Improving Performance of Feedback-Based Real-Time Networks using Model Checking and Reinforcement Learning

Nayak Seetanadi, Gautham

2021

Document Version:
Publisher's PDF, also known as Version of record

Improving Performance of Feedback-Based Real-Time Networks using Model Checking and Reinforcement Learning

Gautham Nayak Seetanadi

LUND
UNIVERSITY

Department of Automatic Control

PhD Thesis TFRT-1129
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2021

To my parents

Abstract

Traditionally, automatic control techniques arose due to need for automation in mechanical systems. These techniques rely on robust mathematical modelling of physical systems with the goal to drive their behaviour to desired set-points. Decades of research have successfully automated, optimized, and ensured safety of a wide variety of mechanical systems.

Recent advancement in digital technology has made computers pervasive into every facet of life. As such, there have been many recent attempts to incorporate control techniques into digital technology. This thesis investigates the intersection and co-application of control theory and computer science to evaluate and improve performance of time-critical systems. The thesis applies two different research areas, namely, model checking and reinforcement learning to design and evaluate two unique real-time networks in conjunction with control technologies. The first is a camera surveillance system with the goal of constrained resource allocation to self-adaptive cameras. The second is a dual-delay real-time communication network with the goal of safe packet routing with minimal delays.

The camera surveillance system consists of self-adaptive cameras and a centralized manager, in which the cameras capture a stream of images and transmit them to a central manager over a shared constrained communication channel. The event-based manager allocates fractions of the shared bandwidth to all cameras in the network. The thesis provides guarantees on the behaviour of the camera surveillance network through model checking. Disturbances that arise during image capture due to variations in capture scenes are modelled using probabilistic and non-deterministic Markov Decision Processes (MDPs). The different properties of the camera network such as the number of frame drops and bandwidth reallocations are evaluated through formal verification.

The second part of the thesis explores packet routing for real-time networks constructed with nodes and directed edges. Each edge in the network consists of two different delays, a worst-case delay that captures high load characteristics, and a typical delay that captures the current network load. Each node in the network takes safe routing decisions by considering delays already encountered and the amount of remaining time.

The thesis applies reinforcement learning to route packets through the network with minimal delays while ensuring the total path delay from source to destination does not exceed the pre-determined deadline of the packet. The reinforcement learning algorithm explores new edges to find optimal routing paths while ensuring safety through a simple pre-processing algorithm. The thesis shows that it is possible to apply powerful reinforcement learning techniques to time-critical systems with expert knowledge about the system.

Acknowledgements

This thesis would not have been possible without the amazing support and supervision of my supervisor Martina Maggio. She has been the pillar of support throughout my PhD right from when I had my interview 5 years ago, believing in me when I have doubted myself. I will be forever grateful for her insight and guidance both with life in the department, and survival out in the wild world of Academia. Also I would like to thank my co-supervisor Karl-Erik Årzen who gave me the opportunity to interview and catch a glimpse of life at the department. He is always ready with great guidance regarding everything from the most appropriate conference for the paper, to the best bar in any given city.

The department provides a great environment that is always friendly, welcoming and intellectually challenging. The regular fikas provide a great break from work to rant about reviewers, fix Ladok problems and even discuss the weather. A special thanks to the whole administration group that always went out of their way to help with problems. Specifically Mika, Ingrid, and Cecilia, thank you for your help with ladok, primula and always finding ways to fix administration issues. Thanks to Anders Nilsson, Anders Blomdell and Pontus Andersson for timely help with all computer and equipment related issues. A special thanks to Eva (secret boss) for always listening to my rants, providing pep talks that never failed and having an unlimited supply of candy helpful during thesis writing crunch time. Life at the department has provided the familial feeling required to get through the dark winters of Sweden.

A thanks to the amazing friends I have made on this journey. The best mentor, Fredrik Bagge Carlson, thank you for showing me your wise ways in life. I have been mentored in a wide variety of life skills although I don't believe that was the aim of the mentorship program at the department. Richard pates, thank you for always pushing me to do better and helping a vegetarian survive in China. Mattias Fält, thank you for giving me a crash course about life in Sweden, making life competitive, and providing me a roof over my head. I will always be thankful for the adventures we four undertook, be it the Tura or cheap Wizz Air flights. Alex, thank you for helping me discuss new research topics and the crash course on tea variety. Thank you Marcus Thelander Andrén and Tommi Berner for great discussion of ideas during workshops and conferences.

