Enhancing Security Through Task Migration in Software-Defined Vehicles

Mohammad Hamad, Zain A. H. Hammadeh, Davide Alessi, Monowar Hasan, Mert D. Pesé, Daniel Lüdtke, and Sebastian Steinhorst, *Senior Member, IEEE*

***Abstract*—The growing trend of software-controlled operation, control, and development of modern vehicles has led to the emergence of the software-defined vehicle (SDV) design paradigm. SDVs contain increasing software components and, like other cyber-physical systems, are more susceptible to cyber-attacks. However, patching vulnerabilities in these systems may take time, exposing them to cyber threats. To limit the effect of an attack, one solution is to *migrate* critical tasks co-located on the same electronic control unit (ECU) with a compromised component to another ECU. However, existing migration solutions, often designed for fault tolerance, introduce overhead and ignore security parameters. This article introduces SHIFTGUARD, *a security-aware, distributed task migration mechanism* for SDVs. We explore various design decisions that may affect the performance of SHIFTGUARD. We implemented and demonstrated the efficacy of SHIFTGUARD on an automotive platform running the controller area network (CAN) protocol and found that the end-to-end latency of the task migration decision is less than 17 ms for a system with 15 tasks hosted in three ECUs. We also performed extensive design-space exploration using a custom-developed simulator. Our experiments with synthetic workloads show that any task migration request has a 76%–100% success rate. In addition, we demonstrate SHIFTGUARD's scalability for large networks of up to 70 ECUs, making it highly suitable for automotive systems with SDV capabilities.**

***Index Terms*—Autonomous vehicles, resilience, security, software-defined vehicles (SDVs), task migration.**

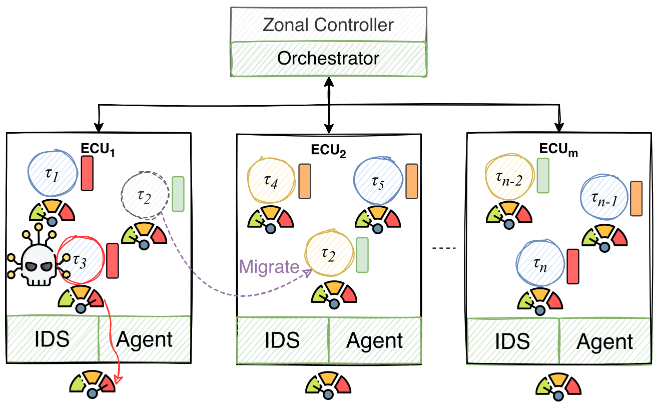


Fig. 1. SHIFTGUARD illustration. The ECUs are connected via a *gateway*. High-critical (yellow) and low-critical (blue) tasks coexist on the same ECU. The tasks have a required security level (e.g., high 🟩, medium 🟧, and low 🟥). An *IDS* monitors task behavior and maintains each task's security level (◔), and the *migration agents* and *orchestrator* handle task migration based on the required level. In this case, τ_2 moves from ECU_1 to ECU_2 as a compromised task τ_3 (red) reduces ECU_1's overall "trustworthiness" (depicted with red meter) needed to run τ_2. The shaded green ▨ components (IDS, agent, and orchestrator) operate in a privileged space.

I. INTRODUCTION

SOFTWARE-DEFINED vehicles (SDVs) are the next big shift in the automotive world [1], where software takes over more and more control of what a car can do. In today's smart vehicles, software handles everything from engine performance to navigation and entertainment, making SDVs a natural evolution of this trend. Current smart vehicles run software systems with over 100 million lines of code [2]. As these vehicles transition to SDVs, this complexity is expected to increase. The software tasks hosted by various embedded systems, called electronic control units (ECUs), vary in *criticality*. In addition, they usually come with distinct security and safety considerations. A common industry practice is that original equipment manufacturers (OEMs) often integrate software components from various third-party vendors or use commercial off-the-shelf (COTS) applications (the majority of which are *low-criticality* tasks) across numerous vehicle subsystems [3]. The complex interactions and dependencies between tasks of different types result in *mixed-criticality* tasks coexisting on the same ECU. In recent years, attackers have exploited vulnerabilities in low-criticality tasks, such as multimedia and connected services, and used them to compromise critical tasks or even the entire system [4]. Achieving a complete separation between critical and noncritical tasks is challenging due to various factors, including hardware limitations (e.g., single core, lack of hardware partitions) and cost implications (e.g., performance overhead, hardware cost, legacy compatibility) [5]. Ensuring the security and reliability

Received 9 February 2025; revised 23 July 2025 and 22 August 2025; accepted 12 September 2025. Date of publication 18 September 2025; date of current version 8 December 2025. This work was supported in part by European Union through Project CyberSecDome under Grant 101120779 and in part by the U.S. National Science Foundation under Grant 2442595. *(Corresponding author: Mohammad Hamad.)*

Mohammad Hamad, Davide Alessi, and Sebastian Steinhorst are with the TUM School of Computation, Information and Technology, Technical University of Munich, 80333 Munich, Germany (e-mail: mohammad.hamad@tum.de).

Zain A. H. Hammadeh and Daniel Lüdtke are with the Institute of Software Technology, German Aerospace Center (DLR), 38108 Brunswick, Germany.

Monowar Hasan is with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99163 USA (e-mail: monowar.hasan@wsu.edu).

Mert D. Pesé is with the School of Computing, Clemson University, Clemson, SC 29634 USA (e-mail: mpese@clemson.edu).

Digital Object Identifier 10.1109/JIOT.2025.3611875

of SDVs will require addressing these challenges through robust design and stringent security measures.

Ensuring the safety and resiliency of SDVs requires safeguarding tasks with high-criticality from compromised ones in the event of a security breach. Automotive systems often use *software rejuvenation*, that is, restarting the task or the entire ECU to handle faults. While this can usually resolve faults caused by transient errors, *it is ineffective against cyber-attacks*, as the vulnerability persists and leaves an opportunity for the attacker to exploit as soon as the task resumes. Moreover, the vulnerabilities responsible for a cyber-attack may *take days or weeks to be fixed* [6]. This waiting period presents a dilemma: either the vehicle continues operating with a compromised component while waiting for a patch from the OEM, or suspending the compromised component reduces functionality (or, worse, safe drivability). To mitigate this risk, one solution is to *migrate* critical tasks to another ECU while waiting for a fix [7], [8]. The existing migration techniques [9], [10], [11] do not consider vehicle security and, therefore, cannot be directly applied to defend against cyber-attacks. Other solutions, such as maintaining redundant copies of critical tasks for use during an anticipated attack [12] or replicating the ECU using redundant hardware [13], come with high costs (due to increased weights and wiring complexity) and are cumbersome to manage/debug.

A. Basic Idea

This article introduces ShiftGuard, a distributed *security-aware task migration mechanism* for SDVs that does *not* require redundancy in hardware or software. When a low-criticality task is compromised, ShiftGuard identifies a suitable host ECU to migrate an *uncompromised* high-criticality task. The migration process ensures that tasks coexist securely on the destination ECU while meeting its scheduling requirements. The migration can occur when the vehicle is in either fail-operational or fail-safe modes (see Section VI). Fig. 1 illustrates the core concept of ShiftGuard, focusing on task migration within a single *zone*.[1] In ShiftGuard, each task is assigned a security level (a runtime measure of how likely it is to be compromised), and each ECU is assigned a trust score (an aggregate of the security levels of tasks it hosts, weighted by their criticality). Each task also has a required security level that must be met by the hosting ECU's trust score. These metrics determine whether tasks must be migrated after an attack is detected. We provide formal definitions and computation methods for these metrics in Section II-C. Trusted agents within each ECU and an orchestrator on the zonal controller facilitate task migration. For instance, when a task τ_3 is compromised (see Fig. 1), the intrusion detection system (IDS), operated in a privileged space, detects the attack and decreases its security level. This, in turn, lowers the overall trustworthiness of hosted ECU_1 (◒). As a result, the uncompromised high-criticality task τ_2, which is hosted on the same ECU_1 and has a specific security level ▯, no longer meets its required security level. Thus, the *migration agent* needs to move it to another ECU. The agent initiates the task migration process and informs the orchestrator, which coordinates with agents on the other connected ECUs. In the example shown in Fig. 1, ECU_2 is selected as the suitable target that meets the required security level and can accommodate τ_2 without violating its requirements. Thus, τ_2 is moved to ECU_2. Note that τ_1 does not need to be relocated since its required security level 🟥 is still met. If there is no *optimal* host location within the zone, then the orchestrator in the zonal controller will extend the request to other zones.

[1]In a *zonal architecture*, ECUs are grouped based on geographic zones. A *zonal controller* manages the ECUs within the zone and handles communication across zones. See Section II-A for additional details.

B. Our Contributions

ShiftGuard is engineered to tackle the *design* and *operational* phases of SDVs. In the design phase, it helps system architects optimize system architecture by simulating various configurations, such as the number of zones and ECUs per zone. The design challenge ShiftGuard addresses is the following: *how do we balance minimizing end-to-end latencies with maximizing successful task migrations?* This process is crucial for identifying the best architecture that enhances resilience and ensures high reliability in SDVs. In the operational phase, ShiftGuard functions as a real-time task migration framework, securely migrating high-criticality tasks when low-criticality tasks are compromised. It ensures that migrations comply with security and scheduling constraints, allowing for secure task coexistence and maintaining system functionality even during cyber threats. The core focus of ShiftGuard is providing a *recovery mechanism* when a task is compromised while awaiting security updates and patches for the vulnerabilities that caused the attack, which may take a considerable time (days or even weeks!). This ensures continued safe operation despite vulnerabilities.

The main contributions of this article are as follows.

1) Introduction of a *security-aware distributed task migration* architecture for SDVs (ShiftGuard) and *formalized analytical models* to support that framework (Section III).
2) *Parameterization of various system attributes* to optimize system behavior, allowing system designers to choose attributes (such as number of zones and ECUs/zone) that ensure security requirements while guaranteeing timing requirements. ShiftGuard does so with minimal performance overhead (Section V).

We demonstrate the *scalability* of ShiftGuard for large networks (up to 70 ECUs in 10 zones). Our results show that ShiftGuard can successfully migrate an affected task running on a relatively vulnerable (less trusted) ECU to a trustworthy one (see Sections IV-B and V). We further validate ShiftGuard's feasibility using a physical hardware setup consisting of four Raspberry Pi boards (Section IV-C). Our experiments show that ShiftGuard's end-to-end latency is less than 17 ms on our hardware testbed with three ECUs and 15 tasks.

II. Background and Models

We now start with a background on SDV (Section II-A). We then introduce the in-vehicle network architecture (Section II-B), security constraints (Section II-C), and the threat model (Section II-D) considered in this article. Key mathematical notations are listed in Table III (see Appendix A).

