Framework Evaluation for Backendbased Functions in the Context of Connected Vehicles

Tim Häberlein, Dr.-Ing. Oliver Manicke, Prof. Dr.-Ing. Bernard Bäker

Professur für Fahrzeugmechatronik, IAD - Chair of Vehicle Mechatronics
Technische Universität Dresden, George-Bähr-Str. 1b,
01062 Dresden
tim.haeberlein@tu-dresden.de

Abstract: Insbesondere durch die Digitalisierung des Automobils und die Weiterentwicklung der Informations- und Kommunikationstechnologien (IKT) ergibt sich die Möglichkeit, Fahrzeugfunktionen mit Datenaustausch außerhalb des Fahrzeugs zu realisieren. Diese sogenannten Connected Vehicles und die damit verbundene geobasierte Datenverarbeitung zur ganzheitlichen und bedarfsgerechten Funktionssteuerung zwischen zentralen Recheneinheiten und Fahrzeugen sind ein aktueller Forschungs- und Entwicklungsschwerpunkt. Im Rahmen dieses Artikels wird ein Basissystem zur Realisierung solcher Funktionen vorgestellt, deren Vor- und Nachteile diskutiert und eimzelne Teilsysteme bewertet.

1 Einleitung und Motivation

Durch die rasante Weiterentwicklung der Informations- und Kommunikationstechnologien (IKT) und vor dem Hintergrund der Digitalisierung des Automobils sowie der immer effizienteren Halbleiterindustrie ergeben sich viele neue Anwendungsmöglichkeiten im Fahrzeug. So wird sich die E/E-Architektur im Hinblick auf die Fahrzeugautomatisierung stark wandeln. Der steigende Ressourcenbedarf durch Algorithmen, welche die Umfelderfassung und Fahrzeugautomatisierung ermöglichen, wird nicht mehr nur durch einzelne ECUs gedeckt werden können, sondern muss durch übergeordnete Zentralrechner ergänzt werden. Die anwendungsbezogene Notwendigkeit zur Integration von neuen Rechentechnologien führt dazu, dass die Elektronik in Fahrzeugen nicht mehr durch simple Mikrokontroller, sondern durch erweiterte System-on-a-Chip (SoC - z. B. nVidia Drive oder MobileEye EyeQ5) bestimmt wird. Das Fahrzeug wird zu einem Rechenzentrum auf Rädern. [1]

Neben der elektronischen Erweiterung des Fahrzeugs spielt auch die Vernetzung mit der Umwelt eine entscheidende Rolle. So werden IKT V2V (Vehicle to Vehicle) und V2X (Vehicle to Everything) Dienste auch als Treiber des automatisierten Fahrens gesehen und benötigen zusätzliche Hard- und Software im Fahrzeug. Neben diesen Diensten führte auch die Verordnung zu eCall-Systemen zur Anbindung von Fahrzeugen an externe Systeme. Der Allgemeine Trend von Cloud-Systemen hält somit

ebenfalls Einzug in Fahrzeuge und wird gleichzeitig zu einem Forschungsgegenstand. Das einzelne Fahrzeug wird zu einem weiteren Sensor in einer gesamten Fahrzeugflotte und mittels sogenannter V2C2V (Vehicle to Cloud to Vehicle) Strukturen können Informationen zurück ins Fahrzeug gelangen und verarbeitet werden. [2]–[5]

Der beschriebene Paradigmenwechsel ermöglicht mittels V2C2V Fahrzeugfunktionen mit einem Datenaustausch außerhalb des Fahrzeugs zu realisieren. Durch die Nutzung fahrzeugexterner Ressourcen kann damit dem Trend einer elektronischen Dilatation des Fahrzeugs entgegengewirkt werden. Diese sich dabei ergebenden backendbasierten Funktionen (backendbased functions) für vernetzte Fahrzeuge (Connected Vehicles) mittels geobasierter Datenverarbeitung zur ganzheitlicher und bedarfsgerechter Funktionssteuerung zwischen externen Recheneinheiten und Fahrzeugen sind ein aktueller Forschungsgegenstand. Im Rahmen dieses Artikels wird ein grundlegendes Framework zur Realisierung dieser Funktionen vorgestellt, deren Vor- und Nachteile diskutiert und einige Teilsysteme in ihrer Umsetzung bewertet. Die Bewertung wird dabei im Besonderen auf die Möglichkeit eingehen, geobasierte vernetzte Funktionen in ihrer Verfügbarkeit zu untersuchen aber auch generisch zu testen und zu prüfen.

