Adversarial Reinforcement Learning for Detecting False Data Injection Attacks in Vehicular Routing

Taha Eghtesad
Informatics and Intelligent Systems
Pennsylvania State University
University Park, USA
tahaeghtesad@psu.edu

Yevgeniy Vorobeychik
Computer Science & Engineering
Washington University in St. Louis
St. Louis, USA
yvorobeychik@wustl.edu

Aron Laszka
Informatics and Intelligent Systems
Pennsylvania State University
University Park, USA
laszka@psu.edu

Published in the Proceedings of the 17th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS 2026).

***Abstract*—In modern transportation networks, adversaries can manipulate routing algorithms using false data injection attacks, such as simulating heavy traffic with multiple devices running crowdsourced navigation applications, to mislead vehicles toward suboptimal routes and increase congestion. To address these threats, we formulate a strategically zero-sum game between an attacker, who injects such perturbations, and a defender, who detects anomalies based on the observed travel times of network edges. We propose a computational method based on multi-agent reinforcement learning to compute a Nash equilibrium of this game, providing an optimal detection strategy, which ensures that total travel time remains within a worst-case bound, even in the presence of an attack. We present an extensive experimental evaluation that demonstrates the robustness and practical benefits of our approach, providing a powerful framework to improve the resilience of transportation networks against false data injection. In particular, we show that our approach yields approximate equilibrium policies and significantly outperforms baselines for both the attacker and the defender.**

***Index Terms*—Transportation networks, False-data injection, Navigation system, Cybersecurity, Deep reinforcement learning, Multi-agent reinforcement learning**

I. Introduction

The rise of crowdsourced navigation applications, such as Google Maps, Waze, and Apple Maps, has revolutionized urban mobility by offering route suggestions for drivers based on real-time traffic conditions. However, these platforms are increasingly vulnerable to a growing threat vector: false data injection (FDI) attacks [1], [2]. These attacks aim to deliberately manipulate crowdsourced data to mislead routing algorithms and disrupt transportation networks.

FDI attacks exploit the decentralized and participatory nature of crowdsourced navigation systems. For example, researchers demonstrated such an attack by placing mobile devices running Google Maps in a cart and slowly dragging it along a street [1]. The system misinterpreted the data as indicating severe congestion and rerouted vehicles away from the area. This illustrates how easily attackers can deceive these systems using relatively simple techniques. In addition to manipulating crowdsourced data, attackers may also target physical traffic sensors, exploiting the poor state of cybersecurity in many transportation deployments. For example, tampering with vehicle-counting sensors or signal-timing systems can distort traffic data and amplify the impact of FDI attacks [3], [4].

FDI attacks pose a serious threat to transportation networks. By injecting false data, attackers can trick navigation systems into rerouting countless vehicles, creating widespread gridlock. The most critical impact of this deliberate congestion is the potential to delay emergency responders, where even minutes lost can have life-threatening outcomes. Beyond this critical impact, cascading effects include longer commute times, increased fuel consumption and vehicle emissions, and disruptions to public transit and logistics, inflicting a significant economic and environmental toll.

One approach to mitigating FDI attacks is to promptly detect them based on anomalous traffic observations, complementing traditional cybersecurity measures, such as access control. Traffic observations can include measurements of physical variables, such as the congestion levels or estimated traffic speeds of specific road segments. By comparing these traffic observations with data recorded during normal operations, it is possible to identify anomalies that indicate an attack. Classical anomaly detection methods, such as applications of statistical and machine learning models, could play a crucial role in identifying unusual traffic patterns.

However, attackers may attempt to carry out stealthy attacks that maximize disruption while minimizing the likelihood of detection. Such stealthy attacks aim to carefully manipulate data to remain within the range of normal variation, thereby evading classical anomaly detection methods. In addition, attackers might adapt their tactics if they are aware that the detector was trained on previously executed attacks, further complicating detection efforts, which need to anticipate such adaptive attacks. The literature has not yet explored strategies to counter such stealthy and adaptive attacks, leaving a critical gap in current defenses.

These challenges underscore the need for a robust detection mechanism against FDI attacks on transportation, capable of thwarting the attackers' efforts to avoid detection. Developing these capabilities is essential for enhancing the resilience of crowdsourced navigation systems against evolving threats.

Recent research has focused on identifying and simulating effective attack strategies. Eghtesad *et al.* [5] introduced a

scalable reinforcement learning algorithm that generates attack strategies capable of maximizing network disruption. Yang *et al.* [6] and Yu *et al.* [7] explored the impact of random attacks, where false demands are distributed across network edges using uniform or Gaussian distributions, and proposed defenses such as trusted, unperturbed sensors to mitigate these threats. However, identifying vulnerabilities is only the first step. The critical open challenge, which we address in this work, is the design of a robust detection mechanism that can withstand strategic, adaptive attackers. We propose a robust detection model that identifies attacks based on the observed travel times of network edges, anticipating a strategic attacker who responds by launching a worst-case stealthy attack.

a) Contributions: Our contributions are threefold: (1) we formulate a strategic zero-sum game between an attacker employing FDI and a defender detecting such attacks, (2) we demonstrate that solving for the Nash equilibrium of this game provides the optimal detection strategy, ensuring resilience against adaptive adversarial strategies, and (3) we leverage a policy-space response algorithm to compute equilibrium strategies efficiently, providing an optimal detection mechanism. This approach ensures minimal disruption to the transportation network under a worst-case attack, thereby enhancing its robustness against FDI manipulations.

b) Organization: Section II reviews related work on attacks against transportation networks and on reinforcement learning for cyber defense. Section III covers the theoretical background of our approach. Section IV introduces our model of transportation networks and false data injection attacks. Section V presents our game-theoretic detection model and our computational approach. Section VI demonstrates the effectiveness of our approach through experiments. Finally, Section VII provides concluding remarks.

