Incremental Over-the-Air Update for Software-Defined Vehicles with C-V2X

Yichen Luo and Sidi Lu
William & Mary, Williamsburg, VA, USA, {yluo11, sidi}@wm.edu

***Abstract*—Software-defined vehicles (SDVs) rely on intricate software systems distributed across multiple electronic control units (ECUs), making errors inevitable and costly to address via conventional recalls. Software over-the-air (OTA) update offers a real-time, cost-efficient solution while enabling software customization. This paper presents a comprehensive OTA framework for SDVs, utilizing a robotic vehicle and an industry-grade roadside unit (RSU). The framework leverages Docker to create isolated environments and Docker Compose to manage rolling updates, minimizing disruptions and enabling rollbacks. We also propose RSU- and vehicle-side mechanisms for incremental update and evaluate the performance of Cellular Vehicle-to-Everything (C-V2X), including V2X and 5G, alongside Wi-Fi for OTA transmission. Our study of model weight (ResNet) and software package (YOLO, UFAST) updates shows that C-V2X achieves over 30× faster transmission than Wi-Fi. Using 5G within C-V2X further reduces transmission by up to 2339× compared to V2X, establishing it as a highly effective OTA solution. Incremental updates enhance efficiency by cutting delays by 2× versus full updates, with a 32KB chunk size yielding the highest success rate and shortest times across distances. Incremental update also reduces transmission by up to 5000× using C-V2X, enabling package downloads even while the SDV is in motion.**

***Index Terms*—Software-defined Vehicles, OTA Update, C-V2X**

I. Introduction

Software-defined vehicles (**SDVs**) are characterized by their reliance on safety-critical software to manage and control core functionalities, allowing for continuous update and improvements throughout their lifecycle [1]. A category within SDVs, autonomous vehicles rely on over 650 million lines of code distributed across more than 150 electronic control units (**ECUs**). This complexity has led to an increase in software-related issues and expensive recalls. A prominent example is Tesla's recall of 3.6 million vehicles due to crash risks linked to its driver-assistance software, which resulted in 17 fatalities, 736 crashes, and financial losses of $532 million in the first quarter of 2023 [2]. Given the associated risks to human life, software over-the-air (**OTA**) update has become a critical solution, enabling real-time error correction, minimizing recall rates, and facilitating continuous software improvements and customization throughout the vehicle's lifecycle.

A. Software OTA Update: Overview and Advancements

Software OTA update offers an efficient and cost-effective solution for software maintenance in vehicles [3]. Tesla's update process involves two main phases: 1) the download phase and 2) the installation phase [4]. Although driving is allowed during the download phase, the process may be disrupted if the vehicle moves beyond the range of a stable local network signal, like Wi-Fi, emphasizing it is limited reliability over distance. This highlights the need for more reliable communication technologies (detailed in Sec. I-B)

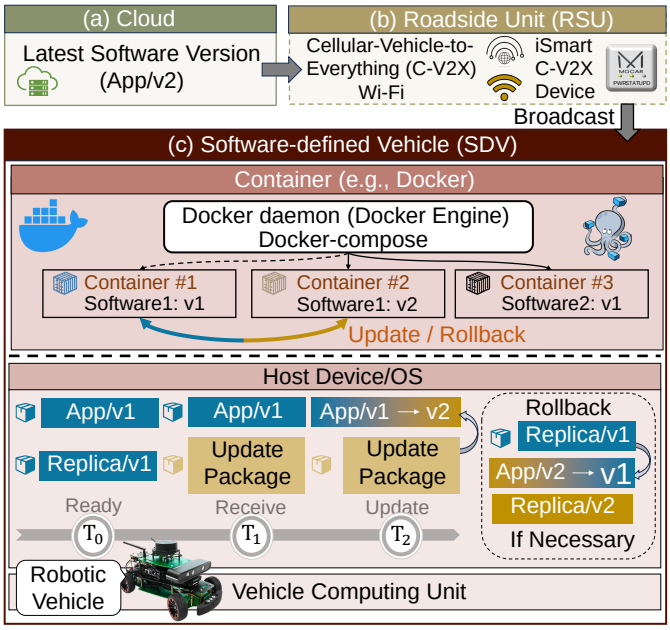

Fig. 1: A simplified representation of the software OTA update framework for SDVs, comprising three components: (a) the cloud, (b) the roadside unit (RSU), and (c) the SDV, demonstrated using a robotic vehicle as an example. The SDV employs containerization technologies (e.g., Docker and Docker-compose) integrated with its host device and operating system (Host Device/OS). Initially, the SDV operates on App/v1, running container 1 (Software1:v1) within the Docker environment, alongside other applications such as Software2:v1. The update process begins on the Host Device by creating the current software version replica (Replica/v1), enabling rollback if necessary. The SDV then receives incremental update packets (App/v2) from the RSU via Cellular-Vehicle-to-Everything (C-V2X) or Wi-Fi communication. Upon detecting App/v2, Docker Compose triggers a process to stop container 1 running Software1:v1 and launches a new container to deploy Software1:v2. The retained Replica/v1 provides the rollback mechanism, allowing the system to revert to App/v1 in the event of an update failure or any other issue.