2 Grundlegende Systembetrachtung eines V2C2V-Frameworks

Auf Basis vorangegangener Betrachtungen [6], [7] wurde ein V2C2V-Framework (vgl. Abbildung 1) entwickelt. Dieses Framework ermöglicht den Datenaustausch zwischen Fahrzeugen und fahrzeugexternen Recheneinheiten und eine Aktuatoransteuerung über fahrzeugexterne Funktionsanteile. Im Folgenden sollen deren Hauptbestandteile (Fahrzeugkomponente und Cloudkomponente) vorgestellt sowie die verwendeten Begriffe eingeordnet werden.

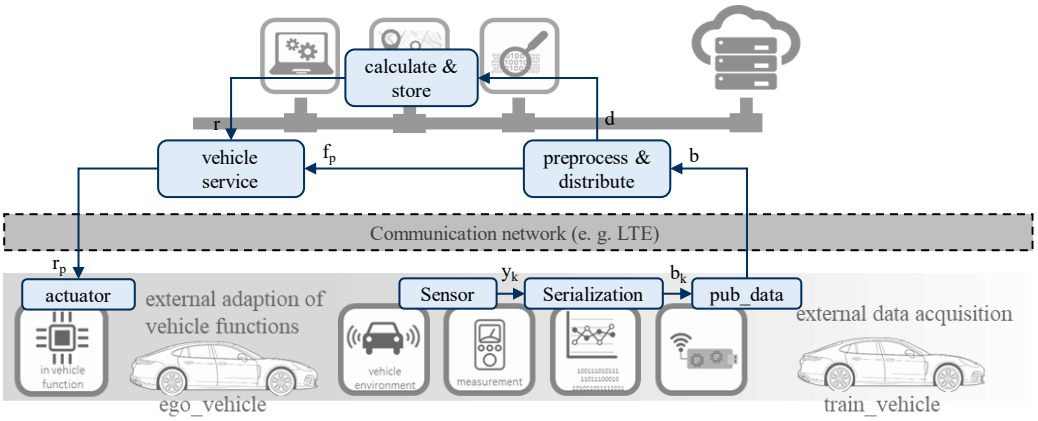

Abbildung 1: V2C2V Framework für Backendbasierte Funktionen

(y_k – sensordata vehicle k; b_k – bytestream vehicle k; b – bytestream all vehicles;
d – processing data; f_p – function data vehicle p; r – storage processing data;
r_p – function input vehicle p)

2.1 Fahrzeugkomponente des V2C2V-Frameworks

Anfallende Sensordaten y_k werden im Fahrzeug über eine Echtzeitplattform aggregiert und über einen zusätzlichen CAN-Bus an ein Vernetzungsmodul (z. B. in -vehicle PC mittels LTE-Modul) übergeben. Die Echtzeitplattform dient als Gateway zum Gesamtbussystem des Fahrzeugs. Da fahrzeuginterne Funktionen harter Echtzeit unterliegen jedoch Vernetzungsmodule nicht, müssen beide Systeme entkoppelt werden. Das Vernetzungsmodul erzeugt mittels Serialisierung einen Datenstrom b_k, welcher mittels MQTT unter einem entsprechenden Topic übertragen wird. Abbildung 2 zeigt den grundsätzlichen Aufbau der Fahrzeugkomponente (ohne Echtzeitplattform).

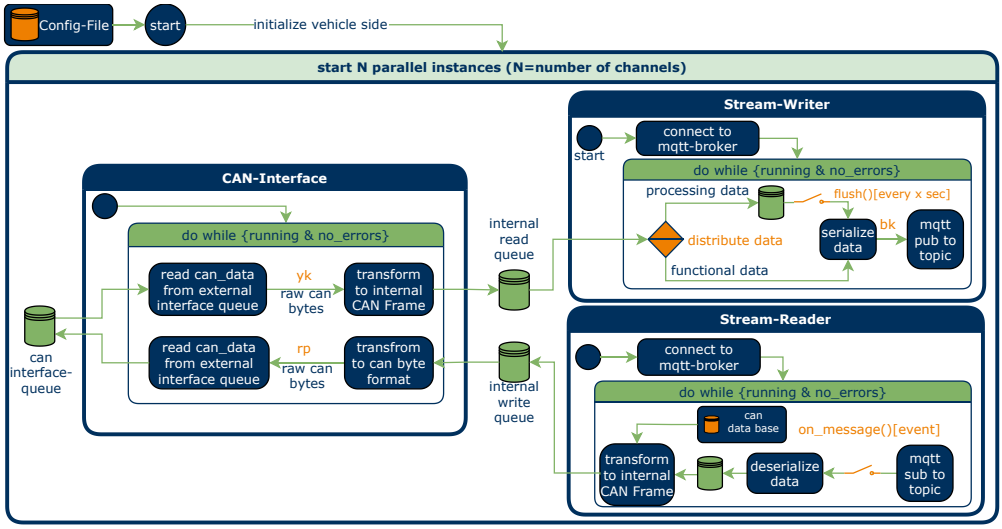

Abbildung 2: V2C2V Framework - Fahrzeugkomponente

Bei der Datenübertragung wird grundsätzlich durch den Stream-Writer in zwei unterschiedliche Nachrichtentypen nach [7] unterschieden:

- processing data – alle anfallenden Sensordaten y_k, welche zur Verarbeitung und Datenanalyse erfasst werden sollen. Hierbei kann die gesamte zur Verfügung stehende Bandbreite der Verbindung genutzt werden. Da der Fokus auf Konsistenz aber geringer Verfügbarkeit liegt müssen diese Daten unter Beachtung von geringer Datengröße bei der Serialisierung und maximaler Sicherheit der mobilen Datenübertragung versendet werden. Das bedeutet nach [7], dass die Daten idealerweise mittels MQTT (QoS = 2) zu übertragen sind. Dabei ist mit einer minimalen Verzögerung im Rahmen dieses Frameworks von ca. 600 – 800 ms zu rechnen.

- functional data – Sensordaten $y_{k,r} \in y_k$, welche zur Berechnung der backendbasierten Funktion notwendig sind. Hierbei wird nur ein Teil der zu Verfügung stehenden Bandbreite genutzt. Der Fokus liegt auf sehr hoher Verfügbarkeit der Daten. Diese sollen innerhalb einer definierten Zeitspanne (weiche Echtzeit) verarbeitet werden. Dementsprechend müssen diese Daten unter Beachtung von Datengröße und entstehenden Latenzen bei der Serialisierung und mit geringerer

Sicherheit der mobilen Datenübertragung versendet werden. Das bedeutet nach [7], dass die Daten idealerweise mittels MQTT (QoS = 0) zu übertragen sind. Dabei ist mit einer minimalen Verzögerung im Rahmen dieses Frameworks von ca. 200 – 300 ms zu rechnen.

Die Aktuatorsignale r_p werden mittels Stream-Reader innerhalb des Vernetzungsmodules empfangen und über die Echtzeitplattform an den jeweiligen Aktor oder das interne Bussystem weitergeleitet. Dabei gilt $y_k, k \in N$ und $r_p, p \in N$ für N-Fahrzeuge.

2.2 Cloud-Komponente V2C2V-Frameworks

Abbildung 3 zeigt den grundsätzlichen Aufbau der Cloud-Komponente des Frameworks. Die Daten der Fahrzeuge werden mittels open-vpn Tunnel an einen MQTT Broker übertragen (open-source Broker mosquitto) [8].

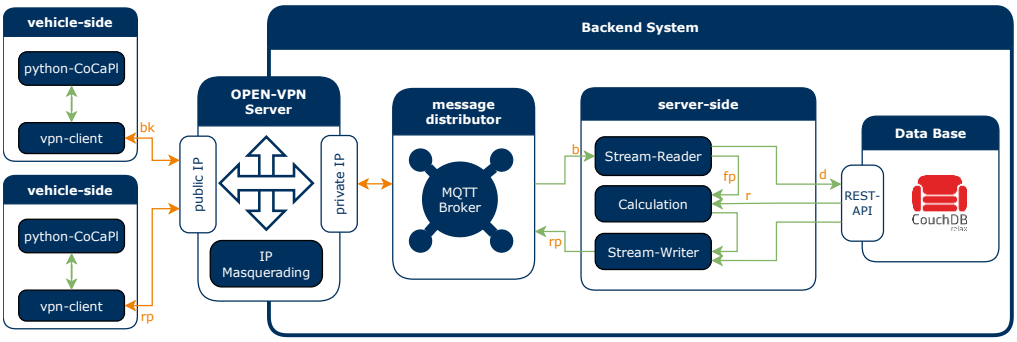

Abbildung 3: V2C2V Framework – Cloud-Komponente

Eine Stream-Reader Funktion (Funktionsweise analog Fahrzeugkomponente, vgl. Abbildung 2), verarbeitet die ankommenden Daten fahrzeugspezifisch (Unterscheidung anhand des MQTT Topics). Je nach Anwendungsfall (processing oder functional) werden diese Daten deserialisiert und weitergeleitet. Zur Datenspeicherung wird eine dokumentenbasierte Cluster-Datenbank [9] genutzt, welche die internen Datenobjekte einfach verarbeiten kann. Mittels Stream-Writer Funktion können je nach Anwendungsfall Aktuatorsignale für backendbasierte Funktionen oder einfache Prozessdaten fahrzeugspezifisch berechnet und versendet werden.