Martin Heyden, thank you for sharing the disc golf madness and much needed golf breaks during thesis writing. A special thanks to Kaoru Yamamoto and Michelle Chong for always checking in on me and buoying my spirits during difficult times. Thank you Kaoru for also proof reading the thesis. Adriana Sanna, thank you for pushing my comfort zone to help me learn more about myself. Leila Jabrane, thank you for the amazing culinary adventures and always reminding me that everything will be good. Radhika, thank you for proof reading the thesis and offering suggestions for improvement from a different perspective.

I would also like to thank my office mates through this journey, Fredrik, Mattias, Frida Heskebeck, Emil Vladu, Sebastian Banert, and a host of visiting researchers including Li Zhu, Amani Jaafer, José Manuel Gonzáles Cava that always made discussions in the office interesting.

I am lucky to have large support from my family and friends all the way from India. Adi and Sachin, thank you for the zoom talks and always being there to help me believe in myself.

My family has been a big pillar of support for the whole duration of my thesis. Thank you Ajja and Anama for showing unparalleled love and affection. Thank you to my biggest believer Kolar Ajja, always worried about prices in Sweden. Mamama, tugele appara mogu ani super khaan javanaka konai match koru jaaina.

Ian Uncle, Thank you for always treating me as your own son, looking out for me and for the supportive messages when most required.

Thank you to my brother Gaurav for being a big link to family back home and keeping me grounded, the way only a brother can. Finally, thank you to my parents whose love, affection, support and guidance have made me who I am today.

Financial Support

Financial support for work in this thesis was provided through the Swedish Research Council (VR) project titled "Feedback Computing in Cyber-Physical Systems" and ELLIIT LU project titled "Co-Design of Robust and Secure Networked Embedded Control Systems".

Contents

Nomenclature

Camera Surveillance Network

Notation	Description
$\mathcal{C} = \{c_1, ..., c_n\}$	The set of n cameras in the network
$\mathcal{M}$	The network manager
$\mathcal{H}$	The limited global available network
$q_{p,w}$	Encoding quality of frame w of camera p
$b_{p,t}$	Fraction of bandwidth allocated to camera p at time t
$B_{p,w}$	Amount of bandwidth allocated to frame w of camera p
$s_{p,w}$	Frame size of frame w of camera p
$s^*_{p,w}$	Estimated frame size of frame w of camera p
$s_{p,max}$	The maximum possible generated frame size of camera p
$\delta s_{p,w}$	Disturbance in frame size of frame w of camera p
π_{alloc}	The allocation period for cameras to transmit images
$e_{p,w}$	Error between the allocated bandwidth and current frame size
k_p	Proportional gain of the camera controller
k_i	Integral gain of the camera controller
$f_{p,t}$	Match between the allocated bandwidth and the frame size
$\lambda_{p,t}$	Emphasis of bandwidth reallocation
τ_{thr}	Triggering threshold of the event-based manager

Real-Time Routing Network

Notation	Description
$e : (x \rightarrow y)$	Directed edge from node x to node y
c^W_{xy}	Worst case transmission time over edge $e : (x \rightarrow y)$
c^T_{xy}	Typical transmission time over edge $e : (x \rightarrow y)$
δ_{it}	Actual transmission time from source i to destination t
c_{xyt}	Minimum guaranteed delay from node x to destination t over edge $e : (x \rightarrow y)$
c_{x_t}	Minimum guaranteed worst-case delay from node x to destination t over all outgoing edges

Abbreviation	**Description**
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
MDP	Markov Decision Process
FSM	Finite State Machine
CTL	Computational Tree Logic
PCTL	Probabilistic Computational Tree Logic
DP	Dynamic Programming
TD	Temporal Difference
RL	Reinforcement Learning

1

Introduction

"You can't ever reach perfection, but you can believe in an asymptote toward which you are ceaselessly striving. "

— Paul Kalanithi, *When Breath Becomes Air*

Control engineering works on the basic premise of sensing a system and actuating it to a desired state. Although its origin is found in physical systems, control engineering concepts have a wide field of application with objectives that range from automation to rejection of disturbances. This thesis applies these control concepts and techniques to computer systems, namely real-time networks. Real-time networks require robust guarantees on their safety with optimality in their operation. Some relevant objectives in real-time networks are robust guarantees on end-to-end delays for time-critical systems, performance evaluation of fair resource allocation schemes in resource constrained systems, and system performance in the presence of an adversary.

