Event-Driven Bandwidth Allocation with Formal Guarantees for Camera Networks

Gautham Nayak Seetanadi[1], Javier Cámara[2], Luis Almeida[3], Karl-Erik Årzén[1], Martina Maggio[1]
[1]Department of Automatic Control, Lund University, Sweden
[2]Institute for Software Research, Carnegie Mellon University, USA
[3]Instituto de Telecomunicações, Universidade do Porto, Portugal

***Abstract*—Modern computing systems are often formed by multiple components that interact with each other through the use of shared resources (*e.g.*, CPU, network bandwidth, storage). In this paper, we consider a representative scenario of one such system in the context of an Internet of Things application. The system consists of a network of self-adaptive cameras that share a communication channel, transmitting streams of frames to a central node. The cameras can modify a quality parameter to adapt the amount of information encoded and to affect their bandwidth requirements and usage. A critical design choice for such a system is scheduling channel access, *i.e.*, how to determine the amount of channel capacity that should be used by each of the cameras at any point in time. Two main issues have to be considered for the choice of a bandwidth allocation scheme: (i) camera adaptation and network access scheduling may interfere with one another, (ii) bandwidth distribution should be triggered only when necessary, to limit additional overhead. This paper proposes the first formally verified event-triggered adaptation scheme for bandwidth allocation, designed to minimize additional overhead in the network. Desired properties of the system are verified using model checking. The paper also describes experimental results obtained with an implementation of the scheme.**

I. Introduction

Modern computing systems in which a multitude of devices compete for network resources suffer from performance issues derived from inefficient bandwidth allocation policies. This problem is often mitigated in bandwidth-constrained systems by introducing run-time device-level adaptations (e.g. adjustment of operation parameters) to ensure correct information transmission [1], [2]. Adapting their behavior, the devices are capable of consequently adjusting their bandwidth requirements. Such scenarios present two main issues: (i) the adaptation at the device level can interfere with network allocation policies; (ii) it is quite difficult to obtain formal guarantees on the system's behavior, given that multiple adaptation strategies (network distribution and device-level adaptation) are active at the same time – and hence the presence of multiple independent control loops may lead to interference and result in disruptive effects [3].

In this paper, we tackle the two aforementioned issues in bandwidth allocation, ensuring the satisfaction of formal properties like convergence to a steady state. We apply our method to a camera surveillance network, in which self-adaptive cameras compete for network resources to send streams of frames to a central node.

The cameras adapt the quality of the transmitted frames every time a new frame is captured. To ensure the satisfaction of control-theoretical properties, a network manager is triggered periodically to schedule network access [4]. The periodic solution is desirable because it is equipped with a formal guarantee of convergence of the system to a single equilibrium in which all cameras are able to transmit their frames, if this equilibrium exists. In the opposite case, the time-triggered action guarantees that no camera can monopolize the network.

Despite this desirable property, the periodic solution has also severe shortcomings that are mainly related to the choice of the triggering period. The system may be too slow in reacting to camera bandwidth requirements if the period of the manager is too large. On the contrary, the system may exhibit poor performance due to the overhead caused by unnecessary actions, if the network manager is triggered too frequently – given a fixed number of cameras, the overhead of the network manager execution is approximately constant, so reducing the network manager period results in higher impact on the network operation. These limitations can effectively harm the performance of the system and should be taken into account when designing a network allocation strategy. Based on these considerations, this paper introduces an event-triggering policy for the network manager that minimizes the impact of the execution overhead on network performance, while taking into account the dynamic needs of the devices, induced by physical constraints and environmental factors – in our case study, for example, image size fluctuations derived from changes in the scenes captured by the cameras.

This paper provides the following contributions:

- A formal model for the problem of allocating bandwidth to adaptive devices. We cast this into the problem of a set of cameras collaborating to deliver the best overall system performance by modifying their bandwidth requirements

Acknowledgments: Gautham Nayak Seetanadi, Karl-Erik Årzén, and Martina Maggio are members of the LCCC Linnaeus Center and the ELLIIT Excellence Center at Lund University. This work was partially supported by the Swedish Research Council (VR) projects "Feedback Computing" and "Power and temperature control for large-scale computing infrastructures", and by the Portugal Regional Operational Programme's project "NanoSTIMA". This material is based on research sponsored by AFRL and DARPA under agreement number FA8750-16-2-0042. 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 or the U.S. Government.

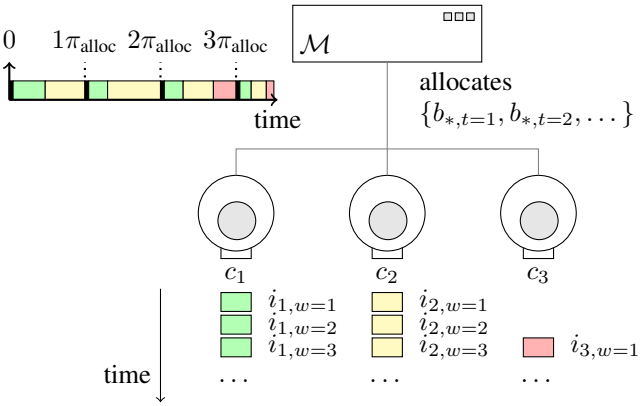

Fig. 1. System architecture.

and the quality of the encoded frames.

- Application of model checking to the event-triggered network manager. An event-triggered solution limits the formal guarantees provided with classical control-theoretical tools [5], which are based on the assumption that the network manager is triggered periodically. Nevertheless, we show how formal convergence guarantees can be achieved employing model checking, even when the network manager is not periodically triggered. We also verify additional properties and synthesize an optimal triggering strategy that minimizes the system's operational cost.
- Implementation and testing of the solution, to combine the theoretical guarantees with experimental validation.

II. Time-triggered activation

This section introduces the system and the notation used in the rest of the paper. The system is composed of a central node receiving video streams from a set of cameras, $\mathcal{C}_t = \{c_1, \dots, c_n\}$, where t represents the current time instant and n is the number of active cameras at time t. The central node also runs a network manager $\mathcal{M}$, in charge of distributing the available network bandwidth $\mathcal{H}$, e.g., $\mathcal{H} = 4Mbps$.

Figure 1 shows the system architecture when there are three cameras that capture frames and the network manager that determines the network access pattern for the cameras. In the network access timeline, black slots are used to show when the network manager uses the network, while the other colors represent the cameras transmitting frames. The third camera, c_3, is turned on when the first two, c_1 and c_2, have already transmitted two frames.

This section assumes a time-triggered activation scheme for the network manager, where the manager periodically senses the performance of the cameras and chooses the bandwidth distribution for the next activation period.

A. The camera

This subsection describes the behavior of the cameras where c_p with $p \in \{1, \dots, n\}$ denotes camera p. The camera captures a stream of frames. Each of these frames is compressed by an adequate encoder – *e.g.*, MJPEG – and sent to the central node via the network. The stream of frames is denoted by $I_p = \{i_{p,1}, \dots, i_{p,m}\}$, where p is the camera identifier and m is the cardinality of the set of frames (the longer the system runs, the more frames each camera produces). Each element $i_{p,w}$ in the set, $w \in \{1, \dots, m\}$, has the following characteristics.

The value $q_{p,w}$ represents the *quality* used for frame encoding, as in MJPEG. The quality is an integer number between 1 and 100, initialized using a parameter $q_{p,0}$, and loosely represents the percentage of information preserved during encoding. The value $\hat{s}_{p,w}$ indicates the estimate of the *size* of the encoded frame. For each of the cameras, depending on the resolution used for the recording and on actual manufacturer parameters, the frame size has a maximum and a minimum value, respectively denoted by $s_{p,\max}$ and $s_{p,\min}$, which we assume to be known.