A. Software-Defined Vehicles

SDVs mark a significant evolution in the automotive industry, where software rather than hardware increasingly controls vehicle functionalities, behaviors, and features. This shift allows for greater flexibility, faster updates, and seamless integration of advanced technologies, fundamentally transforming traditional automotive design [1]. A key aspect of SDVs is the *decoupling of hardware and software*, where software becomes the central driver of vehicle operations, and hardware serves as a shared, flexible resource [14], [15]. This decoupling is supported by hardware abstraction, where standardized interfaces enable software to interact with various hardware components independent of vendor or platform. By being hardware-agnostic, SDVs facilitate seamless interoperability between different systems, enabling broader collaboration across manufacturers and enhancing system adaptability for future advancements. This decoupling of hardware and software in SDVs is crucial in allowing flexible software migration across different ECUs. A critical component driving SDVs is the *zonal architecture*, which shifts away from the traditional domain-based electronic/electrical (E/E) framework [16]. In this architecture, ECUs are organized based on geographic "zones" within the vehicle instead of dedicating each ECU to a specific function or domain. *Zonal controllers* manage all the ECUs and sensors within their respective zones to create a more efficient system. These controllers are linked to a *central computing unit*, which aggregates data and controls from all the zones, enabling centralized decision-making and greater flexibility in software management. It is important to note that the *number of zones can vary* depending on the requirements and complexity of the vehicle [17].

B. System Components and Terminologies

We consider a heterogeneous distributed system with M ECUs grouped into Z *zones* and connected by a *zonal controller* (ζ). These controllers communicate through the *central computing unit*. Each ECU ϵ_i schedules tasks using a fixed-priority real-time scheduling policy. It is important to note that we are targeting SDV systems where ECUs (they are not microcontrollers) are not fully used, saving resources (e.g., CPU and memory) to accommodate new functionalities and support over-the-air updates. Each ECU has a trusted *IDS* (such as RedZone [18]) and a trusted *agent* (α). The agent is responsible for monitoring the health and security of the tasks. When a task needs to be migrated, this agent communicates with the *orchestrator* in the zonal controller. Both the IDS and agent are securely protected from other tasks on the same ECU through isolation mechanisms that ensure their integrity. This is a realistic assumption, as modern ECUs incorporate secure components like hardware-enforced isolation, secure storage, and trusted computing modules for executing critical tasks (for instance, the IDS and agent in our context) [19], [20], [21], [22]. There are N real-time tasks that can be mapped to any ECU. Each task τ_ℓ is defined by a tuple: $\{T_\ell, D_\ell, \text{pri}_\ell, (C_\ell^0, \ldots, C_\ell^{M-1})\}$. In the task parameter tuple, T_ℓ is the period, D_ℓ the relative deadline ($D_\ell \le T_\ell$), pri_ℓ the priority, and C_ℓ^1 the worst case execution time (WCET) when the task is mapped to the ECU ϵ_1. The task is "schedulable" if its response time R_ℓ (i.e., time between arrival and completion) is less than the deadline, i.e., $R_\ell \le D_\ell$. The tasks may have data dependencies—if that is the case, they form a directed acyclic graph (DAG). A DAG is defined as $G_\rho = (V_\rho, E_\rho)$, where $V_\rho = \{v_{\rho,1}, v_{\rho,2}, \ldots, v_{\rho,n}\}$ is the set of vertices and $E_\rho \subseteq V_\rho \times V_\rho$ is the set of directed edges of the DAG. Each vertex $v_{\rho,\ell} \in V_\rho$ represents a task τ_ℓ. An edge $E_{\ell,p} = (v_{\rho,\ell}, v_{\rho,p})$ represents a precedence relationship between the task τ_ℓ represented by $v_{\rho,\ell}$ and the task τ_p represented by $v_{\rho,p}$. The DAG G_ρ has an end-to-end deadline D_ρ^{e2e}. The end-to-end latency R_ρ^{e2e} can be computed using existing techniques such as compositional performance analysis (CPA) [23]. The DAG G_ρ is schedulable if $R_\rho^{\text{e2e}} \le D_\rho^{\text{e2e}}$. Each task τ_ℓ is assigned a *criticality level* κ_ℓ that reflects its criticality and can be determined during the design phase.

C. Security Parameters

We now introduce three security parameters used in ShiftGuard's decision-making process. The *security level* σ_ℓ^i represents the task's current security status, which indicates whether the task is compromised or benign. At runtime, the IDS monitors and updates this level. Let C_{IDS}^i ($0 \le C_{\text{IDS}}^i \le 1$) be the confidence level of the IDS on the ECU ϵ_i, FPR^i ($0 \le \text{FPR}^i \le 1$) is the false-positive rate of the IDS on ϵ_i, and I_ℓ is the impact of the attack on the task τ_ℓ.[2] Hence, we can define the security level σ_ℓ^i as follows:

$$\sigma_\ell^i = C_{\text{IDS}}^i \cdot I_\ell \cdot \left(1 - \text{FPR}^i\right). \tag{1}$$

The *trust score* S_i represents the trustworthiness (from a task's perspective) of the ECU ϵ_i, which is a function of the criticality levels κ_ℓ and security levels σ_ℓ^i of all the tasks running on the ECU. The trust score is *dynamic* and will change in response to any changes in the security levels of the running tasks. We define S_i as follows:

$$S_i = \sum_{\forall \ell \mapsto i} \omega_\ell \cdot \sigma_\ell^i \tag{2}$$

where $\omega_\ell = \hat{g}(\kappa_\ell)$ is an "weighting factor" which is a function of the criticality of the tasks denoted by $\hat{g}()$. The notation $\ell \mapsto i$ denotes that τ_ℓ is mapped to (running on) ϵ_i.

The *required security level* $R\sigma_\ell$ of task τ_ℓ is the minimum security level required to continue operating on the host ECU. This level can be defined during the system design phase in a similar way to how the automotive safety integrity level (ASIL) [26] of each task is defined, albeit we adopt it for a security perspective. The agent will monitor the security levels

[2]Earlier research shows (a) how to determine confidence level and false-positive rates, i.e., C_{IDS}^i and FPR^i, respectively, [24] and (b) the calculation of attack impact factor, I_ℓ [25].

for all the tasks. If $R\sigma_\ell$ is not satisfied, the agent will attempt to migrate τ_ℓ and notify the gateway. An *orchestrator* running on each zonal controller handles migration requests from the agents.

To map a task τ_ℓ to an ECU ϵ_i, we must ensure $S_i \geq R\sigma_\ell$. If a task $\tau_k : k \mapsto i$ gets compromised, its security level decreases, as does the trust score of ϵ_i. To keep the system operational with all the high-criticality tasks running, SHIFTGUARD aims to define a *lightweight migration mechanism* that ensures the security constraints of other benign tasks are satisfied. In doing so, the migration mechanism in SHIFTGUARD may stop a few tasks with low criticality levels to guarantee that the timing and security requirements of other high-criticality, noncompromised tasks are met. Hence, the target ECU can host migrated tasks from the potentially vulnerable originating ECU without losing performance.

D. Threat Model

We assume that an attacker can compromise tasks using known vulnerabilities of the automotive systems [27], [28]. This includes (but is not limited to) vulnerabilities such as buffer overflow [29], remote code execution [30], or injection attacks aimed at modifying the behavior of a compromised task [31]. The adversary can launch an attack on a different ECU starting from the compromised task (a stepping-stone attack [4]). We assume the presence of an intrusion detection module (viz., IDS) that can detect any misbehavior of tasks caused by various attacks. We stress that we *do not make any assumptions on the design of IDSs*. A vast literature exists on automotive IDS (see the surveys [32], [33]) that can be integrated with SHIFTGUARD. As IDSs are not perfect (i.e., often result in false positives or negatives, even if it was shown that such rates are very small [24], [34], [35]), SHIFTGUARD accounts for this inaccuracy in the design; for instance, see (1). We assume that the IDS and the agent are protected from other tasks running on the same ECU through isolation techniques. Furthermore, best practices such as secure boot [36] should be followed to mitigate ECU-level attacks. Protection against privilege escalation attacks that may jeopardize the operation of trusted components (i.e., IDS, agent, and orchestrator) are not within the scope of this work. Although the adversary can launch active spoofing and passive eavesdropping attacks against the communication on the in-vehicle network as a result of the compromised ECU, using existing techniques [37] and protocols such as IPsec, MACsec, and SecOC [38] can mitigate this attack vector. Hence, the migration requests issued by the migration agent are authenticated. Besides, we presume the existence of mechanisms such as those proposed in early work [39] that validate the migrated task before its execution.

III. SHIFTGUARD: DESIGN AND ANALYSIS

In this section, we outline the workflow of SHIFTGUARD (Section III-A) and discuss its latency bounds (Section III-B).

A. SHIFTGUARD Workflow

Fig. 2 presents the steps of SHIFTGUARD. The migration process initiates when any attack is suspected (**Step 1**). If the IDS detects an attack for a compromised task τ_k, it updates the security level σ_k^i and notifies the agent. The agent calculates the trust score S_i of ϵ_i and reevaluates the security requirements of all the tasks $\mapsto i$. If the agent detects that an uncompromised task τ_m no longer meets its required security level $R\sigma_m$ (i.e., $S_i < R\sigma_m$), it initiates a request to the orchestrator on the zonal controller (**Step 2**) indicating the need to migrate τ_m to an ECU with a higher trust score. Upon receiving the request, the orchestrator sends enquiries to all the agents on other ECUs within the same zone to assess the feasibility of accommodating task τ_m (**Step 3**). Next (**Step 4**), each agent, on ϵ_j, receives the request from the orchestrator and conducts a schedulability test to assess whether τ_m can be scheduled. In addition, it evaluates the potential impact on the current local tasks, (i.e., whether they will miss their deadlines). The agent then computes the *hospitality* factor H_j^m depending on the schedulability test to determine whether the ECU can accommodate the request as follows:

$$H_j^m = K - \sum_{\forall l \mapsto j \bigwedge R_l > D_l \bigwedge l \text{ is low-critical}} \kappa_l - \sum_{\forall h \mapsto j \bigwedge R_h > D_h \bigwedge h \text{ is high-critical}} K \cdot \kappa_h \quad (3)$$

where $K = \sum_{\forall \ell < N} \kappa_\ell$ is a constant to scale the hospitality values. By design, the agent considers sacrificing low-criticality tasks to accommodate critical task τ_m. If all the tasks on ϵ_j meet their deadline, then $H_j^m = K$. If at least one low-critical task misses its deadline with the presence of τ_m, then $0 < H_j^m < K$. If at least one critical task misses its deadline, then the hospitality is negative $H_j^m < 0$. In addition, the agent evaluates the trust score of the ECU S_j^m considering τ_m as follows:

$$S_j^m = \sum_{\forall \ell \mapsto j \bigwedge R_\ell \leq D_\ell} \omega_\ell \cdot \sigma_\ell^j. \quad (4)$$

Each agent then sends the computed parameters (H_j^m and S_j^m) to the orchestrator in their zonal controller (**Step 5**). The orchestrator solves an optimization problem to determine the optimal ECU (denoted as ϵ_t) for τ_m (**Step 6**). The solver on the orchestrator works as follows.

