IET Intelligent Transport Systems

WILEY

IET The Institution of Engineering and Technology

ORIGINAL RESEARCH OPEN ACCESS

EVECTOR: An Orchestrator for Analysing Attacks in Electric Vehicle Charging Systems

Devki Nandan Jha[1] | Tomasz Szydlo[1] | Nima Valizadeh[2] | Ringo Sham[1] | Aleksandra Edwards[2] | Amrit Kumar[1] | Amanjot Kaur[4] | Bo Wei[1] | Vijay Kumar[2] | Kai Li Lim[3] | Rajiv Ranjan[1] | Omer Rana[2]

[1]Newcastle University, Newcastle upon Tyne, UK | [2]Cardiff University, Cardiff, UK | [3]The University of Queensland, Brisbane, Australia | [4]Birla Institute of Technology & Science, Pilani, Dubai Campus, Dubai, UAE

Correspondence: Devki Nandan Jha (Dev.Jha@ncl.ac.uk)

Received: 18 April 2026 | **Revised:** 5 August 2026 | **Accepted:** 20 August 2026

ABSTRACT

Electric vehicle (EV) charging infrastructure is critical for the widespread adoption of EVs, ensuring efficient and secure charging processes. Evaluating the security and performance of EV charging systems in real-world infrastructure poses significant challenges due to the diversity of information exchange between vehicles and charging stations/electric vehicle supply equipment (EVSE), including complex network protocols, scale of deployment and a variety of potential threats. Existing simulation frameworks are unable to handle complex security scenarios across these differing data exchange protocols. In this paper, we propose a novel EV orchestration framework: EVECTOR, which addresses the limitations of existing simulation systems by enabling both quantitative and qualitative analyses of EV charging scenarios. EVECTOR (i) combines open-source EV charging simulators, including EVerest and ISO 15118 to support the analysis of a holistic EV charging ecosystem, (ii) provides a simulator-agnostic attack abstraction that lets one attack scenario be expressed once and executed against heterogeneous backends at different layers, (iii) offers a common telemetry schema that normalises logs from event-driven simulators for cross-layer analysis. Moreover, EVECTOR has a modular extension mechanism that lets new simulators or attacks be added without affecting the core orchestrator. We validate the EVECTOR framework through two case studies: (a) a cyber-physical attack, *broken wire* and (b) a cyber-specific attack, *frame fuzzification*. The case studies highlight EVECTOR's effectiveness in providing deeper insights into the security and performance of EV charging systems.

1 | Introduction

The global transition towards sustainable mobility is no longer a future projection but a current reality. EVs are now available from every vehicle manufacturer, offering a sustainable and environmentally friendly alternative to traditional fossil fuel-based vehicles. More than 20 million EVs were sold in 2025, an increase of 20% compared to 2024 [1]. As the number of EVs on the road increases, the infrastructure supporting them becomes crucial. The widespread adoption of EVs relies on reliable and secure charging infrastructure, ranging from home chargers to complex public and fast-charging networks designed to accommodate various vehicle types and charging standards [2, 3]. EV charging involves physical cable connections, integrated with communication protocols and software management systems [4], representing a complex, distributed cyber-physical system. The orchestration of such highly heterogeneous frameworks is further complicated when considering the integration of EV infrastructure within the wider, dynamic ecosystems of smart cities and intelligent transportation networks [5]. Ensuring the seamless integration of components is essential for the reliable functioning of the EV ecosystem.

This increasing reliance on charging infrastructure brings numerous risks and threats across the entire EV ecosystem [6–9]. Attackers can exploit the vulnerabilities in charging protocols

and software management systems to launch various types of attacks, including unauthorised access to EVSE, disruption of charging services, data breaches and the possibility of damaging a vehicle's battery or other critical physical systems. These challenges are further expanded by the dynamic charging behaviour of EV drivers, whose varying charging locations, schedules, and usage patterns increase the complexity of securing the charging infrastructure [10, 11].

The early detection of threats to EV charging infrastructure is critical for ensuring the safety and reliability of the EV ecosystem [12, 13]. Identifying suspicious activities, such as unauthorised access requests, abnormal communication patterns or system failures (malfunctions), allows administrators to act proactively to prevent potential attacks. However, early detection of these threats has significant challenges, which include the complexity and diversity of the EV charging system, vulnerability in physical and software components, integrating emerging systems with legacy systems, and balancing security methods with usability [14, 15]. Existing EV charging security research includes encryption and authentication mechanisms [16–18], use of distributed ledger technologies [19–21] and AI-based anomaly detection techniques [22–24].

To understand the likely impact of attacks on EV charging, we propose an orchestration environment able to combine multiple simulation systems. Each simulation system captures particular types of attacks, and there is no single simulator able to provide end-to-end simulation of components that make up an EV ecosystem [9]. Various simulators/frameworks are available for EV charging and management, each handling specific aspects of the ecosystem, for example, simulators such as EV2gym [25] and ACNSim [26] which focus on EV charging, scheduling and energy distribution. On the other hand, ISO implementation [27] and EVerest [28] support EV charging protocols like ISO 15118 and open charge point protocol (OCPP). Security-oriented frameworks such as EVShield [29] help analyse cyber threats and vulnerabilities in charging infrastructure. Existing simulators however lack the capability to comprehensively orchestrate EV charging scenarios, protocols, and attack vectors in an integrated manner, limiting their effectiveness in holistic security and operational analysis. A key observation underpinning EVECTOR is that no one simulator can capture the diversity of threats within an EV ecosystem. There is a need for an orchestration mechanism to integrate multiple context-specific simulators and the outcome of these integrations for subsequent analysis.

1.1 | Challenges

Implementing a robust simulation orchestrator for EV charging is challenging. To be specific, the main challenges are as follows:

1.1.1 | Complex EV Ecosystem

The EV charging ecosystem can include various types of EVSE (Level 1, Level 2 and DC fast chargers), different EV models and multiple communication protocols (e.g., ISO 15118), network protocols (e.g., OCPP, OCPI), physical connectors and charging standards (e.g., IEC 61851 [Type 2], SAE J1772 [Type 1], SAE J3400 [NACS], CCS, etc.) in different functional categories. Moreover, an EV can have differing battery capacity, charging rates and compatibility with charging standards. Similarly, an EVSE may support unique communication interfaces, power modes and software systems. Identifying the granularity at which this variation can be represented in a simulated environment requires an understanding of each component and its interactions and the potential expected outcome from the simulation.

1.1.2 | Variety of Threats

The threat landscape of the EV ecosystem is highly dynamic, with new vulnerabilities and attacks emerging regularly. Additionally, the protocols and software stack within the EV ecosystem are continuously being updated, which can introduce new types of attacks and expose previously unknown security gaps. Each attack method can exploit different aspects of the EV ecosystem. It is necessary to understand how these attacks operate and the components that they impact.

1.1.3 | Attack Explainability

Attack behaviours can involve protocol exploitation to physical tampering, requiring a simulator to trace the root cause and impact. Conversely, the high volume of data generated by simulation can hide critical attack indicators, requiring effective visualisation and logging mechanisms. Balancing analysis with real-time detection makes it difficult to fully support explainability, as security insights must be both interpretable for analysts and actionable for automated defence.

1.1.4 | Real-Time Simulation Needs

Some aspects of EV charging security, such as real-time attack detection and response, require simulation to operate in real-time, requiring significant computational resources.

1.1.5 | Data Availability

Developing realistic cybersecurity simulations requires representative data for EV charging behaviours, communication traces and attack scenarios. However, such data is rarely publicly available because operational charging infrastructures are largely proprietary, while cyberattack traces are sensitive and difficult to obtain.

Given these challenges, a comprehensive and flexible orchestrator is needed to seamlessly integrate EV charging scenarios, communication protocols, and potential attack vectors. Such an orchestrator should support both realistic EV charging behaviours and associated security assessments, enabling the execution of diverse user and attack scenarios. This paper presents EVECTOR, an Electric VEhicle Charging ATtack ORchestrator to address the challenges discussed above. EVECTOR is designed to provide a modular framework that supports multiple EV charging protocols, dynamic attack emulation, and scalable simulation.

Rather than relying on proprietary datasets, EVECTOR integrates open-source EV charging components to generate synthetic, protocol-compliant telemetry and attack traces under controlled and reproducible conditions. This enables researchers to systematically reproduce experiments and evaluate new attack scenarios without requiring access to confidential operational data.

1.2 | Contribution

- The design and implementation of EVECTOR is described, which combines existing open-source EV charging simulators, including EVerest and ISO 15118, to support simulation and analysis of a holistic EV charging ecosystem.
- EVECTOR introduces a simulator-agnostic attack abstraction and a common telemetry schema that allows the same attack scenario and analysis pipeline to be reused across heterogeneous, functionally different EV charging simulators (event-driven protocol simulators and time-stepped energy/grid simulators).
- We evaluate EVECTOR on both cyber-physical and cyber-only attack scenarios: broken wire and data frame fuzzification attacks. The outcome can be used to assess vulnerabilities and the impact on the EV charging ecosystem.

1.3 | Outline

The remainder of the paper is as follows. The background, along with recent related work, is presented in Section 2. The system architecture of EVECTOR is given in Section 3, specifying the components in the architecture and their interactions. The evaluation of EVECTOR on multiple scenarios is presented in Section 4. We provide a discussion highlighting the limitations of our work in Section 5, with conclusions in Section 6.

2 | Background and Related Work

2.1 | EV Charging Ecosystem

The EV charging process involves a sequence of well-defined steps to ensure safe, efficient, and reliable energy transfer between the EV and the EVSE. Figure 1 shows the various steps in the EV charging process and their connections. An explanation of each step, including the protocols involved, is given below.

1. *Unmated state*: In the unmated state, an EV and EVSE are not physically connected. During this phase, the vehicle is parked near the EVSE, and the driver prepares for the charging session. No communication occurs at this stage. However, some charging systems with data exchange enabled may broadcast availability and identification information. In this case, EVSE may use OCPP to communicate status updates, such as heartbeat signals, to the backend management system.

