Reasoning about Sensing Uncertainty in Decision-Making for Self-Adaptation

Javier Cámara, Wenxin Peng, David Garlan, and Bradley Schmerl

Institute for Software Research, School of Computer Science
Carnegie Mellon University, Pittsburgh PA 15213, USA,
{jcmoreno, garlan, schmerl}@cs.cmu.edu, wenxinp@andrew.cmu.edu

Abstract. Self-Adaptive systems are expected to adapt to unanticipated run-time events using imperfect information about their environment. This entails handling the effects of uncertainties in decision-making, which are not always considered as a first-class concern. This paper contributes a formal analysis technique that explicitly considers uncertainty in sensing when reasoning about the best way to adapt, possibly executing uncertainty reduction operations to improve system utility. We illustrate our approach on a Denial of Service (DoS) attack scenario and present some preliminary results that show the benefits of uncertainty-aware decision-making with respect to using an uncertainty-ignorant approach.

Keywords: Self-Adaptation, decision-making, uncertainty, stochastic games

1 Introduction

Complex software-intensive systems are increasingly relied on in our society to support tasks in different contexts that are typically characterized by a high degree of *uncertainty*. Self-adaptation [12,22] is regarded as a promising way to engineer in an effective manner systems that are *resilient* to run time changes despite the different uncertainties derived from their execution environment (e.g., resource availability, interaction with human actors), goals, or even in the system itself (e.g., faults).

The information and models employed for decision-making in self-adaptive systems are also subject to different types of uncertainty (e.g., sensor readings may be inaccurate, some important aspect of the domain may be abstracted away in models). However, despite the fact that these uncertainties can have a noticeable impact on run-time system behavior, many approaches to engineering self-adaptation do not model the uncertainties that affect the system explicitly or as a first-class entity [20]. Moreover, some types of adaptation tactics can reduce uncertainty at run time (e.g., introducing a CAPTCHA in a web system can reduce the uncertainty about potentially malicious clients controlled by bots accessing the website). These tactics often come at a cost (e.g., CAPTCHA can increase the annoyance of legitimate clients accessing the website, whose sessions are disrupted). So, it is also important to enable systems to reason about the trade-offs of enacting such tactics, quantifying the benefits of uncertainty reduction and balancing them against its cost when trying to achieve system goals.

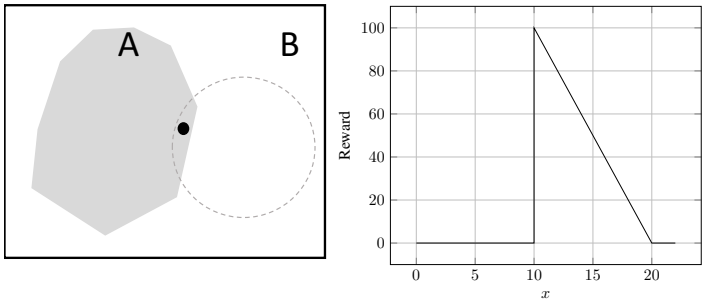
One of the most popular patterns for building self-adaptation into software-intensive systems is IBM's MAPE-K [23], which integrates activities to monitor, analyze, plan,

and execute adaptations in close-loop control over a managed software (sub)system. Furthermore, a central knowledge base that typically includes models about the managed system, its environment, and adaptations, informs the different MAPE activities.

According to categorizations carried out by different authors [17, 24, 25], uncertainty occurs in all activities associated with the MAPE-K loop. In this paper, we focus on the aleatoric uncertainties (i.e., due to the randomness of events) induced by inaccuracies in sensor readings (i.e., deviations from the ideal reading of the sensor), and how its explicit representation and incorporation into reasoning mechanisms can improve decision-making in self-adaptation.

Concretely, we investigate two research questions: **(RQ1)** The extent to which explicit representation and reasoning about sensing uncertainty improves the quality of adaptation decisions, and **(RQ2)** the circumstances under which uncertainty awareness improves the quality of decisions (subject to RQ1 being true).

To motivate our approach, we consider a simple scenario illustrated in Figure 1(a), where the system/environment state space is divided into regions A and B. We assume that the sensors employed to monitor some of the variables that form the system/environment state are not very accurate, and therefore the monitoring infrastructure cannot determine the exact system/environment state (which could be any point in the circle).



(a) (b)

Fig. 1. Simple model scenario.

Figure 1(b) introduces the concept of *reward*, which is an indicator of how well the system is meeting its goals (e.g., minimizing malicious users or maximizing requests served). Every time a system takes some action, it can be rewarded based on how this action impacts the state of the system (and how well the new state aligns with system goals). The higher the reward, the better the decision is. In other words, we assume that the system's target in this scenario is to accumulate as much reward as possible over time by taking a series of actions.