II. Related Work

The vulnerability of transportation networks to various attacks has been extensively studied, and the application of reinforcement learning in attack detection has been explored in other domains. However, the detection of attacks on crowd-sourced navigation applications remains underexplored.

a) Attacks Against Transportation Networks: Malicious actors can exploit vulnerabilities in transportation networks to manipulate drivers' route choices using methods such as sending malicious SMS messages [8], physically altering road signs [9], tampering with traffic signals [3], [10], or injecting false data into crowdsourced navigation applications [1], [2], [5]–[7]. Prior work has explored the vulnerabilities of navigation applications to adversarial manipulations and their impacts on traffic congestion [6]–[8], [11]. Further, Eghtesad *et al.* [5] have developed a reinforcement learning-based framework that determines the worst-case impact of FDI attacks on navigation applications.

b) Reinforcement Learning for Detection: Reinforcement learning is a promising tool for the detection and mitigation of adversarial attacks. Several prior works have explored methods for detecting and mitigating various types of attacks using reinforcement learning. For example, Sargolzaei *et al.* [12] focus on the detection and mitigation of false data injection attacks in distributed control systems. Malialis *et al.* [13] examine network intrusion detection techniques using reinforcement learning, while Hu *et al.* [14] leverage reinforcement learning to address zero-day attacks. Finally, Tong *et al.* [15] investigate strategies for prioritizing relevant alerts in network intrusion detection systems using multi-agent reinforcement learning.

III. Preliminaries

The strategic conflict between an attacker and a defender can be modeled as a stochastic game involving the two players. In this framework, the attacker selects an attack policy, while the defender selects a detection policy. When one player commits to a policy, their opponent's strategic choice transforms into a Markov decision process. If the policies form an equilibrium, neither player is incentivized to deviate and pursue an alternative policy.

a) Markov Decision Processes (MDP): The tuple $\langle S, A, R, T\rangle$ defines an MDP, where S represents the state space, A represents the action space, $R(s^t, a^t) \mapsto r^t \in \mathbb{R}$ is the reward function that determines the reward r^t received for taking action $a^t \in A$ in state $s^t \in S$ at time step t, and $T(s^t, a^t, s^{t+1}) \mapsto [0, 1]$ specifies the probability that taking action a^t in state s^t will result in a transition to state $s^{t+1} \in S$ at the next time step $t + 1$.

A **policy** function in an MDP is a plan of action $\pi(s^t) \mapsto a^t$ that an agent (i.e., a player) will follow. The utility value $u(\pi)$ of a policy function π can be expressed as the discounted sum of rewards that the agent will receive by following the policy:

$$u(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r^t \mid \pi\right] \tag{1}$$

A solution to an MDP is a policy that maximizes the utility of the agent: $\pi^* = \operatorname{argmax}_\pi u(\pi)$. We let $u^* = u(\pi^*)$ denote the utility of this optimal policy. A reinforcement learning (RL) algorithm can be used to computationally find such a policy.

b) Stochastic Games: We can define a stochastic two-player game between the attacker and the defender. This game can be expressed as a zero-sum game $G = (\Pi_p, \Pi_{\bar{p}}, u)$, where players choose an MDP *policy* as their **pure strategy** $\pi_p \in \Pi_p$ from their strategy set Π_p, i.e., set of all policies available to them, as their strategy to play. The utility of the game when player p follows π_p and player $\bar{p}$ follows $\pi_{\bar{p}}$ is denoted $u(\pi_p, \pi_{\bar{p}}) \mapsto \mathbb{R}$. Player p tries to maximize, while their opponent $\bar{p}$ tries to minimize utility by choosing a policy.

We can also define a **mixed strategy** σ_p as a probability distribution over the player's pure strategy set Π_p, where $\sigma_p(\pi_p)$ is the probability of the player choosing $\pi_p \in \Pi_p$. The expected utility given mixed strategies σ_p and $\sigma_{\bar{p}}$ is:

$$u(\sigma_p, \sigma_{\bar{p}}) = \sum_{\pi_p \in \Pi_p} \sum_{\pi_{\bar{p}} \in \Pi_{\bar{p}}} \sigma_p(\pi_p)\sigma_{\bar{p}}(\pi_{\bar{p}})u(\pi_p, \pi_{\bar{p}}). \tag{2}$$

A pair of mixed strategies $\sigma_p^*, \sigma_{\bar{p}}^*$ for a zero-sum game form a **mixed strategy Nash equilibrium** (MSNE) *iff*:

$$u(\sigma_p, \sigma_{\bar{p}}^*) \leq u(\sigma_p^*, \sigma_{\bar{p}}^*) \leq u(\sigma_p^*, \sigma_{\bar{p}}) \quad \forall \sigma_p, \sigma_{\bar{p}} \tag{3}$$

MSNE ensures that neither player p nor their opponent $\bar{p}$ can improve their expected utility by unilaterally deviating from σ_p^* or $\sigma_{\bar{p}}^*$, respectively. *Hence, each player is disincentivized from deviating from their MSNE strategy.*

Given the strategies and utility values, we can compute the MSNE of such zero-sum stochastic games using linear programming [16].

c) Double Oracle (DO): The MSNE strategies in zero-sum games with large strategy sets, where the enumeration of pure strategies is infeasible, can be calculated using the DO algorithm [17]. The algorithm begins by initializing a small subset of the strategy set for each player, denoted $\Pi_p^0 \subset \Pi_p$ and $\Pi_{\bar{p}}^0 \subset \Pi_{\bar{p}}$. At each iteration i, the algorithm computes an MSNE, denoted $\sigma_p^{*,i}$ and $\sigma_{\bar{p}}^{*,i}$, of the subgame G^i that is defined by the current strategy sets Π_p^i and $\Pi_{\bar{p}}^i$.