2. *Mated state*: The mated state begins when the charging cable is physically attached to the EV. A mechanical and electrical connection is established between the EV and EVSE. At this stage, the connection is verified to ensure safety. ISO 15118 comes into play to initiate communication between the EV and EVSE. The

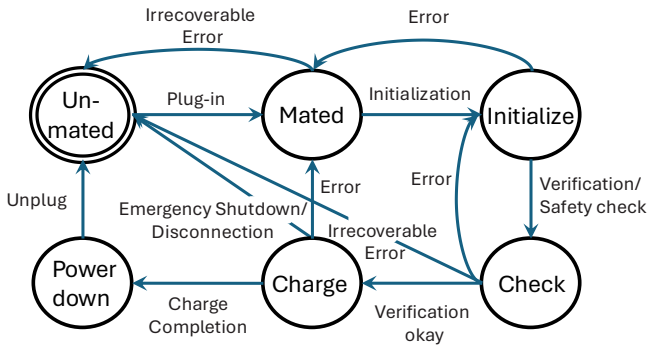


FIGURE 1 | EV charging states.

ISO 15118 protocol facilitates the handshake process to confirm compatibility, charging mode and initial configuration settings.

However, faults such as improper cable attachment or tampered connectors can occur here. A faulty vehicle, an EVSE or a physical tampering attack that damages the charging plug or port might prevent the system from transitioning to the initialisation state. To avoid this, the system may remain in the mated state, or in cases of persistent or irrecoverable issues, revert to the unmated state to prevent further damage.

3. *Initialize state*: Once the physical connection is confirmed, the initialisation phase begins. The EV and the EVSE exchange critical information, such as the vehicle's state of charge (SoC), battery capacity, and required energy levels. This ensures that the charging session is customised for the specific needs of the vehicle. For advanced vehicle-to-grid (V2G) capabilities, which require bidirectional energy transfer and dynamic grid communication, ISO 15118-20 is employed to manage the necessary protocols and secure data exchange. Additionally, OCPP facilitates communication between the EVSE and the backend system, allowing for user authentication, payment verification, and session authorisation. An attack can also happen at this stage, for example, communication spoofing, man-in-the-middle (MITM) attacks can disrupt this phase [30–32]. If a MITM attack occurs, falsifying authentication messages, the system may fail to authenticate and revert to the mated state.

4. *Check state*: The check state ensures the readiness of all components involved in the charging process. Safety checks are conducted to confirm proper grounding, safe voltage levels and compatibility. The ISO 15118 protocol supports safety parameters, such as ensuring earthing and voltage checks are valid. In this stage, faults such as grounding issues, sensor malfunctions or false safety signal attacks for example, injecting malicious data to mimic faults can cause the system to abort the session. In such cases, the system might transition back to the initialisation state to retry safety checks or terminate the session entirely by returning to the unmated state. The ISO 15118 provides support for fault diagnostics, while the OCPP protocol can be used to communicate any detected issues to the backend system for logging and resolution.

5. *Charge state*: The charge state involves the actual transfer of energy from the EVSE to the EV battery. During this phase, the power flow, voltage and temperature are continuously monitored,

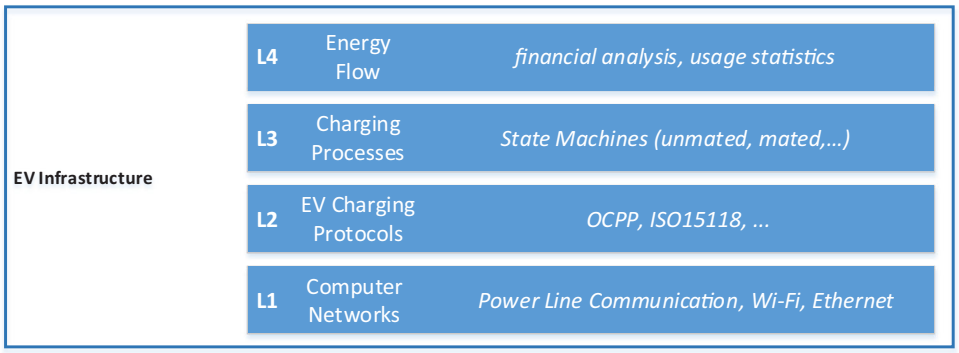


FIGURE 2 | Layered architecture of EV charging infrastructure.

and adjustments are made based on the EV requirements and the EVSE capabilities. The real-time power delivery, checking the charging status and receipt of metering data is maintained by the ISO 15118 protocol. The OCPP protocol keeps the backend system updated on the progress of charging sessions, energy consumption and time elapsed.

Faults such as sudden power surges, over-current conditions or broken cable attacks (physical disconnection during charging) can force the system to halt charging [30, 33]. A broken cable attack or accidental physical disconnection during charging can force the system to halt energy transfer abruptly. The charging system would then detect the loss of connection and transition directly to the unmated state, skipping the standard power-down sequence. Similarly, a severe fault, such as a fire hazard or power surge, might trigger an emergency shutdown protocol that cuts off power and disconnects the session, transitioning directly to the unmated state and protecting the equipment. The system may also move to the mated state in case a charging error, such as voltage mismatch or overheating is detected. In this case, the system will stop energy transfer while maintaining the physical connection.

6. *Power down*: As the charging session nears completion, the power-down phase ensures a safe and gradual reduction in energy transfer. The ISO 15118 protocol coordinates the controlled cessation of energy flow, ensuring all components are safely deactivated. OCPP communicates session termination to the backend system and handles payment processing or receipt generation. This step is critical to prevent abrupt power cuts, which can damage an EV battery or EVSE.

The final step returns the system to the unmated state. The EV is disconnected from the EVSE and the session ends. The EVSE becomes available for the next user.

2.2 | Architecture for EV Charging

The EV charging infrastructure can be divided into four layers depending on the network protocol being used, as shown in Figure 2. Interactions between the different layers are given below.

A. *Computer network layer* (L1) - This layer handles data exchange between EVs, EVSEs, backend servers and grid operators. It ensures secure connectivity using wired (Ethernet) and wireless (Wi-Fi, LTE, 5G) communication.

B. *Protocol layer* (L2) - This layer governs how EVs, EVSEs, and backend systems communicate, handling authentication, payment processing, and charging negotiations using standardised protocols such as ISO 15118 and OCPP. At this level, interoperability support is required to overcome communication problems between devices, especially those from different manufacturers.

C. *Charging layer* (L3) - This layer deals with the physical energy transfer process between the EV and the EVSE, that is, start, duration and energy sent to the EV battery, ensuring power regulation, voltage/current adjustments and safety mechanisms during charging are adhered.

D. *Energy management layer* (L4) - This layer connects EVSEs to the power grid, managing energy demand, balancing loads and ensuring grid stability. Protocols at this layer include IEC 61850 for EV charging for smart grid integration, V2G interactions and dynamic power allocation.

2.3 | Related Work

There are multiple simulation frameworks available to support an EV charging environment. To gain a comprehensive understanding of EV charging, existing simulators and emulators can be broadly classified into three key categories as explained below.

2.3.1 | EV Charging Protocol Simulators/Emulators

This category of simulators/emulators focuses on replicating communication standards to ensure seamless interactions between EVs and EVSE while maintaining compliance with industry protocols. These simulators assist in testing protocol adherence, validating secure charging transactions and ensuring interoperability.

Switch-EV [34] offers tools and platforms designed to implement and test V2G communication standards, with a strong focus on ISO 15118. One of its offerings, RISE V2G [35] and Josev Community [36], serves as a reference implementation for ISO 15118, providing features like plug and charge (PnC) for authentication and billing. OpenV2G [37] is another

TABLE 1 | Comparison of related works.

Framework	Protocol support (ISO15118, OCPP)	Charging behaviour and scheduling	Attack and fault injection	EVCharging configurability	User behaviour configurability	Data logging
RISE V2G [35]	ISO15118	✗	✗	✗	✗	✗
Josev [36]	✓	✗	✗	✗	✗	✗
OpenV2G [37]	ISO15118	✗	✗	✗	✗	✗
EVerest [28]	ISO15118, OCPP	✓	Partial	✓	✗	✓
Mobility House [38]	OCPP	✗	✗	✗	✗	✗
V2G-Sim [39]	✗	✓	✗	✓	✓	✗
EVLibSim [40]/ EV-EcoSim [41]	✗	✓	✗	✓	✓	✗
OPEN [42]	✗	✓	✗	✓	✓	✗
ACN-Sim [26]	✗	✓	✗	✓	✓	✓
Chargym [43]	✗	✓	✗	✓	✓	✗
EV2Gym [25]	✗	✓	✗	✓	✓	✗
CEE [44]	OCPP	✗	✓	✓	✗	✗
EVShield [29]	Partial	✗	✓	✓	✗	✓
EVECTOR (proposed)	ISO15118, OCPP	✓	✓	✓	✓	✓

open-source implementation of the ISO 15118 communication protocol. It supports custom testing scenarios and extensions for research environments. Another open-source EV charging software stack framework is EVerest [28], which currently supports ISO 15118 and OCPP. It is available as Docker containers and provided with a Node-Red user interface. Mobilityhouse OCPP Simulator [38] is a Python implementation of OCPP with support for versions 1.6 and 2.0 and uses the JSON version of the protocol. It also provides examples of how to implement an EVSE and client. Despite their advantages, these simulators/emulators mainly focus on protocol behaviours and interoperability testing without incorporating dynamic attack simulations, security analysis, or integration with realistic EV charging scheduling and energy management scenarios.

2.3.2 | EV Request-Response Simulators

This category includes simulators for EV transactions, charge scheduling, and power management, with the main focus on optimising energy distribution, forecasting charging demands and improving grid efficiency.

V2G-Sim [39] offers rich EV modelling, including EV, EVSE, but is not open-source. EVLibSim [40] and EV-EcoSim [41] support various charging scenarios but are constrained by realistic EV behaviour data. OPEN [42] provides smart energy system simulations but features uniform EV models, making it unsuitable for evaluating attacks. ACN-Sim [26] is an open-source simulator which utilises actual charging session data to model the behaviour of charging systems, including battery behaviour and charge scheduling. Chargym [43] enables cost optimisation with EV simulation but oversimplifies EV behaviour and lacks any real-world data. EV2Gym [25] is another simulator that models EV behaviour based on real data from EVs, EVSEs and their interaction. EV2Gym is designed to assess the behaviour of smart charging algorithms within a standardised platform on a defined scale. While these simulators are effective in load balancing and power management, they lack integration with security mechanisms and do not account for protocol-level details or charging infrastructure vulnerabilities. Their primary focus remains on energy optimisation.

