO). Pridaný vysielač a prijímač umožňuje vyššiu dátovú priepustnosť prostredníctvom nezávislého a oddeleného vysielania zakódovaných dátových singálov (spatial multiplexing), známych ako prúdy (streams). Poskytuje aj väčší dosah. Prenosová rýchlosť je 100 Mb/s a na fyzickej vrstve až 250 Mb/s,
* 802.11ac – vylepšuje 802.11n. Prenosové rýchlosti dosahujú až 433 Mb/s cez spatial multiplexing, prípadne až 1.3 Gbps pri použití troch antén. Pracuje len na frekvencii 5 GHz a podporuje širšie kanály od 80 MHz do 160 MHz.

Okrem toho existujú aj menej známe špecifikácie, ktoré stoja za zmienku:

* 802.11ac Wave 2 – vylepšuje jej predchodkyňu, ktorá používa Multi User MIMO (MU-MIMO) technológiu, kde viacero používateľov s jednou alebo viacerými anténami komunikujú medzi sebou. Teoretická prenosová rýchlosť dosahuje až 6.93 Gb/s,
* 802.11ad – je vo vývoji, operuje na frekvencii 60 GHz a poskytuje prenosovú rýchlosť až do 7 Gb/s,
* 802.11ah – známa ako WiFi HaLow, operuje na frekvencii 900 MHz, teda nižšie ako všetky ostatné. Dosah má takmer 2-krát väčší ako ostatné WiFi technológie a je schopná preniknúť cez steny a iné bariéry (Beal, 2019).

#### 1.2.2.4 IEEE 802.11p - DSRC

V konferenčnom príspevku „IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments“ autori (Abdelgader, a iní, 2014) uvádzajú, že IEEE 802.11p je štandard z roku 2010, ktorý umožňuje bezdrôtový prístup v dopravnom prostredí (Wireless Access in Vehicular Environment – WAVE). WAVE je špecifikované v štandarde s označením IEEE 1609. IEEE 802.11p je základom pre DSRC (Dedicated Short-Range Communication). Vylepšuje IEEE 802.11 a pridáva požadovanú podporu aplikáciám ITS (Intelligent Transportation Systems). Patrí sem napríklad výmena dát medzi vozidlami jazdiacimi vysokou rýchlosťou (V2V) a medzi vozidlami a cestnou infraštruktúrou (V2I). Pracuje na rádiových frekvenciách od 5.850 do 5.925 GHz s rozostupom medzi kanálmi o veľkosti 20, 10 alebo 5 MHz. Umožňuje výmenu informácií na vzdialenosť 1 kilometra s prenosovou rýchlosťou od 3 do 27 Mbit/s pri rýchlosti 260 km/s. Obrázok ? ukazuje, že môže operovať až na 7 kanáloch, pričom frekvenčné pásma 5.860 GHz (kanál 172) a 5.920 GHz (kanál 184) sú určené pre bezpečnosť – prvý poskytuje seriózne bezpečnostné riešenia a druhý predstavuje ochranu voči preťaženiu na ostatných kanáloch. Kanál 178 (5.890 GHz) je zodpovedný za nadviazanie spojenia a vysielanie. Zvyšné kanály sú alokované pre obojsmernú komunikáciu.



Obrázok Frekvenčné pásma kanálov štandardu IEEE 802.11p

#### 1.2.2.5 Porovnanie

Všetky spomenuté technológie pracujú na alebo okolo frekvenčného pásma 2.4 GHz. WiFi, akú poznáme z každodenného života, najčastejšie pracuje vo forme hviezdicovej topológie na rozdiel od zvyšných dvoch, tzn. že komunikácia medzi zariadeniami je realizovaná pomocou sprostredkovateľa (routra). Vie však pracovať aj v mesh topológii, čo je pri V2V systéme základ. Bluetooth a ZigBee sú energeticky úspornejšie oproti WiFi, avšak poskytujú aj nižšie prenosové rýchlosti. Bluetooth vie podľa uvedených informácií poskytnúť najlepší dosah signálu, avšak štandard 802.11p poskytujúci garantovaný minimálne 300 metrový dosah je dostatočný na včasnú reakciu vodiča pri primeranej rýchlosti. Pri dvoch vozidlách idúcich vysokou rýchlosťou (130 km/h) oproti sebe (spolu 260 km/h) je to však málo. Takéto vozidlo prejde za 1 sekundu 36 metrov, 300 metrov dosiahne za ani nie 8,5 sekundy. Obe však majú k dispozícii 150 metrovú brzdnú dráhu, čiže ani nie 4,25 sekundy. Približne 4,25 sekundy by mali vozidlá k dispozícii na zabrzdenie v prípade, že si vymenia informácie čo najskôr (vzdialených od seba rovných 300 metrov).

#### 1.2.2.6 Ostatné bezdrôtové technológie

Za zmienku stojí aj bezdrôtová technológia Z-Wave. Je dosť podobná ZigBee. Rozdielom je, že Z-Wave je vyvíjaný a udržiavaný súkromnou organizáciou a poskytuje nižšie prenosové rýchlosti. Pracuje s maximálne 232 zariadeniami. Z-Wave je považavaný však o niečo spoľahlivejšiu technológiu (Blank, 2016).

Okrem vyššie spomenutých bezdrôtových technológií sme sa dopátrali aj k ďalším, pre nás menej známym technológiam ako ANT, Thread, DASH7, SigFox, nWave, 6LoWPAN. Niektoré z nich majú dosah len niekoľko metrov, iné naopak aj niekoľko kilometrov, všetky však ponúkajú nízke prenosové rýchlosti. Kvôli tomu nemusia byť vhodné na implementáciu V2V systému pri posielaní/prijímaní väčšieho objemu dát.

### 1.2.3 Bezpečnosť V2V systému

„Pridanie komunikácie medzi vozidlami a cestnou infraštruktúrou umožňujú skvelé nové služby (napr. bezpečnú jazdu, rýchejší presun, atď). Z bezpečnostného hľadiska to však vytvára nové hrozby. Je pravdepodobné, že „zlý človek“ sediaci pri ceste s bezdrôtovým prístupom sa môže nabúrať do nášho vozidla alebo do semaforov pred nami (Vamosi, 2017).“

Autor (Vamosi, 2017) vo svojom blogu „New V2V communication could give hackers a free ride“ ďalej uvádza, že v roku 2015 poukázal výskumník Cesar Cerrudo na značnú zraniteľnosť tisícok cestných bezdrôtových senzorov. Senzory používali bezdrôtovú technológiu ZigBee, kde chýbalo šifrovanie a autentifikácia. Prístup k zopár cestným senzorom sa nemusí zdať ako nebezpečné. Avšak, čo ak útočník zmení semafory na červenú a zastaví dopravu vo všetkých smeroch alebo nebodaj opačne? Takéto niečo sa stalo v Los Angeles v roku 2009. Použitá bola síce iná technológia, ale na niekoľko hodín boli odstavené strategické križovatky. Keďže sa vozidlá hýbu, aj v opačnom smere, nie je čas na nadviazanie/akceptovanie spojenia a autentifikáciu, ktoré poznáme zo štandardnej WiFi siete. Preto štandard IEEE 802.11p umožňuje okamžitú komunikáciu hneď ako je vozidlo v dosahu. Prístupovým bodom odovzdáva dočasné tokeny, ktoré zabezpečia hladký a relatívne anonymný prechod.

Autor (Koon, 2019) vo svojom článku hovorí, že NHTSA tiež stanovuje, že systémy V2V nebudú zhromažďovať, vysielať ani zdieľať osobné informácie medzi vozidlami, a ani nebudú umožňovať sledovanie konkrétneho vodiča alebo vozidla.

### 1.2.4 Prekážky brániace zavedeniu V2V systému do praxe

V2V systém stále nie je zavedený v praxi. Čelí viacerým prekážkam, ktoré tomu bránia a ktoré musia byť vyriešené aby sa tak stalo.

Autor (Graham, 2017) uvádza vo svojom článku „Vehicle-to-vehicle communication tech going nowhere“: „Jedným z problémov brániacich implementácii V2V systému je chýbajúca dohoda výrobcov automobilov o jednotných normách, financovaní a ochrane osobných údajov.“ Ďalej dodáva: „Na zabezpečenie silnej komunikačnej siete je potrebná povinná štandardizácia spracovania, analytických schopností a technických možností. Existuje však obava, že dosiahnutie dohody o štandardoch (komunikačného) protokolu, bezpečnostných otázkach a protiopatreniach môže zastaviť akýkoľvek posun trhu vpred.“

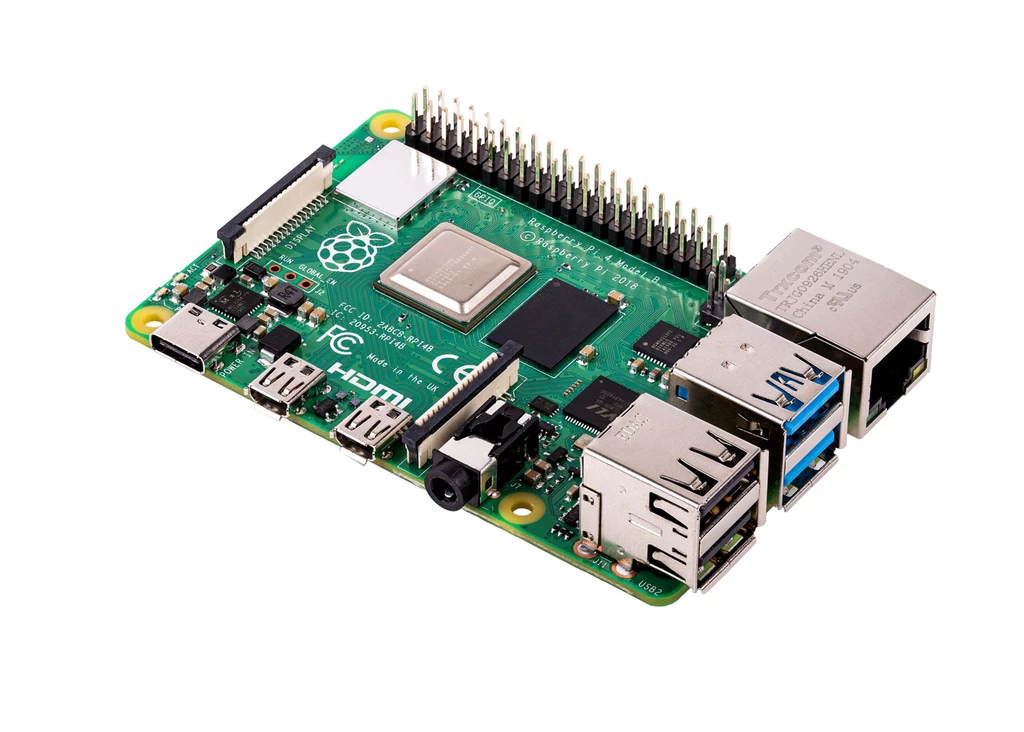
V našej práci sme doteraz opisovali V2V systém ako systém pracujúci na úrovni LAN siete. Ak by V2V systém bol napojený aj na internet (teda WAN), tak autor (Graham, 2017) tvrdí: „Systém V2V produkuje veľa údajov, ktoré vyžadujú škálovateľné cloudové úložisko schopné tieto dáta uložiť.“

„Ďalším veľkým problémom s posunom V2V je nesúlad štátnych a miestnych zákonov, ktorými sa dnes V2V systém riadi. Počas Obamovej administratívy sa regulačné orgány snažili urýchliť uvedenie V2V systému na trh, ale pod administratívou Trumpa sa to zastavilo,“ vo svojom článku uvádza autor (Graham, 2017).

## 1.3 Minipočítač Raspberry Pi

Raspberry Pi je cenovo dostupný počítač veľkosti kreditnej karty, ktorý sa pripája na TV alebo monitor na zobrazovanie obsahu a používa klasickú klávesnicu a myš na ovládanie. Jedná sa o šikovné malé zariadenie, ktoré umožňuje ľuďom rôznych vekových kategórií skúmať výpočtovú vedu a naučiť sa ako programovať v jazykoch ako Scratch, Python a iné. Je schopné zvládnuť všetko čo dokáže klasický desktopový počítač od prehliadania webu až po prehrávanie videí a hrania videohier. Ba čo viac, Raspberry Pi má široké využitie v rôznych oblastiach po celom svete ako zdravotníctvo, priemysel, automatizácia a pod. Projekt vedie vzdelávacia nadácia Raspberry Pi Foundation so sídlom v Spojenom kráľovstve Veľkej Británie (raspberrypi.org, 2019).

Na obrázku ? je znázornený opisovaný minipočítač. Na obrázku sa konkrétne nachádza jeho 4. generácia.



Obrázok Minipočítač Raspberry Pi 4

### 1.3.1 Raspberry Pi 3

Stránka (thepihut.com, 2016) prezentuje Raspberry Pi 3 (model B) v poradí ako 3. generáciu tohto kultového minipočítača. Tak ako jeho predchodcovia, aj tento model má veľkosť kreditnej karty a využitie nájde v rôznych oblastiach. Oproti jeho predchodcom ponúka väčší výkon a síce je 10-krát rýchlejší ako jeho 1. generácia. Taktiež má v sebe zabudované moduly pre bezdrôtovú sieť WiFi a Bluetooth. Technická špecifikácia tohto minipočítača je nasledovná:

* čipset Broadcom BCM2387,
* 4-jadrový procesor s taktom 1,2 GHz a 64 bitovou architektúrou ARM Cortex-A53,
* zabudovaný modul pre bezdrôtovú sieť WiFi štandardu IEEE 802.11b/g/n,
* zabudovaný modul pre bezdrôtovú sieť Bluetooth 4.1,
* pamäť RAM o veľkosti 1 GB,
* 4-krát USB 2.0 konektor,
* 4-pólový stereo výstup a kompozitný video port,
* HDMI výstup v HD kvalite,
* 10/100 Mbit Ethernet konektor pre LAN sieťový kábel,
* CSI konektor pre kameru,
* DSI konektor pre dotykový displej,
* slot pre Micro SD kartu, ktorá slúži ako diskové úložisko,
* napájací zdroj Micro USB,
* 40-pinový GPIO (General-Purpose Input/Output) konektor,

V rámci našej diplomovej práce chceme použiť práve túto generáciu minipočítača.

### 1.3.2 Raspberry Pi 4

V roku 2019 vyšla 4. generácia tohto minipočítača. Stránka (thepihut.com, 2019) uvádza viacero vylepšení oproti 3. generácii. A dosť výrazných. Medzi najhlavnejšie patrí zvýšenie taktu procesora na 1,5 GHz, pri pamäti RAM máme na výber 1 GB, 2 GB alebo 4 GB verziu, moduly WiFi štandardu IEEE 802.11a/c a Bluetooth 5.0, 10/100/1000 Mbit Ethernet konektor pre LAN sieťový kábel, dva USB 3.0 konektory a mnoho ďalších.

## 1.4 OBD (II)

OBD je skratkou pre On-Board Diagnostics. Je to systém, ktorý je zabudovaný v takmer každom vozidle jazdiacom po cestách. Výnimku tvoria vozidlá spred roku 1996. Dnes z výroby nevyjde vozidlo bez tohto systému. V sedemdesiatych a začiatkom osemdesiatych rokov začali výrobcovia používať elektronické zariadenia na kontrolu funkčnosti motora a diagnostiku jeho problémov. V priebehu rokov sa OBD systém stal sofistikovanejším a v polovici 90 rokov bolo predstavené OBD-II. To dokáže skontrolovať takmer úplne celý motor a monitoruje aj podvozok, trup vozidla, príslušenstvo a diagnostickú sieť. Jednoducho povedané, vie nám zhodnotiť zdravotný stav vozidla.

Hlavný dôvod, prečo OBD systém vznikol bol nadmerný smog v Los Angeles v roku 1966. Kongres v tom čase schválil Clean Air Act in 1970 a založil agentúru na ochranu životného prostredia Environmental Protection Agency (EPA). EPA vydala emisné normy a požiadavky na údržbu vozidiel. Aby výrobcovia splnili tieto nároky, začali do svojich vozidiel inštalovať elektronicky riadené systémy ako vstrekovanie paliva alebo zapaľovanie. Senzory potom merali výkon motora a znečistenie. Každý výrobca si najskôr vytvoril vlastný systém, a preto v roku 1988 spoločnosť Society of Automotive Engineers (SAE) štandardizovala konektor známy ako OBD (viď obrázok ?) a testovacie signály.