Once an MSNE is computed, each player calculates their **best response** (BR) pure strategy $\pi_p^{i+1} = \operatorname{argmax}_{\pi_p \in \Pi_p} u(\pi_p, \sigma_{\bar{p}}^{*,i})$ or $\pi_{\bar{p}}^{i+1} = \operatorname{argmin}_{\pi_{\bar{p}} \in \Pi_{\bar{p}}} u(\sigma_p^{*,i}, \pi_{\bar{p}})$, respectively, which is the pure strategy that maximizes or minimizes their utility given the mixed strategy $\sigma_{\bar{p}}^{*,i}$ or $\sigma_p^{*,i}$ of their opponent. These best response strategies from the players' strategy sets (Π_p and $\Pi_{\bar{p}}$) are then added to the player's current strategy sets (Π_p^i and $\Pi_{\bar{p}}^i$):

$$\Pi_p^{i+1} \leftarrow \Pi_p^i \cup \{\pi_p^{i+1}\} \quad \text{and} \quad \Pi_{\bar{p}}^{i+1} \leftarrow \Pi_{\bar{p}}^i \cup \{\pi_{\bar{p}}^{i+1}\} \tag{4}$$

The process is repeated iteratively, updating the strategy sets in each iteration. The algorithm continues until the MSNE utility of the subgames G^i converges to the MSNE of the game G [17].

d) Policy Space Response Oracles (PSRO): In games with enormous policy spaces (i.e., strategy sets), such as video games, it is infeasible to enumerate all the policies when searching for a best response. Further, given the opponent is committed to a policy, it is still infeasible to solve the resulting MDP for the opponent using RL.

The PSRO algorithm [18] extends DO by using Deep Reinforcement Learning (DRL) as an *approximate best response oracle*, which finds an approximate—due to the approximation nature of deep neural networks and reinforcement learning—best response to the opponent's MSNE strategy within the player's pure strategy set: $\pi_p^{i+1} \approx \operatorname{argmax}_{\pi_p \in \Pi_p} u(\pi_p, \sigma_{\bar{p}}^{*,i})$ and $\pi_{\bar{p}}^{i+1} \approx \operatorname{argmin}_{\pi_{\bar{p}} \in \Pi_{\bar{p}}} u(\sigma_p^{*,i}, \pi_{\bar{p}})$. In this framework, each player (i.e., both the attacker and the defender) selects the parameters of a DRL policy as their pure strategy within the game. The strategy set that is defined by all possible DRL policies can also be referred to as the *policy space*, which serves as the basis for the term *policy space response oracles*.

IV. System and Threat Model

The road network is represented as a directed graph $G = (V, E)$, where nodes denote intersections and edges denote road segments. Each edge $e = (u, v) \in E$, where $u, v \in V$, is characterized by a **free-flow travel time** f_e (time to traverse the road segment without traffic), a **capacity** c_e (maximum traffic flow rate of the segment), and two **congestion parameters**, b_e and p_e, which control travel time under congestion. When n_e vehicles are traveling on edge e, the travel time $W_e(n_e)$ of the edge is calculated using the Bureau of Public Roads (BPR) function [19], [20]:

$$W_e(n_e) = f_e \cdot (1 + b_e \, (n_e \,/\, c_e)^{p_e}) \,. \tag{5}$$

The travel demand within the network is defined by a set of trips R. Each trip $r \in R$ is a tuple $r = \langle o_r, d_r, s_r \rangle$, representing a demand of s_r vehicles traveling from an origin node $o_r \in V$ to a destination node $d_r \in V$. Through simulation, we can calculate the number of time steps T_r for vehicle trip r to reach its destination.

Consistent with recent prior work [5], we employ a dynamic, agentic step-wise simulation of traffic, where vehicles make realistic routing decisions sequentially over time, a departure from classical static network-flow models.

A. States and Transitions

The system evolves in discrete time steps t. The **state** is the set of all trip locations $\{l_r^t \in V \cup (E \times \mathbb{N})\}$, where a location can be a node $v \in V$ or a position on an edge $\langle e, \tau \rangle$ with $\tau \in \mathbb{N}$ time remaining.

When a vehicle trip r is at a node ($l_r^t = u$), it dynamically routes based on a stochastic policy. This stochastic policy models limited rationality for a more realistic simulation, representing vehicles that may not completely rely on the navigation application for route planning. In this model, each vehicle calculates the cost-to-go via each adjacent node $v \in N(u)$ as $C_v = w_{(u,v)}^t + d(v, d_r)$, where $d(v, d_r)$ is the shortest path distance based on travel times w_e^t. The probability of choosing edge (u, v) follows a Boltzmann distribution:

$$P(l_r^{t+1} = \{(u, v), \lfloor w_{(u,v)}^t \rceil\}) = \frac{e^{-\theta C_v}}{\sum_{v' \in N(u)} e^{-\theta C_{v'}}}, \tag{6}$$

where the parameter $\theta \geq 0$ controls rationality; as $\theta \to \infty$, the policy approaches the deterministic choice of shortest path.

Then, the locations are updated for $t + 1$. A vehicle trip at a node u moves onto its chosen edge (u, v). A vehicle trip on an edge $\langle e, \tau \rangle$ updates its state: if $\tau > 1$, its new state is $l_r^{t+1} = \langle e, \tau - 1 \rangle$; if $\tau = 1$, it arrives at the edge's destination node, $l_r^{t+1} = v$.

B. Threat Model

The attacker perturbs the observed travel times of the edges. Let $\boldsymbol{a}^t = \{a_e^t \in \mathbb{R}\}_{\forall_{e \in E}}$ denote the perturbations for each edge e. The observed travel time that is used to choose the path (Eq. (6)) is:

$$\hat{w}_e^t = w_e^t + a_e^t. \tag{7}$$

Our threat model diverges from Eghtesad *et al.* [5] by removing the explicit budget constraint on the attack. We argue that this constraint is superfluous because a rational attacker, seeking to maximize expected impact, is already incentivized to self-limit their attack magnitude to balance effectiveness with the inherent risk of detection. An increase in the total

magnitude of the attack increases the probability of the attack being detected and thwarted. Also note that while this threat model permits independent edge modification, an attacker facing the risk of detection is also incentivized to respect traffic flow inter-dependencies. To remain stealthy, an attacker has to mimic legitimate traffic patterns; a sudden change on an isolated edge would be easily detectable.

