Model Checking a Self-Adaptive Camera Network with Physical Disturbances

Gautham Nayak Seetanadi, Karl-Erik Årzén, Martina Maggio
Department of Automatic Control, Lund University
{gautham, karlerik, martina}@control.lth.se

Abstract—**The paper describes the design and verification of a self-adaptive system, composed of multiple smart cameras connected to a monitoring station, that determines the allocation of network bandwidth to the cameras. The design of such a system poses significant challenges, since multiple control strategies are active in the system simultaneously. In fact, the cameras adjust the quality of their streams to the available bandwidth, that is at the same time allocated by the monitoring station. Model checking has proven successful to verify properties of this complex system, when the effect of actions happening in the physical environment was neglected. Extending the verification models to include disturbances from the physical environment is however nontrival due to the state explosion problem. In this paper we show a comparison between the previously developed deterministic model and two alternatives for disturbance handling: a probabilistic and a nondeterministic model. We verify properties for the three models, discovering that the nondeterministic model scales better when the number of cameras increase and is more representative of the dynamic physical environment. We then focus on the nondeterministic model and study, using stochastic games, the behavior of the system when the players (cameras and network manager) collaborate or compete to reach their own objectives.**

I. Introduction

This paper discusses the problem of bandwidth allocation for a self-adaptive video-surveillance system, composed of cameras, connected to a monitoring station. There are two main aspects that should be taken into account: the *cyber* part of the problem (the resource distribution), and the *physical* part (the scene captured by the cameras). The characteristics of the problem call for dynamic resource allocation, which is well studied in autonomic computing. We would like to have guarantees on the behavior of the final solution, hence we resort to model checking, which allows us to formally analyze and prove properties on the complex adaptation scheme.

On the cameras side, adaptivity is needed to match a given bandwidth and storage space. On the monitoring side, varying the amount of bandwidth given to each camera is the key to fulfill dynamic time-varying requirements, like the need for having better images for an indoor area during the day and for the outdoor parking lot during night. Furthermore, scenes can be easier or harder to record. Each frame is processed and encoded, trying to compress it as much as possible, while retaining all the information contained in the original image. This means that each part of the image is either encoded as a new block or as some shifted block with respect to other blocks in previous frames, with small modifications. In fact, the encoded frame size, using a movement-based encoder (like MPEG), depends on many different factors, including (among others) nature, lighting conditions, and the amount of movement detected and encoded in the scene [16]. Even when images are transmitted as simple JPEG frames, the encoded scene makes a difference in the image sizes [33]. This motivates the need to include some considerations about the physics in the network bandwidth distribution protocols. However, due to the unpredictable and everchanging nature of the scene to be encoded, we argue that these considerations should be indirect and come from measurements, rather than prior knowledge. This paper describes the synthesis and verification of a network bandwidth distribution scheme, that reacts to changes in the physical environment without prior knowledge of its characteristics.

Bandwidth distribution requires mechanisms to be in place for multiple nodes to be allocated a certain amount of the network bandwidth with non-violation guarantees [1]. Transmission protocols, like the Flexible Time Triggered for Switched Ethernet (FTT-SE) [30], guarantee the non-violation of the assigned bandwidth. In a recent paper [33], we designed and implemented a scheme for bandwidth allocation and camera adaptation that decouples the two inherent adaptation dimensions: (i) bandwidth distribution, and (ii) adaptation of encoding parameters. The encoding parameters are used to adjust the frame sizes and trade the quality of the resulting stream for its compression.

It is well known that multiple control policies can negatively impact the performance of the system [21], so we needed formal guarantees that this would not happen in this case. We therefore resorted to model checking, and verified properties like the transmission of frames [33]. A model checker ensures that the bandwidth allocation protocol allows cameras to transmit their frames (of a given size). If frame sizes were known (or computable), this would be enough to ensure the transmission of the video streams. However, due to the changes in the recorded physical scenes, this assumption does not hold.

The changes in frame sizes due to the physical world can be seen as a stochastic *disturbance* — some element that randomly affects the scene and is reflected in the encoding. We tried to apply model checking, including a stochastic disturbance in our model. In so doing, we came across the state space explosion problem [12]. This paper discusses the problem we encountered and how we changed our model to

The authors are members of the LCCC Linnaeus Center and the ELLIIT Strategic Research Area at Lund University. This work was supported by the Swedish Research Council through the project “Feedback Computing”, VR 621-2014-6256.

capture the stochastic disturbance and retain some ability to verify properties of our video-surveillance scheme. Specifically, we make the following contributions:

- We incorporated disturbances in both a probabilistic and a stochastic model, expanding on [33].
- We compared the probabilistic and the stochastic model exposing their tradeoffs (capturing real-life behaviour versus the computational complexity of the verification process).
- We described, formally defined, and verified interesting properties of a self-adaptive camera network.
- We compared the system performance in the case of co-cooperative versus non-cooperative behaviors using PRISM-games [10].

The remainder of this paper is organized as follows. Section II states what are the requirements for the video-surveillance system and specifies the model for our system's components, i.e, cameras and bandwidth (network) manager. Section III gives some background about model checking, stochastic games, and the properties that can be verified for our system. We compare cooperative and non-cooperative schemes, highlighting their difference. Section IV compares the previously devised and the proposed model, describes the state space explosion problem and the results obtained with cooperative and non-cooperative strategies. Section V gives an overview of related work and Section VI concludes the paper.

II. System Overview

The system is composed of a monitoring station and a set of cameras that are connected to it via Ethernet networking, a standard setup for a video-surveillance system. The left part of Figure 1 shows an example of such a system. A monitor and network manager $\mathcal{M}$ collects images from n cameras (five in the figure), the i-th one being identified by the letter c_i. The monitor is in charge of assigning the network bandwidth as a fraction of the total bandwidth $\mathcal{H}$, a parameter of the system, e.g., $\mathcal{H} = 4\,[Mb/s]$.

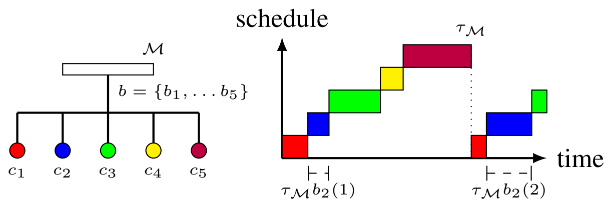


Fig. 1. System Overview

Generally speaking, the network manager assigns a vector b. Each element of this vector b represents the percentage of bandwidth that the corresponding camera can use. This means that the network manager chooses each element b_i such that $\sum_{i=1}^{n} b_i = 1$ and tries to maximize the quality of the camera streams. For the resource allocation, we use the algorithm proposed in [26] adapted to network bandwidth in [33], because it guarantees properties like starvation avoidance and fairness among the different streams.

