electronics

MDPI

Article

Continuous Automotive Software Updates through Container Image Layers

Nicholas Ayres [1], Lipika Deka [2,*] and Daniel Paluszczyszyn [2,3]

[1] Department of Computer Science and Informatics, Cyber Technology Institute, De Montfort University, Leicester LE1 9BH, UK; nick.ayres@dmu.ac.uk
[2] The De Montfort University Interdisciplinary Group in Intelligent Transport Systems (DIGITS), Department of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, UK; paluszcol@dmu.ac.uk
[3] Royal Academy of Engineering Industrial Fellow, HORIBA MIRA Ltd., Nuneaton CV10 0TU, UK
* Correspondence: lipika.deka@dmu.ac.uk

Abstract: The vehicle-embedded system also known as the electronic control unit (ECU) has transformed the humble motorcar, making it more efficient, environmentally friendly, and safer, but has led to a system which is highly dependent on software. As new technologies and features are included with each new vehicle model, the increased reliance on software will no doubt continue. It is an undeniable fact that all software contains bugs, errors, and potential vulnerabilities, which when discovered must be addressed in a timely manner, primarily through patching and updates, to preserve vehicle and occupant safety and integrity. However, current automotive software updating practices are ad hoc at best and often follow the same inefficient fix mechanisms associated with a physical component failure of return or recall. Increasing vehicle connectivity heralds the potential for over the air (OtA) software updates, but rigid ECU hardware design does not often facilitate or enable OtA updating. To address the associated issues regarding automotive ECU-based software updates, a new approach in how automotive software is deployed to the ECU is required. This paper presents how lightweight virtualisation technologies known as containers can promote efficient automotive ECU software updates. ECU functional software can be deployed to a container built from an associated image. Container images promote efficiency in download size and times through layer sharing, similar to ECU difference or delta flashing. Through containers, connectivity and OtA future software updates can be completed without inconveniences to the consumer or incurring expense to the manufacturer.

Keywords: virtualisation; ECU; automotive E/E architecture; containers; over the air; software updates

Citation: Ayres, N.; Deka, L.; Paluszczyszyn, D. Continuous Automotive Software Updates through Container Image Layers. *Electronics* **2021**, *10*, 739. https://doi.org/10.3390/electronics10060739

Academic Editors: Calin Iclodean, Bogdan Ovidiu Varga and Felix Pfister

Received: 7 March 2021
Accepted: 18 March 2021
Published: 20 March 2021

1. Introduction

In 1886, Karl Benz built what was considered the first modern motor vehicle: the Benz Patent-Motorwagen, the humble car has transformed, not just in looks but function. In 1977, General Motors released the Oldsmobile Toronado, which is regarded as the first car to include an electronic control unit (ECU) [1]; this first implementation managed the electronic spark timing of the combustion process. ECUs benefit the driver with a safer, efficient and more comfortable ride but the benefits can also be seen with regard to the vehicle such as lower CO2 emissions, reduced mechanical wear and higher efficiency in operation. Vehicle systems are no longer mechanically linked together, but rather software driven hardware, connected between driver input and vehicle output. Since ECUs were introduced, software has become an integral part of the motorcar similar to any mechanical component which aids in its function and operation.

According to [2], "over 80% of innovations in the automotive industry are now realised by software-intensive systems". Over 100 million lines of software code across

100 ECUs can be found within the automotive E/E architecture of many modern motor vehicles providing vehicle functions from engine management to passenger comfort [3,4]. These diverse functions make the modern motorcar one of the most software-intensive systems we use in our day-to-day lives [3,5,6]. There are regular and periodic preventative and proactive maintenance procedures of a vehicle's physical components throughout its lifetime [7]. However, the same statement cannot be said concerning automotive software. Despite the requirement for reliable software, bugs and errors are unintentional but appear frequently within software code [8,9]. How and why software code contains errors and flaws are varied [10–12]. Problems are often introduced during the various stages of the software life-cycle. For example, bugs and errors in software code can lead to unexpected and sometimes dangerous results in the output of software-driven devices and functions [13–16]. Current automotive software update practices and procedures are problematic because there is no clearly defined mechanism or standard.

Current software update mechanisms often follow the same return or recall mechanism associated with a physical vehicle component failure. The Original Equipment Manufacturer (OEM) may issue a vehicle recall notice, especially if the fault concerns a safety issue. The "return or recall" process has its own associated problems including cost to the manufacturer and inconvenience to the consumer [13,17,18]. In order to address these limitations, a new approach to automotive software updates is required. Software Over the Air (OtA) update mechanisms can update automotive software without the need to return the vehicle to an authorised garage or dealership [19–21]. Using the increasing deployment of on-board vehicle connectivity, OtA updates can deliver new software as and when required [22]. However, current rigid ECU hardware designs do not facilitate or promote an architecture that can benefit from an OtA software update mechanism.

The focus of this paper is to propose and investigate how specific lightweight virtualisation also known as containers can be deployed within the automotive E/E architecture to promote periodic remote OtA software updates [23]. A container-based ECU can address many of the current software updating issues identified within this paper. It can provide a scalable and updateable solution that is not dependant on many applications of individual ECU hardware systems, which is the standard practice in current automotive E/E architecture design. Automotive functionality hosted within containers is a promising technology which can address many of the current inadequacies in automotive software updating and has the potential to deliver a standardised mechanism to promote continual software updates throughout the vehicle's lifetime as well as a platform that can provide new system functionality to the consumer through aftermarket market sales.

2. Automotive E/E Architecture: Software Associated Issues

Vehicle software is considered to be a significant component of the modern motorcar. As the number of ECUs increases, it inevitably results in more lines of software code to drive those systems. As more lines of code are included it raises specific issues related to an increased dependency on software.

2.1. Software Bugs and Errors

Automotive ECU software is often designed, developed, and written by third party suppliers. However, according to [10], "guessing what the designer's intentions were most often results in more bugs". Studies into the quality of software indicate strong correlations between the size of the application and the total number of defects [11]. Reference [12], states that a software system consisting of millions of code lines could have tens of thousands of unknown or undetected bugs. The following chart (Figure 1) highlights the increasing trend of software associated vehicle recalls. In 2018, 8 million vehicles in the U.S. were affected by some form of software defect.

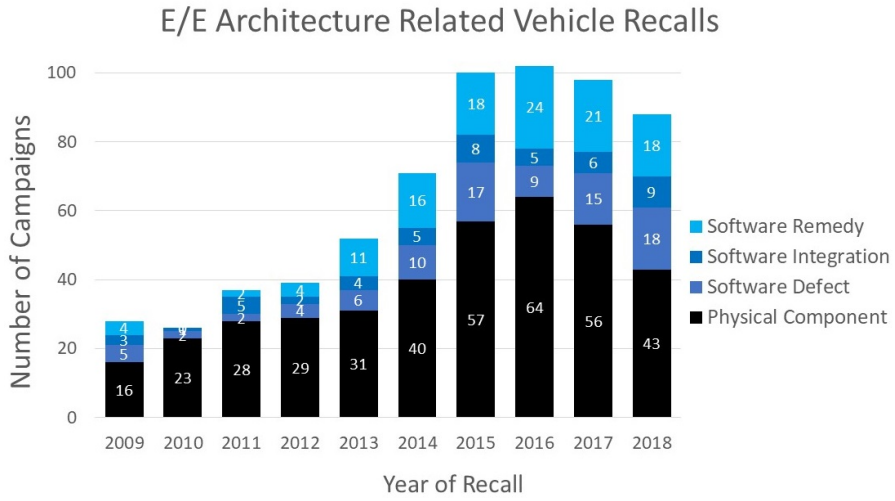

Figure 1. Software related automotive recalls.