3 Analyse der Basiskomponenten des Frameworks

Die Umsetzung backendbasierter Funktionen zur Nutzung fahrzeugexterner Ressourcen ist stark von der sich ergebenden Gesamtlatenz und zur Verfügung stehenden Bandbreite des Systems abhängig. Um eine solche Umsetzung gewährleisten zu können, werden Teilkomponenten des Frameworks im Folgenden näher betrachtet.

3.1 Datenaufkommen und Latenz bei der Serialisierung

Wie unter 2.1 erwähnt, ist es entscheidend, wie Daten serialisiert und versendet werden. Dazu wurden verschiedene Serialisierungsmethoden umgesetzt und hinsichtlich Datenaufkommen und Rechenzeit untersucht (vgl. Abbildung 4). Alle durchgeführten Untersuchungen können unter [10] nachvollzogen werden.

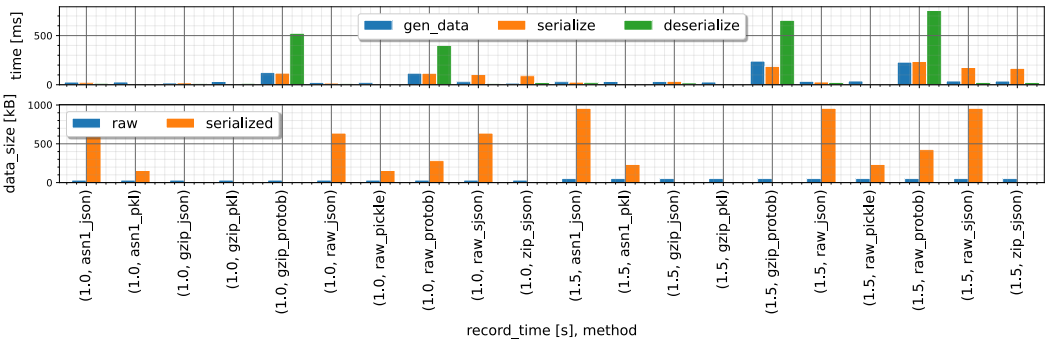

Abbildung 4: Vergleich von Serialisierungsmethoden im Rahmen des V2C2V-Frameworks

Um eine qualifizierte Aussage über die unterschiedlichen Methoden treffen zu können, muss zunächst die anfallende Datenmenge bestimmt werden. Als Beispiel hierzu dient ein CAN-Bus als Datenquelle, wie er im Framework zwischen Echtzeitplattform und Vernetzungsmodul eingesetzt wird. Auf Basis der Bitfolge eines Classical (Extended) Frame Format für ein CAN Data Frame mit $n_{header} = 18\,(32)\ bit$, $n_{trailer1} = crc = 15\ bit$, $n_{trailer2} = 12\ bit$ lässt sich unter Beachtung des worst-case bit stuffings die minimal und maximal mögliche Datenrate f_{frame_min} sowie die maximale Übertragungszeit für ein Frame t_{frame_max} für eine Datenrate f_bit wie folgt bestimmen [11], [12]:

$$t_{frame\_max} = \frac{\left(n_{header} + n_{trailer1} + n_{trailer2} + 8 * n_{bytes} + \left\lfloor \frac{(n_{header} + n_{trailer1} + 8 * n_{bytes})}{5} \right\rfloor\right)}{f_{bit}}$$

$$t_{frame\_min} = \frac{n_{header} + n_{trailer1} + n_{trailer2} + 8 * n_{bytes}}{f\_bit} \qquad (1)$$

$$f_{frame\_min} = \frac{8 * n_{bytes}}{t_{frame\_max}} \qquad f_{frame\_max} = \frac{8 * n_{bytes}}{t_{frame\_min}} \qquad (2)$$

Somit ergibt sich nach (1) und (2) für einen CAN-Bus mit 500 kbit/s eine minimale Datenrate von 209,15 kbit/s (unter Verwendung 29 bit ID) und eine maximale von 290 kbit/s (unter Verwendung 11 bit ID) sowie eine maximale Verzögerung von 0,306 ms.

Unter Berücksichtigung der Wartezeit $T_{\_Warte,M}$ für ein Frame mit bekannter Priorität nach [13, S. 70] lässt sich auch die Grenze der Echtzeitfähigkeit eines CAN-Buses bestimmen. Sofern die Prioritäten der CAN-Frames in Abhängigkeit ihrer Zykluszeit sortiert sind, kann garantiert werden, dass auch bei einer Buslast von über 90 % die

Wartezeit eines zyklischen Frames unter der Senderate liegt (vgl. Ergebnisse aus [10]). Daraus ergibt sich folgender Zusammenhang für die maximal anzunehmende Datenmenge (Frames pro Sekunde):

$$\frac{n_{fames\_max}}{sec} = \left\lfloor \frac{f_{bit}}{n_{header} + n_{trailer1} + n_{trailer2} + 64} * 0{,}9 \right\rfloor / \text{sec} \quad (3)$$

Nach (3) ergeben sich 4090 Frames pro Sekunde als maximal anzunehmende Datenmenge für den aktuell im V2C2V-Framework eingesetzten CAN-Bus. Auf Basis dessen wurden die in Abbildung 4 dargestellten Serialisierungsmethoden untersucht.

