**UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE**

FAKULTA PRÍRODNÝCH VIED

V2V komunikácia použitím minipočítača Raspberry Pi

DIPLOMOVÁ práca

**Nitra 2020 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 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. 51 s.

Diplomová práca sa venuje problematike programovacieho jazyka Lua a tvorbe vzorových aplikácií. Bakalárska práca je rozdelená na teoretickú a praktickú časť. Teoretická časť je zameraná na analýzu súčasného stavu a je rozdelená na 3 časti. Prvá časť sa zaoberá programovacími paradigmami, ktoré programovací jazyk Lua využíva. Druhá časť sa zaoberá charakteristikou programovacieho jazyka Lua. Tretia časť poskytuje praktický pohľad na programovací jazyk Lua. Praktická časť je zameraná na analýzu vlastností, konceptov a postupov typických pre programovanie v programovacom jazyku Lua. Túto analýzu dopĺňajú vzorové aplikácie, ktoré názorne demonštrujú programovací jazyk Lua v praxi. Vo vzorových aplikáciách je dôraz kladený hlavne na ich komplexnosť a jednoduchosť. Bakalárska práca sa tiež zaoberá vložením do hostiteľskej aplikácie, čo názorne demonštrujeme na programovacom jazyku PHP a obchodnej platforme *FXCM Trading Station*.

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. 51 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

[Úvod 9](#_Toc448063776)

[1 Analýza súčasného stavu programovacieho jazyka Lua 10](#_Toc448063777)

[1.1 Lua a Programovacie paradigmy 10](#_Toc448063778)

[1.1.1 Imperatívna paradigma 11](#_Toc448063779)

[1.1.2 Procedurálna paradigma 11](#_Toc448063780)

[1.1.3 Funkcionálna paradigma 11](#_Toc448063781)

[1.1.4 Objektovo orientovaná paradigma 12](#_Toc448063782)

[1.2 Charakteristika programovacieho jazyka Lua 13](#_Toc448063783)

[1.2.1 Rozšíriteľnosť 14](#_Toc448063784)

[1.2.2 Jednoduchosť a veľkosť 14](#_Toc448063785)

[1.2.3 Rýchlosť 14](#_Toc448063786)

[1.2.4 Prenositeľnosť 15](#_Toc448063787)

[1.2.5 Vložiteľnosť 15](#_Toc448063788)

[1.3 Lua v praxi 16](#_Toc448063789)

[1.3.1 Aplikácie 16](#_Toc448063790)

[1.3.2 Moduly 17](#_Toc448063791)

[1.3.3 Online zdroje 17](#_Toc448063792)

[1.3.4 Odborná literatúra 18](#_Toc448063793)

[1.3.5 Vývojové prostredia 18](#_Toc448063794)

[2 Ciele bakalárskej práce 19](#_Toc448063795)

[3 Analýza programovacieho jazyka Lua 20](#_Toc448063796)

[3.1 Základy 20](#_Toc448063797)

[3.1.1 Typy hodnôt 21](#_Toc448063798)

[3.1.2 Operátory 22](#_Toc448063799)

[3.1.3 Riadiace konštrukcie 23](#_Toc448063800)

[3.2 Dátové štruktúry 24](#_Toc448063801)

[3.2.1 Tabuľky 24](#_Toc448063802)

[3.2.2 Polia 25](#_Toc448063803)

[3.2.3 Mapy 25](#_Toc448063804)

[3.2.4 Sety 25](#_Toc448063805)

[3.2.5 Dvojkoncové fronty 26](#_Toc448063806)

[3.2.6 Spájané zoznamy 26](#_Toc448063807)

[Vzorový príklad pr3\_2 26](#_Toc448063808)

[3.3 Funkcie 27](#_Toc448063809)

[3.3.1 Definovanie funkcie 27](#_Toc448063810)

[3.3.2 Volanie funkcie 28](#_Toc448063811)

[3.3.3 Pomenované argumenty 28](#_Toc448063812)

[3.3.4 Variabilný počet argumentov 28](#_Toc448063813)