In this simple scenario, we assume that reward is only associated with metric x, as shown in Figure 1(b). When x is below 10, there is no reward; when x is equal to 10, the reward is at its maximum, and decreases as x moves away from 10. Thus, the entire state space of this model is effectively divided into two regions:

Region A: $x < 10$ **Region B**: $x \geq 10$

Suppose that this system can perform a single action to reduce the value of x by an integral amount. Hence, when the system determines that the current state lies within

region B (according to the *observed* value of x), it should try to decrease the value of x to maximize reward, making it as close as possible to 10 (but without going below 10). However, the sensor that monitors the value of x is not very accurate and the system has to make the best possible decision under the uncertainty that arises due to the inaccuracy of the sensing process. In particular, if the sensor indicates that the value of x is higher than it really is in states close to $x = 10$, there is a risk that the system will reduce the value of x below 10, incurring in a high penalty (due to the fact that no further reward will be accrued).

There is a need to enable formal reasoning mechanisms for self-adaptation to reduce such risks. In this paper, we contribute a formal analysis technique that enables us to quantify the potential benefits of explicitly considering sensing uncertainty in models and decision-making mechanisms for self-adaptation, and produce adaptation decisions with worst-case guarantees. The formal underpinnings of our proposal are based on model checking of stochastic multiplayer games (SMGs) [10]. The main idea behind the approach is analyzing the interplay of a self-adaptive system and its environment in a game. System and environment are modeled as players with independent behaviors (reflecting the fact that processes in the environment – in this case, sensing – cannot be controlled by the system). System and environment players compete against each other, providing a theoretical setup for worst-case scenario analysis.

The remainder of this paper first presents some background on SMG in Section 2, used by a description of our approach in Section 3 illustrated on the simple scenario that we have introduced. Next, Section 4 introduces a more complex self-protecting systems scenario and discusses some results on comparing uncertainty-aware vs. uncertainty-ignorant decision making. Section 5 discusses some related work. Section 6 presents some conclusions and points at directions for future work.

2 Background: Model Checking of Stochastic Multiplayer Games

Automatic verification techniques for probabilistic systems have been successfully applied in a variety of application domains including security [14,26] and communication protocols [21]. In particular, techniques such as probabilistic model checking provide a means to model and analyze systems with stochastic behavior, and enable quantitative reasoning about probability and reward-based properties (e.g., resource usage, time).

Competitive behavior may also appear in systems when some component cannot be controlled, and could behave according to different or even conflicting goals with respect to other components in the system. Self-adaptive systems are a good example of systems in which the behavior of some components that are typically considered as part of the environment (non-controllable software, network, human actors) cannot be controlled by the system. In such situations, a natural fit is modeling a system as a game between different players, adopting a game-theoretic perspective.

Our approach to analyzing self-adaptation builds upon a recent technique for modeling and analyzing stochastic multi-player games (SMGs) extended with rewards [10]. In this approach, systems are modeled as turn-based SMGs, meaning that in each state of the model, only one player can choose between several actions, the outcome of which can be probabilistic. Players in the game can follow strategies for choosing actions in

the game, cooperating in coalition to achieve a common goal, or competing to achieve their own goals. These strategies are guaranteed to achieve optimal expected rewards for the kind of cumulative reward structures that we use in our models.[1]

Reasoning about strategies is a fundamental aspect of model checking SMGs, which enables checking for the existence of a strategy that is able to optimize an objective expressed as a property in a logic called rPATL. Concretely, rPATL can be used for expressing quantitative properties of SMGs, and reasoning about the ability of a coalition of players to collectively achieve a particular goal (e.g., ensuring that the probability of an event's occurrence or an expected reward measure meets some threshold).

rPATL is a CTL-style branching-time temporal logic that incorporates the coalition operator $\langle\langle C \rangle\rangle$, combining it with the probabilistic operator $\mathsf{P}_{\bowtie \mathsf{q}}$ and path formulae from PCTL [2]. Moreover, rPATL includes a generalization of the reward operator $\mathsf{R}^{\mathsf{r}}_{\bowtie \mathsf{x}}$ from [19] to reason about goals related to rewards. An extended version of the rPATL reward operator $\langle\langle C \rangle\rangle \mathsf{R}^{\mathsf{r}}_{\mathsf{max}=?}[\mathsf{F}\ \phi]$ enables the quantification of the maximum accrued reward r along paths that lead to states satisfying state formula ϕ that can be guaranteed by players in coalition C, independently of the strategies followed by the rest of the players. An example of the typical usage of combining the coalition and reward maximization operators is $\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{utility}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{end}]$, meaning "value of the maximum utility reward accumulated along paths leading to an end state that a player sys can guarantee, regardless of the strategies of other players."

3 Approach

In this section, we describe our approach to analyzing uncertainty-aware self-adaptation, illustrating it on the simple scenario described in the introduction. First, we introduce the definition of the formal model for the game, including a description of how reward is collected. The section finishes by describing the analytical process followed to quantify the difference between uncertainty-aware and uncertainty-ignorant decision-making.

3.1 Formal Model Definition

The purpose of this model is to compare uncertainty-aware adaptation, i.e., decision-making that considers explicitly uncertainty information (in this case induced by inaccuracies in sensing), against uncertainty-ignorant adaptation that assumes that there is no uncertainty in the information it employs for decision-making. The model is implemented using PRISM-Games [9], a tool capable of model checking rPATL properties on stochastic multiplayer games.

The model encodes a game played by an environment and a system player, and it can be instantiated in two variants: one in which the system player is uncertainty-aware, and another in which the system is uncertainty-ignorant. The details of how these different variants are used are explained in Section 3.2.

