Safe Networked Robotics With Probabilistic Verification

Sai Shankar Narasimhan, Sharachchandra Bhat, and Sandeep P. Chinchali

***Abstract*—Autonomous robots must utilize rich sensory data to make safe control decisions. To process this data, compute-constrained robots often require assistance from remote computation, or the cloud, that runs compute-intensive deep neural network perception or control models. However, this assistance comes at the cost of a time delay due to network latency, resulting in past observations being used in the cloud to compute the control commands for the present robot state. Such communication delays could potentially lead to the violation of essential safety properties, such as collision avoidance. This article develops methods to ensure the safety of robots operated over communication networks with *stochastic* latency. To do so, we use tools from formal verification to construct a shield, i.e., a run-time monitor, that provides a list of safe actions for any delayed sensory observation, given the expected and maximum network latency. Our shield is minimally intrusive and enables networked robots to satisfy key safety constraints, expressed as temporal logic specifications, with desired probability. We demonstrate our approach on a real F1/10th autonomous vehicle that navigates in indoor environments and transmits rich LiDAR sensory data over congested WiFi links.**

***Index Terms*—Formal methods in robotics and automation, networked robots, teleoperation, probabilistic verification.**

I. Introduction

TODAY, an increasing number of robotic applications require remote assistance, ranging from remote manipulation for surgery [1] to emergency take-over of autonomous vehicles [2]. Teleoperation is even used to control food delivery robots from command centers hundreds of miles away [3]. In these scenarios, network latency is a key concern for safe robot operation since actuation based on delayed state information can lead to unsafe behavior.

Despite the rise of robots operating over communication networks, we lack formal guarantees for their safe operation. Today's approaches for robotic safety range from reachability analysis [4], [5], [6], [7] to shielding that restricts unsafe actions based on a formal safety specification [8], [9], [10], [11]. However, there is little to no research that provides such rigorous safety analysis for networked robotics. This article asks: *How do we ensure safe networked control over wireless networks with stochastic communication delays?*

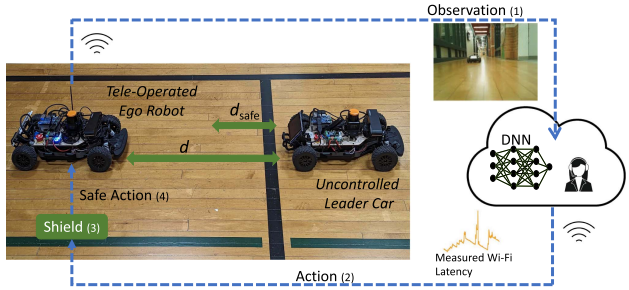


Fig. 1. Safe Networked Control for Robotics: A resource-constrained robot transfers sensor observations (RGB-D images or LiDAR point clouds) through a wireless network with stochastic latency. At the receiving end, a control module or a human teleoperator processes the observation to generate the corresponding action. The action is filtered by the shield, which enforces a particular safety specification that the robot has to maintain. The filtered, "safe" action is then executed by the robot.

Communication delay is the cumulative time taken to send an observation to the cloud and receive an action back at the robot. We develop the intuition that if the interaction between a remotely controlled robot and its environment can be modeled as a Markov Decision Process (MDP), the communication delay is analogous to sensing or actuation delays. Previous works on Networked Control Systems (NCS) have addressed MDPs with delays [12], [13], [14], but often make restrictive assumptions about delay transitions. In our article, we propose *Delayed Communication* MDP, a novel approach to model MDPs with delays that aligns naturally with the transmission of observations and control commands when operating a robot via wireless networks in practice.

Fig. 1 shows our approach, tested on an F1/10th car [15] controlled over a wireless link. Our approach is extremely general - we can either have a human teleoperator or an automatic controller running in the cloud, including Deep Neural Network (DNN) perception models or deep reinforcement learning (RL) based policies. First, sensor observations are transferred via wireless links (step 1) and processed to compute the corresponding control command (step 2). The control command is transmitted back to the robot and filtered by the shield. The shield is a run-time monitor, constructed offline, that disallows actions that violate a safety property. The shield has access to the delay corresponding to the received control command as it runs on the robot. Finally, the shielded action is executed to ensure safe behavior amidst stochastic network latency (steps 3-4).

Manuscript received 7 July 2023; accepted 11 November 2023. Date of publication 7 December 2023; date of current version 14 February 2024. This letter was recommended for publication by Associate Editor D. Brscic and Editor A. Peer upon evaluation of the reviewers' comments. This work was supported in part by Lockheed Martin Corporation, ONR Award N00014-21-1-2379, in part by the National Science Foundation under Grant 2148186, and in part by Federal Agencies and Industry Partners as specified in the Resilient and Intelligent NextG Systems (RINGS) Program. *(Sai Shankar Narasimhan and Sharachchandra Bhat contributed equally to this work.) (Corresponding author: Sai Shankar Narasimhan.)*

The authors are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: nsaishankar@utexas.edu; sharachchandra@utexas.edu; sandeepc@utexas.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3340525, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3340525

We design the shield using tools from formal verification [16], given knowledge of the network latency and a model of the environment transitions.

Shields, as implemented in [8], provide an absolute measure of safety. For networked control with stochastic latency, shielding results in perfectly safe operation at the cost of task efficiency. In this article, we propose a shield synthesis approach that, when combined with the cloud controller, allows the networked control system to meet safety requirements with a desired probability. Our experimental findings indicate that a slight reduction in the desired safety probability leads to a significant increase in task efficiency. In this article,

1) We present the *Delayed Communication* MDP, a novel approach that accurately models the interaction between a remotely controlled robot and the environment, in the presence of stochastic network latency.
2) We propose an algorithm to synthesize a shield that, when executed with the cloud controller, guarantees the desired probability of satisfying a safety property.
3) We demonstrate our approach in simulation as well as on an F1/10th autonomous vehicle that must closely (and safely) follow an unpredictable leader in indoor environments over congested wireless networks (Fig. 1).

II. Related Work

We now survey how our work relates to cloud robotics, networked control systems, shielding, and formal methods.

Cloud Robotics: Cloud robotics [17], [18] studies how resource-constrained robots can offload inference [19], [20], mapping, and control to remote servers [21]. Recent work ([22]) circumvents network latency for teleoperation by predicting the intent of a teleoperator remotely and synthesizing trajectories locally on the robot for handwriting imitation. This approach does not scale well for resource-constrained robots for more complicated tasks like autonomous driving as it involves running DNNs for intent prediction.

Delayed MDPs: Numerous prior works have addressed sensing and actuation delays in MDPs [12], [13], [14], [23], by making restrictive assumptions about the delay transitions. The delay is constant in [12] and [14], while it can only increase or remain constant between consecutive time steps in [13], halting the decision-making when the delay reaches the maximum limit. In contrast, we make no such assumptions. Additionally, prior works have focused on the delay in feedback [23] or cost collection ([13], [14]) in the RL setting. Our objective is to *formally verify the safety of NCS*.