Celkovo dnes existuje 5 základných protokolov s minimálnymi rozdielmi, ktoré OBD-II používa na komunikáciu medzi vozidlom a skenovacím prístrojom. Sú to ISO 9141, KWP2000, SAE J1850 VPW, SAE J1850 PWM a najnovší z roku 2008, CAN. Európske vozidlá používajú protokol štandardu ISO 9141 alebo KWP2000. Príkazy pri každom protokole sú však definované podľa štandardu SAE J1979.

OBD systém má konektor štandardne umiestnený pod prístrojovou doskou na strane vodiča. Cez OBD systém dokážeme aj preprogramovať výkonostné parametre vozidla, nejedná sa však o tzv. prečipovanie vozidla, pretože to OBD-II už nedovoľuje  (obdii.com, 2011).

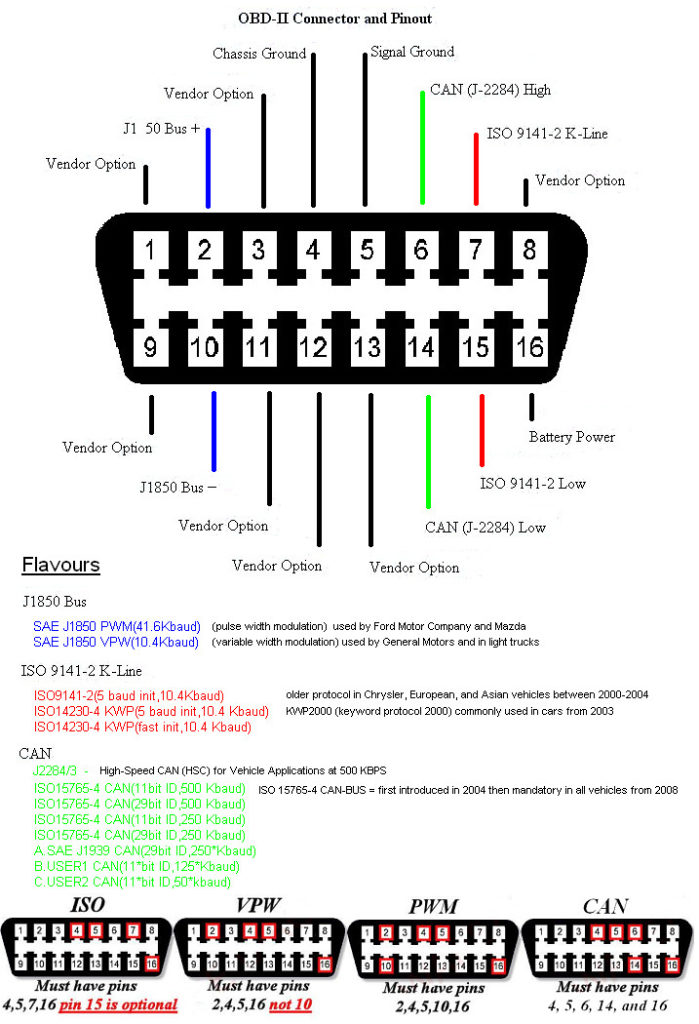
Aby sme získali určitú informáciu o vozidle, musíme dať vozidlu príkaz. OBD-II definuje širokú škálu príkazov označovanú ako PIDs (Parameter IDs). Príkaz je číselný kód pomocou ktorého získame požadovanú informáciu o vozidle ako napríklad rýchlosť vozidla, otáčky motora a podobne. Príkazy sú kategorizované na diagnostické služby, a to nasledovne:

* 01 – aktuálne informácie o vozidle,
* 02 – rámcové informácie o vozidle,
* 03 – uložené chybové kódy,
* 04 – resetuje uložené chybové kódy a hodnoty,
* 05 – výsledky testu monitorovania senzoru kyslíka,
* 06 – výsledky testu monitorovania palubných systémov,
* 07 – nedávne chybové kódy detekované počas aktuálnej alebo poslednej jazdy,
* 08 – kontrola palubného systému,
* 09 – všeobecné informácie o vozidle (napr. VIN číslo),
* 0A – permanentné chybové kódy.

Nie všetky vozidlá však podporujú aj všetky príkazy. Výrobcovia si definujú aj vlastné príkazy (obdsol.com, 2020).

V roku 2015 sa stal emisný škandál automobilky Volkswagen kvôli tomu, že do áut inštalovali softvér, ktorý počas laboratórneho testovania emisií aktivoval emisnú kontrolu, aby spĺňalo emisnú normu. Celkovo bol tento softvér nainštalovaný v 11 miliónoch áut. Nakoniec vysvitlo, že takéto praktiky nepoužívala iba táto automobilka.

My sami si doma môžeme diagnostikovať vozidlo. Stačí si zohnať OBD-II skener, ktorý vysiela informácie prostredníctvom bezdrôtovej siete Bluetooth, spárovať ho s telefónom, stiahnúť si aplikáciu na to určenú a sledovať „živé“ informácie o vozidle. Takýto skener je dôležitou súčasťou aj našej diplomovej práce.



Obrázok Schéma OBD-II konektora a rozloženie PIN-ov podľa protokolov

# 2 Ciele diplomovej práce

V diplomovej práci sa pokúsime vytvoriť funkčný V2V systém pomocou minipočítača Raspberry Pi, programovacieho jazyka Python a rozhrania OBD-II.

Hlavným cieľom diplomovej práce je:

* navrhnúť V2V systém po hardvérovej a softvérovej stránke,
* pokúsiť sa zrealizovať tento návrh krok za krokom – v prípade, ak narazíme na niečo, čo sa nedá zrealizovať podľa vytvoreného návrhu, tak pokúsiť sa nájsť náhradné riešenie a to následne zrealizovať,
* v prípade úspešnej realizácie návrhu otestovať vytvorený V2V systém v praxi a vyhodnotiť dosiahnuté výsledky, a naopak, pri neúspešnej realizácii návrhu identifikovať a vysvetliť príčinu, prečo sa nám ho nepodarilo zrealizovať.

# 3 Návrh, realizácia a testovanie V2V systému

V rámci tejto kapitoly (3) sa pokúsime vytvoriť V2V systém a zistiť či v praxi funguje alebo nie. Táto kapitola je rozdelná na tri podkapitoly. V prvej podkapitole (3.1) si spravíme návrh V2V systému – aký hardvér budeme potrebovať, aký operačný systém použijeme, aké softvérové balíčky budeme potrebovať, na aké logické celky bude rozdelený, aké budú mať jednotlivé logické celky úlohy, kde a ako budeme programovať. V druhej podkapitole (3.2) sa budeme snažiť krok za krokom návrh zrealizovať. Samozrejme to čo vytvoríme budeme priebežne testovať vo vozidle, avšak získané poznatky a problémy, ktoré zásadne nezabránia napredovaniu vývoja, rozoberieme až v nasledujúcej podkapitole (3.3). V tretej podkapitole (3.3), ak sa nám podarí vytvoriť funkčný V2V systém, rozoberieme ako ho vo vozidle spustiť, nastaviť a testovať. Rozoberieme tu aj poznatky a problémy, s ktorými sme sa stretli v priebehu vývoja a testovania. Na záver zhodnotíme, či systém funguje alebo nefunguje, kedy funguje a kedy nefunguje.

## 3.1 Návrh

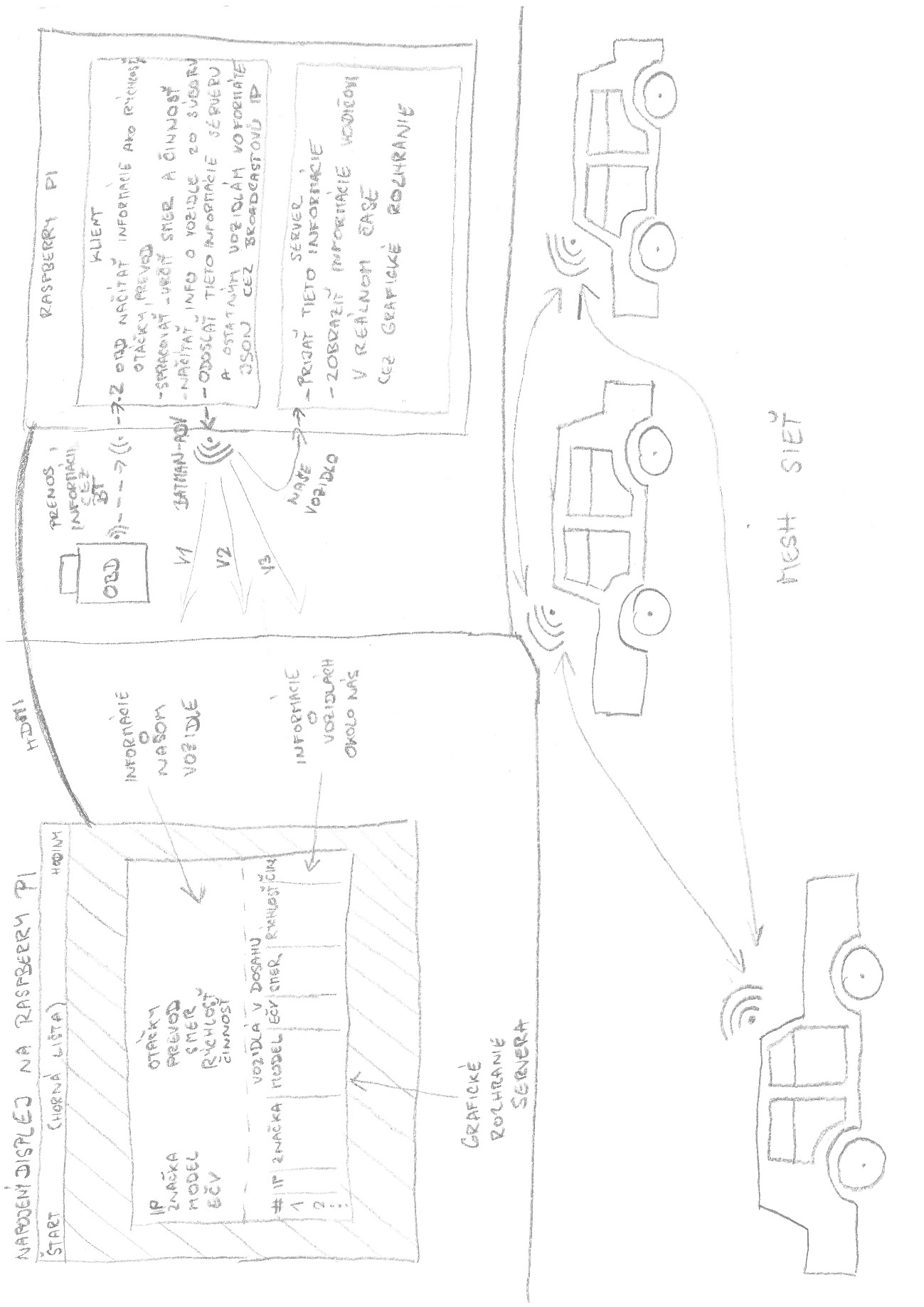
V tejto podkapitole (3.1) popíšeme koncept, z čoho sa V2V systém bude skladať (hardvérová stránka) a ako by mohol fungovať (softvérová stránka). Najskôr popíšeme hardvérovú stránku. Základom bude minipočítač Raspberry Pi. My použijeme generáciu 3 model B (#1). Výhodou práve tohto minipočítača oproti jeho predošlím generáciám je už zabudovaný WiFi modul. Ideálne je ho uložiť do ochrannej krabičky, aby neprišlo k jeho poškodeniu. Budeme potrebovať zobrazovaciu jednotku, teda nejaký displej. My sme sa rozhodli pre 5 palcový dotykový LCD displej značky Waveshare s rozlíšením 800x480 (#2). K nemu potrebujeme dva káble, ktorými sa k minipočítaču pripája, a sú to: USB-A/microUSB pre napájanie (#3) a HDMI pre zobrazovanie (#4). Budeme potrebovať dve microUSB nabíjačky. Jednu do klasickej elektrickej zásuvky (#5) a jednu do zapaľovania vo vozidle s nabíjacím prúdom 2A (#6). Ako úložisko používa minipočítač Raspberry Pi pamäťovú microSD kartu (#7). Minimálne musí mať 8 GB. Aby sme ju mohli vložiť do slotu v počítači, tak budeme k nej potrebovať adaptér (#8). Prenos údajov z vozidla do minipočítača bude zabezpečovať OBD-II skener (#9) cez Bluetooth. My sme sa rozhodli pre „Vgate ELM327 Bluetooth OBD2 V2.1 Scanner.“ Jednotlivé komponenty môžete vidieť na obrázku 7. Okrem týchto komponentov budeme ešte potrebovať veľký monitor s HDMI vstupom, klávesnicu a myš (ideálne 2 v 1), pripojenie do WAN siete cez LAN kábel, a samozrejme vozidlo. Keďže V2V systém je o dvoch a viacerých vozidlách, tak všetky vyššie spomenuté komponenty potrebujeme dva a viackrát, okrem malého displeja a s ním súvisiacich káblov, microUSB napájacieho adaptéra, veľkého monitora s HDMI vstupom, klávesnice, myši, LAN káblu.



Obrázok Hardvérové komponenty nášho V2V systému

Teraz rozoberieme, ako by sme chceli V2V systém riešiť z pohľadu softvérovej stránky. Ako operačný systém použijeme Raspbian, ktorý je šitý na mieru práve pre minipočítač Raspberry Pi. Minipočítač Raspberry Pi generácie 3 podporuje spojenie zariadení cez Bluetooth a operačný systém Raspbian má v sebe zabudovaného softvérového manažéra, ktorý sa o toto stará. Spojenie s OBD-II skenerom by sme chceli riešiť práve cez tohto manažéra. Pri predbežnom zisťovaní, aké informácie vieme získať cez rozhranie OBD-II sme zistili, že informácie ako značka vozidla, model vozidla, evidenčné číslo vozidla (EČV) nedokážeme cez neho získať. Tieto informácie však budeme chcieť zobraziť vodičovi na displeji. Preto ich pre každé vozidlo vyplníme manuálne. Vytvoríme si preto nejaký jednoduchý textový súbor, kde budú uložené tieto informácie. Ako sme spomínali v analýze súčasného stavu, tak V2V systém funguje v sieťovej topológii mesh. Pri hľadaní na internete sme narazili na softvérový balíček BATMAN-ADV, ktorý sa javí ako riešenie tohto problému. Programovanie budeme realizovať v jazyku Python verzie 3.7 a priamo na minipočítači. Operačný systém Raspbian ponúka na vývoj Python aplikácií integrované vývojové prostredie ThonnyIDE, čiže programovať budeme v ňom. Programovať budeme štýlom Dependency Injection. Získavanie informácií o vzidle cez rozhranie OBD-II by sme chceli realizovať pomocou Python balíčka obd. Zo skúseností s programovacím jazykom Python vieme, že ak chceme získať IP adresu zariadenia, tak musíme na to použiť Python balíček netifaces. Tieto dva teda určite budeme musieť doinštalovať, nakoľko nie sú súčasťou programovacieho jazyka Python. Grafické rozhranie budeme riešiť cez Python balíček tkinter. Sieťovú komunikáciu zase cez Python balíček socket. Tieto dva inštalovať nebudeme, pretože sú súčasťou programovacieho jazyka Python. Náš V2V systém bude rozdelený na dva logické celky, t. j. aplikácia klienta a aplikácia servera. Aplikácia klienta bude mať úlohu sluhu. Jej hlavnou úlohou bude načítať informácie o našom vozidle z textového súboru (značka vozidla, model vozidla, evidenčné číslo vozidla – ĚCV), čítať informácie o vozidle cez rozhranie OBD-II (rýchlosť vozidla, počet otáčok motora za minútu, zaradený prevodový stupeň), spracovať tieto informácie (určiť z nich smer jazdy vozidla a činnosť vozidla), rozposlať cez broadcastovú IP adresu všetky tieto informácie vozidlám okolo nás a nášmu vozidlu, resp. aplikácii servera, vo formáte JSON. Aplikácia klienta nebude zobrazovať žiadne informácie vodičovi. Naopak aplikácia servera bude mať úlohu pána. Tá bude pomocou grafického rozhrania zobrazovať aktuálne informácie vodičovi. Jednak bude zobrazovať informácie o našom vozidle, a jednak, v tabuľke „Vozidlá v dosahu,“ informácie o ostatných vozidlách v našom okolí. Z vyššie spomenutých informácií nám u vozidiel v dosahu však nebude zobrazovať počet otáčok motora za minútu a zaradený prevodový stupeň, pretože z pohľadu nášho vozidla to pre nás nie sú dôležité informácie. Ďalší problém, ktorý nám tu vzniká je spustenie V2V systému. Celý tento proces musí byť plne automatizovaný. To znamená, pri spustení minipočítača sa aplikácie klienta aj servera musia spustiť automaticky. V prípade zlyhania sa systém musí vedieť reštartovať bez intervencie používateľa, pretože minipočítač nemusí mať pripojené žiadne periférne zariadenia. Nakoniec, keďže by sme celý náš systém chceli skúsiť aspoň s tromi vozidlami, tak inštalácia nášho systému v operačnom systéme Raspbian musí byť tiež maximálne zjednodušená, resp. automatizovaná. Preto by sme chceli vytvoriť inštalačný skript, ktorý pri jeho spustení spraví túto prácu za nás.