The relationship between the quality used for the encoding $q_{p,w}$, which can be changed by the camera, and the size of the resulting frame $s_{p,w}$ is rather complex (see [6] for exponential models). This complexity is explained by the many factors on which the relationship between quality and frame size depends, including but not limited to the scene that the camera is recording (e.g., the amount of artifacts in the scene), the sensor used by the camera manufacturer, and the amount of light that reaches the sensor. In this work we approximate this relationship using the following affine model

$$\hat{s}^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,\max} + \delta s_{p,w}, \tag{1}$$

where $\delta s_{p,w}$ represents a stochastic disturbance on the frame size. We then saturate the result to ensure that the actual size is between the minimum and the maximum size:

$$\hat{s}_{p,w} = \max\{s_{p,\min}, \min\{s_{p,\max}, \hat{s}^*_{p,w}\}\}. \tag{2}$$

The model above is used to synthesize a controller for the camera that adapts its behavior. The camera automatically changes the quality $q_{p,w}$ to match the amount of network bandwidth that it can use, using an Adaptive Proportional and Integral (PI) controller similar to the one developed in [7][1]. The quality parameter $q^*_{p,w}$ is the control signal and roughly corresponds to the compression level for the frame. The controller uses as a setpoint the channel size dedicated to the transmission of the w-th frame $B_{p,w}$, and measures the size of the frame $s_{p,w}$. In computing the error $e_{p,w}$ we normalize the difference between the setpoint and the measured value, dividing it by the setpoint $B_{p,w}$.

$$\begin{aligned} e_{p,w} &= \frac{\overbrace{B_{p,w-1} - s_{p,w-1}}^{\text{normalized error}}}{B_{p,w-1}} \\ q^*_{p,w} &= k_{p,\text{Proportional}} \cdot e_{p,w} + k_{p,\text{Integral}} \cdot \sum_{t=1}^{w-1} e_{p,t} \end{aligned} \tag{3}$$

The integral action ensures that the stationary error is zero, i.e., that the setpoint is reached whenever possible. In some cases, reaching the setpoint may not be possible, due to the presence of saturation thresholds. We saturate the computed quality $q^*_{p,w}$ using the minimum and maximum quality values which we set at $q_{\min}$ and $q_{\max}$ respectively[2],

$$q_{p,w} = \max\{q_{\min}, \min\{q_{\max}, q^*_{p,w}\}\}. \tag{4}$$

[1]Given the model in Equation (2), an adaptive PI controller is capable of achieving a zero steady-state error and selecting the desired quality.

[2]Ideally, the quality is a number between 1 and 100, since it represents the compression level. However, we impose saturation levels that are based on our prior experience with the equipment, using $q_{\min} = 15$ and $q_{\max} = 85$.

The gains k_p and k_i are parameters of the controller inside the camera and determine how aggressive the adaptation is. We also implemented an anti-windup mechanism in the controller.

B. The network manager

To determine how to distribute the network bandwidth we use the approach proposed in [8] for CPU allocation and extend it to handle network bandwidth allocation. The network has a fixed capacity $\mathcal{H}$. The network manager $\mathcal{M}$ is in charge of allocating a specific amount of the available network bandwidth to each of the cameras. For every instant of time t at which the network manager is invoked, $\mathcal{M}$ selects a vector $b_{*,w}$, whose elements sum to one.

$$\forall t, \mathcal{M} \quad \begin{array}{l}\text{selects } b_{*,t} = [b_{1,t}, \ldots, b_{n,t}] \\ \text{such that } \sum_{p=1}^{n} b_{p,t} = 1\end{array} \tag{5}$$

This means that each of the elements of $b_{*,t}$ determines the fraction of the available bandwidth that is assigned to each video stream until the next network manager activation.

The assignment is enforced periodically, in a Time Division Multiplexed Access (TDMA) fashion. The network manager uses an allocation period $\pi_{\text{alloc}} = 30ms$ during which each active camera is expected to transmit a frame.

We denote by t_w the start time of the transmission of the w-th frame and with $t_{\mathcal{M},w}$ the time when the network manager computed the most recent bandwidth distribution vector $b_{*,w}$ when the w-th frame transmission starts. The manager allows camera c_p to transmit data for the w-th frame for an amount of time that corresponds to the computed fraction of the TDMA period $b_{p,t_{\mathcal{M},w}} \cdot \pi_{\text{alloc}}$. The total amount of data that c_p is allowed to transmit for the w-th frame is $B_{p,w}$.

$$B_{p,w} = b_{p,t_{\mathcal{M},w}} \cdot \pi_{\text{alloc}} \cdot \mathcal{H} \tag{6}$$

If the size of the encoded frame is greater than the amount of data that the camera can transmit, $s_{p,w} > B_{p,w}$, the frame is dropped, as it would be outdated for the next transmission slot. The current transmission slot is lost and cannot be reclaimed by any other camera.

The network manager is periodically triggered with period $\pi_{\mathcal{M}}$, which must be a multiple of π_{alloc} and a parameter in our implementation. In its first invocation, at time 0, the manager equally divides the available bandwidth among the cameras. The following network manager interventions, happening at times $\{\pi_{\mathcal{M}}, 2\pi_{\mathcal{M}}, 3\pi_{\mathcal{M}}, \ldots\}$ assign the bandwidth based on the following relationship, from [8], where the index t denotes the current time instant and $t+1$ the following one.

$$b_{p,t+1} = b_{p,t} + \varepsilon \cdot \{-\lambda_{p,t} \cdot f_{p,t} + b_{p,t} \cdot \sum_{i=1}^{n} [\lambda_{i,t} \cdot f_{i,t}]\} \tag{7}$$

Equation (7) decides the bandwidth assignment for camera p in the next time instant, and introduces the following parameters: (i) ε is a small constant used to limit the change in bandwidth that is allocated at every step. The choice of a suitable value for ε depends on the trade-off between the responsiveness of the manager (higher values making it converge faster, in principle, but also making it likely to have overshoots) and its robustness to disturbances (lower values increase convergence time favoring a more stable behavior in the presence of transient disturbances)[3]; (ii) $\lambda_{p,t} \in (0,1)$ is a weight that denotes the fraction of adaptation that should be carried out by the network manager. A lower $\lambda_{p,t}$ value indicates that the network manager is less willing to accommodate the needs of the p-th camera. The importance of this value lies in the relative difference between the values assigned to all the cameras. If all the cameras have an equal $\lambda_{p,t}$, the network manager is not going to favor any of them. If one of the cameras has a higher value with respect to the others, the network manager is "prioritizing" the needs of that camera over the others. In the following, we assume that $\lambda_{p,t}$ does not change during execution, and use λ_p as a shorthand, for simplicity. A change in the value of λ_p has no impact on our analysis, and can be used to change the network manager preference during runtime; (iii) $f_{p,t}$ is a function that we call the *matching function*, which expresses to what extent the amount of network bandwidth given to the p-th camera at time t is a good fit for the current quality.

The matching function f_{p,t_w} is a concept introduced in [8] and should determine a match between the quality $q_{p,w}$ (which influences the frame size $s_{p,w}$) and the resource allocation $B_{p,w}$ available for the camera when the transmission of the w-th frame happens. In our implementation, we choose to use $(B_{p,w} - s_{p,w})/B_{p,w}$, also equal to the normalized error $e_{p,w}$ in Equation (3) as the matching function. For the analysis in [8] to hold – which proves properties such as *starvation avoidance*, *balance*, *convergence*, and *stability*, discussed in Section II-C –, the matching function should satisfy the following properties:

(P1a) $f_{p,t_w} > 0$ if $B_{p,w} > s_{p,w}$,
(P1b) $f_{p,t_w} < 0$ if $B_{p,w} < s_{p,w}$,
(P1c) $f_{p,t_w} = 0$ if $B_{p,w} = s_{p,w}$;
(P2a) $f_{p,t_w} \geq f_{p,t_{w-1}}$ if $q_{p,w} \leq q_{p,w-1}$,
(P2b) $f_{p,t_w} \leq f_{p,t_{w-1}}$ if $q_{p,w} \geq q_{p,w-1}$;
(P3a) $f_{p,t_w} \geq f_{p,t_{w-1}}$ if $b_{p,t_w} \geq b_{p,t_{w-1}}$,
(P3b) $f_{p,t_w} \leq f_{p,t_{w-1}}$ if $b_{p,w} \leq b_{p,t_{w-1}}$.

These properties entail that the matching function must be positive if the bandwidth given is abundant, negative if it is insufficient, and zero if the match is perfect (P1); that the matching function must increase when the quality is decreased and decrease with increased quality (P2); and, finally, that the matching function increases when more bandwidth is assigned and decreases when bandwidth is removed (P3).

Our implementation choice for the matching function – $e_{p,w}$, from Equation (3) – automatically satisfies properties (P1a-c) and (P3a-b). If one assumes the disturbance $\delta s_{p,w}$ to be negligible, it is possible to use Equation (1) to verify that properties (P2a) and (P2b) hold. Notice that the matching function corresponds to the normalized error used by the camera controller described in Section II-A. In the following we will use $f_{p,t}$ to indicate the value of the matching function over time and f_{p,t_w} to indicate the value of the matching function computed for the frame w transmitted at time t_w. Also, we use b_p to indicate the sequence of bandwidth assignments for camera c_p over time $\langle b_{p,0}, b_{p,1}, \ldots \rangle$ and q_p to indicate

[3]Typical values for ε are between 0.1 and 0.6.

the sequence of quality per frame chosen by the camera $\langle q_{p,w=0}, q_{p,w=1}, \dots \rangle$. An example timeline can be seen in Appendix A.