Shielding and Safe Reinforcement Learning: Our work builds upon safe RL techniques developed for discrete-time systems. The shielding approach ([8], [11]) involves synthesizing a runtime monitor that overwrites the agent's action if it violates the desired safety specification, aiding in safe exploration [9]. Recent work relaxes assumptions on the knowledge of the environment and makes the shielding approach more practical [24], [25]. As the execution of perfectly safe policies restricts exploration in RL, probabilistic shields were introduced in [10] and [26] to trade off safety for exploration. Building on this, recent work implements probabilistic shields through probabilistic logic programs [27]. The definitions of the probabilistic shield in [10] and [26] are similar to ours but these works do not provide theoretical guarantees for safety, which we do. Another probabilistic shielding approach [28] focuses specifically on synthesizing shields that satisfy *bounded* specifications. While the above-mentioned works deal with shielding for safe RL, our work focuses on developing a novel shielding approach for NCS.

The previous approaches deal with finite state models obtained from an abstraction of the continuous state space. [29] proposes an alternative approach using Robust Model Predictive Shielding. Further, Hamilton-Jacobi reachability analysis [4], [5] and Control Barrier Function methods [6], [7] formulate the safe control problem for the continuous system. These methods cannot express rich safety properties, such as "maintain a minimum distance between two vehicles when the delay is above a threshold and visit the landmark before reaching the goal", which is possible in our approach.

III. Background

A *Markov Decision Process* (MDP) is a tuple $\langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathbb{A}, \mathbb{P} \rangle$, where S is a finite state set, Init is a probability distribution over S representing the initial state distribution and Act is a finite set of actions. The transition probability function $\mathbb{P} : \mathrm{S} \times \mathrm{Act} \times \mathrm{S} \to [0, 1]$ is a conditional probability distribution and hence satisfies $\sum_{s' \in \mathrm{S}} \mathbb{P}(s' \mid s, a) = 1$ for every state-action pair $(s, a) \in \mathrm{S} \times \mathbb{A}(s)$, where $\mathbb{A}(s) = \{a \in \mathrm{Act} \mid \exists\ s' \in \mathrm{S}\ s.t.\ \mathbb{P}(s' \mid s, a) \neq 0\}$ is the set of available actions for the state s. A policy π is defined as a mapping from states to actions, $\pi : \mathrm{S} \to \mathrm{Act}$.

We will now introduce safety properties and our approach using the hardware setup in Fig. 1, where a resource-constrained mobile robot must safely follow an unpredictable leader while being controlled remotely over a wireless link with stochastic network delays. Henceforth, we will use the term *agent* to indicate any controlled entity like the robot and *environment* for uncontrolled entities (like the leader car). The agent and the environment together form the *system*.

To capture our desired notion of safety for the system, we first define $\mathrm{S}_{\mathrm{unsafe}}$ to be the set of all *unsafe states*. For example, in our hardware setup, an unsafe state is one where the distance between the two cars, d, is less than the safety threshold d_{safe} (Fig. 1). Then, we define the system to be safe if it never reaches any state in $\mathrm{S}_{\mathrm{unsafe}}$. This can be encapsulated by the Linear Temporal Logic (LTL) [16] safety specification $\square\neg\mathrm{S}_{\mathrm{unsafe}}$, which translates to "always ($\square$) never ($\neg$) be in an unsafe state". Note that our notion of safety is now equivalent to determining the probability with which the system *satisfies the safety specification* $\varphi = \square\neg\mathrm{S}_{\mathrm{unsafe}}$, which can be done efficiently. We use $\mathrm{V}^{\pi}_{\mathcal{M},\varphi}(s)$ to represent the probability of satisfying φ, while executing the policy π starting from the state $s \in \mathrm{S}$. The probability with which the system satisfies φ is then given by the expectation of $\mathrm{V}^{\pi}_{\mathcal{M},\varphi}(s)$ over Init. To compute $\mathrm{V}^{\pi}_{\mathcal{M},\varphi}$, we note that the safety specification $\varphi = \square\neg\mathrm{S}_{\mathrm{unsafe}}$ can be cast into a reachability specification $\theta = \diamond\mathrm{S}_{\mathrm{unsafe}}$, which refers to "eventually ($\diamond$) reach any unsafe state". Now, the probability of satisfying this reachability specification, $\mathrm{V}^{\pi}_{\mathcal{M},\theta}(s)$, is the unique solution to the following system of equations [16]:

$$\text{if } s \in \mathrm{S}_{\mathrm{unsafe}} \Rightarrow \mathrm{V}^{\pi}_{\mathcal{M},\theta}(s) = 1; \text{if } s \not\models \theta \Rightarrow \mathrm{V}^{\pi}_{\mathcal{M},\theta}(s) = 0,$$
$$\text{else } \mathrm{V}^{\pi}_{\mathcal{M},\theta}(s) = \mathbb{E}_{s' \sim \mathbb{P}(s' \mid s, \pi(s))}\left[\mathrm{V}^{\pi}_{\mathcal{M},\theta}(s')\right]. \tag{1}$$

This can be solved using value iteration. Then, the safety probabilities can be computed using the relation $\mathrm{V}^{\pi}_{\mathcal{M},\varphi}(s) = 1 - \mathrm{V}^{\pi}_{\mathcal{M},\theta}(s)\ \forall s \in \mathrm{S}$. We denote the minimum and maximum safety probabilities, across any policy, as $\mathrm{V}^{\min}_{\mathcal{M},\varphi}(s)$ and $\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$ respectively. We refer the readers to [16] for details on how

they can be computed. We also denote the minimum and maximum safety probabilities for a *state-action pair* $(s, a) \in \mathrm{S} \times \mathbb{A}(s)$ by $\mathrm{Q}^{\min}_{\mathcal{M},\varphi}(s, a)$ and $\mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a)$ respectively. For example, $\mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a)$ is computed as

$$\mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a) = \mathbb{E}_{s' \sim \mathbb{P}(s'|s,a)} \left[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s') \right]. \tag{2}$$

The above discussion can be extended to reach-avoid specification, $\neg\, \mathrm{S}_{\text{unsafe}}\ \cup$ goal, which translates to "never ($\neg$) be in an unsafe state until ($\cup$) the goal state (goal) is reached". We note that the reach-avoid specifications can also be cast into a reachability specification, and refer the readers to [16] for further details. We denote the policy corresponding to the maximum safety probability, $\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$, as the optimally safe policy $\pi^{\text{safe}}_{\mathcal{M},\varphi}$, defined as $\pi^{\text{safe}}_{\mathcal{M},\varphi}(s) = \operatorname{argmax}_a \mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a)$. These quantities are necessary to define our *shield* that can ensure a desired safety probability δ for the networked controlled system. For an MDP $\mathcal{M} = \langle \mathrm{S}, \text{Init}, \text{Act}, \mathbb{A}, \mathbb{P} \rangle$, a shield ([8], [10]) is a function, $\mathrm{C} : \mathrm{S} \to 2^{\text{Act}}$, that maps every state $s \in \mathrm{S}$ to a subset of $\mathbb{A}(s)$. During runtime, the shield overwrites the policy only if $\pi(s) \notin \mathrm{C}(s)$.