As stated in [24], automotive recalls can be classified into four groups, three of which specifically relate to automotive software:

- Integrated electronic components—Failure of a physical, electronic component.
- Software integration—Software interfacing failure between different automotive components or systems.
- Software defect—ECU software failure.
- Software remedy—Fault not solely attributed to software failure but was remedied using a software update/patch.

Depending on the software's application and how critical it is to operational safety bugs and software errors can have disastrous consequences [13]. For example, in 1996, the Ariane 5 Flight 501 rocket disintegrated 40 s after launch due to an undiscovered software error within an arithmetic routine installed in the flight computer. The software bug led to the backup and primary systems crashing, which ultimately led to the rocket's failure [25]. According to [26], the second most common reason for a vehicle-related collision is attributed to automotive software bugs.

Many ECU systems within the modern motorcar are safety-related or considered safety-critical. Any failure in an automotive safety-critical system can potentially endanger vehicle occupant safety. Embedded software bugs and errors can cause control flow errors which result in a flawed execution of the program that can lead to sensor or actuator failure or the system hanging or crashing [27]. To mitigate against these types of errors in dependable and safety-critical systems, expensive hardware-based countermeasures such as triple modular redundancy are often required.

2.2. *Software Associated Security Threats*

Vehicles are no longer closed systems that require direct physical access to gain unauthorised entry to the car. Vehicle connectivity is gaining popularity as it offers vehicle occupants a mechanism to connect to services outside of the vehicle via the Internet. However, the potential to compromise automotive security through vehicle-based connectivity now has the potential to come from anywhere. Nevertheless, even though connectivity systems have been incorporated into vehicles over the last few years, "car hacking" has not been widespread due to the limited potential for cyber-crime and cyber-criminals.

In 2015, two security professionals, Charlie Miller and Chris Valasaek, demonstrated how to compromise a motor vehicle remotely through its connectivity system and vulnerabilities within its software code. They gained access through the HMI unit known as the Uconnect system in the Gran Jeep Cherokee target vehicle. Access to the CANbus vehicle network was possible through the design of this device. This system incorporates an interface for particular vehicle operational and media functions. Due to vulnerabilities

in the HMI operating system software, the software update validation mechanism was disabled, which permitted malware injection into the Uconnect software. Once compromised, the system enabled the attackers to remotely inject spoofed CAN frames to ECUs which were responsible for vehicle control. The HMI vulnerability allowed the hackers to interfere with various vehicle subsystems, including interior climate control and vehicle windscreen wipers. They also manipulated safety-critical systems, including shutting down the engine and limited steering control. The Uconnect HMI is a standard product supplied by Fiat Chrysler and is incorporated into numerous vehicle models across several different vehicle makes. This software vulnerability affected 100,000 s of vehicles globally [5,28–30].

As the connected car becomes mainstream, it will ultimately become more of a target for cyber-criminals [31]. Vehicle autonomy and many current ADAS features place the vehicle in level 3 or 4 on the autonomy scale, where level 0 reflects complete driver control and level 5 reflects complete computer control. An intruder's potential to gain remote system access and subsequent unauthorised control of a moving vehicle is an increasing possibility [32]. Vehicle infotainment systems present large attack surfaces that often delivers bi-directional vehicular connectivity. As such, any discovered vulnerabilities within these systems software must be patched promptly to maintain the integrity of the vehicle's subsystems and occupants' safety [33,34].

2.3. Ageing and Out of Date Code

Automotive E/E components, including associated ECU software, is often designed and developed years before a particular vehicle model eventually leaves the sales forecourt. The average vehicle has a life expectancy of between 10 to 15 years, and automotive software must mirror this long-time frame. Automotive system longevity significantly differs from many other software-based systems used in our day to day lives. For example, periodic software updates are routinely applied to general-purpose computing and personal smart devices throughout their lifetime. Regular updates address flaws and bugs in software code, provide security and deliver new or additional system functionality [35,36]. According to [37], software can exhibit signs of ageing where old software versions lose market share and customers to new software products. Furthermore, reliability can decrease because of the introduction of bugs and errors during periodic maintenance.

2.4. Aftermarket Sales and Additional Functionality

Throughout its life, the modern motorcar requires a robust aftermarket industry to sustain vehicle longevity. Currently, the automotive aftermarket sector is predominately concerned with two main revenue streams; services and parts. The service sector includes the maintenance and repair of vehicles, and accounts for approximately 45% of total European aftermarket revenue. The remaining 55% involves the sale of vehicle parts. The global aftermarket industry in 2015 was worth an approximate $760 bn and accounted for 20% of total automobile revenues [4].

Consumers increasingly demand the features and functions they use on their smart devices to be made available within their vehicles. The automotive industry is looking towards connectivity to provide the consumer with new aftermarket automotive features and functions. Figure 2 highlights the most significant influence over new car purchase decision where 10 means in-car technology has the most significant influence, and 1 refers to the car's performance as the predominant factor. In response to this trend, infotainment systems that offer an "Apple-like" experience are predicted to grow from 18 million units in 2015 to 50 million by 2025 [38].

Three of the six top trends surrounding aftermarket sales refer to new and emerging digital technologies, these include:

- Interface digitisation—by 2035, there will be a predicted shift of between 20–30% from physical component replacement to software upgrades of vehicle components, including new digital services which can be purchased on demand [4].

- Car-generated data—connected vehicles generate considerable amounts of telematics and driver data, approximately 25 GB per hour. Through big data analytics, consumer-generated data can be of substantial value to the manufacturer in determining consumer insights, predictive maintenance and remote diagnostics.
- The increasing influence of digital intermediaries—usage-based companies and technology companies are increasingly using vehicle-generated data. These sectors will require mechanisms to facilitate the retrieval and frequent deployment of automotive software.

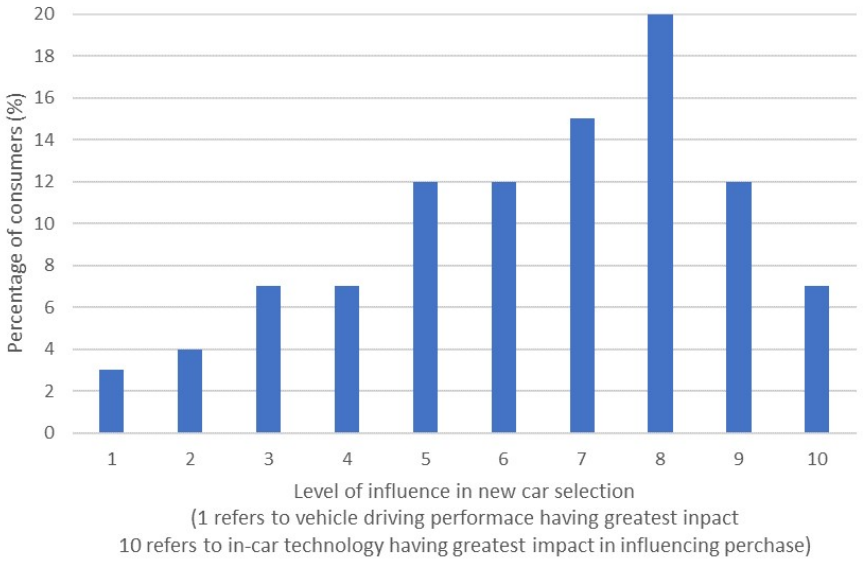

Figure 2. Consumer Preference: New Car Selection.