Moreover, while OTA update is widely implemented in consumer electronics (smartphones, laptops, and IoT devices), directly transferring traditional methods like the waterfall model [5] to SDVs proves inadequate. This limitation arises from the complexity of vehicle hardware, which incorporates over 150 specialized ECUs [1], as well as the intricacy of software systems designed to manage distinct tasks with varying real-time requirements [6], involving real-time tasks (e.g., powertrain control) and non-real-time tasks (e.g., infotainment systems). Additional challenges stem from variable operational conditions, such as inconsistent connectivity in dynamic

driving scenarios [7]. Therefore, a tailored vehicle-RSU-cloud framework (Fig.1) is essential for decoupling hardware from software and facilitating efficient software update (Sec.II).

B. Mainstream Vehicular Communication Mechanism

Efficient communication is essential for seamless software OTA updates in SDVs. Technologies like LTE/5G/6G, Wi-Fi, Dedicated Short Range Communication (**DSRC**), and Cellular-Vehicle-to-Everything (**C-V2X**) enable SDVs to interact with other vehicles, RSUs, and the cloud. Despite over a decade of use, DSRC suffers from low throughput and limited coverage. In contrast, C-V2X offers higher reliability, faster data rates, longer range, and lower latency, meeting the stringent demands of advanced vehicular applications [8].

Building on these advantages, field tests by Ford and Qualcomm in Ann Arbor and San Diego show that C-V2X outperforms DSRC, enhancing connectivity and traffic safety. Consequently, the Federal Communications Commission (**FCC**) reallocated DSRC's frequency band to C-V2X, establishing it as a leading vehicular communication technology widely adopted in the U.S., China, and Europe [9]. C-V2X integrates the V2X network (PC5 interface) with 4G/5G (Uu interface) at a 5.9GHz frequency, where V2X enables direct, stable communication without third-party intermediaries, and 4G/5G supports high-speed data transmission.

C. Unique Challenges and Contributions

Challenge 1: Software-hardware decoupling and modularity. Premium vehicles often contain over 150 million lines of code [1], distributed across heterogeneous hardware platforms developed by various Original Equipment Manufacturers (**OEMs**). Traditionally, the tight coupling between hardware and software causes significant compatibility issues, as software update frequently necessitate corresponding hardware modifications. To address them, decoupling hardware and software offers a solution to enhance modularity, facilitate seamless third-party integration, and ensure software isolation to prevent cross-system interference. Moreover, this approach should aim to support extended software lifecycle management (spanning approximately 15 years) and facilitate efficient OTA update for upgrades and customization.

Challenge 2: Network limitations and localized update. Vehicles often operate in areas with limited network coverage, such as remote regions and underground garages. Also, channel conflicts can occur due to the allocation of resources among users of varying types. Therefore, relying solely on cloud-based OTA update, where SDVs directly retrieve software from the cloud, may fail to ensure smooth and efficient software delivery due to network inconsistencies. Moreover, automotive regulations vary across countries and states, encompassing factors such as privacy and AI behavior. For instance, autonomous vehicles must comply with state-specific guidelines defining how AI-driven systems should respond in particular situations. Thus, OTA update should be tailored to local requirements rather than using uniform global software. Incorporating RSUs to deliver region-specific update is crucial for efficient and localized software distribution.

Challenge 3: Impact of transmission distance and chunk size. Unlike stationary devices, SDVs are in constant motion, causing fluctuating distances between vehicles and RSUs during OTA update. These variations introduce challenges due to fluctuating network conditions that can compromise update reliability. Additionally, selecting an appropriate chunk size for software transmission poses another difficulty. Smaller chunk sizes increase transmission frequency and overall transmission time, whereas larger chunks, although they can theoretically reduce protocol overhead, may lead to network congestion due to buffer overflows, increased packet loss, and retransmissions, ultimately resulting in prolonged transmission time. Thus, understanding transmission distance and chunk size effects is essential for optimizing OTA update performance.

Within this study, we design and evaluate a software OTA update framework (Fig. 1) that operates in two distinct phases: download and installation. We conduct case studies involving the transmission of i) model weights from three ResNet models, which are widely adopted deep convolutional neural networks frequently used in perception tasks for autonomous driving and ii) differential packages for YOLO (object detection) [10] and UFAST (lane detection) [11]. These packages are delivered using C-V2X technology, which integrates V2X and 5G capabilities, as well as Wi-Fi. The primary contributions of this work are outlined as follows:

- To address Challenge 1, we design and implement a software OTA update framework using a robotic vehicle and an industry-grade RSU equipped with an iSmart C-V2X device. The framework utilizes Docker to encapsulate software in isolated containers, enabling software-hardware decoupling and modularization. Docker Compose orchestrates multi-container applications and manages their lifecycles. Upon detecting a new version on the host device, the update is applied to the previous version, and the updated version replaces the old one via a rolling update, ensuring seamless transitions. It supports rollback if issues arise.
- To address Challenge 2, we propose two mechanisms for incremental update. On the RSU side, update is initiated by either i) identifying and transmitting updated model weights or ii) generating differential files for software packages, which encapsulate changes and path information to reduce package size and optimize transmission efficiency. On the SDV side, a layered approach is employed, enabling the host operating system to detect update, apply the required modifications to the current software version, and seamlessly deploy the latest version within the Docker environment.
- To address Challenge 3, we analyze the impact of SDV-RSU distance and chunk size on chunk reception ratio (CRR). CRR decreases as distance increases, with small and large chunk sizes showing greater declines compared to medium-sized chunks. The 32KB chunk size achieves the highest CRR and faster transmission across distances. Incremental update further reduce transmission time by 5000$\times$ using C-V2X, enhancing the feasibility of downloading update packages while the SDV is in motion.