2.3.3 | EV Attack Handling Simulators

Several studies have explored security vulnerabilities/threats in EV charging infrastructure, particularly focusing on the ISO 15118 and OCPP protocols. The cyber-energy emulation (CEE) platform [44] simulates cyberattacks on EV chargers and their impact on the power grid, yet it lacks detailed insights into the specific attack methodologies and does not provide a structured approach for identifying vulnerabilities. Similarly, other work [45] analyses security flaws in OCPP 1.6 but does not explore the adaptability of the simulator to newer protocol versions or diverse system configurations. Sarieddine et al. [46] employ an OWASP-based testbed to identify critical vulnerabilities like man-in-the-middle attacks, and [47] presents a real-time co-simulation testbed that integrates OCPP to analyse threats and power grid impact. EVShield [29] also helps analyse cyber threats and vulnerabilities in the charging infrastructure. Although these simulators provide valuable insights into security threats, they do not fully integrate with EV charging protocol validation, making it difficult to conduct end-to-end testing of EV charging environments. Table 1 shows a comparison of related work with the proposed work.

The frameworks reviewed above each address one of EVECTOR's three concerns in isolation: protocol-level fidelity (e.g., EVerest, OpenV2G), charging/scheduling realism (e.g., ACN-Sim, EV2Gym) or attack injection (e.g., CEE, EVShield). None of

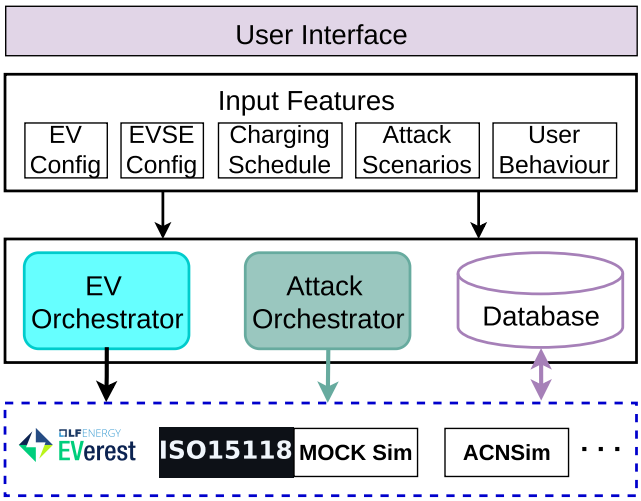


FIGURE 3 | Proposed EVECTOR orchestrator framework.

them combines all three within a single, extensible orchestration layer. EVECTOR technical contribution is the orchestration abstraction itself: a common *attack interface* (Section 3.1.4) decoupled from any one simulator, able to target either the protocol/communication layer (L1–L2) or the charging/energy layer (L3–L4); and a common telemetry schema (Section 3.1.5) that lets logs from functionally different simulators be compared within one analysis pipeline. This is distinct from wrapping a single simulator with a new transport (e.g., MQTT), since a new simulator or attack type can be added by extending a base class rather than re-architecting the system.

3 | System Architecture

This section presents the architecture design and implementation of EVECTOR as illustrated in Figure 3.

3.1 | EVECTOR Architecture

EVECTOR is a modular and adaptable orchestration framework designed to replicate and analyse various aspects of the EV charging ecosystem. It consists of multiple layers, each responsible for a specific function, ensuring flexibility, scalability and extensibility. This enables users to define EV charging scenarios, simulate different behaviours, introduce attack scenarios, and analyse the generated data for security insights. The following sections describe each layer in detail.

3.1.1 | User Interface

This layer provides an interactive interface for users to configure and control the simulation environment, and an intuitive means for users to explore different EV charging scenarios. It allows users to input various parameters, including *EV config* and *EVSE config*. It also enables users to configure specific attack scenarios, such as *broken wire attack*, making it possible to study the impact of various cyber threats/attacks on the EV charging infrastructure.

3.1.2 | Input Layer

This layer serves as the primary configuration point, allowing users to define various aspects of the simulation. The main input features are given below.

1. *EV configuration*: Users can specify the number of EVs participating in the simulation, their battery capacities, charging requirements, SoC and other operational parameters.
2. *EVSE configuration*: The system supports multiple EVSEs (charging stations), enabling users to define different station types, power ratings, charging protocols and availability.
3. *Charging schedule*: Users can input dynamic charging schedules that state when and how EVs should charge, incorporating real-world constraints such as peak and off-peak hours.
4. *Attack scenarios*: This feature enables users to simulate cybersecurity threats by selecting specific attack vectors (e.g., broken wire attack).
5. *User behaviour*: The orchestrator also allows the simulation of different user behaviours, such as preference for certain EVSEs, frequency of charging and interactions with the charging infrastructure.

3.1.3 | EV Orchestrator

This is the core component of EVECTOR, which acts as the engine for simulating the behaviour of EVs and the broader EV charging ecosystem. This component is responsible for replicating real-world charging interactions based on the configurations provided in the input layer. It integrates *EV behaviour simulation*, *EVSE operations*, *charging schedule* and *scenario* execution. It directly interacts with the *simulator suite* to simulate various EVSE operations, including communication with EVs through protocols like ISO 15118 and OCPP. Based on predefined user scenarios, the orchestrator enables running specific test scenarios. Section 3.2.1 describes different classes and their interaction. The modular nature of the EV Orchestrator allows it to be extended easily, supporting new EV and EVSE types, charging models, additional protocols and emerging technologies in the EV ecosystem.

3.1.4 | Attack Orchestrator

The *attack orchestrator* is another crucial component designed to inject security threats such as broken wire attacks into the simulated EV environment. This component enables researchers and analysts to evaluate the resilience of EV charging infrastructures against cyber and physical attacks. The *attack orchestrator* is a modular component. This paper details the design and implementation of *broken cable attack* and *fuzzification attack*. By integrating real-time attack simulation, the *attack orchestrator* allows the system to monitor real-time security risks and effectively validate mitigation strategies. New attack types can be easily added by extending the abstract attack class shown in Figure 5, ensuring the system remains relevant as new threats emerge.

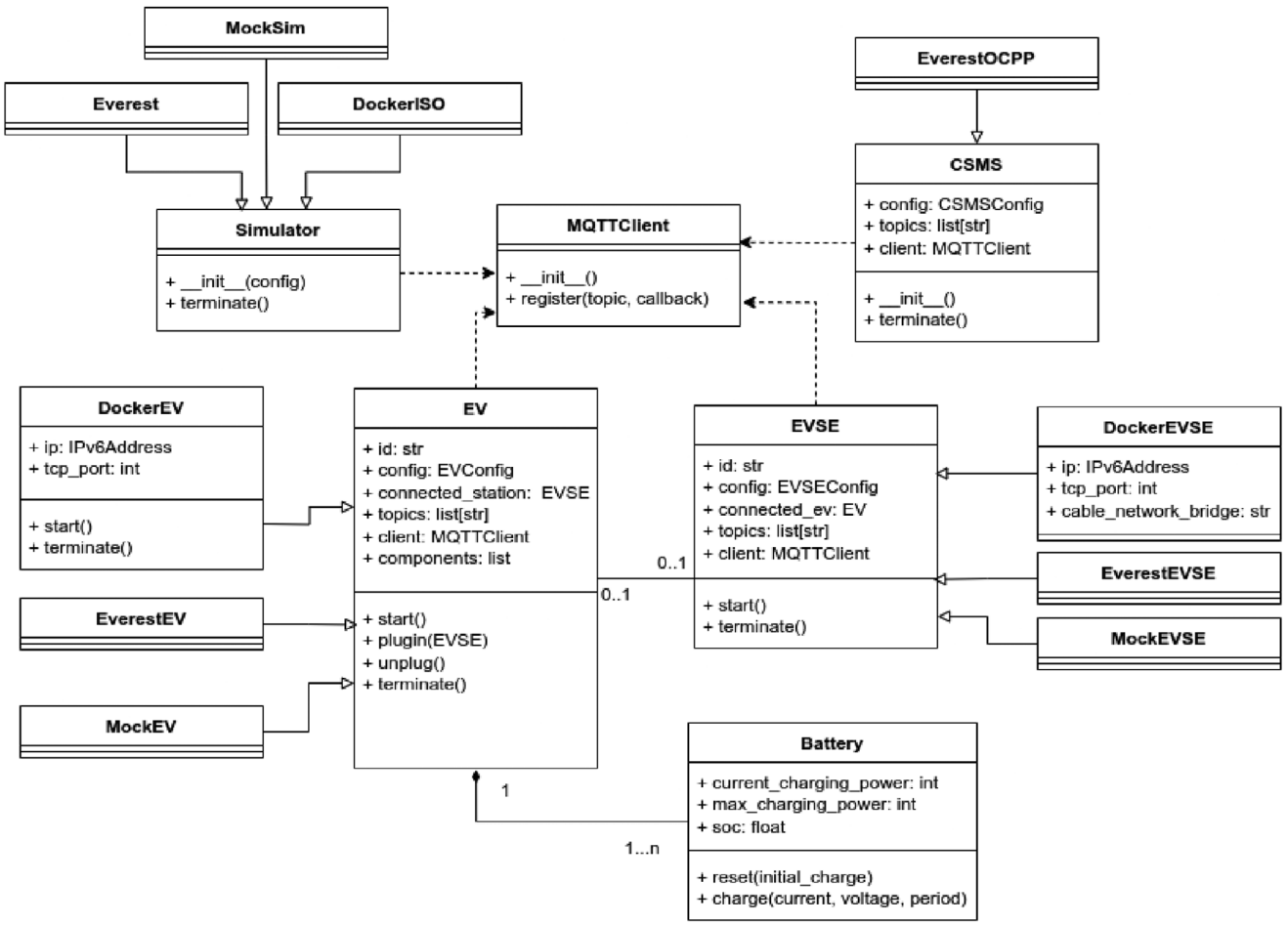


FIGURE 4 | EVECTOR EV orchestrator class diagram.