Defining the Players. There are two players in this model: Environment (env) and System (sys). These two players take turns to take actions. As shown in Listing 1.1,

[1] See Appendix A.2 in [10] for details.

the turn is controlled by the global variable $turn$. There are two other global variables: $real\_x$ represents the real value of x at any given time, whereas obs_x represents the value of x observed by the system (i.e., the value obtained by the inaccurate sensor).

```
1  player sys target_system,[act],sensor,[sense] endplayer
2  player env environment,[generate] endplayer
3  const ENV_TURN, SYS_TURN;
4  global turn:[ENV_TURN..SYS_TURN] init ENV_TURN; // Used to alternate between players
5  global obs_x, real_x:[0..20];
```

Listing 1.1. Player definition.

The game is played in alternating turns by the system and the environment players. A typical cycle of the game works in the following way:

1. The environment generates the real value of x ($real\_x$ - see Listing 1.2, line 4).

```
1  const INIT_X, error, MAX_TURNS;
2  module environment
3    t : [-1..MAX_TURNS] init 0;
4    [] (t>=0 & t<MAX_TURNS) & (turn=ENV_TURN) -> // 1.Generate value of real x
5                  (t'=t+1) & (turn'=SYS_TURN) & (real_x'=INIT_X);
6  endmodule
```

Listing 1.2. Simple environment model definition.

2. The system senses the value obs_x (Listing 1.3, line 2). The uncertainty in the sensing process is modeled by a simple probability distribution, in which there is 0.5 probability that the sensor reads the value accurately (i.e., $obs\_x = real\_x$), and 0.5 probability the reading exceeds the real value of x by a constant $error$ (i.e., $obs\_x = real\_x + error$).

```
1  module sensor
2          [sense] true -> 0.5:(obs_x' = real_x) + 0.5:(obs_x' = real_x+error);
3  endmodule
```

Listing 1.3. Sensor definition.

3. After obtaining the observed value obs_x, the system (Listing 1.4, line 7) can choose to: (a) do nothing (line 11), or (b) reduce the value of $real\_x$, subtracting the value of s_step from $real\_x$ (lines 8-10). s_step is just the saturated value of a constant $step$ supplied as parameter to the model, which represents the maximum magnitude of the modification that the system's effector can carry out on the value of x. For example, if $step = 3$, s_step can take values in $\{1, 2, 3\}$.

```
1  const step;
2  formula s_step = obs_x-step >= 10 | obs_x < 10 ? step : obs_x - 10;
3
4  module target_system
5  expected_x:[0..20] init 0;
6  new_info:[0..1] init 0;
7      [sense] (new_info=0) & (turn=SYS_TURN) -> (new_info'=1);  // 2. Sense
8      [act] (new_info=1) & (turn=SYS_TURN) ->  // 3.a. Act
9            (real_x'=real_x-s_step>=0?real_x-s_step:0) & (new_info'=0)
10       & (expected_x'=obs_x-s_step>=0?obs_x-s_step:0) & (turn'=ENV_TURN);
11     [] (new_info=1)&(turn=SYS_TURN) ->  // 3.b. Do nothing
12                 (expected_x'=obs_x) & (turn'=ENV_TURN) & (new_info'=0);
13 endmodule
```

Listing 1.4. Simple system model definition.

Note that in the listing above, $expected\_x$ encodes the expected value of x from the perspective of the system after its turn is completed (the expected value of x is built on the value of obs_x).

The cycle repeats until the maximum number of turns played by the system and the environment is reached. However, we assume in the rest of the discussion a single-turn game for the sake of clarity (i.e., the game ends after the environment, and then the system, play one turn each).

Collecting Reward. There are three types of rewards in this model. We use each one of them to emulate different types of adaptation (Listing 1.5):

1. rU: reward collected if the system has the accurate information to make a decision, i.e., when system knows $real\_x$.
2. rEU: reward collected if the system can only sense obs_x and the system is unaware of the uncertainty, i.e., the system assumes that obs_x is an accurate reading.
3. rEU_uncertain: reward collected if the system can only see obs_x, but is aware of the uncertainty. In this case, the system knows that there is a 0.5 probability that obs_x is not accurate and it calculates the reward factoring in this probability.

```
1  formula rU = (real_x<10? 0:200−10∗real_x);
2  formula rEU = (expected_x<10? 0:200−10∗expected_x);
3  formula rEU_uncertain = 0.5∗rEU+0.5∗rU;
4
5  rewards "rEU_uncertain" // Expected instantaneous utility reward (uncertainty−aware adaptation)
6     (turn=ENV_TURN) & (t>=1) :rEU_uncertain;
7  endrewards
8
9  rewards "rEU" // Expected instantaneous utility reward (uncertainty−ignorant adaptation)
10    (turn=ENV_TURN) & (t>=1) : rEU;
11 endrewards
12
13 rewards "rU" // Real Instantaneous utility reward
14   (turn=ENV_TURN) & (t>=1) : rU;
15 endrewards
```

Listing 1.5. Simple model reward structure definition.