II. Framework and Mechanism Overview

This section presents a detailed schematic of the proposed software OTA update framework to address Challenge 1. Additionally, two incremental update mechanisms are proposed for the RSU and vehicle sides to tackle Challenge 2.

A. Framework Description

The development of vehicle software models relies on continuous data collection, with the cloud aggregating global data to generate updated software versions distributed to RSUs. Figure 2 illustrates the framework's structure and outlines the steps for updating fixed software while ensuring other software remains unaffected, consisting of two components.

*1) **RSU**:* Once the latest software version is retrieved from the cloud, the RSU identifies either i) the updated software model weights or ii) differential packages along with a path file (as shown in the top-left of Fig.2). This involves comparing versions to pinpoint changes, including file updates, additions, or deletions. The RSU then broadcasts these updates using Wi-Fi or, leveraging iSmart C-V2X devices (SectionII-B), through V2X and 5G communication channels.

C-V2X communication setup. Ensuring successful transmission through C-V2X requires the generation of "customized information messages" for RSU communication. Since the update package exceeds the single message size limitation, data serialization is performed to split the file into manageable segments. These serialized segments are then transmitted using "customized information messages". This process consists of three essential steps, as illustrated in the bottom-left of Fig. 2

(1) Serializing data and embedding messages. Complex data types, such as model weights and software packages, are serialized into a byte-stream format to align with the C-V2X protocol. The serialized data is embedded into "customized information messages," ensuring compliance and enabling efficient data transfer with minimal packet loss.

(2) Implementing Stop-wait protocol. To address throughput asymmetry in C-V2X communication, where sending exceeds receiving throughput [12], we implement a stop-wait protocol with a 0.1-second delay. This method follows SAE J2735 Traveler Information Message (**TIM**) standard [13], reducing retransmissions and ensuring reliable flow. The RSU broadcasts TIMs containing real-time alerts such as traffic updates and road sign information to nearby vehicles.

(3) Compile source code for message transmission. Due to the absence of external libraries and dependencies on the RSU device, "customized information messages" are compiled into a 64-bit executable file for deployment on the RSU.

*2) **Vehicle**:* To decouple hardware and software, we deploy Docker and Docker Compose on the vehicle's computing unit (NVIDIA Jetson NX) running Ubuntu 20.04 as the host operating system. Given the Jetson architecture, JetPack is utilized alongside NVIDIA's L4T-ML (Linux for Tegra - Machine Learning) package, a container image designed to simplify the deployment of machine learning models. This setup supports various vehicle applications, including object detection and lane detection, while ensuring flexibility and modularity through containerization.

Docker and Docker Compose [14] are used (bottom-right of Fig. 2) to avoid dependency conflicts and ensure that updates to specific vehicle software do not affect other applications. Docker provides isolated environments for running software in containers built from layered images, encapsulating required dependencies and configurations to prevent conflicts. Modify-