Let P^ϵ be the set of all the ECUs in one zone that can potentially be a candidate to migrate the task τ_m. Let us also define a binary variable χ_j for every potential target $\epsilon_j \in P^\epsilon$ and τ_m is a member of the DAG G_ρ. Besides, let $R_{\rho,m\mapsto j}^{\text{e2e}}$ denote the end-to-end latency of G_ρ when τ_m is mapped to ϵ_j. The solver computes $R_{\rho,m\mapsto j}^{\text{e2e}}$ for all $\epsilon_j \in P^\epsilon$. We want to find the best ECU for τ_m. We formulate a mixed integer linear programming (MILP) [40] problem as presented below.

Objective:

$$\max_{\chi} \sum_{\forall j \in P^\epsilon} \chi_j \cdot H_j^m. \quad (5)$$

Constraints:

$$\forall j \in P^\epsilon : \chi_j \cdot R\sigma_m \leq S_j^m \quad (6)$$

$$\sum_{\forall j \in P^\epsilon} \chi_j \leq 1 \quad (7)$$

$$\forall j \in P^\epsilon : \chi_j \cdot H_j^m \geq 0 \quad (8)$$

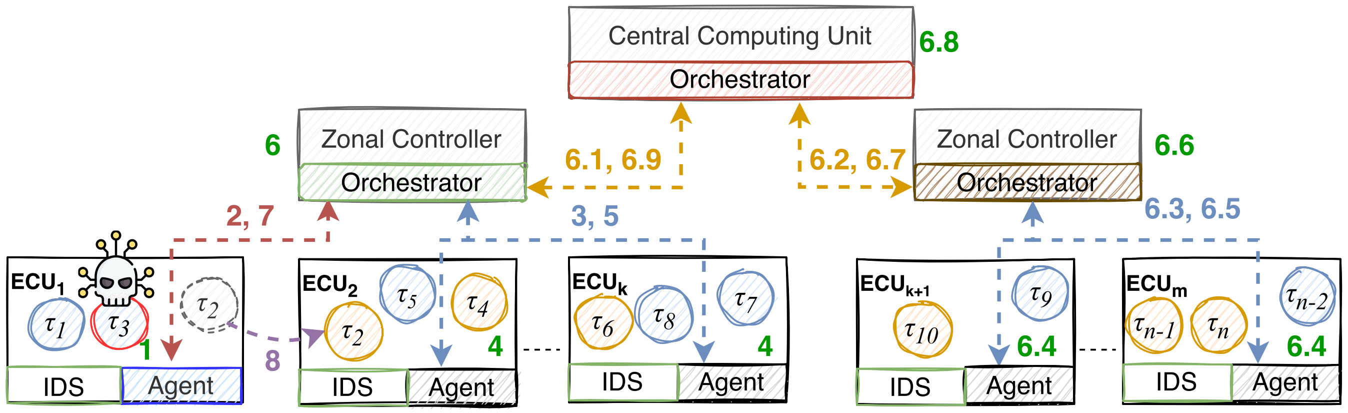


Fig. 2. Workflow of SHIFTGUARD that starts with an attack detection of τ_3 (Step 1) and ends with migration of τ_2 in ECU_2 (Step 8). SHIFTGUARD operates in two phases. (a) First, it checks whether migration is feasible in the local zone (Phase I) and (b) if not, finds an ECU in another zone, which involves orchestrators in other zonal controllers and central computing unit (Phase II).

$$\forall j \in P^{\epsilon} : \chi_j \cdot R^{\text{e2e}}_{\rho,m\mapsto j} \leq D^{\text{e2e}}_{\rho}. \tag{9}$$

The objective function in (5) finds the maximal hospitality factor where the task can be migrated. The constraint in (8) guarantees that no critical task will be stopped to accommodate τ_m. We note that the number of variables and constraints grows linearly with the number of ECUs (agents). If the orchestrator succeeds in finding an ECU ϵ_t to host τ_m (i.e., there exists a feasible solution), it sends the address of ϵ_t to the agent on ϵ_i to start the migration (Step 7). We refer to the steps discussed so far as Phase I.

If the orchestrator fails to find a suitable destination in the local zone, SHIFTGUARD enters into Phase II. In this phase, the orchestrator on the zonal controller initiates a migration request to the orchestrator on the central computing unit (Step 6.1). The orchestrator on the central computing unit sends the hosting request to all other zonal controllers (Step 6.2). The zonal controllers then relay this request to their respective agents (Step 6.3). The agents on other zones compute their trust score and hospitality (i.e., S^m_k and H^m_k) values (Step 6.4) and share them with their zonal controllers (Step 6.5). Each orchestrator uses this information and solves the optimization problem [see (5)–(9)] to determine the most suitable host for τ_m (Step 6.6) and informs its findings to the central computing unit (Step 6.7). Based on the inputs from different orchestrators, the central computing unit's orchestrator invokes the solver to determine the "global" (viz., system-wide) optimal ECU, ϵ_t, who can host τ_m (Step 6.8). The response is forwarded to the orchestrator on the zonal controller (Step 6.9), which is then shared with the agent on ϵ_i to start the migration (Step 7). Finally, the migration process—including moving the task and maintaining the interconnections of the migrated task with existing tasks on the current ECU and tasks running on the other ECUs—can follow the existing AUTOSAR methods [41], [42] (Step 8). To facilitate this, each agent uses a key to sign the migrated task. The same key will be used by the agent on the host ECU to validate the task before running it on the new host. Alternatively, agents can use dedicated pairs of keys (public and private keys) to ensure the integrity of the migrated task. Note that the current SHIFTGUARD implementation does not incorporate cryptographic signing mechanisms, and we leave it for future work. It is worth noting that it is still possible that no feasible host is available for migration, either locally (within the same zone) or globally (across different zones). In such a case, no migration will be executed (more discussion about this case in Section VI). Conversely, in some cases, multiple ECUs could host the task, and they have the same hospitality value. In this situation, we select the ECU with the smallest index that can host the task. Algorithms 1 and 2, presented in Appendix A, demonstrate how SHIFTGUARD operates from the agents' and orchestrators' perspectives, detailing the distinct roles they play in the protocol.

B. Latency Calculation and Bounds

One of the design attributes in SHIFTGUARD is to determine the *end-to-end latency* between Step 2 and Step 7. Note that Step 8 is not part of the latency calculation analysis as this is a platform-specific part and depends on several factors such as the network throughput, network load, and the executable size. Recall that the migration mechanism in SHIFTGUARD has two phases. In Phase I, the migration request is handled and accepted by the zonal controller (which does not include Step 6.1–Step 6.9). Let us denote the end-to-end latency for Phase I as L (i.e., *local* end-to-end latency). In Phase II, the migration request is forwarded to the central computing unit (i.e., includes all the steps presented earlier). We denote the end-to-end latency when entering Phase II as E (i.e., *global* end-to-end latency). Let us introduce a few notations before computing an upper bound on L and E. Let $\text{Com}_{\text{agent}\rightarrow\text{orch}}$ denote an upper bound on the communication time from the agent to the orchestrator in the zonal controller. Likewise, let $\text{Com}_{\text{orch}\rightarrow\text{agent}}$ denote an upper bound on the communication time in the other direction. In addition, let $\text{Com}_{\text{gate}\rightarrow\text{mgate}}$ and $\text{Com}_{\text{mgate}\rightarrow\text{gate}}$ denote the upper bounds on the communication times between a zonal controller and the ceentral computing unit. In addition, let 1) t^s_{agent} denote the maximum time any agent needs to compute its H^m_j and S^m_j and 2) t^s_{orch} denote the maximum time needed by any orchestrator to solve its constraints.

Based on the above notations, we bound the local latency L as follows:

$$L \leq A \left(\text{Com}_{\text{agent}\to\text{orch}} + \text{Com}_{\text{orch}\to\text{agent}} + t^{s}_{\text{agent}}\right) + t^{s}_{\text{orch}} \quad (10)$$

where A is the maximum number of ECUs (agents) connected to one zonal controller. The global latency E is bounded as follows:

$$E \leq Z \cdot L + Z \cdot \left(\text{Com}_{\text{gate}\to\text{mgate}} + \text{Com}_{\text{mgate}\to\text{gate}}\right) + t^{s}_{\text{orch}} \quad (11)$$

where Z is the number of zonal controllers (not including the central computing unit). The parameters t^{s}_{agent} and t^{s}_{orch} can be computed using the existing WCET analysis tools [43]. By empirical measurements, we can evaluate $\text{Com}_{\text{agent}\to\text{orch}}$ and $\text{Com}_{\text{orch}\to\text{agent}}$. Note that the parameters A and Z are the unknowns in our design question (see Section I). A key factor affecting the efficiency of SHIFTGUARD is how often Phase II is involved, i.e., to determine the *failure ratio* to accept the migration requests in the same (local) zone. Let $0 < F \leq 1$ represent this ratio. The challenge is then bound to F, which is a function of various dynamically varying attributes such as the security of the tasks at the moment of migration request and the load on the ECU. The experimental results provide more insights into the design parameters A, Z, E, L, F, as we discuss next.

IV. EVALUATION

We evaluate the efficacy of SHIFTGUARD on two fronts: 1) broader design-space exploration with synthetic workload and large automotive network (Section IV-B) and 2) demonstration on a Raspberry Pi-based hardware testbed running CAN protocol (Section IV-C).

A. Evaluation Criteria

Recall from our design problem (introduced in Section I) that we want to parameterize the attributes (i.e., number of zones and ECUs per zone) that reduce the end-to-end communication latency and increase the chances of successful migration. To solve our design problem, we introduce three questions which we will answer and validate with our experiments.

1) *Q1:* How efficient is it to keep the migrations within the zone?
2) *Q2:* Is it efficient to bypass the local zonal controller and send each migration request to the central computing unit (i.e., maintain no hierarchy)?
3) *Q3:* What is the tradeoff between the *system parameters* (i.e., the number of zones Z and ECUs per zone A), and the *performance parameters* (i.e., the local end-to-end latency L, the global end-to-end latency E, and the failure ratio to accept the migration requests in the local zone F)?

B. Experiments With Synthetic Workload

To get more insights about design parameters that affect feasibility and efficiency and to answer our design questions, we developed a simulator and conducted extensive design-space exploration.

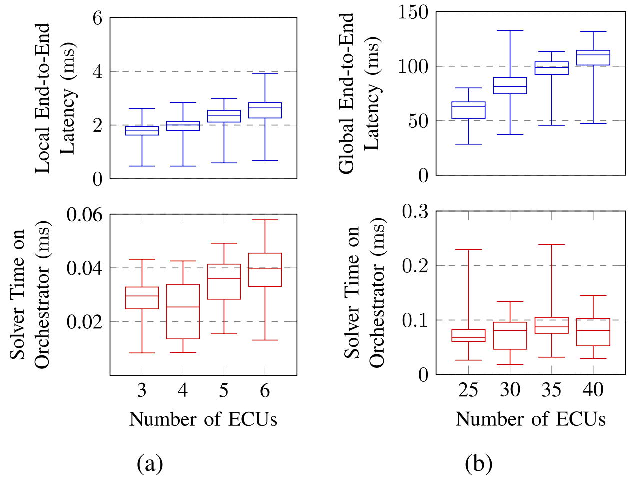


Fig. 3. End-to-end latency and the solver time of two design decisions. (a) Experiment I: the migration is made only within the same zone. (b) Experiment II: a single-layer migration mechanism.