[3.3.5 Viacnásobný výsledok 29](#_Toc448063814)

[3.3.6 Anonymné funkcie 30](#_Toc448063815)

[Vzorový príklad pr3\_3 30](#_Toc448063816)

[3.4 Korutiny 30](#_Toc448063817)

[Vzorový príklad pr3\_4 32](#_Toc448063818)

[3.5 Chyby 32](#_Toc448063819)

[3.5.1 Ošetrenie chýb 34](#_Toc448063820)

[Vzorový príklad pr3\_5 34](#_Toc448063821)

[3.6 Metatabuľky a metametódy 34](#_Toc448063822)

[3.6.1 Aritmetické metametódy 36](#_Toc448063823)

[3.6.2 Bitové metametódy 36](#_Toc448063824)

[3.6.3 Porovnávacie metametódy 37](#_Toc448063825)

[3.6.4 Tabuľkové metametódy 37](#_Toc448063826)

[3.6.5 Ostatné metametódy 38](#_Toc448063827)

[Vzorový príklad pr3\_6 39](#_Toc448063828)

[3.7 Objektovo orientované programovanie 39](#_Toc448063829)

[3.7.1 Návrh triedy 39](#_Toc448063830)

[3.7.2 Návrh odvodenej triedy 41](#_Toc448063831)

[3.7.3 Návrh triedy so súkromnými atribútmi 42](#_Toc448063832)

[Vzorový príklad pr3\_7 42](#_Toc448063833)

[3.8 Moduly 43](#_Toc448063834)

[3.8.1 Submoduly 44](#_Toc448063835)

[3.8.2 Načítanie modulu 44](#_Toc448063836)

[Vzorový príklad pr3\_8 45](#_Toc448063837)

[3.9 Vloženie do hostiteľskej aplikácie 45](#_Toc448063838)

[3.9.1 Vloženie cez programovací jazyk C 46](#_Toc448063839)

[3.9.2 Vloženie cez programovací jazyk Java 46](#_Toc448063840)

[3.9.3 Vloženie cez programovací jazyk PHP 46](#_Toc448063841)

[Vzorový príklad pr3\_9/php: PHP aplikácia volajúca Lua skript 47](#_Toc448063842)

[Vzorový príklad pr3\_9/fxcm: Indikátor predaja a nákupu 47](#_Toc448063843)

[Záver 48](#_Toc448063844)

[Zoznam bibliografických odkazov 49](#_Toc448063845)

[Zoznam príloh 51](#_Toc448063846)

# Zoznam ilustrácií a tabuliek

Obrázok 1 Blížiace sa záchranné vozidlo 11

Obrázok 2 Čiastočná a plná Mesh topológia 14

Obrázok 3 Minipočítač Raspberry Pi 4 15

# Úvod

Programovací jazyk Lua je v súčasnosti celosvetovo rozšírený a uznávaný. Veľké uplatnenie si našiel predovšetkým v hernom priemysle. Väčšinu učebných materiálov nájdeme v anglickom jazyku, čo pre niektorých ľudí môže ešte stále predstavovať prekážku. Výber bakalárskej práce bol založený na snahe oboznámiť sa, pre nás dovtedy s neznámym programovacím jazykom, a tak rozšíriť naše vedomosti o jeden ďalší.

Cieľom diplomovej práce je preskúmať a popísať vlastnosti, koncepty a postupy zaužívané v programovacom jazyku Lua. Druhým cieľom je, na základe získaných vedomostí, vytvoriť kolekciu vzorových aplikácií, ktoré tieto vlastnosti, koncepty a postupy demonštrujú v praxi.

V diplomovej práci sa opierame hlavne o informácie z knihy *Programming in Lua*, ktorej autorom je samotný vynálezca programovacieho jazyka Lua, a oficiálnu webovú stránku *www.lua.org*.

V prvej kapitole sa zaoberáme analýzou súčasného stavu V2V komunikácie. V rámci tejto kapitoly rozoberáme nedávnu históriu z roku 2007, kde boli prvé náznaky na takúto komunikáciu, ...