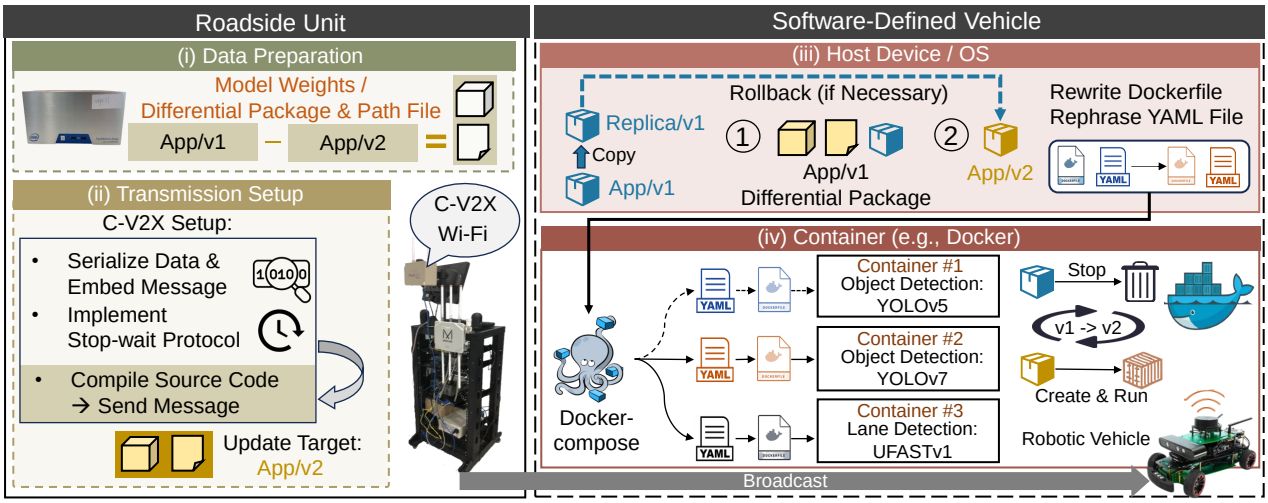

Fig. 2: The software OTA update framework integrates the RSU (left) and the SDV (right), utilizing Docker alongside the host device and operating system (Host Device/OS) to facilitate efficient incremental update. The process is divided into three main stages: pre-update configuration, update process, and container update. Before the update begins, the host device operates on App/v1 and runs associated software containers, such as YOLOv5 in Container 1 and UFASTv1 in Container 3, all managed through Docker Compose to ensure consistent and stable operation. The update process starts with the RSU preparing the necessary data, including model weights, differential packages, and file paths for App/v2. These updates are serialized, embedded into messages, and transmitted to the SDV via C-V2X or Wi-Fi using a stop-wait protocol. On the SDV side, the host device duplicates the current version (App/v1) into Replica/v1 to enable rollback if needed. The differential package is then applied to transition App/v1 to App/v2. Once the new version (App/v2) is detected, the SDV updates its Docker configuration by rewriting the Dockerfile and rephrasing the YAML file for Docker Compose. Docker Compose deactivates the container running YOLOv5 (Software1:v1) and deploys a new container to run YOLOv7 (Software1:v2). This process ensures seamless integration of the updated software while preserving rollback capabilities to address potential issues.

ing the Dockerfile triggers Docker to rebuild only the affected containers, ensuring updates are applied seamlessly and independently without impacting other software functionalities.

Docker Compose simplifies the management of multiple containers by coordinating their configurations and execution through a YAML file. With the addition of automation scripts, containers can be automatically rebuilt when Dockerfiles are updated, streamlining the process of deploying the latest software images for rolling update. Furthermore, in the event of issues with an update, the system can roll back to a previous software version (i.e., a stored replica) on the host device, ensuring stability and reliability throughout the update process.

B. Hardware Platform Description

Figure 3 visualizes the essential hardware components of our OTA update framework, featuring the RSU with the Mocar C-V2X device on the left and the robotic vehicle on the right.
iSmart C-V2X Device. This device offers advanced mobile communication capabilities, operating with a bandwidth of 10MHz or 20MHz and achieving communication latency below 20ms. Equipped with six antennas (V2X1-4 and 5G1-2), it supports both V2X and 5G networks (Sec I).
Robotic Vehicle. The robotic vehicle is powered by an NVIDIA Jetson Orin NX as its onboard computing unit, featuring 16GB of memory. This edge device from NVIDIA's Jetson series offers up to 100 Tera Operations Per Second (TOPS) of AI performance, making it highly suitable for advanced computing tasks.

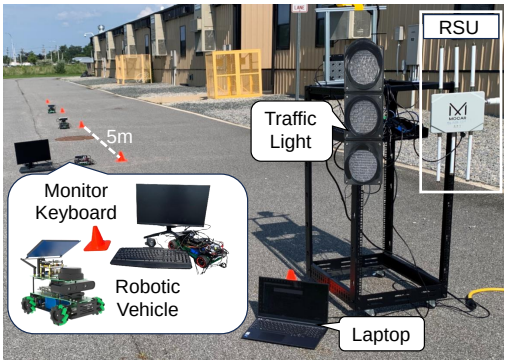

Fig. 3: The hardware overview of the RSU and the robotic vehicle.

C. Incremental Update Mechanisms on the RSU and Vehicle

To address Challenge 2, we introduce incremental update mechanisms for both the RSU and the vehicle. As a case study, we explore the transmission of i) model weights from three ResNet models and ii) differential software packages for YOLO (object detection) [10] and UFAST (lane detection) [11]. The RSU is responsible for retrieving these data types and transmitting them to the SDV, while the vehicle mechanism focuses on successfully executing the received differential packages. These mechanisms are detailed as follows.

1) Incremental Update Mechanism on the RSU: For model weights, the RSU retrieves the latest locally stored weights, ready for deployment. For software packages, the RSU generates differential files encapsulating only the changes between the current and updated software versions. Accompanying these files is a path file specifying modification paths and providing instructions for transitioning from the old version to the new one. This path file guides the SDV's host OS in applying the differential files and updating the previous version. Both differential files and the path file are transmitted via the broadcast, ensuring efficient and reliable update.

2) Incremental Update Mechanism on the SDV: For model weights, the process begins with monitoring a designated directory on the host system for changes, facilitated by a script that detects updates and triggers appropriate actions when new weights are added. The docker-compose.yml file is configured to mount this directory as a volume, ensuring the container can access the updated files. Inside the container, the application logic is programmed to dynamically load the new weights at runtime. Once the monitoring script detects updated weights, it restarts the container to integrate the latest update seamlessly. Finally, the process is validated by placing a new weights file in the monitored directory and verifying that the script detects the update, the container restarts successfully, and the application correctly loads the updated weights.