During an FDI attack, vehicles use $\hat{w}_e^t$ to compute shortest paths. The attacker, assumed to fully observe the environment including the network topology and the vehicle trip origins, destinations, and locations, seeks to maximize the total vehicle travel time:

$$u_a^* = \max_{\{\boldsymbol{a}^1, \boldsymbol{a}^2, \ldots\}} \sum_{r \in R} s_r \cdot T_r. \tag{8}$$

V. Game Theoretic Threat Detection

We present a model for detecting abnormal traffic patterns by formulating the system as a Partially Observable MDP (POMDP) based on the observed travel times $\hat{\boldsymbol{w}}$. Building on this, we extend the model to capture the interaction between an attacker attempting to manipulate traffic patterns by injecting perturbations into the navigation application and a detector aiming to identify such anomalies. This interaction is formulated as a two-player strategic zero-sum game, capturing the adversarial dynamics. To determine the optimal strategies for both players, we leverage the PSRO algorithm to compute the MSNE of the game. The Nash equilibrium guarantees that any non-equilibrium policy will result in suboptimal outcomes: any other attacker strategy will lead to lower total travel time, while any other defense strategy will lead to increased total travel time.

A. Defense Model

The defender observes the perturbed travel times $\hat{\boldsymbol{w}}^t$ at each time step t, and makes a decision whether to raise an alert or not:

$$d^t \in \{0, 1\}. \tag{9}$$

In the event of no attack ($\boldsymbol{a}^t = \mathbf{0}$), the defender observes the nominal travel times $\boldsymbol{w}^t$, but cannot distinguish them from the perturbed times $\hat{\boldsymbol{w}}^t$ observed under attack (without anomaly detection).

If the defender detects an attack at time t and raises an alert ($d^t = 1$), then all future perturbations are prevented, i.e., $\boldsymbol{a}^\tau = 0$ for all remaining steps $\tau \geq t$. In a real-world deployment, detection would trigger mitigation protocols, such as blocking crowdsourced data from untrusted sources or reverting routing logic to historical averages and data from trusted physical sensors. Failure to detect an ongoing attack ($d^t = 0$) enables the attacker to increase vehicle travel times T_r by continuing to perturb observed travel times $\hat{\boldsymbol{w}}^t$ (Section IV-B).

The defender's objective is to minimize the total vehicle travel time, that is, to minimize the attacker's utility. In addition, false positives (i.e., alerts raised when there is no real attack) incur a fixed cost of C_f. We can formulate the defender's objective to minimize the total travel time while minimizing the cost of false alarms:

$$u_d^* = \min_{\{d^1, d^2, \cdots\}} \sum_{r \in R} s_r \cdot T_r + C_f \cdot |F|, \tag{10}$$

where $|F|$ is the number of false positive alerts raised by the defender before an attack is correctly detected, and C_f is the cost of false positive alerts (i.e., cost of wasted time and effort). By incorporating a false positive cost, the defender can automatically manage the trade-off between false positive rate and total travel time. This is crucial since in practice, anomalies may occur without an adversary (e.g., congestion caused by an accident or large public events). Because the defender's objective function includes an explicit penalty for false positives, an optimal detector can learn to ignore legitimate congestion patterns that lack the specific intent or temporal signature of an FDI attack.

Unlike static anomaly detectors that learn a fixed baseline of normal behavior, our defender is trained against an adaptive attacker, learning to identify sophisticated and adaptive attack patterns by observing the history of travel times ($\hat{w}^t$), which reflect both the attacker's direct perturbations and their indirect, cascading effects on traffic congestion.

This challenge is amplified because observed traffic deviates from normal patterns in two ways. First, deviations arise directly from the attacker's perturbations, which alter the travel-time data used by vehicles for routing. Second, these manipulated observations cause vehicles to reroute, leading to indirect, cascading effects that produce real, yet anomalous, traffic congestion. The defender must therefore untangle anomalies caused by both the attacker's false data and the real-world consequences of those manipulations.

B. The Detection Game

We model the interaction between the attacker and the defender as a two-player game. The attacker maximizes its utility, which is the total travel time, while the defender aims to minimize it by correctly detecting adversarial perturbations while reducing the false positive alarm rate. This game is not strictly zero-sum due to the defender's *false alarm penalty* (Eq. 10). However, the defender's false alarm penalty depends only on its own strategy and is independent of the attacker's actions. Except for this penalty, the players' goals (u_a vs. u_d) are directly opposing: any gain for the attacker results in an equivalent loss for the defender in terms of total travel time. As such, the interaction is strategically equivalent to a zero-sum game. To solve the detection game, we employ PSRO (Sec. III-0c), iteratively updating the policy space of both players by identifying the best response of one player to the current MSNE strategy of the other.

C. DRL as Approximate Best-Response Oracle

In line with PSRO, we use DRL algorithms to find a player's approximate best response to their opponent's current MSNE strategy.

Fig. 1: Dividing a two-player game environment into two single-agent reinforcement learnings for the PSRO algorithm: one for player p and one for its opponent $\bar{p}$, where $\bar{p}$ plays with MSNE $\sigma_{\bar{p}}^*$ and p plays with σ_p^*, respectively.

1) Attack Oracle: The attacker's DRL oracle finds a policy to perturb the traffic observations, thereby rerouting the vehicles and increasing the total travel time. Any continuous action DRL algorithm, such as the Deep Deterministic Policy Gradient (DDPG) [21] or Proximal Policy Optimization (PPO) [22], may be applicable.