Advancements in computing have resulted in a large number of edge systems that share a constrained resource. Edge systems in these embedded networks are generally mobile and capable of regulating their resource utilization. In such a system the goal is to distribute resources to all edge systems while ensuring fairness and no resource wastage. This thesis considers a network of surveillance cameras that are capable of regulating their bandwidth through image encoding. An event-triggered manager distributes bandwidth to all the cameras in the network in a fair and efficient manner. It is not trivial to provide guarantees on the behaviour of the event-based resource allocation scheme that reacts only when required. The safety requirements of such systems can be provided through application of formal verification and model checking. New research in model checking tools provides new methods for translating computer science requirements into formal descriptions that are suitable for formal verification.

Model checking of complex computer systems leads to a large state space leading to memory issues during property verification. State space explosion problem during system verification is well known in model checking and has been widely studied [Clarke, Klieber, Nováček, and Zuliani, 2012]. There have been recent attempts in mitigating the problem of state space explosion. These methods either reduce the footprint of the state space by discarding unnecessary states [Baier, Clarke, Hartonas-Garmhausen, Kwiatkowska, and Ryan, 1997] or

relaxing guarantees on system performance through statistical model checking (SMC) [Legay, Delahaye, and Bensalem, 2010]. SMC simulates the system for a finite number of executions and provides statistical bounds on safety guarantees during verification of system properties. Although this is an relaxed verification bound, statistical model checking handles much larger state-spaces compared to classical model checking as it does not store the whole state space in memory. The later part of the thesis obtains robust system performance bounds with smaller state spaces through usage of reinforcement learning.

Reinforcement Learning (RL) is an alternate approach of guaranteeing system safety while only exploring necessary states of the system. Systems for RL are modelled using Markov Decision Processes (MDPs) similar to model checking and build the state space of the system through exploration. The thesis constructs real-time networks using a dual-delay model introduced recently in [Baruah, 2018] and introduces safe RL algorithms for routing packets without violating pre-determined packet deadlines. RL algorithms optimize system performance by exploiting information from previously explored states, and take new actions that explore previously unknown states to obtain new information about the system. Exploration of new states is however a risky action, and is opposite to the concept of safety required for real-time networks. RL exploration can be however made safe through expert knowledge of the system that does not take unsafe actions. This safe reinforcement learning leads to the generation of smaller, more efficient state spaces and allows application of powerful RL algorithms to time critical networks.

Contributions of the Thesis

The thesis is based on the following *six* publications. Papers I, II, III and IV explore application of control concepts together with model checking to real-time camera networks. Papers V and VI explore application of safe reinforcement learning for packet routing in real-time communication networks.

Paper I

> G. Nayak Seetanadi, L. Oliveira, L. Almeida, K.-E. Årzén, and M. Maggio (2018). "**Game-Theoretic Network Bandwidth Distribution for Self-Adaptive Cameras**". In: *SIGBED Review. Association for Computing Machinery*

Real-time camera networks consisting of a large number of cameras compete for bandwidth which is constrained. The cameras record a stream of images and transmit them to a central manager over a shared channel. The central manager regulates the share of the bandwidth allocated to each camera. In this paper we evaluate the need for control at each camera ensuring adherence to the allocated bandwidth. This bandwidth regulation is performed regulating the quality (compression) of each frame using a PI controller. The central manager allocates

bandwidth to each camera in the network using a game-theoretic bandwidth allocation scheme.

The main idea for the paper was obtained through previous research on game-theoretic application manager done by M. Maggio and K.-E. Årzén. The use of PI control for frame rate control arose from discussions with all the co-authors. L. Oliveira and L. Almeida were involved with initial experiments conducted at FEUP Porto, Portugal and contributed to the sections of the paper that concern image capture. G. Nayak Seetanadi performed the experiments and recorded data for bandwidth allocation schemes. The manuscript was written by G. Nayak Seetanadi with input and comments from M. Maggio and K.-E. Årzén.

