Science of Computer Programming 167 (2018) 51–69

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reasoning about sensing uncertainty and its reduction in decision-making for self-adaptation

Javier Cámara [a,*], Wenxin Peng [b], David Garlan [a], Bradley Schmerl [a]

[a] Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA
[b] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA

ARTICLE INFO

Article history:
Received 29 December 2017
Received in revised form 27 June 2018
Accepted 3 July 2018
Available online 17 July 2018

Keywords:
Adaptive system
Uncertainty
Uncertainty awareness
Uncertainty reduction
Decision making

ABSTRACT

Adaptive systems are expected to adapt to unanticipated run-time events using imperfect information about themselves, their environment, and goals. 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, together with uncertainty reduction mechanisms to improve system utility. We illustrate our approach on a Denial of Service (DoS) attack scenario and present results that demonstrate the benefits of uncertainty-aware decision-making in comparison to using an uncertainty-ignorant approach, both in the presence and absence of uncertainty reduction mechanisms.
© 2018 Elsevier B.V. All rights reserved.

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 [17,28] is regarded as an effective way to engineer systems that are *resilient* to run time changes despite of the different uncertainties present in their execution environment (e.g., system loads, resource availability, interaction with human actors), goals, or even in the system itself (e.g., the existence and location of faults).

Having inaccurate or missing information could lead a self-adaptive system to make bad decisions that make the system behave worse, rather than improve the system. However, despite the fact that these uncertainties can have a significant negative impact on run-time system behavior [26] (and ultimately, on goal satisfaction), many approaches to engineering self-adaptation do not explicitly represent uncertainty or consider it when deciding what actions to take in the system.

Moreover, for many systems there are tactics that could be employed by self-adaptive systems to reduce uncertainty so that they could make better decisions. For example, introducing CAPTCHA[1] into a web system could reduce uncertainty about whether clients are bots or humans and therefore might be part of a Distributed Denial of Service (DDoS) attack, and an autonomous drone could get closer to a target to get a better image to reduce uncertainty about potential targets. Without explicitly considering uncertainty, a self-adaptive system would not be able to reason about whether to use these tactics. Furthermore, these tactics often have associated costs and risks (for example, CAPTCHA can increase the annoyance of legitimate clients accessing the website, whose sessions are disrupted). Ignoring uncertainty prevents reasoning about

* Corresponding author.
E-mail addresses: jcmoreno@cs.cmu.edu (J. Cámara), wenxinp@google.com (W. Peng), garlan@cs.cmu.edu (D. Garlan), schmerl@cs.cmu.edu (B. Schmerl).

[1] 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).

https://doi.org/10.1016/j.scico.2018.07.002
0167-6423/© 2018 Elsevier B.V. All rights reserved.

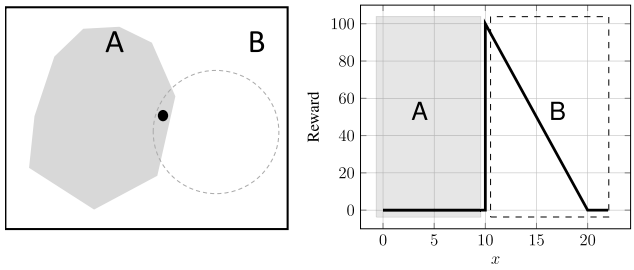


Fig. 1. Simple model scenario.

whether having the added information would be worth the cost. Hence, it is important to represent uncertainty to enable systems to reason about the trade-offs of enacting such tactics, quantifying the benefits of uncertainty reduction, and balancing them against associated costs when trying to achieve system goals.

One of the most popular patterns to build self-adaptation into software-intensive systems is IBM's MAPE-K [29], which integrates activities to **m**onitor, **a**nalyze, **p**lan, and **e**xecute adaptations in closed-loop control over a managed software system. Furthermore, a central **k**nowledge 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 [22,31,34], 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) introduced 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. Furthermore, we explore when uncertainty reduction tactics are worth using.

Concretely, we investigate three research questions: **(RQ1)** To what extent can explicit representation and reasoning about environment sensing uncertainty improve the quality of adaptation decisions?, **(RQ2)** Under what circumstances does environment sensing uncertainty awareness improve the quality of decisions?, and **(RQ3)** To what extent can environment sensing uncertainty reduction improve the quality of adaptation decisions both in the presence and absence of uncertainty awareness?

Uncertainty awareness targets a broad class of problem that is manifest in real software-intensive systems. For example, reconnaissance or delivery drones (with inaccurate or temporarily lost GPS) may need to get as close as possible to a *no-fly zone*[2] without violating it (e.g., to minimize the route distance to deliver a payload), and smart buildings in which sensors may be inaccurate or failure-prone need to minimize energy consumption without jeopardizing safety-critical system properties.

To illustrate our approach, consider the simple scenario shown in Fig. 1(a), which exhibits some of the main traits of the class of problems mentioned above. In this scenario, the system/environment state space is divided into regions A and B. While region A is to be avoided because we assume that it is out of the desirable operation range of the system, region B represents acceptable operating conditions. 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 within the dashed circle).

Fig. 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 collect reward 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 Fig. 1(b). When x is below 10, there is no reward; when x is equal to 10, the reward is maximum, and the reward decreases as x moves away from 10. Thus, the state space of this model is divided in two regions:

$$\textbf{Region A}: \; x < 10 \qquad\qquad \textbf{Region B}: \; x \geq 10$$

We assume 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

[2] https://www.faa.gov/uas/where_to_fly/no_drone_zone/.

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 a high penalty (due to the fact that no reward will be accrued).

To reduce such risks, decision making in self-adaptive systems needs to be able to identify regions of the state space where uncertainty has an impact and what to do in these cases. 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) [15]. The main idea behind the approach is analyzing the interplay of a self-adaptive system and its environment in a competitive game. System and environment are modeled as players whose behavior is independent (reflecting the fact that processes in the environment – in this case, sensing – cannot be controlled by the system). The relevance of our contribution resides in its potential for using our models in adaptive systems to make good decisions about (a) when to model uncertainty explicitly, and (b) when to take advantage of uncertainty reduction tactics.

In [12] we introduced a formal reasoning technique that improves decision making in regions of the state space in which uncertainty makes a difference. In this paper, we provide more details about this approach. Furthermore, we extend this technique to reason about when to reduce that uncertainty to improve self-adaptive system performance.

The remainder of this paper first presents some background on SMGs in Section 2. A description of our approach follows in Section 3. This section illustrates this approach on the simple scenario that we have introduced and provides some insights about the performance of uncertainty-aware and uncertainty-ignorant decision making, as well as about uncertainty reduction. Next, Section 4 introduces a more complex self-protecting systems scenario and discusses some results on comparing uncertainty-aware vs. uncertainty-ignorant decision making, as well as on uncertainty reduction both in presence and absence of uncertainty awareness. 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

Probabilistic model checking has been successfully used to analyze systems that operate subject to uncertainty by modeling them as probabilistic systems. It has been applied in a variety of application domains and scenarios that include security [19,35], communication protocols [27], and human-in-the loop adaptation [10] to enable quantitative reasoning about probability and reward-based properties (e.g., resource use, 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 on a recent technique for modeling and analyzing stochastic multiplayer games (SMGs) extended with rewards [15]. 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 a coalition to achieve a common goal, or competing to achieve their own goals. The algorithms used to compute these strategies are guaranteed to achieve optimal expected rewards for the kind of cumulative reward structures that we use in our models.[3]

In this paper, we illustrate our approach to modeling the SMG using the syntax of the PRISM language [30] for Stochastic Multiplayer Games and Markov Decision Processes (MDPs). A PRISM SMG model is built as a set of processes or *modules* (delimited by keywords module/endmodule, which are encoded as a set of commands:

$$[action]\ guard \rightarrow p_1 : u_1 + \cdots + p_n : u_n$$

Where *guard* is a predicate over the model variables (which can be either boolean or bounded-range integers, cf. Listing 2, line 3). Each update u_i describes a transition that the process can make (by executing *action*) if the guard is true. An update is specified by giving the new values of the variables, and has an assigned probability $p_i \in [0, 1]$. Multiple commands with overlapping guards (and probably, including a single update of unspecified probability) introduce local nondeterminism. An example of a simple module definition can be observed in Listing 2.