IV. Problem Formulation

In this section, we formally define our safe networked control problem. We make the following three key assumptions:

- The agent-environment interaction is available as an MDP $\mathcal{M}_b = \langle \mathrm{S}, \text{Init}, \text{Act}, \mathbb{A}, \mathbb{P} \rangle$, where the state and action sets are discrete and finite. For the continuous case, we obtain finite sets by abstracting the continuous state and action spaces. We term this as the *Basic* MDP. In our hardware setup, the state set S consists of bins of possible distances between the cars, the action set Act consists of bins of allowed ego-robot velocities and the transition probability function $\mathbb{P}$ captures the leader's unpredictability modeled using an assumed range of velocities. This is a standard assumption since the offline computation of safe control policies typically require knowledge of the agent-environment interaction [4], [10], [26].
- We assume a sufficient understanding of the stochasticity in communication delay, which we model as a transition probability function $\mathbb{P}_\tau$ with an upper bound $\tau_{\max}$ on the delay. Later, in Section VI, we show how to obtain $\mathbb{P}_\tau$ from the collected time-series datasets of communication delays.
- Finally, we assume the cloud controller π_{cloud} is available as a mapping from the state set S to the action set Act for the *Basic* MDP. For the discrete case, this mapping is trivial as it is π_{cloud} itself. For the continuous case, the mapping can be easily obtained even for complex DNN controllers [30]. Later, we show how to relax this assumption for cases like human-teleoperation. Note that the cloud controller π_{cloud} is unaware of the communication delay.

We now explain the practical effects of delays on NCS. Consider an agent sending timestamped observations to the cloud. The cloud processes these observations to extract the system state information, generates a corresponding action, and appends the same timestamp to it before sending it back to the agent. We define communication delay as the time difference between the current time and the timestamp of the received action. Formally, at time t, the communication delay is τ_t if the received action, a_t, corresponds to the delayed state $s_{t-\tau_t}$. We refer to $s_{t-\tau_t}$ as the latest available system state at time t. Between two consecutive time steps t and $t+1$, only one of the following three events can occur.

- *Case 1:* The agent receives no action from the cloud. This implies that the latest available system state at $t+1$ is still $s_{t-\tau_t}$ and the delay $\tau_{t+1} = \tau_t + 1$.
- *Case 2:* The agent receives an action with a timestamp equal to the current time, implying no delay, i.e., $\tau_{t+1} = 0$.
- *Case 3:* The agent receives an action with an older timestamp, implying $\tau_{t+1} > 0$ and $\tau_{t+1} \leq \tau_t$.

To model these events, we represent the delay transitions as a conditional probability distribution $\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t)$, $\mathbb{P}_\tau : \Omega \times \Omega \to [0, 1]$ where $\Omega = \{0, 1, .., \tau_{\max}\}$ is the set of integer delay values. As the delay cannot increase by more than 1 (*Case 1*), we have $\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t) = 0$ if $\tau_{t+1} > \tau_t + 1$.

Problem: We are given the *Basic* MDP $\mathcal{M}_b$, the delay transition probability function $\mathbb{P}_\tau$ with an upper bound on delay $\tau_{\max}$ and the cloud controller π_{cloud}. Our aim is to ensure safe networked control such that the system satisfies the safety specification $\square\neg\mathrm{S}_{\text{unsafe}}$ with probability δ, where $\mathrm{S}_{\text{unsafe}}$ denotes the unsafe states.

V. Approach

Our approach to safe networked control is based on shield construction. The shield construction for safe networked control requires an MDP that is cognizant of the delay. However, the *Basic* MDP $\mathcal{M}_b$ does not account for any delay. Therefore, from $\mathcal{M}_b$, we first create a *Delayed Communication* MDP (DC $-$ MDP) that accounts for the stochastic communication delay. This is outlined in Section V-A. Consequently, in Section V-B, we describe our approach for shield construction for any MDP and any safety specification.

A. Delayed Communication Markov Decision Processes

To design the DC $-$ MDP from the *Basic* MDP $\mathcal{M}_b$, we first note that in the presence of delay, the state transition model can no longer rely only on the current state s_t and the current executed action a_t to determine the next state s_{t+1}. This is because s_t is not known when the delay is not zero. For a system with delay τ_t at time t, the maximum information available about the system at t is the latest observed system state $s_{t-\tau_t}$ and the action buffer, i.e., sequence of actions executed from $t - \tau_t$ to $t-1$, $a_{t-\tau_t}, \ldots, a_{t-1}$. Therefore, determining whether an action a_t is safe with respect to the property $\square\neg\mathrm{S}_{\text{unsafe}}$ should intuitively rely on $s_{t-\tau_t}$ and $a_{t-\tau_t}, \ldots, a_{t-1}$. Hence, we incorporate this maximum information available about the system into the state of the DC $-$ MDP. Note that the action buffer's length is the delay τ_t, which can vary. So, we introduce $\tau_{\max} - \tau_t$ number of place-holder actions, ϕ, to ensure the action buffer's length is always $\tau_{\max}$. Now, we define the state at time t, x_t, as $(s_{t-\tau_t}, (a_{t-\tau_t}, \ldots, a_{t-1}, \phi, \ldots, \phi), \tau_t)$ and the state space for the DC $-$ MDP as $\mathrm{X}_d \in \mathrm{S} \times (\text{Act} \cup \{\phi\})^{\tau_{\max}} \times \Omega$, where $\Omega = \{0, 1, \ldots, \tau_{\max}\}$ is the set of all possible delays. Without loss of generality, the initial delay, τ_0 is 0, i.e., the latest available system state at the beginning of any task execution is the initial system state. Let s_{Init} be the set of initial states for $\mathcal{M}_b$, then we define the initial state probability distribution of the DC $-$ MDP, Init_d, to only have non-zero probabilities for the states in the list $s_{\text{Init}} \times \{(\phi, \phi, \ldots, \phi)\} \times \{0\}$.

Let the state $x_t = (s_{t-\tau_t}, (a_{t-\tau_t}, \ldots, a_{t-1}, \phi, \ldots, \phi), \tau_t)$ and the action a_t. We now relate the three possible events described in Section IV to the state transitions in X_d.

- *Case 1.* $\tau_{t+1} = \tau_t + 1$*:* The latest available system state remains the same. Thus x_{t+1} is $(s_{t-\tau_t}, (a_{t-\tau_t}, \ldots, a_t, \phi, \ldots, \phi), \tau_t + 1)$. The occurrence of this event is governed only by the delay transition, hence the probability of this event is $\mathbb{P}_\tau(\tau_t + 1 \mid \tau_t)$.
- *Case 2.* $\tau_{t+1} = 0$*:* The latest available system state is the current system state s_{t+1}. Thus x_{t+1} is $(s_{t+1}, (\phi, \ldots, \phi), 0)$. This event depends on the delay transition with probability $\mathbb{P}_\tau(0 \mid \tau_t)$ and the system transition from $s_{t-\tau_t}$ to s_{t+1} by executing $\tau_t + 1$ actions $a_{t-\tau_t}, \ldots, a_t$, with probability $\mathbb{P}(s_{t+1} \mid s_{t-\tau_t}, a_{t-\tau_t}, \ldots, a_t)$.
- *Case 3.* $0 < \tau_{t+1} \leq \tau_t$*:* The latest available system state is delayed by τ_{t+1}. Thus x_{t+1} is $(s_{t+1-\tau_{t+1}}, (a_{t+1-\tau_{t+1}}, \ldots, a_t, \phi, \ldots, \phi), \tau_{t+1})$. Similar to *Case 2*, the occurrence of this event is governed by the delay transition with probability $\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t)$ and the system transition with probability $\mathbb{P}(s_{t+1-\tau_{t+1}} \mid s_{t-\tau_t}, a_{t-\tau_t}, \ldots, a_{t-\tau_{t+1}})$.