3.2 Analytical Approach

To compare the uncertainty-aware vs. uncertainty-ignorant adaptation, we use rPATL specifications that enable us to analyze:

1. R_{real}: The maximum utility that the system can obtain when it has the accurate information (in our scenario, when the system tries to maximize the reward based on $real\_x$). We can get this value by generating a strategy for the property:

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{1}$$

2. $R_{u-ignorant}$:The maximum utility that adaptation is able to obtain without factoring uncertainty. To obtain this value, we proceed in two steps:

(a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs (there is no uncertainty in the expected value of x because the value of obs_x is accurate):

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{2}$$

(b) We verify Property 1 under the generated strategy for Property 2. This quantifies the real utility achieved (based on the value of $real\_x$), under the strategy generated based on the beliefs of the system (i.e., the value of x is obs_x, and it coincides with the real one).

3. $R_{u-aware}$:The maximum utility that the adaptation is able to obtain when considering uncertainty. To quantify this value, we proceed in two steps:
   (a) First, we generate a strategy using the following property that quantifies the maximum expected accrued reward that the system "believes" it can guarantee based on its beliefs. However, in this case the system is aware that the probability of $real\_x = obs\_x$ is only 0.5, so the strategy generated already accounts for the possibility of inaccurate readings. This is encoded in the reward $rEU\_uncertain$ in Listing 1.5 (line 5).

$$\langle\langle \mathsf{sys} \rangle\rangle \mathsf{R}^{\mathsf{rEU\_uncertain}}_{\mathsf{max}=?}[\mathsf{F}^{\mathsf{c}}\ \mathsf{t} = \mathsf{MAX\_TURNS}] \tag{3}$$

   (b) We verify Property 1 under the generated strategy for Property 3. This quantifies the real reward under the strategy for uncertainty-aware decision-making.

Experiment and observations for the simple scenario. For our experiment, we collected the value of reward for uncertainty-aware and uncertainty-ignorant adaptation with both sensor $error$ and actuator impact $step$ taking values in $\{1, 3\}$. The range of values for x explored is $\{0, \ldots, 20\}$.

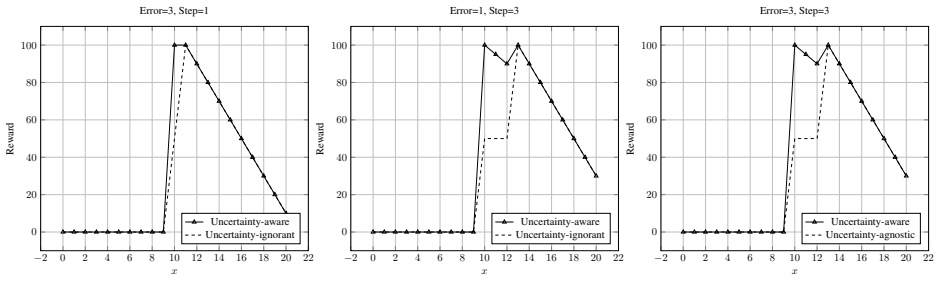


Fig. 2. Simple model scenario results.

Figure 2 compares the reward obtained by uncertainty-aware and uncertainty-ignorant adaptation (i.e., $R_{u-aware}$ and $R_{u-ignorant}$, respectively). Looking at the results, we can make the following observations:

1. *When in a "safe" region, uncertainty does not matter.* When the value of x is in region A ($x \geq 10$), and not close to the threshold $x = 10$, the reward obtained is not affected by uncertainty in any way, since there is no risk that the system will modify the value below the threshold, leading to a loss of reward. In practice, both

uncertainty-aware and ignorant adaptations will choose to reduce the value of x to obtain more reward. This can be observed in Figure 2, where the reward obtained by both adaptation variants converge as x moves to higher values, away from the threshold $x = 10$. Similarly, when the system is in region B ($x < 10$) uncertainty does not make any difference, since there is nothing that the system can do to collect more reward. So, both adaptation variants will behave in the same way.

2. *When close to the boundary between regions, uncertainty-aware adaptation performs better*. When the system is in region A, but in values that are close to the boundary between regions A and B, there is a chance that the system will make a sub-optimal decision due to the uncertainty in sensing. Concretely, in the uncertainty-ignorant variant of adaptation, the system can determine that it is safe to reduce the value of x by a given amount based on the value of obs_x (when in reality, the value of x will go below 10 and reward will not be collected). This penalizes uncertainty-ignorant adaptation with respect to the uncertainty-aware variant, which is already accounting for the likelihood of an undesirable outcome, and is more conservative when choosing to reduce the value of x. Figure 2 shows how the different choices of adaptation variants lead to increased rewards in uncertainty-aware adaptation when the value of x is close to the boundary between regions.
3. *The difference between adaptation approaches is greater when sensor error is paired with actuator impact*. As sensor error increases, we would expect to see uncertainty-ignorant adaptation's reward progressively decrease. However, this is only true if sensor error is paired with higher actuator impact values, since otherwise the limited scope of the actuator mitigates the potentially detrimental effects that making the wrong choice would have on reward. For instance, if $error = 3$, but step=1, the plot in Figure 2 (left) shows that there is little performance difference between the two variants of adaptation. This is because, even if the system makes the wrong choice under uncertainty-ignorant adaptation, e.g., when $x = 12$, the actuator can at most reduce x to a value 11, incurring only a light penalty, compared to uncertainty-aware adaptation. However, if we consider the same value of $x = 12$ when $step = 3$ (center, right), the difference in reward between approaches is much more pronounced because in situations in which reducing x is the wrong choice, it is more likely that x will go under the threshold $x = 10$, incurring a higher penalty.