1) *Simulator Design and Parameters:* Our simulator can accommodate multiple ECUs (represented by the agents), zonal controllers (orchestrators), and a central computing unit. Each entity in the simulator is designed as a Linux process.

a) Simulation setup: We generated N periodic tasks with unique priorities (0 to $N-1$). We randomly grouped three to five tasks in one DAG and allocated three to five tasks to each ECU. The utilization of each ECU was randomly assigned by following a normal distribution with $(0.6, 0.1)$ as mean and standard deviation. The task utilizations were generated using the UUniFast algorithm [44]. We assigned task periods randomly as follows: $T_\ell = f \cdot p$ where $f \in \{1,2,3,4,5,6,7,8\}$, and $p \in \{280, 340, 450, 500\}$ and considered implicit deadlines (i.e., $D_\ell = T_\ell$). The criticality levels were $1 \leq \kappa_\ell \leq 10$, where κ_ℓ scales with the priority. Tasks with $\kappa_\ell < 6$ are considered noncritical tasks. The security level $1 \leq \sigma^{i}_{\ell} \leq 10$ was generated randomly such that it was smaller than 5 (i.e., middle of ten levels) for noncritical tasks. The required security level $R\sigma_\ell$ for each task τ_ℓ was computed as follows: 1) $R\sigma_\ell = 0.4 \cdot \sigma^{i}_{\ell}$ (for noncritical task) or 2) $R\sigma_\ell = 3 \cdot \sigma^{i}_{\ell}$ (critical task). We scheduled each ECU using a preemptive fixed-priority policy, and the zonal controllers used time-division multiple access (TDMA) arbitration.

b) Performance metrics: We performed event-driven simulation to mimic the workflow of an automotive system. The simulator initializes the entities and randomly selects an ECU (agent) and a noncritical task running on that ECU (victim task) to launch the attack. If the security requirement is violated (e.g., security level drops), the agent sends a migration request. We measure the following parameters to understand the performance of SHIFTGUARD: 1) solver time; 2) end-to-end latency; and 3) the number of rejected requests.

2) *Experiment I:* Our first set of experiments focuses on the relationship between L and A and between F and A (i.e., Q1). We ran the simulation 100× for every $A = 3, 4, 5, 6$ considering $Z = 1$. We compute the solver run time t^{s}_{orch}, the local end-to-end latency L, and the number of rejected requests. Fig. 3(a) presents t^{s}_{orch} and L. Table I shows the

TABLE I
REJECTED MIGRATION REQUESTS UNDER DIFFERENT DESIGN DECISIONS. FEWER REJECTIONS SIGNIFY BETTER EFFICIENCY. WE COLOR-CODED THE CELLS FROM GREEN (BEST) TO RED (WORST) FOR VISUAL EASE

	Experiment I				Experiment II*				Experiment III											
									Scenario 1				Scenario 2				Scenario 3			
Agents (M)	3	4	5	6	25	30	35	40	25	30	35	40	15	20	25	30	30	30	28	24
Zone (Z)	1	1	1	1	N/A	N/A	N/A	N/A	5	6	7	8	5	5	5	5	5	6	7	8
M **per** Z (A)	3	4	5	6	N/A	N/A	N/A	N/A	5	5	5	5	3	4	5	6	6	5	4	3
Rejected req. (%)	24	16	18	10	3	0	0	0	2	0	0	0	0	2	2	1	1	0	0	2

*Experiment II does not involve any zonal controllers (independent of A and Z).

number of rejected requests. We found that L is less than 4 ms. However, the number of rejected requests is less than 24. Therefore, a migration request can be fulfilled with a probability ≥76%. Increasing the number of ECUs will further improve the chances of accepting migration requests.

Observation 1. Despite low latency, the relatively high failure ratio suggests that keeping the migration within the zone is *not* very efficient (answers Q1 — *is it efficient to keep the migrations within the zone?).*

3) Experiment II: This set of experiments addresses Q2. In this setup, the agent sends the migration request to the orchestrator in the central computing unit directly (i.e., no zonal controller is included). The orchestrator in the central computing unit asks all other agents (located in ECUs) in the system directly to check whether they can host the task that requested migration. We computed t^s_{orch}, E, and the number of rejected requests. We repeated the experiments 100× for each value of M. Fig. 3(b) and Table I report our findings. As the result shows, the solver time t^s_{orch} is about five times larger than the previous case (intrazone) due to the increasing number of agent communications. The experiment shows the scalability of the optimization problem presented in (5)–(9). The solver takes less than 0.3 ms, where the number of variables (agents) reaches 40. The latency E increases with the number of agents. Fig. 3(b) illustrates that the majority of the end-to-end latency is spent on the communication where the solver time represents a very small portion. The advantage of this approach is the very low number of rejected migration requests. The probability of accepting a migration request increases to ≥97%. The main disadvantage is the high latency (E) observed for each migration request, without the guarantee that the destination ECU will belong to the same zone as the task's original zone. This may or may not be acceptable, depending on the target application's requirements.

We now present a tradeoff between techniques explored in Experiments I and II.

Observation 2. A flat hierarchy results in *near-zero failure rates*. However, this comes at a cost, i.e., *latency increases* for migration requests (answers Q2 — *maintain a hierarchical architecture or no-hierarchy?).*

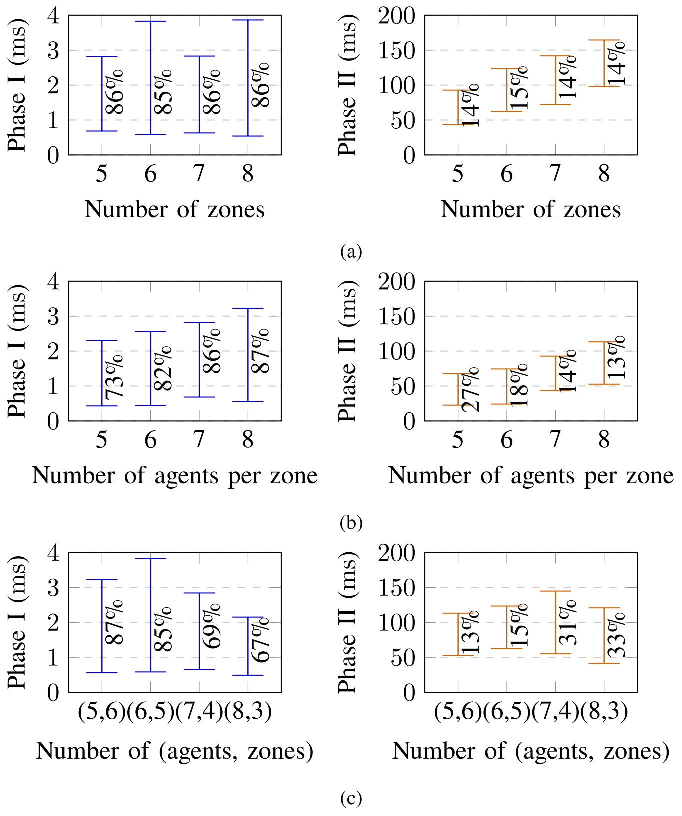


Fig. 4. Local end-to-end latency L for Phase I, the global end-to-end latency E for Phase II, and failure ratio F as percentage for different values of ECUs per zone (agents) A and number of zones Z. (a) Scenario 1: $A = 5$ and Z varies. (b) Scenario 2: $Z = 5$ and A varies. (c) Scenario 3: A and Z varies inversely. The failure ratio F decreases when the number of agents per zone A increases.

4) Experiment III: The final set of synthetic experiments aims to examine the tradeoff between the number of zones Z and ECUs per zone A, i.e., explore Q3. We consider three scenarios as presented below.

Scenario 1: We fixed the value of $A = 5$, and then we ran the simulation with $Z = \{5, 6, 7, 8\}$ and 100 trials for each. This experiment examines the impact of Z on local end-to-end latency L, global end-to-end latency E, and failure ratio F. Fig. 4(a) and Table I present the results. As seen from the plots, E grows linearly with Z while L is unaffected. The parameter F, which appears as % on Fig. 4, shows no dependency on Z.We further investigated the relationship between Z and F in our last setup (Scenario 3).

Scenario 2: We fixed the value of $Z = 5$, and then we tested with $A = \{3, 4, 5, 6\}$ and ran $100\times$ for each. The focus of this experiment is understanding the impact of A on E and again on F and L. We find that E grows linearly with A. This is aligned with the relationship between A and F we observed in Experiment I, viz., the larger the A, the smaller the F. Moreover, L grows linearly with A. Fig. 4(b) and Table I present the results.

Observation 3. Investigating the trade-off between the number of zones Z and ECUs per zone A reveals:
(i) There exists a direct relation between (1) A and local end-to-end latency L, and (2) between Z and global end-to-end latency E,
(ii) There also exists an inverse relationship between A and the failure ratio F, and
(iii) It is possible to *tune* optimal values of A and Z based on design requirements, but there are no theoretical optimal values (answers Q3 — *what are the trade-offs between system and performance parameters?).*

Scenario 3: In this setup, $A = \{6, 5, 4, 3\}$ and $Z = \{5, 6, 7, 8\}$ vary such that when Z increases, A decreases (i.e., $M = \{30, 30, 28, 24\}$). Table I and Fig. 4(c) present the results. L does not increase with Z, and E does not decrease with A, which is also reflected in (11). This experiment confirms that no relationship exists between F and Z.

C. Experiments With Hardware Testbed

1) Testbed Setup: To demonstrate the feasibility and assess the overhead of using SHIFTGUARD at runtime, we conducted tests on a real hardware testbed comprising four Raspberry Pi 4 Model B embedded boards. This choice was made due to their comparable processing power and memory capacity to modern automotive ECUs [45]. Our test platforms (see Fig. 5) are equipped with 1.5-GHz ARM-Cortex A72 64-bit SoC and are mounted with either an RS485 controller area network (CAN) HAT or a 2-CH CAN HAT. Both the CAN HATs' modules adopt an MCP2515 CAN controller and an SN65HVD230 CAN transceiver. All the four boards are interconnected using CAN and communicate at 500-kbits/s bitrate. Three of these devices represent ECUs, each hosting various tasks, an agent, and an IDS. The fourth one serves as the zonal controller. One of the three ECUs hosts a vulnerable task susceptible to a buffer overflow attack triggered by receiving malicious data as an external message. Subsequently, the agent initiates the migration process in response to the attack.