Paper II

> G. Nayak Seetanadi, J. Camara, L. Almeida, K.-E. Årzén, and M. Maggio (2017). "**Event-Driven Bandwidth Allocation with Formal Guarantees for Camera Networks**". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*

In this paper we build on work from Paper I by providing guarantees on the behaviour and safety of the real-time camera network. We also implement and evaluate a event-triggered manager compared to the periodic manager from Paper I. The different entities in the real-time network, cameras and the manager are modelled using Markov Decision Processes (MDP) and properties are verified using the PRISM model checker [Kwiatkowska, Norman, and Parker, 2011]. We discuss the choice of the triggering threshold τ_{thr} and its effect on system performance.

The event-triggered manager was discussed as a natural progression from the previous publication by G. Nayak Seetanadi and his supervisors. J. Camara brought expertise with model checking and building of models using MDP, used for verification of the camera network. L. Almeida helped with image encoding on the real camera test-bed. G. Nayak Seetanadi and M. Maggio performed verification with PRISM model checker. G. Nayak Seetanadi performed experiments on the camera test-bed. The manuscript was written by G. Nayak Seetanadi with input from M. Maggio and comments from the other co-authors.

Paper III

> G. Nayak Seetanadi, K.-E. Årzén, and M. Maggio (2020). "**Control-Based Event-Driven Bandwidth Allocation Scheme for Video-Surveillance Systems**". In: *Taylor & Francis Cyber-Physical Systems*. Under Review

In this paper we extend the camera models from Paper II to incorporate disturbances that are inherent in image capture and encoding. We construct two models, one without disturbances and another with disturbances as probabilities. The amount of frames dropped is evaluated for different values of controller

gain for the different models. We also compare classical model checking which provides definite guarantees with statistical model checking that provides statistical guarantees. On the experimental side, this paper incorporates a larger number of cameras and triggering thresholds, τ_{thr}.

The idea for the paper was discussed as an extension to the preceding paper by all the co-authors. G. Nayak Seetanadi and M. Maggio performed verification of the camera models with PRISM model checker. G. Nayak Seetanadi performed experiments on the camera test-bed. The manuscript was written by G. Nayak Seetanadi with inputs an comments from the co-authors.

Paper IV

> G. Nayak Seetanadi, K.-E. Årzén, and M. Maggio (2019). "**Model Checking a Self-Adaptive Camera Network with Physical Disturbances**". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*

In this paper we evaluate properties of the camera network such as number of frames sent, frames dropped and manager interventions using three different models. The deterministic and probabilistic models from Paper III are compared with a non-deterministic model. The non-deterministic camera model uses stochastic MDPs to inject disturbances into the frame sizes. In this paper we also evaluate the performance of the system when the cameras and the manager compete or collaborate to fulfill their unique objectives using PRISM games.

The paper expands the complexity of camera models and the idea was devised entirely by G. Nayak Seetanadi. G. Nayak Seetanadi and M. Maggio developed the three camera models used for verification. G. Nayak Seetanadi performed model checking using PRISM for all the camera models. The manuscript was written by G. Nayak Seetanadi with inputs and comments from all the co-authors.

Paper V

> G. Nayak Seetanadi, K.-E. Årzén, and M. Maggio (2020). "**Adaptive Routing with Guaranteed Delay Bounds Using Safe Reinforcement Learning**". In: *Proceedings of the 28th International Conference on Real-Time Networks and Systems (RTNS)*

Model checking has the drawback of requiring large state spaces to model real systems with Markov Decision Processes (MDPs). This leads to relaxation of model complexity or reducing the stringent requirements for system verification. In this paper we choose to explore the MDP using Reinforcement Learning (RL) instead. Using RL allows the system to build the model of a system dynamically using exploration. Random exploration can lead to problem in time sensitive systems. We apply RL to safely transmit packets through a real-time network

ensuring that no packets violate their deadline. We use a pre-processing algorithm to evaluate worst-case transmission times. RL then chooses a path with minimum transmission times depending on the current network load. We ensure the algorithm performs safe exploration using information obtained in the pre-processing stage.

The idea for the paper was brought forward by K.-E. Årzén and M. Maggio after hearing a presentation at RTSS 2018. RL algorithms use MDPs similar to model checking used in the previous papers. RL algorithms however build state-space through exploration and thus do not require large memory. The RL algorithm and the routing model were designed and implemented by G. Nayak Seetanadi. The manuscript was written by G. Nayak Seetanadi with inputs and comments from all the co-authors.