The right side of Figure 1 shows a schedule example. The time is divided into manager periods $\tau_{\mathcal{M}}$, each of them corresponding to the transmission of a single frame from all the cameras. The manager partitions each period giving some time to each of the cameras. For example, camera c_2 is allowed to transmit in the first network manager period for a chunk of time $\tau_{\mathcal{M}} b_2(1)$ and in the second one for a time $\tau_{\mathcal{M}} b_2(2)$. During the time allocated to its slot, c_2 can use the full network bandwidth, therefore effectively being able to transmit a frame of size up to $\tau_{\mathcal{M}} \cdot b_2(k) \cdot \mathcal{H}[Mb]$. In the example shown in the figure, the network manager decides to increment the percentage of time allocated to c_2, which gets more bandwidth in the second transmission slot with respect to the first one. If a camera does not complete the transmission of a frame during its slot, then the frame is dropped. Similarly, only a given camera is able to transmit during its slot, and if the slot is not fully utilized the corresponding time is wasted.

To fulfill the bandwidth requirements, for each frame w, the i-th camera adjusts the quality of the video stream $q_{i,w}$ according to a very simple control strategy, resembling the TCP congestion window approach [40]. The video stream quality is a parameter that corresponds roughly to the percentage of information that is retained from the original image. Given the implementation of the video encoding, the quality value belongs to the interval 1 to 100, but in our system we do not allow a quality lower than 15, to ensure we preserve some information from the original frame. The congestion window algorithm increases and decreases the quality as required. Specifically, if the quality for the current frame resulted in a succesful transmission, the quality is increased slowly, incrementing it by 1. If the current frame is not transmitted correctly due to lack of bandwidth, the quality is halved. For other problems (*e.g* queue management, voice over IP), this simple algorithm has been proven successful [20], [40].

Depending on the make and model, each camera has a maximum frame size, which we indicate with $s_{i,\max}$. Denoting with $s_{i,w}$ the size of frame w produced by camera i, we use the following linear relationship to model the frame size in its most general form.

$$s_{i,w} = 0.01 \cdot q_{i,w} \cdot s_{i,\max} + \delta s_{i,w} \qquad (1)$$

The calculated frame size is then saturated between predetermined values of minimum and maximum allowable frame sizes as shown in (2).

$$s_{i,w} = \max\{s_{i,\min}, \min\{s_{i,\max}, s^{*}_{i,w}\}\}. \qquad (2)$$

Here, the factor 0.01 has the effect of correctly scaling the quality value. $\delta s_{i,w}$ is a disturbance acting on the frame size, that depends on the scene that is recorded. In previous work [33], we modeled the system using this equation successfully, but we set the disturbance to zero, to enable model checking and avoid the state space explosion problem. In this paper we investigate how to handle the disturbance so that model checking is still able to give us some guarantees about the system behavior.

III. Verification and Model Checking

This section introduces Markov Decision Processes (MDPs) and their extension to Stochastic Multi-Player Games (SMGs), both of which are used to model the camera network behavior including disturbances. Both the formalisms are supported by the PRISM model checker [24], which we use to conduct our study. The section first introduces the relevant concepts and then describes their application to the context of our problem. We compare three different models: the deterministic one without disturbances introduced in [33], and two different ways of handling disturbances: a probabilistic and a non-deterministic model. We then discuss the properties that we verify, corresponding to guarantees on the camera network behavior.

A. *Basic Concepts*

Markov Decision Processes (MDPs): *A MDP is a tuple $M = \{S, \overline{s}, A, Pr\}$, where S is a finite set of states, $\overline{s} \in S$ is the initial state. A is a finite set of actions, $Pr : S \times A \to S$ is a probabilistic (possibly partial) transition function mapping state-action pairs to probability distributions over S.* — MDPs describe the evolution of a system when this depends both on the action taken and on the current state the system is in. MDPs are the prefereed method to model discrete-time systems that exhibit both nondeterministic and probabilistic behavior [11]. We use MDPs to design the deterministic, probabilistic, and non-deterministic versions of our camera network models. Also, using the same formalism to define the three models allows us to compare them effectively.

Stochastic Multi-Player Games (SMGs): *A SMG is a tuple $G = \{\pi, S, (S_\pi, S_p), \overline{s}, A, Pr, L\}$, where π is a finite set of players. S is a finite set of states, partitioned into player states S_π and probabilistic states S_p. $\overline{s}$ is the set of initial states (one for each player). A is a finite set of actions, $Pr : S \times A \to S$ is a probabilistic (possibly partial) transition function mapping state-action pairs to probability distributions over S. L is an action labeling function used to synchronize transitions happening in the space of different players.* — SMGs extend MDPs distinguishing between several types of non-deterministic choices [23]. Each choice corresponds to an action performed by a different player. In this paper, we use SMGs to evaluate the concept of competitive vs collaborative players in the camera network, i.e., to evaluate what happens if the cameras cooperate with the manager towards a common objective or if they try to maximize only their own reward.

Probabilistic vs Nondeterministic: In the model checking terminology, a *probabilistic* model is a model that includes some transition probability. From any state, the system can envolve according to different transitions, satisfying a given probability distribution. For example, from state s_1, taking the action a does not change the state with a probability of 0.05, imply transitioning to state s_2 with a probability of 0.45 and drives the system to state s_3 with a probability of 0.5. In a *nondeterministic* model, there is no predetermined probability distribution for the transitions. From state s_1, taking the action a can lead to three alternatives: staying in s_1, transition to s_2, or transition to s_3. The probabilities of taking one of the the two transitions is not specified.

```
formula update_bw = ... // update bandwidth of all cameras
module NetworkManager
  [] (rm = rm_init) -> 1 : (rm' = rm_calc_bw); // initialization
  [end] (rm = rm_end) -> 1 : (rm' = rm_end); // self-absorbing
  [man_inter] (rm = rm_calc_bw) & (!end)-> 1 :
   (rm' = rm_alloc_bw) & (bw' = update_bw);
  // bw_allocated: allocating bandwidth to camera and waiting,
  // end if reached the maximum number of frames
  [bw_allocated] (rm = rm_alloc_bw) & (frames < max_frames) &
   (!end) -> 1 : (rm' = rm_wait) & (frames' = frames + 1);
  [bw_allocated] (rm = rm_alloc_bw) & (frames >= max_frames) &
   (!end) -> 1 : (end' = true) & (rm' = rm_end); // final
  // last_cam_sent: check if recalculation is needed/requested
  // or not and switch to corresponding state
  [last_cam_sent] (rm = rm_wait) & (want_rm) & (!end) ->
   1 : (rm' = rm_calc_bw); // recalculation
  [last_cam_sent] (rm = rm_wait) & (!want_rm) & (!end)->
   1 : (rm' = rm_alloc_bw); // no recalculation
endmodule
```

Listing 1. Network Manager, PRISM Code

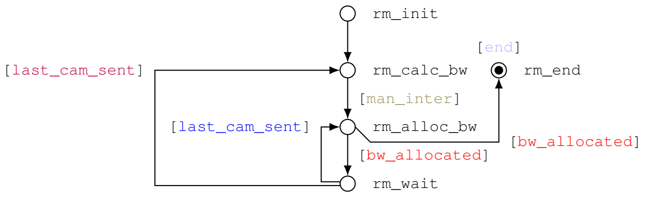


Fig. 2. Network Manager, MDP Representation

B. *Modeling the Problem*

We here describe three ways of modeling the camera network behavior using MDPs.