3.1.5 | Database Module

The *database module* is responsible for storing all the logs and data generated during the simulation. The system uses MongoDB,[1] a NoSQL database, which provides efficient storage and retrieval of structured and unstructured data. Key functionalities of this layer include storing EV charging logs and system performance metrics. The NoSQL-based structure enables high-speed queries and flexible data storage, making it ideal for handling large-scale simulations while maintaining data integrity.

3.1.6 | Simulator Suite

The base layer of the orchestrator is the *simulator suite*, which integrates multiple EV charging simulators and emulators to provide a robust testing environment. The current version supports: *(I) EVerest* [28]: A powerful open-source software stack that supports EV charging infrastructure testing. (*(II) ISO 15118 simulator*[27]: A dedicated simulator to test charging communication using the ISO 15118 protocol, ensuring compatibility with real-world implementations. *(III) Mock simulator*: A lightweight, configurable simulator designed for testing scenarios that do not require full-scale simulation complexity. *(IV) ACNSim* [26]: A simulator for modelling the behaviour of EV and EVSE charging. The *simulator suite* is modular in design, allowing new simulators to be easily added. This adaptability ensures that the system remains future-proof and can accommodate evolving EV charging technologies.

3.2 | EVECTOR Design and Implementation

This section discusses the design and implementation of EVECTOR, including the class diagram of *EV orchestrator*, the charging sequence diagram of *EV orchestrator* and the integration of *EV orchestrator* with *attack orchestrator* and *database*.

3.2.1 | EV Orchestrator

Figure 4 represents the core architecture of the *EV orchestrator*—structured around three primary classes: EV, EVSE and simulator. These classes encapsulate the functionalities of an EV, an EVSE and the simulation environment, respectively. MQTT facilitates communication between these components, ensuring real-time data exchange.

The EV class models the behaviour of an EV within the simulation. It includes two primary functions that interact with the EVSE class: *plug in* and *plug out*. Additionally, each EV instance

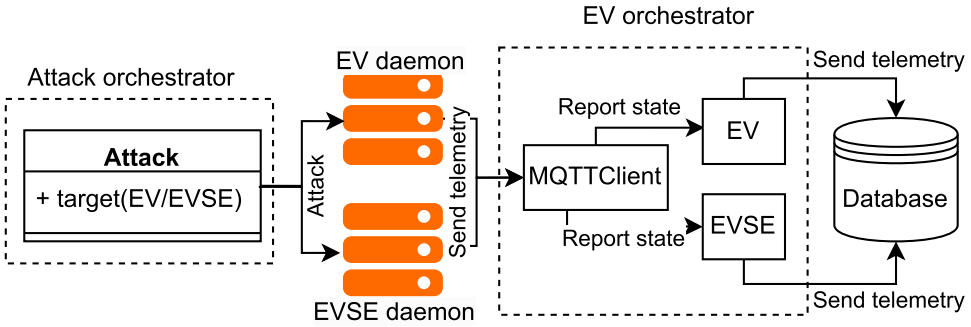


FIGURE 5 | Interaction between EVECTOR components.

contains at least one instance of the battery class, which simulates the behaviour of an EV battery during the charging process. The EVSE class is responsible for managing the charging session, handling power delivery and modifying runtime variables such as current and voltage levels based on the simulation conditions.

The simulator class serves as the orchestrator, responsible for configuring, deploying, and managing instances of different simulators. It abstracts the complexities of different simulation environments, providing a unified interface for handling various EV charging protocols and behaviours.

Each of these base classes, EV, EVSE and simulator, is further extended into subclasses to support different simulator implementations. In our current implementation, we have integrated three distinct simulators: *MockSim*, *Docker ISO* and *EVerest*. Correspondingly, each of the three main classes has dedicated subclasses for these simulators. EV class extends to MockEV, DockerEV and EverestEV, each representing an EV model within its respective simulation environment. EVSE class extends to MockEVSE, DockerEVSE and EverestEVSE, each defining the behaviour of an EVSE according to the underlying simulator. Finally, the simulator class is extended to MockSim, DockerISO and EVerest, handling the deployment and execution of their respective simulation environments.

This modular and extensible design ensures that new simulators can be easily integrated into the framework by simply extending the base classes. The use of MQTT for inter-component communication enables seamless interaction between different simulator instances, making the system adaptable for future enhancements and real-world testing scenarios.

3.2.2 | Interaction of Attack Orchestrator and Database With EV Orchestrator

The integration between the EV orchestrator, attack orchestrator and database involves multiple stages of communication as shown in Figure 5. The EV daemon and the EVSE daemon represent the simulation suite, executing the charging process. The attack orchestrator performs the attack directly on the EV and EVSE daemons. As the attack continues, the MQTT client inside the EV orchestrator collects the telemetry from the daemons and invokes the instances of EV and EVSE. These instances will update their internal state representation based on the telemetry of the daemons, as well as record the telemetry data to the database. The database serves as historical storage for charging data metrics, allowing for post-simulation analysis. Listing 1 shows a sample telemetry record generated by the EVECTOR orchestrator. Each record contains a timestamp, the originating module, the communication protocol, and the message direction, together with the decoded ISO 15118 payload. The hierarchical structure preserves protocol semantics while providing a simulator-independent representation that can be uniformly processed by EVECTOR's other components.

4 | Evaluation

This section evaluates EVECTOR for two separate types of attacks: (A) broken wire attack and (B) frame fuzzification attack. These two attacks were selected because they impact the orchestrator at different layers of EV charging through the same attack interface, demonstrating cross-layer reuse of the orchestration mechanism rather than generality across many attack types in absolute terms. The detailed implementation and analysis are given below.

4.1 | Case 1: Broken Wire attack

A broken wire attack in an EV charging system occurs when an attacker deliberately manipulates the communication or power transfer wires between the EV and EVSE [33]. This can disrupt charging sessions, cause failures in power delivery, or even trick the system into falsely detecting a disconnected vehicle. If the control signal is compromised, the EVSE may be unable to regulate power flow properly, leading to potential safety hazards or denial-of-service (DoS) attacks.

Table 2 categorises broken-wire attack scenarios on EV charging infrastructure based on their scope, impact and attack methods. Individual-targeted attacks primarily exploit the physical network layer (L1) to disable a single charger. This prevents an EV from charging, causing inconvenience to the driver, but with limited overall impact. Group-level attacks escalate the disruption by targeting the charging process layer (L3), potentially affecting multiple chargers in a given area. By interrupting communication between the chargers and the energy management system, these attacks can make EVSEs in a region unavailable, forcing drivers to seek alternative locations and creating bottlenecks. Large-scale attacks pose the most severe risk, potentially disrupting all chargers across an entire city. By manipulating vulnerabil-

```
"timestamp": "2026-07-24 14:05:27,377",
"module": "iso15118.shared.exi_codec",
"protocol": "iso:15118:2:2013:MsgDef",
"direction": "receive",
"mac": null,
"message": {
  "V2G_Message": {
    "Header": {
      "SessionID": "B61E40277D65DB08"
    },
    "Body": {
      "SessionSetupRes": {
        "ResponseCode": "OK_NewSessionEstablished",
        "EVSEID": "UK123E1234",
        "EVSETimeStamp": 1784901927
      }
    }
  }
}
```


LISTING 1 | Sample EVECTOR telemetry record generated by the orchestrator.

TABLE 2 | Disruptions caused by the broken wire attack.

Scope of the attack	Chargers availability	Outcome of the attack	Attack mimicking technique	Layer	Result	Effect
Single driver	Limited number of chargers unavailable	Frustration of the driver impacting daily schedule	Physical network layer attack between EV and the EVSE	L1, L2	ISO15118 comms. disruption; communication error raised	Charging process stopped; charger state switched to mated
Group of drivers	Chargers unavailable in a region	EVSE bottlenecks—as users try different EVSEs	Error condition in the EVSEs	L3, L4	Charging process stopped; reduction in the power consumed for charging the cars	Unavailability of EVSEs in the region

ities at multiple layers, these attacks could disable widespread charging infrastructure, leading to significant traffic congestion and power distribution challenges. While the specific methods for large-scale attacks are not always detailed, the table highlights how different layers of the EV charging system can be exploited.

The attack orchestrator generates the broken wire attack in two scenarios: (A) L1–L2 level: The Pumba chaos engineering tool [48] is used to emulate network impairments by injecting controlled packet loss into the communication channel to interrupt the EV-EVSE communication that disrupts the ISO 15118 protocol execution. This provides a reproducible and configurable representation of communication disruptions at the network and protocol levels, enabling the evaluation of charging-session failures and recovery behaviour under cyberattacks. Since EVECTOR targets software, protocol, and system-level security evaluation, this abstraction offers a practical and scalable approach while remaining compatible with the framework's containerised architecture. (B) L3–L4 level: The charging process is disrupted in the EV orchestrator to raise the error flag.

4.1.1 | L1–L2 Level attack

This scenario is modelled by containing a pair of EV and EVSE daemons connected to a separate network bridge that acts as the charging cable between the two. The EV daemon will establish a connection with the EVSE to initiate charging. Pumba internally uses the Linux tool *tc* for network traffic control to manipulate any network interfaces of a machine. For the L1–L2 attack, Pumba emulates packet loss on the connection bridge that the EV and EVSE daemon are using, which effectively simulates the act of severing the physical communication wire. After the attack is initiated, all communications between the two actors are lost, and the charging session should abort as the ISO 15118 protocol requires heartbeat messages to be exchanged during the charging loop. Figure 6 shows the packet count during the

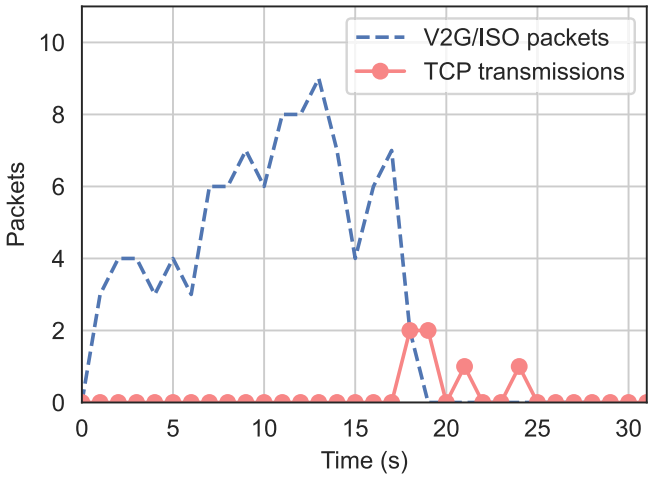


FIGURE 6 | Packet capture during a broken wire attack.