$$
\mathbb{P}_d(x_{t+1} \mid x_t, a_t) =
\begin{cases}
\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t), \\
\text{if } \tau_{t+1} = \tau_t + 1 \\
x_{t+1} = (s_{t-\tau_t}, (a_{t-\tau_t}, \ldots, a_t, \phi, \ldots, \phi), \tau_t + 1) \\
\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t) \underbrace{\sum_{s_{t-\tau_t+1} \in \mathrm{S}} y_{t-\tau_t} \cdots \sum_{s_t \in \mathrm{S}} y_{t-1}\, y_t}_{\tau_t + 1 \text{ terms}}, \\
\text{if } \tau_{t+1} = 0, x_{t+1} = (s_{t+1}, (\phi, \ldots, \phi), 0) \\
\mathbb{P}_\tau(\tau_{t+1} \mid \tau_t) \underbrace{\sum_{s_{t-\tau_t+1} \in \mathrm{S}} y_{t-\tau_t} \cdots \sum_{s_{t-\tau_{t+1}+1} \in \mathrm{S}} y_{t-\tau_{t+1}-1}\, y_{t-\tau_{t+1}}}_{\tau_t - \tau_{t+1} + 1 \text{ terms}}, \\
\text{if } 0 < \tau_{t+1} \leq \tau_t \\
x_{t+1} = (s_{t+1-\tau_{t+1}}, (a_{t+1-\tau_{t+1}}, \ldots, a_t, \phi, \ldots, \phi), \tau_{t+1}) \\
0 \quad \text{otherwise},
\end{cases}
\tag{3}
$$

Consequently, we define the transition probability function for the DC − MDP $\mathbb{P}_d : \mathrm{X}_d \times \mathrm{Act} \times \mathrm{X}_d \to [0, 1]$ as shown in (3). In (3), $y_{t-\tau_t} = \mathbb{P}(s_{t-\tau_t+1} \mid s_{t-\tau_t}, a_{t-\tau_t})$ is the one-step transition probability from the system state $s_{t-\tau_t}$ to $s_{t-\tau_t+1}$ while executing the action $a_{t-\tau_t}$. We note that the system transition probabilities in *Case 2* and *Case 3* can be factorized into the $\tau_t + 1$ and $\tau_t - \tau_{t+1} + 1$ terms in (3) respectively. Now, we prove that $\mathbb{P}_d$ is a valid conditional probability distribution with support over the state space X_d. First we note that since the conditional distributions $\mathbb{P}_\tau, y_t \geq 0$, $\mathbb{P}_d \geq 0$. Next, we show that $\sum_{x_{t+1} \in \mathrm{X}_d} \mathbb{P}_d(x_{t+1} \mid x_t, a_t) = 1$. Substituting the transition probabilities from *Cases 2,3* in place of the factorized terms in (3),

$$
\begin{aligned}
&\sum_{x_{t+1} \in \mathrm{X}_d} \mathbb{P}_d(x_{t+1} \mid x_t, a_t) = \mathbb{P}_\tau(\tau_t + 1 \mid \tau_t) \\
&+ \mathbb{P}_\tau(0 \mid \tau_t) \sum_{s_{t+1}} \mathbb{P}(s_{t+1} \mid s_{t-\tau_t}, a_{t-\tau_t}, \ldots, a_t) \\
&+ \sum_{\tau'=1}^{\tau_t} \mathbb{P}_\tau(\tau' \mid \tau_t) \sum_{s_{t-\tau'}} \mathbb{P}(s_{t-\tau'} \mid s_{t-\tau_t}, a_{t-\tau_t}, \ldots, a_{t-\tau'-1}).
\end{aligned}
\tag{4}
$$

The inner summations in the second and third terms of the right-hand side of (4) equate to 1. Hence, $\sum_{x_{t+1} \in \mathrm{X}_d} \mathbb{P}_d(x_{t+1} \mid x_t, a_t) = \sum_{\tau' \in \Omega} \mathbb{P}_\tau(\tau' \mid \tau) = 1$.

Thus, the DC − MDP $\mathcal{M}_d$ is the tuple $\langle \mathrm{X}_d, \mathrm{Init}_d, \mathrm{Act}, \mathbb{A}, \mathbb{P}_d \rangle$. From the definition of the state space of the DC − MDP, we denote the unsafe states for the DC − MDP, $\mathrm{X}_{\mathrm{unsafe}}$, as a subset of $\mathrm{S}_{\mathrm{unsafe}} \times (\mathrm{Act} \cup \{\phi\})^{\tau_{\max}} \times \Omega$, where $\mathrm{S}_{\mathrm{unsafe}}$ is the unsafe states set for the *Basic* MDP. In other words, the DC − MDP state at time t, $x_t = (s_{t-\tau_t}, (a_{t-\tau_t}, \ldots, a_{t-1}, \phi, \ldots, \phi), \tau_t)$, is unsafe if $s_{t-\tau_t} \in \mathrm{S}_{\mathrm{unsafe}}$. Additionally, we show how to construct the DC − MDP when only $\tau_{\max}$ is known, and $\mathbb{P}_\tau$ is not. Since the delay is upper-bounded by $\tau_{\max}$, the action corresponding to the observation $s_{t-\tau_{\max}}$ is always available at timestep t. Therefore, we take $s_{t-\tau_{\max}}$ as the latest available system state and consider the delay to be a constant and equal to $\tau_{\max}$. Consequently, the initial delay is set to $\tau_{\max}$ and the action buffer is set to $\{(a_s, a_s, \ldots, a_s)\}$, where a_s is the action that does not affect the agent's state. In our hardware setup, a_s is the ego-velocity of 0 m/s.

B. Shield Design for Safe Networked Control

In this section, we show how to construct a shield for any MDP $\mathcal{M} = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathbb{A}, \mathbb{P} \rangle$, a specification $\varphi = \square \neg \mathrm{S}_{\mathrm{unsafe}}$ and a policy π. The shield should ensure that when π is executed in the presence of the shield, the initial state distribution should satisfy φ with at least the desired safety probability δ. First, we formally define the shield.

Definition 1: The ϵ-shield, $\mathrm{C}_\epsilon : \mathrm{S} \to 2^{\mathrm{Act}}$, for any state $s \in \mathrm{S}$, and $\epsilon \in [0, 1]$ is

$$
\mathrm{C}_\epsilon(s) = \begin{cases} \{a \mid \mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a) \geq \epsilon\} & \text{if } \mathrm{V}^{\max}_{\mathcal{M},\varphi}(s) \geq \epsilon, \\ \{\mathrm{argmax}_a \mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s, a)\} & \text{if } \mathrm{V}^{\max}_{\mathcal{M},\varphi}(s) < \epsilon. \end{cases} \tag{5}
$$

During run-time, the action executed is different from $\pi(s)$ only if $\pi(s) \notin \mathrm{C}_\epsilon(s)$; in which case an action from $\mathrm{C}_\epsilon(s)$ is chosen. Hence, the ϵ-shield is *minimally intrusive*. Now, we show there exists ϵ such that the run-time monitoring of π by the ϵ-shield C_ϵ provides a safety probability greater than or equal to δ for the initial states.