1) Network Manager: In all the three models, the network manager behaves in the same way. The manager is implemented as a module in PRISM, as shown in Listing 1 and the corresponding MDP representation is shown in Figure 2. In the code, `rm` is used to keep track of the network manager state. The state `rm_init` indicates the initialization phase for the network manager. The state `rm_calc_bw` corresponds to the state when the network manager is computing the new vector b and then the subsequent bandwidth assignment. The state `rm_alloc_bw` indicates that the the allocation is being performed (within the period). The state `rm_wait` corresponds to the state when the manager is waiting to be invoked. Finally, `rm_end` denotes the end of the execution (when the preset maximum number of frames for the verification procedure is reached). The labels indicate the transition names, so that other modules can synchronize their transisitions. Specifically, `[man_inter]` indicates that the network manager is intervening to reallocate bandwidth. It has computed the bandwidth and transitions to the state in which it allocates it. The figure gives a visual representation of the same concepts. States are represented as circles and transitions from one state to another are represented by directional arrows. PRISM allows for synchronization of multiple modules using labels, which are indicated using square brackets. The black circle in the `rm_end` state represents the fact that the state is a self-absorbing state. Listing 1 contains two `[bw_allocated]` transitions, to distinguish between the end state `rm_end` and the wait state `rm_wait`.The guards of the two transitions are different, therefore there is no non-determinism in the manager MDP — it is always clear, depending on the state of the system, which transition is going to be taken.

```
formula framesize = ... // compute frame size from equation (1)
formula update_q = ... // update the quality

module DeterministicCamera
  // init: synchronization with manager on bandwidth allocation
  [bw_allocated] (cam = cam_init) -> 1 : (cam' = cam_calc_fr);
  // entering computation of frame size, unlabeled transition
  // for all the cameras except the last, labeled to indicate
  // the end of the scheduling round
  [- or last_cam_sent] (cam = cam_calc_fr) ->
    1 : (cam' = cam_wait) & (q' = update_q) &
        (s'= framesize);
  // return to computation after all the cameras have sent
  // cycling until the end of the allocation by the manager
  [bw_allocated] (cam = cam_wait) -> 1 : (cam' = cam_calc_fr);
endmodule
```

Listing 2. Camera Deterministic Model, PRISM Code

```
formula framesize = ... // compute frame size from equation (3)
formula framesize_disturbance = ... // with disturbance
formula update_q = ... // update the quality

module ProbabilisticCamera
  // init: synchronization with manager on bandwidth allocation
  [bw_allocated] (cam = cam_init) -> 1 : (cam' = cam_calc_fr);
  // entering computation of frame size, probabilistic
  [- or last_cam_sent] (cam = cam_calc_fr) ->
    0.7 : (cam' = cam_wait) & (q' = update_q) &
          (s'= framesize) +
    0.3 : (cam' = cam_wait) & (q' = update_q) &
          (s'= framesize_disturbance);
  // return to computation after all the cameras have sent
  [bw_allocated] (cam = cam_wait) -> 1 : (cam' = cam_calc_fr);
endmodule
```

Listing 3. Camera Probabilistic Model, PRISM Code

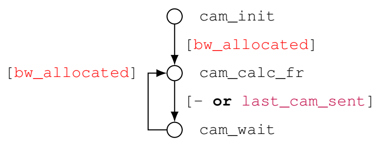


Fig. 3. Camera Deterministic Model

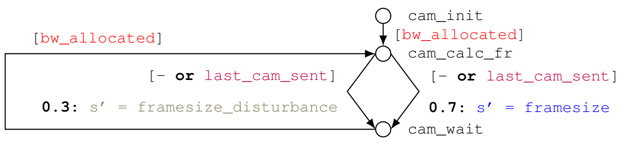


Fig. 4. Camera Probabilistic Model

2) Camera – Deterministic Model: The first model we introduce for the behavior of the cameras is a completely deterministic model, constructed similarly to the one presented in [33]. This model is our reference for comparison. It does not model the disturbances, which is the feature we want to introduce in this paper. The model consists of one network manager module as shown in Figure 2 and Listing 1 and n copies of a single camera model, depending on the number of cameras in the network. Listing 2 shows the PRISM code and Figure 3 shows the state space diagram of a single camera, when modeled in the deterministic way.

The camera stays in the initial state `cam_init` until the `[bw_allocated]` transition is taken by the network manager. When the network manager computes the bandwidth allocation for the cameras, the camera can take the transition that moves it into the computation of the framesize of the currently captured image. The formula `update_q` in the PRISM code is here used as a compact notation for the computation of the next quality level $q_{i,w}$ according to the congestion window algorithm. The formula `framesize` is used as a compact way of describing the computation of the next framesize defined in Equation (1), with $\delta s_{i,w}$ set to zero. The camera then transitions to the `cam_wait` state, where it waits for all the cameras to finish their transmission. The last camera module has a labelled transition `[last_cam_sent]` to the wait state (all the other cameras have no label for this transition), which allows us to synchronize with the network manager's actions. The behavior is repeated until a maximum number of frames is transmitted by each camera in the network, specified as a parameter at the model checking level.

3) Camera – Probabilistic Model: The first alternative that we propose to include disturbances with respect to the deterministic behavior is the use of a probabilistic model. In general, probabilities are a good tool to encode elements like hardware failures. We can include probabilisties in our model as shown in Figure 4 and in Listing 3. The (deterministic) network manager code is unchanged. All the stochastics resides in the capturing of the image. We would like to include the disturbance $\delta s_{i,w}$ introduced in [33] – and shown in Equation (1). With respect to prior work, however, it has been shown that the disturbance acting on the frame size would affect the camera quality as a multiplicative term [16]. Rather than modeling the disturbance as shown in [33], we introduce a term capturing unexpected quality variation $\delta q_{i,w}$ (which is used for example to model different light conditions or encoding capabilities).

Compared to the deterministic version, the transition that links the `cam_calc_fr` state to the `cam_wait` can be taken with two different effects. Once the bandwidth is determined, different paths can be taken, modeling a stochastic disturbance. In 70% of the cases there is no disturbance. On the contrary, in the remaining 30% of the cases, the computation of the frame size is carried out according to a different formula, that includes the additional term $\delta q_{i,w}$. More precisely, we use the following formula:

$$s_{i,w} = \begin{cases} (0.7) & 0.01 \cdot q_{i,w} \cdot s_{i,\max} \\ (0.3) & 0.01 \cdot (q_{i,w} + \delta q_{i,w}) \cdot s_{i,\max} \end{cases} \qquad (3)$$

The first line in Equation (3) is assigned a probability of 0.7. This captures the behavior of the camera when it is operating in its normal conditions. In this state, the relationship between the quality $q_{i,w}$ and the frame size $s_{i,w}$ is linear and there is no disturbance. On the contrary, the second line is assigned a probability of 0.3 and captures the behavior when a disturbance occurs. Using a static multiplicative disturbance (additive on the quality, $\delta q_{i,w}$), we generate frame sizes of higher values for the same input quality $q_{i,w}$. The probabilities can be tuned to model varying behaviors in the real world and other alternative paths can be added to the model. The specific values used for the disturbance terms can be obtained via profiling as specified in [16], and the values we used were found using the testbed developed for [33].