Paper VI

> G. Nayak Seetanadi (2020). "**Adaptive Routing for Real-Time Networks with Dynamic Deadlines using Safe Reinforcement Learning**". In: *ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI)*. (Under Review)

We build on the work done in Paper V by providing safety guarantees on the behaviour of our algorithm. This paper provides algorithms for safe addition and removal of nodes in the network, essential for a large-scale communication network. We also expand the state space to ensure its adaptability to varying deadline changes from each packet to the next. The paper evaluates the algorithm against state of the art algorithms for packet transmission in real-time networks and provides a discussion on their appropriate usage depending on the network configuration.

The paper was a natural progression as reviews for previous work noted the inability of the algorithms to adapt to network changes. G. Nayak Seetanadi came up with the idea to extend the algorithm for addition and deletion of nodes from the network, as well as the discussion on the performance of the various real-time routing algorithms in different network conditions. The manuscript was written by G. Nayak Seetanadi with comments from his supervisors.

2

Camera Surveillance Network

This chapter provides background on the camera surveillance network explored in Papers I, II, III and IV. The chapter describes the modelling of the camera network and image encoding with disturbances that arise during image capture. It gives an introduction to model checking and its application to the camera network for property verification and safety guarantees.

2.1 Camera Network Model

The camera network in the thesis models a surveillance system consisting of n cameras, $\mathcal{C} = \{c_1, ..., c_n\}$, and a central manager, $\mathcal{M}$. The cameras capture a stream of images and transmit them to the central manager over a shared channel with limited network bandwidth $\mathcal{H}$. The cameras are self adaptive in nature and regulate their bandwidth usage by capturing images and encoding each image with a quality factor. The manager $\mathcal{M}$ allocates a fraction of the total available bandwidth to each camera in the network ensuring the allocation is fair. The manager also ensures that the total bandwidth allocated to the cameras is less than or equal to the total amount of available bandwidth $\mathcal{H}$. The amount of bandwidth allocated has to closely match the requirements of the cameras as images whose frame sizes exceed the allocated bandwidth are dropped and lead to loss of information.

Cameras

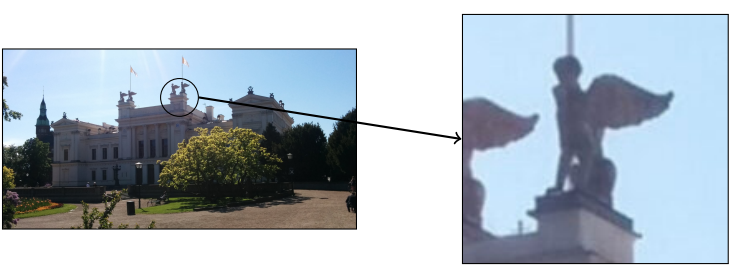
The camera $c_{p\in\{1,...,n\}}$ captures a frame w and encodes it using a quality factor $q_{p,w}$. The quality factor regulates the size of the resulting image changing the camera bandwidth usage during each frame transmission.

Camera Encoding Images captured by the cameras are generally large in size and thus unsuitable for direct transmission over the share communication channel. The images contain redundant information which leads to inefficient utilization of the constrained bandwidth. Encoding an image with appropriate tech-

(a) Original Image (Size 798 kB)

(b) Image Encoded with Quality $q = 10$ (Size 1.1 kB)

(c) Image Encoded with Quality $q = 50$ (Size 260 kB)



(d) Image Encoded with Quality $q = 85$ (Size 498 kB)

Figure 2.1 Effect of Encoding on Image Size and Perceived Quality

niques reduces its size while retaining most information. The choice of the encoding algorithm affects the resulting frame size and determines the amount of bandwidth utilization by the camera. Figure 2.1 shows the loss of information that occurs during encoding of images using different qualities and resulting frame sizes. As seen in the figure, encoding an image with a small quality factor leads to smaller image sizes and artifacts in the encoded image. Using a higher quality leads to better preservation of images at the cost of larger frame sizes.

This camera surveillance network with adaptive camera and the manager consists of two independent control loops. The control loops regulate the constrained bandwidth to ensure a good match between the camera frame size and bandwidth allocated to the camera.