- Methoden mit sehr hohem Zeitaufwand: raw_protobuf, gzip_protobuf, raw_sjson, zip_sjson – sind ungeeignet

- Methoden mit sehr hoher Datengröße: ans1_json, asn1_pkl, raw_json – sind ungeeignet

In Abbildung 5 sind die $Latenz$ $(= time_{serialize} + time_{deserialize})$ sowie das Verhätnis der Datenrate zur maximalen Datenrate des CAN-Buses der restlichen Methoden dargestellt. Die Serialisierung der Daten mittels json und das anschließende Komprimieren mittels gzip liegt nahe der eigentlichen roh-Datenraten. Das bedeutet, im ungünstigsten Fall können nicht mehr alle Daten serialisiert werden und es entsteht ein Flaschenhals. Die Methode gzip_pkl ist der Methode raw_pickle hinsichtlich des anfallenden Datenvolumenes deutlich überlegen (> Faktor 100). Hinsichtlich Latenz ist die unkomprimierte Variante ca. doppelt so schnell. Daraus ergeben sich die folgenden Resultate:

- raw_pickle: Nettodatenrate je CAN-Bus: 156 KB/s (bei einem Protokolldurchsatz von ca. 25 % nach [14]) Bruttodatenrate (ISO/OSI Layer 4-7): ca. 624 KB/s
  raw_pkl: Latenz bei 10 Hz Senderate: 0,35 ms; bei 1 Hz Senderate: 3,2 ms

- gzip_pkl: Nettodatenrate je CAN-Bus: 0,8 KB/s respektive 3,2 KB/s
  gzip_pkl: Latenz bei 10 Hz Senderate: 0,5 ms; bei 1 Hz Senderate: 4 ms

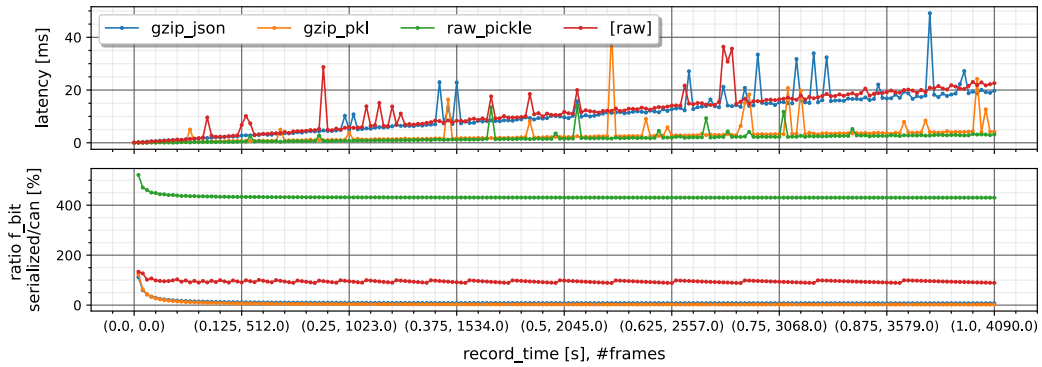

Abbildung 5: Latenz und Verhältnis der Datenrate bezogen auf die Nettodatenrate eines 500 kbit/s CAN-Buses von Serialisierungsmethoden im Rahmen des V2C2V-Frameworks

3.2 Validierung der Datenerfassung des gesamten Frameworks

Um sicherzustellen, dass alle im Fahrzeug versendeten Daten exakt in der externen Serverkomponente verarbeitet werden können sowie konsistent in der Datenbank gespeichert werden, wurde die Datenerfassungskette validiert. Dabei wurde zusätzlich zur Datenerfassung mittels des V2C2V-Frameworks (Fahrzeugbus – Echtzeitcontroller – Vernetzungsmodul – MQTT – Backend-System – Datenbank) eine Erfassung mittels Vector CANoe (Fahrzeugbus – Vector VN1611 – Messrechner mit CANoe – blf-file) durchgeführt.