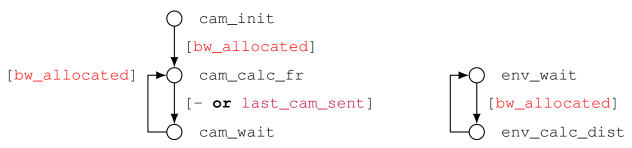
Clearly, the probabilistic choice added increases the complexity of the model and the number of states that it contains, which the model checker has to deal with. Experimental results on the scalability of the probabilistic model are described in Section IV-A.

```
formula framesize_disturbance = ... // from equation (4)
formula update_q = ... // update the quality

module NondeterministicCamera
  [bw_allocated] (cam = cam_init) -> 1 : (cam' = cam_calc_fr);
  [- or last_cam_sent] (cam = cam_calc_fr) ->
   1 : (cam' = cam_wait) & (q' = update_q) &
       (s'= framesize_disturbance); // with disturbance
  [bw_allocated] (cam = cam_wait) -> 1 : (cam' = cam_calc_fr);
endmodule

module Environment
  // generate a new disturbance as a nondeterministic choice
  // between a set of values, and move to wait state
  [bw_allocated] (env = env_wait) ->
   1 : (env' = env_calc_dist) & (dist' = new_dist);
  [] (env = env_calc_dist) -> 1 : (env' = env_wait);
endmodule
```

Listing 4. Camera Nondeterministic Model, PRISM Code



(a) Camera (b) Environment

Fig. 5. Camera Nondeterministic Model

4) Camera – Nondeterministic Model: Here we present an alternative to the probabilistic model for including a disturbance: a nondeterministic model. Nondeterminism is present in the vast majority of cyber-physical systems. Often, these systems are composed of multiple possible outcomes, with dependencies from other entities (in our case from the physical environment). The MDP representation of model is presented in Figure 5, and its PRISM code is shown in Listing 4. The network manager is unchanged and the camera model is the same as the deterministic one, with the only exception of the frame size computation, that occurs according to the formula that includes the disturbance.

$$s_{i,w} = 0.01 \cdot (q_{i,w} + \delta q_{i,w}) \cdot s_{i,\max} \tag{4}$$

We use Equation (4) to model the stochasticity of the image capture. The equation is the same as the second line in Equation (3), used in the probabilistic model. The difference is that for each w frame of i camera, we can introduce a random value for $\delta q_{i,w}$ (which we choose to be between -20 and 20, using our implementation to find representative values for the most common scenes). We specify alternative paths with each of these paths corresponding to a disturbance vector with an amount of disturbance to each of the cameras (e.g., for three cameras we could select the disturbance vector $< \delta q_{1,1} = 10, \delta q_{2,1} = -5, \delta q_{3,1} = 0 >$ for the first frame, and then the vector $< \delta q_{1,2} = -10, \delta q_{2,1} = 5, \delta q_{3,1} = -5 >$ for the second frame). This models the randomness in the physical environment representing a random amount of noise affecting the captured image. One of the paths here is $< \delta q_{1,1} = 0, \delta q_{2,1} = 0, \delta q_{3,1} = 0 >$. This corresponds to the steady state behavior of the system when there are no disturbances (similarly to the deterministic model).

The value of $\delta q_{i,w}$ is determined using another module (named `Environment`) as shown in the PRISM listing. The `Environment` module is synchronized using the same labels as the manager and the camera module. The environment acts for every frame collected by the cameras and can select among nondeterministic alternatives for the disturbances, including zeros (to resemble the deterministic behavior).

The inclusion of another module greatly increases the number of states of the model, which poses significant limitations for the model checker. The state explosion problem is discussed in Section IV-A, that includes verification results.

C. Stochastic Games

PRISM-Games [10] extends PRISM incorporating the verification of competitive and collaborative behavior with SMGs for nondeterministic models. Games are modeled turn-based where the players of SMGs are in control of choices that are nondeterministic. Using PRISM-Games we can investigate the behavior of our camera network and network manager when the different entities collaborate with one another to reach a common goal, or when they each have a separate goal and are only concerned with that one.

In our system, there are naturally $n + 1$ players. The first n players are the cameras. Each of them wants to maximize the number of frames transmitted correctly (possibly with a weight depending on the quality — the higher the quality, the better). The last player is the network manager, who wants to maximize the utilized bandwidth allocated to the cameras. In a collaborative framework, the manager would allocate the bandwidth fairly to all the cameras, and the cameras would use the allocated bandwidth minimizing the effect of the stochastic disturbance. In a competitive framework, the manager would only try to maximize its own benefit and each camera would do the same, irrespective of the others' objective functions.

PRISM-Games allows to declare only two players, either collaborating or competing with one another. This is however not a limitation in this case, since all the objective functions for the n cameras do not depend on each another and maximizing one does not change the value of the others. We can therefore declare one player as the manager and one player as the set of cameras and compare the collaborative versus competitive game, where they pursue their own objectives. Our results for this comparison are discussed in Section IV-C.

D. Properties

This section covers the different properties that we can verify on our system.

Models developed in PRISM can be augmented with rewards: real values associated with certain states or transitions of the model. Listing 5 shows the rewards constructed. Reward `"rm_calls"` awards a reward of 1 whenever the transition corresponding to [`man_inter`] is taken. Similarly rewards `"cam_fr_dropped"` and `"cam_fr_sent"` rewards 1 when a camera drops or sends a frame respectively. Finally reward `"cost"` is a combination of values, penalizing each dropped frames by 10 and each manager intervention by 1.

Listing 6 shows the properties that we use to evaluate the three different MDP models: the deterministic one presented in Section III-B2, the probabilistic one presented in Section III-B3, and the nondeterministic one discussed in Section III-B4.The identifier **`R`** indicates a *reward* of the specified type. Rewards are used in PRISM to encode quantitative

```
rewards "rm_calls"
  [man_inter] true : 1; // +1 when manager intervenes
endrewards
rewards "frame_dropped"
  [] cam = cam_fr_drop : 1; // +1 with every dropped frame
endrewards
rewards "frame_sent"
  [] cam = cam_fr_sent : 1; // +1 with every sent frame
endrewards
rewards "cost"
  [man_inter] true : 1; // +1 when manager intervenes
  [] cam = cam_fr_drop : 10; // +10 with every dropped frame
endrewards
```

Listing 5. Reward Structures

```
R{"rm_calls"}max =? [ F end ] // MDP-1: Maximum interventions
R{"rm_calls"}min =? [ F end ] // MDP-2: Minimum interventions
R{"frames_dropped"}max =? [ F end ] // MDP-3 Maximum dropped
R{"frames_dropped"}min =? [ F end ] // MDP-4: Minimum dropped
R{"frames_sent"}max =? [ F end ] // MDP-5: Maximum frames sent
R{"frames_sent"}min =? [ F end ] // MDP-6: minimum frames sent
R{"cost"}min=? [ F end ] // MDP-7: Minimum operational cost
```

Listing 6. Properties of MDP models

verification. The term reward indicates a positive quantity, but can be used to quantify costs as well [24]. For example, **R**{"rm_calls"}**max** in Property MDP-1 indicates the maximum value of the reward associated with the (number of) network manager calls. The identifier `[F end]` indicates that the reward is calcualted when the `end` state is reached. The `end` state is reached when a preset number of frames are sent by the cameras.