C. System behavior

From a theoretical perspective, the resource allocation and camera adaptation schemes are not different from the CPU allocation and service level adjustment proposed in [8]. The behavior of the system has therefore been analyzed and some properties have been proven [5]. Here we only give a brief summary of these properties.

- *Starvation avoidance.* A positive amount of resource is guaranteed for all cameras that have a non-zero weight, i.e., $\forall\{t,p\}, \lambda_p > 0 \Rightarrow b_{p,t} > 0$.
- *Balance.* The balance property holds in case of overload conditions. The network is overloaded at time t when the capacity $\mathcal{H}$ is not enough to guarantee that all the cameras have a matching function greater or equal to zero $(\forall p, q_{p,t} = q_{\min}, f_{p,t} \leq 0) \wedge (\exists i, f_{i,t} < 0)$. In this case, it is guaranteed that no camera can monopolize the available bandwidth at the expense of the others.
- *Convergence.* The amount of bandwidth allocated to each camera and the streams' quality converge to a stationary point which corresponds to a fair resource distribution (a distribution in which the matching function is zero for all the cameras) whenever possible (in non-overload conditions) both in case of synchronous (the cameras update their quality at the same time) [5, Theorem 4.1] and asynchronous updates (the cameras update their quality potentially at different times) [5, Theorem 4.2].
- *Scalability.* One of the reasons behind this resource allocation strategy is its linear time complexity. The bandwidth to be allocated can be computed in linear time with respect to the number of cameras, according to Equation (7). This can scale to many cameras and is therefore beneficial in a complex setup. Appendix C shows the overhead introduced by the network manager's execution for a varying number of cameras. The execution overhead shown is per network manager activation. With periodic execution, for large period values, the impact of the overhead tends to become negligible, but the system is less responsive. On the contrary, when the network manager period is short, the overhead is significant, but the system is more responsive.

III. Towards event-triggered activation

As briefly summarized in Section II-C, using time-triggered activation for the network manager allows us to guarantee some desired properties of the system behavior. One key assumption, needed to derive the formal proof of convergence, is the periodic computation of the resource distribution.

Time-triggered activation, however, has two major drawbacks. The first drawback is the difficulty in choosing an appropriate value for the period $\pi_{\mathcal{M}}$, which has a remarkable impact on the system performance. The second drawback is the additional overhead imposed by periodic activation of the network manager, when only negligible adjustments are needed. In the following, we briefly discuss these drawbacks, using data from our implementation to back up our claims.

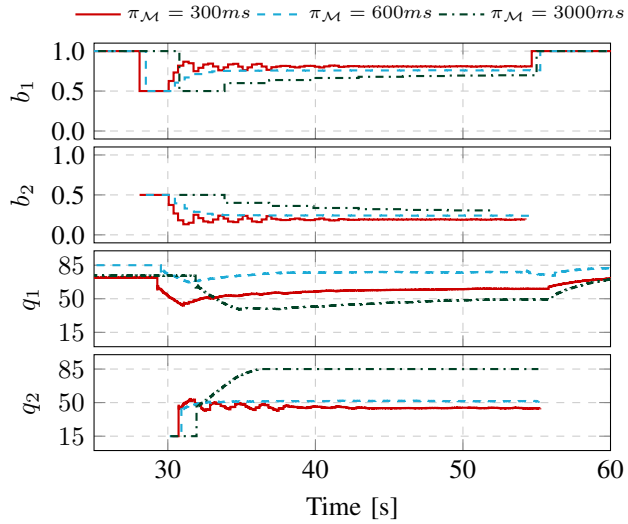

Fig. 2. Time-triggered activation with different network manager periods: bandwidth allocation (b_1 and b_2) and image quality (q_1 and q_2).

A. Period choice

An experiment was conducted with a network composed of two cameras and a network manager, running the system for $90s$. The first camera, c_1, was active for the entire duration of the experiment, $[0s, 90s]$, pointing at a scene with many artifacts. The second camera, c_2, was transmitting during the interval $[30s, 55s]$ and directed towards a scene that was easier to encode, with fewer artifacts. More precisely, c_2 is turned on a few seconds before, notifies the network manager and gets some bandwidth allocated (around $28s$), but starts transmitting frames only after an initial handshake. The values of λ_1 and λ_2 are equal and set to 0.5. $k_{1,\text{Proportional}}$ and $k_{2,\text{Proportional}}$ are set to 10 and the values of $k_{1,\text{Integral}}$ and $k_{2,\text{Integral}}$ are set to 10. Finally, the value of ε in the network manager is set to 0.5.

To highlight the criticality of this choice, we ran the experiment using different periods: $\pi_{\mathcal{M}} = 300ms$, $\pi_{\mathcal{M}} = 600ms$, and $\pi_{\mathcal{M}} = 3000ms$. Figure 2 shows the percentage of bandwidth assigned to the cameras (b_1 and b_2) and the quality set by the camera controllers (q_1 and q_2) in the three different cases, in the most interesting time interval, when the second camera is joining and when both cameras are active.

When the network manager runs with period $\pi_{\mathcal{M}} = 600ms$, the system shows the desired behavior. Immediately after c_2 joins the network, the bandwidth is automatically redistributed, half of it being assigned to the new camera. The reduction of the available bandwidth for c_1 causes a quality drop. While the quality q_1 is decreased, the quality q_2 settles to a value that allows the camera to transmit the frames. On the contrary, when the network manager period is longer ($\pi_{\mathcal{M}} = 3000ms$), the system still reaches a steady state, but the quality q_2 is much higher than in the previous case, and the reduction in quality for q_1 is substantial, which is undesirable given that the first camera is capturing a scene with more artifacts than the second one. Looking at the frame size, one discovers that the higher quality does not come with additional information being transmitted, since the camera is pointing to a scene that has fewer artifacts. Hence, in this case the network manager

would have done a much better job accommodating the needs of c_1. Finally, the plots for $\pi_{\mathcal{M}} = 300ms$ show that the system converges to values that are similar to the ones observed when $\pi_{\mathcal{M}} = 600ms$. However, the quality of the image produced by the first camera is lower than it could be, and the system shows oscillations.

For this specific execution conditions, the choice of the period $\pi_{\mathcal{M}} = 600ms$ is the best among the three tested choices, but different execution conditions can lead to other values being preferable. To develop a solution that is not tied to a specific use case and is applicable in practice, one has to design an event-based intervention policy that triggers network manager interventions only when necessary.

B. Resource allocation overhead

Even if we assume that we have the knowledge required to select the best period for a specific scenario, when the system reaches an equilibrium, it is in a fixed point where things do not change unless there is a change in the execution conditions. In such cases, there is no need to carry out substantial changes in the amount of resources assigned and one should carefully evaluate the overhead of computing a new resource allocation, balancing it against the benefit obtained in terms of overall system performance.

To evaluate the computational overhead of determining a new resource distribution, we measured the time it takes for the network manager to: (i) retrieve data about the size of the last frame sent by the cameras, compute the matching function for the cameras using $f_{p,t} = e_{p,w}$ as specified in Equation (3), (ii) compute the new resource distribution vector $b_{*,t}$ according to Equation (7), and (iii) inform the cameras about their transmission slots' duration.

We collected 5000 samples of the duration of the manager's execution with a network of two cameras and computed the following statistics. The average overhead is $0.0030s$, with a standard deviation of $0.0284s$ (1% of the total time if the activation period is $\pi_{\mathcal{M}} = 300ms$, 0.5% when $\pi_{\mathcal{M}} = 600ms$ and 0.1% when $\pi_{\mathcal{M}} = 3000ms$). Maximum and minimum values are respectively $0.9863s$ and $0.0001s$. The maximum value is most likely caused by additional workload and should happen infrequently. However, if this was the real computation time, the network would not be able to operate at all when $\pi_{\mathcal{M}} = 300ms$ or $\pi_{\mathcal{M}} = 600ms$, since the entire time in the period is spent for the network manager computation and none is left for camera transmissions. When $\pi_{\mathcal{M}} = 3000ms$, 33% of the time in the period would be devoted to the network manager execution. Note that the time complexity for the execution of the network manager increases linearly with the number of cameras as specified in Section II-C, which is the best alternative when the network manager should take into account the needs of all the cameras (meaning that the network manager needs to at least compute a performance metric for each of the cameras) [8]. This means that using a different algorithm for the network manager computation will achieve no significant scalability benefit and improvements on execution time should be obtained in a different way – *e.g.* skipping executions.