For software packages, the host device duplicates the current version (App/v1) as Replica/v1 to ensure rollback capability if needed. The differential file, containing only essential changes, is applied to App/v1 to update it to the new version (App/v2). After the update, the SDV OS updates the Dockerfile and modifies the YAML configuration for Docker Compose. Docker Compose then deactivates the container running the old software version (e.g., Software1:v1) and launches a new container for the updated version (e.g., Software1:v2).

III. OTA Update Results in Wi-Fi

To address Challenge 3, this section examines the impact of SDV-RSU distance and chunk size on the one-time transmission success rate, utilizing the widely adopted Wi-Fi protocol, which is commonly used for building local networks [1].

A. Evaluation Metrics

We define the chunk reception ratio (CRR) as an evaluation metric to assess the reliability of data transfers. It is calculated using the formula: $CRR = \frac{N_{\text{received}}}{N_{\text{total}}} \times 100\%$ where N_{received} represents the number of chunks successfully received, and N_{total} denotes the total number of chunks transmitted.

B. Quantitative Analysis of Distance and Chunk Size

The ResNet-18 model weights (45MB) are broadcasted 15 times at distances of 5m, 10m, and 15m to evaluate performance. Four chunk sizes, 8KB, 16KB, 32KB, and 65KB, are tested. The 65KB chunk size is chosen as the maximum, aligning with the User Datagram Protocol (UDP) standard, which defines 65KB as the optimal upper limit for efficient data transmission [15]. Our design leverages "customized information messages" based on TIM, which primarily relies on broadcast communication without application-layer retransmission mechanisms [16]. Consequently, UDP is selected to meet these non-reliable transmission requirements, ensuring lightweight and efficient data delivery. This choice aligns with

[1]https://www.geeksforgeeks.org/ieee-802-11-architecture/

our later use of C-V2X for file transmission, where the priority is on low-latency and efficient broadcast communication.

1) Impacts of Transmission Distance: Figure 4(a) shows the CRR for transmitting ResNet-18 model weights as the distance increases, using different chunk sizes. While all chunk sizes achieve a higher CRR at 5 meters, the CRR drops significantly at 10 and further at 15 meters. For instance, at 10 meters, the 16KB chunk size achieves a CRR of 77%, lower than the 94% with 32KB. At 15 meters, the CRR for 16KB drops to 59%, while 32KB retains better performance at 75%, showing a sharper decline over longer distances. These OTA transmission limitations stem from 5GHz Wi-Fi characteristics: i) susceptibility to signal instability causing disconnections and ii) its design prioritizes higher speeds over an extended range (IEEE 802.11 standard [17]).

Figure. 4(a) depicts a decline in CRR as the transmission distance increases, with the effect being more pronounced for smaller chunk sizes (e.g., 8KB and 16KB) and the largest chunk size (65KB) compared to the medium chunk size (32KB). Notably, using smaller chunks does not ensure higher CRR. This is because i) smaller chunks necessitate more transmissions to complete the file transfer, and ii) the additional protocol overhead from extra headers offsets the reduced data size per packet, leading to a lower CRR.

Recall that we conduct fifteen transmissions to transfer the complete ResNet-18 model weights across various distances using different chunk sizes. The average total transmission time for four chunk sizes over Wi-Fi is shown in Fig. 4(b). To mitigate the impact of outliers, the transmission time is calculated using the Gaussian mean. For a given chunk size, the transmission time nearly doubles as the distance increases from 5 to 10 meters. While larger chunk sizes reduce transmission time at shorter distances, their advantage diminishes over longer distances due to packet loss and increased overhead.

As shown in Fig.4(a) and Fig.4(b), the 32KB chunk size achieves the highest reliability (highest CRR) across various distances while maintaining relatively short transmission times. Medium-sized chunks carry more data per packet, reducing protocol overhead. In contrast, larger chunks are more prone to IP fragmentation, increasing the risk of packet loss and retransmission, as fragmented packets are more likely to encounter transmission issues. Additionally, larger chunk sizes place a greater demand on buffer space and memory, leading to higher system load and reduced success rates.

2) Impacts of Chunk Size: Building on the 100% CRR observed at a 5-meter distance (Fig.4(a)), we analyze the chunk sizes impact on the total transmission time for ResNet-18, ResNet-50, and ResNet-101 at this distance, minimizing external influences. ResNet-50 (94MB) is about twice the size of ResNet-18 (45MB), while ResNet-101 (174MB) is roughly four times larger. Figure5(a) illustrates the total transmission times for the three models using four chunk sizes: 8KB, 16KB, 32KB, and 65KB. The results show that increasing the chunk size reduces the total transmission time across all models, with ResNet-101 requiring the longest time, followed by ResNet-50 and ResNet-18. Doubling the chunk size nearly halves the transmission time, with the most significant improvements seen at smaller chunk sizes, particularly 8KB.