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

V tretej kapitole sa venujeme jednotlivým aspektom programovacieho jazyka Lua. V rámci nej rozoberáme najskôr základné aspekty jazyka ako úvod do neho, dátové štruktúry a funkcie. Následne sa presúvame k pokročilejším aspektom jazyka ako korutiny, chyby, metatabuľky a metametódy, objektovo orientované programovanie a moduly. Na záver sa zaoberáme možnosťami jeho vloženia do hostiteľskej aplikácie. Aby sme jednotlivé aspekty demonštrovali v praxi, vytvorili sme kolekciu vzorových aplikácií.

# 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:

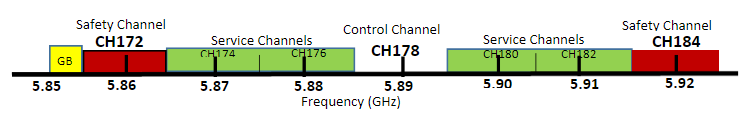
* 802.11 – poskytuje prenosovú rýchlosť 1 až 2 Mb/s na frekvencii 2.4 GHz podľa toho, či používa metódu vysielania rádiového signálu Frequency Hopping Spread Spectrum (FHSS) alebo Direct Sequence Spread Spectrum (DSSS),
* 802.11a – vylepšená 802.11, ktorá poskytuje prenosovú rýchlosť až do 54 Mb/s na 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 bakalárskej práce

V diplomovej práci chceme predstaviť programovací jazyk Lua vývojárom softvéru, ktorí sa s týmto skriptovacím jazykom ešte nestretli, ale majú skúsenosti s inými vyššími programovacími jazykmi, prípadne so skriptovacími jazykmi pri tvorbe webových stránok (napr. Java, C, C++, PHP, JavaScript).

Hlavným cieľom bakalárskej práce je:

* preskúmať a popísať vlastnosti, koncepty a postupy typické pre programovanie v programovacom jazyku Lua,
* na základe získaných vedomostí a vlastných praktických skúseností vytvoriť kolekciu vzorových aplikácií, ktoré budú názorne demonštrovať použitie programovacieho jazyka Lua v praxi,
* poukázať na význam programovacieho jazyka Lua pre profesionálnu prax a sformulovať odporúčania pre záujemcov o použitie programovacieho jazyka Lua vo vlastných aplikáciách.

# 3 Návrh, realizácia a testovanie V2V systému

V rámci tejto kapitoly

## 3.1 Návrh

Každý kus zdrojového kódu, ktorý interpreter programovacieho jazyka Lua. Pri programovací využívame princíp Dependency Injection.

## 3.2 Realizácia

Každý kus zdrojového kódu, ktorý interpreter programovacieho jazyka Lua... Na programovanie využijeme Thonny IDE.

### 3.2.1 Inštalácia OS Raspbian

V programovacom

### 3.2.1 Inštalácia Bluez

V programovacom

### 3.2.1 Párovanie OBD s minipočítačom (súbor obd2rpi/pairwithobd)

V programovacom...

### 3.2.1 Príprava vývojového prostredia

V programovacom

### 3.2.1 Inštalácia a konfigurácia BATMAN-ADV

V programovacom

### 3.2.1 Inštalácia programovacieho jazyka Python a balíčkov

Netifaces, obd

### 3.2.1 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 skenera, bluetooth párovací kód, značka vozidla, model vozidla a nakoniec evidenčné číslo vozidla (EČV). Tieto informácie nevieme získať cez OBD, 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 ďalšom a riadku program nezaujíma. My sme si tam uložili bluetooth MAC adresy všetkých našich OBD skenerov. Príklad súboru vehicleinfo.txt môžete vidieť v zdrojovom kóde ?.