Executing the network manager fewer times reduces the overhead for the system. This indicates that avoiding unnecessary calls to the network manager code would be very beneficial for the system and would improve its performance. Leaving the periodic solution in place, this benefit can be achieved using a longer period for the periodic activation scheme. However, the experiment presented in Section III-A demonstrated that a larger activation period is not always a viable alternative. This motivates our investigation of an event-triggered activation scheme.

IV. Event-triggered activation

The purpose of an event-based solution is to quickly react only when needed, avoiding unnecessary interventions. To design an event-based network manager, we need to define a set of event-triggering rules that determine when the network manager is invoked and can intervene, and to describe how the corresponding events are handled. In the following, we use t^- to indicate the time instant that precedes t, and – at time t – we denote the most recent frame that camera c_p transmitted by i_{p,w_t}. Using this notation, w_t represents the index of the last frame the camera transmitted at time t.

The choice done in this paper is to employ the following two triggering rules and to trigger an event at time t if:

- $\mathcal{C}_{t^-} \neq \mathcal{C}_t$, *i.e.*, the network manager is triggered if the set of cameras changes. In this case, one or more cameras are either removed or added to the set[4]. The network manager handles this event by re-distributing the bandwidth equally among the active cameras at time t, $\forall c_p \in \mathcal{C}_t = \{c_1, \ldots, c_n\}, b_{p,t} = 1/n$. At time t, after the network manager performs the allocation, it sets a timeout τ_{unresp}, a multiple of π_{alloc}. If there is one or more cameras that have not transmitted any frame in the time interval $(t, t+\tau_{\text{unresp}}]$, said cameras are removed from the set, triggering the same event at time $t + \tau_{\text{unresp}}$. This mimics the behavior of the time-triggered network manager, as seen in the example shown in Figure 2.
- $(\exists c_p \text{ s.t. } \tau_{\text{thr}} < |e_{p,w_t}|) \wedge (t \bmod \pi_{\text{alloc}} = 0)$, *i.e.*, when the transmission round completes for all the cameras, at least one of them has a normalized error whose absolute value is higher than a specific threshold τ_{thr}, a parameter of the triggering policy. Just like in its time-triggered counterpart, the event-based version of the network manager allocates the bandwidth fractions as specified in Equation (7).

With this event-triggering policy, the critical implementation choice is the value τ_{thr}. The normalized error will often be included in the interval $[-1, 1]$, although this is not guaranteed by the expression in Equation (3). Nonetheless, when the absolute value of the normalized error is higher than 1, the camera has encoded a frame whose size is double with respect to the channel size. Therefore, we assume that a value of 1 or higher should always trigger the network manager intervention. We determine that τ_{thr} is bounded to the open interval $(0, 1)$. Within the given interval, assigning a low value

[4]Given the TDMA bandwidth reservations, changes to the set of cameras that occur when a round is in progress are deferred to the closest multiple of π_{alloc}, therefore $(t \bmod \pi_{\text{alloc}} = 0)$.

to τ_{thr} forces the network manager to intervene often. On the contrary, when τ_{thr} approaches 1, the network manager is triggered less often and relies more on the adaptation done by the cameras.

More complex triggering policies can be defined, some examples can be found in [9], [10]. Despite their complexity, in event- and self-triggered controllers the triggering rules are usually based on measurements from the system.

The main drawback of the transition to an event-triggered network manager is that the properties proved in Section II-C do not necessarily hold in the event-based implementation. The proof of the properties relies on the fact that the network manager acts at pre-determined time instants. This cannot be guaranteed in an event-based implementation, in which manager interventions depend on the triggering rule.

In the following section, we therefore use model checking to obtain formal guarantees on the behavior and the convergence of the system. We therefore provide a background on model checking and show the results obtained with our model.

V. Formal methods

This section introduces our use of model checking to verify properties and to select optimal alternatives for the resource manager implementation. We first introduce some background on Probabilistic Model Checking (PMC) in Section V-A. Then we introduce the model of the system in Section V-B and the properties we prove in Section V-C. Finally, we discuss an optimal network manager activation strategy generated by our model checker in Section V-D.

A. Background on model checking

Probabilistic Model Checking (PMC) [11] is a set of formal verification techniques that enable modeling of systems that exhibit stochastic behavior, as well as the analysis of quantitative properties that concern costs/rewards (e.g., resource usage, time) and probabilities (e.g., of violating a safety invariant).

In PMC, systems are modeled as state-transition systems augmented with probabilities such as discrete-time Markov chains (DTMC), Markov decision processes (MDP), probabilistic timed automata (PTA), and properties are expressed using some form of probabilistic temporal logic, such as probabilistic reward computation-tree logic (PRCTL) [12], which state that some probability or reward meet some threshold.

An example of a probability-based PRCTL property is $\mathsf{P}_{\geq 1}[\mathsf{G}\ \mathcal{H}_{\mathsf{alloc}} = \mathcal{H}]$, which captures the invariant "the allocated bandwidth to the cameras is always equal to the total available bandwidth." In this property, the probability quantifier P states that the path formula within the square brackets (globally[5] $\mathcal{H}_{alloc} = \mathcal{H}$) is satisfied with probability 1. We assume that $\mathcal{H}_{alloc}$ is the sum of allocated bandwidth to all cameras.

Reasoning about strategies[6] is also a fundamental aspect of PMC. It enables checking for the existence of a strategy that is able to satisfy a threshold or optimize an objective expressed in PRCTL, in systems described using formalisms that support the specification of nondeterministic choices, like MDP.

For example, employing the PRCTL reward minimization operator $\mathsf{R}^{\mathsf{r}}_{\mathsf{min}=?}[\mathsf{F}\ \phi]$ enables the synthesis of a strategy that minimizes the accrued reward r along paths that lead to states finally satisfying the state formula ϕ. An example of a property employing this operator for strategy synthesis is $\mathsf{R}^{\mathsf{nmi}}_{\mathsf{min}=?}[\mathsf{F}\ \mathsf{t} = \mathsf{t}_{\mathsf{max}}]$, meaning "find a strategy that minimizes the number of total network manager interventions (captured in reward nmi) throughout a system execution period $(0, \mathsf{t}_{\mathsf{max}})$, *i.e.*, when time is equal to $\mathsf{t}_{\mathsf{max}}$."

In this section, we illustrate probabilistic modeling of the camera network system using the high-level language of the probabilistic model checker PRISM [15], in which DTMCs and MDPs can be expressed as processes formed by sets of commands like the following

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

where $guard$ is a predicate over the model variables. Each update u_i describes a transition that the process can make (by executing $action$) if the guard is satisfied. An update is specified by giving the new values of the variables, and has a probability $p_i \in [0, 1]$[7]. In an MDP model, multiple commands with overlapping guards introduce local nondeterminism and allow the model checker to select the alternative that resolves the nondeterministic choice in the best possible way with respect to the property captured at the strategy synthesis level.

B. System model

The model consists of a set of *modules*, each of them capturing the behavior of one of the entities in the system. The entire model is composed of one module per camera, one module for the network manager and one module for the scheduler. The model imposes turns in executing all the components: it starts with the cameras performing their actions – sending a frame each – and then continues with the scheduler, which checks if the network manager should be executed, calling it if necessary. During its execution, the network manager changes the bandwidth allocation. Then the scheduler passes again the turn to the set of cameras. The PRISM code for the camera and the manager is displayed in Appendix B, while in the following we describe the Scheduler, shown in Listing 1.

The scheduler maintains two state variables, choice_done and calling_nm (lines 4-5), which keep track of when the choice about invocation is made and if the network manager is to be called or not. The code in this model can be employed in two alternative ways. If the model checker is executed with constant synth_schedule set to true (line 1), it synthesizes a strategy by resolving the nondeterminism between actions best_dont (lines 8-12) and best_do (lines 13-17), whose guards overlap. The strategy minimizes a cost function defined

[5]The G modality in PRCTL, read as "globally" or "always" states that a given formula is satisfied across all states in a sequence that captures a system execution trace. The semantics of G in PRCTL is analogous to those found in other temporal logics like CTL [13] or LTL [14].

[6]Strategies – also referred to a as *policies* or *adversaries* – resolve the nondeterministic choices of a probabilistic model (in this case MDP), selecting which action to take in every state.