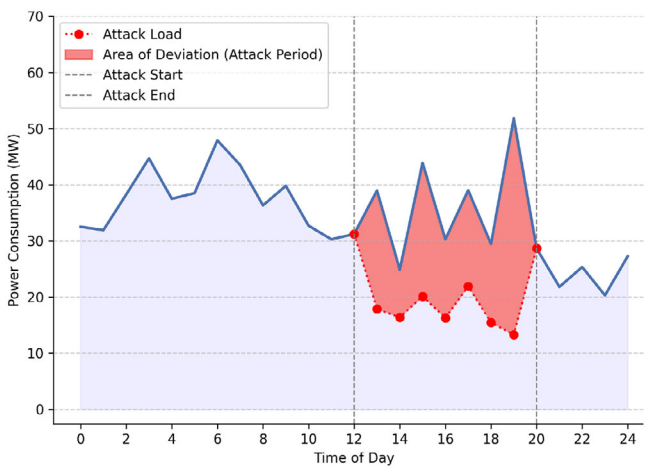


FIGURE 7 | Power consumption pattern at EVSE: Impact of broken wire attack showing deviation from normal load (MW) over 24-h period.

evaluation. During the attack, starting from the 18th second, TCP error packets have been observed (*TCP transmissions*), which is indicated by the red lines. The amount of time to terminate a charging session depends on which state the charging session is currently in. In this case, the EVSE was in charge state and needs to respond with a `ChargingStatusRes` on ISO 15118 message to the EV during the charging loop. The timeout window for that state is 2 s, after which the session terminates without any message exchanges.

4.1.2 | L3–L4 Level Attack

The effects of a broken wire attack on an EVSE's power consumption over a 24-h period are illustrated in Figure 7. Here, the blue line represents the expected (normal) power consumption pattern, while the red dotted line indicates the compromised load factor during the attack period, which occurs between the two vertical dashed lines (approximately from hour 12 to hour 20). During this attack window, a significant deviation in power consumption is observed, as highlighted by the red shaded area. The actual power delivered drops well below the expected levels, showing a fluctuation in the range of 15–20 MW, whereas under normal conditions, the power demand in this timeframe was expected to be around 30–40 MW. This results in an estimated 40%–50% reduction in power delivery, significantly affecting EVSE performance.

This drop in delivered power is a downstream consequence of the protocol-level disruption characterised in Section 4.1.1. When Pumba severs the L1–L2 communication/protocol path, the EVSE stops receiving the periodic `ChargingStatusRes`/heartbeat exchange required during the charging loop. Once the 2-s ISO 15118 timeout elapses without a valid exchange, the session terminates by protocol timeout rather than by an explicit disconnect command from the *attack orchestrator*. Figure 7 measurements aggregate many such timeout-driven terminations across the EVSE's sessions over the attack window (hour 12–20). The 40%–50% reduction corresponds to the fraction of sessions interrupted by protocol timeout during the window. Such a pattern is characteristic of a broken wire attack, where a deliberate disruption in the charging infrastructure leads to a sharp decline in power utilisation. This reduction in available power can cause severe charging delays, bottlenecks at stations, and user inconvenience, particularly in high-demand scenarios. In a larger context, such attacks could also strain the energy grid by causing unexpected demand shifts, leading to load imbalances and operational inefficiencies.

4.2 | Case 2: Frame Fuzzification Attack

This section implements and evaluates a fuzzification attack targeting the OCPP protocol implementation within the EV charging ecosystem. Given the increasing reliance on OCPP for secure and reliable communication between devEVSE and central management systems (CSMS), it is crucial to assess its resilience against protocol-based fuzzification. Our approach systematically evaluates a generic OCPP 2.0.1 implementation using the EVECTOR, with a focus on identifying security weaknesses through Protocol-based fuzzification at the L2 level. This attack extends OCPPStorm [49], with the *attack orchestrator* carrying out the fuzzification attack and evaluate the implementation.

The *attack orchestrator* initiates the process by systematically sending OCPP requests (with message code 2) while recording the backend responses. The key components of the attack are: *(A) Fuzzer*, which operates independently of the orchestrator and injects malformed or unexpected OCPP messages. *(B) Orchestrator attack handler*, which manages attack execution and interacts with the language-based fuzzer. *(C) Message validator* to ensure the correctness of messages before they are processed by the backend. For this attack, we assume that the fuzzer operates as a malicious EVSE, either bypassing authentication layers or functioning in a controlled test environment where security constraints are deliberately removed. For the implementation, we utilize the ISLa fuzzer [50], which applies grammar-based fuzzification techniques to explore vulnerabilities in OCPP message exchanges. ISLa's capabilities in generating context-sensitive inputs align with the complexities of OCPP messages. Our attack strategy specifically targets the robustness of the OCPP backend (Citrine-OS) within the EVerest simulator stack. To ensure the

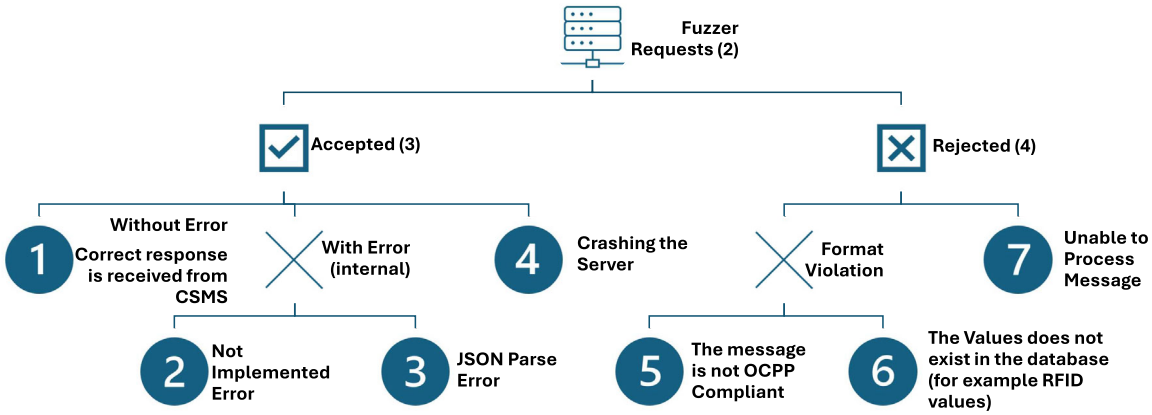


FIGURE 8 | Possible outcome scenarios for fuzzification.

validity of the generated messages, we integrate the typed-ocpp library,[2] which verifies compliance with the OCPP standard.

4.2.1 | Fuzzification Strategy

The attack consists of two primary fuzzification strategies, as discussed below.

Randomised Message Fuzzification (Language-Based). This strategy involves sending a randomised sequence of OCPP messages without following a structured transaction flow. Each of the 10 most commonly used OCPP messages is sent 100 times in an unordered manner. The objective is to test how the backend handles unexpected message sequences and excessive load against the normal protocol behaviour.

This approach is particularly useful for: (a) stress-testing the backend by sending repeated requests within a short timeframe, (b) identifying inconsistencies in response handling, where the backend might incorrectly process messages outside the expected workflow and (c) evaluating resilience against high-frequency malformed requests and understanding the system's rate-limiting mechanisms. For instance, an *Authorise request* message may fail if the backend does not recognise the RFID key due to the absence of prior *BootNotification* or *Registration* steps. Although somewhat forced, this method allows us to analyse the system's resilience to unexpected message patterns and its ability to handle stress under randomised loads.

State-Based Language Fuzzification. This approach follows structured communication flows to assess how the backend processes logically ordered transactions. Messages are categorised into four functional groups: *(A) Start-up phase messages*, for example, `BootNotification`, `FirmwareStatusNoti- fication`, and `ClearCacheRequest`, *(B) operational phase messages*, for example, `Heartbeat`, `StatusNotificationRequest`, *(C) user interaction messages*, for example, `AuthorizeRequest`, `Get15118 EV-CertificateRequest`, `NotifyCustomerInformation` and *(D) firmware and custom data handling*, for example, `PublishFirmwareSt- atusNotificationRequest`, `DataTransferRequest`. A real behaviour of EV charging is represented by executing these messages in the correct sequence. Each sequence of messages is executed 100 times to assess system consistency. The fuzzer begins with a valid initial sequence but deliberately injects errors or malformed messages at specific transition points. The aim is to analyse how the backend processes unexpected sequences and whether it can recover gracefully from erroneous transactions. This method helps uncover protocol weaknesses related to state transitions and error recovery mechanisms, data validation failures, and potential bypass scenarios.

4.2.2 | Fuzzification Attack Results

Based on our observations, the messages can either be *accepted* or *rejected*, as shown in Figure 8. If CSMS accepts the message without any error, the return code is `(3,1)` ①. The message can also be accepted with an error generating the return code as `(3,2)` and `(3,3)` for not implemented error ② and JSON parse error ③, respectively. The request may get accepted, crashing the server with a return code `(3,4)` ④. Moreover, some messages were also rejected. The message can be rejected due to non-compliant format violation ⑤ or non-existent value ⑥. Finally, if the CSMS is not able to process some messages, they are rejected with an error code of `(4,7)` ⑦.

The results of the fuzzification attacks are summarised in Table 3. This table compares randomised message fuzzification with state-based fuzzification. The observed responses provide insights into how the backend processes messages under various conditions.

In randomised message fuzzification, where each message was sent independently 100 times, certain messages, such as *Heartbeat* and *FirmwareStatusNotification*, were consistently accepted. This suggests the independence of such messages on prior states or primarily function as boot messages. In contrast, *AuthorizeReq* had a 78% acceptance rate, with 19% intermediate responses and 3% failures. This variability is likely due to the absence of a valid RFID token in some instances. Since *BootNotification*, such as registering an EVSE, must precede other messages in a typical OCPP transaction, it consistently caused the server to stop, likely for security reasons. Similarly, *ClearCacheReq* was mostly rejected, reinforcing that the backend enforces security mecha-

TABLE 3 | Fuzzification attack analysis in EVECTOR.