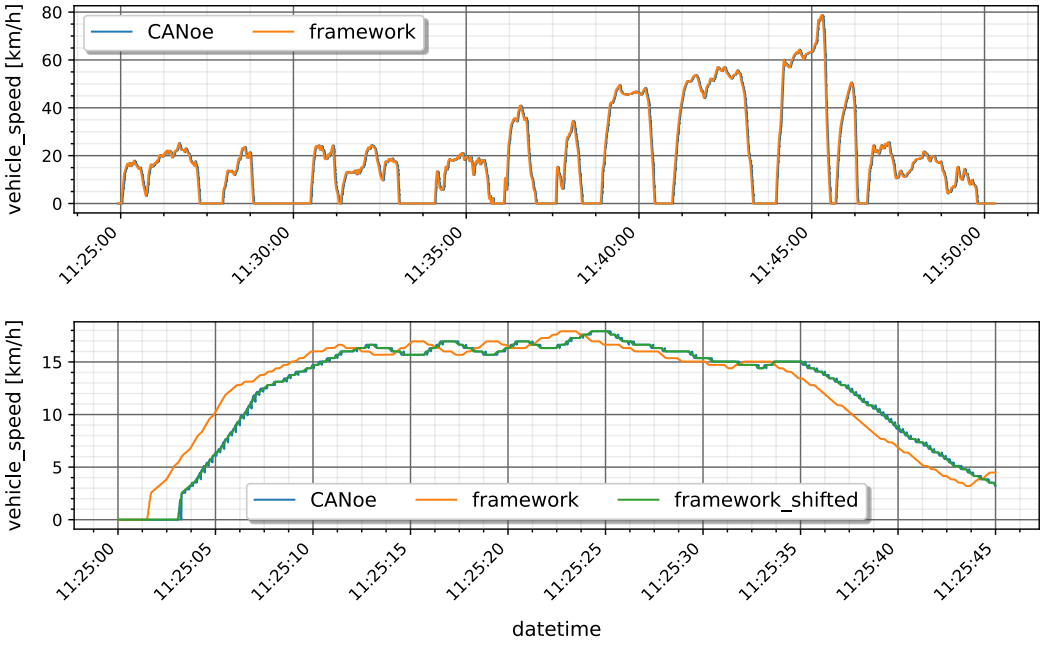

Abbildung 6: Vergleich Datenerfassung mittels Vector CANoe und V2C2V-Frameworks; oben: vor time shift, unten: nach time shift mit zeitlichem zoom

Abbildung 6, oben zeigt beispielhaft die Kurven der Geschwindigkeit des Fahrzeugs für den Messzeitraum von 25 min. Der Korrelationskoeffizient nach Pearson der beiden Kurven beträgt 0,99 und es lässt sich kaum ein Unterschied feststellen. Wird ein kürzerer Messzeitraum betrachtet (Abbildung 6, unten), ist ein Zeitversatz erkennbar. Dieser Zeitversatz entsteht durch die unterschiedlichen Systemzeiten der beiden Messsysteme. Der Korrelationskoeffizient beträgt in diesem Fall nur noch 0,94.

$$R_{xy}(t_{delay}) = (x * y)[n] = \sum_{m=-\infty}^{\infty} \bar{x}\,[m]\; y\,[m + t_{delay}] \qquad (4)$$

Mittels Kreuzkorrelation kann ein Zeitversatz t_{delay} = 1,56 Sekunden nach [15] mittels (4) bestimmt werden. Eine zeitliche Verschiebung der Daten (framework_shifted) führt zu einem Korrelationskoeffizienten von 0,99. Diese Untersuchung zeigt, dass die mittels des Frameworks erfassten Daten als valide angesehen werden können.

3.3 Verfügbarkeit von backendbasierten Funktionen

Da die Datenverbindung zwischen Fahrzeug und fahrzeugexterner Funktionskomponente stark schwankenden Latenzen unterliegt [7] müssen diese erfasst und als Parameter in die Funktion zur Laufzeit einbezogen werden. Mittels des beschriebenen Frameworks wird daher die Umlaufzeit (RTD) zwischen Stream-Writer im Sever zum Stream-Reader im Fahrzeug und zurück mit einer festen Zykluszeit ständig erfasst.