The attacker fully observes the traffic environment, including the network's topology and the vehicles' locations and route choices. This information must be engineered into useful machine-learning features that can be used as a state representation for the DRL oracle. We adopt five edge features from prior work [5]: the number of vehicles on any node whose unperturbed shortest paths include edge e (s_e^t), the number of vehicles immediately taking the edge ($\hat{s}_e^t$), the number of vehicles currently on the edge (m_e^t), the number of vehicles on any edge whose shortest paths include the edge ($\tilde{s}_e^t$), and the number of vehicles currently on the edge (n_e^t). Additionally, we incorporate two new edge features based on the network topology: **edge capacity** (c_e) and **free-flow time** (f_e). Together, these edge features form the attack oracle's observation vector:

$$\mathbf{o}_a^t = \left\langle \langle s_e^t, \hat{s}_e^t, m_e^t, \tilde{s}_e^t, n_e^t, c_e, f_e \rangle : \ \forall_{e \in E} \right\rangle. \tag{11}$$

The attacker's action is the observation perturbation applied to each edge e of the network:

$$\boldsymbol{a}^t = \langle a_e^t \,|\, a_e^t \geq 0 : \forall_{e \in E} \rangle. \tag{12}$$

To translate the attacker's objective (Eq. (8)) into an equivalent stepwise reward signal, we can reward the adversarial DRL agent by the number of vehicles that are still traveling through the network:

$$r_a^t = \sum \{ s_r \,|\, l_r^t \neq d_r : \forall_{r \in R} \}. \tag{13}$$

2) Defense Oracle: On the defense side, the oracle is simpler in comparison since it outputs a binary decision: whether to raise an alarm or not (Eq. (9)). Consequently, RL algorithms such as Deep Q-Networks (DQN) [23] or PPO are sufficient to approximate the defender's best response. The reduced complexity of the defender's decision space ensures that these methods are computationally efficient and effective in finding optimal detection policies.

The reinforcement learning algorithm for the defender observes the perturbed edge travel times of the network, same as the vehicles:

$$\boldsymbol{o}_d^t = \hat{\boldsymbol{w}}^t = \left\langle \hat{w}_e^t : \forall_{e \in E} \right\rangle. \tag{14}$$

The defender operates in a partially observable MDP; to improve the efficiency of DRL, we provide the defender with a history of observations $\boldsymbol{h}$, where $\boldsymbol{h}_t = \langle \boldsymbol{o}_d^{t-|H|}, \cdots, \boldsymbol{o}_d^t \rangle$. We set $|H| = 5$ in our experiments. The defender's policy outputs a binary action, representing the detection decision at time step t: $a_d^t \in \{0, 1\}$. Finally, the defender receives a reward based on the number of vehicles that are still traveling through the network (with a negative sign, so that the number is minimized) and a penalty $-C_f$ for a false alarm since reinforcement learning maximizes rewards:

$$r_d^t = -\sum \{ r_s \,|\, l_r^t \neq d_r : \forall_{r \in R} \} - p \tag{15}$$

$$p = \begin{cases} C_f & \text{if there is no attack at time step } t \text{ and } d^t = 1 \\ 0 & \text{if an attack is correctly detected or } d^t = 0. \end{cases}$$

D. Solving the Strategic Detection Game

By iteratively generating approximate best-responses using DRL (see Sec. III-0d) against the opponents' current MSNE strategies, the DO algorithm refines the strategy sets until convergence.

We employ PPO [22] as the DRL algorithm for both the attack and defense best-response oracles. For the attack oracle, PPO optimizes a multivariate Gaussian action distribution with a diagonal covariance matrix to output continuous perturbation values, which are then exponentially scaled to ensure that they are positive. For the defense oracle, PPO optimizes a Bernoulli action distribution for the binary decision of whether to raise an alarm or not.

DO is guaranteed to converge to an MSNE of the game with exact BR oracles [17], but not with approximate BR oracles. We show empirically that our DRL-based approximate BR oracles are general and effective for both the adversary and the defender, so in practice, DO converges to an MSNE in only a few iterations with our oracles. This demonstrates the feasibility and practicality of the proposed approach for real-world traffic monitoring systems.

VI. Experimental Analysis

Network topologies and vehicle data, such as origins, destinations, and counts, are provided by the Transportation Networks for Research Core Team [19]. We used Sioux Falls, SD as the testbed for our approach. Further, we generated two experimental networks using the Grid model with Random Edges (GRE) [24]. GRE stochastically generates a random graph with similar characteristics as a real-world transportation network. For this network, we also generated the edge attributes, e.g., capacity and free flow time, based on the same distribution as Sioux Falls, SD [19]. In all cases, vehicle counts for each origin-destination pair are randomized by $\pm 0.05\%$.

The players' initial strategy sets of the PSRO contain only one policy of No Attack ■ and No Defense ■, where both

(a) 3x2 GRE Graph: 6 Nodes, 16 Edges (b) 5x4 GRE Graph: 20 Nodes, 55 Edges (c) Sioux Falls, SD: 24 Nodes, 76 Edges

Fig. 2: Experimental evaluation and comparison of our approach ■ to alternative strategies. First, we compare various baseline attack strategies ■ to our equilibrium solutions; here, a higher total travel time would indicate a more effective attack (**higher is better**). Second, we compare various defense strategies ■ to our equilibrium solutions; here, a lower total travel time would indicate a more effective defense (**lower is better**). The results show that our approach ■ outperforms all of the alternatives, inducing higher travel times than other attacks and securing lower travel times than other defenses. The results demonstrate that our approach ■ outperforms the alternatives in both roles; crucially, our equilibrium-based defender is robust against these alternative attacks without prior training on them, which demonstrates the success of our algorithm.

players take zero actions and the environment operates under ***nominal*** conditions. In each iteration of PSRO, we first train an adversarial DRL and then a detection DRL.