Moreover, the different player coalitions (player/endplayer keywords) also have to be specified in the model, indicating which players control the different processes and actions (when more than one process participate in the action). An example of such specification can be observed in Listing 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 [15] that is specifically tailored to checking properties in SMGs. 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, like ensuring that the probability of an event's occurrence or an expected reward measure meets some threshold.

[3] See Appendix A.2 in [15] for details.

Table 1
Example rPATL properties.

Formula	Description
$\langle\langle a, b\rangle\rangle P_{max \geq 0.9}$[G !critical]	"Players a and b have a strategy to ensure that the system will never experience a critical failure with at least probability 0.9"
$\langle\langle a\rangle\rangle P_{max=?}$[F success]	"Value of the maximum probability of eventual successful program termination that a strategy by player a can guarantee, independently of the strategies of other players."
$\langle\langle sys\rangle\rangle R^{utility}_{max=?}$[F end]	"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."

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 $P_{\bowtie q}$ and path formulae from PCTL [2]. Moreover, rPATL includes a generalization of the reward operator $R^{r}_{\bowtie x}$ from [24] to reason about goals related to rewards.

Some typical examples of properties that can be checked in rPATL can be observed in Table 1. The property on top shows boolean formula satisfiability, which checks the maximum probability of a joint player strategy satisfying some probability bound in an invariant expressed by a path formula similar to the ones found in standard CTL (globally, there is not a critical failure). The second property shows a quantification formula, which is denoted by a question mark in the quantifier, instead of the threshold used in boolean formula satisfiability. In addition to quantification, an additional difference with the first formula is that in this case, the property is about the strategy of a single player a, and the rest of the players of the game are considered adversarial. Moreover, the path formula is about a liveness property (eventual successful program termination).

The last property shows the use of the quantitative version[4] of the rPATL reward operator $\langle\langle C\rangle\rangle R^{r}_{max=?}[F\ \phi]$, which 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 players.

Our work builds on probabilistic model checking to analyze system performance (characterized in terms of utility) when reasoning explicitly about sensing uncertainty and uncertainty reduction.

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. We start by introducing the definition of the formal model for the game, including a description of how reward is collected. Next, we describe the analytical process followed to quantify the difference between uncertainty-aware and uncertainty-ignorant decision-making. Finally, we present and discuss the results obtained from analyzing our simple scenario in two different versions: without and with uncertainty reduction. The reason for this two-stage presentation is twofold. First, we show that uncertainty-awareness in decision-making can have benefits on its own, independently of whether uncertainty reduction mechanisms are available to the system. Second, this simplifies the presentation and avoids conflating uncertainty reduction and uncertainty awareness, which are different concepts that do not necessarily coexist in systems.

3.1. Formal model definition

In this section, we introduce a model that captures the simple scenario described in the introduction. The purpose of the model is to compare uncertainty-aware adaptation, i.e., decision-making that considers explicitly uncertainty information (in this case stemming from 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 [14], 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 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 in performing actions. As shown in Listing 1, the turn is controlled by the global variable turn. There are two other global variables: real_x represents the *actual* value of x at a given time (this variable is initialized by constant INIT_X, line 5), whereas obs_x represents the value of x *observed* by the system (i.e., the value communicated by the inaccurate sensor to the system). In player definitions shown in lines 1 and 2, labels represent processes, as well as actions (between brackets) under the control of the player.

[4] The reward quantifier can also be used for boolean formula satisfiability.

Listing 1 Player definition.

```
1  player sys target_system, [act], sensor, [sense] endplayer
2  player env environment, [generate] endplayer
3  const ENV_TURN, SYS_TURN, INIT_X; // Turn and x initialization constants
4  global turn:[ENV_TURN..SYS_TURN] init ENV_TURN; // Used to alternate between players
5  global obs_x, real_x:[0..20] init INIT_X;    // Observed and actual values of x
```

The game is played in alternating turns by the system and the environment players. In the first turn, the actual value of x (real_x) is initialized to INIT_X. Then, a typical cycle of the game works in the following way:

1. The environment process checks that the maximum number of turns has been exceeded, and if that is not the case, yields the turn to the system player (Listing 2, line 4).

Listing 2 Simple environment model definition.

```
1  const MAX_TURNS;
2  module environment
3      t:[0..MAX_TURNS] init 0;
4      [generate] (t<MAX_TURNS) & (turn=ENV_TURN) -> (t'=t+1) & (turn'=SYS_TURN);
5  endmodule
```

2. The system senses the value obs_x (Listing 3, line 2). The uncertainty in the sensing process is modeled by a simple probability distribution. In this case we use 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).

Listing 3 Sensor definition.

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

3. After obtaining the observed value obs_x, the system (Listing 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 impact that the system's actuator can make on the qualities of the system (in this case, on the value of x). For example, if step = 3, s_step can take values in $\{1, 2, 3\}$.

Listing 4 Simple system model definition.

```
1  const step;
2  formula s_step = obs_x-step >= 10 | obs_x < 10 ? step : obs_x - 10;
3  // s_step is the saturated value of step (decrement/actuation on variable x): it is set to the minimum of (step, x-10)
4
5  module target_system
6  expected_x:[0..20] init 0;
7  new_info:[0..1] init 0;
8      [sense] (new_info=0) & (turn=SYS_TURN) -> (new_info'=1);     // 2. Sense
9      [act](new_info=1) & (turn=SYS_TURN) ->                        // 3.a. Act
10         (real_x'=real_x-s_step>=0?real_x-s_step:0) & (new_info'=0) &
11         (expected_x'=obs_x-s_step>=0?obs_x-s_step:0) & (turn'=ENV_TURN);
12     [](new_info=1)&(turn=SYS_TURN) ->                          // 3.b. Do nothing
13                 (expected_x'=obs_x) & (turn'=ENV_TURN) & (new_info'=0);
14 endmodule
```

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 environ-

ment, and then the system play one turn each). This is achieved by setting the constant MAX_TURNS = 1 in our game model.

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

Listing 5 Simple model reward structure definition.

```
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
```

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:

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

2. $R_{u\text{-}ignorant}$: The maximum utility that adaptation is able to obtain without factoring in 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):

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

   (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. $R_{u\text{-}aware}$: 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 5 (line 5), which makes use of the corresponding formula defined in line 3.

$$\langle\langle sys\rangle\rangle R^{rEU_uncertain}_{max=?}[F\ t = MAX_TURNS] \tag{3}$$

   (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.