00:00:00:33:33:33

1234

Škoda

Octavia

NR751GR

Zdrojový kód Súbor vehicleinfo.txt

### 3.2.1 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).

#### 3.2.1.1 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

#### 3.2.1.1 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

#### 3.2.1.1 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

#### 3.2.1.1 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

### 3.2.1 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).

#### 3.2.1.1 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

#### 3.2.1.1 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

#### 3.2.1.1 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:

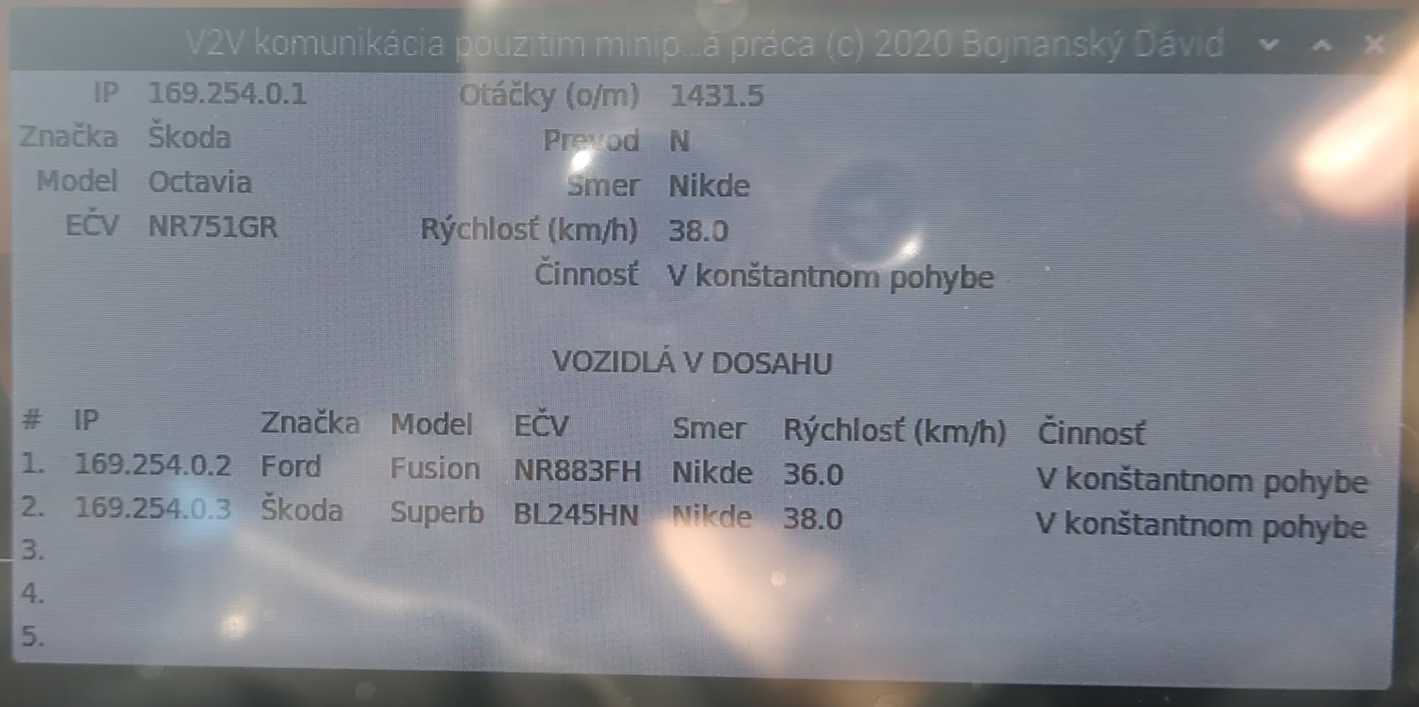
    def foreignVehicle(self, json):

        return Vehicle.fromJson(None, json)

Zdrojový kód Súbor vehicleserver/objectmaker.py

#### 3.2.1.1 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

#### 3.2.1.1 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

#### 3.2.1 Služby systemd (priečinok lib-systemd-system)

V programovacom

#### 3.2.1.1 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

#### 3.2.1.1 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

#### 3.2.1.1 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

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

V programovacom

## 3.3 Praktické testovanie a výstup

vykoná, sa nazýva *chunk*. Jednotlivé príkazy, podľa konvencie, by mali byť umiestnené

#### 3.2.1 Spustenie systému vo vozidle

V programovacom

#### 3.2.1 Funknčnosť

Problém, ktorý sa nám následne objavil a nepodarilo sa nám ho vyriešiť je, že tento príkaz (GEAR) stále vráti hodnotu 0. Skúšali sme aj kvalitnejšie OBD skenery, avšak bez výraznejšieho úspechu – aplikácia sa nedokázala spustiť.