- **Image Fitting at Camera** $c_{p\in\{1,\ldots,n\}}$: The camera p regulates the quality factor, $q_{p,w}$, of each frame w to ensure it fits the amount of bandwidth allocated by the manager, $B_{p,w}$.
- **Bandwidth allocation by Manager** $\mathcal{M}$: The manager regulates the amount of bandwidth allocated, $B_{p,w}$, to each camera in the network using a game-theoretic approach considering the amount of available bandwidth available globally and the current camera requirements.

Figure 2.2 shows a camera network consisting of three cameras connected to a central manager. An example timeline of bandwidth distribution is also shown on the left. The black bar in the timeline indicates the start of a new transmission slot at multiples of the allocation period π_{alloc}. The manager $\mathcal{M}$ in the figure is triggered periodically and recalculates the amount of bandwidth allocated to all cameras in the network during each allocation period. Only cameras c_1 and c_2 are active during the first two allocation periods. $\mathcal{M}$ allocates bandwidth equally to both cameras in the first allocation period as it has no information about the camera requirements. The manager allocates more bandwidth to camera c_2 from the second period onward as the camera is capturing a dynamic scene with larger frame sizes, thus it requires bigger portion of the available bandwidth compared to camera c_1. Camera c_3 joins the network in the beginning of the third allocation period. The bandwidth is initially distributed equally to all three cameras in the network during the third allocation period and allocates bandwidth to the cameras according to their relative bandwidth requirements during the subsequent periods.

The quality factor and the encoding technique used for encoding the image determines the resulting frame size of the image. Motion JPEG and H.264 (and recently H.265) are among the most widely used standards for video compression. In this thesis, we use Motion JPEG (MJPEG) to encode images captured by the cameras. MJPEG is an intra-frame encoding format where the video is simply a sequence of individual JPEG images with no inter-frame compression. Inter-frame compression used in H.264 and H.265, generates videos with smaller bandwidth consumption compared to intra-frame compression, but a loss of frame leads to large amount of useless frames due to inter-frame dependence.

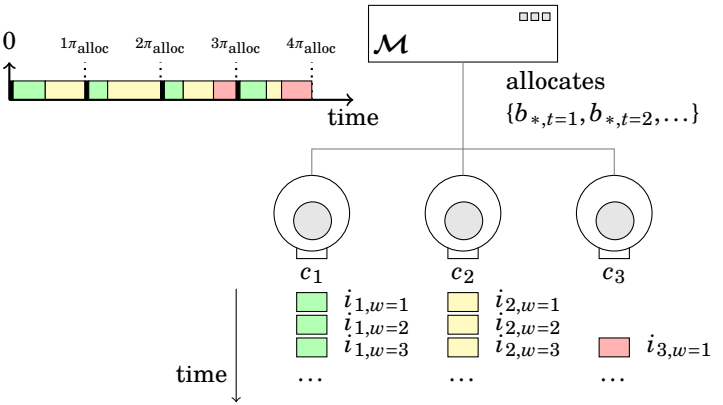


Figure 2.2 Camera network system architecture [Nayak Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017]

The camera network drops frames when the camera frame size is larger than the allocated bandwidth. Thus we choose MJPEG as the encoding technique due to its simplicity and inter-frame independence.

The exact relationship between encoding quality and the resulting frame size is non-trivial due to disturbances like motion, sudden scene changes that arise during image capture [Edpalm, Martins, Maggio, and Årzén, 2018]. Previous research in the area of fitting an image to a pre-determined size is performed with the use of complex models for specific encoding techniques. [Ding and Liu, 1996] describes a rate-quantization model to estimate frame-size s^* when using MJPEG encoding. The estimated frame size $s^*_{p,w}$ for camera p and frame number w was determined as follows.

$$s^*_{p,w} = \alpha + \frac{\beta}{q^{\lambda}_{p,w}} \tag{2.1}$$

where α and β are internal parameters of the camera and λ regulates the quantization curves for different frames. [Silvestre, Almeida, Marau, and Pedreiras, 2007] uses the rate-quantization model to estimate the MJPEG frame-size s^* and fits the generated frames to bandwidth allocated by the manager $B_{p,w}$ for each frame. Recent encoding standards have complex inter-frame encoding and dependencies. [Edpalm, Martins, Maggio, and Årzén, 2018] [Edpalm, Martins, Årzén, and Maggio, 2018] shows that estimation of instantaneous bandwidth utilization for H.264 video compression is non-trivial. These models lead to somewhat accurate prediction of the resulting $s^*_{p,w}$ with the drawback of large computational overhead.