Properties MDP-1 and MDP-2 are used to track the reward `"rm_calls"` which is incremented every time the manager changes the bandwidth allocation. Similarly the frames sent and dropped are rewarded in properties MDP-3, MDP-4 and MDP-5, MDP-6 respectively. Finally, property MDP-7 tracks the operational cost for the system. Denoting with d_i the number of dropped frames for camera i and with m_c the number of manager calls, the cost is defined as $m_c + 10 \cdot \sum_1^n d_i$, which means it considers every dropped frame contribution to the cost 10 and every manager call contribution to the cost 1. We designed this cost to be multi-objective, as we believe that skipping the transmission of frames incurs information loss, but at the same time we would like to avoid frequent manager interventions. The trade-off can be set to different values (for example, manager intervention could be penalized more than dropped frames in steady state). Section IV-A describes our results.

Similarly, Listing 7 shows the properties that we use to evaluate the cooperative vs competitive behavior of the system using SMGs. The properties of Listing 6 are augmented to indicate the dominant player the property refers to. This is either the manager, the camera, or both (in the cooperative scenario). The prepended label, delimited with <<>>, indicates the player who maximizes or minimizes the current objective. Section IV-C describes the results we obtained for these properties.

IV. Results

This section presents our verification results. First, we analyze the scalability of the three proposed models. Then we discuss the two settings of cooperating entities versus competing ones (i.e., we show the results obtained when the

```
// SMG-1 Maximum interventions, decision maker: manager
<<manager>> R{"rm_calls"}max =? [ F end ]
// SMG-2 Minimum interventions, decision maker: manager
<<manager>> R{"rm_calls"}min =? [ F end ]
// SMG-3 Maximum interventions, decision maker: cameras
<<cameras>> R{"rm_calls"}max =? [ F end ]
// SMG-3 Minimum interventions, decision maker: cameras
<<cameras>> R{"rm_calls"}min =? [ F end ]
// SMG-5 Maximum interventions, cooperative
<<manager, cameras>> R{"rm_calls"}max =? [ F end ]
// SMG-6 Minimum interventions, cooperative
<<manager, cameras>> R{"rm_calls"}min =? [ F end ]
// SMG-7 Maximum frames dropped, manager
<<manager>> R{"frames_dropped"}max =? [ F end ]
// SMG-8 Minimum frames dropped, manager
<<manager>> R{"frames_dropped"}min =? [ F end ]
// SMG-9 Maximum frames dropped, cameras
<<cameras>> R{"frames_dropped"}max =? [ F end ]
// SMG-9 Minimum frames dropped, cameras
<<cameras>> R{"frames_dropped"}min =? [ F end ]
// SMG-11 Maximum frames dropped, cooperative
<<manager, cameras>> R{"frames_dropped"}max =? [ F end ]
// SMG-12 Maximum frames dropped, cooperative
<<manager, cameras>> R{"frames_dropped"}min =? [ F end ]
// SMG-13 Maximum frames sent, manager
<<manager>> R{"frames_sent"}max =? [ F end ]
// SMG-14 Minimum frames sent, manager
<<manager>> R{"frames_sent"}min =? [ F end ]
// SMG-15 Maximum frames sent, cameras
<<cameras>> R{"frames_sent"}max =? [ F end ]
// SMG-16 Minimum frames sent, cameras
<<cameras>> R{"frames_sent"}min =? [ F end ]
// SMG-17 Maximum frames sent, cooperative
<<manager, cameras>> R{"frames_sent"}max =? [ F end ]
// SMG-18 Minimum frames sent, cooperative
<<manager, cameras>> R{"frames_sent"}min =? [ F end ]
// SMG-19 Minimum operational cost, manager
<<manager>> R{"cost"}min =? [ F end ]
// SMG-20 Minimum operational cost, cameras
<<cameras>> R{"cost"}min =? [ F end ]
// SMG-21 Minimum operational cost, cooperative
<<manager, cameras>> R{"cost"}min =? [ F end ]
```

Listing 7. Properties of SMG models

cameras and the manager collaborates to reach an objective and when they pursue the objective on their own).

We wrote the code for the deterministic, probabilistic, and nondeterministic models using the PRISM language and its model checker. We use the *explicit* engine for model checking, since it is the only engine that handles SMGs. This means that the model checker only uses explicit-state data structures, storing models as sparse matrices. The explicit engine is also a good fit for our requirements, since our models have a very large state space, but only a fraction of the states is actually reachable.

We selected a maximum number of 10 frames for our model and a set $\mathcal{H}$ to 4Mb/s, which resembled the experimental setup used in [33]. The other parameters (e.g., the period of the resource manager) were set according to the previous experiments [33][1]. The models were built on a computer with 128GB of RAM, that was allocated to PRISM. This is necessary to build large models for model checking.

A. Scalability

Here, we evaluate how the three proposed models scale with the number of cameras, i.e., with the size of the problem. Table I shows the number of states for the PRISM models for an increasing number of cameras n and Figure 6 shows a visual representation of the same numbers (notice the y-axis is logaritmic scaled). As expected from the discussion in [33], the deterministic model scales exponentially with the number of cameras in the system. The same is true for the other models, although the model size greatly increases when the model includes the effect of disturbances (with a similar growth rate). Notice that increasing the number of frames used

[1]The code for the experiments is publicly available at: https://github.com/gauthamnayaks/camnetverification/tree/master/icac19

TABLE I
SCALABILITY ANALYSIS: STATE SPACE GROWTH

n	Deterministic	Probabilistic	Nondeterministic
2	111	404	6908
3	277	246790	18190
4	771	11996	43915
5	2245	70652	148776
6	6651	–	491603
7	19833	–	1555671
8	59328	–	4786617

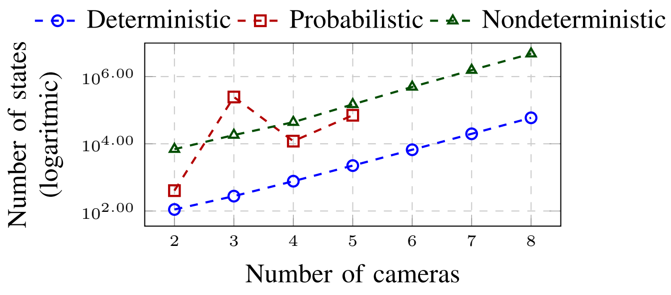


Fig. 6. Scalability Analysis: State Space Growth Plot

for the evaluation would result in larger models, having the same growth characteristics.