3. Automotive Software Updating

In the modern motorcar, almost all aspects of vehicle operation require considerable amounts of software code [3,9,39]. However, as with all software, automotive software needs to be periodically updated. In an increasingly software-centric automotive E/E architecture, new software installations may be required several times during a vehicle's lifetime.

The process of online or OtA software updates has been seen in personal computing technology and more recently in our smart devices, which are updated periodically to provide software bug fixes and the latest security patches, and add new software functionality or install newer software and operating system versions. This is enabled through the device's own connectivity hardware. However, this wide-scale software update mechanism is in an infant stage in automobiles.

Any future automotive software update mechanism must present minimal disruption to the customer and be cost-effective to the manufacturer and supplier. There are several principle reasons why it is vital to periodically update automotive software, these include:

- Addressing system failure through software errors and bugs.
- Patching or enhancing the system and software security.
- Adding value post-sale through aftermarket content.

Current automotive software update practices and procedures are problematic because there is no clearly defined mechanism or standard. Historically, when a common fault was discovered within a particular installed physical component of a vehicle, the OEM could issue a vehicle recall notice [40], especially if the fault reflects a severe safety issue.

The current mechanisms for automotive software updates are ad hoc at best. This paper has identified three common mechanisms, including:

- Manufacturer-initiated vehicle recall process.
- Guided user intervention.
- Over the air update.

3.1. Software Update Mechanism: Manufacturer-Initiated Recall Process

Vehicle recalls are relatively common. For example, since 1966 in the U.S., over 390 million vehicles have been recalled due to safety issues [41]. Like a physical component, a software-related problem, depending upon the severity, needs to be addressed and resolved. The recall mechanism, for both physical and software-related issues, requires the vehicle's return to a qualified engineer to rectify the problem [13]. Vehicle recalls are an expensive exercise for the manufacturer [13,17,18,42]. They are also a disruptive and time-consuming procedure for the customer [42,43].

The process of a physical component fix may differ from a software fix. Physical components are replaced with new ones, often because of mechanical wear or a fault in the original component design or construction. In contrast, a software fault may require specialist equipment and a new software version installed on the existing hardware. However, this is not always possible with older embedded systems. Legacy ECU systems have their code pre-set at component manufacture. According to [44], high hardware optimisation often results in ECUs with minimal resources where limited storage, memory and processing capacity cannot accommodate additional lines of new software code. Limitations in ECU hardware resources require a similar exchange of hardware to repair a software-related fault. As such, like a physical component, ECU hardware exchange may be the only option to repair a software-related defect. This has led to a state where more than 50% of error-free hardware is replaced with entirely new hardware to resolve a software-related issue [44].

Incurred manufacturer maintenance costs can be high if a previously undetected software error or design flaw requires a vehicle recall [45]. A much higher cost multiplier to repair a software fault post-production is applied when compared with identifying the same fault much earlier in the software development life cycle. Cost is not the only factor in this update process. Customer confidence and brand loyalty can also be affected by software bugs and errors [6]. In recent years this has been an issue with the highly publicised Grand Jeep Cherokee cyber-attack [5,28,30].

3.2. Software Update Mechanism: Guided User Intervention

This mechanism uses a physical input port installed inside the vehicle. Many modern cars provide a physical connection port for their owners' portable electronic devices, such as external Global Positioning System (GPS) and personal mobile devices, including MP3 players, mobile phones and tablets. These devices are often connected to the vehicle via a universal serial bus (USB) port. Using this port, vehicle owners are able to undertake their own software update either by inserting a supplied preloaded removable storage device or downloading a specific update from the manufacturer onto a USB device. Notably, Fiat Chrysler employed this type of update following the 2015 Grand Jeep Cherokee remote cyber-attack. Using the postal system, Fiat Chrysler distributed preloaded USB memory sticks with updated software to 1.4 million affected customers [30]. However, there are problems associated with this type of update mechanism. These include the following:

- Limited port functionality.
- Inaccessible code.
- Basic understanding of computing technology.
- Willingness to undertake the task.

If any of the above prerequisites cannot be achieved, the software update will not be completed and it will be left unresolved. This software update method relies heavily on the customer having a particular level of technical knowledge and a willingness to perform the update process themselves. For example, there may be a reluctance to complete a necessary software update task due to a fear that their actions could "break the car", rendering it unserviceable and them responsible for any additional repair costs. Furthermore, there are inherent security risks. This method is open to potential exploitation from malicious threat actors that could enable unauthorised vehicle system access or the introduction of malware into the vehicle through compromised storage devices or software download files [46].

3.3. Software Update Mechanism: Over the Air (OtA) Update

OtA mechanisms can update automotive software without the need to return the vehicle to an authorised garage or dealership, or relying on the customer to update themselves. Using on-board vehicle connectivity, OtA updates can deliver new software as and when required. There are several options which can provide OtA software updates:

3.3.1. Dedicated Short-Range Communication (DSRC)

DSRC is an 803.11p-based wireless communication technology used for vehicle to infrastructure (V2I) and vehicle to vehicle (V2V) communication to aid and support ADAS and autonomous driving technologies. This communication technology can be used to transfer software updates between fixed infrastructure or vehicles [47–49]. However, the primary issue with DSRC and automotive software updates is the relatively short time frames involved in V2I and V2V, especially when vehicles are travelling in the opposite direction.

3.3.2. Cellular Networks

In contrast to DSRC, cellular network technology (3G, 4G, and 5G) can provide a stable high bandwidth communication mechanism. Software updates are downloaded by connecting to a particular cell tower within range, regardless of vehicle speed and travel direction. However, coverage may be restricted due to geographical limitations. Nevertheless, by using the extensive scope of cellular networks, future automotive software updates can be transmitted and downloaded to the target vehicle regardless of that vehicle's location. New software, when released, can also be downloaded.

3.3.3. Fixed Location Wireless Local Area Network (WLAN)

This is another potential option for receiving software downloads. Updates can be sent to the target vehicle while parked, for example, at home or at work. Tesla has been using this OtA update mechanism from 2017 by using P2P wireless connections to download software from Tesla servers to target vehicles [50].

Whichever form of OtA update mechanism is chosen, requires a vehicle connectivity solution. There are three modes of connectivity operation, depending upon the connection hardware type employed in the vehicle:

- Mirrored—applications stored on a paired portable smart device are replicated onto the vehicle's HMI unit. The application processing is usually performed on the smart device with screen updates sent to the HMI via a physical or wireless connection [5].
- Tethered—this type of connection uses the paired device's communication technology. Applications are installed to the vehicle's HMI unit and application data processing is performed within the car.
- Embedded—a vehicle with this type of connectivity does not rely on a paired smart device but uses its own connectivity hardware and installed applications.

There has been a widespread introduction of Long-Term Evolution (LTE) technology within the motor vehicle using one of the three aforementioned connectivity types in recent years.

4. The Significance of OtA Software Updates

There are several benefits associated with OtA software updates, making it a promising technology for the automotive industry. Through using lightweight virtualisation technology and OtA software updates can provide several specific benefits:

- Reduction in vehicle recalls and associated costs.
- Vehicles can be updated in locations other than a dealership or maintenance garage.
- Centralised software—software updates can be distributed directly to the target vehicles without distributing to dealers and maintenance garages.