2) Results: We conducted the experiments $100\times$. We measured the local end-to-end latency L starting from the agent's request for a task migration to the receipt of the outcome from the orchestrator. We also measured t^s_{agent} (the maximum time any agent needs to compute its H^m_j and S^m_j) and t^s_{orch} (the maximum time needed by the orchestrator on any zonal controller to solve its constraints). As Fig. 6 depicts, the orchestrator requires less than 0.2 ms to identify the optimal destination location (if it exists). Similarly, the agent requires a relatively short time to compute H^m_j and S^m_j (less than 0.7 ms). To put this in context, the end-to-end latency is between 13 and 17 ms in most cases. This further demonstrates the efficacy of SHIFTGUARD as the end-to-end latency for CAN messages ranges from a few milliseconds to seconds [46], [47]. Our experiments suggest that the most time-consuming part is attributed to the message exchanges between agents and the orchestrator, which is also influenced by the communication link (see Section VI for a compassion of the latency when using Ethernet instead of CAN).

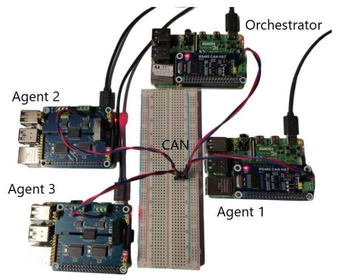


Fig. 5. Hardware testbed setup with four Raspberry Pi units: one representing the orchestrator and three representing agents, all are interconnected via CAN bus.

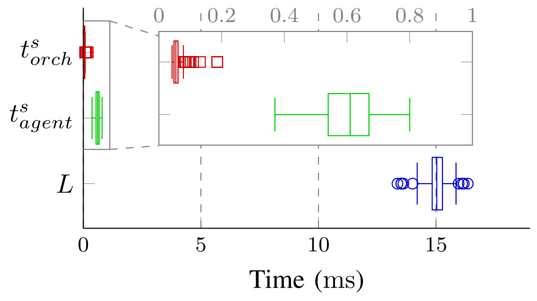


Fig. 6. SHIFTGUARD performance on hardware testbed. The solver on the orchestrator t^s_{orch} and agents t^s_{agent} consumes less than 0.2 and 0.8 ms, respectively, to find a migration host for a setup with 15 tasks.

Key Findings. SHIFTGUARD is deployable in a commodity automotive setup with protocols such as CAN. The overhead of SHIFTGUARD is minimal, with the orchestrator computation time under 0.2 ms and agent processing time less than 0.7 ms.

V. PARAMETERIZING SHIFTGUARD

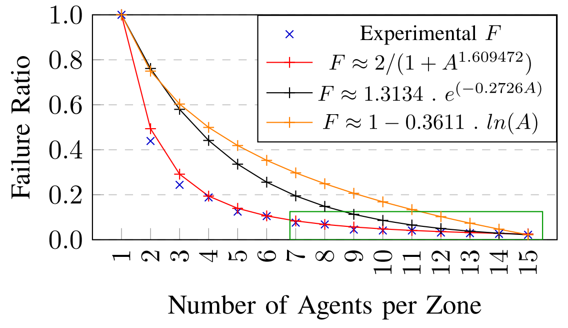
Our experiments show that SHIFTGUARD can guarantee fulfilling up to 100% of all the migration requests while only 33%–13% of all the migration requests may suffer from an end-to-end latency $\leq E$. About 67%–87% of all the migration requests will be fulfilled within a latency $\leq L$, which is at least $30\times$ shorter than E. However, all the results are functions of A and Z. We can formulate an MILP optimization problem to further evaluate the tradeoffs between A and Z.

Objective:

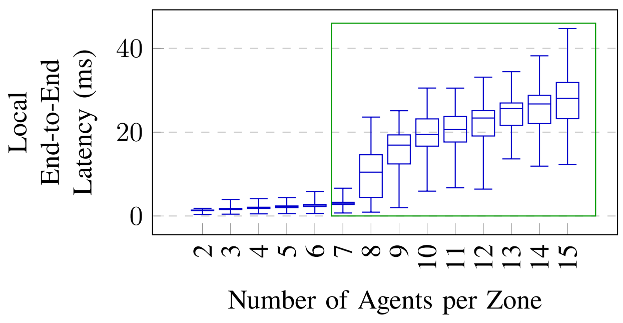
$$\min_{A,Z} \; FE - (1 - F)L. \tag{12}$$

Constraints:

$$0 < F = f(A, S, \Lambda) \leq 1 \tag{13}$$



(a) Failure Ratio F vs. Number of Agents A



(b) Local End-to-End Latency L vs. Number of Agents A

Fig. 7. Failure ratio to accept a migration request within the local zone F and the local end-to-end latency L as functions of the number of agents (ECUs) per zone A. The blue curve in (a) represents our results. The red curve illustrates the approximation of the results represented by the power-law-based fit function $(2/(1 + A^{1.609472}))$, which is the best representation of our data. The black and the orange curves represent the exponential decay and the logarithmic fit functions, respectively. The green rectangle in (a) highlights the recommended number of agents per zone. In (b), green rectangle signifies the corresponding end-to-end latency for the recommended values of A.

$$L \leq A\left(\text{Com}_{\text{agent}\rightarrow\text{orch}} + \text{Com}_{\text{orch}\rightarrow\text{agent}} + t^{s}_{\text{agent}}\right) + t^{s}_{\text{orch}} \tag{14}$$

$$E \leq Z \cdot L + Z \cdot \left(\text{Com}_{\text{gate}\rightarrow\text{mgate}} + \text{Com}_{\text{mgate}\rightarrow\text{gate}}\right) + t^{s}_{\text{orch}} \tag{15}$$

$$A \times Z \leq M : A \in \mathbb{N}^{*} \wedge Z \in \mathbb{N}^{*} \leq A. \tag{16}$$

In the above equations, S represents the trust scores of the ECUs, and Λ represents the load on the ECUs when agents receive the hosting requests. In this optimization problem, the failure ratio F and the local end-to-end latency L are functions of the number of agents in one zone, i.e., A, as (13) and (14) show. While the values of F decrease for the larger number of agents, L actually increases. To illustrate this relationship and its impact on the optimization, we computed F and L for $A = \{2, \ldots, 15\}$. For each value of A, we ran the experiment 1000×. Fig. 7 presents the results. For instance, Fig. 7(a) confirms that having more ECUs in one zone can increase the probability of accepting the migration request locally, i.e., reducing F significantly. However, the improvement becomes slower after $A > 7$. For $A = 7$, we obtained $F = 0.076$, and for $A = 8$, we have $F = 0.068$. The local end-to-end latency L will increase dramatically: for $A = 7$, the latency $L \leq 6.63$, while for $A = 8$, the latency increases to $L \leq 23.61$. Hence, we select $A = 7$ as an optimal value in our experiments (Section IV-B). F is a function of other parameters, namely, the load on the ECU and the trust scores of the ECU. Both change dynamically due to the migration of tasks and cyberattacks. We chose to explore the power-law model because it naturally captures diminishing returns: as the number of ECUs per zone increases, the failure ratio decreases, but with smaller and smaller improvements. This behavior aligns with long-tail decay patterns seen in power-law distributions. Therefore, we show here an example of defining a fit curve of the observed results. We followed three well-known models: power law $(\alpha/(1 + A^{\beta}))$, exponential decay $(\alpha \cdot e^{(\beta A)})$, and the logarithmic $(\alpha + \beta \ln(A))$. As any safe fit curve should be above all the experimental data, we added this constraint to the fit function. The three curves are illustrated in Fig. 7(a). We computed the coefficient of determination (R^2) and the root-mean-squared error (RMSE) of each fit curve. The

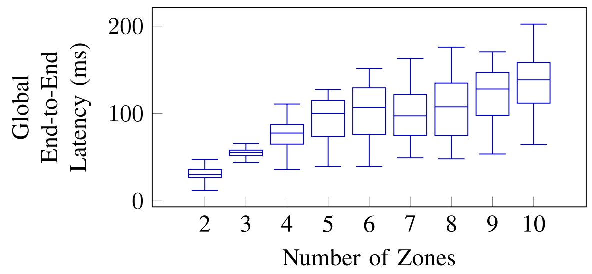


Fig. 8. Relationship between the global end-to-end latency E and the number of zones Z when $A = 7$, showing an increase in E with the increase in Z. The figure shows the scalability of SHIFTGUARD up to 70 ECUs ($Z = 10$ and $A = 7$).

power-law-based fit curve explains the experimental data very well ($R^2 = 0.9937$, RMSE $= 0.0196$). Note that the MILP is sensitive to the failure ratio F. Therefore, the designer should select the best safe fit curve for it.

The second parameter in the optimization is the number of zones Z. Hence, we fix the number of agents to $A = 7$. From (15), the global end-to-end latency is a function of Z. The relationship between E and Z is illustrated in Fig. 8 for $Z = \{2, \ldots, 10\}$. For each value of Z, we ran the experiment 1000×. Our experiments find the suitable number of zones for a given latency constraint. For example, if $E \leq 150$ ms, SHIFTGUARD should not use more than $Z = 6$ zones. This experiment shows the scalability of SHIFTGUARD up to 70 ECUs ($Z = 10$ and $A = 7$).

VI. DESIGN CONSIDERATIONS

A. Lack of Migration Candidates

There may be cases when no feasible host is available for migration, either locally (within the same zone) or globally (across different zones). To investigate the likelihood of such a scenario, we analyzed the global failure probability (ϕ) across two–ten zones. We ran the experiment 1000× for each zone. Our findings, illustrated in Fig. 9, suggest that the probability of such an event is *minimal* for most cases and *almost impossible* for systems with a higher number of

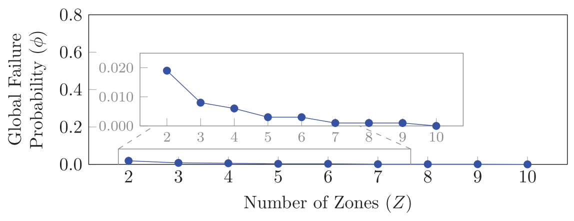


Fig. 9. Global failure probability (ϕ) when using different numbers of zones (Z). The ϕ remains negligible across varying zone numbers (Z).

zones (viz., seven or more). However, despite its rarity, safety-critical automotive systems must have functionalities to tackle such events. One approach is sacrificing the *least critical task* among those present on one of the ECUs within the same zone, provided that the task earmarked for migration is not the least critical one. We note that sacrificing such a task does *not* mean stopping the task completely for the whole duration till receiving an update. Instead, it keeps working without guaranteeing to meet deadlines for all jobs (note: often time-critical systems can tolerate a few deadline misses [48]).

B. Stress Testing Under Multi-ECU Attacks

Another scenario where finding suitable migration hosts becomes challenging is when attackers exploit multiple tasks on different ECUs at the same time [49]. While such attacks are harder to implement in practice, SHIFTGUARD can handle concurrent migration requests. However, supporting such requests is limited by the availability and capacity of uncompromised ECUs to host the tasks requiring migration. To examine this scenario in more detail, we conducted a new experiment. We modeled a system with $Z = 6$ zones, each containing $A = 7$ agents. We assumed that one zone is under attack, with two, three, or four ECUs compromised. In addition, the ECUs were placed under high-load conditions with utilization levels $U = 0.7, 0.8, 0.9$. All other experimental settings followed those in previous evaluations. We measured the local failure ratio F (i.e., failure to find a host in the same zone) and the global failure probability ϕ (i.e., failure to find a host anywhere in the system). Each configuration was tested with 100 000 runs. The results, shown in Fig. 10, indicate that system load has a major impact on the ability of SHIFTGUARD to find a valid migration target. In extreme cases where $U = 0.9$, the local failure ratio exceeded 95.23%, and the global failure probability also increased significantly, with $\phi \geq 68.64\%$. Interestingly, the number of compromised ECUs within the attacked zone had a smaller impact than expected. Across all the tested values (two–four compromised ECUs), the availability of suitable migration targets remained more strongly correlated with system load than with the number of attacks.