Návrh riešenia nášho V2V systému po hardvérovej aj softvérovej stránke môžete vidieť na obrázku 8.



Obrázok Návrh riešenia nášho V2V systému z pohľadu hardvérovej a softvérovej stránky

## 3.2 Realizácia

V tejto podkapitole (3.2) sa pustíme do samotnej realizácie V2V systému. V podkapitole 3.1 sme načrtli jednoduchý koncept, ako by to mohlo fungovať a ako by sme to chceli zrealizovať. Tak ako sme to napísali, tak takej štruktúry sa budeme držať a pôjdeme krok za krokom. Ak narazíme na problém, že sa to nedá riešiť ako sme to navrhli, tak na to upozorníme a opíšeme nové riešenie.

### 3.2.1 Inštalácia operačného systému Raspbian

Existuje viacero známych distribúcií operačných systémov, ktoré sú prispôsobené pre minipočítač Raspberry Pi, ako napríklad Ubuntu MATE, Ubuntu Core, Ubuntu Server, Windows 10 IoT Core a mnoho ďalších. My sme sa však rozhodli pre oficiálny operačný systém Raspbian, vytvorený priamo od nadácie Raspberry Pi Foundation, ktorá za minipočítačmi stojí. Nejdeme tu do detailu popisovať postup inštalácie tohto operačného systému, pretože ten je možné nájsť v oficiálnom návode „Installing operating system images“ na URL adrese: https://www.raspberrypi.org/documentation/installation/installing-images/README.md. Pri inštalácii sme sa aj my držali práve tohto návodu. Pri inštalácii máme na výber, či ju vykonáme cez inštalátor NOOBS alebo bez neho, čiže priamym stiahnutím obrazu (súbor .img) operačného systému Raspbian. My sme ju realizovali bez neho a tento obraz sme stiahli na URL adrese: https://www.raspberrypi.org/downloads/raspbian/. Na výber máme viacero obrazov, my sme sa však rozhodli pre „Raspbian Buster with desktop and recommended software.“ Za poznamenie stojí, že súbor obrazu je zaobalený v .zip súbore, čiže sťahujeme .zip súbor a nie priamo .img súbor. Prvá časť inštalácie spočíva v zapísaní .img súboru operačného systému Raspbian na pamäťovú microSD kartu pomocou aplikácie balenaEtcher (v prostredí Windows). Následne pamäťovú microSD kartu zasunieme do minipočítača, pripojíme periférne zariadenia a zapneme ho. Druhá časť inštalácie spočíva v preklikaní sa rôznymi nastaveniami samotnej inštalácie operačného systému, ako napríklad akú jazykovú mutáciu používať, aké rozloženie klávesnice používať, aké časové pásmo používať a podobne. Je vhodné si zvoliť nejaký jedinečný názov pre minipočítač v sieti, napríklad rpi-1, rpi-2 a podobne. Buď ho upraviť priamo pri inštalácii alebo cez príkaz raspi-config, prípadne sudo nano /etc/hostname.

### 3.2.2 Inštalácia softvérového balíčka BlueZ

Pri prvých pokusoch o spárovanie minipočítača Raspberry Pi s OBD-II skenerom sme dospeli k tomu, že zabudovaný Bluetooth manažér v operačnom systéme Raspbian nepostačuje na splnenie našich požiadaviek – poloautomatizácia párovania a automatizácia nadviazania spojenia pri spustení minipočítača. Preto sme pátrali po alternatíve na internete a dostali sme sa k softvérovému balíčku BlueZ. „BlueZ je oficiálny Bluetooth protokol v linuxe. Jedná sa o projekt s otvoreným zdrojovým kódom distribuovaný pod licenciou GNU General Public Licence (GPL),“ uvádza sa na stránke (bluez.org, 2020). V operačnom systéme Raspbian ho cez príkazový riadok nainštalujeme pomocou príkazu sudo apt-get install bluez. Inštaláciou tohto softvérového balíčka získame k dispozícii príkaz bluetoothctl, ktorým vieme spárovať OBD-II skener s minipočítačom, pričom tento proces vieme vďaka tomu aj poloautomatizovať, a ktorý rozoberáme v kapitole 3.2.4. Okrem neho poskytuje aj príkaz rfcomm, pomocou ktorého vieme nadviazať spojenie s OBD-II skenerom a tento proces vieme taktiež vďaka tomu automatizovať.

### 3.2.3 Informácie o našom vozidle (súbor vehicleinfo.txt)

Súbor vehicleinfo.txt v sebe nesie základné informácie o našom vozidle. Súbor má jednoduchú štruktúru – jedna hodnota na jednom riadku a nepoužíva validáciu ako napríklad JSON. Štruktúra je nasledovná: Bluetooth MAC adresa OBD-II skenera, Bluetooth párovací kód, značka vozidla, model vozidla a nakoniec evidenčné číslo vozidla (EČV). Tieto informácie nevieme získať cez rozhranie OBD-II, preto ich musíme manuálne určiť pre každé vozidlo. Tento súbor je programom čítaný teda po riadkoch a číta len prvých 5 riadkov. To čo už je na 6. a ďalších riadkoch program nezaujíma. My sme si tam uložili Bluetooth MAC adresy všetkých našich OBD-II skenerov. Príklad súboru vehicleinfo.txt môžete vidieť v zdrojovom kóde 1.

00:00:00:33:33:33

1234

Škoda

Octavia

NR751GR

Zdrojový kód Súbor vehicleinfo.txt

### 3.2.4 Príkaz pre spárovanie OBD-II skenera s minipočítačom (súbor obd2rpi/pairwithobd)

Pri úplne prvom spustení minipočítača vo vozidle ešte nemáme žiadnym spôsobom prepojený OBD-II skener a minipočítač. V našom prípade riešime toto prepojenie cez bezdrôtovú sieť Bluetooth. Obe zariadenia teda musia podporovať tento typ siete. Prv než môžu obe zariadenia medzi sebou komunikovať, musia sa najskôr spárovať. Toto spárovanie nám stačí spraviť len raz. Kedy a ako ho vykonáme si povieme v oddieli 3.3.1. Tu si len rozoberieme nami vytvorený príkaz spustiteľný v príkazovom riadku, ktorý toto párovanie realizuje v jednom kroku. Implementáciu tohto príkazu môžete vidieť v zdrojovom kóde 2. Samotnému súboru musíme v jeho vlastnostiach nastaviť oprávnenie na exekúciu v príkazovom riadku. Tento príkaz využíva aj softvérový balíček expect, ktorý nainštalujeme pomocou príkazu sudo apt-get install expect.

Expect je program, ktorý „sa rozpráva“ s inými interaktívnymi programami alebo skriptami, ktoré vyžadujú interakciu používateľa. Pracuje tak, že keď iný program alebo skript očakáva vstup, tak mu ho poskytne podľa definovaných pravidiel bez interakcie používateľa. Je to nástroj, ktorý slúži na automatizáciu (Ebrahim, 2017).

Ako prvé si zo súboru vehicleinfo.txt vytiahneme Bluetooth MAC adresu OBD-II skenera a párovací kód. Spustíme interaktívny príkaz bluetoothctl cez spawn a potom čakáme na znak mriežky. Všeobecne platí, že keď je príkaz bluetoothctl pripravený prijímať (ďalšie) príkazy, tak vypíše znak mriežky. Ak je teda pripravený, pošleme mu prvý príkaz, aby zapol Bluetooth. Ak je úspešne zapnutý, pošleme mu príkaz aby zapol agenta. Úloha agenta je riadiť párovanie. Ak je úspešne zapnutý, registrujeme štandardného agenta, ktorý riadi párovanie pomocou párovacieho kódu. Ak existuje spojenie minipočítača s OBD-II skenerom, tak toto spojenie odstránime. Spustíme vyhľadávanie zariadení v okolí, počkáme 20 sekúnd a potom vyhľadávanie zastavíme. Ak sme našli OBD-II skener, tak s ním vytvoríme dôveryhodný vzťah a spustíme samotný proces párovania. Počas párovania si to vypýta párovací kód, ktorý sa automaticky zadá podľa toho, aký sme vytiahli zo súboru vehicleinfo.txt. Spojenie medzi OBD-II skenerom a minipočítačom by v tejto chvíli malo byť úspešne vytvorené. Následne už len ukončíme interakciu.

#!/usr/bin/expect -f

set address [exec head -1 /home/pi/Desktop/v2v/vehicleinfo.txt]

set pin [exec head -2 /home/pi/Desktop/v2v/vehicleinfo.txt | tail -1]

set prompt "#"

spawn bluetoothctl

expect $prompt

send -- "power on\n"

expect "succeeded"

expect $prompt

send -- "agent on\n"

expect "registered"

expect $prompt

send -- "default-agent\n"

expect "successful"

expect $prompt

send -- "remove $address\n"

expect -re "not available|removed"

expect $prompt

send -- "scan on\n"

expect "started"

expect $prompt

sleep 20

send -- "scan off\n"

expect "stopped"

expect $prompt

send -- "trust $address\n"

expect "succeeded"

expect $prompt

send -- "pair $address\n"

expect "PIN"

send -- "$pin\n"

expect "successful"

expect $prompt

send "quit\n"

expect "eof"

Zdrojový kód Súbor obd2rpi/pairwithobd

### 3.2.5 Príkaz pre nadviazanie spojenia OBD-II skenera a minipočítača (súbor obd2rpi/obd2rpi)

Keď už máme úspešne spárované obe zariadenia (OBD-II skener a minipočítač), tak ďalší krok je nadviazať medzi nimi spojenie, aby si mohli vymieňať informácie. Toto spojenie nám treba nadviazať pri každom spustení minipočítača, a pri každom zlyhaní tohto spojenia sa musí dokázať obnoviť. Celý tento proces bude musieť byť automatizovaný, pretože k minipočítaču nebudú musieť byť pripojené žiadne periférne zariadenia. Preto si vytvoríme príkaz spustiteľný v príkazovom riadku, ktorý toto spojenie nadviaže v jednom kroku. Implementáciu tohto príkazu môžete vidieť v zdrojovom kóde 3. Samotnému súboru musíme v jeho vlastnostiach nastaviť oprávnenie na exekúciu v príkazovom riadku. Tento príkaz využíva aj softvérový balíček bluetoothctl, presnejšie príkaz rfcomm z neho. Bluetooth MAC adresu OBD-II skenera si vyťahujeme zo súboru vehicleinfo.txt.

rfcomm connect hci0 "$(head -1 /home/pi/Desktop/v2v/vehicleinfo.txt)"

Zdrojový kód Súbor obd2rpi/obd2rpi

### 3.2.6 Inštalácia a konfigurácia softvérového balíčka BATMAN-ADV

„BATMAN-ADV pracuje iba na vrstve 2 modelu ISO/OSI. Používa a smeruje (alebo lepšie mostuje) ethernetové rámce. Emuluje virtuálny sieťový prepínač všetkých zúčastnených uzlov. Preto sa zdá, že všetky uzly sú lokálne prepojené, takže všetky vyššie prevádzkové protokoly nebudú ovplyvnené žiadnymi zmenami v sieti. Podporuje takmer akýkoľvek protokol, najmä však: IPv4, IPv6, DHCP, IPX,“ uvádza stránka (kernel.org, 2020). Inštaláciu vykonáme cez príkaz sudo apt-get install batctl.

#### 3.2.6.1 Konfigurácia

Keď už máme BATMAN-ADV nainštalované, tak nastavíme sieťové rozhranie, pod ktorým bude bude bezdrôtovo komunikovať s okolitými vozidlami. Aby sme mali všetky súbory tohto projektu pokope, tak v koreňovom adresáry projektu sme si vytvorili súbor /etc-network/interfaces, ktorého obsah je uvedený v zdrojovom kóde ?. Súbor /etc/network/interfaces sa bude odkazovať práve na tento súbor. Viac si však k tomu povieme v kapitole ?. V súbore /etc-network/interfaces sa nachádza iba jediný príkaz (source-directory), ktorý slúži na importovanie súborov z priečinka /etc/network/interfaces.d. Máme dva takéto súbory. Jeden pre sieťové rozhranie wlan0 (WiFi) a druhé pre sieťové rozhranie bat0 (BATMAN-ADV, t. j. ad-hoc mesh sieť).

# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd

# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:

source-directory /etc/network/interfaces.d

Zdrojový kód Súbor /etc-network/interfaces

Sieťové rozhranie wlan0 je definované v súbore /etc-network-interfaces.d/wlan0 podľa zdrojového kódu ?. Nastavenia pre toto sieťové rozhranie musia byť na všetkých minipočítačoch identické, t. j. MTU veľkosť, kanál, SSID, mód a MAC adresa prístupového bodu.

auto wlan0

iface wlan0 inet manual

    mtu 1532

    wireless-channel 1

    wireless-essid VehicleNetwork

    wireless-mode ad-hoc

    wireless-ap 02:12:34:56:78:9A

Zdrojový kód Súbor /etc-network-interfaces.d/wlan0

Sieťové rozhranie bat0 je definované v súbore /etc-network-interfaces.d/bat0 podľa zdrojového kódu ?. Tu platí to isté, nastavenia musia byť identické na všetkých minipočítačoch s výnimkou IP adresy. Tú nastavujeme rozdielne od vozidla k vozidlu, pričom je jedinečná, čo je zároveň vhodné, pretože každé vozidlo vieme jednoznačne identifikovať (v sieti). Na svete však existuje veľké množstvo vozidiel a súčasná IPv4 nie je vhodná na takéto použitie, hoci mi ju tu používame. Riešením tohto problému je IPv6.

auto bat0

iface bat0 inet static

    address 169.254.0.?

    netmask 255.255.255.0

    pre-up /usr/sbin/batctl if add wlan0

Zdrojový kód Súbor /etc-network-interfaces.d/bat0

Na záver ešte musíme určiť sieťové rozhranie (wlan0), ktoré má sieťové rozhranie bat0 používať na komunikáciu, pretože bat0 je virtuálne sieťové rozhranie. Následne obe sieťové rozhranie aktivujeme pomocou príkazu ifconfig. Celé to zakomponujeme do súboru, ktorý bude slúžiť ako príkaz, čiže treba mu navyše nastaviť aj oprávnenia pre exekúciu v príkazovom riadku. Implementáciu môžete vidieť v zdrojovom kóde ?.

#!/bin/bash

# Povieme BATMAN-ADV, ktoré sieťové rozhranie má používať

sudo batctl if add wlan0

# Zapneme sieťové rozhrania

sudo ifconfig wlan0 up

sudo ifconfig bat0 up

Zdrojový kód Súbor batman/batman

### 3.2.7 Príprava vývojového prostredia

Samotné programovanie systému budeme realizovať na minipočítači v operačnom systéme Raspbian. Ako textový editor budeme používať ThonnyIDE, ktorý je súčasťou operačného systému od inštalácie. Ak náhodou nie je, tak ho doinštalujeme jednoducho cez príkazový riadok príkazom sudo apt-get install python3-thonny.