[7]In our model, we do not take advantage of probabilities. For each action, we define a single update rule, with implicit probability 1. However, we could incorporate probabilistic choices with limited and localized modifications. Probabilistic update rules can be beneficial when modeling physical phenomena like disturbances, occurring with a certain probability and affecting the modules, e.g., changes in frame size due to artifacts in the images.

in this case as a penalty for every dropped frame and for each network manager intervention (cf. Section V-D).

```
1  const bool synth_schedule = true; // prism synthesis
2  const bool event = true; // event−based version
3  module scheduler
4    choice_done: bool init false;
5    calling_nm: bool init false;
6    // prism synthesis (synth_schedule = true) best_dont and best_do:
7    // conditions are the same, choices are different
8    [best_dont] (synth_schedule) & // if prism synthesis
9                (turn = sche) &
10               (rounds < max_frames) &
11               (!choice_done) −> // choice is not yet done
12     (calling_rm' = false) & (choice_done' = true); // choice no
13   [best_do] (synth_schedule) & // if prism synthesis
14               (turn = sche) &
15               (rounds < max_frames) &
16               (!choice_done) −> // choice is not yet done
17     (calling_rm' = true) & (choice_done' = true); // choice yes
18   // decision without synthesis
19   // if some of the cameras set want_nm to true, call the manager
20   [decision] (!synth_schedule) & // only if not prism synthesis
21               (turn = sche) &
22               (rounds < max_frames) &
23               (!choice_done) −> // choice is not yet done
24     (calling_nm' = want_nm) & (choice_done' = true);
25   // network manager calls or not after decision is made above
26   [] (turn = sche) & (rounds < max_frames) & (choice_done) &
27      (calling_nm) −> // let's call the manager
28      (turn' = nmng) & (calling_nm' = false); // giving turn to manager
29   [] (turn = sche) & (rounds < max_frames) & (choice_done) &
30      (!calling_rm) −> // not calling the manager (or already called)
31      (turn' = cam1) & // giving turn to first camera
32      (rounds' = rounds + 1) & // advancing rounds
33      (choice_done' = false) & // reset for next iteration
34      (want_nm' = event ? false : true) & // time−based acts every time
35      (nmchange' = false); // have not performed any change
36 endmodule
```

Listing 1. Scheduler module description

Alternatively, if the model checker is called with the variable synth_schedule set to false, the scheduler can be executed either in the time-triggered version, where the period is simply a constant representing π_{alloc} or in the event-triggered version (the constant event should be set to true in line 2). In the event-triggered version, the action labelled decision (lines 20-24) determines if the network manager should be called or not. If the network manager is not to be called, the last action (lines 29-35) is performed and the turn is passed to the camera. In the opposite case (lines 26-28), the network manager is called and, when executed, it returns the control to the scheduler that then passes the turn to the first camera (lines 29-35).

C. Formal guarantees

With respect to the properties discussed in Section II-C, *starvation avoidance* and *balance* depend on how Equation 7 is constructed and are not influenced by the event-triggering rule, as long as the network manager is triggered at least one time when the set of cameras changes. Our triggering scheme guarantees that the network manager is triggered once when a new camera joins the system or leaves it, therefore starvation avoidance and balance are satisfied.

On the contrary, in the case of *convergence* we have no *a priori* guarantee that the property holds in the event-triggered version of the network manager. In fact, the proof of convergence depends on the assumption that the network manager is periodically triggered and changes the resource assignment at specific time instants. We therefore want to express the property using a PRCTL formula and check it resorting to the probabilistic model checker capabilities.

We can formalize convergence for our system as the PRCTL formula $\mathsf{P}_{\geq 1}[\mathsf{F}(\mathsf{G!}(\mathsf{any\_change\_event}))]$, where any_change_event is defined in our formal model as the disjunction $\mathsf{nmchange} \vee \mathsf{c1change} \vee \ldots \vee \mathsf{cnchange}$. The property can be interpreted as "the probability that the system will eventually reach a state for which no change event occurs in the remainder of the execution is 1". In practice, this translates to checking that for all potential execution paths, the system always reaches a state from which the resource manager does not make any further updates to bandwidth assignment, and the cameras do not make any further adjustments in quality.

We checked this property for the event-triggered version of our model in executions with $\mathsf{max\_frames} = 30$ (which means that convergence must occur before a cycle of 30 frames per camera is completed), for a network of two cameras, and for all the combination of values of λ_1, λ_2 and τ_{thr} belonging to the vector $[0.01, 0.02, \ldots 0.99]$, and assessed its satisfaction.

In addition to the assessment of properties described in Section II-C, we can also conduct sanity checks, like assessing the satisfaction of invariants. An example is checking that the system always takes advantage of all the available bandwidth, assigning it to the cameras. We can formalize this property as $\mathsf{P}_{\geq 1}[\mathsf{G}\ \mathsf{used\_bw} = \mathsf{max\_bw}]$, where used_bw is a formula defined as the summation of all bw_x variables in the model, where x is the camera number (see Listing 3, lines 4-5).

Finally, we can also compare different design alternatives for the triggering rules, to see which alternative exhibits the most desirable behaviour. One example is checking the number of resource manager interventions during the execution of the system, which should ideally be minimized. We can capture this property in PRCTL as $\mathsf{R}^{\mathsf{nm\_calls}}_{\mathsf{max}=?}[\mathsf{F}\ \mathsf{rounds} = \mathsf{max\_frames}]$, which can be interpreted as "maximum number of manager interventions (encoded in reward structure nm_calls) until the end of the execution (the maximum number of frames defined above as max_frames)". This property relies on the definition of the reward structure nm_calls that captures the total number of network manager interventions (encoded in variable nm_interventions in Listing 3, line 11).

D. Event-triggering policy synthesis

In addition to checking properties on a given version of the formal model, we can also use the model checker to synthesize triggering policies for the network manager that are optimal with respect to a specific property. The key idea behind the synthesis is leaving the intervention choice underspecified in the model as a nondeterministic choice between actions whose guards overlap – in our case between the scheduler actions best_dont and best_do (Listing 1, lines 8 and 13, respectively). In such a way, the model checker can resolve the nondeterminism by synthesizing a policy that optimizes an objective function embedded in a PRCTL formula.

In this case, we are interested in minimizing the undesirable behaviours of the system, that include the number of dropped frames, as well as the amount of network manager interventions. We wrap both metrics into

a single cost function that we label total_cost, encoded as the sum of penalty_frames $\cdot$ (drop1 + ... + dropn) and penalty_intervention $\cdot$ (nm_interventions), where dropx is the number of dropped frames for camera x at the end of the execution (Listing 2, line 23). The constants penalty_frames and penalty_interventions capture the relative importance of both penalized aspects of system operation, in our case set to 10 for dropped frames and 1 for network manager interventions. The two constants are set to separate the undesired behaviours by an order of magnitude and penalize dropped frames more than resource redistribution.

Based on the definition of total_cost, we can define a reward structure that captures its value at the end of the execution of the system, and employ it for synthesis in the PRCTL property $\mathsf{R}^{\mathsf{total\_cost}}_{\mathsf{min}=?}[\mathsf{F\ rounds} = \mathsf{max\_frames}]$, which instructs the model checker to "find a strategy that minimizes the penalty of operating the system based on the total_cost accrued throughout the execution of the system."

The synthesized strategy instructs the network manager to wait until the cameras have reached convergence (two transmitted frames) before acting twice to distribute network bandwidth in the best possible way. The optimal strategy in this case is the sequence ⟨best_dont, best_dont, best_do, best_do⟩ followed by an infinite sequence of best_dont. The synthesis of this "optimal" sequence does not take into account further changes that happen in the system – *e.g.*, the scene of one camera has more artifacts. The strategy minimizes the cost for operating the system in the current conditions, without knowledge of what will happen in subsequent time instants.

After determining the best policy for minimizing the total_cost, we can fix it on the model and check other properties like the one defined for convergence in Section V-C. We checked convergence, as well as the invariant for full bandwidth allocation, which were always satisfied for the same set of λ_1 and λ_2 (weights on camera vs. network manager adaptation) values described in V-C.

For some set of parameters, the total_cost of running the optimal policy is the same as some specific values for the threshold τ_{thr}. For some other set of parameters, on the contrary, the minimum total_cost that is achieved using this strategy cannot be achieved with any value of the threshold, showing that the threshold based policy is not necessarily optimal. In a static environment, the strategy synthesized by PRISM is the best choice to minimize the total_cost of operation. However, usually the execution environment is dynamic and the policy should react to changes that may happen, like additional artifacts in one of the images. Despite this strategy being optimal, at run time it should be coupled with an algorithm that detects when the strategy should be re-applied, and re-triggers the strategy start or the policy synthesis process. From a practical standpoint, the threshold-based policy shows very good properties and is therefore a good triggering rule, though potentially sub-optimal.