4 Case Study: Denial of Service Attack (DoS)

In this section, we describe our approach on a more complex scenario where an enterprise web infrastructure similar to the Znn.com system experiences a DoS attack [26]. When web infrastructure experiences unusually high traffic, the cause of the high traffic might be malicious (e.g., the system is experiencing a DoS attack) or legitimate (some content has suddenly become popular - e.g., the slashdot effect). Treating legitimate users as DoS attackers by mistake, applying strategies like blocking their requests for accessing the website could be harmful to the business. Thus, uncertainty about such situations should be considered when applying defensive adaptation strategies to the system, evaluating carefully the benefit and cost of different adaptation choices, possibly including actions that can reduce uncertainty.

To facilitate the understanding of the DoS scenario, we structure our model in a similar way to the simple model described earlier and make the following assumptions:[2]

1. *The entire space is divided into two regions:* (i) *DoS*, in which we assume that the system is experiencing an attack, and (ii) *Normal*, in which the system does not experience any anomalous activity that indicates a DoS attack.
2. *Regions are associated with specific metrics.* The system's state is determined by a single metric that captures the estimated *percentage of malicious clients* accessing the system (mc). This metric is an abstract concept used as proof of concept. We assume that if mc is above a given threshold, the system is in the DoS region of the state space; otherwise the system is considered to be in the $Normal$ region.
3. *The system does not know the real value of metrics.* The observable value of the metric mc may not reflect its real value.
4. *Uncertainty in sensing is represented by a probability distribution.* We employ a normal distribution to model the observed percentage of malicious clients mc.
5. *The uncertainty-aware version of the system has knowledge about the probability distribution function that captures uncertainty in sensing.* To simplify the problem, we assume that the system has knowledge of the probability distribution function that represents how observed values are generated during the sensing process. In the real world, this knowledge may be obtained from historical data, for instance.

The two main extensions with respect to the simple scenario presented earlier are: (i) a richer set of actions or *tactics* that the system can carry out to influence state variables including those that reduce uncertainty, and (ii) a more sophisticated notion of reward that factors in metrics along more than one dimension of concern.

- *Tactics*. In the simple previous example, the system can either do nothing or act on the value of x by decreasing it. In the real world, a system may have a richer variety of adaptation actions, or *tactics*, to respond to run-time events. In this model, we divide tactics into two kinds:
  - *Uncertainty Reduction Tactics*. This kind of tactic can reduce uncertainty (in our scenario the uncertainty associated with the sensing of system metrics). This type of tactic often comes at a cost. For example, introducing CAPTCHA[3] can reduce uncertainty about the maliciousness of clients accessing the system (by determining which ones are controlled by bots), but it will increase the annoyance of legitimate users, who find their activities disrupted by the CAPTCHA.
  - *Non Uncertainty Reduction Tactics*. Tactics that do not reduce uncertainty like *blackholing* clients (i.e., dropping their incoming requests) do not provide any new information (e.g., about who is controlling the clients). Decisions of whether to exercise these tactics are therefore highly dependent on the quality of the information available to the system (i.e., the observed values of metrics).

[2] A full listing of the PRISM-Games model can be found in [5].

[3] CAPTCHA is a type of challenge-response test used in computing to determine whether or not the user is human (https://en.wikipedia.org/wiki/CAPTCHA).

– *Reward Model*. In the simple scenario, the reward was related only to one dimension. However, in this scenario there are two dimensions of concern: security and user experience, and we assume them to be of equal importance. Security is directly related to the metric percentage of malicious clients mc (lower is better). User experience is also affected by system's choice of applying different tactics. For example, introducing CAPTCHA will increase the difficulty of legitimate users accessing system services and therefore increase their annoyance. We consider therefore user annoyance (ua) as an additional metric for the user experience dimension of concern.

4.1 Formal Model Definition

This section provides a high-level description of the game model for the DoS scenario. The scenario is modeled as a stochastic game involving two players that represent the system (*sys*) and the environment (*env*):

– The system player consists of two processes or *modules* that represent the *target system* and the *gauge* that collects observed values of the mc metric. These two modules are synchronized by a shared action *gaugeMc*.
– The environment player consists of the *generator* and *environment* modules, which are synchronized via the *generateMc* shared action.

When the game starts, the environment player first generates the real value of the mc ($generateMc$), and it yields the turn to the system player. Next, the system player gauges mc ($gaugeMc$), producing its observed value. The system player then infers the region of the state space (DoS or Normal) based on the observed system metric, chooses one of the available tactics to execute ($blackHole$, $captcha$, or chooses not to do anything), and returns the turn to the environment.

The game contains three global variables: (i) $real\_mc$ is the real percentage of malicious clients, ranging from [0,100], represents the real system metric, (ii) obs_mc is observed percentage of malicious clients, and (iii) std_mc is the standard deviation associated with the perceived percentage of malicious clients. Note that obs_mc and std_mc together describe the uncertainty function for the observed system metric.