C. When to Perform Migration

Migration can occur when the vehicle is *fail-operational* or *fail-safe* modes. For instance, when the vehicle is safely stopped/parked (i.e., the bus usage is minimal), a migration may trigger (fail-safe). However, it does not preclude migration events from occurring when the vehicle is running if the bus usage and message latency are acceptable (fail-operational). The choice of operation (by the designers) depends on the corresponding end-to-end latency (either in Phase I or Phase II) and the real-time requirements of the migrated tasks. For instance, if the latency is less than the job's deadline, migration can occur in fail-operational mode. Conversely, if the latency exceeds the job's deadline, the system may switch to fail-safe mode to ensure safety. In such a case, migration should be performed when the car is parked to ensure the availability of the bus while also facilitating all the required message exchanges. We note that the communication link influences end-to-end latency. To emphasize this point, we repeated the measurement conducted in Section IV-C on the hardware testbed, using Ethernet to connect different ECUs and the gateway instead of CAN. The higher bitrate of Ethernet results in a 50% reduction in end-to-end latency compared with CAN. The results are shown in Fig. 11. However, it is important to note that network congestion or degraded link conditions can unpredictably increase latency, which may delay or temporarily prevent migration. In such cases, the system can defer migration until conditions improve by switching to fail-safe mode, where the vehicle is in a controlled state (e.g., parked), ensuring stable communication and safe execution of the migration process.

D. Security Considerations

We assume a host-based IDS to detect task-level anomalies, which serves as the trigger for task migration. While the internal design of the IDS is beyond the scope of this work, we acknowledge that IDS imperfections can affect system behavior. False negatives may delay or suppress necessary migrations, potentially compromising safety by allowing unsafe task colocation. False positives, on the other hand, may lead to unnecessary migrations, increasing both computational and communication overhead. These issues become more serious if false positives occur frequently. We further stress that a good IDS should be cognizant of (repeated) false positives. While we did not explicitly model repeated IDS errors, Fig. 10 partially reflects this case by simulating multiple concurrent migration triggers under various system load conditions. These stress scenarios show what could happen if the IDS produces frequent alerts, whether accurate or not. The results show that SHIFTGUARD maintains stable operation in these high-load conditions. However, testing system behavior under targeted IDS misbehavior, such as a series of low-confidence or incorrect alerts, is left for future work. It is also possible to include safeguards such as minimum-confidence thresholds to reduce the impact of frequent or noisy IDS outputs. Finally, while the migration mechanism enhances security, it also momentarily expands the system's attack surface by introducing additional message exchanges and control logic. However, these communications are authenticated and integrity-protected using standard in-vehicle security protocols, and trusted components (IDS, agent, orchestrator) are isolated from untrusted tasks, limiting the risk of exploitation during migration.

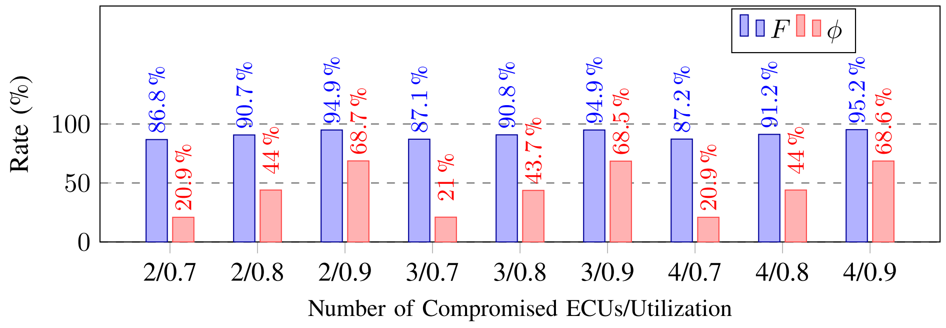


Fig. 10. Comparison of the failure ratio (F) and the global failure probability ϕ versus Number of compromised ECUs under high load.

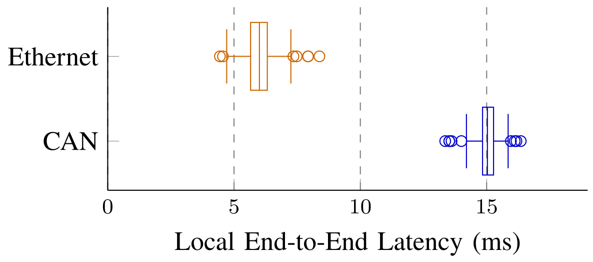


Fig. 11. Comparison of end-to-end migration latency L using CAN and Ethernet. Ethernet achieves approximately 50% lower latency compared with CAN due to higher bitrate.

VII. Related Work

Task migration is primarily used to ensure safety [10], [12], [13] and optimize resource utilization [50], [51] by intelligently redistributing computational tasks among various ECUs within the same vehicle. Various techniques exist to address anomalous tasks and ensure safety. One approach is to deploy redundant ECUs for critical tasks, either on a one-to-one basis [9] or using a single redundant ECU for groups of ECUs [13]. However, these solutions pose challenges such as increased costs or the potential bottleneck of a single node serving multiple others. Other solutions include maintaining backup instances of tasks on predefined ECUs [12] or dedicated backup hardware [13]. However, these techniques result in a single point of failure (for instance, when the ECU is compromised) and are also limited to protecting only a small number of tasks (specifically, those with a backup instance). Another possibility is to integrate extra hardware and maintain a backup instance for every task across different ECUs [52]. When an ECU fails, its tasks' backup instances migrate to the extra hardware. But this approach is limited in migrating only a few tasks and faces scalability issues.

Limited literature exists on the task migration problem without redundant hardware or instances. For example, Baik et al. [50] propose a task migration approach where low-criticality tasks are migrated to neighboring devices to ensure that high-criticality tasks have sufficient resources and can meet their deadlines. Nevertheless, such a solution does not consider the case of migrating high-criticality tasks to another ECU for any reason. Another solution [11] proposes to migrate critical tasks from a failed ECU to replaceable ECUs using network connections. However, this approach does not scale well, as migration is confined to the ECU with a connection to the failed ECU. In addition, it relies on stable network connectivity, which could be a potential point of failure.

A. Comparative Analysis

Table II shows comparison of ShiftGuard with existing research. The comparison is based on five criteria: 1) whether they are built for (or can be adaptable to) SDVs ("SDV Compatibility" column in Table II); 2) the need for *redundancy* in hardware or software components ("Redundancy" column); 3) the *dynamicity* of the operation and its capacity to migrate different tasks (not predefined or limited set of tasks) at run time ("Dynamic" column); 4) the *distributed* nature of the solution, allowing for migration without reliance on a single component to make the decision ("Distributed" column); and 5) the *security* awareness for the migration process, i.e., whether the migration is intended to minimize security threats ("Security" column).

Unlike many existing solutions [12], [13], [52], ShiftGuard does not require any dormant instances of tasks or backup hardware. Thus, it avoids any overhead inherited from redundant hardware or software. In addition, it adapts to runtime changes and is not limited to the predefined migration policy as considered in prior work [11], [12], [13], [52]. In addition, our solution was built in a distributed nature and avoids needing a single node to decide the migration process, as was the case in earlier research [10], [11], [13], [50]. Finally, ShiftGuard is designed to improve security posture, using novel metrics (e.g., "trust score" and "security level"), unlike most work focusing on task migration for safety or resource utilization purposes.

One could argue that many of these solutions can be extended to cover security awareness. This is not always true—solutions that use redundant software components or hardware [12], [13], [52] are limited to protecting these specific tasks or are restricted by the capacity of the redundant hardware. Other solutions that aim to migrate noncritical tasks only [50] cannot be extended for security. This is because migrating noncritical tasks—which are more vulnerable to attacks—without considering the security requirements of existing tasks on the host ECU could potentially introduce a new attack surface on the destination ECU. Finally, most solutions are not compatible with SDV from the ground up, but some can be retrofitted, while our proposed solution is

TABLE II
COMPARISON OF SHIFTGUARD TO THE STATE OF THE ART

Solution	SDV Compatibility	Redundancy	Dynamic	Distributed	Security
Baik et al. [50]	◔	None	●	○	○
Baik et al. [11]	◔	None	○	○	○
Chen et al. [52]	○	HW & SW	○	◑	○
Delgadillo et al. [10]	◔	None	◑	○	○
Huang et al. [13]	○	HW	○	○	○
Weiss et al. [12]	◔	SW	○	●	○
SHIFTGUARD	●	None	●	●	●

● Yes ◑ Partially or N/A ◔ Not from ground up, but can be retrofitted ○ No

built with SDV capabilities in mind. We stress 1) the differing nature of SHIFTGUARD from existing work; 2) the difficulty in reproducing previous results and lack of open-source releases; and 3) the use of different platforms make empirical comparison between our work and the other research discussed above unfeasible and irrelevant. In addition, as most prior work was not originally designed with security in mind, direct security comparisons are limited. Instead, we highlight how ShiftGuard fills this gap by explicitly incorporating security metrics into the migration process.

VIII. CONCLUSION

This article introduces SHIFTGUARD, a distributed, security-aware task migration mechanism for SDVs. It aims to facilitate task migration in response to security exploits targeting tasks on ECUs without relying on hardware or software redundancy. Our experiments on real hardware testbeds show that SHIFTGUARD can be deployed with low overhead (in the millisecond range). In addition, our comprehensive synthetic test cases show that any task migration request has a probability of 76%–100% to be successfully fulfilled and scaled to 70 ECUs in ten zones. Using techniques such as those proposed in this work, modern software-controlled vehicles can maintain operational efficiency, safety, and security when a task is compromised. As a result, vendors can minimize downtime to fix the vulnerabilities that led to the breach.

APPENDIX

A. Table of Notations

The list of mathematical notations used in the paper is summerized in Table III.

B. Agent and Orchestrator Roles

Algorithms 1 and 2 present how the SHIFTGUARD is operated from the agents' and the orchestrators' perspective and the different roles they play during the protocol. The color used in each part of the algorithm represents different roles for the agent (as a migration requester on the local ECU or on a potential host ECU) and for the orchestrator in various locations (on the local zonal controller, another zonal controller, or the central computing unit). The colors correspond to those used for the agent and orchestrator in Fig. 2.