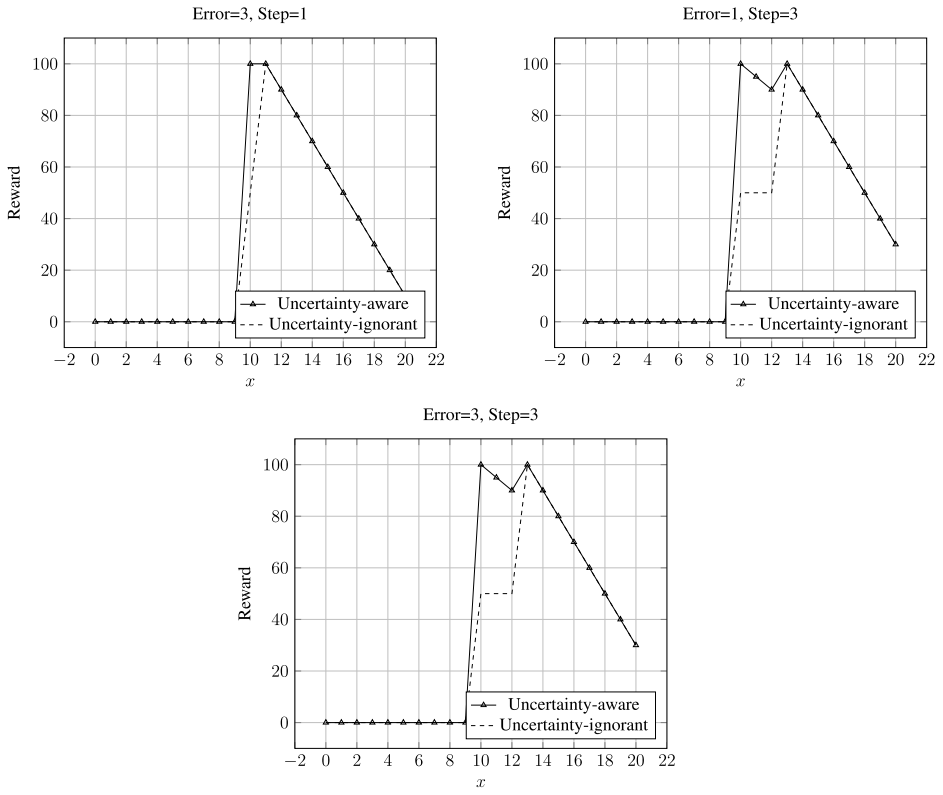


Fig. 2. Simple model scenario results.

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 compares the reward obtained by uncertainty-aware and uncertainty-ignorant adaptation (i.e., $R_{u\text{-}aware}$ and $R_{u\text{-}ignorant}$, respectively). In the figure, x indicates the value of the variable before actuation. 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 B ($x \geq 10$), 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 uncertainty-ignorant adaptations will choose to reduce the value of x to obtain more reward. This can be observed in Fig. 2, in which the reward obtained by both adaptation variants get closer in value as x moves to higher values, away from the threshold $x = 10$. Similarly, when the system is in region A ($x < 10$) 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. Fig. 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 Fig. 2 (left) shows that there is little performance difference between the two

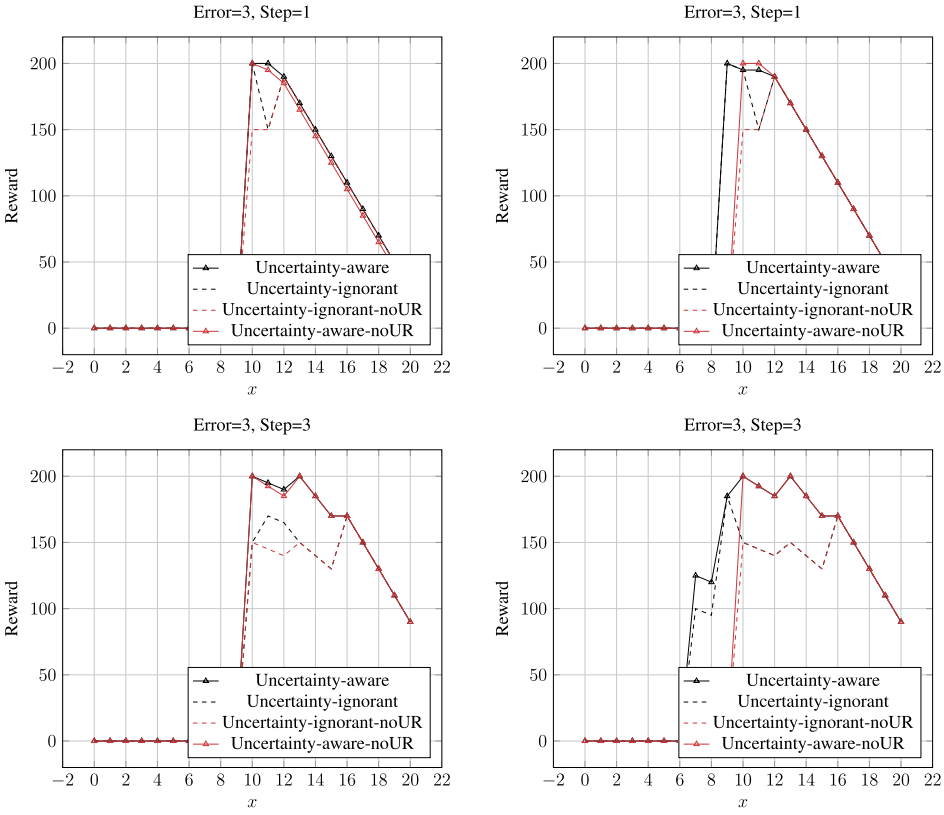


Fig. 3. Simple model with uncertainty reduction: results with no cost for uncertainty reduction (left) and cost associated to increasing the value of x (right).

variants of adaptation. This is because, even if uncertainty-ignorant adaptation makes the wrong choice, e.g., when $x = 12$, the actuator can at most reduce x to 11, incurring only a light penalty. 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. Moreover, it is worth noticing that higher error for the same level of actuation does not necessarily mean that the outcome is always going to be worse. Once the error threshold necessary to take the wrong decision is crossed, the decision (and its outcome) are not going to change. This can be observed if we contrast the results of the center and right plots on the figure.

Experiments and observations for the simple scenario with uncertainty reduction. In the prior set of experiments, the self-adaptive system only had the option of making a more conservative choice in regions in which uncertainty made a difference. Often, there are opportunities to obtain more information to reduce that uncertainty (e.g., adding more probes, or increasing the rate at which probes collect information). To explore this situation, we extend our simple scenario model above with a tactic for uncertainty reduction that eliminates the uncertainty in the sensor. In the new model, before the execution of the uncertainty reduction tactic, the probability distribution of observing the real value of x remains the same as in the original model, i.e., $P(\text{real_x} = \text{obs_x}) = 0.5$, and $P(\text{real_x} = \text{obs_x} + \text{error}) = 0.5$. However, when the tactic is executed, the observed value of x (obs_x) becomes equal to the real value (real_x), i.e., $P(\text{real_x} = \text{obs_x}) = 1$, and $P(\text{real_x} = \text{obs_x} + \text{error}) = 0$. We also extend our game model to be played in two turns, instead of one, so that the uncertainty reduction tactic can be used and its benefits exploited in the next turn.

Fig. 3 shows the analytical results of our simple scenario with uncertainty reduction. The plots on the left show the results of scenarios in which there is no cost associated with side effects of executing the uncertainty reduction tactic. In contrast, the plots on the right show the results for scenarios in which the execution of the uncertainty reduction tactic causes the value of variable x to be incremented by step. This increment results in reward loss (note that reward is maximum when $x = 10$, and it decreases as the value of x increases). This side effect models potentially adverse side effects that uncertainty reduction tactics may cause in real scenarios (e.g., reducing uncertainty about potentially malicious clients by executing a CAPTCHA disrupts the experience of legitimate users accessing a web infrastructure). All four plots also show the results without uncertainty-reduction tactics both for uncertainty-aware and uncertainty-ignorant decision-making to provide better context. The results in the figure reveal the following:

1. *Uncertainty-aware adaptation always performs as well as, or even better than, uncertainty-ignorant adaptation.* Independently of whether there is uncertainty reduction available. This observation is in line with our expectations, and is consistent with observations (1) and (2) described in the results for the simple scenario without uncertainty reduction.
2. *Uncertainty-aware adaptation can exploit uncertainty-reduction to its advantage*. We can observe in the two plots on the right how uncertainty-aware adaptation improves with uncertainty reduction when it gets close to the boundary between regions A and B. This is also aligned with our expectations, since uncertainty awareness enables decision-making to factor in the benefits of reducing uncertainty in the overall outcome of the game.
3. *Uncertainty reduction is ignored by uncertainty-ignorant adaptation when there is a cost to it*. When there is a cost, the uncertainty reduction tactic is not selected because uncertainty-ignorant decision-making cannot detect the advantage of reducing uncertainty. This can be observed in the bottom-right plot vs. the bottom-left plot in which there is a positive delta in reward for uncertainty-ignorant adaptation combined with uncertainty reduction for $x \in \{10, 13\}$. This result is analogous to prior work we carried out on latency-aware adaptation [11]. When comparing latency-aware adaptation with latency-ignorant adaptation, we observed that ignoring latency information disabled the ability to select low-latency tactics because they were perceived as less beneficial than others with higher or equal impact on quality, but higher latency. In the same way, ignoring uncertainty masks the potential benefits of uncertainty reduction tactics.
4. *Uncertainty reduction does not always improve uncertainty-aware adaptation*. This can be explained by the fact that uncertainty-awareness already accounts for the uncertainty, so reducing uncertainty may not always be worth the cost it entails, when compared with the reward benefit that the reduced uncertainty can provide. Actually, it is worth noting that uncertainty-aware adaptation with uncertainty reduction can do slightly worse than without it in specific cases. In particular, when there is cost for uncertainty reduction (top, right plot) there is an area in which there is some chance of using uncertainty reduction, incur the cost, but not obtaining any benefit because the actuation impact step is too small to incur in the 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 [35]. 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, and applying tactics like blocking their requests for accessing the website, could be detrimental to 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 understanding of the DoS scenario, we structure our model in a similar way to the simple model described earlier and make the following assumptions:

1. *The space is divided into two regions:* (i) DoS, in which we assume that the system is experiencing an attack, and (ii) Non-DoS, in which any anomalous activity that the system may experience is not related to a DoS attack (e.g., the system is experiencing low response due to a normal high-load environment).
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 is an abstract metric used as a 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 Non-DoS region.
3. *The system does not know the real value of the variables being measured.* 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 for instance, from historical data.

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 our simple synthetic scenario, 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 in two categories:

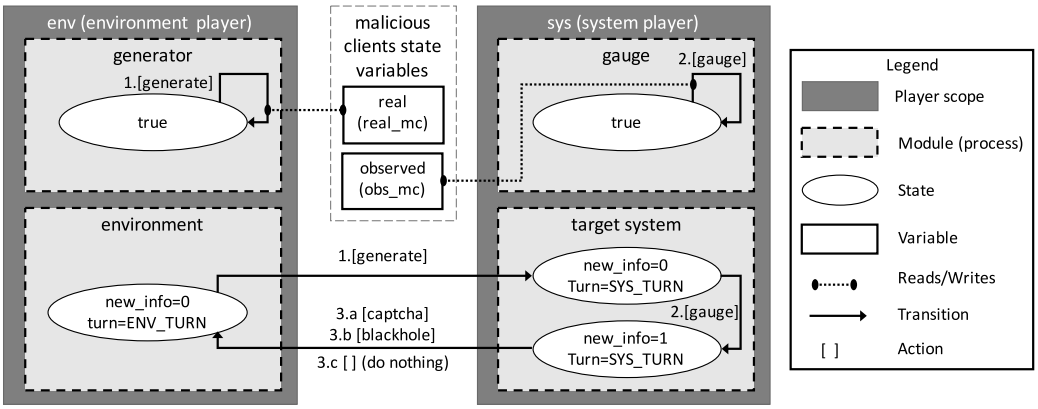


Fig. 4. DoS scenario game model overview.

- *Tactics that reduce uncertainty*. 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 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 challenge they are presented with.
- *Tactics that do not reduce uncertainty*. 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).

- *Reward Model.* In the simple scenario, the reward was related to only one dimension. However, in this scenario there are two dimensions of concern: *security* and *user experience*. We assume them to be of equal importance, although this is not necessary for our model. Security is directly related to the metric percentage of malicious clients mc (lower is better). User experience is also affected by the 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. Note that blackholing clients may also have detrimental side effects when it blocks legitimate traffic by mistake and leads to bad user experience. This is more likely to happen when there is high uncertainty about the maliciousness of the clients accessing the system.

4.1. Formal model definition

This section provides a high-level description of the game model for the DoS scenario.[5] The scenario is modeled as a stochastic game involving two players that represent the system (sys) and the environment (env) (Fig. 4):

- 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 [gauge].
- The environment player consists of the generator and environment modules, which are synchronized via the [generate] shared action.

When the game starts, the environment player first generates the real value of the mc ([generate]), and it yields the turn to the system player. Next, the system player gauges mc ([gauge]), producing its observed value. The system player then infers the region of the state space (DoS or Non-DoS) 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.

Defining players and global game variables. The game contains three global variables: (i) real_mc is real percentage of malicious clients, ranging from [0,100], represents the real system metric, (ii) obs_mc is the observed percentage of malicious clients, and (iii) std_mc (or σ_{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. Listing 6 shows the definition of players, turns, and global variables.

[5] A full listing of the model is available in [5].

Listing 6 Player definition.

```
player sys target_system, [blackhole], gauge, [gauge], [captcha] endplayer
player env generator, [generate], environment endplayer
const ENV_TURN=1, SYS_TURN=2;
global turn:[ENV_TURN..SYS_TURN] init ENV_TURN; // Used to alternate between players
global std_mc : [0..100];   //Observed standard deviation of malicious clients
global obs_mc:[0..100];     //observed percentage of malicious clients
global real_mc:[0..100];    //real percentage of malicious clients
```

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 (Listing 7).[6]

Listing 7 Module definition for generator.

```
module generator
    [generate] true->(real_mc'=real_mc);
endmodule
```

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:

$$P(\text{obs_mc} = y|\text{real_mc} = x) = \frac{1}{\sigma_{mc}\sqrt{2\pi}} e^{-(x-y)^2/2\sigma_{mc}^2}$$

This probability density function is actually a conditional probability distribution of the observed value of the metric, given a real value, i.e., $P(\text{obs_mc}|\text{real_mc})$. In this scenario, we encode this function using six points to simulate this normal distribution because the SMG formalism does not support the specification of continuous probability distributions (Listing 8).

Listing 8 Module definition for gauge.

```
module gauge
        [gauge] true->0.34:(obs_mc' = (real_mc+1*std_mc<=100?real_mc+1*std_mc:100))+
                      0.34:(obs_mc' = (real_mc-1*std_mc>=0?real_mc-1*std_mc:0))+
                      0.14:(obs_mc' = (real_mc+2*std_mc<=100?real_mc+2*std_mc:100))+
                      0.14:(obs_mc' = (real_mc-2*std_mc>=0?real_mc-2*std_mc:0))+
                      0.02:(obs_mc' = (real_mc+3*std_mc<=100?real_mc+3*std_mc:100))+
                      0.02:(obs_mc' = (real_mc-3*std_mc>=0?real_mc-3*std_mc:0));
endmodule
```

Selecting 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 adaptive 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 of 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 that obs_mc = real_mc.
5. eua_dos: Expected user annoyance after practicing a tactic if the system is currently at the DoS region.

[6] Note that in this game, we assume that the time frame during which the scenario executes is relatively short, and therefore we abstract any independent evolution of the environment variables (e.g., the value of real_mc is modified only by system tactics, once the initial conditions of the game are set). However, this does not constitute a fundamental limitation of the approach, which can be extended to relax this assumption [32].

Table 2
Simple impact specification of tactics in a DoS adaptation scenario.

real_mc	Non-DoS	DoS
[captcha]	−10	−30
[blackhole]	−30	−30

ua	Non-DoS	DoS
[captcha]	+10	+10
[blackhole]	+50	+10

6. eua_normal: Expected user annoyance after executing a tactic if the system is currently at the Non-DoS 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.