Generating the Real Value of Metric. *Generator* is a module that is responsible for generating the real value of metric ($real\_mc$) during every turn of the environment.

Gauging Information. The *gauge* module is responsible for gauging information. This process is crucial to our scenario model because it encodes how observed values of mc are generated from the real values of the variable (i.e., it captures the source of aleatoric uncertainty in the sensing process). Concretely, the observed value of metric can be captured as the function:

$$P(x) = \frac{1}{std\_mc\sqrt{2\pi}} e^{-(x-obs\_mc)^2/2std\_mc^2} \quad \text{or} \quad f(x) = P[X = x]$$

This probability density function is actually a conditional probability distribution of the observed value of the metric, given a real value ($P(obs\_mc|real\_mc)$). In this scenario, we encode this function using six points to simulate this normal distribution.

Selecting and Applying Tactics. After the system obtains the information about system metrics, it can choose a tactic for execution (or do nothing). The model has two variants that capture two alternative selection strategies:

1. *Uncertainty-ignorant*. The system does not have knowledge about the real value of the metric mc. It is also oblivious to the fact that there is uncertainty in the gauging process and therefore treats the observed value as the real information, selecting tactics based on this information.
2. *Uncertainty-aware*. The system has no knowledge of the real value of mc. However, the system has knowledge about the uncertainty in the gauging process and evaluates the expected result considering the probability distribution over different system states and selects tactics based on that.

The adaptation decision is evaluated for the selection strategies based on the value of the following set of variables:

1. $real\_mc$: Real value of metric mc.
2. emc: Expected percentage of malicious client after executing a tactic, assuming that $obs\_mc = real\_mc$.
3. ua: Real value for the metric user annoyance.
4. eua: Expected user annoyance after executing a tactic assuming $obs\_mc = real\_mc$
5. eua_dos: Expected user annoyance after executing a tactic if the system is currently at the DoS region.
6. eua_normal: Expected user annoyance after executing a tactic if the system is currently at the Normal region. This variable and eua_dos are used to calculate the expected reward when the system is aware of the uncertainty.

These six variables are used to calculate three types of reward (real, expected by uncertainty-aware, and expected by uncertainty-ignorant decision-making). By maximizing different types of reward, we can employ our formal model to generate adaptation decisions for: (a) adaptation based on real information, (b) uncertainty-ignorant adaptation, and (c) uncertainty-aware adaptation.

Table 1 summarizes the effect of exercising different tactics at different system states. For example, blackholing in both regions (DoS and Normal) reduces the $real\_mc$ by 30% but it increases user annoyance by 50% if the system is not in DoS, and by 10% if it is (reflecting the assumption that most clients will correspond to malicious users[4]).

Collecting Reward. Rewards are calculated based on both user annoyance and the percentage of malicious clients. In this case, the reward we employ for our game encodes a simple utility function in which both metrics contribute to the overall utility calculation with a weight of 0.5. We employ three types of rewards to analyze uncertainty-aware and uncertainty-ignorant decision-making.

$real\_mc$	Normal	DoS
IntroduceCAPTCHA	-10	-10
Blackhole	-30	-30
ua	Normal	DoS
IntroduceCAPTCHA	+10	+10
Blackhole	+50	+10

Table 1. Simple impact specification of tactics in a DoS adaptation scenario.

1. rIU (real utility): This reward is calculated based on the real value of the percentage of malicious clients ($real\_mc$).

[4] In this model, we assume that the effect of tactics is deterministic.

2. rEIU (expected utility for uncertainty-ignorant decision making): Is calculated based on the observable information about the percentage of malicious clients (obs_mc), and it is unaware of the uncertainty in sensing.
3. rEIU_uncertain (expected utility for uncertainty-aware decision making): Is also calculated based on the observed value of the metric mc (obs_mc). However, this alternative considers the uncertainty in sensing, since it draws the values for reward calculation based on all the possibilities captured in the probability distribution.

The calculation of $rEIU\_uncertain$ is the key of this model. When collecting reward in uncertainty-aware adaptation, we derive all the potential real values of metric mc, based on its observed value, and calculate the expected reward based on the probability distribution of these real values. In other words, *the system must have knowledge of the probability distribution of real values of the metric conditioned to its observed value* $P(real\_mc \mid obs\_mc)$ *to calculate* $rEIU\_uncertain$.

The observed value (obs_mc) is normally distributed given a real value ($real\_mc$), based on the joint probability mass function of two discrete random variables:

$$P(X = x, Y = y) = P(Y = y \mid X = x) * P(X = x) = P(X = x \mid Y = y) * P(Y = y)$$

4.2 Experiments and Observation

To compare uncertainty-aware with uncertainty-ignorant adaptation, we use the analytical process described in Section 3. For our experiment, we collected the value of reward for uncertainty-aware and uncertainty-ignorant adaptation with a DoS threshold value of 60, and different sensor standard deviations in the distribution that captures sensing uncertainty for mc $\sigma_{obs\_mc} \in \{10, 20\}$. The range explored for mc is $\{0, \ldots, 100\}$.