- Time to market—new software can be distributed as and when required rather than waiting for the customer to return the vehicle or waiting for periodic maintenance schedule.
- Convenience—updates can be performed at the customer's desired location and discretion, reducing vehicle downtime.
- Mandatory updates—new and updated software, especially where safety is concerned, can be pushed to the target vehicle without waiting for customer participation.
- Increase in safety—OtA software updates can reduce the time a vehicle is operated under faulty conditions.
- Proven technology—OtA updates are widespread in the telecommunication industry, which has provided users with new updated software via OtA mechanisms.

Reference [51] has predicted that vehicle connectivity could be available in all new motor vehicles by 2025. In 2015, [21] suggested OtA software updates were an attractive technology for the OEM and the customer, with cost savings expected to reach $35 billion by 2022.

5. Current Automotive Software Re-Flashing Techniques

The current practice of updating automotive ECUs involves software flashing or re-flashing techniques [20,52,53]. The operating system and functional software of an ECU are generally held within embedded FLASH memory. Depending upon the model, modern motor vehicles can have hundreds of megabytes of FLASH memory spread across their ECUs. Under the return or recall mechanism, flashing or re-flashing software is often completed by authorised personnel requiring the vehicle to be offline. New software is delivered to the target ECU in one of two formats—full binary and diff/Delta file [32].

5.1. Full Binary Re-Flashing

ECU firmware is updated in its entirety through a process known as re-flashing, which conforms to ISO 14229-3/UDS and ISO 15765-2/DoCAN. As part of this process, the entire ECU software image is replaced with a newer version and the time taken to update the software can often take hours to complete. This in part depends on the size of the software update, the destination memory, protocol and whether encryption is used. The previously installed software has no relevance on the new update, which can be beneficial if the previous version requires replacing in its entirety rather than upgrading specific parts. The size of the image binary impacts the time taken to transmit and download the file. The new updated software image must also be stored within the target ECU, which requires redundant storage, potentially of an undetermined fixed amount in order to accommodate any future software update.

5.2. Difference/Delta File

Diff/delta file flashing is a concept that compares the base file with the new version file and creates a delta or difference file, thus reducing the size of the update [50]. Compared with a full binary software update, a diff/delta software update is approximately below 10% of the full binary file size. Diff/delta files are much quicker to transmit, decreasing overall transmission time by up to 90%. This method requires considerably less redundant storage but it is reliant on the previous ECU software version. A patching algorithm block erases the old data and writes new data in its place.

6. Container-Based Software Updating

Current software upgrades and bug fixes require the car to be shut down while being updated and subsequently brought back online when complete. Looking towards the automotive industry's future, container-based ECU software can provide a platform to facilitate software updates [23].

Containers, as depicted in Figure 3, represents a newer virtualisation technology which differs from conventional hosted (Type 2) and bare-metal (Type 1) virtualisation technolo-

gies. Individual functionality can be provided by small programs hosted within multiple containers that do not require the heterogeneity provided by full system virtualisation which comes at a cost when considering small scale embedded computing devices.

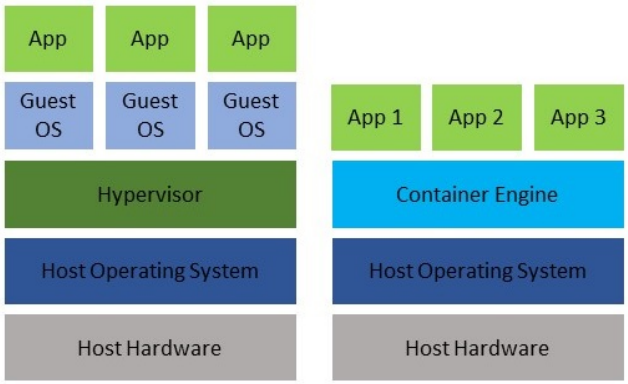

Figure 3. Full System Virtualisation and Container Architectures.

Using ECU container virtualisation in conjunction with OtA updates can address the problems associated with customer disruption and the intrinsic delay between the availability of a new software update and the deployment of that update to the target vehicle. Through vehicle connectivity, new automotive software updates can be pushed or pulled to the target vehicle at any time. For example, a software update pull request could be initiated by the consumer as part of an aftermarket software upgrade or additional automotive functionality. A push update could be applied by the Original Equipment Manufacturer (OEM) or vehicle manufacturer, to resolve an identified software bug or vulnerability thus circumventing the return/recall process and the inherent delays this entails. However, the current primary focus of OtA is on applying a new update when the vehicle is solely offline. The modern motorcar is a system which operates using many subsystems of mixed-criticality. When in operation, there are numerous safety-critical and continual service systems that require a real-time response, for example, engine management and occupant safety systems. The criticality of the software-related issue often determines the required type of software update response. This research has identified three distinct container-based automotive software update modes: offline, online and dynamic.

6.1. Offline Update

Offline updates are initialised when the vehicle is powered down. Once the software update verification and initial container creation are complete, any updates applied are available when the vehicle is next started, similar to a system proposed by [54]. This process mirrors the current return or recall procedure but does not incur any associated disruption to the manufacturer or consumer, or recall costs [19,20,55]. Furthermore, this type of update mechanism can be used for multiple system updates, which may affect numerous subsystems across different automotive domains or involve safety-critical systems that cannot be updated safely with the vehicle in operation.

6.2. Online Update

New software updates can be pushed or pulled to the vehicle using onboard connectivity and applied while the vehicle is powered up but not in operation. The update process is initiated and a new container is created from the new updated image. The affected subsystem is then temporarily shut down before the new container's initialisation with the updated software. This update method could be applied to any automotive system but only where a system's required initialisation does not incur long time delays. For example, small and frequent periodic updates and software security patches would be ideal candidate systems and functions.

6.3. Dynamic Update

DSU do not require the system to be taken offline [56]. As such, they provide an essential service where systems must offer a 100% uptime [57,58]. Taking a system offline to fix bugs, improve system performance, or extend functionality causes delay and disruption. Driverless vehicular technology promises non-stop long-haul trucks and round-the-clock lift-hailing rides and therefore the window to administer software updates become shorter and downtime is a significant disruption [59–61]. For the purposes of this research, a DSU refers to a vehicle sub-system that can be updated and made available once completed, without the vehicle requiring shut down and while it is still in a mode of operation. This type of automotive update is suited ideally to any automotive function which is not involved in vehicle operation or safety. Potential systems could include security software updates and patches, any software relating to autonomous driving functions which are not in operation and passenger-related systems relating to comfort, heating and occupant-vehicle interaction.

7. Implementation and Evaluation of Container-Based Software Updating

Containers offer many benefits to current and future automotive E/E architectures [23]. For example, they provide a standardised environment that can facilitate automotive embedded software updates and their hardware is not fixed to a particular version or type of software. Consolidation is a crucial benefit of container ECUs where multiple containers operate on larger, more resource capable embedded hardware platforms. Containers are constructed from images based on a layered architecture, image layers represent specific data, software, hardware and network configuration parameters.