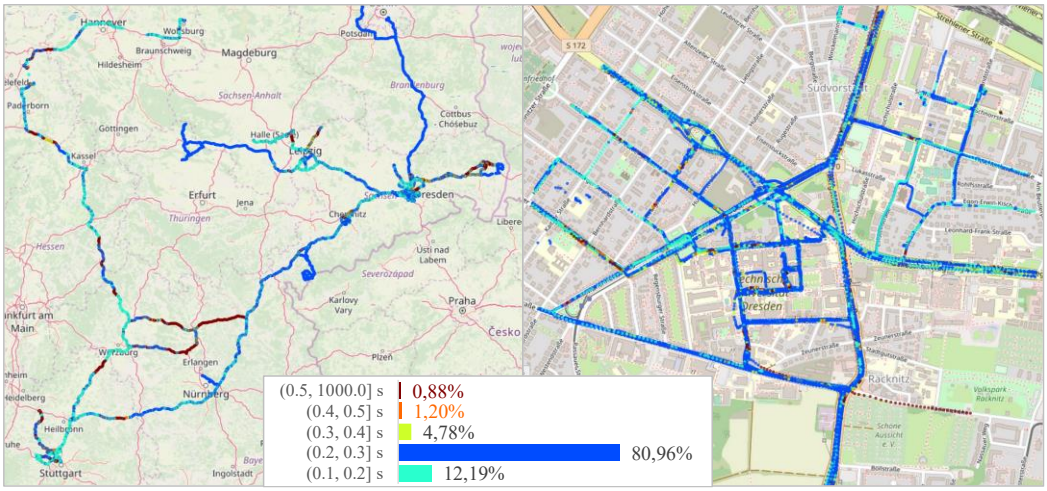

Abbildung 7: durchschnittliche Round-Trip E2E Latenzen des Frameworks

Abbildung 7 zeigt die durchschnittliche räumliche und zeitliche Verteilung der resultierenden RTD des entwickelten Frameworks an. Die Werte wurden über einen Zeitraum von > 6 Monaten erfasst und bestätigen die Annahmen aus [7]. Mehr als 93 % aller gemessenen RTDs liegen unter 300 ms. Dies bedeutet für eine backendbasierte Funktion ein delay für y_r und r_p von 150 ms. Das heißt, die serverseitige Softwarekomponente erhält in der Regel 150 ms verzögerte Daten und muss mind. 150 ms in die Zukunft prädizieren. Im Falle einer Fahrzeuggeschwindigkeit von 50 km/h ergibt sich eine Unsicherheit von ca. 4 m. Bezogen auf backendbasierte Funktionen, deren Auswertung geobasiert erfolgt (vgl. [6]) können diese Ungenauigkeit im Rahmen der Positionierungenauigkeit teilweise ohne Prädiktion kompensiert werden.

4 Zusammenfassung und Ausblick

Im Rahmen der Arbeit wurde gezeigt, wie die Struktur eines V2C2C-Frameworks aufgebaut werden muss, mittels welcher backendbasierte Funktionen umgesetzt werden können. Das umgesetzte System wurde hinsichtlich Datenkonsistenz und auftretender Latenzen validiert und ausgewertet.

Mittels backendbasierter Funktionen ist im Rahmen dieses Frameworks eine bedarfsgerechte Funktionssteuerung zwischen einer zentralen externen Recheneinheit und Fahrzeugen möglich, welche jedoch in weiterführenden Arbeiten untersucht werden

müssen. Beispielsweise sollte in Abhängigkeit der aktuell auftretenden RTD ein gleitender Horizont zur Prädiktion der Aktuatoransteuerung integriert werden, sodass nicht mehr Prädiktionsdaten als notwendig berechnet werden. Je nach Zeitkonstanten der Fahrzeugfunktion könnten Funktionsanteile aus dem Fahrzeug in eine externe Einheit ausgelagert werden und die on-board Logik des Fahrzeugsystems ersetzen.