We used the default hyperparameters from Stable Baselines 3 [25], which are validated by prior work [26], and left random seeds to their default values in NumPy and PyTorch, as our algorithm exhibits low sensitivity to random initialization. Further, to verify the significance of our results, after the Nash equilibrium model is trained, we collect 64 episodic rewards and run permutation statistical tests on them, comparing the means of the distributions.

A. Baselines

On these two network models and the Sioux Falls, SD model, we compare our solution approach to attack baselines such as the greedy attack [5] and the Gaussian [7] attack. We also compare our defense approach with state-of-the-art anomaly detection [27].

1) Attack Baselines: In **Greedy** ■ from Eghtesad *et al.* [5], the adversarial agent counts the number of vehicles s_e^t passing through each edge e as their unperturbed shortest path to their destination at the current time step t. Then, it divides the budget B proportionally to the number of vehicles: $\boldsymbol{a}^t = \frac{\langle s_e^t \colon \forall e \in E \rangle}{\sum_{e \in E} s_e^t} \cdot B$.

In **Gaussian** ■ from Yu *et al.* [7], the attacker divides the environment into multiple subcomponents; we use k-means graph clustering [5] to divide the environment into subcomponents. The attacker then applies a normal Gaussian perturbation to a subcomponent i, dissuading vehicles from passing through i:

$$a_e^t = \begin{cases} 0 & \text{if} \quad e \notin i \\ \sim \mathcal{N}(\hat{B} \cdot c_e, \frac{1}{10} \cdot c_e) & \text{otherwise} \end{cases} : \forall_{e \in E},$$

where $\hat{B}$ is the budget applied to each edge in subcomponent i.

2) Defense Baselines: We adopt an anomaly detection algorithm based on the **Bayesian** ■ process from Laszka *et al.* [27]. We collect nominal (i.e., without attack) baseline experiences x. The observed travel time of each edge $\hat{w}_e^t$ becomes a random variable for $t \in |H|$, where $|H|$ is the length of the observation history for the defender. The mean and covariance are calculated over the trajectories to form a multivariate normal distribution:

$$F_{\hat{w}} \sim \begin{cases} \text{Mean}(\hat{w}_e^t) = \mathbb{E}_x[\hat{w}_e^t] \\ \text{Cov}(\hat{w}_{e^i}^{t_i}, \hat{w}_{e^j}^{t_j}) = \mathbb{E}_x[(\hat{w}_{e^i}^{t_i} - \mathbb{E}_x[\hat{w}_{e^i}^{t_i}]) \\ \qquad \cdot (\hat{w}_{e^j}^{t_j} - \mathbb{E}_x[\hat{w}_{e^j}^{t_j}])] \end{cases} \tag{16}$$

The detection decision is based on comparing the likelihood of a history of observations $\hat{\boldsymbol{w}}_{\boldsymbol{H}}$ according to this distribution with a threshold likelihood τ: $a_d^t = 0$ if $F_{\hat{w}}(\hat{\boldsymbol{w}}_H^t) > \tau$, otherwise $a_d^t = 1$.

B. Numerical Results

To demonstrate the effectiveness of our approach, we need to show that the calculated equilibrium attack and defense policies satisfy Eq. (3). In other words, 1) when the equilibrium attacker is playing against an alternative defense ■ policy, it achieves a higher total travel time, and 2) when the equilibrium

defender is playing against an alternative attack ■ policy, it achieves a lower travel time.

Figure 2 shows the numerical results of our equilibrium strategies against the baseline attacks and defenses. *Nominal* ■ shows the normal total travel time of vehicles without any attacks. Comparison of *Nominal* ■ to our approach shows that our equilibrium defender limits total travel time deviations by 35%, 24%, and 38% against the worst-cast attacker in the 3x2 GRE, 5x4 GRE, and Sioux Falls, SD networks, respectively.

Our equilibrium attack strategy is 19%, 11%, and 22% (episode samples=64, p-value=0.0002) more effective compared to the best (i.e., highest) alternative attack baseline in 3x2 GRE graph, 5x4 GRE graph, and Sioux Falls, SD networks, respectively. Our equilibrium defense strategy is 4%, 34%, and 14% (episode samples=64, p-value=0.0002) more robust compared to the best (i.e., lowest) alternative defense baseline.

No Defense ■ scenario represents the situation where the attacker executes its equilibrium policy (i.e., a worst-case attacker), but there is no detection mechanism. A maximum achievable total travel time exists because of the simulation's finite time horizon.

VII. Conclusion

We address FDI attacks in crowdsourced navigation applications by formulating a strategic zero-sum game solved via policy space response oracles using deep reinforcement learning as best-response oracles. Our approach yields a robust detection mechanism that limits travel time deviations to 34%, outperforming state-of-the-art baselines by 22%. This game-theoretic framework ensures resilient urban mobility against adaptive, worst-case adversarial threats.

Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under Awards No. CNS-1952011, CCF-2403758, and IIS-2214141, Office of Naval Research under Award No. N00014-24-1-2663, Army Research Office under Award No. W911NF-25-1-0059, and by the U.S. Department of Energy (DOE) under Award No. DE-EE0011188. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF, ONR, ARO, and the U.S. DOE. We thank the anonymous reviewers for their feedback on our work and for their suggestions to improve our manuscript.

References

[1] B. Schoon, "Google Maps 'hack' uses 99 smartphones to create virtual traffic jams," https://9to5google.com/2020/02/04/google-maps-hack-virtual-traffic-jam/, 2020.

[2] C. Eryonucu and P. Papadimitratos, "Sybil-based attacks on google maps or how to forge the image of city life," in *15th ACM Conference on Security and Privacy in Wireless and Mobile Networks (ACM WiSec '22)*, 5 2022, pp. 73–84.

[3] Y. Feng, S. Huang, Q. A. Chen, H. X. Liu, and Z. M. Mao, "Vulnerability of traffic control system under cyberattacks with falsified data," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2672, no. 1, pp. 1–11, December 2018.