A container image incorporates one or more layers (as can be seen in Figure 4), which define all required software, libraries and binaries, and configuration settings for any subsequent containers created from that image. Therefore, a container-based ECU must also conform to the three principles of safety, security and transparency:

- Safety—new software containers can be rolled back to the 'last known good' image and known safe containers can be reinstated.
- Security—new container images can be either pulled from an authorised repository to the target vehicle or pushed by the manufacturer. All image layers use, for example, SHA256 encryption and the checksum's validation before the image goes 'live'.
- Transparency—new container images, once validated, can be checked within a sand-box area of the vehicle's automotive E/E architecture before deploying live containers, ensuring the updated system's safe and continued service.

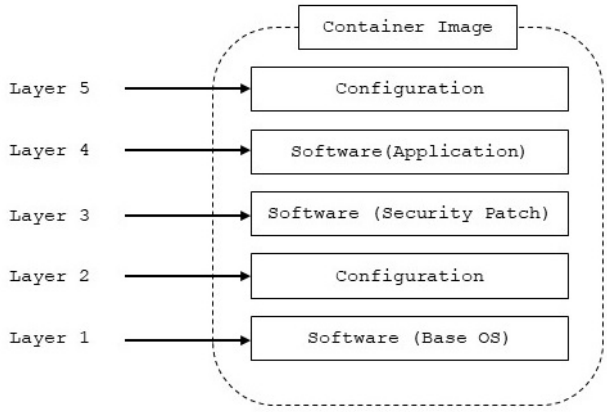

Figure 4. Container Image Layers.

Multiple containers can be created from the same image, which consists of several read-only layers. Any change to the image is specific to a particular layer. If a change is made within the image this alteration is contained within the appropriate layer the change refers to. Small image configuration changes or an update to a specific piece of software

within the image will prompt the system to download only the layers which pertain to those particular changes. A further benefit of this layered approach is the ability to share image layers between separate images. Multiple images that share common layers promote efficiency. A layered design boosts image download speed and minimises the overall image footprint and storage requirements.

To evaluate the benefits of the proposed approach, a test system which closely resembles a typical ECU hardware architecture is required. Previous research into embedded systems and engine management has successfully used the ARM processor-based Raspberry Pi to simulate an ECU [62,63] . The Raspberry Pi version 3B hardware specification used is suitably equipped to host the container software [64–66]. The ECU testbed operating system was Raspbian Lite which used Docker, a container virtualization technology. The high level-programming language chosen is Python as it provides flexibility in accessing the GPIO pins of the Raspberry Pi.

The following test case illustrates how container-based ECUs can promote software update efficiency through image layer duplication. Layer duplication in this context refers to any container image which has the same software version or set of configurations. In this test case two separate image downloads are presented with and without layer duplication. The following results when layer duplication is used show a reduction in time to download and required storage footprints in a potential future automotive software update procedure.

7.1. Individual Container Image Downloads

The example in Figures 5 and 6 illustrates two separate alpine-python images that consist of three individual layers which define the configuration and required software for any container created from that image. The two images alpine-python2 and alpine-python3, both of which share a common Linux based OS (alpine), can be seen in the unique identifier layers cbdbe7a5bc2a and 136e07eea1d6. However, each image has a different version of application software (python2 and python3) with the unique identifier layers f890c68la889 and la5281d56ld0 respectively.

Figure 7 displays the time taken to download and extract both of the alpine images including the total size on disk the two images require in MB. This individual image download is a standard procedure in full binary software updating. If both images are downloaded independently, each image is downloaded in its entirety and bears no relationship with the other image, even though they both share the same underlying OS (alpine).

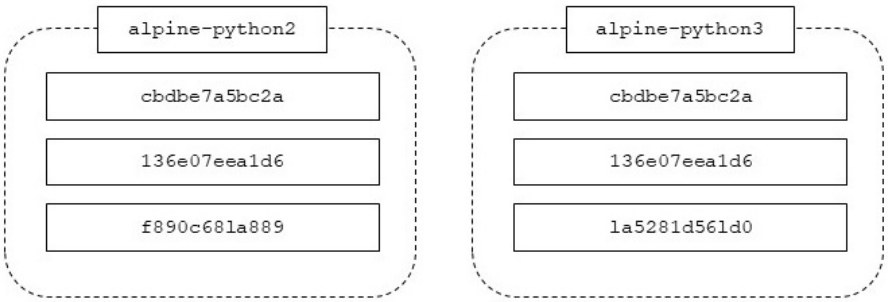

Figure 5. Independent Image Download Layers.

```
master01:~/images docker pull digitalgenius/alpine-python2-pg
Using default tag: latest
latest: Pulling from digitalgenius/alpine-python2-pg
cbdbe7a5bc2a: Pull complete
136e07eea1d6: Pull complete
f890c681a889: Pull complete
Digest: sha256:c4cff01d2c59c13fb729c158bb309194fb469bc28117e70b535bf48d4aac82de
Status: Downloaded newer image for digitalgenius/alpine-python2-pg:latest
docker.io/digitalgenius/alpine-python2-pg:latest
```

```
master01:~/images# docker pull digitalgenius/alpine-python3-pg
Using default tag: latest
latest: Pulling from digitalgenius/alpine-python3-pg
cbdbe7a5bc2a: Pull complete
136e07eea1d6: Pull complete
1a5281d561d0: Pull complete
Digest: sha256:0b1512a41129f127bd512185873e351e89cd647a6b32856c8f4ba4aef1b9921b
Status: Downloaded newer image for digitalgenius/alpine-python3-pg:latest
docker.io/digitalgenius/alpine-python3-pg:latest
```

Figure 6. Independent Image Download.

Image Name	Version	Compressed Image size	Total size on disk after download and extraction	Time taken to download and extract image	Additional download time required
alpine-python	2	115.2MB	883MB	23.4335s	-
alpine-python	3	153.53MB		31.3837s	7.9502s

Figure 7. Image Download, Extraction Times and Storage Requirements.

7.2. Container Image Sharing Downloads

The following test case uses the same alpine-python images. However, before a new image is downloaded, the container host will examine all locally stored images and check each image layer unique ID key which was generated during image creation. Any duplicate layers that share the same image layer ID are ignored. Only those unique layers relating to the new image are downloaded. In this test, an existing image contains two identical layers in common with the new image. Therefore, only one layer of the new image is downloaded, which is observed in Figure 8 (layers indicated by the red outlines in the figure) and Figure 9.

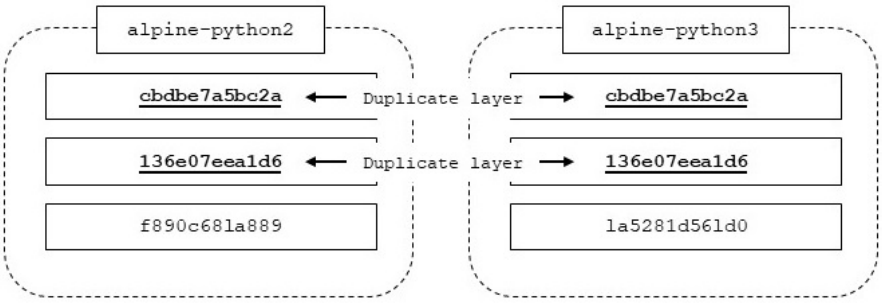

Figure 8. Image Repository Comparison Download.

```
master01:~/images/alpine-python2-pg
default tag: latest
latest: Pulled from digitalgenius/alpine-python2-pg
cbdbe7a5bc2a:
136e07eea1d6:
f890c681a889:
Digest: sha256:c4cff01d2c59c13fb729c158bb309194fb469bc28117e70b535bf48d4aac82de
Status: Current stored image from digitalgenius/alpine-python2-pg:latest
docker.io/digitalgenius/alpine-python2-pg:latest
```

```
master01:~/images# docker pull digitalgenius/alpine-python3-pg
Using default tag: latest
latest: Pulling from digitalgenius/alpine-python3-pg
cbdbe7a5bc2a: Already exists
136e07eea1d6: Already exists
1a5281d561d0: Pull complete
Digest: sha256:0b1512a41129f127bd512185873e351e89cd647a6b32856c8f4ba4aef1b9921b
Status: Downloaded newer image for digitalgenius/alpine-python3-pg:latest
docker.io/digitalgenius/alpine-python3-pg:latest
```

Figure 9. Image Repository Comparison Download.