Executing Tactics. Table 2 summarizes the effect of exercising different tactics at different system states. For example, blackholing in both regions (DoS and Non-DoS) 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[7]).

Listing 9 shows how the tactic [blackhole] updates the six variables discussed the previous section. Lines 1–2, 4–5, and 7–8 show the encoding of the formulas that update all relevant state variables for malicious clients and user annoyance.

All the formulas in these lines follow a common pattern, and are intended to simply decrease the value of a variable by some delta. The extra code only checks that the updated value is saturated and does not go beyond the range of possible values specified by the variable (numerical values in PRISM-games are always bounded and violating their range in updates results in failures in model checking). For instance, the formula bh_f_mc in line 1 can be interpreted as:

```
If (real_mc+bh_mc_delta >= 0) then
    If (real_mc+bh_mc_delta <= 100) then
        bh_f_mc = real_mc+bh_mc_delta
    else
        bh_f_mc = 100// Value saturated to maximum in range
else
    bh_f_mc = 0 // Value saturated to minimum in range
```

In the formula, bh_mc_delta is the value for the delta for tactic blackhole on real_mc in Table 2 (which in this case is negative).

Then, ines 12–14 show how the different variables are updated based on the aforementioned formulas when it is the turn of the system, and there is new information available from the gauge.

Listing 9 blackhole tactic definition.

```
1  formula bh_f_mc = real_mc+bh_mc_delta >= 0? (real_mc+bh_mc_delta<=100? real_mc+bh_mc_delta :100):0;
2  formula bh_f_emc = obs_mc+bh_mc_delta >= 0? (obs_mc+bh_mc_delta<=100? obs_mc+bh_mc_delta : 00):0;
3
4  formula bh_f_ua_non_dos = ua+50 >=0 ? (ua+50<=100? ua+50 : 100) : 0;
5  formula bh_f_ua_dos = ua+10 >=0 ? (ua+10<=100? ua+10 : 100) : 0;
6
7  formula bh_f_ua = (real_mc>dos_threshold)?bh_f_ua_dos:bh_f_ua_non_dos;  // Real user annoyance
8  formula bh_f_eua = (obs_mc>dos_threshold)?bh_f_ua_dos:bh_f_ua_non_dos; // Expected user annoyance
9  ...
10 module target_system
11 ...
12 [blackhole] (new_info=1) & (turn=SYS_TURN)-> (real_mc'=bh_f_mc) & (emc' = bh_f_emc)
13                          & (ua'=bh_f_ua)& (eua'=bh_f_eua) & (eua_dos'=bh_f_ua_dos)
14                          & (eua_non_dos'=bh_f_ua_non_dos) & (turn'=ENV_TURN) & (new_info'=0);
15 ...
16 endmodule
```

[7] In this model, we assume that the effect of tactics is deterministic, although more sophisticated non-deterministic effects of tactics can be incorporated using languages like the one described in [9].

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 (Listing 10, lines 1–2). We employ three types of rewards to analyze uncertainty-aware and uncertainty-ignorant decision-making.

1. rIU (real utility, lines 3–5): This reward is calculated based on utility metrics based on the real value of the percentage of malicious clients (real_mc) and user annoyance (formulas uM and uA, defined in Listings 11 and 12, respectively).
2. rEIU (expected utility for uncertainty-ignorant decision making, lines 7–9): Is calculated based on the observable information about the percentage of malicious clients (obs_mc) employing formulas EuM and EuA (defined in Listings 11 and 12, respectively) and it is unaware of the uncertainty in sensing.
3. rEIU_uncertain (expected utility for uncertainty-aware decision making, lines 11–17): 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 encoded in Listing 10 (lines 12–17). The value of this reward is computed using the six utility functions that calculate utility based on each of the possible values of mc that correspond to the points of the discrete probability distribution (EuM_pos_X, where X is the number of the point in the discrete probability distribution, ranging from one to six – Listing 11, lines 18–22).

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 | 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:

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

Listing 10 Reward functions.

```
1  const double wA=0.5; // Weight for user annoyance utility
2  const double wM=0.5; // Weight for security (malicious clients level) utility
3  rewards "rIU" // Real Instantaneous utility reward
4    (turn=ENV_TURN) & (t>=1) : wM*uM + wA*uA;
5  endrewards
6
7  rewards "rEIU" // Expected Instantaneous utility reward, uncertainty-ignorant
8     (turn=ENV_TURN) & (t>=1) : wM*EuM + wA*EuA;
9  endrewards
10
11 rewards "rEIU_uncertain" // Expected Instantaneous utility reward, uncertainty-aware
12    (turn=ENV_TURN) & (t>=1) :0.34*(wM*EuM_pos_one+wA*ua_pos_one)+
13                    0.14*(wM*EuM_pos_two+wA*ua_pos_two)+
14                    0.02*(wM*EuM_pos_three+wA*ua_pos_three)+
15                    0.34*(wM*EuM_neg_one+wA*ua_neg_one)+
16                    0.14*(wM*EuM_neg_two+wA*ua_neg_two)+
17                    0.02*(wM*EuM_neg_three+wA*ua_neg_three);
```

Listing 11 shows the utility profile for security, based on the values of the metric for client maliciousness. Moreover, Listing 12 shows the utility profile for user experience (calculated based on user annoyance ua) and security (calculated based on percentage of malicious client). Both profiles are defined using an encoding similar to the ones described in [35], where a detailed discussion of their rationale can be found. Table 3 describes the set of points used to define the utility functions as a simple linear interpolation.

In Listing 11, formula uM (lines 1–6) encodes the contribution to utility of the level of malicious clients (lower is better) based on the real system metric real_mc. The formula encodes a simple linear interpolation function using the points defined on the left column of Table 3. In contrast, formula EuM (lines 9–13) encodes contribution to utility in a similar way, but in

Table 3
Utility functions for DoS.

U_M	U_A
0 : 1.00	0 : 1.00
5 : 1.00	100 : 0.00
20 : 0.80	
50 : 0.40	
70 : 0.00	

this case to the version of expected utility based on the expected value of the level of malicious clients emc, which assumes that obs_mc=real_mc.

Listing 11 Client maliciousness utility calculation (excerpt).

```
1  // 'Maliciousness' utility function based on real_mc
2  formula uM = (real_mc>=0 & real_mc <=5? 1 : 0)
3          +(real_mc>5  & real_mc <=20? 1+(0.80-1)*((real_mc-5)/(20-5)) :0)
4          +(real_mc>20 & real_mc <=50? 0.80+(0.40-0.80)*((real_mc-20)/(50-20)) :0)
5          +(real_mc>50 & real_mc <=70? 0.40+(0.00-0.40)*((real_mc-50)/(70-50)) :0)
6          +(real_mc>70 ? 0:0);
7
8  // 'Maliciousness' utility function based on emc (uncertainty-ignorant)
9  formula EuM = (emc>=0 & emc <=5? 1 : 0)
10         +(emc>5  & emc <=20? 1+(0.80-1)*((emc-5)/(20-5)) :0)
11         +(emc>20 & emc <=50? 0.80+(0.40-0.80)*((emc-20)/(50-20)) :0)
12         +(emc>50 & emc <=70? 0.40+(0.00-0.40)*((emc-50)/(70-50)) :0)
13         +(emc>70 ? 0:0);
14
15 //Derive 6 possible mc for uncertainty-aware decision making
16 formula pos_one_std =   obs_mc +1*std_mc<=100?obs_mc +1*std_mc:100;
17 ...
18 formula EuM_pos_one = (pos_one_std>=0 & pos_one_std <=5? 1 : 0)
19         +(pos_one_std>5  & pos_one_std<=20? 1+(0.80-1)*((pos_one_std-5)/(20-5)) :0)
20         +(pos_one_std>20 & pos_one_std <=50? 0.80+(0.40-0.80)*((pos_one_std-20)/(50-20)) :0)
21         +(pos_one_std>50 & pos_one_std <=70? 0.40+(0.00-0.40)*((pos_one_std-50)/(70-50)) :0)
22         +(pos_one_std>70 ? 0:0);
```

Finally, for uncertainty-aware utility calculation, we derive the six possible values of mc that correspond to the points in the discrete probability distribution that we employ to model sensing error. Each of these is computed based on the formulas EuM_pos_X_std (line 16 shows the encoding for position one), where X is the number of the point in the distribution. The corresponding utility contribution based on that value of mc is encoded in functions EuM_pos_X (lines 18–22 shows the encoding for position one).