Message	Accepted (code 3, 1)	Accepted (code 3, 2)	Accepted (code 3, 3)	Stopping the server (code 3, 4)	Rejected (code 4, 5)	Rejected (code 4, 6)	Rejected (code 4, 7)	Average latency
Heartbeat	100%	0	0	0	0	0	0	10 ms
AuthorizeReq	78%	19%	3%	0	0	0	0	11 ms
BootNotification	0	0	0	100%	0	0	0	48 ms
ClearCacheReq	0	0	0	0	92%	0	8%	3 ms
FirmwareStatus- Notification	100%	0	0	0	0	0	0	26 ms
DataTransfer-Req	0	100%	0	0	0	0	0	8 ms
Get15118EV-CertificateReq	0	0	100%	0	0	0	0	573 ms
NotifyCustomer-Information	100%	0	100%	0	0	0	0	8 ms
StatusNoti-ficationReq	100%	0	100%	0	0	0	0	16 ms
PublishFirm-wareStatusNot-ificationReq	0	34%	0	0	66%	0	0	10 ms

nisms against unauthorised cache manipulations. *DataTransferReq* was fully accepted in all cases, indicating potentially weaker validation in the backend. Meanwhile, *Get15118EV-CertificateReq* exhibited high latency (573 ms), likely due to the cryptographic overhead of certificate verification or the search for certificates in the backend or databases.

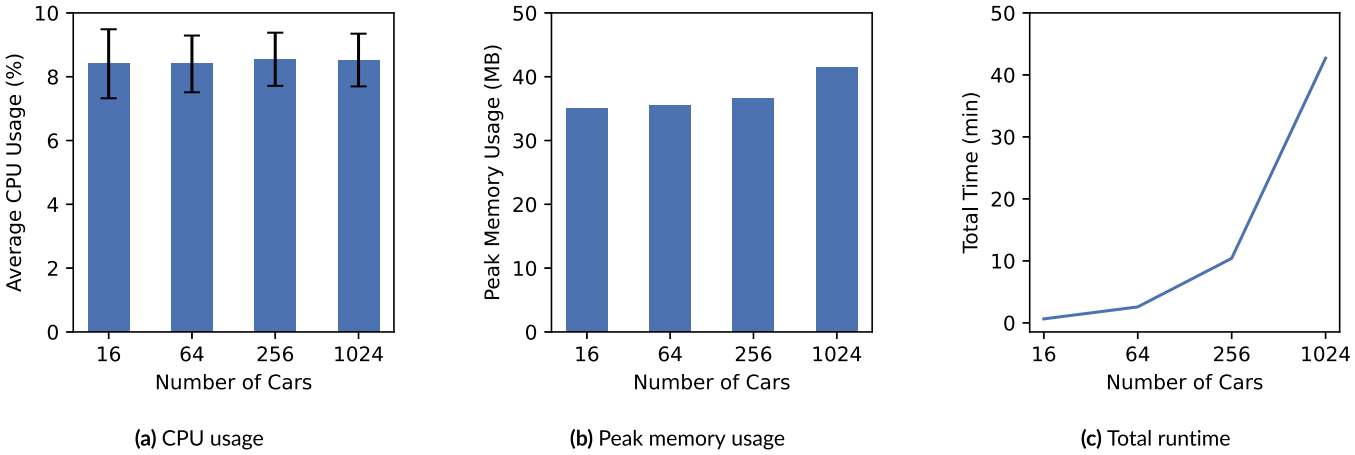
In state-based fuzzification, messages were sent in structured sequences, leading to improved system response accuracy. *AuthorizeReq* exhibited a higher success rate, confirming that adherence to the correct transaction flow enhances acceptance. *NotifyCustomerInformation* and *StatusNotificationReq*, which had fluctuating responses in randomised tests, demonstrated greater consistency when executed in proper sequences, suggesting their reliance on prior information. Similarly, *PublishFirmwareStatusNotificationReq*, which produced mixed results in randomised message fuzzification, followed a more predictable pattern when sent in structured sequences.

Beyond reporting protocol compliance, the given scenario exposed several implementation-level weaknesses in the tested backend. First, the unconditional server termination triggered by out-of-sequence or duplicated *BootNotification* messages (Table 3, codes 3 and 4) indicates that the connection and registration handler neither gracefully rejects nor rate-limits repeated boot requests. Consequently, a single malformed or replayed *BootNotification* is sufficient to crash the backend, exposing a low-cost denial-of-service vulnerability rather than simply a protocol violation. Second, the unconditional acceptance of *DataTransferReq* messages (100% across all trials) suggests that vendor-specific payloads are forwarded without schema validation or whitelisting. This behaviour aligns with prior reports identifying *DataTransferReq* as a common injection point in OCPP backends [46], and our results provide implementation-level evidence that this weakness persists in the evaluated CSMS. Finally, *Get15118EVCertificateReq* exhibited a response latency of 573 ms, almost 50× higher than the next slowest message, indicating that certificate processing likely involves an expensive, uncached operation. This disproportionately expensive code path could be exploited through repeated requests to amplify computational load and facilitate resource-exhaustion attacks. Collectively, these findings demonstrate that EVECTOR's fuzzification capability reveals concrete security-relevant implementation flaws that extend beyond protocol conformance testing, providing actionable insights into denial-of-service, input validation, and resource management weaknesses.

4.3 | EVECTOR Overhead

To evaluate the system overhead of EVECTOR, we conducted experiments using the MockSim. Figure 9 shows the performance overhead as the simulated environment scales from 16 to 1024 EV and EVSE instances. In this experimental setup, each EV connects to an EVSE simultaneously with a completely depleted battery and only disconnects once the vehicle is fully charged. We evaluated three primary metrics: average CPU usage, peak memory usage, and the total runtime of the test. The results indicate that the average CPU usage remains nearly constant in all cases, staying approximately 8.5%. Peak memory usage demonstrates an exceptionally small increase relative to the number of vehicles, scaling from roughly 35 to 40 MB. In contrast, the total runtime scales roughly proportionally to the number of simulated cars, taking approximately 40 min to process 1024 vehicles.

As scaling up the simulation to hundreds of vehicles generates a massive and continuous influx of EV charging logs and telemetry, it is critical that data storage does not become a system bottleneck. Therefore, we also evaluated the latency of EVECTOR's database module. As EVECTOR relies on MongoDB, a NoSQL database specifically selected to handle large-scale simulations by providing high-speed queries and flexible data storage while maintaining data integrity. Under a stress test of 100,000 read/write operations, the database demonstrated an average write latency of 0.11 ms and an average read latency of 0.12 ms. Overall, these findings collectively demonstrate that the structural overhead introduced by the EV orchestrator and the database module is marginal.



(a) CPU usage (b) Peak memory usage (c) Total runtime

FIGURE 9 | EVECTOR overhead.

5 | Discussion

While EVECTOR offers an advanced environment for exploring cyber-physical threat vectors, it is important to explicitly define its operational limitations and validation boundaries. First, our framework is entirely simulation-based. Although we execute production-grade open-source software controllers (e.g., ISO15118 and EVerest), the framework currently lacks full physical validation via physical hardware-in-the-loop (HIL) testbeds or real physical EVSE component integration. Second, our modelling of physical-layer faults, such as the broken wire attack implemented via Pumba network manipulation, operates at the digital link and packet communication bounds (Layers 1–3). This method does not replicate continuous physical-layer analogue anomalies, such as signal integrity degradation, high-frequency wave reflections, or shifting contact resistances along physical lines. Future expansions will focus directly on resolving these constraints.

5.1 | Hardware-in-the-Loop and Field Validation

We are developing a programmatic hardware bridging adapter to connect EVECTOR to physical EVSE controllers running embedded operating systems. This will allow researchers to validate simulated attack behaviours against actual hardware components under safe operating parameters. Moreover, the physical layer faults in the current implementation, such as broken wire attack is implemented via Pumba network manipulation, operates at the digital link and packet communication bounds (Layers 1–3). This method does not replicate continuous physical-layer analogue anomalies, such as signal integrity degradation, high-frequency wave reflections or shifting contact resistances along physical lines. Future work will extend this beyond binary packet loss to intermittent connectivity and analogue signal degradation, ideally validated against a small hardware test setup.

5.2 | Integration of Additional EV Simulators

Expanding EVECTOR to support a wider range of EV simulators, including proprietary and HIL simulators, will improve its applicability across different charging environments. This will enable more realistic testing scenarios and broader compatibility with real-world EV infrastructure.

5.3 | Incorporation of Additional Attack Types

While the current implementation of *attack orchestrator* focuses on broken wire and frame fuzzification attacks, future iterations will incorporate a diverse range of cyber threats, such as relay attacks, man-in-the-middle attacks, and side-channel attacks on charging communication protocols. This will provide a comprehensive security assessment of vulnerabilities across multiple layers of the charging ecosystem.

5.4 | Development of a User Interface for EVECTOR

We are working on developing a graphical user interface for EVECTOR to enhance usability, allowing security researchers and infrastructure operators to configure attack scenarios, monitor real-time simulation data and visualise results more intuitively. This will improve accessibility and ease of deployment for broader adoption in industry and academia.

5.5 | Closed-Loop Time Synchronisation for Large-Scale Scenarios

We are working on developing a shared simulation clock or barrier synchronisation mechanism between event-driven and time-stepped backends to support large-scale, tightly coupled urban attack scenarios. Such a mechanism would enable deterministic coordination between protocol-, firmware- and grid-level simulators, ensuring bounded-latency interactions while preserving temporal consistency across heterogeneous simulation engines.

5.6 | Predictive Model for Future Attack Prediction

Leveraging the collected data from the attack simulations, machine learning models will be developed to predict potential

(future) cyberattacks on EV charging infrastructure. By analysing historical attack patterns and system behaviour, EVECTOR could provide early warnings and proactive defence mechanisms, reducing the risk of large-scale disruptions.