TABLE III
KEY MATHEMATICAL NOTATIONS USED IN THIS WORK

Not.	Description
τ_ℓ	Task
T_ℓ	Period of τ_ℓ
D_ℓ	Relative deadline of τ_ℓ
pri_ℓ	Priority of τ_ℓ
C_ℓ^i	Worst-case execution time of τ_ℓ when it is scheduled on ϵ_i
ω_ℓ	A weighting factor based on the criticality τ_ℓ
G_ρ	Direct acyclic graph (DAG)
D_ρ^{e2e}	End-to-end deadline of G_ρ
R_ρ^{e2e}	End-to-end response time of G_ρ
ϵ_i	ECU_i
σ_ℓ^i	The current security level of τ_ℓ on ϵ_i
C_{IDS}^i	The confidence level of the IDS on ϵ_i
FPR^i	The false positive rate of IDS on ϵ_i
I_ℓ	The impact of the attack on task τ_ℓ
R_ℓ	Worst-case response time of τ_ℓ
$R\sigma_\ell$	The required security level of task τ_ℓ
S_i	Trust score of the ϵ_i
Z	Number of zones in the network
H_j^m	Hospitality, the ability to accommodate task that seeks to migrate τ_m on the ϵ_j
A	Number of agents, i.e., ECUs in one zone
S_j^m	Trust score of the ϵ_j considering the task that seeks to migrate τ_m
F	The failure ratio to accept the migration request in the same (local) zone
E	The end-to-end latency between sending the migration request and receiving the response across multiple zones
L	The end-to-end latency between sending the migration request and receiving the response within one zone
Φ	The global failure probability

Algorithm 1 shows the two roles of the agent (*requester* and *responder*). Each agent checks continuously whether there is any task τ_k that is compromised (Line 3). If this is the case, the agent recomputes the trust score of the ECU S_i and checks whether the required security level $R\sigma_m$ is still satisfied (Lines 5–7). If there exists a task τ_m for which $R\sigma_m$ is violated, the agent moves it from **Step 1** to **Step 2** in the SHIFTGUARD protocol. In this step, a migration request is initialized (Line 9) and sent to the local zonal controller (Line 11). The agent waits for an answer from the local zonal controller (Line 13) in **Step 7**. Finally, if SHIFTGUARD managed to find a new host that satisfies the required security level of τ_m, the agent moves to **Step 8** and starts the migration process (Lines 14 and 15). If all the tasks are secure on ϵ_j, the agent on that ECU may receive (**Step 3** and **Step 6.3**) a request from the local zonal controller to host a task (Line 21). The agent first checks the schedulability of the hosted tasks considering the

Algorithm 1 SHIFTGUARD Operation—Agent

```
1  /* Agent as requester on $\epsilon_i$ */
2  /* $\tau_k$ reported compromised by the IDS */
3  if $is\_compromised(\tau_k)$ then
4      /* Calculate trust score using Eq. (2) */
5      $S_i \leftarrow \sum_{\forall m \mapsto i} \omega_m \cdot \sigma_m^i$
6      for $\tau_m : m \mapsto i$ do
7          if $R\sigma_m > S_i$ then
8              /* Generate a migration request message */
9              $msg \leftarrow \{T_m, D_m, prim_m, (C_m^0, \ldots, C_m^{M-1}), R\sigma_m\}$
10             /* Send a request to the orchestrator */
11             $send\_request(msg)$
12             /* Receive the response from the orchestrator */
13             $dest \leftarrow receive\_response()$
14             if $dest \neq None$ then
15                 $migrate(\tau_m, dest)$
16                 $return$
17             end
18         end
19     end
20 end
21 /* Agent as host on $\epsilon_j$ */
22 $msg \leftarrow receive\_orch\_request()$
23 $schedulability\_test(\{\forall \ell \mapsto j\} \bigcup \{m\})$
24 if $R_m \leq D_m$ then
25     /* Calculate hospitality using Eq. (3) */
26     $H_j^m \leftarrow K - \sum_{\forall l \mapsto j \, \wedge \, R_l > D_l \, \wedge \, l \text{ is low-critical}} \kappa_l - \sum_{\forall h \mapsto j \, \wedge \, R_h > D_h \, \wedge \, h \text{ is high-critical}} K.\kappa_h$
27     /* Calculate the trust score (Eq. 4) */
28     $S_j^m \leftarrow \sum_{\forall \ell \mapsto j \, \wedge \, R_\ell \leq D_\ell} \omega_\ell \cdot \sigma_\ell^j$
29 else
30     /* Not possible to host the task, flag with $None$ */
31     $H_j^m \leftarrow None$
32     $S_j^m \leftarrow None$
33 end
34 /* Send the response to the orchestrator */
35 $send\_agent\_response(\{H_j^m, S_j^m\})$
```

Algorithm 2 SHIFTGUARD Operation—Orchestrator

```
1  /* Orchestrator on the local zonal controller */
2  $msg \leftarrow receive\_request()$
3  foreach $\alpha \in Agents$ do
4      /* Send the request to each agent */
5      $send\_orch\_request(msg)$
6      /* Receive agent's response: hospitality and trust score */
7      $(\mathcal{H}, \mathcal{S}) \leftarrow receive\_agent\_response()$
8  end
9  /* Find a destination by solving Eqs. (5)-(9) */
10 $dest \leftarrow find\_dest(\mathcal{H}, \mathcal{S})$
11 if $dest \neq None$ then
12     $send\_response(dest)$
13 else
14     /* Forward its ID and the request to central computing
         unit */
15     $send\_ccu\_request(\zeta_{id}, msg)$
16     $dest = receive\_ccu\_response()$
17     $send\_response(dest)$
18 end
19 /* Orchestrator on the central computing unit */
20 $(\zeta_{id}, msg) \leftarrow receive\_ccu\_request()$
21 foreach $\zeta \in Zones$ do
22     $send\_zone\_request(msg)$
23     $(\mathcal{H}, \mathcal{S}) \leftarrow receive\_zone\_response()$
24 end
25 /* Find a destination by solving Eqs. (5)-(9) */
26 $dest \leftarrow find\_dest(\mathcal{H}, \mathcal{S})$
27 /* Send the response to the requested zonal controller $\zeta_{id}$ */
28 $send\_ccu\_response(\zeta_{id}, dest)$
29 /* Orchestrator on another zonal controller */
30 $msg \leftarrow receive\_zone\_request()$
31 foreach $\alpha \in Agents$ do
32     /* Send the request to each agent in the zonal controller
         */
33     $send\_orch\_request(msg)$
34     /* Receive agent's response: hospitality and trust score */
35     $(\mathcal{H}, \mathcal{S}) \leftarrow receive\_agent\_response()$
36 end
37 /* Find a destination by solving Eqs. (5)-(9) */
38 $find\_dest(\mathcal{H}, \mathcal{S})$
39 $send\_zone\_response(\{H_j^m, S_j^m\})$
```

new task (Line 22). If the new task can meet its deadline, the agent computes the hospitality and security parameters (Lines 23–27); otherwise, the agent rejects the hosting request by assigning None (Lines 28–32). Lines 22–31 represent **Step 4** and **Step 6.4**. Next, the agent sends the response back to the zonal controller (Line 34) in **Step 5** and **Step 6.5**, see Fig. 2.

Algorithm 2 illustrates the roles of the orchestrator based on their location (i.e., on the **central computing unit**, **local zonal controller**, or **another zonal controllers**). When the orchestrator is running on a zonal controller, it may receive a migration request from a local agent (Line 2) as part of **Step 2**. The orchestrator forwards the message to the other local agents as a *host* request (Lines 3–5), which is **Step 3** in the protocol. In Line 7, the orchestrator waits and collects all the values of hospitality and security (**Step 5**). Using optimization formulation in (5)–(9), the orchestrator tries to find the best ECU to host the task τ_m (Line 10). This step is **Step 6** in the protocol. If a destination is defined, the orchestrator sends it to the agent, which initializes the migration request (Lines 11 and 12). That completes **Step 7**. If the migration request is not accepted locally, **Step 7** will be delayed and instead, the orchestrator on the local zonal controller will initialize a migration request and send it to the central computing unit (Lines 13–15). This action is **Step 6.1**. The orchestrator then will be waiting for the response from the central computing unit (Line 16), which comes in **Step 6.9**. In **Step 7**, the orchestrator forwards the answer to the dedicated agent (Line 17). The orchestrator on the central computing unit may receive a migration request from a zonal controller (Line 20). If this is the case, the orchestrator forwards the message as a *host* request to all other zonal controllers (Lines 21 and 22) in **Step 6.2**. The orchestrator on the central computing unit waits and collects (Line 23) the responses in **Step 6.7**. Then, in **Step 6.8**, it solves (5)–(9) to find the best destination for the task τ_m (Line 26). Finally, the orchestrator on the central computing unit sends the response to the orchestrator on the zonal controller (Line 28) in **Step 6.9**. The last role the orchestrator may play is on a zonal controller when it receives a host request from the central computing unit (Line 30). In this role, the orchestrator forwards the request to the agents on the directly connected ECUs (Lines 31–33) in **Step 6.3**. Like other roles, the orchestrator collects the hospitality and security values, computes potential destination, and sends its hospitality and security values as a response to the central computing unit (Lines 35–39) in **Step 6.5**–**Step 6.7**, respectively.

Acknowledgment

Any findings, opinions, recommendations, or conclusions expressed in this article are those of the authors and do not necessarily reflect the sponsors' views.

References

[1] Z. Liu, W. Zhang, and F. Zhao, "Impact, challenges and prospect of software-defined vehicles," *Automot. Innov.*, vol. 5, no. 2, pp. 180–194, Apr. 2022.

[2] P. E. Ross, "How software is eating the car," IEEE Spectr., New York, NY, USA, Tech. Rep., 2020.

[3] K. Huang, B. Hu, L. Chen, A. Knoll, and Z. Wang, "Adas on Cots with OpenCL: A case study with lane detection," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 559–565, Apr. 2018.

[4] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," Black Hat, Las Vegas, NV, Tech. Rep., 2014, p. 94.

[5] A. Burns and R. Davis, "Mixed criticality systems—A review," Dept. Computer Science, University of York, York, U.K., Tech. Rep., 2013.

[6] *ENISA Good Practices for Security of Smart Cars*, Eur. Union Agency Netw. Inf. Secur. (ENISA), Athens, Greece, 2019.

[7] M. Hamad, M. Tsantekidis, and V. Prevelakis, "Red-zone: Towards an intrusion response framework for intra-vehicle system," in *Proc. 5th Int. Conf. Vehicle Technol. Intell. Transp. Syst.*, 2019, pp. 148–158.

[8] M. Hamad et al., "REACT: Autonomous intrusion response system for intelligent vehicles," *Comput. Secur.*, vol. 145, Oct. 2024, Art. no. 104008.

[9] P. Bergmiller, *Design and Safety Analysis of a Drive-by-Wire Vehicle*. Berlin, Germany: Springer, 2013.

[10] O. Delgadillo, B. Blieninger, J. Kuhn, and U. Baumgarten, "An architecture to enable machine-learning-based task migration for multi-core real-time systems," in *Proc. IEEE 14th Int. Symp. Embedded Multicore/Many-Core Syst.-Chip (MCSoC)*, Dec. 2021, pp. 405–412.