In Listing 12 we show the encoding of the user annoyance contribution to utility. Functions uA and EuA in lines 1–2 compute utility for the real (ua) and observed (eua) values of ua, respectively (let us recall that eua corresponds to the expected user annoyance after executing a tactic assuming obs_mc=real_mc).

Listing 12 User annoyance utility calculation.

```
1  //User Annoyance utility function
2  formula uA = (ua>=0 & ua <=100? 1-(ua/100):0);
3  formula EuA = (eua>=0 & eua <=100?  1-(eua/100):0);
4  formula EuA_non_dos = (eua_non_dos>=0 & eua_non_dos <=100?  1-(eua_non_dos/100):0);
5  formula EuA_dos = (eua_dos>=0 & eua_dos <=100?  1-(eua_dos/100):0);
```

Reducing uncertainty. To reduce uncertainty, we incorporate the tactic [captcha] into the target_system module (Listing 13). The tactic can be fired only once (the tactic command is guarded by the predicate !captcha_enabled, which becomes false once the tactic is fired) and has the effect of setting the value of σ_{mc} to zero, so effect in practice is that the sensor "becomes accurate". Lines 1–8 encode the updates to all of the real and observed state variables for malicious clients and user annoyance (the value of the updates are based on the impacts of tactics specified in Table 2), and employ a similar pattern to the one described for the tactic blackhole in Listing 9.

Listing 13 captcha tactic definition.

```
1  formula c_f_mc = real_mc+c_mc_delta >=0 ? (real_mc+c_mc_delta<=100? real_mc+c_mc_delta : 100) : 0;
2  formula c_f_emc = obs_mc+c_mc_delta >=0 ? (obs_mc+c_mc_delta<=100? obs_mc+c_mc_delta : 100) : 0;
3
4  formula c_f_ua_normal = ua+60 >=0 ? (ua+60<=100? ua+60 : 100) : 0;
5  formula c_f_ua_dos = ua+30 >=0 ? (ua+30<=100? ua+30 : 100) : 0;
6
7  formula c_f_ua = (real_mc>dos_threshold)?c_f_ua_dos:c_f_ua_normal;
8  formula c_f_eua = (obs_mc>dos_threshold)?c_f_ua_dos:c_f_ua_normal;
9  ...
10 [captcha] (new_info=1) & (turn=SYS_TURN) & (!captcha_enabled)-> (real_mc'=c_f_mc) & (emc' = c_f_emc)
11                        & (ua'=c_f_ua)& (eua'=c_f_eua)
12                        & (eua_dos'=c_f_ua_dos)&(eua_non_dos'=c_f_ua_non_dos) & (std_mc'=0)
13                        & (turn'=ENV_TURN)&(new_info'=0) & (executed'=1) & (captcha_enabled'=true);
```

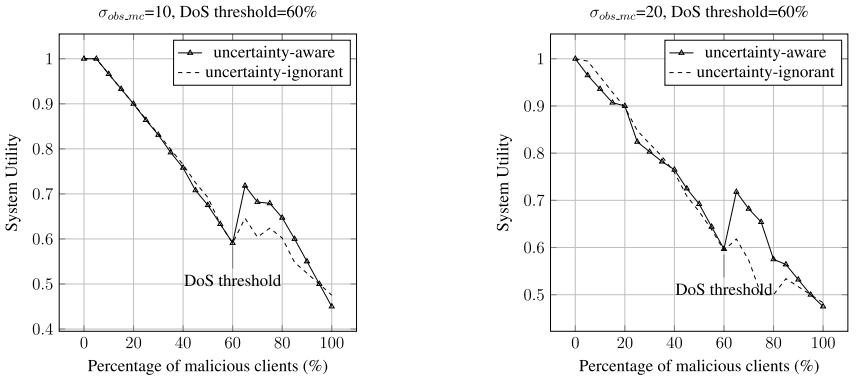


Fig. 5. DoS scenario results without uncertainty reduction.

4.2. Experiments and observation

We divide this section in two blocks. The first one explores the results of the DoS scenario without uncertainty reduction (just [blackhole] tactic), whereas the second part discusses the DoS scenario in an extended version in which there is an additional [captcha] tactic available that reduces the uncertainty in the percentage of malicious clients observed in the system.

DoS without uncertainty reduction. To compare decision-making that incorporates uncertainty-awareness, with uncertainty-ignorant adaptation, we use the analytical process described in Section 3. For our experiment, we employed a DoS game model on which we only considered the non-uncertainty reduction tactic (*blackhole*), 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 $\sigma_{\mathsf{mc}} \in \{10, 20\}$. The range explored for mc is $\{0, \ldots, 100\}$.

Fig. 5 shows the result of our experiments. From these results, we can draw the following observations[8]:

1. ($\neg$RB $\Rightarrow$ UA=UI) *When far from the region boundary, uncertainty does not matter.* When the value of mc is in region DoS or Non-DoS (mc ≥ 60), and moves away from the threshold, the utility obtained both by uncertainty-aware and ignorant adaptations is similar.
2. (RB $\Rightarrow$ UA$>$UI) *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 Non-DoS, 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. 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 making a decision.
3. ($\sigma \propto$ UA-UI) *The difference between adaptation approaches is greater when standard deviation of the sensed value of the variable 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_{mc} is twice the one on the left.

Our formal model allows us to see that these observations are consistent with the results obtained in our simple scenario, reinforcing the evidence indicating that uncertainty awareness can make a difference in the boundary between regions of the state space in which system dynamics change.

DoS with uncertainty reduction. We now compare the performance of uncertainty-aware and uncertainty-ignorant decision-making in the presence of uncertainty reduction. To carry out our experiments, we considered a model in which the uncertainty reduction tactic [captcha] is used along with the [blackhole] tactic. The parameters for our experiment are mc $\in \{0, \ldots, 100\}$, $\sigma_{\mathsf{mc}} = 20$, and a DoS threshold $\in \{40, 60\}$.

[8] We provide a synthetic notation to facilitate the intuition behind every observation. RB denotes region boundary, UA and UI are uncertainty-aware and uncertainty-ignorant adaptation, σ denotes standard deviation, $\propto$ denotes direct proportionality, and UR stands for uncertainty reduction.

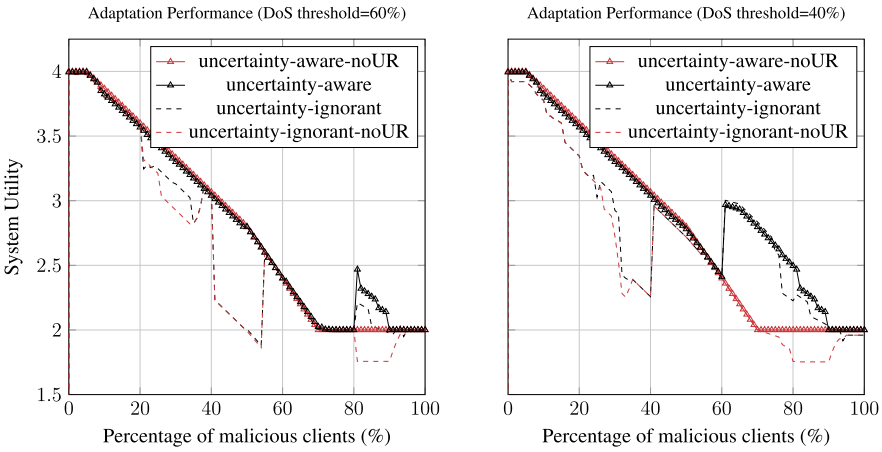


Fig. 6. DoS scenario results with uncertainty reduction.