#### 3.2.1 Dosah

Pri položení na palubovke a vystrčené von cez okno

# Záver

V2V is a noble idea. But the implementation is far more complicated than most people realize. It involves many aspects including technology, wireless spectrum, many governmental regulatory bodies, cooperation of automakers and adaptation of the technology as a whole. On the other hand, as IoT becomes more prevalent and applicable across many sectors, the development and implementation of V2V systems stands to benefit. Sensors, data storage, and analytics will become increasingly cost-effective, ensuring the widespread prevalence and network effects of V2V systems. Another factor is the infrastructure including traffic lights, freeway structure etc. Vehicle-to-infrastructure, or the increased connectivity between vehicles as well as between vehicles and components of roadway infrastructure, is the likely next step after V2V. However, since the roadway infrastructure is not uniform across the board in the U.S., V2I may face even more challenges than V2V. All told, V2V will continue to evolve slowly. The year may be 2030 before we see vehicles equipped with V2V systems capable of detecting the sudden stop of other vehicles in front. But the full implementation of V2V including V2I may be many years away.

Čo by sa ešte dalo spraviť:

GPS, pípanie ak niekto brzdí...

# 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/>.

# Zoznam príloh

Príloha A – Vzorový zdrojový kód

Príloha B

* Príloha B.1 – PHP aplikácia volajúca Lua skript
* Príloha B.2 – Cenový graf bez indikátora
* Príloha B.3 – Voliteľné parametre indikátora
* Príloha B.4 – Cenový graf s indikátorom

Príloha C – CD nosič

* Príloha C.1 – Zdrojový kód vzorových aplikácií
* Príloha C.2 – Bakalárska práca vo formáte PDF

Príloha A

**local** mod = {}

-- číta nepárne riadky básničky zo súboru poem-odd-rows.txt

**local** **function** readOddRows(dirname, i)

**for** line **in** **io**.lines(dirname .. '/poem-odd-rows.txt') **do**

**local** dot = i < 10 **and** ". " **or** ". "

i = **coroutine**.yield(i .. dot .. line)

**end**

**return** " KONIEC SÚBORU poem-odd-rows.txt"

**end**

-- číta párne riadky básničky zo súboru poem-even-rows.txt

**local** **function** readEvenRows(dirname, i)

**for** line **in** **io**.lines(dirname .. '/poem-even-rows.txt') **do**

**local** dot = i < 10 **and** ". " **or** ". "

i = **coroutine**.yield(i .. dot .. line)

**end**

**return** " KONIEC SÚBORU poem-even-rows.txt"

**end**

**function** mod.readPoemFromTwoFiles(dirname)

**local** oddRowsCo = **coroutine**.create(readOddRows)

**local** evenRowsResume = **coroutine**.wrap(readEvenRows)

**local** i = 0

**local** isEvenRowsCoDead = **false**

**repeat**

**io**.read() -- ENTER

**if** **coroutine**.status(oddRowsCo) == 'suspended' **then**

**local** ok, str

i = i + 1

**if** i == 1 **then**

ok, str = **coroutine**.resume(oddRowsCo, dirname, i)

**else**

ok, str = **coroutine**.resume(oddRowsCo, i)