Figure 5(b) shows the transmission time ratios of ResNet-50 and ResNet-101 relative to ResNet-18 across chunk sizes at a

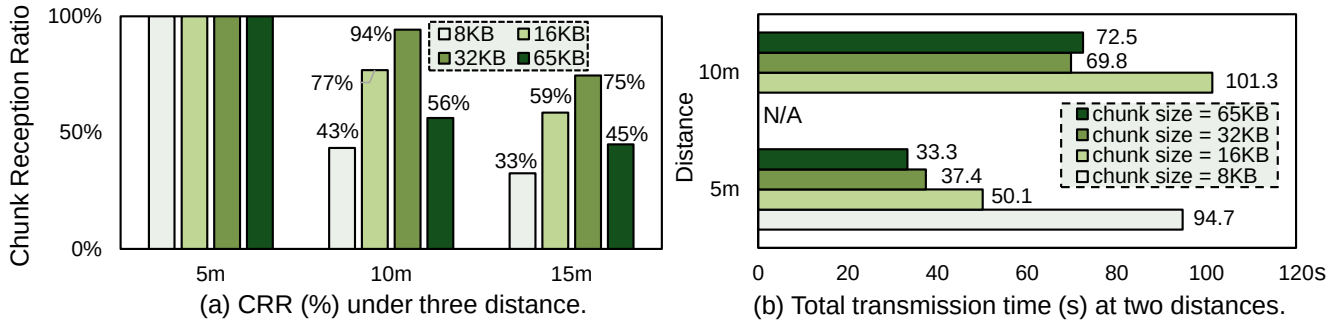

Fig. 4: (a) The chunk reception retio (CRR) (%) for ResNet-18 model weights with varying chunk sizes and transmission distances using Wi-Fi; (b) Average transmission time, calculated using the Gaussian mean, for complete ResNet-18 with different chunk sizes over Wi-Fi.

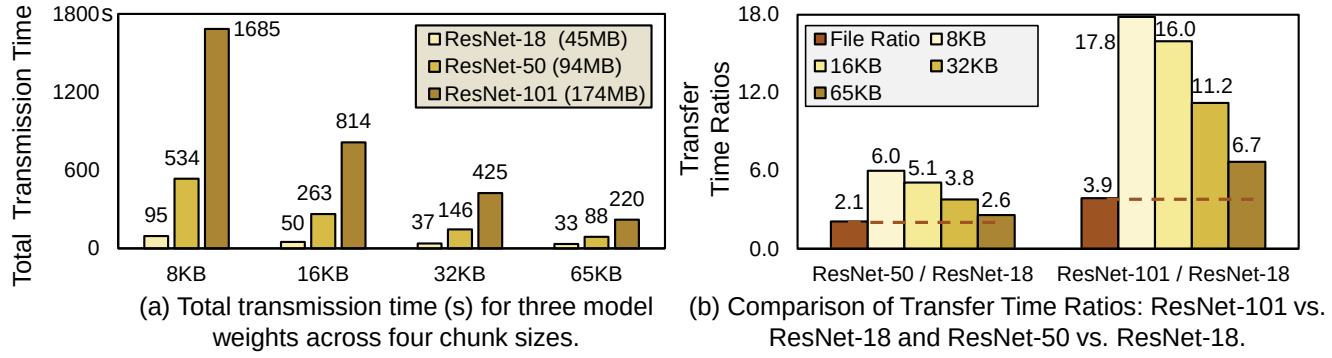

Fig. 5: (a) The impact of chunk sizes on the total transmission time for ResNet-18, ResNet-50, and ResNet-101 at a 5-meter distance; (b) The transmission time ratios of ResNet-50 and ResNet-101 compared to ResNet-18 for various chunk sizes at a 5-meter distance.

5-meter distance over Wi-Fi. ResNet-101 consistently exhibits the highest transmission time ratio, followed by ResNet-50. At the 8KB chunk size, ResNet-50's ratio reaches 6.0, far exceeding its file size ratio of 2.1 relative to ResNet-18, indicating inefficiency with smaller chunks. As chunk sizes increase, the ratios decrease and align more closely with the actual file size ratio, improving transfer efficiency. A similar trend is observed for ResNet-101, demonstrating that *at shorter distances with a high one-time transmission success rate, larger chunk sizes enable more efficient transfers and transmission times proportional to file sizes.*

IV. OTA Update Results in C-V2X

Although 5GHz Wi-Fi faces challenges over longer transmission distances, C-V2X provides wide-area communication capabilities with an approximate range of 600 meters [18], [19], presenting a viable alternative. This motivates us to conduct further testing using the C-V2X protocol.

A. Experiment Result in C-V2X

First, we measure total transmission time and compare C-V2X with Wi-Fi. Then, we analyze the transmission time differences between full and incremental software update.

1) Comparison of C-V2X and Wi-Fi: Table I presents the transmission performance of model weights using C-V2X (V2X and 5G) and Wi-Fi. For Wi-Fi, only the shortest transmission times from Fig.4(b) are included. C-V2X (5G) demonstrates significant advantages over Wi-Fi, achieving up to 9× faster transmission for ResNet-18, 18× for ResNet-50, and 30× for ResNet-101. As C-V2X integrates both V2X and 5G, their performance for transmitting large files is further analyzed. A comparison in TableI reveals that 5G reduces the transmission time for ResNet-50 from 158 minutes to just 5 seconds. For larger models, 5G outperforms V2X with improvements of 1236× for ResNet-18, 1972× for ResNet-50, and 2339× for ResNet-101.