[11] J. Baik, H. Jeong, and K. Kang, "Poster: Network-centric approach using task migration for drive-by-wire vehicle resilience," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2020, pp. 1–2.

[12] P. Weiss and S. Steinhorst, "Predictable timing behavior of gracefully degrading automotive systems," *Design Autom. Embedded Syst.*, vol. 27, nos. 1–2, pp. 103–138, Jun. 2023.

[13] H. Huang, K.-L. Leu, and Y.-Y. Chen, "An effective multiple-level fault-tolerant framework for FlexRay network systems," in *Proc. Int. Conf. Connected Vehicles Expo (ICCVE)*, Oct. 2015, pp. 54–55.

[14] F. Zhao, H. Song, and Z. Liu, "Identification and analysis of key technical elements and prospects for software-defined vehicles," in *Proc. Intell. Connected Vehicles Symp. (ICVS)*, Jan. 2022.

[15] M. De Vincenzi et al., "Contextualizing security and privacy of software-defined vehicles: State of the art and industry perspectives," 2024, *arXiv:2411.10612*.

[16] M. Haeberle et al., "Softwarization of automotive E/E architectures: A software-defined networking approach," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Dec. 2020, pp. 1–8.

[17] M. Bornemann. (Jan. 2021). *Zone Controllers Build Bridge to Tomorrow's Technology*. Accessed: Oct. 25, 2024. [Online]. Available: https://www.aptiv.com/docs/default-source/white-papers/2021_aptiv_whitepaper_zonecontroller.pdf

[18] M. Hamad, Z. A. H. Hammadeh, S. Saidi, V. Prevelakis, and R. Ernst, "Prediction of abnormal temporal behavior in real-time systems," in *Proc. 33rd Annu. ACM Symp. Appl. Comput.*, Apr. 2018, pp. 359–367.

[19] B. Blazevic, M. Peter, M. Hamad, and S. Steinhorst, "TEEVseL4: Trusted execution environment for virtualized seL4-based systems," in *Proc. IEEE 29th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2023, pp. 67–76.

[20] C. Göttel, P. Felber, and V. Schiavoni, "Developing secure services for IoT with OP-TEE: A first look at performance and usability," in *Proc. 19th IFIP WG 6.1 Int. Conf. Distrib. Appl. Interoperable Syst.*, Jun. 2019, pp. 170–178.

[21] C. Plappert, A. Fuchs, and R. Heddergott, "Analysis and evaluation of hardware trust anchors in the automotive domain," in *Proc. 17th Int. Conf. Availability, Rel. Secur.*, Aug. 2022, pp. 1–11.

[22] S. K. Valappil, L. Vogel, M. Hamad, and S. Steinhorst, "Advanced IDPS architecture for connected and autonomous vehicles," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2024, pp. 1779–1785.

[23] J. Rox and R. Ernst, "Compositional performance analysis with improved analysis techniques for obtaining viable end-to-end latencies in distributed embedded systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 15, no. 3, pp. 171–187, Jun. 2013.

[24] S. Stachowski, R. Gaynier, and D. J. LeBlanc, "An assessment method for automotive intrusion detection system performance," Transp. Res. Inst. (UMTRI), Univ. Michigan, Ann Arbor, MI, USA, Tech. Rep. DOT HS 812 708, 2019.

[25] Road Vehicles—Cybersecurity Engineering, ISO/SAE Standard 21434:2021, 2021. [Online]. Available: https://www.iso.org/standard/70918.html

[26] ISO 26262: Road Vehicles—Functional Safety, Standard 26262, ISO, Geneva, Switzerland, 2018.

[27] S. Checkoway et al., "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Secur. Symp. (USENIX Secur. 11)*, Aug. 2011, p. 6.

[28] X. Sun, F. R. Yu, and P. Zhang, "A survey on cyber-security of connected and autonomous vehicles (CAVs)," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6240–6259, Jul. 2022.

[29] Y. Li, M. Liu, C. Cao, and J. Li, "Communication-traffic-assisted mining and exploitation of buffer overflow vulnerabilities in ADASs," *Future Internet*, vol. 15, no. 5, p. 185, May 2023.

[30] "Mercedes-Benz MBUX Security Research Report," Tencent Secur. Keen Lab, Shenzhen, China, Tech. Rep., 2021.

[31] F. Sommer, J. Dürrwang, and R. Kriesten, "Survey and classification of automotive security attacks," *Information*, vol. 10, no. 4, p. 148, Apr. 2019.

[32] T. Limbasiya, K. Z. Teng, S. Chattopadhyay, and J. Zhou, "A systematic survey of attack detection and prevention in connected and autonomous vehicles," *Veh. Commun.*, vol. 37, Oct. 2022, Art. no. 100515.

[33] W. Wu et al., "A survey of intrusion detection for in-vehicle networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 3, pp. 919–933, Mar. 2020.

[34] B. Lampe and W. Meng, "A survey of deep learning-based intrusion detection in automotive applications," *Expert Syst. Appl.*, vol. 221, Feb. 2023, Art. no. 119771. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423002725

[35] S. Wang, H. Zhou, H. Zhao, Y. Wang, A. Cheng, and J. Wu, "A zero false positive rate of IDS based on Swin transformer for hybrid automotive in-vehicle networks," *Electronics*, vol. 13, no. 7, p. 1317, Mar. 2024.

[36] P. Lapczynski, "Achieving a root of trust with secure boot in automotive RH850 and R-car devices—Part 1," Renesas Electron. Corp., Tokyo, Japan, Tech. Rep., 2021.

[37] A. Anwar, A. Anwar, L. Moukahal, and M. Zulkernine, "Security assessment of in-vehicle communication protocols," *Veh. Commun.*, vol. 44, Dec. 2023, Art. no. 100639.

[38] M. De Vincenzi et al., "A systematic review on security attacks and countermeasures in automotive Ethernet," *ACM Comput. Surveys*, vol. 56, no. 6, pp. 1–38, Jun. 2024.

[39] M. Hamad, J. Schlatow, V. Prevelakis, and R. Ernst, "A communication framework for distributed access control in microkernel-based systems," in *Proc. 12th Annu. Workshop Operating Syst. Platforms Embedded Real-Time Appl. (OSPERT)*, Jul. 2016, pp. 11–16.

[40] C. A. Floudas, *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. London, U.K.: Oxford Univ. Press, 1995.

[41] S. Saudrais and K. Chaaban, "Automatic relocation of AUTOSAR components among several ECUs," in *Proc. 14th Int. ACM Sigsoft Symp. Compon. based Softw. Eng.*, Jun. 2011, pp. 199–204.

[42] AUTOSAR. (Nov. 2021). *Specification Manifest*. [Online]. Available: https://www.autosar.org/fileadmin/standards/R21-11/AP/AUTOSAR_TPS_ManifestSpecification.pdf

[43] R. Wilhelm et al., "The worst-case execution-time problem—Overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, pp. 1–53, Apr. 2008.
[44] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005.
[45] Renesas Electronics.(2021). *Renesas R-Car V3H*. Accessed: Feb. 5, 2025. [Online]. Available: https://www.renesas.com/en/document/fly/renesas-r-car-v3h-r33pf0004ed
[46] M. D. Pesé, J. W. Schauer, J. Li, and K. G. Shin, "S2-CAN: Sufficiently secure controller area network," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2021, pp. 425–438.
[47] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, no. 3, pp. 239–272, Apr. 2007.
[48] Z. A. H. Hammadeh, S. Quinton, and R. Ernst, "Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 6, pp. 1–25, Dec. 2019, doi: 10.1145/3356865.
[49] K. Serag, R. Bhatia, V. Kumar, Z. B. Celik, and D. Xu, "Exposing new vulnerabilities of error handling mechanism in CAN," in *Proc. 30th USENIX Secur. Symp. (USENIX)*, Aug. 2021, pp. 4241–4258. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/serag
[50] J. Baik, J. Lee, and K. Kang, "Task migration and scheduler for mixed-criticality systems," *Sensors*, vol. 22, no. 5, p. 1926, Mar. 2022.
[51] T. Megel, R. Sirdey, and V. David, "Minimizing task preemptions and migrations in multiprocessor optimal real-time schedules," in *Proc. 31st IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 37–46.
[52] Y.-Y. Chen and C.-M. Lyu, "ECU-level fault-tolerant framework for safety-critical FlexRay network systems," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Oct. 2012, pp. 553–558.

Mohammad Hamad received the Ph.D. (Dr.-Ing.) degree in computer engineering from the Institute for Data Technology and Communication Networks, Technical University of Braunschweig, Braunschweig, Brunswick, Germany, in 2020.

He is a Research Group Leader with the Embedded Systems and the Internet of Things Professorship, Technical University of Munich, Munich, Germany. His research interests include autonomous system security.

Zain A. H. Hammadeh received the Ph.D. (Dr.-Ing.) degree from TU Braunschweig, Brunswick, Germany, in 2019.

In 2019, he joined the Institute of Software Technology, German Aerospace Center (DLR), Brunswick. His research interests include real-time computing systems and space cybersecurity.

Davide Alessi received the Bachelor of Computer Science degree from the University of Rome Tor Vergata, Roma, Italy, in 2020, and the Master of Informatics degree from Technical University of Munich, Munich, Germany, in 2024.

His research interests include vehicle security testing.

Monowar Hasan received the M.S. degree in electrical and computer engineering from the University of Manitoba (UM), Winnipeg, MB, Canada, in 2015, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, in 2020.

He is a Computer Science Assistant Professor at Washington State University (WSU), Pullman, WA, USA. His current research interests include exploring security and resiliency of cyber-physical systems.

Mert D. Pesé received the dual B.S. degree in electrical engineering and computer science and the M.S. degree in electrical engineering from the Technical University of Munich, Munich, Germany, in 2014, 2015, and 2016, respectively, and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, MI, USA, in 2022.

He is an Assistant Professor of computer science and the Founding Director of the TigerSec Laboratory, School of Computing, Clemson University, Clemson, SC, USA. His current research focuses on autonomous vehicle security, adversarial machine learning, generative AI applications in security, and automotive data privacy.

Daniel Lüdtke received the Diploma (Dipl.-Ing.) degree in computer engineering from TU Berlin, Berlin, Germany, in 2003.

He joined the German Aerospace Center (DLR), Brunswick, Germany, in 2010, where he is the Department Head of the Flight Software, Institute of Software Technology. His department researches and develops reliable and resilient real-time software for aircraft and spacecraft.

Sebastian Steinhorst (Senior Member, IEEE) received the M.Sc. (Dipl.-Inf.) and Ph.D. (Dr.Phil.Nat.) degrees in computer science from Goethe University Frankfurt, Frankfurt, Germany, in 2005 and 2011, respectively.

He is an Associate Professor with the Technical University of Munich, Munich, Germany, where he leads the Embedded Systems and the Internet of Things Group, Department of Electrical and Computer Engineering. He was a Co-Program PI at the Electrification Suite and Test Laboratory of the Research Center, TUMCREATE, Singapore. His research interests include design methodology and hardware/software architecture co-design of secure distributed embedded systems for use in IoT, automotive, and smart energy applications.