Another question that we can answer using model checking is finding the best threshold value for a specific situation. This is similar to the optimal strategy synthesis, and entails finding the best threshold value for the specific conditions that the system starts from, assuming that nothing else interferes – *e.g.*, additional artifacts in the images captured by the cameras. The result is the threshold that minimizes the total operational cost for the system, given the current status.

VI. Experimental results

This section introduces the experimental validation we conducted with our camera network. To dynamically change the amount of bandwidth allocated, we need an underlying architecture that supports reservations with bandwidth adaptation. For this, we use Flexible Time Triggered (FTT) paradigm [1], which enforces adaptive hard reservations. In our implementation we use the Switched Ethernet (SE) implementation FTT-SE [16]. FTT-SE uses trigger messages from the master (the network manager) to the slaves (the cameras) to change the allocated bandwidth, providing guarantees on minimum bandwidth allocation [2].

We show the behavior of our implementation with an experimental result with three or four physical units: the network manager and two or three cameras. Each unit runs Fedora 24. The first unit runs the network manager and has a Intel Core i7-4790, 8 core CPU with 32 GB RAM. The other units are off-the-shelf Logitech C270 cameras. Results with three cameras are shown in Appendix D. Notice that our experiments are stress tests, as the network bandwidth is not enough to transmit all the frames, and frame dropping must occur to guarantee correct operation.

Our experiment has the following setup: $k_{1,\text{Proportional}} = 10$, $k_{2,\text{Proportional}} = 10$, $k_{1,\text{Integral}} = 0.5$, $k_{2,\text{Integral}} = 1$, $q_{1,0} = q_{2,0} = 15$, $q_{1,\max} = q_{2,\max} = 85$, $q_{1,\min} = q_{2,\min} = 15$, $\lambda_1 = 0.7$, $\lambda_2 = 0.3$, $\varepsilon = 0.4$, $\pi_{\text{alloc}} = 30\,\text{ms}$, $\mathcal{H} = 4\,\text{Mbps}$. We deliberately set a low total available bandwidth to stress the system in under-provisioning conditions, and make sure that adaptation is needed. We ran the experiment varying the value of the threshold τ_{thr}, and with the PRISM strategy. We also computed the best threshold value with the PRISM model checker, discovering that its value is 0.3.

In the conducted experiment, both the cameras were added to the network simultaneously. Camera c_1 recorded a scene with many artifacts and c_2 recorded a simpler scene. Theoretically c_1 would require a larger amount of bandwidth generating larger sized images at lower qualities and c_2 would require a lower amount of bandwidth. Figure 3 shows the results of the experiment. The red dashed line represents the PRISM strategy, that settles providing more bandwidth to c_1 and less to c_2. Despite having less bandwidth, the quality of c_2 reaches the maximum level of 85, while the quality of c_1 oscillates to absorb changes in the scene and adapt to the current execution conditions. The other lines represent the execution with different values of the threshold (lower values of τ_{thr} are represented with lighter lines). Independently of what the threshold is, the event-triggered network manager behaves similarly to the PRISM strategy, allocating more bandwidth to c_1. Figure 4 shows the metrics collected for different schemes during the experiment. Looking at the interventions we can observe that the network manager acts more often

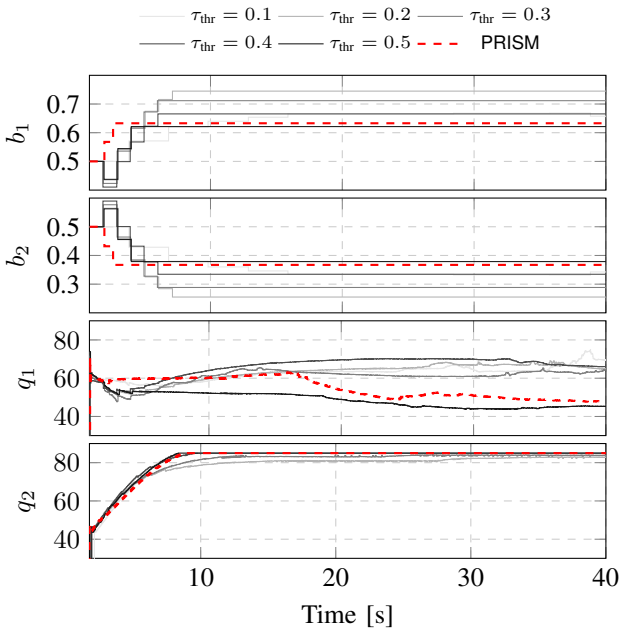

Fig. 3. Event-triggered activation with $\tau_{\text{thr}} \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the optimal strategy synthesized by PRISM: bandwidth allocation (b_1 and b_2) and image quality (q_1 and q_2).

when the threshold is lower, with the PRISM strategy having 2 interventions. Note that in the first 100 frames, the solution synthesized by the model checker drops approximately 10% less frames than any of the other solutions for both cameras, and minimizes overall cost.

This is consistent with the model checking results, which indicate that the PRISM strategy improves over all the possible threshold-based policies, and that an event-triggered policy with a threshold of $\tau_{\text{thr}} = 0.3$ minimizes total cost. When the system runs for more time, however, the cameras will record images with different artifacts, and the optimal policy and the the best threshold, computed by the model checker, will not necessarily hold anymore.

Indeed, if we compute the same metrics for a longer time frame during which the recorded images are subject to changes and modifications unknown to the PRISM strategy, the results differ. For all the alternatives, the percentage of dropped frames decreases, indicating that the system is capable of adjusting to even extreme underprovisioning. Computing the metrics for the entire run of the experiment, the lowest total cost is 13731, achieved when the system uses an event-triggering strategy with $\tau_{\text{thr}} = 0.1$, while the PRISM strategy achieves a cost of 14172. One of the drawbacks of the PRISM strategy is the lack of adaptation to real-time changes in the camera environment which might lead to bandwidth requirements different than the one initially allocated. This confirms the need for constant adaptation and re-evaluation of the optimal strategy, but it also demonstrates the potential for efficient solution generation using model checking.

VII. Related work

The topic of self-adaptive cameras has been investigated in the scope of video transmission over the Internet or in local area networks [17], [18], [19], [7], [20], [21], focusing on video transmission and image compression. The former led to protocols such as RTP, RTSP, SIP and their improvements. These protocols measure key network parameters, such as bandwidth usage, packet loss rate, and round-trip delays, to cope with network load conditions, controlling the load submitted to the network [22] or using traffic prioritization [23].

The latter led to standards such as MJPEG, JPEG2000, MPEG-4, H.264 and more recently MPEG-H and H.265 that explore redundant information within sequences of frames. These techniques frequently impose strong delays and additional processing in the camera.

Surveillance – as other domains like augmented reality [24], industrial supervised multimedia control [19], multimedia embedded systems [25], automated inspection [26] and vehicle navigation [27] – impose limitations on the acceptable delays. In these cases, image compression is frequently preferred to video compression for the lower latency incurred and lower memory and computing requirements. Nevertheless, any compression also incurs variability in transmission frame sizes that further complicates the matching with the instantaneous network conditions and motivated substantial research into adaptive techniques [19], [25], [7]. These works focused on adapting streams to what the network provides, without network scheduling. Scheduling can be achieved using network reservations (channels), as with RSVP or lower layer real-time protocols, with the risk of poor network bandwidth efficiency. The work in [6] addressed this problem using adaptive network channels provided by a global network manager that tracks the actual use that each camera is doing of its allocated bandwidth. In this paper, we use an approach based on control theory for quality adaptation similar to the one applied in [7]. On top of that, we complement the camera adaptation strategy with network bandwidth distribution. We also employ model checking to verify desirable properties of the system. Better performance can be obtained [20] using domain-knowledge to optimize the bandwidth allocation, but this paper assumes no prior knowledge. The behavior of cameras that transmit streams over wireless networks has also been investigated in [21], highlighting the need for adaptation, in this case reducing or increasing the amount of processing done at the node level. The approach solves a more complex problem with respect to the one discussed in this paper, but does not provide any analytic guarantee. In contrast, our solution is equipped with the guarantees obtained via model checking.

Probabilistic model checking has been employed to verify performance properties of a video streaming system in [28], where 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 [29]. In contrast with this work, we also utilize the model checker for strategy synthesis. Strategy synthesis via PMC has been used to optimize run-time properties of cloud-based systems, balancing performance and operation cost [30]. Contrary to all the mentioned papers, we use PMC to verify a class of properties that is typically related with control-theoretical guarantees, like convergence.