During the alpine-python3 image download, the two duplicate layers (cbdbe7a5bc2a and 136e07eea1d6) are not downloaded. These two layers represent the alpine OS which both python images share. Only the updated python version layers f890c68la889 and 1a5281d561d0 are pulled from a repository. A repository in this context could be the manufacturer, third party supplier, or one which is stored locally within the vehicle similar to the container image distribution acceleration mechanism proposed by [67–69]. The benefits of layer sharing between container images include reducing the download time for any software update and minimising overall image footprint. Using the performance monitoring tools within the container software the results in Figure 10 highlight the reduced size on disk of both images, which was observed at 44.96%. A reduction in download times between the alpine-python3 images was 5.6346 s across the two tests. Reducing storage requirements for individual software images benefits automotive systems by minimising associated storage hardware costs. Furthermore, layer sharing promotes quicker download speeds which reduces the impact on OtA bandwidths.

Image Name	Version	Layer	Layer Size	Size on Disk after download and extraction	Time Taken to Download and Extract Image	Additional Download Time Required
alpine-python	2	cbdbe7a5bc2a	2.81MB	486MB	23.4335s	-
		136e07eea1d6	38.92MB			
		f890c681a889	73.47MB			
alpine-python	3	1a5281d561d0	111.80MB		25.7491s	5.6346s

Figure 10. Shared Image Layers Size on Disk and Download Times.

7.3. Image Update: Result and Discussion

The test cases in Section 7 examines the benefits surrounding software update size and speed of download. The two alpine-python images share a common OS (alpine); however, the application software within each image comprises of different versions. The test case first examined the specific download and extract times for each image when downloaded

independently. The total download and extract time for the two images was 54.8172 s. The overall size on disk for the two images was recorded at 883 MB. These results provided a baseline for a standard software version upgrade, which is similar to full binary re-flashing techniques. The second part of the test used the same two images but used image layer sharing provided by the container software. There were similarities between the two images as they were both built using the same OS (alpine). As such, because both images share two of the three image layers. Due to layer duplication the OS layers of the new image were not downloaded. This reduced the overall download time by 5.6346 s, which was a 10.28% reduction in total overall download and extraction times. Furthermore, when using container layer sharing, overall storage requirements were reduced considerably by 397 MB or 44.96% when compared with independent image downloads

With each new vehicle model, more and more software are included, adding to the burden of addressing software-related problems. Automotive software update practices have followed the same vehicle recall procedure as when a physical component fails. However, the recall process incurs consumer inconvenience, system downtime, high monetary costs for the manufacturer and potentially reduced brand reputation and customer loyalty. The motor industry is attempting to address the limited available options regarding automotive software updates by using new automotive enabled connectivity.

To illustrate the benefits of container-based software updating, the image download test outlined in Sections 7.1 and 7.2 was conducted. As new software is made available it can be pulled from a remote central software repository. It is envisaged that any future software download would be conducted through automotive connectivity using a static or mobile-based communication network and the Internet, or a central repository which is either hosted with the vehicle manufacturer or third-party supplier.

Consideration has been given to develop the experimental setup to represent an actual ECU hardware environment. The necessary container image layers used within this test case were downloaded over the existing University Internet connection media. Hence, the reported time to download is as it would be in a real scenario using ground/location-based WiFi. However, it is envisioned that vehicular on-board connectivity solutions could be used to connect to other communication technologies including DSRC or the LTE network (4G and 5G). These could be used to provide a mobile connection and download method. Furthermore, within this test-case scenario, bandwidth is not a primary consideration as any future software download can take place over a time-period, unless in certain, very rare, safety critical scenarios, which is not within the scope of the proposed approach. Container-based software image layers have minimum storage overhead compared to the traditional ECU architecture-based updates, as only layers pertaining to the update is downloaded as opposed to all the layers within the new software image. The efficiencies provided by container image layers and layer duplication can minimise the size requirements of redundant storage associated with future software updates, given that component costs can escalate sharply due to high vehicle production numbers and the lifespan of a vehicle model, both of which are an important consideration in ECU design.

8. Conclusions

Container-based ECUs, as proposed and evaluated within this paper, can promote automotive software updates, particularly OtA software updates in conjunction with vehicle connectivity. This can reduce significantly the need to recall vehicles when encountering a software-related problem because the new software can be deployed to a target vehicle remotely, as and when required. Notably, by using containers in this way, overall vehicle security can be maintained and any potential software vulnerabilities can be addressed. This has significant implications for the automotive industry.

The number of vehicles recalls in the U.S. associated with a software fault has risen dramatically by 1400% since 2010. Vehicle recalls are highly disruptive to the consumer, expensive for the manufacturer and can, in some cases, reduce brand reputation. It is estimated that resolving a software-related error post-sale is 30 times more expensive than

compared with fixing the same issue during the early stages of the SDLC. Compounding this, the current automotive software update practices and procedures are not keeping pace with the rapid increase in the number of lines of software code. The research findings presented in this paper demonstrate that these problems may be overcome by using container-based ECUs, whereby errors, bugs and vulnerabilities cannot only be addressed promptly and effectively, but also throughout the vehicle's lifetime. Additionally, container-based OtA software updates can significantly reduce disruption to consumers. Consumers are also able to incorporate additional or new functionality into a container-based ECU, which can generate additional revenue for the manufacturer in terms of their aftermarket sales. The proposal holds promise of a paradigm shift in the automotive E/E architecture and the way software update is performed.

Author Contributions: Conceptualization, N.A. and L.D.; methodology, N.A., L.D. and D.P.; software, N.A.; validation, N.A., L.D. and D.P.; investigation, N.A.; data curation, N.A.; writing—original draft preparation, N.A.; writing—review and editing, N.A., L.D. and D.P.; visualization, N.A.; supervision, L.D. and D.P.; project administration, L.D.; funding acquisition, L.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The submitted research is part of N.A.'s PhD thesis and the PhD Bursary has been funded internally by De Montfort University, UK.