6 | Conclusion

This paper presented EVECTOR, a multilayer orchestrator that integrates existing simulators for the EV-charging ecosystem. The motivation for utilising different backend simulators is to enable each to be used for its specific design objectives, preventing complexity and dependencies that may be introduced if these are integrated. While EVerest is one of the most widely used simulators for charging station, ACN-Sim provides a better mechanism for considering aggregate charging schedules when multiple EVs connect to a charging station. Depending on the type of vulnerability assessment being carried out, EVECTOR can invoke the relevant backend simulator automatically. Additionally, the EVECTOR orchestrator also supports conducting cyberattacks on the EV ecosystem and a database to systematically evaluate threats in EV charging systems, thereby providing a much wider vulnerability testing environment, and one which can be adapted and extended.

We evaluated EVECTOR using two different attack scenarios: a broken wire attack (a cyber-physical attack) and a frame fuzzification attack focused on a particular communication protocol used in the EV charging process (a cyberattack focused on altering data frames), generated using the attack orchestrator. The results demonstrate that EVECTOR can seamlessly orchestrate and execute these attacks while accurately monitoring their effects on the EV ecosystem, offering valuable insights into potential vulnerabilities and provide the basis to improve system resilience. Moreover, the framework's modularity allows for the integration of new attack models, security policies, and countermeasures, making it a powerful tool for ongoing research in EV security and threat mitigation. Although the paper discusses the broken wire and frame fuzzification attacks, new attack types can be implemented by extending the abstract `Attack` class. For example, a distributed denial of service attack against the grid-side control interface could target the EVSE's connection to the energy management layer with a high-rate stream of well-formed but excessive control messages, reusing the rate/latency instrumentation already used in the fuzzification (Section 4.2).

Author Contributions

Devki Nandan Jha: conceptualization, investigation, methodology, project administration, software, supervision, visualization, writing – original draft, writing – review and editing. **Tomasz Szydlo:** conceptualization, investigation, methodology, software, supervision, validation, visualization, writing – original draft, writing – review and editing. **Nima Valizadeh:** investigation, methodology, software, validation, visualization, writing – original draft, writing – review and editing. **Ringo Sham:** investigation, methodology, software, validation, visualization, writing – original draft, writing – review and editing. **Aleksandra Edwards:** investigation, software, validation, writing – original draft. **Amrit Kumar:** methodology, software, validation, visualization, writing – original draft, writing – review and editing. **Amanjot Kaur:** investigation, methodology, software, validation, writing – original draft. **Bo Wei:** conceptualization, supervision, writing – review and editing. **Vijay Kumar:** investigation, writing – review and editing. **Kai Li Lim:** writing – review and editing. **Rajiv Ranjan:** conceptualization, funding acquisition, project administration, supervision, writing – review and editing. **Omer Rana:** conceptualization, funding acquisition, investigation, methodology, project administration, supervision, validation, writing – original draft, writing – review and editing.

Acknowledgements

This study was partly supported by the National Edge AI Hub for Real Data: Edge Intelligence for Cyberdisturbances and Data Quality (UKRI EPSRC EP/Y028813/1); EPSRC UK-Australia Centre in a Secure Internet of Energy: Supporting Electric Vehicle Infrastructure at the 'Edge' of the Grid (EPSRC, EP/W003325/1) and the UK-US 'Clean Energy and Equitable Transport Solutions' (CLEETs) Centre (https://www.cleets-global-center.org/).

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request. We will release our Orchestrator code on GitHub after paper acceptance.

Endnotes

[1] https://www.mongodb.com/

[2] https://github.com/jacoscaz/typed-ocpp

References

1. C. Lester, "Global EV Sales Reach 20.7 Million Units in 2025, Growing by 20%," Benchmark Mineral Intelligence, accessed 25 February 2026, https://source.benchmarkminerals.com/article/global-ev-sales-reach-20-7-million-units-in-2025-growing-by-20.

2. R. P. Narasipuram and S. Mopidevi, "A Technological Overview & Design Considerations for Developing Electric Vehicle Charging Stations," *Journal of Energy Storage* 43 (2021): 103225.

3. M. S. Mastoi, S. Zhuang, H. M. Munir, et al., "An In-Depth Analysis of Electric Vehicle Charging Station Infrastructure, Policy Implications, and Future Trends," *Energy Reports* 8 (2022): 11504–11529.

4. S. S. Acharige, M. E. Haque, M. T. Arif, N. Hosseinzadeh, K. N. Hasan, and A. M. T. Oo, "Review of Electric Vehicle Charging Technologies, Standards, Architectures, and Converter Configurations," *IEEE Access* 11 (2023): 41218–41255.

5. M. Li, Y. Wang, P. Peng, and Z. Chen, "Toward Efficient Smart Management: A Review of Modeling and Optimization Approaches in Electric Vehicle-Transportation Network-Grid Integration," *Green Energy and Intelligent Transportation* 3, no. 6 (2024): 100181.

6. J. Antoun, M. E. Kabir, B. Moussa, R. Atallah, and C. Assi, "A Detailed Security Assessment of the EV Charging Ecosystem," *IEEE Network* 34, no. 3 (2020): 200–207.

7. J. Johnson, B. Anderson, B. Wright, et al., *Cybersecurity for Electric Vehicle Charging Infrastructure*, SAND2022-9315 (Sandia National Laboratories, 2022).

8. M. D. Mwanje, O. Kaiwartya, D. N. Jha, A. M. Khasawneh, A. Sadiq, and Y. Cao, "Misbehaviour Prediction in CAV Network Using Aggregated Trust Analysis," *International Journal of Intelligent Transportation Systems Research* 23, no. 3 (2025): 1–17.

9. A. Kaur, N. Valizadeh, D. N. Jha, et al., "Cybersecurity Challenges in the EV Charging Ecosystem," *ACM Computing Surveys* 58, no. 1 (2025): 1–32.

10. Z. Li, C. Gong, Y. Lin, et al., "Continual Driver Behaviour Learning for Connected Vehicles and Intelligent Transportation Systems: Framework, Survey and Challenges," *Green Energy and Intelligent Transportation* 2, no. 4 (2023): 100103.

11. E. Sayegh, "Is Cybersecurity The Achilles' Heel Of The Electric Vehicle Revolution?," Forbes, accessed 27 February 2026, https://www.forbes.com/sites/emilsayegh/2024/03/12/is-cybersecurity-the-achilles-heel-of-the-electric-vehicle-revolution/.

12. A. Hamedshnain, L. Hussein, K. Sandeep, D. Arapbaeva, and S. Ashirboyev, "Building Cybersecurity Defenses for EV Charging Infrastructure in Smart Cities," in *Adaptive Technologies for Sustainable Growth* (CRC Press, 2026), 306–311.

13. M. Aybar-Mejía, S. Cuevas, H. Encarnación, et al., "Securing the EV Charging Ecosystem: Protocol-Level Vulnerabilities, Defenses, and a Research Roadmap," *Journal of Electrical and Computer Engineering* 2025, no. 1 (2025): 5770202.

14. T. Aljohani and A. Almutairi, "A Comprehensive Survey of Cyberattacks on EVs: Research Domains, Attacks, Defensive Mechanisms, and Verification Methods," *Defence Technology* 42 (2024): 31–58.

15. J. Ye, L. Guo, B. Yang, et al., "Cyber–Physical Security of Powertrain Systems in Modern Electric Vehicles: Vulnerabilities, Challenges, and Future Visions," *IEEE Journal of Emerging and Selected Topics in Power Electronics* 9, no. 4 (2020): 4639–4657.

16. N. Hettiarachchi, S. Hakak, and K. Mandal, "A Survey on Authentication Protocols of Dynamic Wireless EV Charging," *Computer Communications* 230 (2025): 108008.

17. T.-V. Nguyen, H. Sun, H. Wang, and R. Q. Hu, "Authentication and PHY-Security Schemes for Electric Vehicle Dynamic Wireless Charging," *IEEE Transactions on Vehicular Technology* 73, no. 2 (2023): 1698–1712.

18. R. P. Parameswarath, P. Gope, and B. Sikdar, "User-Empowered Privacy-Preserving Authentication Protocol for Electric Vehicle Charging Based on Decentralized Identity and Verifiable Credential," *ACM Transactions on Management Information Systems (TMIS)* 13, no. 4 (2022): 1–21.

19. M. T. Rana, M. Numan, M. Yousif, T. Hussain, A. Z. Khan, and X. Zhao, "Enhancing Sustainability in Electric Mobility: Exploring Blockchain Applications for Secure EV Charging and Energy Management," *Computers and Electrical Engineering* 119 (2024): 109503.

20. A. Aydeger, E. Zeydan, J. Mangues-Bafalluy, S. S. Arslan, and Y. Turk, "Enhancing Electric Vehicle Security and Privacy Through Decentralized Identity Management," *Digital Threats: Research and Practice* 6, no. 3 (2025): 1–20.

21. M. Abid, M. K. Benkaddour, M. Benouis, and A. Bouazza, "Decentralized Anomaly Detection in Electric Vehicle Supply Equipment via Federated Learning and Blockchain Integration," *Computing* 108, no. 3 (2026): 41.

22. A. Hussain, A. Yadav, and G. Ravikumar, "Anomaly Detection Using Bi-Directional Long Short-Term Memory Networks for Cyber-Physical Electric Vehicle Charging Stations," *IEEE Transactions on Industrial Cyber-Physical Systems* 2 (2024): 508–518.

23. H. Jahangir, S. Lakshminarayana, and H. V. Poor, "Charge Manipulation Attacks Against Smart Electric Vehicle Charging Stations and Deep Learning-Based Detection Mechanisms," *IEEE Transactions on Smart Grid* 15, no. 5 (2024): 5182–5194.

24. A. Sharma, S. Rani, and M. Shabaz, "Artificial Intelligence-Augmented Smart Grid Architecture for Cyber Intrusion Detection and Mitigation in Electric Vehicle Charging Infrastructure," *Scientific Reports* 15, no. 1 (2025): 21653.

25. S. Orfanoudakis, C. Diaz-Londono, Y. E. Yılmaz, P. Palensky, and P. P. Vergara, "EV2Gym: A Flexible V2G Simulator for EV Smart Charging Research and Benchmarking," *IEEE Transactions on Intelligent Transportation Systems* 26, no. 2 (2024): 2410–2421.

26. Z. Lee, D. Johansson, and S. H. Low, "ACN-Sim: An Open-Source Simulator for Data-Driven Electric Vehicle Charging Research," in *Proceedings of the Tenth ACM International Conference on Future Energy Systems* (ACM, 2019), 411–412.