We were able to build the probabilistic model only for up to five cameras. On the contrary, we were able to build and verify the nondeterministic model for up to eight cameras in the system. Even though the nondeterministic model has a larger number of states with respect to the probabilistic one, we noticed that the probabilistic model is interestingly difficult to build. Also, in the case of three cameras, the resulting probabilistic model has a large number of states. We initially attributed this to a parameter conflict (a particularly difficult set of parameters, that could cause larger fluctuations, i.e., the weights used in the network manager multiplied with the update step would result in not changing the bandwidth correctly and needing a lot of refinement). However, despite a deeper exploration, we were not able to find the parameters that create the conflict. The only insight that we have on the problem is that PRISM handles the involved variables as integer numbers. This could create problems in the initial step (and potentially in subsequent ones), precisely in the 3 cameras state, since allocating (equally) the 100% bandwidth to three cameras results in a 1% unutilized bandwidth.

B. *Property verification*

In this section, we compare the results we obtain when we verify the properties described in Section III-D (Listing 6). The results are shown in Figure 7. The x-axis represents the number of cameras in the model, n. To provide a comparison, we use a maximum number of 8 cameras (but only 5 in the probabilistic case since that is the biggest model we could have). The y-axis shows the numbers obtained for the properties. Specifically, we show the maximum and minimum number of manager interventions in the top-left plot, the maximum and minimum number of dropped frames in the top-right plot, the maximum and minimum number of sent frames in the bottom-left corner, and the minimum operational cost in the bottom-right corner. For the deterministic case, the maximum and minimum numbers always coincide, since there is no disturbance in the model and there is no uncertain path. This is the same even in the probabilistic case as the probabilities are resolved deterministically.

Manager Interventions: The number of interventions of the resource manager is upper bounded by 10, for our example. In the probabilistic model, the worst case is computed preserving the probability distribution (specified as having a disturbance 30% of the times and not having any in 70% of the cases), which explains the difference in the worst case for the number of interventions. In fact, if a disturbance is present in the first few frames, the camera controller more aggressively regulates the quality, therefore not invoking the network manager as much. The most interesting information obtained from the figure, however, is the confidence band given by the difference between the maximum and minimum number of interventions in the nondeterministic case. The nondeterministic model still captures the worst case, but also allows us to determine that a specific disturbance configuration could lead to a better outcome (i.e., it could lead to the network manager intervening less). This means that in some cases, the presence of different stochastic disturbances occurring in the scenes recorded allows the system to converge to a satisfactory quality set and bandwidth allocation quicker than expected.

Frames Dropped: A similar behavior can be observed for the dropped frames. The deterministic model drops the least amount of frames, since it does not require continuous adaptation to handle disturbances. The best case (minimum number of dropped frames) for the nondeterministic model coincides with the deterministic model. This shows that our model is accurate as the disturbances cause the drop in frames. The band between the minimum and maximum values for the nondeterministic model (more or less between 8 and 24 dropped frames in total for 8 cameras) gives us useful information, allowing us to qualify the potential system behavior and how disruptive disturbances can be. As system manufacturers, we can then wonder if we need to tighten our bandwidth requirements (including more monitors or increasing the network bandwidth) or if we can be satisfied with the maximum frame loss that we verify could happen. The probabilistic model shows a peak for the model with three cameras, which seems to be related to the anomaly with the number of states (and contributes to our parameter interference hypothesis).

Frames Sent: The graph that depicts the number of sent frames also illustrates the effect of disturbances. Again the best case scenario (maximum sent frames) is the same for the deterministic and nondeterministic model. This means that the best case scenario is here achieved when no disturbance is present. The presence of disturbances causes a different minimum frames sent in the nondeterministic model. The nondeterministic model shows that it captures better the interplay between the physical dynamics, compared to its probabilistic counterpart. In fact, the probabilistic counterpart forces the probability distribution to respect the 30% and 70% rule also in the best case, not capturing strange circumstances in which this distribution — only empirically found — may not be

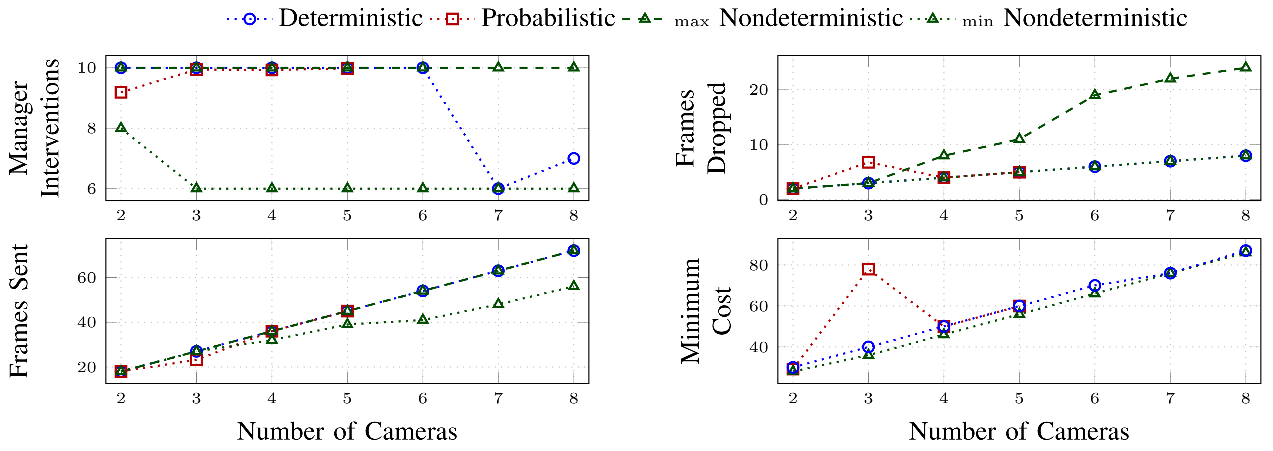


Fig. 7. Property verification for Deterministic, Probabilistic, and Nondeterministic Model

respected.

Minimum Cost: The final graph shows the minimum cost (where the cost is computed according to the formula introduced in Section III-D, penalizing frame drops and manager interventions). The nondeterministic model shows a potentially lower cost, due to its ability to capture the more realistic behavior of the camera system and corner cases (in this case, the circumstances in which the disturbance is actually helpful with respect to the bandwidth allocation and leads to faster convergence).

C. *Collaborative vs. Competitive*

This section provides a deeper analysis of the nondeterministic model presented in Section III-B4 (as it offers a realistic representation of the disturbance and better scalability properties). We define the problem as a Stochastic Game, as shown in Section III-C. We use PRISM-Games [10] to compare the system's behavior when the objective is to minimize or maximize a property (among the ones defined in Section III-D, Listing 7 — i.e., either minimizing the number of manager interventions, or minimizing the number of dropped frames, or maximizing the number of frames sent, or minimizing the cost, defined as a linear combination of the number of manager interventions and the dropped frames, and specifically as $m_c + 10 \cdot \sum_1^n d_i$, where m_c is the number of manager intervention and d_i is the number of frames dropped by the i-th camera).