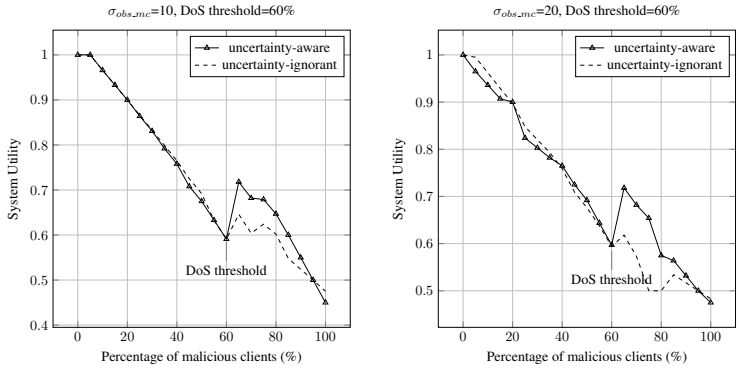


Fig. 3. DoS scenario results.

Figure 3 shows the result of our experiments. From these results, we can draw the following observations (which are consistent with the earlier example):

1. *When far from region boundary, uncertainty does not matter.* When the value of mc is in region DoS or Normal ($mc \geq 60$), and moves away from the threshold, the utility obtained both by uncertainty-aware and ignorant adaptations is similar.

2. *When close to the boundary between regions, uncertainty-aware adaptation performs better*. When the system is in region DoS, but in values that get close to the boundary between regions DoS and Normal, there is a chance that the system will make a sub-optimal decision due to uncertainty. Concretely, uncertainty-ignorant adaptation can determine that it is safe to blackhole clients based by the value of obs_mc and might incur in penalties for blackholing potentially legitimate clients.
3. *The difference between adaptation approaches is greater when standard deviation is higher*. As the standard deviation in sensor inaccuracies increases, we can see how the utility obtained by uncertainty-ignorant adaptation decreases. If we observe the plots, focusing on the range 60%- 90% of percentage of malicious clients, there is a noticeable drop in the utility obtained by the uncertainty-ignorant approach in the plot on the right, in which the standard deviation σ_{obs_mc} is doubled with respect to the one on the left.

5 Related Work

Uncertainty management has been studied by many authors in the field of self-adaptive systems, but not so far in managing sensing uncertainty. Possibility theory has been mainly used in approaches that deal with uncertainty in objectives, helping to assess the positive and negative consequences of uncertainty [1, 16, 27]. Other approaches employ probabilistic verification and estimates of the future environment and system behavior for optimizing the system's operation. These proposals target the mitigation of uncertainty due to *parameters over time* [3, 4, 15].

Although these approaches have shown promising results in dealing with different types of uncertainty they do not cover uncertainty that is directly caused by the information that is used as sensing input to the decision-making agent. Such uncertainty is especially important when the self-adaptive system is managing a cyber-physical system. In this case, attackers may exploit compromised sensors and effectors to steer a system into unsafe states that not only have an impact on the software, but ultimately on the physical context of the system.

The work in [18] is concerned with the estimation and control of linear systems when some of the sensors or actuators are corrupted by an attacker. The authors of [13] tackle a similar problem, with a stronger focus on sensing and state estimation in continuous-time linear systems, for which an attacker may have control over some of the sensors and inject (potentially unbounded) additive noise into some of the measured outputs. To characterize the resilience of a system against such sensor attacks, the authors introduce a notion of "observability under attacks" that addresses the question of whether or not it is possible to uniquely reconstruct the state of the system by observing its inputs and outputs over a period of time, with the understanding that some of the available system's outputs may have been corrupted by the attacker. The authors of [11] study CPS subject to dynamic sensor attacks, relating them to the system's strong observability. This work identifies necessary and sufficient conditions for an attacker to create a dynamically undetectable sensor attack and relates them to system dynamics.

Our approach can be regarded as complementary to these works, since it would enable us to potentially exploit the information provided by these approaches to improve

decision-making and provide worst-case scenario guarantees. In [7] we reported on an analogous application of this technique to quantify the benefits of employing information about the latency of tactics for decision-making in proactive adaptation, comparing it against approaches that make the simplifying assumption of tactic executions not being subject to latency.

6 Conclusions and Future Work

This paper has described an analysis technique based on model checking of stochastic multi-player games that enables us to quantify the benefits in adaptation performance of factoring sensing uncertainty explicitly into decision-making. Our results show that although uncertainty-aware adaptation is not guaranteed to perform better than uncertainty-ignorant adaptation in all cases, *it does perform at least comparably in all cases* (RQ1), and *performs better in boundary regions of the state space in which the dynamics of the system may change* (RQ1 and RQ2). This is a relevant finding, because systems that exhibit variability in the effects of adaptation tactics that depend on specific run-time conditions may obtain a remarkable benefit in terms of improved reliability and performance by factoring uncertainty into decision-making.