Fig. 6 shows the analytical results of our DoS scenario with uncertainty reduction. The plot on the right shows the results for DoS threshold values of 60%; the left plot shows it at 40%. The plots also show the results without uncertainty reduction tactics both for uncertainty-aware and uncertainty-ignorant decision-making to provide better context. The results in the figure reveal the following:

1. (UA+UR≥UA) *Uncertainty-aware adaptation with uncertainty reduction always performs as well as, or better than without it.* This observation is in line with our expectations.[9] Since uncertainty awareness enables strategy synthesis to discriminate under which cases it might be beneficial to reduce uncertainty to obtain other benefits later on, uncertainty-aware adaptation with uncertainty reduction is expected to perform at least as well as without it, and better if the conditions allow it. We can observe that this is the case in both plots, in which uncertainty-aware decision making with reduction performs better than without it in ranges that correspond to the DoS region (as we might expect), in which $mc \in \{80, \ldots, 90\}$ for DoS threshold 60%, and $mc \in \{60, \ldots, 90\}$ for DoS threshold 40%.
2. (UI+UR≥UI) *Uncertainty-ignorant adaptation with uncertainty reduction also performs as well as, or better than without it.* This observation is also in line with our expectations. Although uncertainty-ignorant adaptation is not equipped to reason about the impact of uncertainty on goal satisfaction, adaptation choices can coincide with the ones present in optimal strategies, even if the selection process has been different. We can observe an instance of this situation in the right-hand side plot of the figure, in which uncertainty-ignorant adaptation with uncertainty reduction performs similarly to the uncertainty-aware variant with uncertainty reduction, incorporating the [captcha] tactic in the strategies used in the range 60–75% of mc.
3. (UA+UR≥UI+UR) *Uncertainty-aware adaptation always performs equally or better than uncertainty-ignorant adaptation when they both use uncertainty reduction.* This is also consistent with our expectations. While the occasional improvement experienced by uncertainty-ignorant adaptation derived from uncertainty reduction is purely coincidental, uncertainty-awareness enables the system to fully exploit the benefits of uncertainty reduction and avoids making the wrong trade-offs. We can observe this in both plots (ranges 75–90% in the right-hand side plot, and 80–90% on the left), in which uncertainty-ignorant adaptation cannot fully exploit the benefits of uncertainty reduction, compared to the uncertainty-aware variant.
4. (UI+UR≥UA) *Uncertainty-ignorant adaptation with uncertainty reduction can perform better than uncertainty-aware adaptation without uncertainty reduction.* As we might expect, the benefits of uncertainty awareness do not always surpass those of a richer adaptation tactic repertoire. We can observe this in our experiments, in which the solution space for uncertainty-aware adaptation without uncertainty reduction is more constrained than the space of uncertainty-ignorant adaptation with uncertainty reduction. We can observe instances of this situation in the ranges 60–90% and 80–86% of the right and left-hand side plots, respectively, in which the optimal strategy incorporates [captcha] (when it is available).

In summary, our results show that both uncertainty-aware and uncertainty-ignorant adaptation can benefit from uncertainty reduction, and they always perform equally or better than without it. However, uncertainty-ignorant adaptation improvements with uncertainty reduction are purely coincidental. In contrast, uncertainty-aware adaptation improves consistently in the presence of uncertainty reduction. Our formal model allows us not only to demonstrate these (perhaps

[9] Note that in our results, we always refer to the capacity of improving performance in decision making. Actual improvement on system implementations might depend on other factors like the size of the state space (which might prevent timely run-time reasoning).

intuitive) results, but also to quantify the benefits and provide a basis for reasoning about when and when not to reduce uncertainty.

5. Related work

Uncertainty management has been studied by many authors in the field of self-adaptive systems, but not 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 [1,21,36]. Other approaches employ probabilistic verification and estimates of the future environment and system behavior for optimizing the system's operation. These proposals target mitigation of uncertainty due to *parameters over time* [3,4,20].

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 [23] 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 [18] 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 [16] 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.

Researchers in artificial intelligence have been working on approaches for reasoning under uncertainty for a very long time, compared to other areas like self-adaptive systems or CPS. There is a wealth of paradigms, algorithms, and techniques that is too vast to summarize in this section. However, notable approaches to reasoning under uncertainty include the use of probabilistic planners [25] that deal with aleatoric uncertainty and even epistemic uncertainty via reasoning about partial observability [13].

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 [11] 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. A general description of the SMG-based analysis technique for the interplay of a self-adaptive system and its environment is provided in [6]. None of the works mentioned above can reason explicitly about the aleatoric uncertainty (or its reduction) in the information that the system employs for decision making.

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. The technical novelty of our approach resides in the definition of patterns to systematically encode environment sensing uncertainty in adaptation scenarios, and an analysis method to explore the solution space while factoring in such uncertainty. This ability shows potential for using our models in adaptive systems to make good decisions about (a) when to model uncertainty explicitly, and (b) when to take advantage of uncertainty reduction tactics. Our results show that although uncertainty-aware adaptation does not guarantee 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). With respect to uncertainty reduction, our results show that both uncertainty-aware and uncertainty-ignorant adaptation can benefit from it, and they always perform equally or better than without it (RQ3). Moreover, uncertainty-ignorant decision-making improvements with uncertainty reduction are purely coincidental and never surpass the performance of uncertainty-aware adaptation, which in contrast improves consistently in the presence of uncertainty reduction (RQ3). These are relevant findings, 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, and incorporating uncertainty reduction mechanisms, if available.

With respect to future work, we plan on extending decision-making under uncertainty to reason with only partial knowledge about the uncertainty function, and study the impact on accuracy of the discretization of the probability distribution of sensing (which currently is sampled using six points). The current version of our approach assumes that information about the probability distribution for sensing ($P(observed\ value \mid real\ value)$) 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 of estimating the real state of the system, e.g. by automatically refining throughout subsequent system executions the knowledge that the system has about $P(real\ value \mid observed\ value)$. The second avenue for future work will explore ways to identify regions of the state space where uncertainty has an impact, and hence reduce the search space and overhead of reasoning under uncertainty by exploiting the fact that in some regions of the state space, explicit modeling and management of uncertainty matter more than in other regions in which uncertainty awareness and reduction are not able to improve decision making. We also plan to 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 [10,7].

Finally, although there is no fundamental limitation to carrying out run-time decision-making using our technique, the feasibility of timely decision-making depends on the specific case study and the computational cost of analyzing the associated state space. Previous experiences in developing techniques for decision-making in self-adaptation using probabilistic model checking of MDP and SMG have shown feasibility in systems similar to the one described in our DoS case study [8, 32]. In any case, we plan to explore stochastic dynamic programming techniques to improve scalability. Our previous experience has shown that decision time can be reduced by one order of magnitude with respect to standard probabilistic model checking in proactive self-adaptation scenarios [33].