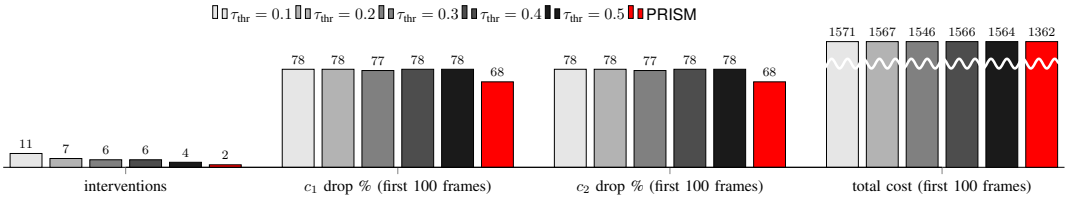

Fig. 4. Aggregate metrics for the experiment: resource manager interventions, dropped frames and cost.

VIII. Conclusion

This paper presented an event-triggered bandwidth allocation scheme designed to complement self-adaptive cameras that adapt the quality of their video streams to match specific bandwidth assignment. The main contributions provided by the paper are: (i) the design of an adaptation scheme capable of handling the requirements of multiple adaptive entities while preserving the independence of the single-entity adaptation, (ii) the implementation and testing of the adaptation scheme, and (iii) the use of model checking to verify desirable properties of the system.

The adaptation scheme has been tested with a network of up to three cameras and different strategies for the invocation of the network manager. Future work include testing using more network elements, and investigating the online generation and enactment of the most cost effective strategy, as determined by the model checker. We also plan to verify additional properties, especially with respect to the involved parameters for both the camera and the network manager adaptation. Finally, introducing more complex network topology will further increase the applicability of the proposed technique.

References

[1] 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.

[2] 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.

[3] J. Heo and T. Abdelzaher. Adaptguard: Guarding adaptive systems from instability. In *6th ACM International Conference on Autonomic Computing*, 2009.

[4] Gautham Nayak Seetanadi, Luis Oliveira, Luis Almeida, Karl-Erik Arzen, and Martina Maggio. Game-theoretic network bandwidth distribution for self-adaptive cameras. In *15th International Workshop on Real-Time Networks*, 2017.

[5] G.C. Chasparis, M. Maggio, E. Bini, and K.-E. Årzén. Design and implementation of distributed resource management for time-sensitive applications. *Automatica*, 64, 2016.

[6] 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.

[7] 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.

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

[9] W.P.M.H. Heemels, K.H. Johansson, and P. Tabuada. An introduction to event-triggered and self-triggered control. In *IEEE Conference on Decision and Control*, 2012.

[10] X. Wang and M.D. Lemmon. Event-triggering in distributed networked control systems. *IEEE Transactions on Automatic Control*, 56(3), 2011.

[11] M.Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, 2007.

[12] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *1st International Workshop Formal Modeling and Analysis of Timed Systems*, 2003.

[13] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, 1982.

[14] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *International Symposium on Protocol Specification, Testing and Verification*, 1996.

[15] M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *23rd International Conference on Computer Aided Verification*, 2011.

[16] R. Marau, L. Almeida, and P. Pedreiras. Enhancing real-time communication over cots Ethernet switches. In *International Workshop on Factory Communication Systems*, 2006.

[17] B. Vandalore, W. Feng, R. Jain, and S. Fahmy. A survey of application layer techniques for adaptive streaming of multimedia. *Real-Time Imaging*, 7(3), 2001.

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

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

[20] Laszlo Toka, András Lajtha, Éva Hosszu, Bence Formanek, Dániel Géhberger, and János Tapolcai. A Resource-Aware and Time-Critical IoT framework. In *IEEE International Conference on Computer Communications INFOCOM*, May 2017.

[21] Tan Zhang, Aakanksha Chowdhery, Paramvir (Victor) Bahl, Kyle Jamieson, and Suman 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. ACM.

[22] V. Veeraraghavan and S. Weber. Fundamental tradeoffs in distributed algorithms for rate adaptive multimedia streams. *Comput. Netw.*, 52(6), 2008.

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

[24] R. Razavi, M. Fleury, and M. Ghanbari. Low-delay video control in a personal area network for augmented reality. *IET Image Processing*, 2(3), 2008.

[25] 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.

[26] A. Kumar. Computer-vision-based fabric defect detection: A survey. *IEEE Transactions on Industrial Electronics*, 55(1), 2008.

[27] D.A. de Lima and A.C. Victorino. A hybrid controller for vision-based navigation of autonomous vehicles in urban environments. *IEEE Transactions on Intelligent Transportation Systems*, 17(8), 2016.

[28] 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.

[29] T.T.B. 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.

[30] G.A. Moreno, J. Cámara, D. Garlan, and B.R. Schmerl. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In *10th ACM/SIGSOFT Joint Meeting on Foundations of Software Engineering*, 2015.

APPENDIX A
A TIMELINE EXAMPLE

This appendix introduces an example of the system timeline, to familiarize with the terminology and illustrate the different quantities involved in the system's behavior.

Figure 5 shows these quantities graphically. The period of the network manager $\pi_{\mathcal{M}}$ is equal to three times the period of the allocation, $\pi_{\mathcal{M}} = 3\pi_{\text{alloc}} = 90ms$. The index t counts the network manager interventions, while the index w counts the frame transmitted. At time $0ms$, which corresponds to $t = 0$, the network manager decides the initial fraction of bandwidth to be assigned to both cameras ($b_{1,0}$ and $b_{2,0}$). This initial assignment determines the value of the actual amount of bandwidth that each frame is allowed to consume in each camera until the next manager intervention ($B_{1,1} - B_{1,3}$ for camera c_1, and $B_{2,1} - B_{2,3}$ for camera c_2). At time $90ms$ ($t = 1$), the network manager chooses a different allocation, affecting the next three frames for the cameras. For each frame, the cameras determine a quality, that in turn affects the frame size.

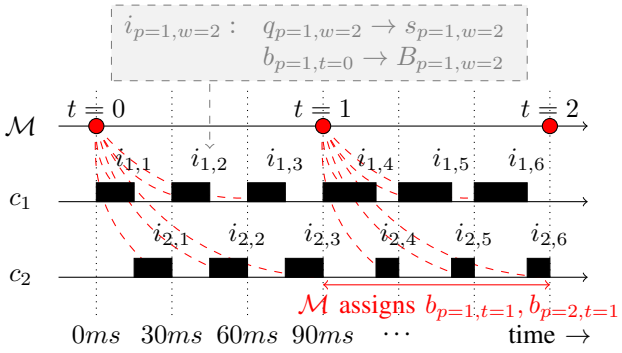

Fig. 5. Example of system timeline.

The gray box in the Figure shows relevant dependencies for $i_{1,2}$, the second image transmitted by the first camera. The quality $q_{1,2}$ determines the frame size $s_{1,2}$. The bandwidth allocation computed in the first network manager intervention at $t = 0$, $b_{p=1,t=0}$, determines the size of the channel that the frame is allowed to use $B_{p=1,w=2}$. The following quality $q_{1,3}$ will then be computed using the difference between the network bandwidth allocated to the frame $B_{1,2}$ and the size of the encoded frame $s_{1,2}$.

APPENDIX B
MODEL OF CAMERA AND NETWORK MANAGER BEHAVIOR

Listing 2 shows the description of the behavior of an arbitrary camera in the network – in this case c_1. The first part of the code (lines 1-7) introduces terms and constants that are then used for the camera state update: the proportional and integral gain $k_{1,\text{Proportional}}$ and $k_{1,\text{Integral}}$, the minimum and maximum quality $q_{\min}$ and $q_{\max}$, and the minimum and maximum frame size $s_{1,\min}$ and $s_{1,\max}$. The second part of the code (lines 9-11) introduces the expressions that should be computed for the model checking and for the camera controller: the calculation of the frame size according to Equation (2), the update of the quality in the controller according to Equation (4), and the calculation of the normalized error, or matching function, as the term $e_{1,w}$ in Equation (3).

```
1  // ******************** CAMERA PARAMETERS
2  const double k1pro = 10.0; // proportional gain
3  const double k1int = 5.0; // integral gain
4  const int minimum_quality = 15; // minimum quality
5  const int maximum_quality = 85; // maximum quality
6  const int minimum_framesize = 64; // minimum of X bytes
7  const int maximum_framesize = 100000; // maximum of X bytes
8  // ********************* CAMERA FORMULAS
9  formula framesize1 = ... // compute frame size according to equation (2)
10 formula update_q1 = ... // control action according to equation (3) and (4)
11 formula f1 = ... // matching function e_{1,w} according to equation (3)
12 // ********************* CAMERA MODULE
13 module c1
14   q1: [minimum_quality..maximum_quality] init maximum_quality;
15   s1: [minimum_framesize..maximum_framesize] init maximum_framesize;
16   tran1: int init 0; // number of frames transmitted by camera 1
17   drop1: int init 0; // number of frames dropped by camera 1
18   [] (turn = cam1) -> (turn' = cam2) // next in round
19       & (q1' = update_q1) // update quality with controller
20       & (s1' = framesize1) // compute framesize
21       & (c1change' = (q1 = update_q1 ? false : true)) // check if change
22       & (tran1' = compute_tn1 <= t1 ? tran1+1: tran1) // if transmitted
23       & (drop1' = compute_tn1 > t1 ? drop1+1: drop1) // if dropped
24       & (want_nm' = f1 > threshold_event | f1 < -threshold_event ? true :
             want_nm); // check threshold
25 endmodule
```