Definition 2: The modified policy, $\pi_\epsilon : \mathrm{S} \to \mathrm{Act}$, for any state $s \in \mathrm{S}$, policy π, and ϵ-shield C_ϵ is

$$
\pi_\epsilon(s) = \begin{cases} \pi(s) & \text{if } \pi(s) \in \mathrm{C}_\epsilon(s), \\ \text{pick from } \mathrm{C}_\epsilon(s) & \text{if } \pi(s) \notin \mathrm{C}_\epsilon(s). \end{cases} \tag{6}
$$

Observe that the *modified policy* π_ϵ is a result of the run-time monitoring of π by the ϵ-shield C_ϵ. In other words, π_ϵ is the policy that is executed during networked control.

Proposition 1: The safety probability for a state $s \in \mathrm{S}$ while executing π_ϵ, $\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)$, is lower bounded by $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)$ where the MDP $\mathcal{M}_\epsilon = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathrm{C}_\epsilon, \mathbb{P} \rangle$. The lower bound $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)$ is a non-decreasing function of ϵ.

Proof: First, we note that executing π_ϵ for $\mathcal{M}$ is equivalent to executing π for $\mathcal{M}_\epsilon = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathrm{C}_\epsilon, \mathbb{P} \rangle$, where the allowed action set for each state s is given by $\mathrm{C}_\epsilon(s)$. Hence, the minimum safety probability for $\mathcal{M}_\epsilon$ denoted by $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)$ is the lower bound for $\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)$. Now, we show by contradiction that for $\epsilon, \bar{\epsilon} \in [0, 1]$ and $\epsilon < \bar{\epsilon}$, $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s) \leq \mathrm{V}^{\min}_{\mathcal{M}_{\bar{\epsilon}},\varphi}(s)$ for any state $s \in \mathrm{S}$. Assume for the two MDPs, $\mathcal{M}_\epsilon$ and $\mathcal{M}_{\bar{\epsilon}}$, $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s) > \mathrm{V}^{\min}_{\mathcal{M}_{\bar{\epsilon}},\varphi}(s)$ for some state $s \in \mathrm{S}$. This implies that

Algorithm 1: Shield Design.

Input: MDP $\mathcal{M} = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathbb{A}, \mathbb{P} \rangle$, policy π, safety specification $\varphi = \Box \neg \mathrm{S}_{\mathrm{unsafe}}$, desired safety probability δ
Output: ϵ-shield, C^*_ϵ.
1: Initialize ϵ-shield, $\mathrm{C}^*_\epsilon(s) = \mathbb{A}(s) \ \ \forall s \in \mathrm{S}$.
2: Compute $\mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s,a)$ for all state-action pairs in $\mathcal{M}$.
3: **for** $\epsilon \leftarrow [0, \eta, 2\eta, \ldots\ 1]$ **do**
4: Determine $\mathrm{C}_\epsilon(s)$ for each state $s \in \mathrm{S}$ as in (5)
5: Determine the *modified policy* π_ϵ as in (6).
6: Compute $\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)$ for all states in S as in Section III.
7: **if** $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)] \geq \delta$ **then** $\mathrm{C}^*_\epsilon = \mathrm{C}_\epsilon$
8: **break**
9: **end if**
10: **end for**
11: **return** C^*_ϵ.

the policy that corresponds to $\mathrm{V}^{\min}_{\mathcal{M}_{\bar\epsilon},\varphi}(s)$ does not exist for $\mathcal{M}_\epsilon$, and hence $\mathrm{C}_\epsilon(s') \subset \mathrm{C}_{\bar\epsilon}(s')$ for some $s' \in \mathrm{S}$. But, from the definition of ϵ-shield, if $\epsilon < \bar\epsilon$, then $\mathrm{C}_\epsilon(s) \supseteq \mathrm{C}_{\bar\epsilon}(s) \ \forall s \in \mathrm{S}$, which is a contradiction. Thus, the lower bound on $\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)$ is a *non-decreasing* function of ϵ. ■

Remark 1: For the MDP $\mathcal{M} = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathbb{A}, \mathbb{P} \rangle$ and the safety property φ, note that the safety probability for any $s \in \mathrm{S}$ is upper-bounded by $\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$. So, for the initial state distribution, the upper bound on the safety probability is $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)]$. Hence, any choice of the desired safety probability δ should satisfy $\delta \leq \mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)]$.

The Algorithm 1 takes as input the MDP $\mathcal{M}$, policy π, specification φ, and desired safety probability δ, and outputs the synthesized ϵ-shield C^*_ϵ. In line 2, we compute the maximum safety probability for all state-action pairs in $\mathcal{M}$, as explained in Section III, (2). Then, we gradually (based on the granularity η) vary the parameter ϵ from 0 to 1 until the safety probability for the initial state distribution, while executing the *modified policy* π_ϵ, is greater than or equal to the desired safety probability δ (lines 3-10).

Theorem 1. (Termination with guaranteed safety): For a given MDP $\mathcal{M}$ and a policy π, Algorithm 1 always terminates with an ϵ-shield, C^*_ϵ as in (5), such that the modified policy π_ϵ, a combination of π and C^*_ϵ ((6)), satisfies the safety property $\varphi = \Box \neg \mathrm{S}_{\mathrm{unsafe}}$ for the initial state distribution with a probability greater than or equal to the desired safety probability δ, where $\delta \leq \mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)]$.

Proof: From Proposition 1, we know that $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)$ is a *non-decreasing* function of $\epsilon \ \forall s \in \mathrm{S}$. For $\epsilon = 0$, note that $\mathrm{C}_\epsilon(s) = \mathbb{A}(s)$, and therefore $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s) = \mathrm{V}^{\min}_{\mathcal{M},\varphi}(s)$. Moreover, for $\epsilon = 1$, note that $\mathrm{C}_\epsilon(s) = \{\mathrm{argmax}_a \mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s,a)\}$ from (5). This implies π_ϵ is the same as the optimally safe policy, $\pi^{\mathrm{safe}}_{\mathcal{M},\varphi}$, from Section III. Consequently, we have $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s) = \mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$. To summarize, $\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)$ is a non-decreasing function of ϵ that lies between $\mathrm{V}^{\min}_{\mathcal{M},\varphi}(s)$ and $\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$.

Since expectation is a linear operation, $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)]$ is also a non-decreasing function of ϵ that lies between $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\min}_{\mathcal{M},\varphi}(s)]$ and $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)]$. Therefore, for any desired safety probability $\delta \leq \mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)]$ (from Remark 1), there exists an $\epsilon \in [0,1]$ such that $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)] \geq \delta$. Finally, since $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\pi_\epsilon}_{\mathcal{M},\varphi}(s)]$ is lower bounded by $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)]$ (Proposition 1), we conclude that the Algorithm 1 always terminates with the ϵ-shield, C^*_ϵ, that guarantees the desired safety probaility δ. ■

We note that irrespective of the choice to pick any action from $\mathrm{C}_\epsilon(s)$ when $\pi(s) \notin \mathrm{C}_\epsilon(s)$ ((6)), Algorithm 1 returns an ϵ-shield, C^*_ϵ, which guarantees the desired safety probability. For example, one could select actions from $\mathrm{C}_\epsilon(s)$ prioritizing either task-efficiency or safety ($\mathrm{argmax}_a \mathrm{Q}^{\max}_{\mathcal{M},\varphi}(s,a)$). We also note that the shield design in [10] and [26] is similar to our ϵ-shield definition. However, our key novelty is that unlike [10] and [26] which do not provide any guarantee on achieving the required safety probability, our approach (Algorithm 1) returns C^*_ϵ which guarantees the required safety probability.