With respect to future work, we plan on extending decision-making under uncertainty to reason only with partial knowledge about the uncertainty function. The current version of our approach assumes that information about $P(observed\ value \mid real\ value)$ is known to the system, so it can derive $P(real\ value \mid observed\ value)$. A next logical step is to study how systems can gradually improve their ability to estimate the real state of the system, i.e. by automatically refining throughout subsequent system executions the knowledge that the system has about $P(real\ value \mid observed\ value)$. A second avenue for future work will investigate reasoning about uncertainty reduction and uncertainty-aware decision-making in human-in-the-loop adaptation, where human operators may provide information that is inaccurate, continuing the line started in [6,8].

Acknowledgments. This material is based on research sponsored by AFRL and DARPA under agreement number FA8750-16-2-0042 and by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the AFRL, DARPA, ONR or the U.S. Government. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. This material has been approved for public release and unlimited distribution.

References

1. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 18th Requirements Engineering Conference (RE 2010). pp. 125–134 (2010)

2. Bianco, A., de Alfaro, L.: Model checking of probabalistic and nondeterministic systems. In: FSTTCS. LNCS, vol. 1026. Springer (1995)
3. Calinescu, R., Kwiatkowska, M.Z.: Using Quantitative Analysis to Implement Autonomic IT Systems. In: ICSE (2009)
4. Calinescu, R., et al.: Dynamic QoS Management and Optimization in Service-Based Systems. IEEE Trans. Software Eng. 37(3) (2011)
5. Cámara, J., Garlan, D., Kang, W.G., Peng, W., Schmerl, B.: Uncertainty in self-adaptive systems: Categories, management, and perspectives. Tech. Rep. CMU-ISR-17-110, Institute for Software Research, Carnegie Mellon University (2017)
6. Cámara, J., Garlan, D., Moreno, G.A., Schmerl, B.: Evaluating trade-offs of human involvement in self-adaptive systems. In: Managing Trade-Offs in Self-Adaptive Systems (2016)
7. Cámara, J., Moreno, G.A., Garlan, D.: Stochastic game analysis and latency awareness for proactive self-adaptation. In: 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS. pp. 155–164 (2014)
8. Cámara, J., Moreno, G.A., Garlan, D.: Reasoning about human participation in self-adaptive systems. In: 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS. pp. 146–156 (2015)
9. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: A model checker for stochastic multi-player games. In: Piterman, N., Smolka, S. (eds.) Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13). LNCS, vol. 7795, pp. 185–191. Springer (2013)
10. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods in System Design 43(1), 61–92 (2013)
11. Chen, Y., Kar, S., Moura, J.M.F.: Cyber-physical systems: Dynamic sensor attacks and strong observability. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1752–1756 (2015)
12. Cheng, B.H.C., et al.: Software engineering for self-adaptive systems: A research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer (2009)
13. Chong, M.S., Wakaiki, M., Hespanha, J.P.: Observability of linear systems under adversarial attacks. In: 2015 American Control Conference (ACC). pp. 2439–2444 (July 2015)
14. Deshpande, T., Katsaros, P., Smolka, S., Stoller, S.: Stochastic game-based analysis of the dns bandwidth amplification attack using probabilistic model checking. In: Dependable Computing Conference (EDCC), 2014 Tenth European. pp. 226–237 (May 2014)
15. Epifani, I., et al.: Model Evolution by Run-Time Parameter Adaptation. In: ICSE. IEEE CS (2009)
16. Esfahani, N., Kouroshfar, E., Malek, S.: Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. pp. 234–244. ESEC/FSE '11, ACM (2011)
17. Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: de Lemos, R., Giese, H., Muller, H., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II, LNCS, vol. 7475, pp. 214–238. Springer (2013)
18. Fawzi, H., Tabuada, P., Diggavi, S.: Secure estimation and control for cyber-physical systems under adversarial attacks. IEEE Trans. on Aut. Control 59(6), 1454–1467 (June 2014)
19. Forejt, V., et al.: Automated verification techniques for probabilistic systems. In: SFM. LNCS, vol. 6659. Springer (2011)
20. Garlan, D.: Software engineering in an uncertain world. In: Future of Software Engineering Research (FoSER). pp. 125–128 (2010)
21. He, K., Zhang, M., He, J., Chen, Y.: Probabilistic model checking of pipe protocol. In: Theoretical Aspects of Software Engineering (TASE). pp. 135–138 (2015)

22. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing - degrees, models, and applications. ACM Comput. Surv. 40(3) (2008)
23. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36 (2003)
24. Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D.: A classification of current architecture-based approaches tackling uncertainty in self-adaptive systems with multiple requirements. In: Managing Trade-offs in Adaptable Software Architectures. Elsevier (2016)
25. Ramirez, A.J., Jensen, A.C., Cheng, B.H.C.: A taxonomy of uncertainty for dynamically adaptive systems. In: 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS. pp. 99–108 (2012)
26. Schmerl, B.R., Cámara, J., Gennari, J., Garlan, D., Casanova, P., Moreno, G.A., Glazier, T.J., Barnes, J.M.: Architecture-based self-protection: composing and reasoning about denial-of-service mitigations. In: Symposium on the Science of Security, HotSoS. p. 2 (2014)
27. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B., Bruel, J.: Relax: Incorporating uncertainty into the specification of self-adaptive systems. In: Requirements Engineering Conference, 2009. RE '09. 17th IEEE International. pp. 79–88 (Aug 2009)