### 3.2.8 Inštalácia programovacieho jazyka Python a balíčkov

Náš systém budeme programovať v programovacom jazyku Python verzie 3.7, ktorý by mal byť súčasťou nainštalovaného operačného systému Raspbian. Overiť to vieme cez príkazový riadok príkazom python3 --version. Ak tento príkaz vypíše chybu typu príkaz neexistuje, tak ho nainštalovaný nemáme. Následne ho nainštalujeme príkazom sudo apt-get install python3.7. Zatiaľ s určitosťou vieme, že budeme potrebovať dva Python balíčky, ktoré nie sú súčasťou balíčka python3.7. Sú to balíčky netifaces a obd. Oba nainštalujeme pomocou príkazu sudo pip3 install netifaces https://github.com/brendan-w/python-OBD/archive/master.zip. V druhom prípade sme použili odkaz na GitHub archív, pretože balíček obd poskytuje (v dobe písania) staršiu verziu, kde je chyba, a v tomto archíve už je táto chyba opravená.

### 3.2.9 Klientská aplikácia (priečinok vehicleclient)

Priečinok vehicleclient v sebe nesie súbory pre aplikáciu klienta. V našom prípade plní aplikácia klienta funkciu sluhu a má 4 úlohy:

* čítať informácie z OBD nášho vozidla,
* spracovať tieto informácie,
* rozposielať spracované informácie cez WiFi do prostredia okolo nášho vozidla, resp. ďalším vozidlám so spustenou aplikáciou servera (vrátane nášho vozidla),
* monitorovať „zdravie“ pripojenia OBD skenera cez bluetooth.

Z toho vyplýva, že aplikácia klienta žiadne informácie nezobrazuje. Aplikácia klienta sa skladá zo súborov mycommands.py, myvehicle.py, obdmonitor.py a vehicleclient.py (hlavný súbor aplikácie).

#### 3.2.9.1 Definovanie vlastných OBD príkazov (súbor mycommands.py)

Knižnica Python-OBD definuje najpoužívanejšie OBD príkazy, ktoré sú definované v štandarde OBD-II. Nájdeme tu napríklad príkaz pre získanie aktuálnej rýchlosti vozidla alebo otáčok motora. Ak niektorý príkaz nedefinuje, tak pomocou triedy OBDCommand nám umožňuje si takýto príkaz definovať. To sme sa pokúsili využiť pri probléme ako získať aktuálne zaradený prevodový stupeň motora. Knižnica Python-OBD takýto príkaz nedefinuje, tak sme sa snažili nájsť viac informácií na Google. Podarilo sa nám nájsť, že takýto príkaz štandard OBD-II definuje. Patrí do diagnostickej služby 01 a PID má A4. Slovne je opísaný ako „Transmission Actual Gear.“ Podľa dokumentácie knižnice Python-OBD, ktorá hovorí ako si vytvoriť vlastný príkaz, sme sa pokúsili tento príkaz implementovať. Nachádza sa práve v tomto súbore (mycommands.py). Implementáciu môžete vidieť v zdrojovom kóde ?. Tam, kde konvertujeme bajty na číslo, tak to delíme číslom 4, pretože zdroj uvádza, že táto informácia má veľkosť 4 bajty. Telo funkcie gearDecoder je založené na príklade uvedenom v spomínanej dokumentácii. Samotný príkaz je definovaný na konci súboru. Ako prvý argument je uvedený názov príkazu, následne jeho popis, číselný kód v binárnej podobe, počet bajtov informácie, odkaz na funkciu dekódujúcu informáciu, ECU (Engine Control Unit) filter na odfiltrovanie prichádzajúcich správ, a logická hodnota pre rýchlu odpoveď – súvisí len s knižnicou Python-OBD.

from obd import OBDCommand

from obd.protocols import ECU

from obd.utils import bytes\_to\_int

def gearDecoder(messages):

    data = messages[0].data

    # Odseknúť službu a PID

    data = data[2:]

    # Konvertovať bajty na číslo

    gear = bytes\_to\_int(data) / 4

    return gear

# PID (A4, resp. 01A4) a počet bajtov (4) nájdený tu:

# https://en.wikipedia.org/wiki/OBD-II\_PIDs

# Aj pri zaradenej rýchlosti však dostávame hodnotu 0.

# Skúšali sme aj iné (kvalitnejšie) OBD skenery,

# avšak pri nich sa aplikácia ani nespustila

GEAR = OBDCommand("GEAR", "Transmission Actual Gear",

                  b"01A4", 4, gearDecoder, ECU.ALL, False)

Zdrojový kód Súbor vehicleclient/mycommands.py

#### 3.2.9.2 Informácie o našom vozidle (súbor myvehicle.py)

Ďalšou časťou aplikácie klienta je trieda MyVehicle, ktorá je nositeľom informácií o našom vozidle. Medzi tieto informácie patrí: sieťová IP adresa, značka vozidla, model vozidla, evidenčné číslo vozidla (EČV), počet otáčok motora za minútu, zaradený prevodový stupeň, smer jazdy vozidla, rýchlosť vozidla, konštantná rýchlosť vozidla a činnosť vozidla. Všetky tieto informácie sú v triede MyVehicle reprezentované vlastnosťami triedy. Pri vytvorení novej inštancie tejto triedy majú uvedené vlastnosti nastavenú hodnotu None. Pred použitím musíme vytvorený objekt najskôr inicializovať. To zabezpečuje metóda init. Ako argumenty prijíma názov sieťového rozhrania (v našom prípade to je bat0) a názov súboru s informáciami o našom vozidle (v našom prípade to je vehicleinfo.txt). Metóda init najskôr zavolá internú metódu \_figureOutMyIp, ktorá pomocou knižnice netifaces a názvu sieťového rozhrania zistí, akú sieťovú IP adresu má priradené dané sieťové rozhranie. Tú potom uloží do vlastnosti ip. Hneď na to zavolá internú metódu \_loadInfoAboutMe, ktorá prečíta súbor s informáciami o našom vozidle, vytiahne odtiaľ značku, model a evidenčné číslo vozidla a tie potom uloží do vlastností brand, model a vrn. Keď už máme objekt inicializovaný, môžeme bezpečne volať ďalšie metódy. Pokiaľ chceme aktualizovať zvyšné vlastnosti, robíme tak pomocou metódy update. Tá prijíma ako argumenty počet otáčok motora za minútu, zaradený prevodový stupeň a rýchlosť vozidla. Počet otáčok motora za minútu uloží do vlastnosti rotates, rýchlosť vozidla zase do vlastnosti speed. Pred uložením rýchlosti však najskôr aktualizujeme činnosť vozidla (vlastnosť action), pretože, aby sme mohli určiť činnosť vozidla, musíme porovnať predchádzajúcu a aktuálnu rýchlosť vozidla. Aktualizáciu činnosti zabezpečuje interná metóda \_updateAction, ktorá prijíma ako argument rýchlosť vozidla. V zdrojovom kód ? môžete vidieť rôzne podmienky podľa ktorých sa určuje činnosť vozidla (stojí na mieste, zrýchľuje, brzdí, takmer stojí na mieste, zastavuje, spomaľuje), no základ tvorí porovnanie s aktuálnou rýchlosťou vozidla. Ak nie je splnená žiadna z podmienok, tak vozidlo sa nachádza v konštantom pohybe a vtedy sa navyše do vlastnosti constantSpeed uloží aktuálna rýchlosť vozidla, ktorá je základom pre overenie, či sa vozidlo stále pohybuje v konštantom pohybe s rozdielom +/–2 km/h. Pred aktualizáciou činnosti vozidla sa ešte aktualizujú vlastnosti gear (teda zaradený prevodový stupeň) a direction (smer jazdy vozidla). OBD-II poskytuje hodnotu prevodového stupňa v číselnej podobe. Interná metóda \_updateGear premieňa túto číselnú hodnotu na textovú, ktorú poznáme zo skutočnosti. Ak je hodnota rovná 0, tak máme zaradený „neutrál.“ Ak je hodnota menšia ako 0, tak máme zaradenú „spiatočku.“ V opačnom prípade máme zaradený „dopredný“ prevodový stupeň. Smer jazdy vozidla sa určuje na základe zaradeného prevodového stupňa a zabezpečuje to interná metóda \_updateDirection. Ak je zaradený „neutrál,“ tak smer jazdy je „nikde.“ Ak je zaradená „spiatočka,“ tak smer jazdy je „dozadu.“ Ak nie je zaradený ani jeden z týchto prevodových stupňov, tak je zaradený „dopredný“ prevodovoý stupeň a smer jazdy je „dopredu.“ Metóda toJson zoberie všetky aktuálne vlastnosti a zaobalí ich hodnoty do formátu JSON, ktorý využívame ako formát správy posielanej do aplikácie servera. Prijíma logický argument, či správa bude poslaná nášmu serveru alebo do prostredia ostatným vozidlám. Význam je taký, aby sme do prostredia neposielali zbytočné, prípadne citlivé dáta. Napríklad v našom prípade, je zbytočné, aby ostatné vozidlá dostali informáciu, aké otáčky má náš motor alebo akú rýchlosť máme zaradenú. Metóda toEncodedJson sa správa tak isto ako metóda toJson, len je doplnená o zakódovanie výstupu. Poslednou metódou je echo, ktorá vráti alternatívnu hodnotu v prípade, že hodnota vlastnosti je None. Implementáciu triedy MyVehicle môžete vidieť v zdrojovom kóde ?.

import json

import netifaces

class MyVehicle:

    ip = None              # IP adresa

    brand = None           # Značka

    model = None           # Model

    vrn = None             # EČV

    rotates = None         # Otáčky (za minútu)

    gear = None            # Prevodový stupeň

    direction = None       # Smer jazdy (číselný kód)

    directionAsText = None # Smer jazdy

    speed = None           # Rýchlosť (km/h)

    constantSpeed = None   # Konštantná rýchlosť (pomocná premenná)

    action = None          # Činnosť (číselný kód)

    actionAsText = None    # Činnosť

    def init(self, netIfName, infoFilename, nfs = netifaces):

        self.\_figureOutMyIp(netIfName, nfs)

        self.\_loadInfoAboutMe(infoFilename)

    def \_figureOutMyIp(self, netIfName, netifaces):

        bat0 = netifaces.ifaddresses(netIfName)

        self.ip = bat0[netifaces.AF\_INET][0]["addr"]

    def \_loadInfoAboutMe(self, infoFilename):

        with open(infoFilename, "r") as fp:

            fp.readline()         # MAC adresa OBD skenera

            fp.readline()         # Bluetooth párovací kód

            brand = fp.readline() # Značka

            model = fp.readline() # Model

            vrn = fp.readline()   # EČV

            # Z konca odstrániť znak nového riadku

            self.brand = brand[:-1]

            self.model = model[:-1]

            self.vrn = vrn[:-1]

    def update(self, rotates, gear, speed):

        self.rotates = rotates

        self.\_updateGear(gear)

        self.\_updateDirection()

        # Činnosť sa určuje na základe porovnania

        # aktuálnej (speed) a predchádzajúcej

        # rýchlosti (self.speed)

        self.\_updateAction(speed)

        self.speed = speed

    def \_updateGear(self, gear):

        if gear == None:

            self.gear = None

        elif gear == 0:

            self.gear = "N"

        elif gear < 0:

            self.gear = "R"

        else:

            self.gear = "D:" + str(gear)

    def \_updateDirection(self):

        if self.gear == None:

            self.direction = None

            self.directionAsText = None

        elif self.gear == "N":

            self.direction = 0

            self.directionAsText = "Nikde"

        elif self.gear == "R":

            self.direction = -1

            self.directionAsText = "Dozadu"

        else:

            self.direction = 1

            self.directionAsText = "Dopredu"

    def \_updateAction(self, speed):

        if speed == None:

            self.action = None

            self.actionAsText = None

        elif speed == 0:

            self.action = 0

            self.actionAsText = "Stojí na mieste"

        # Pri zrýchľovaní rozlišujeme,

        # či je vozidlo v konštantnom pohybe

        elif self.action != 2 and speed > self.speed:

            self.action = 1

            self.actionAsText = "Zrýchľuje!"

        # Detto

        elif self.action == 2 and speed >= self.constantSpeed + 3:

            self.action = 1

            self.actionAsText = "Zrýchľuje!"

        elif speed <= self.speed - 10:

            self.action = -4

            self.actionAsText = "BRZDÍ!!!"

        elif speed <= 5 and speed < self.speed:

            self.action = -1

            self.actionAsText = "Takmer stojí na mieste"

        elif speed <= 15 and speed < self.speed:

            self.action = -2

            self.actionAsText = "Zastavuje!"

        elif speed <= self.speed - 3:

            self.action = -3

            self.actionAsText = "Spomaľuje!"

        else:

            self.constantSpeed = speed

            self.action = 2

            self.actionAsText = "V konštantnom pohybe"

    def toJson(self, forMyServer = False):

        data = {

            "aboutMe": forMyServer,

            "ip": self.ip,

            "brand": self.brand,

            "model": self.model,

            "vrn": self.vrn,

            "direction": self.direction,

            "directionAsText": self.directionAsText,

            "speed": self.speed,

            "action": self.action,

            "actionAsText": self.actionAsText,

        }

        if forMyServer:

            data.update({

                "rotates": self.rotates,

                "gear": self.gear,

            })

        return json.dumps(data)

    def toEncodedJson(self, forMyServer = False):

        return self.toJson(forMyServer).encode()

    def echo(self, name, alt = "Zisťujem..."):

        val = getattr(self, name)

        if val == None:

            return alt

        else:

            return val

Zdrojový kód Súbor vehicleclient/myvehicle.py

#### 3.2.9.3 Monitorovanie stavu spojenia OBD skenera s minipočítačom cez bluetooth (súbor obdmonitor.py)

Pre nadviazanie bluetooth spojenia minipočítača RaspberryPi s OBD skenerom používame protokol RFCOMM, ktorý realizuje sériovú komunikáciu. V linuxovom jadre je implementovaný ako príkaz rfcomm. Tento príkaz nám funguje bez problémov, avšak sme našli jeden problém. V prípade, že sme vytiahli OBD skener z konektora, tak operačný systém si stále myslel, že spojenie je aktívne. Keď aj sme OBD skener napojili do konektora naspäť, tak spojenie už nefungovalo. Preto sme sa snažili nájsť riešenie. Nadviazanie spojenia je realizované vrámci systemd služby v2v-obd2rpi.service (rozoberáme ju v kapitole ?). Preto sme vytvorili triedu ObdMonitor, ktorá monitoruje „zdravie“ bluetooth spojenia medzi minipočítačom Raspberry Pi a OBD skenerom. Pri vytvorení objektu tejto triedy musíme ako argument zadať názov služby (v našom prípade to je v2v-obd2rpi.service). Metóda updateTime slúži na aktualizáciu vlastnosti \_lastUpdateAt. Táto vlastnosť reprezentuje čas, kedy naposledy sme z OBD skenera dostali nejakú informáciu. Tento čas sa v podstate aktualizuje každých pár milisekúnd. Pre overenie bluetooth spojenia slúži metóda isAlive. V prípade, že od času poslednej aktualizácie (\_lastUpdateAt) prešlo viac ako 10 sekúnd, tak vtedy považujeme bluetooth spojenie za stratené. V takom prípade reštartujeme službu v2v-obd2rpi.service. Samozrejme, po reštarte môže dôjsť k viacerým scenárom, tie však rozoberieme v kaptiole ?. Implementáciu môžete vidieť v zdrojovom kóde ?.

import subprocess

import time

class ObdMonitor:

    \_lastUpdateAt = None

    def \_\_init\_\_(self, serviceName,

                 t = time.time, execute = subprocess.call):

        self.\_serviceName = serviceName

        self.\_time = t

        self.\_exec = execute

    def updateTime(self, \_ = None):

        self.\_lastUpdateAt = self.\_time()

    def isAlive(self):

        if (self.\_time() > self.\_lastUpdateAt + 10):

            self.\_exec([

                "/bin/systemctl",

                "restart",

                self.\_serviceName,

            ])

            return False

        else:

            return True

Zdrojový kód Súbor vehicleclient/obdmonitor.py

#### 3.2.9.4 Vstupný súbor aplikácie klienta (súbor vehicleclient.py)

Súbor vehicleclient.py plní funkciu hlavného súboru aplikácie klienta. To znamená, že pri spustení aplikácie klienta sa ako prvý spracúva práve tento súbor. Definujeme v ňom triedu VehicleClient, ktorá zase v kontexte tried plní funkciu hlavnej triedy. Na začiatku triedy máme definované konštanty. Konštanty ALL\_IP a ALL\_PORT predstavujú broadcast-ovú IP adresu s portom, kde „počúvajú“ všetky vozidlá v okolitom prostredí. Konštanta ALL\_ADDR predstavuje tzv. tuple zložený z IP adresy a portu. Zvyšné tri konštanty predstavujú názov sieťového rozhrania (bat0), názov súboru s informáciami o našom vozidle (.../vehicleinfo.txt) a názov služby, ktorá nadväzuje bluetooth spojenie s OBD skenerom. Metóda \_\_init\_\_ je konštruktor, ktorý prijíma ako argumenty požadované závislosti (princíp dependency injection). Ako prvý argument, ak neberieme do úvahy parameter self, prijíma objekt nášho vozidla (MyVehicle), potom objekt UDP socketu (socket.socket), objekt OBD v asynchrónnej forme (obd.Async), objekt pre monitorovanie zdravia OBD (ObdMonitor). Za konštruktorom nasleduje definícia internej metódy \_registerShutdownHandler, ktorá v bežiacom procese aplikácie klienta registruje kus kódu, ktorý sa vykoná pri ukončení daného procesu so signálom INT alebo TERM. Kus kódu, ktorý sa vykoná pri ukončení daného procesu je uvedenýv metóde stop. V zásade sa ukončí spojenie s OBD skenerom a uzavrie sa UDP socket. Ďalej nasledujú interné metódy \_initUdpSock a \_initObd. V \_initUdpSock povieme UDP socketu, že bude vysielať formou broadcastu. V \_initObd zase registruje nami vytvorený OBD príkaz v objekte OBD, registrujeme OBD príkazy, ktorých hodnoty majú byť sledované v reálnom čase a nakoniec spustíme objekt OBD, aby začal čítať informácie z OBD skenera. Za povšimnutie stojí aj to, že pri registrácii OBD príkazu RPM uvádzame ako druhý argument odkaz na metódu updateTime objektu ObdMonitor. Na čo slúži táto metóda sme vysvetlovali v kapitole ?. Jedná sa o to, že pri každej novej hodnote RPM v reálnom čase sa zavolá aj metóda updateTime. Metóda start v podstate volá jednotlivé interné metódy. Inicializujeme tu aj objekt nášho vozidla (MyVehicle). Pred internou metódou \_run voláme manuálne metódu updateTime objektu ObdMonitor. Má to tiež charakter incializácie pred použitím práve v internej metóde \_run. Na koniec sme si nechali internú metódu \_run. Tá spustí cyklus podmienený zdravým spojenia s OBD, ktorý je po každej iterácii prerušený na 1 sekundu. Na začiatku cyklu získame nami požadované hodnoty: počet otáčok motora za minútu, zaradený prevodový stupeň a rýchlosť vozidla. Následne aktualizujeme informácie o našom vozidle cez metódu update. Keď to máme, tak najskôr pošleme tieto informácie aplikácii servera nášho vozidla (pri volaní funkcie toEncodedJson je ako argument explicitne uvedená hodnota True) a potom do prostredia okolo nás (pri volaní funkcie toEncodedJson je ako argument implicitne uvedená hodnota False). Ako sme uviedli, tak naša trieda VehicleClient má aj metódu start. Samotné spustenie aplikácie klienta sa udeje práve po zavolaní tejto metódy. A to realizujeme práve pod definíciou tejto triedy. Implementáciu môžete vidieť v zdrojovom kóde ?.

import mycommands

import netifaces

import obd

import signal

import socket

import sys

import time

from myvehicle import MyVehicle

from obdmonitor import ObdMonitor

class VehicleClient:

    ALL\_IP = "169.254.0.255" # Všetci v dosahu (broadcast) vrátane nás

    ALL\_PORT = 20000

    ALL\_ADDR = (ALL\_IP, ALL\_PORT)

    NETWORK\_INTERFACE\_NAME = "bat0"

    INFO\_FILENAME = "/home/pi/Desktop/v2v/vehicleinfo.txt"

    OBD\_SERVICE\_NAME = "v2v-obd2rpi.service"

    def \_\_init\_\_(self, vehicle, udpSock, obd, obdMon, allAddr = ALL\_ADDR):

        self.\_allAddr = allAddr

        self.\_vehicle = vehicle

        self.\_udpSock = udpSock

        self.\_obd = obd

        self.\_obdMon = obdMon

    def \_registerShutdownHandler(self):

        signal.signal(signal.SIGINT, self.stop)

        signal.signal(signal.SIGTERM, self.stop)

    def \_initUdpSock(self):

        self.\_udpSock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)

    def \_initObd(self):

        self.\_obd.supported\_commands.add(mycommands.GEAR)

        self.\_obd.watch(obd.commands.RPM, self.\_obdMon.updateTime)

        self.\_obd.watch(mycommands.GEAR)

        self.\_obd.watch(obd.commands.SPEED)

        self.\_obd.start()

    def \_run(self, sleep):

        while self.\_obdMon.isAlive():

            rotates = self.\_obd.query(obd.commands.RPM).value

            gear = self.\_obd.query(mycommands.GEAR).value

            speed = self.\_obd.query(obd.commands.SPEED).value

            if rotates != None:

                rotates = rotates.magnitude

            if speed != None:

                speed = speed.magnitude

            self.\_vehicle.update(rotates, gear, speed)

            # Odošleme informácie o našom vozidle lokálnemu serveru

            self.\_udpSock.sendto(self.\_vehicle.toEncodedJson(True),

                                (self.\_vehicle.ip, self.\_allAddr[1]))

            # Odošleme informácie o našom vozidle do prostredia

            # okolo (vrátane) nás

            self.\_udpSock.sendto(self.\_vehicle.toEncodedJson(),

                                 self.\_allAddr)

            sleep(1)

    def start(self, netIfName = NETWORK\_INTERFACE\_NAME,

              infoFilename = INFO\_FILENAME,

              nfs = netifaces, sleep = time.sleep):

        self.\_registerShutdownHandler()

        self.\_vehicle.init(netIfName, infoFilename, nfs)

        self.\_initUdpSock()

        self.\_initObd()

        self.\_obdMon.updateTime()

        self.\_run(sleep)

    def stop(self, sigNum = None, csf = None):

        try:

            self.\_obd.close()

        except Exception as e:

            print(e, file=sys.stderr)

        try:

            self.\_udpSock.close()

        except Exception as e:

            print(e, file=sys.stderr)

if \_\_name\_\_ == "\_\_main\_\_":

    vehicle = MyVehicle()

    udpSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    obdConn = obd.Async()

    obdMon = ObdMonitor(VehicleClient.OBD\_SERVICE\_NAME)

    client = VehicleClient(vehicle, udpSock, obdConn, obdMon)

    try:

        client.start()

    except KeyboardInterrupt:

        pass

    finally:

        client.stop()

Zdrojový kód Súbor vehicleclient/vehicleclient.py

### 3.2.10 Serverová aplikácia (priečinok vehicleserver)

Priečinok vehicleserver v sebe nesie súbory pre aplikáciu servera. V našom prípade plní aplikácia servera funkciu pána a má 4 úlohy:

* prijať spracované informácie vyslané aplikáciou klienta od nášho a ostatných vozidiel okolo nás,
* opätovne spracovať prijaté informácie z pohľadu aplikácie servera,
* udržovať aktualizovaný zoznam vozidiel okolo nás (pridávať nové vozidlá v dosahu do zoznamu a odstraňovať staré vozidlá mimo dosahu zo zoznamu),
* zobrazovať informácie v reálnom čase a v grafickej podobe vodičovi vozidla.

Aplikácia servera sa skladá zo súborov vehicle.py, foreignvehicles.py, objectmaker.py, gui.py a vehicleserver.py (hlavný súbor aplikácie).

#### 3.2.10.1 Informácie o našom alebo cudzom vozidle (súbor vehicle.py)

Prvou časťou aplikácie servera je súbor vehicle.py, ktorý definuje triedu Vehicle. Táto trieda je nositeľom informácií o našom alebo cudzom vozidle, čiže z tohto pohľadu je univerzálna. Informácie, ktoré táto trieda nesie, sú uložené vo vlastnostiach triedy a sú to: IP adresa vozidla, značka vozidla, model vozidla, evidenčné číslo vozidla (EČV), počet otáčok motora za minútu, zaradený prevodový stupeň, smer jazdy vozidla, rýchlosť vozidla a činnosť vozidla. Okrem týchto vlastností je definovaná ešte interná vlastnosť \_lastUpdateAt, ktorá reprezentuje čas poslednej aktualizácie informácií o vozidle. Všetky vlastnosti sú prednastavené na hodnotu None. Ako prvá je definovaná metóda update, ktorá aktualizuje informácie o vozidle podľa zadaných argumentov a navyše aktualizuje aj čas poslednej aktualizácie informácií (internej vlastnosti \_lastUpdateAt nastaví aktuálny čas v danom momente). Metóda isReachable overuje, či vozidlo je ešte v dosahu. Ak je aktuálny čas (argument time) menší alebo rovný ako čas poslednej aktualizácie informácií o vozidle (interná vlastnosť \_lastUpdateAt) + 3 sekundy, tak dané vozidlo je stále v dosahu. Táto metóda sa používa len pri overovaní prítomnosti cudzích vozidiel (teda nie nášho, to by nemalo zmysel). Predposlednou metódou je metóda echo, ktorá vráti alternatívnu hodnotu v prípade, že hodnota vlastnosti je None. Poslednou metódou je metóda fromJson. Tá prijíma dáta typu JSON, ktoré boli vyslané aplikáciou klienta. Môžeme ju zavolať ako metódu triedy alebo existujúceho objektu. V prípade, že ju zavoláme ako metódu triedy, tak vytvorí nový objekt a podľa prijatých dát aktualizuje informácie pomocou metódy update. V opačnom prípade, teda ak ju zavoláme ako metódu existujúceho objektu, urobí vlastne to isté len bez prvého kroku – vytvorenia nového objektu, a namiesto toho použije existujúci objekt. Za povšimnutie stojí ešte parameter requiresAboutMe. Ak je jeho hodnota True, tak informácie sa aktualizujú iba v prípade, že dáta obsahujú informácie o našom vozidle. Tu platí jednoduché pravidlo. Ak objekt Vehicle predstavuje naše vozidlo, tak tento parameter sa explicitne nastavuje na hodnotu True. Ak predstavuje cudzie vozidlo, tak ho explicitne nastavovať nemusíme, pretože implicitne je nastavenýna hodnotu False. Implementáciu triedy Vehicle môžete vidieť v zdrojovom kóde ?.

import json

import time

class Vehicle:

    ip = None

    brand = None

    model = None

    vrn = None

    rotates = None

    gear = None

    direction = None

    directionAsText = None

    speed = None

    action = None

    actionAsText = None

    \_lastUpdateAt = None

    def update(self, ip, brand, model, vrn, rotates, gear,

               direction, directionAsText, speed, action,

               actionAsText, t = time.time):

        self.ip = ip

        self.brand = brand

        self.model = model

        self.vrn = vrn

        self.rotates = rotates

        self.gear = gear

        self.direction = direction

        self.directionAsText = directionAsText

        self.speed = speed

        self.action = action

        self.actionAsText = actionAsText

        self.\_lastUpdateAt = t()

    def isReachable(self, time):

        return time <= self.\_lastUpdateAt + 3

    def echo(self, name, alt = "Zisťujem..."):

        val = getattr(self, name)

        if val == None:

            return alt

        else:

            return val

    def fromJson(self, data, requiresAboutMe = False):

        data = json.loads(data)

        if requiresAboutMe and not data.get("aboutMe"):

            return None

        if self == None:

            self = Vehicle()

        self.update(data.get("ip"),

                    data.get("brand"),

                    data.get("model"),

                    data.get("vrn"),

                    data.get("rotates"),

                    data.get("gear"),

                    data.get("direction"),

                    data.get("directionAsText"),

                    data.get("speed"),

                    data.get("action"),

                    data.get("actionAsText"))

        return self

Zdrojový kód Súbor vehicleserver/vehicle.py

#### 3.2.10.2 Zoznam cudzích vozidiel okolo nás (súbor foreignvehicles.py)

Vozidlá okolo nás, teda cudzie vozidlá, musíme udržovať v neuástle aktualizovanom zozname. Ak sa nejaké vozidlo dostane do nášho okolia, tak sa do tohto zoznamu musí pridať, a naopak, ak sa dostane mimo dosah, tak sa z tohto zoznamu musí odstrániť. Preto sme vytvorili triedu s názvom ForeignVehicles, ktorá plní túto úlohu. Má viaceré metódy vrátane magických. Samotný zoznam vozidiel je uložený v internej vlastnosti \_vehicles. Iterátor je zase uložený v internej vlastnosti \_it. Iterovanie objektu v cykle zabezpečujú magické metódy \_\_iter\_\_ a \_\_next\_\_. Magická metóda \_\_len\_\_ vracia počet vozidiel v zozname pri zavolaní funkcie len s objektom ako argument. Magická metóda \_\_getitem\_\_ umožňuje získať vozidlo zo zoznamu indexovým spôsobom ([i]). Metóda has overuje, či zoznam obsahuje vozidlo s danou IP adresou. Metóda get získa vozidlo podľa danej IP adresy alebo keď v zozname také vozidlo nie je tak, tak vráti None. Metóda add pridá na koniec zoznamu vozidlo. Posledná metóda removeAllUnreachable odstráni zo zoznamu všetky vozidlá, ktoré sa stratili z dosahu. Implementáciu triedy ForeignVehicles môžete vidieť v zdrojovom kóde ?.

class ForeignVehicles:

    \_it = None

    def \_\_init\_\_(self):

        self.\_vehicles = []

    def \_\_iter\_\_(self):

        self.\_it = iter(self.\_vehicles)

        return self

    def \_\_next\_\_(self):

        return next(self.\_it)

    def \_\_len\_\_(self):

        return len(self.\_vehicles)

    def \_\_getitem\_\_(self, i):

        return self.\_vehicles[i]

    def has(self, ip):

        return self.get(ip) != None

    def get(self, ip):

        for v in self.\_vehicles:

            if v.ip == ip:

                return v

        return None

    def add(self, vehicle):

        self.\_vehicles.append(vehicle)

    def removeAllUnreachable(self, time):

        for i, v in enumerate(self.\_vehicles):

            if not v.isReachable(time):

                del self.\_vehicles[i]

Zdrojový kód Súbor vehicleserver/foreignvehicles.py

#### 3.2.10.3 Továreň na tvorbu objektov (súbor objectmaker.py)

Trieda ObjectMaker predstavuje pomocnú triedu na tvorbu objektov. Nakoniec obsahuje len metódu, ktorá vytvorí objekt cuzdieho vozidla (trieda Vehicle). Implementáciu triedy ObjectMaker môžete vidieť v zdrojovom kóde ?.

from vehicle import Vehicle

class ObjectMaker:

    def foreignVehicle(self, json):

        return Vehicle.fromJson(None, json)

Zdrojový kód Súbor vehicleserver/objectmaker.py

#### 3.2.10.4 Grafické rozhranie (súbor gui.py)

Aplikácia servera získané informácie, o našom vozidle a cudzích vozidlách, zobrazuje vodičovi v reálnom čase. Trieda Gui reprezentuje grafické rozhranie. Jedná sa o klasické okno, aké poznáme napríklad z Windows-u. Nejdeme do detailu opisovať ako táto trieda funguje, pretože to nie je z pohľadu tejto aplikácie kľúčové. Niečo málo si však povieme. Grafické rozhranie je založené na knižnici Tkinter. Trieda Gui je odvodená od triedy Frame. Nový objekt vytvoríme zavolaním konštruktora s argumentami objektu nášho vozidla a zoznamu cudzích vozidiel. Interná metóda \_createMyVehicleView vytvára pohľad pre informácie o našom vozidle (vrchná časť okna). Naopak, interná metóda \_createForeignVehiclesView vytvára pohľad vo forme tabuľky pre informácie o cudzích vozidlách v našom okolí (spodná časť okna). Metódy updateMyVehicle a updateForeignVehicles aktualizujú tieto pohľady. V prvom prípade o našom vozidle a v druhom prípade o cudzích vozidlách v našom okolí. Metóda invokeLater je obdoba funkcie SwingUtilities.invokeLater z programovacieho jazyka Java. Pomocou tejto metódy vieme zabezpečiť, že požadovaný kód bude bežať vo vlákne grafického rozhrania. Grafické rozhranie je možné manipulovať len z tohto vlákna. Skúšali sme to aj z hlavného vlákna, avšak k požadovaným zmenám (t. j. k prepísaniu informácií) nedošlo. Nakoniec sú tu definované metódy show a close, kde v prvom prípade zobrazí okno a v druhom prípade ho zatvorí. Implementáciu triedy Gui môžete vidieť v zdrojovom kóde ? a ukážku okna na obrázku ?.