Table I: Total transmission times for three model weights via V2X, 5G, and Wi-Fi communication.

Model Name	Communication	Transmission Time
ResNet-18	V2X	75min
ResNet-18	5G	**3.64s**
ResNet-18	Wi-Fi	33.31s
ResNet-50	V2X	157.83min
ResNet-50	5G	**4.80s**
ResNet-50	Wi-Fi	87.89s
ResNet-101	V2X	290.9min
ResNet-101	5G	**7.46s**
ResNet-101	Wi-Fi	220.26s

2) Comparison of Incremental and Full Update: Next, we evaluate the proposed incremental update strategies against full update in terms of download and execution times, following the two-phase process of download and installation (introduced in Sec. I). Download time refers to the transmission time from the RSU to the vehicle, while execution time denotes the duration required to apply the differential package, which occurs during the software update on the SDV. Table II demonstrates the entire OTA update pipeline delay (download and execution time).

Table II: The package size, transmission time, and execution time for both full and incremental update on YOLO and UFAST models are analyzed. A value of 0 signifies the absence of execution overhead, as full update rely on replacing the entire package.

Model Name	Update Type	Package Size	Download Time	Execution Time
YOLO	Full	41.5MB	32.70s	0
v5 to v7	Incremental	776.5KB	0.63ms	15.08s
UFAST	Full	25.5MB	21.97s	0
v1 to v2	Incremental	70.9KB	0.42ms	11.47s

The full update requires considerably longer download times due to their larger package sizes. For example, downloading the full YOLO package takes 32.70 seconds, while UFAST requires 21.97 seconds. In contrast, incremental update is significantly more efficient, with YOLO's download time reduced to just 0.63 milliseconds and UFAST's to 0.42 milliseconds. In terms of execution time, the SDV system takes 15.08 seconds to apply the YOLO (v5 to v7) update and 11.47 seconds for the UFAST (v1 to v2) update. While full update eliminates execution time overhead by replacing the entire package, they still introduce delays that are twice as long as those observed with incremental update. Moreover, incremental update drastically reduces package size: for UFAST, the full package is 25.5MB with a 21.97-second transmission time, whereas the differential package is only 70.9KB.

V. Observation and Discussion

⋆ **Observation 1**: C-V2X (5G) emerges as an effective solution for facilitating software OTA update while enabling real-time task execution during the update process.

This observation is validated by Table I. Compared to V2X and Wi-Fi, 5G delivers exceptionally high bandwidth and peak data rates, reaching several Gbps through cellular networks, making it highly efficient for transferring large files.

Although V2X cannot achieve the same large file transmission performance as 5G due to protocol constraints, it excels in broadcasting essential road information, such as traffic light states and vehicle speeds, through direct communication among vehicles, infrastructure, and pedestrians. This makes V2X effective for cooperative control in high-speed, open environments, significantly improving driving safety.

⋆ **Observation 2**: The incremental update drastically reduces package size and download time compared to the full update, increasing the feasibility of downloading update packages while the SDV is in motion.

Discussion: Table II confirm this observation. The full update requires significantly longer download times due to their larger package sizes. In contrast, the incremental update is far more efficient, reducing transmission times from seconds to milliseconds. Although incremental update introduces execution overhead on the SDV to apply the differential changes, this overhead occurs during the installation phase rather than the download phase. Since the installation phase can be performed while the SDV is safely parked in a secure location,

this additional time is less critical. Even when considering both download and execution times, the incremental update remains significantly faster than the full update, which still takes more than twice as long due to their larger package sizes.

⋆ **Observation 3**: The medium chunk size (32KB) achieves the highest CRR compared to smaller sizes (8KB and 16KB) and the largest size tested (65KB), across varying transmission distances. It also results in shorter transmission times.

Discussion: This observation is supported by Fig. 4 (a) and Fig. 4 (b). Compared to smaller chunks, medium chunks transmit more data per packet, reducing protocol overhead caused by additional header information. On the other hand, while larger chunks can carry even more data in a single transmission, they are more prone to IP fragmentation. This occurs when larger packets exceed the Maximum Transmission Unit (MTU) of the network path, necessitating fragmentation. Fragmented packets have a higher risk of loss, and losing any fragment requires retransmission of the entire packet, significantly degrading network performance. Additionally, larger chunks consume more system buffers and memory, increasing system load and lowering transmission success rates.

⋆ **Observation 4**: Smaller chunk sizes cannot ensure a higher CRR. Under stable network conditions at shorter distances, larger chunk sizes may be more efficient for OTA update.

Discussion: Figures 4(a)-(b) and 5(a)-(b) support this observation. Smaller chunk sizes increase the total number of packets, each carrying additional protocol metadata (e.g., headers and checksums) to ensure data integrity. This metadata introduces overhead, which grows as chunk sizes shrink, reducing bandwidth efficiency. Additionally, a higher packet count raises the likelihood of packet loss in unstable network conditions, further impacting efficiency. In contrast, larger chunk sizes minimize protocol overhead and streamline data transfer, offering greater efficiency under stable conditions.