**end**

**if** ok **then** **print**(str) **else** **print**(" CHYBA: " .. str) **end**

**end**

**io**.read() -- ENTER;

**do**

i = i + 1

**local** ok = **pcall**(**function** ()

**if** i == 2 **then**

**print**(evenRowsResume(dirname, i))

**else**

**print**(evenRowsResume(i))

**end**

**end**)

isEvenRowsCoDead = **not** ok

**end**

**until** **coroutine**.status(oddRowsCo) == 'dead' **and** isEvenRowsCoDead

**end**

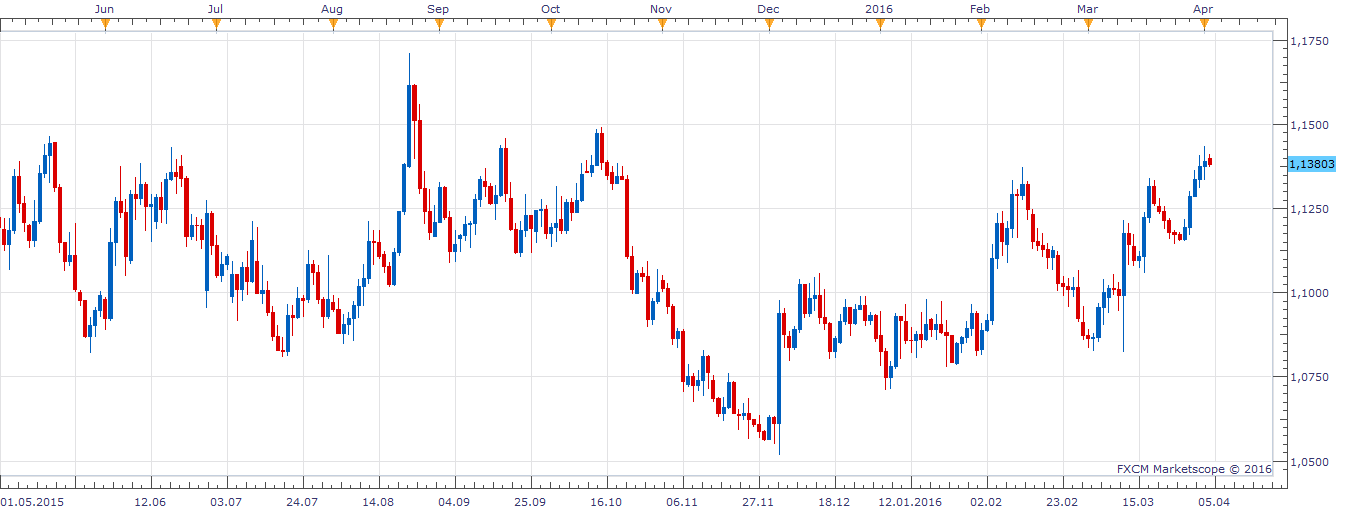
**return** mod

Príloha B

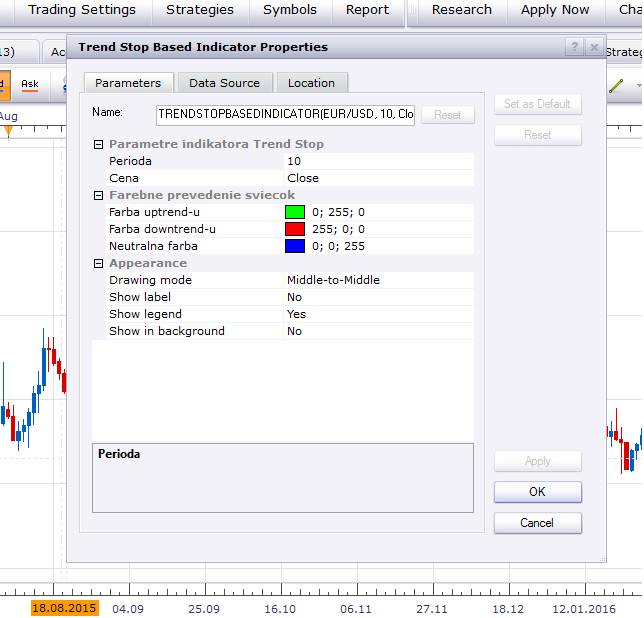
**Príloha B.1: PHP aplikácia volajúca Lua skript**



**Príloha B.2: Cenový graf bez indikátora**



**Príloha B.3: Voliteľné parametre indikátora**



**Príloha B.4: Cenový graf s indikátorom**

****