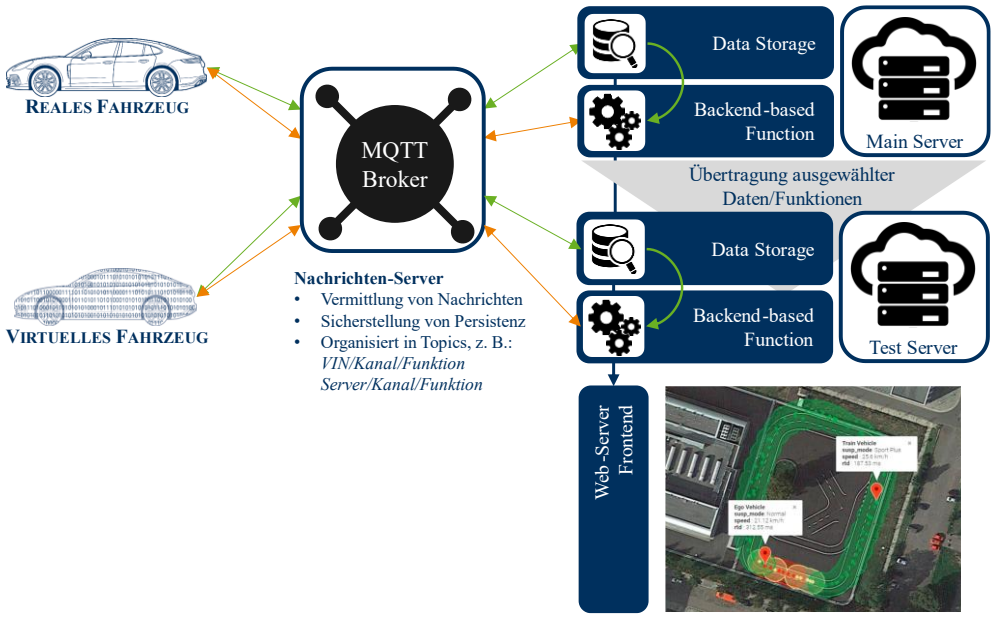

Abbildung 8: Test-Setup mittels Virtueller Fahrzeuge

Neben der Implementierung neuer Funktionen können bereits vorhandene mittels gezielter Nachrichtenverteilung (wie in Abbildung 8 dargestellt) auf realen Fahrzeugen und/oder digitalen Abbildern getestet werden. So ist es möglich, Funktionen unter realen Bedingungen zu testen, ohne dass sich das Zielfahrzeug in einer solchen Umgebung befinden muss.

5 Literaturverzeichnis

[1] Falk Meissner, Konstantin Shirokinskiy, und Michael Alexander, „Computer on Wheels: Disruption in automotive electronics and semiconductors“, ROLAND BERGER GMBH, München, Jan. 2020. Zugegriffen: 15. September 2022. [Online]. Verfügbar unter: https://www.rolandberger.com/de/Insights/Publications/Das-Auto-wird-zu-einem-Computer-auf-R%C3%A4dern.html

[2] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder, und A. Mouzakitis, „A Taxonomy and Survey of Edge Cloud Computing for Intelligent Transportation Systems and Connected Vehicles“, *IEEE Transactions on Intelligent Transportation Systems*, Bd. 23, Nr. 7, S. 6206–6221, Juli 2022, doi: 10.1109/TITS.2021.3084396.

[3] VDA, „Connected-Car-Technologie“. https://www.vda.de/de/themen/digitalisierung/connected-car-technologie (zugegriffen 14. September 2022).
[4] *Verordnung (EU) 2015/758*. 2015. Zugegriffen: 14. September 2022. [Online]. Verfügbar unter: http://data.europa.eu/eli/reg/2015/758/oj/deu
[5] L. Zhang, X. Yin, J. Shen, und H. Yu, „Cloud-aided moving horizon state estimation of a full-car semi-active suspension system“, in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Okt. 2016, S. 000527–000532. doi: 10.1109/SMC.2016.7844293.
[6] Tim Häberlein, Bernard Bäker, und Oliver Manicke, „Backendbasierte Fahrzeugfunktionen – Herausforderungen zur durchgängigen Funktionsüberprüfung“, in *Diagnose in mechatronischen Fahrzeugsystemen XII: neue Verfahren für Test, Prüfung und Diagnose von E/E-Systemen im Kfz*, Dresden, 2018, S. 229–235.
[7] Tim Häberlein, Andreas Unger, Bernard Bäker, und Oliver Manicke, „Framework conditions for the implementation of backend-based functions in the context of connected vehicles“, in *ELIV 2017, VDI-Berichte. Bd. 2299*, Bonn, Okt. 2017, S. 45–46. doi: 10.51202/9783181022993-45.
[8] „Eclipse Mosquitto“, *Eclipse Mosquitto*, 6. Juli 2020. https://mosquitto.org/documentation/ (zugegriffen 15. September 2022).
[9] „Apache CouchDB“. https://couchdb.apache.org/ (zugegriffen 15. September 2022).
[10] Tim Häberlein, „th89dd/framework_validation_autotest · GitHub“, *Framework Evaluation for Backendbased Functions in the Context of Connected Vehicles*. https://github.com/th89dd/framework_validation_autotest (zugegriffen 15. September 2022).
[11] ISO, „ISO 11898-1:2015“, International Organization for Standardization, Dez. 2015. Zugegriffen: 8. September 2022. [Online]. Verfügbar unter: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/36/63648.html
[12] ISO, „ISO 11898-2:2016“, International Organization for Standardization, Dez. 2016. Zugegriffen: 8. September 2022. [Online]. Verfügbar unter: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/72/67244.html
[13] W. Zimmermann und R. Schmidgall, *Bussysteme in der Fahrzeugtechnik Protokolle, Standards und Softwarearchitektur ; mit 103 Tabellen*, 5., Aktualisierte und erw. Aufl. Wiesbaden: Springer Vieweg, 2014. Zugegriffen: 8. September 2022. [Online]. Verfügbar unter: http://www.gbv.de/dms/tib-ub-hannover/774407069.pdf
[14] V. Sarafov, „Comparison of IoT Data Protocol Overhead“, gehalten auf der Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM), München, 2018. doi: 10.2313/NET-2018-03-1_02.
[15] „scipy.signal.correlation_lags — SciPy v1.9.1 Manual“. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlation_lags.html (zugegriffen 16. September 2022).