Obrázok Grafické rozhranie aplikácie servera

from tkinter import \*

class Gui(Frame):

    \_ipLabel = None

    \_brandLabel = None

    \_modelLabel = None

    \_vrnLabel = None

    \_rotatesLabel = None

    \_gearLabel = None

    \_directionLabel = None

    \_speedLabel = None

    \_actionLabel = None

    \_afterId = None

    def \_\_init\_\_(self, master, vehicle, foreignVehicles):

        Frame.\_\_init\_\_(self, master)

        self.\_master = master

        self.\_vehicle = vehicle

        self.\_foreignVehicles = foreignVehicles

        self.\_foreignVehicleLabels = []

    def \_createLabel(self, table, name, title, row, col):

        l = Label(table, text=title)

        l.grid(row=row, column=col \* 2, sticky=E)

        l = Label(table, text="Zisťujem...")

        l.grid(row=row, column=(col \* 2) + 1,

               sticky=W, padx=(10, 50))

        setattr(self, name + "Label", l)

    def \_createMyVehicleView(self):

        t = Frame(master=self.\_master)

        t.grid(row=0, column=0, sticky=W)

        self.\_createLabel(t, "\_ip", "IP", 0, 0)

        self.\_createLabel(t, "\_brand", "Značka", 1, 0)

        self.\_createLabel(t, "\_model", "Model", 2, 0)

        self.\_createLabel(t, "\_vrn", "EČV", 3, 0)

        self.\_createLabel(t, "\_rotates", "Otáčky (o/m)", 0, 1)

        self.\_createLabel(t, "\_gear", "Prevod", 1, 1)

        self.\_createLabel(t, "\_direction", "Smer", 2, 1)

        self.\_createLabel(t, "\_speed", "Rýchlosť (km/h)", 3, 1)

        self.\_createLabel(t, "\_action", "Činnosť", 4, 1)

    def \_createForeignVehiclesView(self):

        table = Frame(master=self.\_master)

        table.grid(row=1, column=0, sticky=W)

        titles = ["#", "IP", "Značka", "Model", "EČV",

                  "Smer", "Rýchlosť (km/h)", "Činnosť"]

        l = Label(table, text="VOZIDLÁ V DOSAHU")

        l.grid(row=0, column=0,

               columnspan=len(titles),

               pady=(20, 10))

        for i, t in enumerate(titles):

            l = Label(table, text=t)

            l.grid(row=1, column=i, sticky=W, padx=(0, 10))

        for i in range(5):

            self.\_foreignVehicleLabels.append([])

            for j in range(len(titles)):

                l = Label(table)

                l.grid(row=i + 2, column=j,

                       sticky=W, padx=(0, 10))

                self.\_foreignVehicleLabels[i].append(l)

    def updateMyVehicle(self):

        self.\_ipLabel["text"] = self.\_vehicle.echo("ip")

        self.\_brandLabel["text"] = self.\_vehicle.echo("brand")

        self.\_modelLabel["text"] = self.\_vehicle.echo("model")

        self.\_vrnLabel["text"] = self.\_vehicle.echo("vrn")

        self.\_rotatesLabel["text"] = self.\_vehicle.echo("rotates")

        self.\_gearLabel["text"] = self.\_vehicle.echo("gear")

        self.\_directionLabel["text"] = self.\_vehicle.echo("directionAsText")

        self.\_speedLabel["text"] = self.\_vehicle.echo("speed")

        self.\_actionLabel["text"] = self.\_vehicle.echo("actionAsText")

    def updateForeignVehicles(self):

        for i, row in enumerate(self.\_foreignVehicleLabels):

            row[0]["text"] = str(i + 1) + "."

            if i < len(self.\_foreignVehicles):

                v = self.\_foreignVehicles[i]

                row[1]["text"] = v.echo("ip")

                row[2]["text"] = v.echo("brand")

                row[3]["text"] = v.echo("model")

                row[4]["text"] = v.echo("vrn")

                row[5]["text"] = v.echo("directionAsText")

                row[6]["text"] = v.echo("speed")

                row[7]["text"] = v.echo("actionAsText")

            else:

                for j in range(7):

                   row[j + 1]["text"] = ""

    def invokeLater(self, runner):

        self.\_afterId = self.after(200, runner)

    def show(self, invokeLaterRunner):

        self.\_master.title("V2V komunikácia ... 2020 Bojnanský Dávid")

        self.\_master.resizable(True, False)

        self.\_createMyVehicleView()

        self.\_createForeignVehiclesView()

        self.invokeLater(invokeLaterRunner)

        self.\_master.mainloop()

    def close(self):

        if self.\_afterId != None:

            self.after\_cancel(self.\_afterId)

        self.\_master.quit()

Zdrojový kód Súbor vehicleserver/gui.py

#### 3.2.10.5 Spúšťací skript aplikácie servera (súbor vehicleserver.py)

Súbor vehicleserver.py plní funkciu hlavného súboru aplikácie servera. To znamená, že pri spustení aplikácie servera sa ako prvý spracúva práve tento súbor. Definujeme v ňom triedu VehicleServer, ktorá zase v kontexte tried plní funkciu hlavnej triedy. Na začiatku triedy máme definované konštanty IP a PORT, ktoré predstavujú IP adresu s portom, kde prijíma aplikácia servera informácie vyslané našim alebo cudzími vozidlami. Za povšimnutie stojí, že hodnota konštanty IP je nastavená na prázdny reťazec. To znamená, že aplikácia servera „počúva“ na všetkých priradených sieťových adresách (vrátane tej broadcastovej). Konštanta ADDR predstavuje tzv. tuple zložený z IP adresy a portu. Zvyšné dve konštanty predstavujú názov sieťového rozhrania (bat0) a veľkosť bufferu v bajtoch. Magická metóda \_\_init\_\_ je konštruktor, ktorý prijíma ako argumenty požadované závislosti (princíp dependency injection). Ako prvý argument, ak neberieme do úvahy parameter self, prijíma objekt továrne na tvorbu objektov (ObjectMaker), potom objekt nášho vozidla (Vehicle), objekt UDP socketu (socket.socket), objekt zoznamu cudzích vozidiel (ForeignVehicles), objekt grafického rozhrania (Gui). Za konštruktorom nasleduje definícia internej metódy \_registerShutdownHandler, ktorá v bežiacom procese aplikácie servera registruje kus kódu, ktorý sa vykoná pri ukončení daného procesu so signálom INT alebo TERM. Kus kódu, ktorý sa vykoná pri ukončení daného procesu je uvedenýv metóde stop. V zásade sa zatvorí grafické rozhranie a uzavrie sa UDP socket. Ďalej nasledujú interné metódy \_initUdpSock a \_figureOutMyIp. V \_initUdpSock povieme UDP socketu, že informácie budeme prijímať z broadcastu a že nechceme blokovať vlákno, ak žiadna informácia nie je k dispozícii. V \_figureOutMyIp zase zisťujeme, aká IP adresa je priradená k sieťovému rozhraniu podľa mena (bat0). Používame na to knižnicu netifaces. Predposlednou (internou) metódou je \_udpSockReceiver. Z UDP socketu prečíta prijaté JSON dáta. Následne podľa IP adresy identifikujeme, či sa prijaté dáta týkajú nášho alebo cudzieho vozidla. Ak sa týkajú nášho vozidla a sú kompletné (teda obsahujú aj informácie ako počet otáčok motora za minútu a zaradený prevodový stupeň), tak aktualizujeme informácie o našom vozidle pomocou metódy fromJson, kde za povšimnutie stojí druhý argument True. Následne aktualizujeme aj pohľad o našom vozidle. Ak sa jedná o cudzie vozidlo, tak sa ho podľa IP adresy pokúsime nájsť v zozname. Ak neexistuje, tak ho vytvoríme aj s danými informáciami a pridáme ho do zoznamu. Ak existuje, tak ho aktualizujeme pomocou metódy fromJson. Keďže sme nastavili UDP socket tak, aby neblokoval vlákno, tak musíme ošetriť aj chybu ku ktorej dochádza v prípade, že sa snažíme zo streamu čítať, avšak nič v ňom nie je. Vtedy je vyvolaná chyba BlockingIOError. V takom prípade ju jednoduchu vedome odignorujeme. Nakoniec prejdeme celý zoznam, či je aktuálny. Ak sa v ňom nachádzajú vozidlá, ktoré sú mimo dosah, tak sa z neho odstránia. Potom aktualizujeme pohľad cudzích vozidiel. Úplne na záver naplánujeme ďalšie spustenie tejto metódy o pár milisekúnd neskôr. Ako je zrejmé, celá táto činnosť sa odohráva vo vlákne grafického rozhrania, čo v praxi nie je dobrý nápad, pretože to môže znekvalitniť tzv. user experience napríklad spomalenou reakciou grafického rozhrania. Ideálne je tieto dve veci navzájom oddeliť. Pre jednoduchosť sme to však nechceli komplikovať. Posledná metóda start v podstate volá jednotlivé interné metódy a nakoniec zobrazí grafické rozhranie v podobe okna. Samotné spustenie aplikácie servera sa udeje práve po zavolaní tejto metódy. A to realizujeme práve pod definíciou tejto triedy. Implementáciu môžete vidieť v zdrojovom kóde ?.

import netifaces

import signal

import socket

import sys

import time

from foreignvehicles import ForeignVehicles

from gui import Gui

from objectmaker import ObjectMaker

from tkinter import Tk

from vehicle import Vehicle

class VehicleServer:

    # Uvedená IP ("") reprezentuje priradenú IP-čku z rozsahu

    # 169.254.0.1-254 (unicast) a 169.254.0.255 (broadcast).

    # To znamená, že server počúva na dvoch IP-čkách

    IP = ""

    PORT = 20000

    ADDR = (IP, PORT)

    NETWORK\_INTERFACE\_NAME = "bat0"

    BUFFER\_SIZE = 1024

    \_ip = None

    def \_\_init\_\_(self, new, vehicle, udpSock,

                 foreignVehicles, gui, addr = ADDR):

        self.\_addr = addr

        self.\_new = new

        self.\_vehicle = vehicle

        self.\_udpSock = udpSock

        self.\_foreignVehicles = foreignVehicles

        self.\_gui = gui

    def \_registerShutdownHandler(self):

        signal.signal(signal.SIGINT, self.stop)

        signal.signal(signal.SIGTERM, self.stop)

    def \_figureOutMyIp(self, netIfName, netifaces):

        bat0 = netifaces.ifaddresses(netIfName)

        self.\_ip = bat0[netifaces.AF\_INET][0]["addr"]

    def \_initUdpSock(self):

        self.\_udpSock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)

        self.\_udpSock.setblocking(False)

        self.\_udpSock.bind(self.\_addr)

    def \_udpSockReceiver(self):

        try:

            bufferSize = VehicleServer.BUFFER\_SIZE

            (json, (ip, port)) = self.\_udpSock.recvfrom(bufferSize)

            if ip == self.\_ip or ip == "127.0.0.1":

                if self.\_vehicle.fromJson(json, True) != None:

                    self.\_gui.updateMyVehicle()

                # Inak ignoruj nekompletné dáta o mne

            else:

                foreignVehicle = self.\_foreignVehicles.get(ip)

                if foreignVehicle == None:

                    foreignVehicle = self.\_new.foreignVehicle(json)

                    self.\_foreignVehicles.add(foreignVehicle)

                else:

                    foreignVehicle.fromJson(json)

        except BlockingIOError:

            pass

        self.\_foreignVehicles.removeAllUnreachable(time.time())

        self.\_gui.updateForeignVehicles()

        self.\_gui.invokeLater(self.\_udpSockReceiver)

    def start(self, netIfName = NETWORK\_INTERFACE\_NAME, nfs = netifaces):

        self.\_registerShutdownHandler()

        self.\_figureOutMyIp(netIfName, nfs)

        self.\_initUdpSock()

        self.\_gui.show(self.\_udpSockReceiver)

    def stop(self, sigNum = None, csf = None):

        try:

            self.\_gui.close()

        except Exception as e:

            print(e, file=sys.stderr)

        try:

            self.\_udpSock.close()

        except Exception as e:

            print(e, file=sys.stderr)

if \_\_name\_\_ == "\_\_main\_\_":

    new = ObjectMaker()

    vehicle = Vehicle()

    udpSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    tk = Tk()

    foreignVehicles = ForeignVehicles()

    gui = Gui(master=tk, vehicle=vehicle,

              foreignVehicles=foreignVehicles)

    server = VehicleServer(new, vehicle, udpSock,

                           foreignVehicles, gui)

    try:

        server.start()

    except KeyboardInterrupt:

        pass

    finally:

       server.stop()

Zdrojový kód vehicleserver/vehicleserver.py

### 3.2.11 Služby Systemd (priečinok lib-systemd-system)

Keď spustíme „holý“ minipočítač (t. j. bez displeja, myšky, klávesnice) vo vozidle, tak chceme, aby bez intervencie používateľa spároval OBD skener s minipočítačom a spustil aplikáciu klienta. V prípade, že je displej napojený, tak aby spustil aj aplikáciu servera. Na riešenie tohto problému je vhodný systém služieb Systemd, ktorý je súčasťou operačného systému Raspbian.

„Systemd obsluhuje jednotky, sleduje ich stav, vie ich reštartovať pri zlyhaní, logovať chyby. Umožňuje obmedzovať zdroje, ako dostupnú pamäť, využitie procesora a pod. Ďalej sleduje procesy pomocou cgroups namiesto klasického PID, vďaka čomu sa nemôžu vyviazať ani pri tzv. double-forking, a preto vždy vieme, ktoré procesy daná jednotka vlastní. Systemd tiež sprostredkováva komunikáciu medzi procesmi a umožňuje nastaviť spúšťanie jednotiek až vo chvíli, keď sú naozaj potrebné (tzv. on-demand loading) (Knížek, 2016).“

#### 3.2.11.1 Služba párovania OBD skenera s minipočítačom (súbor v2v-obd2rpi.service)

Prvou službou je služba zabezpečujúca párovanie OBD skenera s minipočítačom Raspberry Pi. Táto služba je automaticky spustená pri boote systému hneď po tom, ako skupina služieb multi-user.target úspešne beží (sekcia [Install]). Služba v2v-obd2rpi.service je silne závislá na skupine služieb multi-user.target, tzn. že ak skupina služieb multi-user.target nie je úspešne spustená, tak služba v2v-obd2rpi.service sa ani len nepokúsi spustiť (sekcia [Unit]). V sekcii [Service] definujeme správanie samotnej služby. V PIDFile= sme nastavili cestu k PID súboru, ktorý počas života služby v sebe uchováva číslo procesu, ku ktorému služba patrí. V ExecStart= uvádzame cestu ku skriptu, ktorý sa má vykonať pri spustení tejto služby, no ešte predtým toto spustenie oddialime o 10 sekúnd v nastavení ExecStartPre= (oproti absolútnemu štartu systému je to tiež 10 sekúnd). Štandardný výstup (stdin) a chybu (stderr) presmerujeme do logových súborov (StandardOutput= a StandardError=). V KillSignal= sme nastavili signál, ktorý sa má poslať procesu, pri jeho ukončení (SIGINT je obdoba Ctrl+C v linuxe). Toto je dôležité z pohľadu protokolu RFCOMM, ktorý používame na spárovanie, a ktorý sa v príkazovom riadku ukončuje práve kombináciou kláves Ctrl+C. Restart= hovorí o tom, že služba sa vždy musí pokúsiť opäť spustiť - reštartovať (skúša to donekonečna). RestartSec= určuje o koľko sekúnd sa má reštart služby oddialiť. Implementáciu môžete vidieť v zdrojovom kóde ?. Tu som chcel rozobrať aj nejaké scenáre?