Data Availability Statement: Data available on request due to restrictions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bereisa, J. Applications of Microcomputers in Automotive Electronics. *IEEE Trans. Ind. Electron.* **1983**, *30*, 87–96. [CrossRef]
2. Haghighatkhah, A.; Banijamali, A.; Pakanen, O.P.; Oivo, M.; Kuvaja, P. Automotive software engineering: A systematic mapping study. *J. Syst. Softw.* **2017**, *128*, 25–55. [CrossRef]
3. Petri, R.; Springer, M.; Zelle, D.; McDonald, I.; Fuchs, A.; Krauß, C. Evaluation of lightweight TPMs or automotive software updates over the air. In Proceedings of the World's Leading Automotive Cyber Security Conference, Detroit, MI, USA, 1–2 June 2016.
4. Breitschwerdt, D.; Cornet, A.; Kempf, S.; Michor, L.; Schmidt, M. *The Changing Aftermarket Game and How Automotive Suppliers can Benefit from Arising Opportunities*; McKinsey & Company: New York, NY, USA, 2017.
5. Coppola, R.; Morisio, M. Connected car: Technologies, issues, future trends. *ACM Comput. Surv.* **2016**, *49*, 1–36. [CrossRef]
6. Riggs, C.; Rigaud, C.E.; Beard, R.; Douglas, T.; Elish, K. A Survey on Connected Vehicles Vulnerabilities and Countermeasures. *J. Traffic Logist. Eng.* **2018**, *6*, 11–16. [CrossRef]
7. Levitt, J. *Complete Guide to Preventative and Predictive Maintenance*, 1st ed.; Industrial Press: New York, NY, USA, 2003.
8. Hangal, S.; Lam, M.S. Tracking down software bugs using automatic anomaly detection. In Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, Orlando, FL, USA, 25 May 2002; pp. 291–301.
9. Onuma, Y.; Terashima, Y.; Nozawa, M. Improved Software Updating for Automotive ECUs. *Atlanta* **2016**, *2*, 319–324.
10. Noergaard, T. *Embedded Systems Architecture A Comprehensive Guide for Engineers and Programmers*; Elsevier: Oxford, UK, 2005.
11. Ebert, C.; Jones, C. Embedded software: Facts, figures, and future. *Computer* **2009**, *42*, 42–52. [CrossRef]
12. Heiser, G. Hypervisors for consumer electronics. In Proceedings of the 2009 6th IEEE Consumer Communications and Networking Conference, Las Vegas, NV, USA, 10–13 January 2009; pp. 1–5.
13. Sax, E.; Reussner, R.; Guissouma, H.; Klare, H. *A Survey on the State and Future of Automotive Software Release and Configuration Management*; KIT: Amsterdam, The Netherlands, 2017; pp. 1–19.
14. Martyn, A. Automatic Braking Systems in Some Nissan Rogues are Going Rogue. 2019. Available online: https://www.consumeraffairs.com/news/automatic-braking-systems-in-some-nissan-rogues-is-going-rogue-safety-group-says-032719.html (accessed on 12 March 2021).
15. Buckland, K. Toyota Issues Second Prius Recall in a Month on Crash Risk. 2018. Available online: https://www.bloomberg.com/news/articles/2018-10-05/toyota-issues-second-prius-recall-in-a-month-on-crash-risk (accessed on 23 January 2021).
16. Shepardson, D. Fiat Chrysler Recalls 5.3 Million Vehicles for Cruise Control Defect. 2018. Available online: https://www.reuters.com/article/us-fiat-chrysler-recall/fiat-chrysler-recalls-4-8-million-u-s-vehicles-for-cruise-control-defect-idUSKCN1IQ1QY (accessed on 14 January 2021).
17. Drolia, U.; Wang, Z.; Pant, Y.; Mangharam, R. AutoPlug: An automotive test-bed for electronic controller unit testing and verification. In Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), Washington, DC, USA, 5–7 October 2011; pp. 1187–1192.

18. Lönn, H.; Freund, U. Automotive architecture description languages. In *Automotive Embedded Systems Handbook;* CRC Press: Boca Raton, FL, USA, 2009.
19. Furst, S.; Bechter, M. AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform. In Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), Toulouse, France, 28 June–1 July 2016; pp. 215–217.
20. Onuma, Y.; Nozawa, M.; Terashima, Y.; Kiyohara, R. Improved software updating for automotive ECUs: Code compression. In Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Atlanta, GA, USA, 10–14 June 2016; Volume 2, pp. 319–324.
21. Braun, L.; Armbruster, M.; Gauterin, F. Trends in vehicle electric system design: State-of-the Art Summary. In Proceedings of the 2015 IEEE Vehicle Power and Propulsion Conference (VPPC), Montreal, QC, Canada, 19–22 October 2015; pp. 1–6.
22. Rouse, M. OTA Update (Over-the-Air Update). 2018. Available online: https://searchmobilecomputing.techtarget.com/definition/OTA-update-over-the-air-update (accessed on 4 November 2019).
23. Ayres, N.; Deka, L.; Passow, B. Virtualisation as a Means for Dynamic Software Update within the Automotive E/E Architecture. In Proceedings of the 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Leicester, UK, 19–23 August 2019; pp. 154–157.
24. Steinkamp, N.; Levine, R.; Roth, R. Automotive Defect and Recall Report, Stout Risius Ross. 2019. Available online: https://www.stout.com/zh-cn/insights/report/2019-automotive-defect-and-recall-report (accessed on 19 February 2021).
25. Lions, J.L.; Luebeck, L.; Fauquembergue, J.L.; Kahn, G.; Kubbat, W.; Levedag, S.; Mazzini, L.; Merle, D.; O'Halloran, C. Ariane 5 Flight 501 Failure Report by the Inquiry Board. 1996. Available online: http://sunnyday.mit.edu/nasa-class/Ariane5-report.html (accessed on 15 January 2021).
26. Lin, P.-S.; Wang, Z.; Guo, R. *Impact of Connected Vehicles and Autonomous Vehicles on Future Transportation*; ASCE Library: Hsinchu, Taiwan, 2016.
27. Thati, V.B.; Vankeirsbilck, J.; Pissoort, D.; Boydens, J. Hybrid Technique for Soft Error Detection in Dependable Embedded Software. In Proceedings of the 2019 IEEE XXVIII International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 12–14 September 2019.
28. Miller, C.; Valasek, C. *Adventures in Automotive Networks and Control Units*; DEF CON: Las Vegas, NV, USA, 2013.
29. Woo, S.; Jo, H.J.; Kim, I.S.; Lee, D.H. A Practical Security Architecture for in-vehicle CAN-FD. *IEEE Trans. Intell. Transp. Syst.* **2016**, *17*, 2248–2261. [CrossRef]
30. Automotive IQ. Automotive Software Development Reliability and Safety. 2017. Available online: https://www.automotive-iq.com/electrics-electronics/reports/automotive-software-development-reliability-safety-1 (accessed on 21 February 2021).
31. Boucherat, X. Make it Safe, Make it Profitable: The Writing's on the Wall for the Connected Car. 2016. Available online: https://www.automotiveworld.com/articles/make-safe-make-profitable-writings-wall-connected-car/ (accessed on 10 January 2021).
32. Howden, J.; Maglaras, L.; Ferrag, M.A. The Security Aspects of Automotive Over-the-Air Updates. *Int. J. Cyber Warf. Terror.* **2020**, *10*, 64–81. [CrossRef]
33. Happel, A.; Ebert, C. *Security in Vehicle Networks of Connected Cars*; Springer: Wiesbaden, Germany, 2015.
34. Alam, M. The Software Defined Car: Convergence of Automotive and Internet of Things. In *Wireless World in 2050 and Beyond: A Window into the Future!* Springer Series in Wireless Technology; Springer: Berlin, Germany, 2016; pp. 83–92.
35. Chowdhury, T.; Lesiuta, E.; Rikley, K.; Lin, C.W.; Kang, E.; Kim, B. Safe and Secure Automotive Over-The-Air Updates. In *Computer Safety, Reliability and Security*; Springer: Cham, Switzerland, 2018; pp. 172–187.
36. Quain, J.R. With Benefits and Risks Sofwtare Updates are Coming to the Car. 2018. Available online: https://digitaltrends.com/cars/over-the-air-software-updates-cars-pros-cons/ (accessed on 16 February 2021).
37. Parnas, D.L. Software aging. In Proceedings of the 16th International Conference on Software Engineering, orrento, Italy, 16–21 May 1994; pp. 279–287.
38. Breitschwerdt, D.; Cornet, A.; Michor, L.; Müller, N.; Salmon, L. Performance and disruption—A perspective on the automotive supplier landscape and major technology trends. *Hg. v. McKinsey and Company, zuletzt geprüft am* **2016**, *7*, 2018.
39. Holmes, F. Over-the-Air Updates Moving from 'Nice to Have' to 'Vital'. 2018. Available online: https://www.automotiveworld.com/articles/over-the-air-updates-moving-from-nice-to-have-to-vital/ (accessed on 11 January 2021).
40. Halder, S.; Ghosal, A.; Conti, M. Secure over-the-air software updates in connected vehicles: A survey. *Comput. Netw.* **2020**, *178*, 107343. [CrossRef]
41. NHTSA. National Highway Traffic Safety Administration. 2020. Available online: Https://nhtsa.org (accessed on 10 January 2021).
42. Mckenna, D.; Automotive, B.U.; Semiconductors, N.X.P. Making Full Vehicle OTA Updates a Reality. NXP. 2016. Available online: http://www.nxp.com/automotivesecurity (accessed on 4 March 2021).
43. Odat, H.A.; Ganesan, S. Firmware over the air for automotive, fotamotive. In Proceedings of the IEEE International Conference on Electro/Information Technology, Milwaukee, WI, USA, 5–7 June 2014; pp. 130–139.
44. Broy, M. *Challenges in Automotive Software Engineering*; ACM: Shanghai, China, 2006; pp. 33–42.
45. Kopetz, H. *Design Principles for Distributed Embedded Applications*; Springer: New York, NY, USA, 2011.

46. Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H.; Savage, S. *Comprehensive Experimental Analyses of Automotive Attack Surfaces*; USENIX: San Francisco, CA, USA, 2011.
47. Guo, J.; Balon, N. *Vehicular Ad Hoc Networks and Dedicated Short-Range Communication*; University of Michigan: Ann Arbor, MI, USA, 2006.
48. Vegni, A.M.; Biagi, M.; Cusani, R. Smart Vehicles, Technologies and Main Applications in Vehicular Ad hoc Networks. 2013. Available online: https://www.intechopen.com/books/vehicular-technologies-deployment-and-applications/smart-vehicles-technologies-and-main-applications-in-vehicular-ad-hoc-networks (accessed on 2 March 2021).
49. Patterson, A. The Evolution of Embedded Architectures for the Next Generation of Vehicles. *ATZelektronik Worldwide* **2017**, *12*, 26–33. [CrossRef]
50. Stegar, M.; Boano, C.A.; Niedermayr, T.; Karner, M.; Hillebr, J.; Roemer, K.; Rom, W. An Efficient and Secure Automotive Wireless Software Update Framework. *IEEE Trans. Ind. Informatics* **2017**, *14*, 2181–2193. [CrossRef]
51. Gissler, A. Automotive World Ltd, The Auto Industry must Get Connected to Fend Off Marginalisation. 2016. Available online: https://www.automotiveworld.com/articles/auto-industry-must-get-connected-fend-marginalisation/ (accessed on 15 December 2020).
52. Habermas, C.S. General Motors Corporation. Method and System for Remote Reflash. U.S. Patent 7,366,589 B2, 29 April 2008.
53. Link, M.C.; Hughes Telematics, Inc. Methods and Systems for Software Upgrades. U.S. Patent US 2009/0119657 A1, 7 May 2009.
54. Tobolski, T.; Esselink, C.E.; Westra, M.R.; Ellis, J.T. Silent in-Vehicle Software Updates. U.S. Patent No. US10140109B2, 27 November 2018.
55. Herberth, R.; Körper, S.; Stiesch, T.; Gauterin, F.; Bringmann, O. Automated Scheduling for Optimal Parallelization to Reduce the Duration of Vehicle Software Updates. *IEEE Trans. Veh. Technol.* **2019**, *68*, 2921–2933. [CrossRef]
56. Seifzadeh, H.; Abolhassani, H.; Moshkenani, M.S. A survey of dynamic software updating. *J. Softw. Evol. Process.* **2013**, *25*, 535–568. [CrossRef]
57. Neamtiu, I.; Hicks, M.; Stoyle, G.; Oriol, M. Practical dynamic software updating for C. *ACM Sigplan Not.* **2006**, *41*, 72–83. [CrossRef]
58. Hayden, C.M.; Smith, E.K.; Hardisty, E.A.; Hicks, M.; Foster, J.S. Evaluating Dynamic Software Update Safety Using Systematic Testing. *IEEE Trans. Softw. Eng.* **2012**, *28*, 1340–1354. [CrossRef]
59. Hörl, S. Agent-based simulation of autonomous taxi services with dynamic demand responses. *Procedia Comput. Sci.* **2017**, *109*, 899–904. [CrossRef]
60. Shankwitz, C. *Long-haul Truck Freight Transport and the Role of Automation: Collaborative Human—Automated Platooned Trucks Alliance (CHAPTA)*; Western Transport Institute: Bozeman, MT, USA, 2017.
61. Simpson, J.R.; Mishra, S.; Talebain, A.; Gollias, M. An Estimation of the Future Adoption Rate of Autonomous Trucks by Freight Organizations. *Res. Transp. Econ.* **2019**, *76*, 100737. [CrossRef]
62. Walter, J.; Fakih, M.; Grüttner, K. Hardware-based real-time simulation on the raspberry pi. In Proceedings of the 2nd Workshop on High Performance and Real-time Embedded Systems, Vienna, Austria, 20 January 2014.
63. Vaughan, A.; Bohac, S.V. An extreme learning machine approach to predicting near chaotic HCCI combustion phasing in real-time. *arXiv* **2013**, arXiv:1310.3567.
64. Krylovskiy, A. Internet of things gateways meet Linux containers: Performance evaluation and discussion. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 222–227.
65. Hurst, W.; Shone, N.; El Rhalibi, A.; Happe, A.; Kotze, B.; Duncan, B. Advancing the micro-CI testbed for IoT cyber-security research and education. *Cloud Comput.* **2017**, *2017*, 139.
66. Johnston, S.J.; Cox, S.J. The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams. *Electronics* **2017**, *3*, 51. [CrossRef]
67. Suarez, A.J.; Windsor, S.K.; Hayrapetyan, N.; Gerdesmeier, D.R.; Prakash, P.K.; Amazon Technologies Inc. Software Container Registry Container Image Deployment. U.S. Patent 10,002,247, 19 June 2018.
68. Suarez, A.J.; Windsor, S.K.; Hayrapetyan, N.; Gerdesmeier, D.R.; Prakash, P.K.; Amazon Technologies Inc. Software Container Registry Service. U.S. Patent 10,261,782, 16 April 2019.
69. Zhao, A.; Cao, Y.; Peng, L.; Junping, Z.H.A.O.; Durazzo, K.; EMC IP Holding Co LLC. Container Image Distribution Acceleration. U.S. Patent 10,291,706, 14 May 2019.