[4] S. Jamalzadeh, K. Barker, A. D. González, and S. Radhakrishnan, "Protecting infrastructure performance from disinformation attacks," *Scientific Reports*, vol. 12, no. 1, p. 12707, 7 2022.

[5] T. Eghtesad, S. Li, Y. Vorobeychik, and A. Laszka, "Multi-agent reinforcement learning for assessing false-data injection attacks on transportation networks," in *23rd International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '24, 5 2024, pp. 508–515.

[6] Y.-T. Yang, H. Lei, and Q. Zhu, "Strategic information attacks on incentive-compatible navigational recommendations in intelligent transportation systems," *arXiv preprint arXiv:2310.01646*, October 2023.

[7] Y. Yu, A. J. Thorpe, J. Milzman, D. Fridovich-Keil, and U. Topcu, "Sensing resource allocation against data-poisoning attacks in traffic routing," *arXiv preprint arXiv:2404.02876*, April 2024.

[8] M. Waniek, G. Raman, B. AlShebli, J. C.-H. Peng, and T. Rahwan, "Traffic networks are vulnerable to disinformation attacks," *Scientific Reports*, vol. 11, no. 1, p. 5329, March 2021.

[9] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '18)*, 6 2018, pp. 1625–1634.

[10] A. Laszka, B. Potteiger, Y. Vorobeychik, S. Amin, and X. Koutsoukos, "Vulnerability of transportation networks to traffic-signal tampering," in *7th International Conference on Cyber-Physical Systems*, ser. ICCPS 2016, April 2016, pp. 1–10.

[11] S. Raponi, S. Sciancalepore, G. Oligeri, and R. Di Pietro, "Road Traffic Poisoning of Navigation Apps: Threats and Countermeasures," *IEEE Security & Privacy*, vol. 20, no. 3, pp. 71–79, May 2022.

[12] A. Sargolzaei, K. Yazdani, A. Abbaspour, C. D. Crane, and W. E. Dixon, "Detection and Mitigation of False Data Injection Attacks in Networked Control Systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4281–4292, June 2020.

[13] K. Malialis and D. Kudenko, "Distributed response to network intrusions using multiagent reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 270–284, May 2015.

[14] Z. Hu, P. Chen, M. Zhu, and P. Liu, "Reinforcement learning for adaptive cyber defense against zero-day attacks," in *Adversarial and Uncertain Reasoning for Adaptive Cyber Defense*. Springer, 2019, pp. 54–93.

[15] L. Tong, A. Laszka, C. Yan, N. Zhang, and Y. Vorobeychik, "Finding Needles in a Moving Haystack: Prioritizing Alerts with Adversarial Reinforcement Learning," *AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 946–953, 4 2020.

[16] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[17] H. B. McMahan, G. J. Gordon, and A. Blum, "Planning in the presence of cost functions controlled by an adversary," in *20th International Conference on Machine Learning*, ser. ICML '03, 2003, pp. 536–543.

[18] M. Lanctot, V. Zambaldi, A. Gruslys, A. Lazaridou, K. Tuyls, J. Pérolat, D. Silver, and T. Graepel, "A unified game-theoretic approach to multiagent reinforcement learning," in *31st Conference on Neural Information Processing Systems (NeurIPS 2017)*, 2017.

[19] Transportation Networks for Research Core Team, "Transportation Networks for Research," https://github.com/bstabler/TransportationNetworks/, 2020.

[20] United States. Bureau of Public Roads, *Traffic Assignment Manual for Application with a Large, High Speed Computer*, ser. Traffic Assignment Manual for Application with a Large, High Speed Computer. U.S. Department of Commerce, Bureau of Public Roads, Office of Planning, Urban Planning Division, 1964, no. v. 2.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, September 2015.

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, July 2017.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through

deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] W. Peng, G. Dong, K. Yang, and J. Su, "A random road network model and its effects on topological characteristics of mobile delay-tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 12, pp. 2706–2718, 12 2014.

[25] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[26] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," *arXiv preprint arXiv:2006.05990*, June 2020.

[27] A. Laszka, W. Abbas, Y. Vorobeychik, and X. Koutsoukos, "Detection and mitigation of attacks on transportation networks as a multi-stage security game," *Computers & Security*, vol. 87, p. 101576, 11 2019.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and others, "PyTorch: An imperative style, high-performance deep learning library," in *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

Appendix

A. Hardware Configuration

We performed all experiments, including neural network training, on a workstation with 2 AMD EPYC 7763 CPUs, each with 64 cores, 1TB of RAM, and an NVIDIA RTX A5000 GPU with 24GB of VRAM. The CPU is capable of executing 128 concurrent simulations. Since our models are relatively small, we needed only 64GB RAM for simulation and inference and 2GB VRAM for model training.

B. Software Configuration

The implementation of our simulation is an extension of the source code of Eghtesad *et al.* [5]. All of our source code, including our extended simulation and our computational framework, is available under an open-source license. We developed the environments for the attack and defense oracles using Python in the standard format of Farama Gymnasium (formerly known as OpenAI Gym). For the single-agent DRL algorithm, we used Stable Baselines 3 [25], which internally uses PyTorch as its neural operations framework. In addition, we used *scikit-learn* to solve the linear program that finds the equilibrium of the subgame G^i [16].

When training a model, we drew multiple trajectories (or experiences) for each update from simulations running concurrently. We moved these trajectories to the GPU for training. When running the PSRO algorithm, obtaining one experience requires executing the other agent's previously trained policy. We duplicated this policy, loaded it for each instance of the environment into the main RAM, and executed it on the CPU.

Each independent DRL algorithm can be executed for simulation, data collection, and training at 879, 281, and 198 steps per second on average for the 3x2, 5x4, and Sioux Falls networks, respectively. The substantial difference comes from the computational cost of simulating the vehicles' routing decisions and movements.