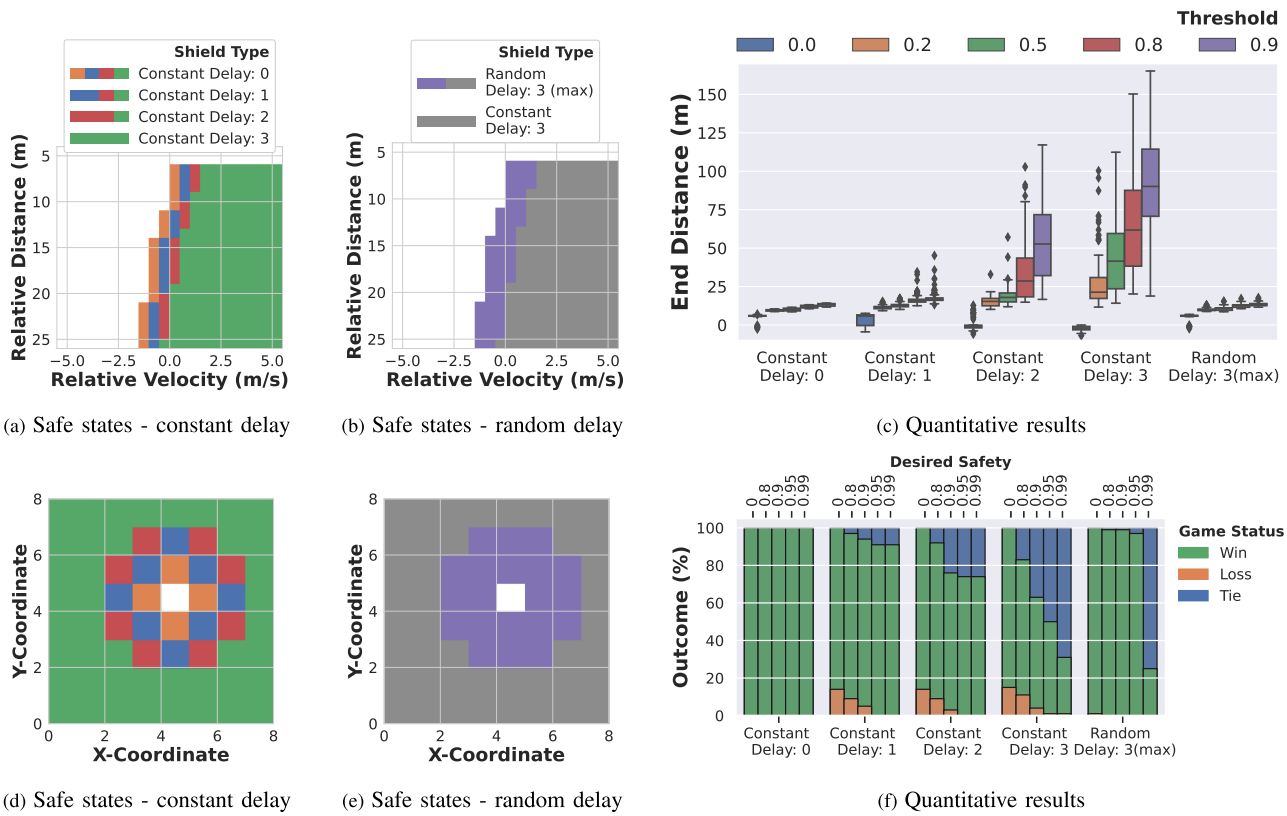
Remark 2: Algorithm 1 can be modified to yield an ϵ-shield even when π is not known, in cases like human-teleoperation. Since the modified policy cannot be computed without π, we instead check for $\mathbb{E}_{s\sim \mathrm{Init}}[\mathrm{V}^{\min}_{\mathcal{M}_\epsilon,\varphi}(s)] \geq \delta$ in line 7 of the Algorithm 1. This guarantees safety probability of at least δ for any modified policy π_ϵ.

Hence, for safe networked control, we construct the DC − MDP (refer to Section V-A), and given the safety specification $\varphi = \Box \neg \mathrm{S}_{\mathrm{unsafe}}$ and the cloud controller π_{cloud}, we use Algorithm 1 to construct the ϵ-shield, C^*_ϵ, which ensures a safety probability greater than or equal to δ for the networked control system. The same can be extended to reach-avoid specifications by casting them into reachability specifications (refer to [16]). More broadly, our approach works for any specification that can be cast into a reachability specification.

VI. Experiments

Now, we show empirically that the shield ensures safety in the presence of communication delays. We analyze the behavior of the agent with shields constructed using two different DC − MDPs: "constant delay" when only $\tau_{\max}$ is known, and "random delay" when $\mathbb{P}_\tau$ is modeled additionally. We test on three environments,

- A 2D 8×8 gridworld simulation setup where the controlled robot, initialized at (0,0), is tasked with reaching the goal at (7,7) while avoiding collision with a dynamic obstacle. Each episode runs for 50 timesteps. An episode is considered a *win* if the robot reaches the goal without colliding, a *loss* if there is a collision or a *draw* otherwise. The cloud controller is learned using tabular Q-learning.
- A car-following simulation setup where the ego robot has to follow the leader car with a minimum safety distance of 5 m. The system state consists of relative distance and relative velocity. The leader car can accelerate anywhere between -0.2 m/s ^{2} and 0.2 m/s^2, and the ego robot can accelerate between -0.5 m/s^2 and 0.5 m/s^2. Each episode runs for 100 s. The cloud controller is a pre-trained RL agent that maximizes distance traveled and minimizes collisions with the leader. We discretize the relative distance and relative velocity to obtain a finite state space.
- The hardware setup (Fig. 1) with two F1/10th cars [15]. The ego robot is equipped with a laser rangefinder. The generated point cloud is transmitted over WiFi to a remote server (the cloud). Here the state is estimated and a time-optimal control command is sent back over WiFi to the robot. The robot has to follow the leader as quickly as possible while maintaining a safe distance of at least 0.2 m.



(a) Safe states - constant delay (b) Safe states - random delay (c) Quantitative results

(d) Safe states - constant delay (e) Safe states - random delay (f) Quantitative results

Fig. 2. Shielding leads to safe networked control in simulations. Top row - car following simulation results; Bottom row - gridworld simulation results. (a) and (d) Show the set of safe initial states with maximum safety probability greater than 0.95 for the *Delayed Communication* MDP for the constant delay case. The set of safe states expands as the maximum delay ($\tau_{\max}$) decreases. This is depicted by the legend that has multiple colors attributed to lower latencies. (b) and (e) Compares the set of safe initial states with maximum safety probability greater than 0.95 for the random and constant delay cases with $\tau_{\max} = 3$ in both the cases. The set of safe states is larger in the case of random delay as the shield exploits the knowledge of the delay transitions to allow the agent to act more aggressively. The white color represents initial states that have maximum safety probability less than 0.95. For the gridworld setup, the obstacle is located at (4,4). (c) and (f) Show that as latency increases, the system tends to be conservative, leading to increased distances in the car-following scenario, and an increased number of ties in the gridworld case.