Listing 2. Camera module description

Finally, the last part (lines 13-25) contains the camera module, which captures the logic of the state update for the camera. The camera can only perform one action to update its state variables during its turn (lines 18-24). This update includes yielding the turn to the next camera in the list (line 18), or to the scheduler (in the last camera case). The action also updates the internal value for the current quality parameter $q_{1,w}$, determines the frame size based on the old quality value, computes a boolean value that assess if there was a change in quality (used for property verification, cf. Section V-C), and determines if the frame was transmitted or dropped. Finally, for the event-triggered network manager version, the action determines if the camera triggers an intervention of the network manager based on the value of the threshold (line 24).

```
1  formula update_bw1 = ... // update bw according to equation (7)
2  formula update_bw2 = ... // update bw according to equation (7)
3  module nm
4    bw1: [min_bw..max_bw] init floor(max_bw / num_cameras);
5    bw2: [min_bw..max_bw] init ceil(max_bw / num_cameras);
6    nm_interventions: int init 0;
7    [] (turn = nmng) -> (want_nm' = false) & // reset because done
8      (bw1' = update_bw1) & // update camera 1
9      (bw2' = update_bw2) & // update camera 2
10     (nmchange' = (bw1 = update_bw1 ? false : true)) &
11     (nm_interventions' = nm_interventions+1) &
12     (turn' = sche); // go back to the scheduler
13 endmodule
```

Listing 3. Network manager module description

The code for the network manager (Listing 3) is similar, in structure, to the code for a camera. When invoked, the manager performs a single action updating the bandwidth distribution (lines 8-9), it determines if there was a change updating a boolean value (line 10), it updates a counter that keeps track of the number of performed interventions (line 11), and finally it passes the turn to the scheduler (line 12).

APPENDIX C
OVERHEAD EVALUATION

In our experimental evaluation, we have considered a system composed of two cameras. However, in a classical IoT setup, there are a multitude of devices sharing the network. As recalled in Sections II and III, one of the advantages of using the allocation policy proposed in [8] is its linear time complexity with respect to the number of cameras.

To collect the overhead data, we have randomized the values of $\lambda_{\{1..m\}}$ where m is the number of cameras for the experiment. We have then measured the overhead of invoking the function that: (i) computes the network bandwidth distribution according to Equation (7), (ii) saturates the computed values using a minimum and maximum threshold, (iii) saves the old values for the following iteration. We measured the time to invoke the function 10^5 times and computed its average to obtain a reasonable estimate of the overhead. Figure 6 shows the average overhead in nanoseconds, when the number of cameras varies from 1 to 100. The complexity increases linearly with the number of cameras, which makes this algorithm practical for modern networks.

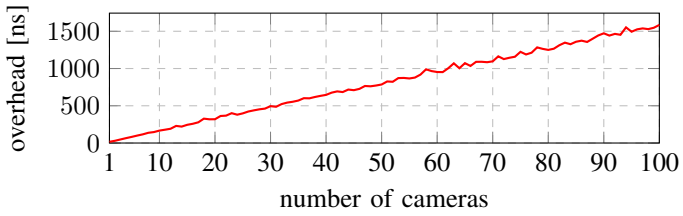

Fig. 6. Average overhead measured for the computation of the network allocation.

APPENDIX D
ADDITIONAL RESULTS

This appendix presents additional experiments that assess the performance of the complete system. This set of experiments was conducted with a system that includes three cameras. We run the system with the time-triggered solution described in Section II, and with the event-triggered network manager described in Section IV, each of them using different parameters. For the time-triggered network manager, the only solution parameter is the manager period $\pi_{\mathcal{M}}$. In our experiments the period belongs to the set $\{600ms, 3000ms, 6000ms\}$. For the event-triggered solution, we experimented with the triggering threshold τ_{thr} belonging to the set $\{0.1, 0.2, 0.3\}$. As explained in Section V the optimal triggering policy, despite its optimality in terms of cost minimization, is not a viable solution for long experiments, because the environment dynamically changes and the network manager should be aware of these changes to trigger the "reaction-to-changes protocol". We have therefore excluded the PRISM strategy from this set of experiments.

Table I reports the results of this set of experiments. Each row represents a one-hour long run of the system using a specific strategy. The first three rows show the time-triggered (TT) solutions, while remaining rows represent the event-triggered (ET) solutions. The parameters chosen for the solutions follow in parenthesis. The columns represent the percentage of frames that are correctly transmitted for the three cameras, c_x Tx% for camera x, the number of interventions of the network manager nm_{int} and the total cost computed as defined in Section V-D. For correct operation, frame dropping must occur, as the network bandwidth is not enough for all the cameras. This is because we plan to stress the system and test it in extreme conditions. The event-triggered version of the network manager achieves lower running costs and higher percentages of transmitted frames for the cameras, with a very small number of interventions in the system. In the event-triggered category, a threshold $\tau_{\text{thr}} = 0.2$ minimizes the total cost for running the system, achieving the highest percentage of transmitted frames with a low number of interventions. The resource manager intervenes only 5 times. This is a negligible overhead, especially compared to the time-triggered policy that achieves the best cost, which has period $\pi_{\mathcal{M}} = 3000ms$ and intervenes 958 times. The higher number of interventions does not result in a higher number of transmitted frames.

TABLE I
ADDITIONAL EXPERIMENT: 3-CAMERAS SYSTEM

strategy	c_1 Tx%	c_2 Tx%	c_3 Tx%	nm_{int}	total cost
TT ($\pi_{\mathcal{M}} = 600ms$)	37.66	37.04	40.46	4460	1069140
TT ($\pi_{\mathcal{M}} = 3000ms$)	45.18	43.14	45.83	958	956198
TT ($\pi_{\mathcal{M}} = 6000ms$)	35.99	37.83	36.63	435	1092125
ET ($\tau_{\text{thr}} = 0.1$)	48.08	47.92	56.59	7	848977
ET ($\tau_{\text{thr}} = 0.2$)	53.88	49.26	59.75	5	789735
ET ($\tau_{\text{thr}} = 0.3$)	43.46	45.48	51.25	3	920403

To confirm the validity of these results, we executed each experiment 10 times, and computed the total cost for each execution. We display the cost using box plots, in Figure 7. The event-triggered strategy has lower median values in terms of cost, although a couple of outliers are shown with $\tau_{\text{thr}} = 0.3$. This implies that $\tau_{\text{thr}} = 0.2$ is a more flexible value, as also discussed for the single run results shown in Table I. The time-triggered solutions have higher median cost, and higher maximum values. At the same time, the time-triggered solution may achieve lower cost (see for example $\pi_{\mathcal{M}} = 6000ms$) but seems to be less predictable (the range is wider). The experiments highlight that using an event-triggered solution increased the overall predictability of the system.

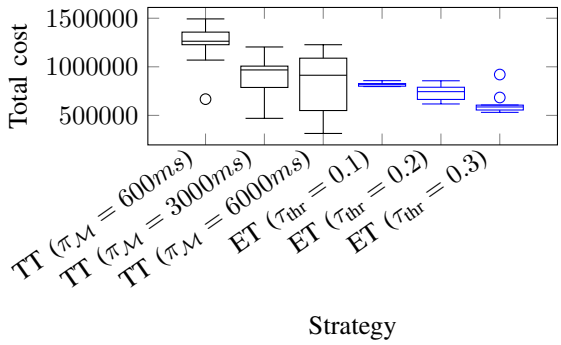

Fig. 7. Total cost box plot for 10 1-hour long executions traces.