C. Hyperparameters

We adopted the hyperparameters for the Proximal Policy Optimization (PPO) algorithm, used for both the attack and defense oracles, from the default configurations of the Stable Baselines 3 [25] library, which themselves are based on the well-established defaults from OpenAI Baselines. This decision was informed by the comprehensive large-scale empirical study conducted by Andrychowicz *et. al.* [26]. Their findings provide strong evidence that these default parameters represent a highly competitive and robust baseline, making them a suitable and valid choice for our experiments. We provide a detailed breakdown of these parameters in Table I.

1) Random Seeds: To manage stochasticity, we adhered to the default random number generators provided by Python, NumPy, and PyTorch [28] without setting explicit global seeds. This approach ensures that the inherent randomness in the training process, such as weight initialization and environment interactions, is handled by the standard, well-vetted procedures of these libraries. This allows our results to reflect the general performance of the methodology rather than being tied to a specific, potentially fortunate, random seed.

D. Statistical Tests

To validate the significance of our experimental results, we applied permutation tests to compare the performance of our equilibrium policies with each baseline approach. For each comparison, we collected 64 episodic rewards from both our trained policy and the baseline approach. The permutation test then assessed the null hypothesis that the two sets of rewards were drawn from the same random distribution, i.e., that the difference between our proposed approach and the baseline approach is due to chance only. For this assessment, we had to calculate a p-value, which is the probability of observing a difference as large as the one in our experimental results if the null hypothesis were true. We utilized the robust implementation of this statistical test provided by the SciPy library. This non-parametric approach is particularly well-suited for our analysis since it does not rely on any assumptions about the underlying distribution of the utilities. For every comparison, our test rejected the null hypothesis, demonstrating that the difference between our proposed approach and the baseline approach is statistically significant.

TABLE I: Hyperparameters of the Attack Oracle, Defense Oracle, and the Double Oracle (DO) Algorithm

Component	Hyperparameter	Value	Source File
Attack Oracle Hyperparameters			
Training	Total Training Timesteps	5,000,000	`models/double_oracle/trainer.py`
	Algorithm	Proximal Policy Optimization (PPO)	`models/trainer.py`
	Learning Rate	0.0003 (default)	`stable_baselines3/ppo/ppo.py`
	PPO Number of Steps (n_steps)	50	`models/trainer.py`
	Batch Size	64 (default)	`stable_baselines3/ppo/ppo.py`
	PPO Number of Epochs (n_epochs)	10 (default)	`stable_baselines3/ppo/ppo.py`
	Discount Factor (Gamma)	0.99 (default)	`stable_baselines3/ppo/ppo.py`
	GAE Lambda	0.95 (default)	`stable_baselines3/ppo/ppo.py`
	Clip Range	0.2 (default)	`stable_baselines3/ppo/ppo.py`
	Entropy Coefficient (ent_coef)	0.01	`models/trainer.py`
	Value Function Coefficient (vf_coef)	0.5 (default)	`stable_baselines3/ppo/ppo.py`
	Max Gradient Norm	0.5 (default)	`stable_baselines3/ppo/ppo.py`
	Normalize Advantage	True (default)	`stable_baselines3/ppo/ppo.py`
Neural Network	Policy	MlpPolicy	`models/trainer.py`
	Network Architecture	`dict(pi=[64, 64], vf=[64, 64])` (default)	`stable_baselines3/common/policies.py`
	Activation Function	Tanh (default)	`stable_baselines3/common/policies.py`
Defense Oracle Hyperparameters			
Training	Total Training Timesteps	2,000,000	`models/double_oracle/trainer.py`
	Algorithm	Proximal Policy Optimization (PPO)	`models/double_oracle/trainer.py`
	Learning Rate	0.0003 (default)	`stable_baselines3/ppo/ppo.py`
	PPO Number of Steps (n_steps)	50	`models/double_oracle/trainer.py`
	Batch Size	64 (default)	`stable_baselines3/ppo/ppo.py`
	PPO Number of Epochs (n_epochs)	10 (default)	`stable_baselines3/ppo/ppo.py`
	Discount Factor (Gamma)	0.99 (default)	`stable_baselines3/ppo/ppo.py`
	GAE Lambda	0.95 (default)	`stable_baselines3/ppo/ppo.py`
	Clip Range	0.2 (default)	`stable_baselines3/ppo/ppo.py`
	Entropy Coefficient (ent_coef)	0.01	`models/double_oracle/trainer.py`
	Value Function Coefficient (vf_coef)	0.5 (default)	`stable_baselines3/ppo/ppo.py`
	Max Gradient Norm	0.5 (default)	`stable_baselines3/ppo/ppo.py`
	Normalize Advantage	True (default)	`stable_baselines3/ppo/ppo.py`
Neural Network	Policy	MlpPolicy	`models/double_oracle/trainer/trainer.py`
	Network Architecture	`dict(pi=[64, 64], vf=[64, 64])` (default)	`stable_baselines3/common/policies.py`
	Activation Function	Tanh (default)	`stable_baselines3/common/policies.py`
Double Oracle (DO) Parameters			
DO Parameters	DO Iterations	10	`models/double_oracle/trainer.py`
	Environment Horizon	50	`models/double_oracle/trainer.py`
	Number of Parallel Environments (n_envs)	128	`models/double_oracle/trainer.py`
	Evaluation Episodes	50	`models/trainer.py`
	Post-Training Testing (Numerical Report) Epochs	64	`models/double_oracle/trainer.py`
Environment Hyperparameters			
	Softmin Policy (θ)	1.0	`transport_env/MultiAgentEnv.py`
	K-Means Graph Clustering (n_components)	4	`models/double_oracle/trainer.py`
	False Positive Cost (C_f)	1.0	`transport_env/AdvEnv.py`
	History Size	5	`models/double_oracle/trainer.py`
	GRE generation parameter (p, q)	0.6057, 0.3162	`generate_gre_graph.py`