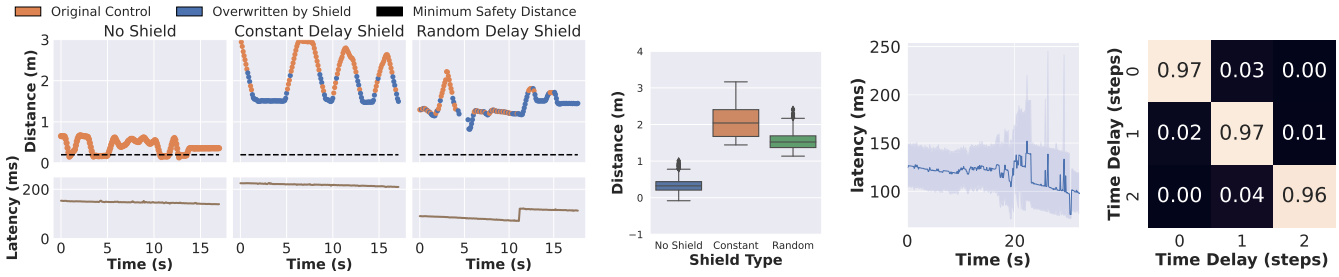
For the car-following and the hardware setup, the safety specification is $\square\neg S_{\text{unsafe}}$, where S_{unsafe} consists of states where the distance between the cars is less than 5 m and 0.2 m respectively. For the gridworld, it is $\neg\ S_{\text{unsafe}}\ \cup$ goal, where S_{unsafe} is the set of states where the robot and the obstacle are in the same location, i.e., collision. In our simulation environments, we experimented with the maximum delay ranging from 0 to 3 time steps for both constant and random delay. For random delay, we assume a delay transition probability function, $\mathbb{P}_\tau$, where the delay is mostly 0 and changes to other values with low probability.

How does the performance of our safe networked control approach vary with communication delay? The safety of the teleoperated robot reduces when the communication delay increases. We observe this in Fig. 2(a) and (d) for the two simulation setups, for the constant delay case. The set of states for which maximum safety probability $V^{\max}_{\mathcal{M},\varphi}(s)$ (Section III) is greater than a δ value shrinks with increasing delay. It shows that when the delay is large it is safer for the robot to stay farther away from the dynamic obstacle (gridworld) and for the ego robot to maintain a larger relative distance and velocity between itself and the leader car (car-following).

The shields ensure the desired safety probability δ for different delays. However, for the same safety probability, the task performance degrades with increasing delay due to increasing uncertainty in the system state. We show this quantitatively for the two simulation setups with constant delay. In the gridworld, with larger delays, the shield increasingly restricts the robot from moving aggressively toward the goal to avoid collisions. As such, it effectively sacrifices a win for a tie. Similarly, in the car-following scenario, the distance maintained from the leader robot increases (Fig. 2(c)). We observe a similar trend in our hardware setup that runs on a wireless network with stochastic delays (Fig. 3(a)). During the initial 10 s when the delay is less than 100 ms, the average distance maintained is less than 1.25 m. Then, when the delay is about 200 ms, the ego-robot starts to maintain a larger distance of around 1.5 m.

How does δ affect the safety-efficiency trade-off? Our key insight is that we can vary δ to trade off safety for task efficiency. We observe this in the constant delay case, where increasing δ leads to increasingly conservative behavior with more restrictions from the shield. In the gridworld, Fig. 2(f) shows fewer wins and more ties, an indicator of reduced task efficiency. Additionally, the number of losses decreases as safety is prioritized. Similarly, for car-following, the average distance maintained from the leader car increases (Fig. 2(c)). On the other hand, $\delta = 0$ is the un-shielded approach, which leads to a violation of the safety specification.

How does incorporating the delay transition probability function $\mathbb{P}_\tau$ affect safety and efficiency? We now illustrate that by



(a) Qualitative Results. (b) Quantitative Results. (c) WiFi traces. (d) Delay transition model

Fig. 3. Real World Demonstration Results. (a) Shows recorded trajectories from our hardware setup. Without our safe networked control approach, the system fails to satisfy the safety specification "always remain at least 0.2 meters away". However, our approach can satisfy the safety specification for constant and random delays. (b) The shield for the random delay case exploits the knowledge of delay transitions in Wi-Fi rather than assuming only the maximum latency, which allows the ego robot to follow the leader car at a closer distance. We set $\epsilon = 0.95$ to construct the ϵ-shield. In (c) and (d), we show how the delay transition probability function, $\mathbb{P}_\tau$, is estimated using multiple runs of the Wi-Fi latency time-series data. Here, $\mathbb{P}_\tau$ is a conditional probability distribution with three possible delays (0,1,2), where each delay is a bin of size 100 ms.

incorporating $\mathbb{P}_\tau$, our safe networked control approach performs more efficiently since the DC $-$ MDP model is more accurate. Whereas, when only $\tau_{\max}$ is known, the model is less accurate as it assumes the observation from $\tau_{\max}$ steps before to be the latest available system state even if the delay is small and more recent system states are available. We compare the DC $-$ MDP for constant delay against the DC $-$ MDP for random delay with $\tau_{\max} = 3$ in both cases. Firstly, Fig. 2(b) and 2(e) show that the set of states for which maximum safety probability $\mathrm{V}^{\max}_{\mathcal{M},\varphi}(s)$ (Section III) is greater than a δ value is larger when $\mathbb{P}_\tau$ is incorporated. Secondly, we observe more wins and fewer draws in the gridworld, and lower aggregate distance maintained for the car following setup as seen in Fig. 2(f) and 2(c) respectively. For the hardware setup, $\mathbb{P}_\tau$ is obtained experimentally (see Fig. 3(d)). Similar to the car following setup, the distance maintained between the two robots is less in the case of random delay when compared to constant delay (see Fig. 3(b)). This difference in safety distance is statistically significant with a Wilcoxon p-value < 0.001. To summarize, we infer that incorporating $\mathbb{P}_\tau$ in our DC $-$ MDP design allows for efficient task performance *without* compromising safety.

Does a minimally intrusive shield always lead to safety? Fig. 3(a) shows the state trajectory in the presence and absence of the ϵ-shield, and the instances when the ϵ-shield overwrites the cloud controller of the hardware setup. The ϵ-shield overwrites control commands when close to the leader car (relatively unsafe), and is inactive when further away. For example, in the constant delay case, the shield is inactive when the distance is above $\sim$2.25 m, and still ensures safety.