Acknowledgements

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] L. Baresi, L. Pasquale, P. Spoletini, Fuzzy goals for requirements-driven adaptation, in: 18th Requirements Engineering Conference (RE 2010), 2010, pp. 125–134.
[2] A. Bianco, L. de Alfaro, Model checking of probabilistic and nondeterministic systems, in: P.S. Thiagarajan (Ed.), FSTTCS, in: LNCS, vol. 1026, Springer, 1995, pp. 499–513.
[3] R. Calinescu, et al., Dynamic QoS management and optimization in service-based systems, IEEE Trans. Softw. Eng. 37 (3) (2011) 387–409.
[4] R. Calinescu, M.Z. Kwiatkowska, Using quantitative analysis to implement autonomic IT systems, in: 31st International Conference on Software Engineering, Proceedings, ICSE 2009, May 16–24, 2009, Vancouver, Canada, IEEE, 2009, pp. 100–110.
[5] J. Cámara, D. Garlan, W.-G. Kang, W. Peng, B. Schmerl, Uncertainty in Self-Adaptive Systems: Categories, Management, and Perspectives, Technical Report CMU-ISR-17-110, Institute for Software Research, Carnegie Mellon University, 2017, Available at http://acme.able.cs.cmu.edu/pubs/uploads/pdf/CMU-ISR-17-110.pdf.
[6] J. Cámara, D. Garlan, G.A. Moreno, B. Schmerl, Analyzing self-adaptation via model checking of stochastic games, in: R. de Lemos, D. Garlan, C. Ghezzi, H. Giese (Eds.), Software Engineering for Self-Adaptive Systems 3, in: LNCS, vol. 9640, Springer, 2018, in press.
[7] J. Cámara, D. Garlan, G.A. Moreno, B. Schmerl, Evaluating trade-offs of human involvement in self-adaptive systems, in: Managing Trade-Offs in Self-Adaptive Systems, 2016.
[8] J. Cámara, D. Garlan, B.R. Schmerl, A. Pandey, Optimal planning for architecture-based self-adaptation via model checking of stochastic games, in: R.L. Wainwright, J.M. Corchado, A. Bechini, J. Hong (Eds.), Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13–17, 2015, ACM, 2015, pp. 428–435.
[9] J. Cámara, A. Lopes, D. Garlan, B.R. Schmerl, Impact models for architecture-based self-adaptive systems, in: I. Lanese, E. Madelaine (Eds.), Formal Aspects of Component Software – 11th International Symposium, FACS 2014, in: LNCS, vol. 8997, Springer, 2014, pp. 89–107.
[10] J. Cámara, G.A. Moreno, D. Garlan, Reasoning about human participation in self-adaptive systems, in: P. Inverardi, B.R. Schmerl (Eds.), 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18–19, 2015, IEEE Computer Society, 2015, pp. 146–156.
[11] J. Cámara, G.A. Moreno, D. Garlan, B.R. Schmerl, Analyzing latency-aware self-adaptation using stochastic games and simulations, ACM Trans. Auton. Adapt. Syst. 10 (4) (2016) 1–28, 23.
[12] J. Cámara, W. Peng, D. Garlan, B. Schmerl, Reasoning about sensing uncertainty in decision-making for self-adaptation, in: Proceedings of the 15th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems (FOCLASA 2017), in: LNCS, Springer, 2018, in press.
[13] A. Cassandra, M. Nodine, S. Bondale, S. Ford, D. Wells, Using POMDP-based state estimation to enhance agent system survivability, in: IEEE 2nd Symposium on Multi-Agent Security and Survivability, 2005, Aug 2005, pp. 11–20.
[14] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, A. Simaitis, PRISM-games: a model checker for stochastic multi-player games, in: N. Piterman, S. Smolka (Eds.), Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13), in: LNCS, vol. 7795, Springer, 2013, pp. 185–191.
[15] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, A. Simaitis, Automatic verification of competitive stochastic systems, Form. Methods Syst. Des. 43 (1) (2013) 61–92.
[16] Y. Chen, S. Kar, J.M.F. Moura, Cyber-physical systems: dynamic sensor attacks and strong observability, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 1752–1756.
[17] B.H.C. Cheng, et al., Software engineering for self-adaptive systems: a research roadmap, in: B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee (Eds.), Software Engineering for Self-Adaptive Systems, in: LNCS, vol. 5525, Springer, 2009, pp. 1–26.

[18] M.S. Chong, M. Wakaiki, J.P. Hespanha, Observability of linear systems under adversarial attacks, in: 2015 American Control Conference (ACC), July 2015, pp. 2439–2444.
[19] T. Deshpande, P. Katsaros, S. Smolka, S. Stoller, Stochastic game-based analysis of the DNS bandwidth amplification attack using probabilistic model checking, in: Tenth European Dependable Computing Conference (EDCC), May 2014, 2014, pp. 226–237.
[20] I. Epifani, et al., Model evolution by run-time parameter adaptation, in: ICSE, IEEE CS, 2009, pp. 111–121.
[21] N. Esfahani, E. Kouroshfar, S. Malek, Taming uncertainty in self-adaptive software, in: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, ESEC/FSE '11, ACM, 2011, pp. 234–244.
[22] N. Esfahani, S. Malek, Uncertainty in self-adaptive software systems, in: R. de Lemos, H. Giese, H. Muller, M. Shaw (Eds.), Software Engineering for Self-Adaptive Systems II, in: LNCS, vol. 7475, Springer, 2013, pp. 214–238.
[23] H. Fawzi, P. Tabuada, S. Diggavi, Secure estimation and control for cyber-physical systems under adversarial attacks, IEEE Trans. Autom. Control 59 (6) (June 2014) 1454–1467.
[24] V. Forejt, et al., Automated verification techniques for probabilistic systems, in: SFM, in: LNCS, vol. 6659, Springer, 2011, pp. 53–113.
[25] M. Fox, D. Long, PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. 20 (1) (Dec. 2003) 61–124.
[26] D. Garlan, Software engineering in an uncertain world, in: Future of Software Engineering Research (FoSER), 2010, pp. 125–128.
[27] K. He, M. Zhang, J. He, Y. Chen, Probabilistic model checking of pipe protocol, in: Theoretical Aspects of Software Engineering (TASE), 2015, pp. 135–138.
[28] M.C. Huebscher, J.A. McCann, A survey of autonomic computing – degrees, models, and applications, ACM Comput. Surv. 40 (3) (2008).
[29] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (2003) 41–50.
[30] M. Kwiatkowska, et al., PRISM 4.0: verification of probabilistic real-time systems, in: Proc. of CAV'11, in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[31] S. Mahdavi-Hezavehi, P. Avgeriou, D. Weyns, 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.
[32] G.A. Moreno, J. Cámara, D. Garlan, B.R. Schmerl, Proactive self-adaptation under uncertainty: a probabilistic model checking approach, in: E.D. Nitto, M. Harman, P. Heymans (Eds.), Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30–September 4, 2015, ACM, 2015, pp. 1–12.
[33] G.A. Moreno, J. Cámara, D. Garlan, B.R. Schmerl, Efficient decision-making under uncertainty for proactive self-adaptation, in: S. Kounev, H. Giese, J. Liu (Eds.), 2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17–22, 2016, IEEE Computer Society, 2016, pp. 147–156.
[34] A.J. Ramirez, A.C. Jensen, B.H.C. Cheng, A taxonomy of uncertainty for dynamically adaptive systems, in: 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2012, pp. 99–108.
[35] B.R. Schmerl, J. Cámara, J. Gennari, D. Garlan, P. Casanova, G.A. Moreno, T.J. Glazier, J.M. Barnes, Architecture-based self-protection: composing and reasoning about denial-of-service mitigations, in: L.A. Williams, D.M. Nicol, M.P. Singh (Eds.), Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, HotSoS 2014, Raleigh, NC, USA, April 08–09, 2014, ACM, 2014, p. 2.
[36] J. Whittle, P. Sawyer, N. Bencomo, B.H.C. Cheng, J. Bruel, RELAX: incorporating uncertainty into the specification of self-adaptive systems, in: RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31–September 4, 2009, IEEE Computer Society, 2009, pp. 79–88.