[Unit]

Requires=multi-user.target

After=multi-user.target

[Service]

PIDFile=/run/v2v-obd2rpi.pid

ExecStartPre=/bin/sleep 10

ExecStart=/home/pi/Desktop/v2v/obd2rpi/obd2rpi

StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-obd2rpi.log

StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-obd2rpi.err

KillSignal=SIGINT

Restart=always

RestartSec=10s

[Install]

WantedBy=multi-user.target

Zdrojový kód Súbor lib-systemd-system/v2v-vehicleclient.service

#### 3.2.11.2 Služba aplikácie klienta (súbor v2v-vehicleclient.service)

Ďalšou službou je služba zabezpečujúca spustenie aplikácie klienta, ktorá je definovaná podobným spôsobom ako v2v-obd2rpi.service. Používa PID súbor, oddialenie štartu o 10 sekúnd (oproti absolútnemu štartu systému je to 20 sekúnd), presmerovaný štandardný výstup a chybu do logových súborov, nekonečne veľa pokusov na reštart služby ako aj 10 sekundovú pauzu medzi jednotlivými pokusmi o reštart. Hlavným rozdielom oproti v2v-obd2rpi.service je to, že táto služba je silne závislá na službe v2v-obd2rpi.service namiesto skupinovej služby multi-user.target. Pokiaľ služba v2v-obd2rpi.service úspešne nebeží (teda spojenie medzi OBD skenerom a minipočítačom bolo úspešne nadviazané), tak táto služba sa ani len nepokúsi o spustenie. Používa sa tu ešte silnejší stupeň závislosti BindTo=. Plní tú istú funkciu ako Requires=, avšak navyše hovorí o tem, že ak služba v2v-obd2rpi.service zlyhá, tak automaticky sa ukončí aj táto služba. Ako sme už spomínali predtým, tak hlavným súborom aplikácie je pre nás vehicleclient/vehicleclient.py. Spustenie aplikácie klienta sa realizuje v nastavení ExecStart=. Na spustenie používame interpreter jazyka Python3. Implementáciu môžete vidieť v zdrojovom kóde ?.

[Unit]

BindsTo=v2v-obd2rpi.service

After=v2v-obd2rpi.service

[Service]

PIDFile=/run/v2v-vehicleclient.pid

ExecStartPre=/bin/sleep 10

ExecStart=/usr/bin/python3 /home/pi/Desktop/v2v/vehicleclient/vehicleclient.py

StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleclient.log

StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleclient.err

Restart=always

RestartSec=10s

[Install]

WantedBy=v2v-obd2rpi.service

Zdrojový kód Súbor lib-systemd-system/v2v-vehicleclient.py

#### 3.2.11.3 Služba aplikácie servera (súbor v2v-vehicleserver.service)

Poslednou službou je služba aplikácie servera. Používa identickú definíciu ako služba v2v-vehicleclient.service vrátane silnej závilosti BindTo=. Hlavný súbor tu však predstavuje vehicleserver/vehicleserver.py. Rozdielom je aj to, že táto služba je silne závislá na v2v-vehicleclient.service a skupinovej službe graphical.target. Opäť, ak tieto služby nebežia, tak sa nespustí ani táto. A naopak, ak jedna z nich zlyhá, tak sa ukončí aj táto. Závislosť na skupinovú službu graphical.target tu používame preto, že hlavnou úlohou aplikácie servera je zobrazovať informácie vodičovi. Ak však zobrazovacia jednotka nie je k dispozícii, tak nemá význam túto službu ani len spúšťať. V definícii tejto služby sú navyše nastavenia Environment=. Slúžia práve na nastavenie grafického prostredia. Bez týchto nastavení, by sa nám samotná aplikácia nezobrazila na zobrazovacej jednotke. Implementáciu môžete vidieť v zdrojovom kóde ?.

[Unit]

BindTo=v2v-vehicleclient.service graphical.target

After=v2v-vehicleclient.service graphical.target

[Service]

Environment="DISPLAY=:0"

Environment="XAUTHORITY=/home/pi/.Xauthority"

PIDFile=/run/v2v-vehicleserver.pid

ExecStartPre=/bin/sleep 10

ExecStart=/usr/bin/python3 /home/pi/Desktop/v2v/vehicleserver/vehicleserver.py

StandardOutput=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleserver.log

StandardError=append:/home/pi/Desktop/v2v/lib-systemd-system/log/v2v-vehicleserver.err

Restart=always

RestartSec=10s

[Install]

WantedBy=v2v-vehicleclient.service graphical.target

Zdrojový kód Súbor lib-systemd-system/v2v-vehicleserver.service

### 3.2.12 Automatizácia inštalácie a konfigurácie prostredia pomocou skriptu (súbor install)

Problém s akým sme sa od začiatku vývoja stretli bolo skoordinovanie programovania s testovaním v aute. Keď sme niečo naprogramovali v Pythone, otestovať sa dalo len v aute. K dispozícií sme síce mali malý displej, ale písať niečo zmysluplnú na ňom neprichádzalo do úvahy. Na prípadné malé opravy kódu alebo pozeranie logov to však stačilo. Okrem toho nám vo vozidle chýbal internet a samotné bezdrôtové sieťové rozhranie bolo nakonfigurované už na použitie s BATMAN-ADV. Internet bol potrebný hlavne kvôli inštalovaniu rôznych balíčkov, či už do operačného systému alebo priamo do Python-u. Internet sme v minipočítačoch zabezpečovali cez LAN kábel. Celý proces programovania vyzeral asi nasledovne:

* vnútri v dome zapnúť minipočítač a pripojiť potrebné periférne zariadenia (myš, klávesnica, LAN kábel, HDMI kábel k veľkému monitoru),
* naprogramovať niečo zmysluplné, nainštalovať chýbajúce balíčky a vypnúť minipočítač,
* presunúť sa do auta, naštartovať auto, zapnúť minipočítač a pripojiť potrebné perfirérne zariadenia (myš, klávesnica, HDMI kábel k malému displeju),
* otestovať naprogramovanú funkcionalitu, pričom človek zistí, že to nefunguje, hľadať riešenie (či už na internete alebo v hlave), skúsiť ho opraviť priamo v aute ak sa dá, a tak podobne).

A toto sa dookola opakovalo. Takéto presúvania medzi autom a „kanceláriou,“ zapínanie a vypínanie minipočítača, štartovanie a vypínanie auta stálo veľa času a bolo to dosť otravné. Viackrát sme museli dobre premyslieť čo chceme spraviť a ako to spraviť, aby sme túto činnosť minimalizovali. Potom sa k tomu pridalo druhé auto. Nebolo nič nezvyčajné, že nám na ulici stáli dve naštartované vozidlá s jedným vodičom, no niekedy aj bez dozoru.

Keď už do toho prišlo aj to druhé vozidlo, tak na obidvoch vozidlách muselo byť spustené to isté, resp. musel tam bežať ten istý program. Po každej zmene mirror-ovať SD kartu nebolo riešenie, nastavenia by sa aj tak museli meniť. Na udržiavanie aktuálnosti oboch minipočítačov bol potrebný centrálny repozitár. Na tom nám poslúžil GitHub. Keď už bol program funkčný a úspešné otestovaný na dvoch autách, tak sme to chceli skúsiť aj na troch. Keby to chceme skúsiť aj na viac ako troch, tak sme potrebovali celý proces inštalácie automatizovať. Inštaláciu operačného systému Raspbian, jeho konfiguráciu a stiahnutie repozitára z GitHub-u síce musíme spraviť manuálne, ale všetko ostatné už vieme spraviť cez inštalačný skript. Jeho implementáciu môžete vidieť v zdrojovom kóde ?. Jedná sa o klasický bash skript. Súboru treba nastaviť aj oprávnenie pre exekúciu. Najskôr zmeníme aktuálny pracovný adresár na ten, kde máme celý projekt uložený. Nemôžme ho uložiť nikde inde, iba na plochu používateľa pi do priečinka v2v. Celý projekt uvažuje iba s touto lokáciou. Nainštalujeme balíčky potrebné pre projekt z pohľadu operačného systému a potom aj z pohľadu programovacieho jazyka Python. Nastavenie sieťového rozhrania spočíva v premenovaní súboru bat0.example na bat0, v ktorom musíme opraviť IP adresu, zmenení vlastníka na používateľa pi a vytvorení symbolických odkazov. Ďalej musíme registrovať služby. Tiež pre ne vytvoríme symbolické odkazy, vytvoríme adresár kam sa budú ukladať logy z jednotlivých služieb a zmeníme vlastníka na používateľa pi. Potom už len aktivujeme služby. Ďalším krokom je premenovať súbor vehicleinfo.txt.example na vehicleinfo.txt a zmeniť vlastníka na používateľa pi. V ňom musíme zmeniť už spomínané informácie. Posledný krok je aktivácia BATMAN-ADV. To spočíva v doplnení 3 rôznych príkazov do 3 rôznych súborov. Registrujeme BATMAN-ADV ako modul, sieťovému rozhraniu wlan0 zakážame DHCP, a príkaz batman (rozoberaný v kapitole ?) vykonáme pri spustení interaktívneho shell-u. Toto posledné riešenie asi nie je úplne ideálne, ale funguje. Lepšie by bolo tento príkaz vykonať po inicializácii sieťového modulu. Na záver sa ešte vypíše, čo je potrebné spraviť. To sme si ale priebežne napísali.

#!/bin/bash

cd /home/pi/Desktop/v2v

# Nainštalovať OS závislosti

apt-get update

apt-get install -y batctl bluez expect python3.7

# Nainštalovať Python závislosti

pip3 install netifaces https://github.com/brendan-w/python-OBD/archive/master.zip

# Sieťové rozhranie

cp etc-network-interfaces.d/bat0.example etc-network-interfaces.d/bat0

chown pi:pi etc-network-interfaces.d/bat0

ln -sf "$(pwd)/etc-network/interfaces"         /etc/network/interfaces

ln -sf "$(pwd)/etc-network-interfaces.d/bat0"  /etc/network/interfaces.d/bat0

ln -sf "$(pwd)/etc-network-interfaces.d/wlan0" /etc/network/interfaces.d/wlan0

# Služby

ln -sf "$(pwd)/lib-systemd-system/v2v-obd2rpi.service"       /lib/systemd/system/v2v-obd2rpi.service

ln -sf "$(pwd)/lib-systemd-system/v2v-vehicleclient.service" /lib/systemd/system/v2v-vehicleclient.service

ln -sf "$(pwd)/lib-systemd-system/v2v-vehicleserver.service" /lib/systemd/system/v2v-vehicleserver.service

mkdir -p lib-systemd-system/log

chown pi:pi lib-systemd-system/log

systemctl daemon-reload

systemctl enable v2v-obd2rpi.service

systemctl enable v2v-vehicleclient.service

systemctl enable v2v-vehicleserver.service

# vehicleinfo.txt

cp vehicleinfo.txt.example vehicleinfo.txt

chown pi:pi vehicleinfo.txt

# Aktivovať BATMAN-ADV

echo "" >> /etc/modules

echo "batman-adv" >> /etc/modules

echo "" >> /etc/modules

echo "" >> /etc/dhcpcd.conf

echo "denyinterfaces wlan0" >> /etc/dhcpcd.conf

echo "" >> /etc/dhcpcd.conf

echo "" >> /home/pi/.bashrc

echo "$(pwd)/batman/batman" >> /home/pi/.bashrc

echo "" >> /home/pi/.bashrc

# Čo ešte treba urobiť

echo ""

echo ""

echo ""

echo "Úlohy:"

echo "------"

echo "1. V súbore ./etc-network-interfaces.d/bat0 oprav IP adresu"

echo "2. V súbore ./vehicleinfo.txt oprav údaje o aute"

Zdrojový kód Súbor install

### 3.2.13 Kompletný zdrojový kód

Kompletný zdrojový kód máme uložený aj v našom GitHub repozitáre na URL adrese: https://github.com/david-bojnansky/diploma-v2v.

## 3.3 Praktické testovanie a zhodnotenie výsledkov

Systém sa nám podarilo doprogramovať a ostáva ho už len otestovať v praxi a získané výsledky následne stručne zhodnotíme.

### 3.3.1 Spustenie systému vo vozidle

Keď sa už nachádzame vo vozidle a máme všetko pripravené, tak si musíme položiť otázku, či už OBD skener máme spárovaný s minipočítačom Raspberry Pi alebo nie (určite ho nemáme pri absolútne prvom spustení minipočítača vo vozidle). Ak nie, tak k minipočítaču musíme pripojiť aj periférne zariadenia ako displej, klávesnicu a myš. Do OBD konektora zapojíme OBD skener a naštartujeme vozidlo. Minipočítač zapojíme do autonabíjačky. Keď sa načíta operačný systém, tak otvoríme príkazový riadok a spustíme príkaz /home/pi/Desktop/v2v/obd2rpi/pairwithobd (za prepokladu, že MAC adresu skenera a párovací PIN kód máme v súbore vehicleinfo.txt správne zadané). Párovací proces by sa mal zahájiť a úspešne ukončiť po cca 30 sekundách. Následne reštartujeme minipočítač. Do minúty od načítania operačného systému by sa malo zobraziť grafické rozhranie s priebežne aktualizujúcimi sa informáciami o našom vozidle. Ak sa tieto informácie priebežne aktualizujú, tak všetko je v poriadku a minipočítač môžeme vypnúť. Potom môžeme odpojiť všetky periférne zariadenia. Minipočítač si položíme na palubovú dosku. Odteraz už môžeme spúšťať minipočítač bez periférnych zariadení a do minúty by sa mal V2V systém spustiť a začať vysielať informácie o našom vozidle do okolia. Tento postup zopakujeme pre aspoň jedno ďalšie vozidlo. Pokiaľ chceme v niektorom vozidle sledovať informácie o našom vozidle a o vozidlách okolo nás v reálnom čase, tak k minipočítaču pripojíme displej. Po spustení minipočítača by malo nabehnúť grafické rozhranie s priebežne aktualizujúcimi sa informáciami o našom ako aj o ostatných vozidiel okolo nás.