VI. Conclusion

This study introduces a software OTA update framework utilizing robotic vehicles and industry-grade RSUs. We propose mechanisms for implementing the incremental update on both the RSU and vehicle sides, facilitating efficient the SDV software update. The framework's performance is evaluated using C-V2X (V2X and 5G) and Wi-Fi for OTA transmissions. By focusing on ResNet model weights and software packages (YOLO and UFAST series), our findings identify a 32KB chunk size as optimal for transmission. Moreover, the results demonstrate that C-V2X (5G) significantly outperforms both Wi-Fi and C-V2X (V2X), emphasizing its suitability for OTA update. The framework further supports the incremental update, cutting download times by over 5000× compared to the full update, significantly reducing OTA pipeline delays. In future work, we plan to extend our study to explore OTA update strategies involving one RSU serving multiple vehicles.

Acknowledgement

This work is supported in part by the National Science Foundation (NSF) under grant CNS-2348151 and the Commonwealth Cyber Initiative (CCI) under grant HC-3Q24-048. We would also like to express our sincere gratitude to Dr. Weisong Shi and William He from the CAR Lab at the University of Delaware for their valuable support.

References

[1] S. Lu and W. Shi, "Vehicle computing: Vision and challenges," *Journal of Information and Intelligence*, vol. 1, no. 1, pp. 23–35, 2023.

[2] K. Korosec, "Tesla recalls 362,758 vehicles over full self-driving software safety concerns," Available: https://techcrunch.com/2023/02/16/tesla-recalls-362758-vehicles-over-fsd-beta-software-safety-concerns/, February 2023.

[3] R. Mocnik, D. S. Fowler, and C. Maple, "Vehicular over-the-air software upgrade threat modelling," 2023.

[4] Tesla, "Software Updates," Available: https://www.tesla.com/en_qa/support/software-updates#tesla-accordion-v2-9640-can-i-drive-my-vehicle-during-a-software-update, 2024.

[5] H. J. Christanto and Y. A. Singgalen, "Analysis and design of student guidance information system through software development life cycle (sdlc) and waterfall model," *Journal of Information Systems and Informatics*, vol. 5, no. 1, pp. 259–270, 2023.

[6] M. Sokcevic, "Real-Time Automotive - Safe Systems for the Future," Available: https://www.tttech-auto.com/knowledge-platform/real-time-automotive-safe-systems-future, January 2021.

[7] M. Evans, T. Talty, C. Dietrich, and X. Gomez, "5G network connectivity automated test and verification for autonomous vehicles using UAVs," SAE Technical Paper, Tech. Rep., 2022.

[8] E. Moradi-Pari, D. Tian, M. Bahramgiri, S. Rajab, and S. Bai, "Dsrc versus lte-v2x: Empirical performance analysis of direct vehicular communication technologies," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 5, pp. 4889–4903, 2023.

[9] R. Molina-Masegosa, S. S. Avedisov, M. Sepulcre, J. Gozalvez, Y. Z. Farid, and O. Altintas, "Towards effective v2x maneuver coordinations: state machine, challenges and countermeasures," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*. IEEE, 2024, pp. 1–5.

[10] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023.

[11] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*. Springer, 2020, pp. 276–291.

[12] G. T. Su and R. R. Rajkumar, "Mpmp: A protocol to transmit long messages for v2x applications," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*. IEEE, 2024, pp. 1–7.

[13] S. International, "Candidate Improvements to Dedicated Short Range Communications (DSRC) Message Set Dictionary [SAE J2735] Using Systems Engineering Methods(STABILIZED Oct 2020) J3067_202010," Available: https://www.sae.org/standards/content/j3067_202010/, October 2020.

[14] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing docker and docker-compose specifications: A developers' survey," *Ieee Access*, vol. 10, pp. 2318–2329, 2021.

[15] P. I. Mackenzie, I. S. Owens, B. Burchell, K. W. Bock, A. Bairoch, A. Belanger, S. F. Gigleux, M. Green, D. W. Hum, T. Iyanagi *et al.*, "The udp glycosyltransferase gene superfamily: recommended nomenclature update based on evolutionary divergence," *Pharmacogenetics and Genomics*, vol. 7, no. 4, pp. 255–269, 1997.

[16] Z. Jin, H. Li, J. Huang, X. Wang, Z. Tan, P. Dong, and Z. Fei, "Raptor-like coded broadcasting for efficient v2x communications," *Electronics*, vol. 12, no. 18, p. 3951, 2023.

[17] S. S. Srivastava, S. Makh, and P. Rodge, "Review on 5g and wi-fi 6 wireless internet connectivity," in *ICT Analysis and Applications: Proceedings of ICT4SD 2022*. Springer, 2022, pp. 135–142.

[18] S. Chen, J. Hu, Y. Shi, L. Zhao, and W. Li, "A vision of c-v2x: Technologies, field testing, and challenges with chinese development," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 3872–3881, 2020.

[19] K. Sehla, T. M. T. Nguyen, G. Pujolle, and P. B. Velloso, "Resource allocation modes in c-v2x: from lte-v2x to 5g-v2x," *IEEE Internet of Things Journal*, vol. 9, no. 11, pp. 8291–8314, 2022.