Figure 8 shows our results. As in the previous results, the x-axis encodes the number of cameras, and on the y-axis we represent the values obtained for the property we are evaluating. The dotted line with square markers shows the results obtained when the manager is the only player trying to optimize the property. The dashed line with triangle markers shows the value obtained when the player representing the set of cameras is trying to optimize for the property on its own. Finally, the solid lines with circles represent the results of the collaborative game setting.

Manager Interventions: The leftmost plot in the top line shows the achievable results in terms of minimization of manager interventions. If the manager is the only player in charge of the optimization, it can achieve a lower number of interventions (penalizing other aspects of the system). On the contrary, if the decision is collaboratively taken by players or taken only by the cameras, the (worst case) number of interventions cannot be minimized.

Frames Dropped: The network manager alone is not able to minimize the (worst case) on the number of frames dropped. However, both when the cameras are in charge of the decisions and in the collaborative version, the number of frames dropped can be lowered compared to the only action of the manager.

Frames Sent: The plot of the number of frames sent is probably the most interesting one, and fully reveals the nature of the trade-off between the decision makers. When the number of camera grows, if the manager is the only one in charge of the decision, the (best case) number of frames sent is lower then if there is collaboration or if the cameras are in charge of the decision. The figure also reveals that the cameras alone are able to take better decisions that lead to sending more frames, rather than the collaborative decision. This shows that the problem is indeed very complex and the interplay between the different control strategies (at the camera level and at the network manager level) is difficult to design. The players can exploit the different aspects of the problem to obtain a less fair (a larger distance from the collaborative) result.

Minimum Cost: Finally. the minimum cost that can be achieved is the same when the cameras are deciding and when there is collaboration, and higher if the network manager is the only decision maker.

V. Related Work

This section discusses related research. The paper deals with different modeling and verification techniques for a self-adaptive camera network. We classify the related work in two categories. First, we discuss work related to our problem: self-adaptive camera networks. Then, we describe work related to the application of model checking to complex systems.

Self-adaptive video transmission has been given some attention in the past [39], [13], [32], [41], [38], [43], [6], leading to alternative protocols and improvements. They typically

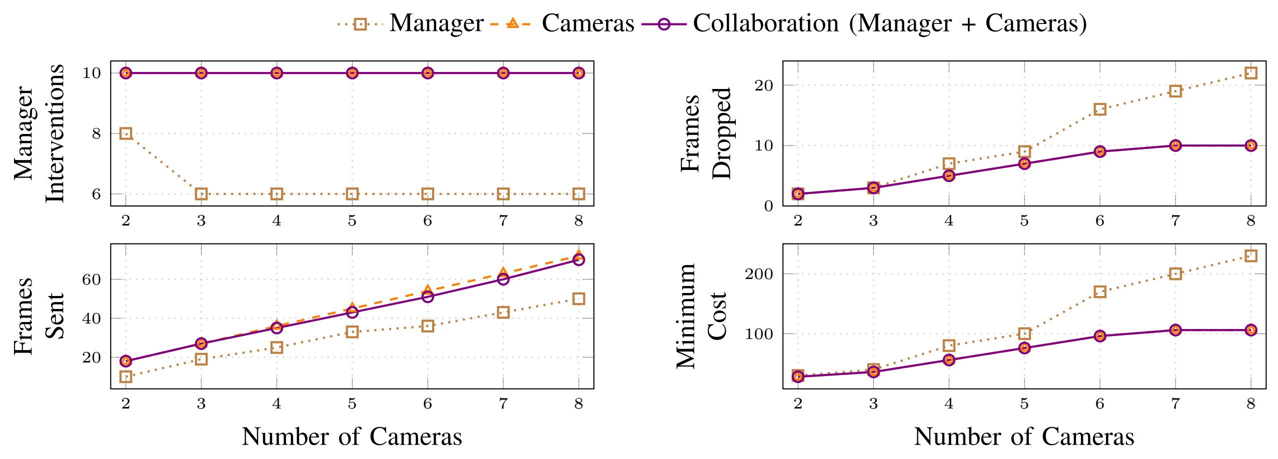


Fig. 8. Cooperative vs. Competitive Optimization

explore alternative encoding of redundant information within a sequence of frames, paving the way for standards such as MJPEG, JPEG200, MPEG-H and H.265. Specifically, self-adaptive cameras adapt their streams to provisions from the network [32], [31], [41], without considering network scheduling. In the scheduling domain, [36] uses adaptive network channels, supervised by a global network manager, to track the effective bandwidth used by the cameras, but do not include any camera stream adaptation. Our network is more complicated due to the interplay of the two characteristics and their (possibly) conflicting objectives (e.g., maximizing the quality of the streams and minimizing allocation changes). Our previous paper [33] does combine the adaptation strategy for the cameras and the bandwidth allocation, verifying properties of the overall system in absence of disturbances. In this work, we improve on previous results including disturbance management strategy and the relationship with the physical environment the cameras record.

There has also been work done on temporal logic verification using simulation [17]. This work deals with verification of a continuous dynamical system using a finite number of trajectories. In our case, we model the camera network as a discrete system and verify the system as opposed to its evolution.

We employ model checking [11] to verify desirable properties of the complex system composed of cameras and network manager. We used the PRISM model checker [24]. In the literature, PRISM has been used for the verification of properties of a wide-variety of systems [23], among which we find: randomized distributed algorithms [25], probabilistic software [22], nontechnology designs [28], communication protocols [15], self-adaptive software [4], security [2], anonymous protocols [35]. Specifically, we use Probabilistic Model Checking (PMC). PMC was instrumental in the verification of properties of a video streaming service in [27]. In this work, high-fidelity simulations are abstracted into probabilistic higher-level models for analysis. PMC is also employed for verification of safety and timeliness properties in air traffic control systems [19]. In contrast with these works, we study bounded disturbances and focused on the cooperation or competition of different entities in the network, using stochastic games. For this, several models exist, like concurrent games [34], [9], and partial-observing games [8], [7]. We modeled our system as a turn-based game [14].

There also exist various case studies done to evaluate different control and networked systems. Some of them are microgrid management system, decision making for sensor networks, reputation protocol for user-centeric networks [37], UAV mission planning [18], pan-tilt zoom cameras [29], aircraft power distribution [3] and self-adaptive architectures [5]. These work focused on strategy synthesis, whereas we concentrate on the verification of properties and reward/cost optimization.

There has also been some work on stochastic analysis of automotive systems [42]. This work focused more on the timing analysis which we do not pursue in our work.

VI. Conclusion

This work focuses on the use of model checking in cyber-physical systems. In particular, we have selected one problem, the verification of properties of a bandwidth allocation scheme for self adaptive cameras, and we studied the interplay between the cyber and the physical part of the system. The changes in the scenes the cameras record induce disturbances and uncertainty. This physical element interacts with the cameras adaptation strategy and the network manager bandwidth allocation policy. We used model checking to verify properties of the closed-loop system (the system in which all camera controllers and the network bandwidth allocation are active simultaneously) in the presence of disturbances.