What are the practical effects of discretizing the state space and communication delay? We explain the effects of discretization on the time taken for the DC $-$ MDP's shield construction and the achievable safety probabilities. To assess the effect on the time taken for the shield construction, we quantify the time complexity of Algorithm 1, which mainly depends on line 2 (maximum safety probability for all state-action pairs). Since this is a value iteration procedure, the time complexity of Algorithm 1 is in the order of $\mathcal{O}(|\mathrm{S}|^2|\mathrm{Act}|)$ for any MDP $\mathcal{M} = \langle \mathrm{S}, \mathrm{Init}, \mathrm{Act}, \mathbb{A}, \mathbb{P} \rangle$ (refer to [16]). From the *Basic* MDP to DC $-$ MDP, the state space increases exponentially with $\tau_{\max}$ from $|\mathrm{S}|$ to the order of $|\mathrm{S}|(|\mathrm{Act}|+1)^{\tau_{\max}}(\tau_{\max}+1)$. So, the time complexity of Algorithm 1 for the DC $-$ MDP also increases accordingly. However, note that *the shield construction is an offline process*, and for practically observed delay values (Fig. 3(c)), our approach scales well. We also provide a comprehensive analysis of the state space size, time taken to compute line 2 in Algorithm 1, and the size of the synthesized shield for the DC $-$ MDP for our simulation environments in Table I. Note that even for $\tau_{\max} = 3$, the time taken to compute the maximum safety probabilities is only close to an hour and the shield size is less than 50 MB. The effect of discretization on safety probability depends on the environment and the discretization method used, which is beyond the scope of this article.

TABLE I
RUN TIME AND MEMORY ANALYSIS FOR THE SHIELD CONSTRUCTION

	Metrics	Constant delay: 0	Constant delay: 1	Constant delay: 2	Constant delay: 3	Random delay: 3 (max)
Car-following	States	484	2420	12100	60500	75504
	Time (s)	0.015	0.044	0.299	2.173	34.08
	Memory (KB)	12	68	360	1978	2645
Gridworld	States	8192	40960	204800	1024000	1277952
	Time (s)	1.694	19.711	175.84	1424.44	3956.81
	Memory (KB)	440	1448	6808	36210	48546

In this table, we show the number of states, time taken to compute the maximum safety probabilities (value iteration, line 2, algorithm 1), and the memory occupied by the shield for both the car-following and the grid-world simulation environments. The value iteration process terminates when the maximum change in the safety probability for any state between two consecutive iterations is less than 10^{-6}.

VII. CONCLUSION AND FUTURE DIRECTIONS

This article provides a novel approach to accurately model the networked control system transitions, in the presence of stochastic communication delays, as an MDP. Consequently, we use the MDP to synthesize shields for safe networked control. We demonstrate the efficiency of our approach on simulation and hardware setups. Our work is timely since we are seeing a surge of teleoperated robots. As future work, we believe that exploring state space reduction techniques to handle the exponential growth of state space in DC $-$ MDP and exploring solutions for continuous-time systems with delay using HJ reachability are promising directions.

ACKNOWLEDGMENT

The work solely reflects the opinions and conclusions of its authors and does not represent the views of any sponsor.

REFERENCES

[1] I. E. Rassi and J.-M. E. Rassi, "A review of haptic feedback in tele-operated robotic surgery," *J. Med. Eng. Technol.*, vol. 44, no. 5, pp. 247–254, 2020.

[2] O. E. Marai, T. Taleb, and J. Song, "AR-based remote command and control service: Self-driving vehicles use case," *IEEE Netw.*, vol. 37, no. 3, pp. 170–177, May/Jun. 2023.

[3] "Who's driving that food delivery bot? It might be a Gen Z gamer," 2022. Accessed Feb. 3, 2022. [Online]. Available: https://www.latimes.com/business/story/2022-03-17/california-autonomous%-sidewalk-food-delivery-robots-coco-starship-kiwibot

[4] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi reachability: A brief overview and recent advances," in *Proc. IEEE 56th Annu. Conf. Decis. Control*, 2017, pp. 2242–2253.

[5] J. F. Fisac, N. F. Lugovoy, V. R. Royo, S. Ghosh, and C. J. Tomlin, "Bridging Hamilton-Jacobi safety analysis and reinforcement learning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 8550–8556.

[6] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 3387–3395.

[7] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath, "Reinforcement learning for safety-critical control under model uncertainty, using control Lyapunov functions and control barrier functions," in *Proc. Robot.: Sci. Syst. (RSS)*, 2020.

[8] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2669–2678.

[9] S. Carr, N. Jansen, S. Junges, and U. Topcu, "Safe reinforcement learning via shielding under partial observability," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 14748–14756.

[10] N. Jansen, B. Könighofer, S. Junges, A. Serban, and R. Bloem, "Safe reinforcement learning using probabilistic shields," in *Proc. 31st Int. Conf. Concurrency Theory*, 2020, vol. 171, pp. 3:1–3:16.

[11] B. Könighofer, F. Lorber, N. Jansen, and R. Bloem, "Shield synthesis for reinforcement learning," in *Proc. Int. Symp. Leveraging Appl. Formal Methods*, 2020, pp. 290–306.

[12] S. Adlakha, S. Lall, and A. Goldsmith, "Networked Markov decision processes with delays," *IEEE Trans. Autom. Control*, vol. 57, no. 4, pp. 1013–1018, Apr. 2012.

[13] K. V. Katsikopoulos and S. E. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 568–574, Apr. 2003.

[14] E. Derman, G. Dalal, and S. Mannor, "Acting in delayed environments with non-stationary Markov policies," in *Proc. Int. Conf. Learn. Representations*, 2020.

[15] M. O'Kelly et al., "F1/10: An open-source autonomous cyber-physical platform," 2019, *arXiv:1901.08567*.

[16] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[17] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation.," *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[18] J. Kuffner, "Cloud-enabled robots," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2010.

[19] A. K. Tanwani, N. Mor, J. D. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 4559–4566.

[20] S. Chinchali et al., "Network offloading policies for cloud robotics: A learning-based approach," *Auton. Robots*, vol. 45, pp. 997–1012, 2021.

[21] G. Mohanarajah, D. Hunziker, R. D'Andrea, and M. Waibel, "Rapyuta: A cloud robotics platform," *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 2, pp. 481–493, Apr. 2015.

[22] N. Tian, A. K. Tanwani, K. Goldberg, and S. Sojoudi, "Mitigating network latency in cloud-based teleoperation using motion segmentation and synthesis," in *Proc. 19th Int. Symp. Robot. Res.*, 2022, pp. 906–921.

[23] T. Lancewicki, A. Rosenberg, and Y. Mansour, "Learning adversarial Markov decision processes with delayed feedback," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 7281–7289.

[24] S. Pranger, B. Könighofer, M. Tappler, M. Deixelberger, N. Jansen, and R. Bloem, "Adaptive shielding under uncertainty," in *Proc. Amer. Control Conf.*, 2021, pp. 3467–3474.

[25] B. Könighofer, J. Rudolf, A. Palmisano, M. Tappler, and R. Bloem, "Online shielding for reinforcement learning," *Innovations Syst. Softw. Eng.*, vol. 19, pp. 379–394, 2023.

[26] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, "Reinforcement learning with probabilistic guarantees for autonomous driving," 2019, *arXiv:1904.07189*.

[27] W.-C. Yang, G. Marra, G. Rens, and L. D. Raedt, "Safe reinforcement learning via probabilistic logic shields," 2023, *arXiv:2303.03226*.

[28] D. Aksaray, Y. Yazıcıoğlu, and A. S. Asarkaya, "Probabilistically guaranteed satisfaction of temporal logic constraints during reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 6531–6537.

[29] S. Li and O. Bastani, "Robust model predictive shielding for safe reinforcement learning with stochastic dynamics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 7166–7172.

[30] C. Liu et al., "Algorithms for verifying deep neural networks," *Foundations Trends Optim.*, vol. 4, no. 3-4, pp. 244–404, 2021.