27. EcoG-io, "Implementation of the ISO 15118 Communication Protocol," GitHub, accessed 25 February 2026, https://github.com/EcoG-io/iso15118.

28. "EVerest," LF Energy , accessed 25 February 2026, https://lfenergy.org/projects/everest/.

29. "EVShield Library Reference," mindsensors.com, accessed 25 February 2026, https://www.mindsensors.com/reference/EVShield/html/.

30. M. Ghafouri, E. Kabir, B. Moussa, and C. Assi, "Coordinated Charging and Discharging of Electric Vehicles: A New Class of Switching Attacks," *ACM Transactions on Cyber-Physical Systems (TCPS)* 6, no. 3 (2022): 1–26.

31. S. Hamdare, D. J. Brown, O. Kaiwartya, Y. Cao, and M. Jugran, "MitM Cyber Risk Analysis in OCPP Enabled EV Charging Stations," in *2024 4th Intelligent Cybersecurity Conference (ICSC)* (IEEE, 2024), 151–157.

32. S. Hamdare, D. J. Brown, D. N. Jha, et al., "Cyber Defense in OCPP for EV Charging Security Risks," *International Journal of Information Security* 24, no. 3 (2025): 134.

33. S. Köhler, R. Baker, M. Strohmeier, and I. Martinovic, "Brokenwire: Wireless Disruption of CCS Electric Vehicle Charging," preprint, arXiv, February 7, 2022, https://doi.org/10.48550/arXiv.2202.02104.

34. Switch-EV, "Switch-EV," GitHub, accessed 25 February 2026, https://github.com/SwitchEV.

35. Switch-EV, "RISE V2G," GitHub, accessed 25 February 2026, https://github.com/SwitchEV/RISE-V2G.

36. EcoG-io, "Josev," GitHub, accessed 25 February 2026, https://github.com/EcoG-io/josev.

37. Ecognize, "OpenV2G," GitHub, accessed 25 February 2026, https://github.com/Ecognize/openv2g.

38. The Mobility House, "Mobilityhouse OCPP Simulator," GitHub, accessed 25 February 2026, https://github.com/mobilityhouse/ocpp.

39. S. Saxena, "Vehicle-to-Grid Simulator," OSTI , accessed February 25, 2026, https://www.osti.gov/biblio/1437011.

40. E. S. Rigas, S. Karapostolakis, N. Bassiliades, and S. D. Ramchurn, "EVLibSim: A Tool for the Simulation of Electric Vehicles' Charging Stations Using the EVLib Library," *Simulation Modelling Practice and Theory* 87 (2018): 99–119.

41. E. Balogun, E. Buechler, S. Bhela, S. Onori, and R. Rajagopal, "EV-EcoSim: A Grid-Aware Co-Simulation Platform for the Design and Optimization of Electric Vehicle Charging Infrastructure," *IEEE Transactions on Smart Grid* 15, no. 3 (2023): 3114–3125.

42. T. Morstyn, K. A. Collett, A. Vijay, et al., "OPEN: An Open-Source Platform for Developing Smart Local Energy System Applications," *Applied Energy* 275 (2020): 115397.

43. G. Karatzinis, C. Korkas, M. Terzopoulos, et al., "Chargym: An EV Charging Station Model for Controller Benchmarking," in *IFIP International Conference on Artificial Intelligence Applications and Innovations* (Springer, 2022), 241–252.

44. A. Sanghvi and T. Markel, "Cybersecurity for Electric Vehicle Fast-Charging Infrastructure," in *2021 IEEE Transportation Electrification Conference & Expo (ITEC)* (IEEE, 2021), 573–576.

45. S. Y. Mattepu, M. Richter, and S. Balischewski, "OCSS: An Application to Simulate Multiple Charging Stations Which Uses an Open Charge Point Protocol for Communication," in *2022 IEEE 13th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)* (IEEE, 2022), 1–5.

46. K. Sarieddine, M. A. Sayed, S. Torabi, et al., "Uncovering Covert Attacks on EV Charging Infrastructure: How OCPP Backend Vulnerabilities Could Compromise Your System," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security* (ACM, 2024), 977–989.

47. K. Sarieddine, M. A. Sayed, D. Jafarigiv, R. Atallah, M. Debbabi, and C. Assi, "A Real-Time Cosimulation Testbed for Electric Vehicle Charging and Smart Grid Security," *IEEE Security & Privacy* 21, no. 4 (2023): 74–83.

48. A. Ledenev, "Pumba—Chaos Testing for Docker," Terra Nullius, accessed March 5, 2026, https://alexei-led.github.io/post/pumba_docker_chaos_testing/.

49. G. Coppoletta, A. Kaur, N. Valizadeh, O. Rana, R. Gjomemo, and V. Venkatakrishnan, "OCPPStorm: A Comprehensive Fuzzing Tool for OCPP Implementations," paper presented at the Symposium on Vehicle Security and Privacy (VehicleSec) (alongside NDSS Symposium), San Diego, CA, February 26–March 1, 2024.

50. D. Steinhöfel and A. Zeller, "Input Invariants," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022* (Association for Computing Machinery, 2022), 583–594.

Appendix: EV Orchestrator State Transition in EVECTOR

The diagram in Figure A1 shows the series of interactions involved in the EV charging process in the EV orchestrator of EVECTOR. The simulation begins with the user specifying a predefined charging scenario, defining the configuration of various components, including EV type,

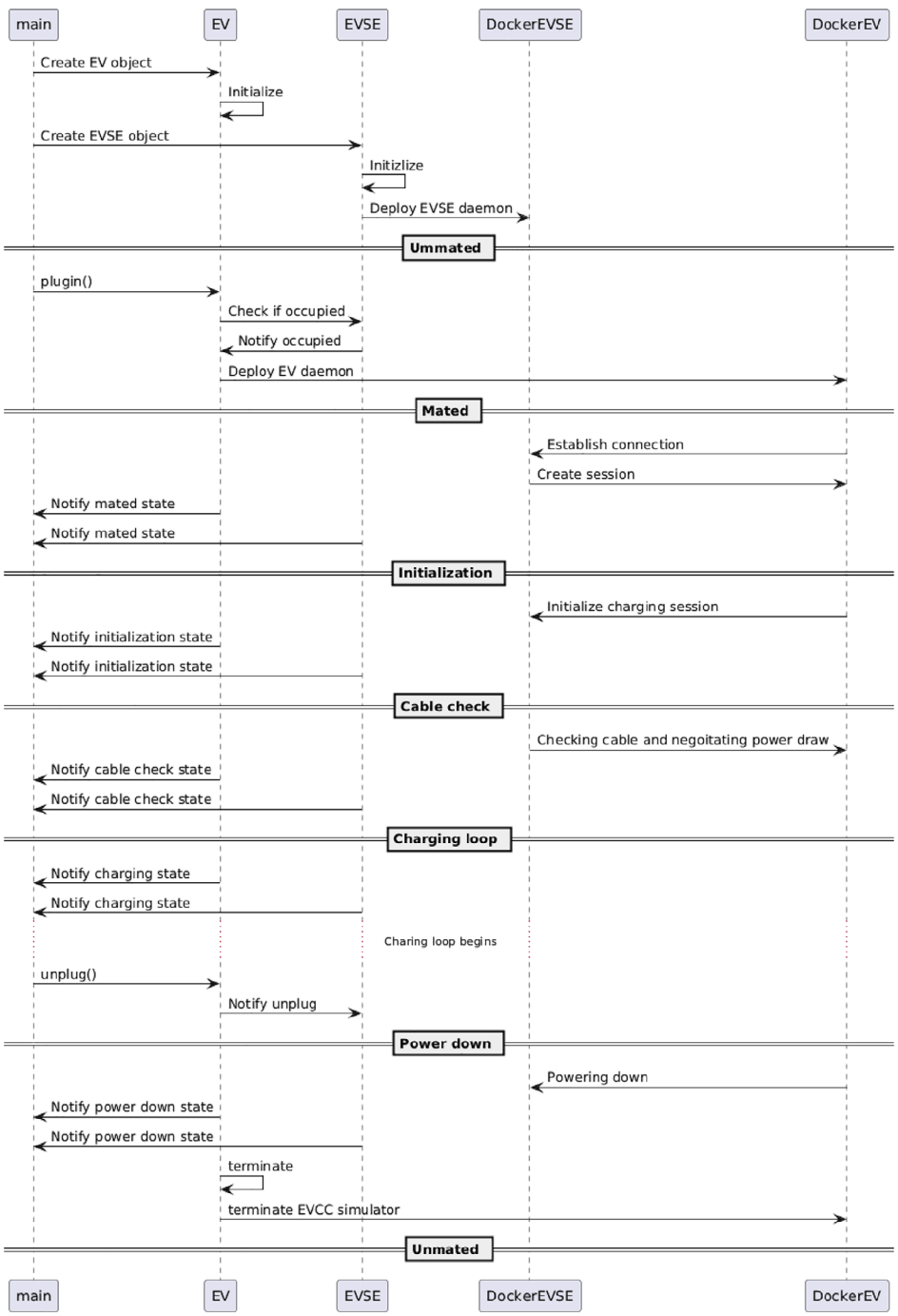


FIGURE A1 | Sequence diagram showing normal charging process in EVECTOR.

EVSE type, charge required and EVSE charge rate. It also allows a user to specify any conditional triggers that affect the charging system's behaviour.

Given the configuration, the simulator first creates and initialises EV and EVSE objects. Depending on the simulator specified in the configuration, respective instances of EV and EVSE daemons will be created. Figure A1 shows the interaction for the docker ISO simulator. Initially, the EVSE is unmated. The EV checks the state of EVSE, and if it is unoccupied, an EV daemon is deployed. Next, communication between the EV and EVSE daemons will be established, and the charging session will be initialised. Following that, a compulsory cable check operation will be performed, and if that is successful, charging will start. The charging will finish when either the EV battery is charged to 100% or a user initiates an interrupt. This leads to an unplug event, and the EVSE daemon will start powering down. The simulator is notified of that operation, which terminates the EV daemon. After the process finishes, all the system logs are collected and stored in the database for further processing.

By leveraging the modular architecture described in the class diagram, the system ensures flexibility in testing various charging scenarios and protocols, supporting both software-based simulations and real-world EVSE interactions.