### 3.3.2 Všeobecná funkčnosť systému

To čo sme si naplánovali v návrhu (kapitola ?), tak to sa nám aj podarilo zrealizovať. Výsledok je možné vidieť na obrázku ?. Informácie, ktoré sú zobrazené na obrázku, sú zachytené počas testovacej jazdy s dvomi ďalšími vozidlami. V hornej časti okna sa zobrazujú informácie o našom vozidle. V jeho ľavej časti sú IP adresa v mesh sieti, značka vozidla, model vozidla a evidenčné číslo vozidla. Ako sme už spomínali, tak tieto informácie nevieme získať z OBD a preto sú fixne definované v súbore. V strednej časti sa nachádza počet otáčok motora za minútu, zaradený prevodový stupeň, smer jazdy, rýchlosť vozidla a aktuálna činnosť vozidla. Z týchto informácií fungujú len 3, a to počet otáčok motora za minútu, rýchlosť vozidla a činnosť vozidla. Prvé dve sú získané priamo z OBD a činnosť vozidla je menená na základe rýchlosti vozidla. Naopak, nefunguje zaradený prevodový stupeň a smer jazdy. Smer jazdy sme chceli založiť na zaradenom prevodovom stupni. Mali sa zobrazovať hodnoty ako nikde, dopredu alebo dozadu. Python knižnica, ktorú sme použili komunikáciu s OBD, nemala definovaný príkaz pre získanie zaradeného prevodového stupňa. Preto sme vytvorili podľa oficiálnej dokumentácie tejto knižnice vlastný OBD príkaz, ktorý mal takúto informáciu z OBD získať. Žiaľ, neustále nám vracia hodnotu 0. Snažili sme sa dopátrať k možným riešeniam, ale nepodarilo sa nám to. Skúšali sme aj kvalitnejšie OBD skenery, avšak bez výraznejšieho úspechu. Dokonca sa nám v niektorých prípadoch nepodarilo spustiť ani systém. Je pravda, že sme nemali k dispozícií len vozidlá s manuálnou prevodovkou. Pri automatickej by bol výsledok možno iný. V spondnej časti okna sa zobrazuje tabuľka s informáciami o vozidlách okolo nás, ktoré sú v dosahu. Ak sa vozidlo stratí z dosahu, tak sa vozidlo z tabuľky vymaže. Informácie, ktoré o týchto vozidlách zobrazujeme, sú IP adresa v mesh sieti, značka vozidla, model vozidla, evidenčné číslo vozidla, smer jazdy, rýchlosť vozidla a aktuálna činnosť vozidla. So smerom jazdy tu máme ten istý problém. Je dôležité poznamenať, že žiadne informácie o cudzích vozidlách nespracúvame v našom vozidle, ale rovno ich tak prijímame. To znamená, že napríklad aktuálnu činnosť cuzdieho vozidla nevypočítavame, ale rovno ju tak príjmame. Pri testovaní sme odskúšali aj to, či v prípade, ak vytiahneme OBD skener z konektora (čím systém znefunknčníme) a potom ho zapojíme naspäť, sa systém sám reštartuje a spustí. Dospeli sme k záveru, že toto nám funguje. Platí to aj v prípade zavretia okna aplikácie servera. Týmto sme sa snažili docieliť zjednodušený kiosk mód. Celkovo môžeme teda konštatovať, že náš systém v praxi funguje. Priebeh testovania opísaných vlastností sme zachytili aj na niekoľko video záznamov:

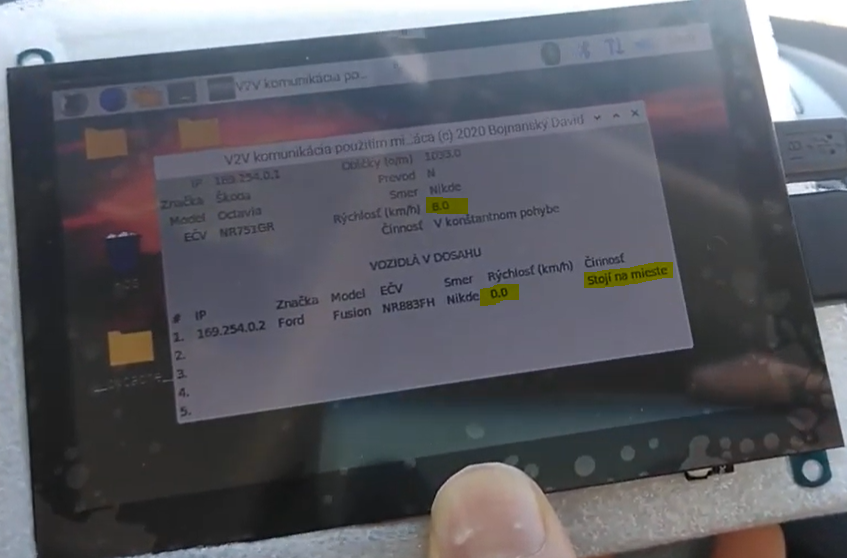
* Vozidlá idúce za sebou I. (https://youtu.be/kq7yeCqjYpg)
* V tomto videu spomíname, že vozidlo idúce priamo pred nami sa často stráca z nášho dosahu na rozdiel od toho pred ním. Po testovaní sme však zistili, že v týchto vozidlách boli nechtiac zamenené minipočítače. Z toho vyplýva, že vozidlo priamo pred nami sa menej často strácalo z dosahu ako to pred ním.
* Vozidlá idúce za sebou II. (https://youtu.be/ZFmHJJL8\_uo)
* Automatický reštart systému (https://youtu.be/rrhvNowX0fs)



Obrázok Zobrazované informácie a ich funkčnosť

### 3.3.3 Dosah signálu

Pri testovacej jazde, kde sme šli 3 vozidlá za sebou s odstupom približne do 10 metrov, sa stávalo, že vozidlá pred nami sa nám na chvíľu stratili z dosahu. V niektorých prípadoch však spojenie fungovalo aj na viac ako 10 metrov. Všetky 3 vozidlá mali minipočítač položený na palubnej doske. Myslíme si však, že ak by sme minipočítače umiestnili napríklad na strechu vozidiel, tak by sme dostali stabilnejší signál a aj väčší dosah. Nemali sme však dostatočne dlhé napájacie káble. Spravili sme však aj test, kde jedno vozidlo stálo na mieste a druhé sa pomaly pohybovalo smerom od neho. Pri tomto teste sme zistili, že signál bol stabilný približne do 50 metrov. Pri viac ako 50 metrov sa spojenie už nedokázalo nadviazať. Jedenkrát sa stalo, že spojenie sa na chvíľu prerušilo a opäť sa nadviazalo. Stalo sa to približne po 20 metroch. Toto testovanie máme zdokumentované vo videu „Dosah signálu“ (https://youtu.be/NyofHGUne5c).



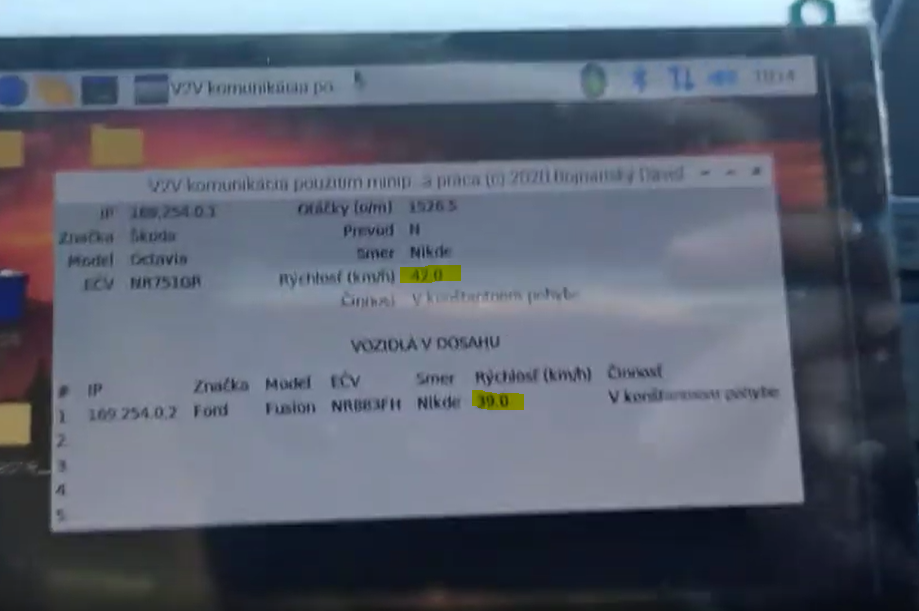
Obrázok Testovanie dosahu signálu

### 3.3.4 Míňajúce sa vozidlá

Otestovali sme aj prípad, keď dve vozidlá idú oproti sebe, minú sa a pokračujú ďalej v smere svojej jazdy. Týmto testom sme chceli hlavne zistiť, či tieto vozidlá dokážu medzi sebou nadviazať spojenie a vymeniť si informácie. Testovali sme to pri rýchlosti 40 km/h oboch vozidiel. V bode keď prechádzali okolo seba bola teda celková rýchlosť 80 km/h. Výsledkom tohto testu je, že vozidlá dokázali medzi sebou nadviazať spojenie a vymeniť si informácie. Tento test máme zdokumentovaný aj vo videu „Vozidlá, kt. sa míňajú“ (https://youtu.be/ac9dYk9JT6I).



Obrázok Testovanie nadviazania spojenia míňajúcich sa vozidiel



Obrázok Celková rýchlosť míňajúcich sa vozidiel je 80 km/h

# Záver

V2V systém je postavený na skutočne silnej myšlienke. Avšak jeho realizácia je oveľa zložitejšia, než si väčšina ľudí uvedomuje. Zahŕňa mnoho technologických aspektov, ale aj mnoho vládnych regulačných orgánov a spoluprácu výrobcov vozidiel. Ako sme v prvej kapitole uviedli, tak hlavná myšlienka V2V systému bola rozšírená o komunikáciu so všetkým okolo nás (V2X), kde patrí dopravná infraštruktúra, chodci, internetová sieť a podobne.

Hlavným cieľom našej diplomovej práce bolo vytvoriť funkčný V2V systém, otestovať ho v praxi a vyhodnotiť dosiahnuté výsledky. Dovolíme si tvrdiť, že to sa nám podarilo. Podarilo sa nám na začiatku navrhnúť V2V systém po hardvérovej aj softvérovej stránke, a krok za krokom sme tento návrh realizovali. Hoci sme narazili na menšie problémy, tie sa nám podarilo vyriešiť. Vytvorený V2V systém sme následne otestovali a získané výsledky sme vyhodnotili. Zjednodušene môžeme povedať, že systém nám funguje okrem jednej časti. Tá, ale nie je kritická pre funkčnosť systému ako takého. Konkrétne sa nám cez rozhranie OBD-II nepodarilo získať zaradený prevodový stupeň a preto sme nemohli jednoznačne určiť smer jazdy. Nie sme si úplne istý, ale táto informácie je dostupná zrejme iba vo vozidlách s automatickou prevodovkou a my sme mali k dispozícii len s manuálnou.

Na záver ešte pridávame tipy ako by sa dal náš systém vylepšiť pre zlepšenie zážitku vodiča z jazdy:

* GPS – pomocou neho by sme vedeli určiť smer jazdy vozidla, zistiť ako ďaleko je od nás iné vozidlo a podobne,
* pípací zvuk signalizujúci brzdenie vozidla pred nami (vozidlo pred nami by sa určovalo na základe porovnania našich a jeho GPS súradníc),
* automatizácia počiatočného spárovania s OBD skenerom pri spustení minipočítača (nemyslíme automatizáciu nadviazania spojenia s OBD skenerom, pretože to máme spravené).

# Zoznam bibliografických odkazov

KARPAT, M. 2007. Komunikácia medzi autami - vyskúšali sme technológiu budúcnosti. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://auto.sme.sk/c/3469935/komunikacia-medzi-autami-vyskusali-sme-technologiu-buducnosti.html>.

THEPIHUT.COM. 2016. Raspberry Pi 3 Model B. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://thepihut.com/products/raspberry-pi-3-model-b>.

THEPIHUT.COM. 2019. Raspberry Pi 4 Model B. [online]. [cit. 2019-11-30]. Dostupné na internete: <https://thepihut.com/products/raspberry-pi-4-model-b>.

NHTSA.GOV. 2019. About NHTSA. [online]. [cit. 2019-12-01]. Dostupné na internete: < https://www.nhtsa.gov/about-nhtsa>.

NHTSA.GOV. 2019. Vehicle-to-Vehicle Communication. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication>.

HALL-GEISLER, K. 2017. All new cars could have V2V tech by 2023. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://techcrunch.com/2017/02/02/all-new-cars-could-have-v2v-tech-by-2023/>.

STANDARDS.IEEE.ORG. 2010. IEEE 802.11p-2010 - IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. [online]. [cit. 2019-12-01]. Dostupné na internete: <https://standards.ieee.org/standard/802\_11p-2010.html>.

DORMEHL, L a EDELSTEIN, S. 2019. Sit back, relax, and enjoy a ride through the history of self-driving cars. [online]. [cit. 2019-12-06]. Dostupné na internete: <https://www.digitaltrends.com/cars/history-of-self-driving-cars-milestones/>.

ARENA, F. a PAU, G. 2019. An Overview of Vehicular Communications. [online]. [cit. 2019-12-06]. Dostupné na internete: <https://www.mdpi.com/1999-5903/11/2/27/pdf>.

MISHRA, S. a iní. 2015. System on Chip Interfaces for Low Power Design. 406 s. ISBN 978-0128017906.

OBDII.COM. 2011. OBD-II Background. [online]. [cit. 2019-12-06]. Dostupné na internete: <http://www.obdii.com/background.html>.

TECHOPEDIA.COM. 2019. What is ZigBee?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.techopedia.com/definition/4390/zigbee>.

ELECTRONICS-NOTES.COM. 2019. What is ZigBee Technology. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.electronics-notes.com/articles/connectivity/zigbee/what-is-zigbee-technology-tutorial.php>.

TECHOPEDIA.COM. 2019. What is Bluetooth?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.techopedia.com/definition/26198/bluetooth>.

10NAJS.SK. 2018. Čo je to Bluetooth? Aký má dosah? Bluetooth 5 vs Bluetooth 4. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://10najs.sk/co-je-to-bluetooth/>.

RASPBERRYPI.ORG. 2019. What is Raspberry Pi?. [online]. [cit. 2019-12-07]. Dostupné na internete: <https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>.

JIANG, D. a DELGROSSI, L. 2008. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. [online]. [cit. 2019-12-09]. Dostupné na internete: <https://www.researchgate.net/publication/224314431\_IEEE\_80211p\_Towards\_an\_International\_Standard\_for\_Wireless\_Access\_in\_Vehicular\_Environments>.

BEAL, V. 2019. 802.11 IEEE wireless LAN standards. [online]. [cit. 2019-12-09]. Dostupné na internete: <https://www.webopedia.com/TERM/8/802\_11.html>.

BLANK, E. 2016. Z Wave Vs ZigBee: Which Is Better For Your Smart Home?. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://thesmartcave.com/z-wave-vs-zigbee-home-automation/>.

ABDELGADER, A. a LENAN, W. 2014. The Physical Layer of the IEEE 802.11p WAVE Communication Standard: The Specifications and Challenges. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://www.researchgate.net/publication/279474688\_The\_Physical\_Layer\_of\_the\_IEEE\_80211\_p\_WAVE\_Communication\_Standard\_The\_Specifications\_and\_Challenges>.

VAMOSI, R. 2017. New V2V communication could give hackers a free ride. [online]. [cit. 2019-12-26]. Dostupné na internete: <https://www.synopsys.com/blogs/software-security/v2v-communication-hackers/>.

GRAHAM, K. 2017. Vehicle-to-vehicle communication tech going nowhere. [online]. [cit. 2019-12-28]. Dostupné na internete: <http://www.digitaljournal.com/tech-and-science/technology/vehicle-to-vehicle-communication-tech-going-nowhere/article/503010>.

KOON, J. 2019. Will Vehicle-to-Vehicle Communication Ever Take Off?. [online]. [cit. 2019-12-28]. Dostupné na internete: <https://new.engineering.com/story/will-vehicle-to-vehicle-communication-ever-take-off>.

JAIN, G. 2018. Talking Cars: A Survey of Protocols for Connected Vehicle Communication. [online]. [cit. 2019-12-28]. Dostupné na internete: <https://wiprodigital.com/2018/06/20/talking-cars-a-survey-of-protocols-for-connected-vehicle-communication/>.

OBDSOL.COM. 2020. OBD MODES. [online]. [cit. 2020-02-06]. Dostupné na internete: <https://www.obdsol.com/knowledgebase/obd-software-development/obd-services-modes/>.

KNÍŽEK, J. 2016. Nebojte se systemd: co to je a co umí?. [online]. [cit. 2020-02-19]. Dostupné na internete: <https://www.root.cz/clanky/nebojte-se-systemd-co-to-je-a-co-umi/>.

EBRAHIM, M. 2017. Expect Command And How To Automate Shell Scripts Like Magic. [online]. [cit. 2020-02-20]. Dostupné na internete: <https://likegeeks.com/expect-command/>.

KERNEL.ORG. 2020. batman-adv – The Linux Kernel documentation. [online]. [cit. 2020-02-28]. Dostupné na internete: <https://www.kernel.org/doc/html/v4.15/networking/batman-adv.html>.

# Zoznam príloh

Príloha A – CD nosič

* Príloha A.1 – Zdrojový kód
* Príloha A.2 – Diplomová práca vo formáte PDF