We incurred into the scalability problem, and we discovered that — for this specific problem — a nondeterministic model is far more scalable and representative than a probabilistic model. We wrote our models to be as scalable as possible, containing as few labels and synchronization points as possible and trying to reduce the number of resulting states. We also made sure to represent the system as realistically as possible. As a result, we were able to verify models up to 8 cameras with nondeterministic transitions. We used these models to study the difference between collaboration and competition, and what happens when players are greedy. As a result, we unveiled interesting trade-offs that are inherent in any adaptive bandwidth allocation system.

References

[1] L. Almeida, P. Pedreiras, J. Ferreira, M. Calha, J. A. Fonseca, R. Marau, V. Silva, and E. Martins. Online QoS adaptation with the flexible time-triggered (FTT) communication paradigm. In *Handbook of Real-Time and Embedded Systems*, 2007.

[2] S. Basagiannis, P. Katsaros, A. Pombortsis, and N. Alexiou. Probabilistic model checking for the quantification of dos security threats. *Computers & Security*, 28(6):450 – 465, 2009.

[3] N. Basset, M. Kwiatkowska, U. Topcu, and C. Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 256–271, Berlin, Heidelberg, 2015.

[4] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, Sept. 2012.

[5] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey. Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 428–435, New York, NY, USA, 2015.

[6] D. Cao, T. Nguyen, and L. Nguyen. Improving the video transmission quality over IP network. In *5th International Conference on Ubiquitous and Future Networks*, 2013.

[7] K. Chatterjee and L. Doyen. Partial-observation stochastic games: How to win when belief fails. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 175–184, June 2012.

[8] K. Chatterjee, L. Doyen, S. Nain, and M. Y. Vardi. The complexity of partial-observation stochastic parity games with finite-memory strategies. In *Foundations of Software Science and Computation Structures*, pages 242–257, Berlin, Heidelberg, 2014.

[9] K. Chatterjee and R. Ibsen-Jensen. Qualitative analysis of concurrent mean-payoff games. *Inf. Comput.*, 242(C):2–24, June 2015.

[10] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. PRISM-games: A model checker for stochastic multi-player games. In *Proc. 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, volume 7795 of *LNCS*, pages 185–191, 2013.

[11] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. 2018.

[12] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Berlin, Heidelberg, 2012.

[13] A. Communication. White paper: Digital video compression: Review of the methodologies and standards to use for video transmission and storage, 2004.

[14] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.

[15] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of bluetooth device discovery. *International Journal on Software Tools for Technology Transfer*, 8(6):621–632, Nov 2006.

[16] V. Edpalm, A. Martins, K.-E. rzén, and M. Maggio. Camera Networks Dimensioning and Scheduling with Quasi Worst-Case Transmission Time. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:22, Dagstuhl, Germany, 2018.

[17] G. E. Fainekos, A. Girard, and G. J. Pappas. Temporal logic verification using simulation. In *Formal Modeling and Analysis of Timed Systems*, pages 171–186, Berlin, Heidelberg, 2006.

[18] L. Feng, C. Wiltsche, L. Humphrey, and U. Topcu. Controller synthesis for autonomous systems interacting with human operators. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ICCPS '15, pages 70–79, New York, NY, USA, 2015.

[19] T. Hanh and D. Van Hung. Verification of an air-traffic control system with probabilistic real-time model-checking. Technical report, UNU-IIST United Nations University International Institute for Software Technology, 2007.

[20] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. Control engineering for computing systems - industry experience and research challenges. *IEEE Control Systems Magazine*, 25(6):56–68, Dec 2005.

[21] J. Heo and T. Abdelzaher. Adaptguard: Guarding adaptive systems from instability. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 77–86, New York, NY, USA, 2009. ACM.

[22] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In *Verification, Model Checking, and Abstract Interpretation*, pages 182–197, Berlin, Heidelberg, 2009.

[23] M. Kwiatkowska. Model checking and strategy synthesis for stochastic games: From theory to practice. In *Proc. 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, 2016.

[24] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591, 2011.

[25] M. Z. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using cadence smv and prism. In *Proceedings of the 13th International Conference on Computer Aided Verification*, CAV '01, pages 194–206, London, UK, UK, 2001.

[26] M. Maggio, E. Bini, G. Chasparis, and K.-E. Årzén. A game-theoretic resource manager for rt applications. In *25th Euromicro Conference on Real-Time Systems*, 2013.

[27] T. Nagaoka, A. Ito, K. Okano, and S. Kusumoto. Qos analysis of real-time distributed systems based on hybrid analysis of probabilistic model checking technique and simulation. *Transactions on Information and Systems*, E94.D(5), 2011.

[28] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla. Evaluating the reliability of nand multiplexing with prism. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10):1629–1637, Oct 2005.

[29] N. Ozay, U. Topcu, R. M. Murray, and T. Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 45–54, April 2011.

[30] P. Pedreiras and L. Almeida. The flexible time-triggered (FTT) paradigm: an approach to qos management in distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, 2003.

[31] N. Ramos, D. Panigrahi, and S. Dey. Dynamic adaptation policies to improve quality of service of real-time multimedia applications in IEEE 802.11e WLAN networks. *Wirel. Netw.*, 13(4), 2007.

[32] B. Rinner and W. Wolf. An introduction to distributed smart cameras. *Proceedings of the IEEE*, 96(10), 2008.

[33] G. N. Seetanadi, J. Cámara, L. Almeida, K. Årzén, and M. Maggio. Event-driven bandwidth allocation with formal guarantees for camera networks. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 243–254, 2017.

[34] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[35] V. Shmatikov. Probabilistic analysis of anonymity. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pages 119–128, June 2002.

[36] J. Silvestre-Blanes, L. Almeida, R. Marau, and P. Pedreiras. Online QoS management for multimedia real-time transmission in industrial networks. *IEEE Transactions on Industrial Electronics*, 58(3), 2011.

[37] A. Simaitis. *Automatic Verification of Competitive Stochastic Systems*. PhD thesis, Department of Computer Science, University of Oxford, 2014.

[38] L. Toka, A. Lajtha, É. Hosszu, B. Formanek, D. Géhberger, and J. Tapolcai. A Resource-Aware and Time-Critical IoT framework. In *IEEE International Conference on Computer Communications INFOCOM*, May 2017.

[39] B. Vandalore, W. Feng, R. Jain, and S. Fahmy. A survey of application layer techniques for adaptive streaming of multimedia. *Real-Time Imaging*, 2001.

[40] S. Varma. *Internet Congestion Control*. San Francisco, CA, USA, 1st edition, 2015.

[41] X. Wang, M. Chen, H. M. Huang, V. Subramonian, C. Lu, and C. D. Gill. Control-based adaptive middleware for real-time image transmission over bandwidth-constrained networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(6), 2008.

[42] H. Zeng, M. D. Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic analysis of can-based real-time automotive systems. *IEEE Transactions on Industrial Informatics*, 5(4):388–401, Nov 2009.

[43] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 426–438, New York, NY, USA, 2015.