In this thesis we do not use complex non-linear models to obtain the exact frame size estimations. We rely on simple linear models to get approximate estimation of the frame size, and then regulate bandwidth utilization of the cameras

in combination with PI controllers. The PI controller adapts to changes in the scene being recorded and treats them as disturbances.

$$s^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,max} \tag{2.2}$$

Equation (2.2) shows the generation of an image with size $s^*_{p,w}$ using quality $q_{p,w}$. The encoding quality $q_{p,w}$ is generally bounded and its value is dependent on the encoding technique that is applied. The MJPEG $q_{p,w} \in (0, 100]$. $s_{p,max}$ is the maximum size of the frame generated and it is dependent on a number of factors such as the number of pixels, movement, noise and so on, see [Edpalm, Martins, Maggio, and Årzén, 2018] for an analysis on the relationship between $q_{p,w}$ and $s_{p,max}$. The authors show that the relationship between encoding quality and generated frame size is non-trivial and is susceptible to disturbances that arise due to changes in the scenes. These disturbances cause the generated images to have different frame sizes when encoded with the same quality factor. This uncertainty is captured in Equation (2.3)

$$s^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,max} + \delta s_{p,w} \tag{2.3}$$

The PI controller minimizes the normalized error $e_{p,w}$ between the bandwidth allocated by the manager $B_{p,w-1}$ and the current frame size $s_{p,w-1}$ as shown in Equation (2.4)

$$e_{p,w} = \frac{B_{p,w-1} - s_{p,w-1}}{B_{p,w-1}} \tag{2.4}$$

The normalized error limits the error between -1 and 1 in most operating conditions, except for cases in which the image disturbances are too extreme. The frame size is then regulated with a PI controller as shown in Equation (2.5)

$$q_{p,w} = k_p \cdot e_{p,w} + k_i \cdot \sum_{t=1}^{w-1} e_{p,t} \tag{2.5}$$

The values of k_p and k_i are the proportional and integral gains of the controller. This control design achieves zero-steady state error in conditions with no disturbances during image capture, i.e. when $\delta s_{p,w} = 0$. The controller treats scene changes as load disturbances and regulates image quality to minimize it.

Image Quality Human perception of quality of an image does not correlate directly with the encoding quality q or frame-size s. The most widely used metrics to determine image quality are Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) which are simple to calculate. However, [Wang, Bovik, Sheikh, and Simoncelli, 2004] showed that high performance in these metrics do not necessarily correlate with better human perception of images. [Wang, Bovik, Sheikh, and Simoncelli, 2004] also proposed a new Structural Similarity Index Measure (SSIM) that has been widely studied since.

Paper I applies SSIM to compare the transmitted quality of images of two cameras. However, using SSIM in camera networks to evaluate image quality is impractical as it is dependent on comparison of two images. This requires the camera to save the non-encoded images and compare them to the encoded images online leading to high computational costs. The images can also be compared offline after image transmission however this information is outdated. SSIM was used offline for comparing images encoded with different regulation strategies in Paper I.

Manager

The network manager $\mathcal{M}$ allocates a fraction of the total bandwidth to each camera in the network through a game-theoretic approach. The game-theoretic resource manager is based on [Maggio, Bini, Chasparis, and Årzén, 2013] where the authors allocated computational resources to real-time applications periodically. Real-time applications are capable of regulating their service levels similar to cameras. The resource manager in [Maggio, Bini, Chasparis, and Årzén, 2013] was triggered periodically with constant resource usage evaluation and reallocation. This approach is not efficient for the camera network as camera scene changes are seldom and constant bandwidth evaluations can lead to unnecessary bandwidth reallocations.

The camera network manager receives the images transmitted by the cameras and evaluates the match between the allocated bandwidth and its utilization by the cameras. The manager also ensures that the total amount of bandwidth allocated to the cameras does not exceed the global available bandwidth $\mathcal{H}$. At each instant t that the manager is invoked, it allocates a fraction of the bandwidth $b_{c\in\mathcal{C},t}$ such that

$$\forall\, t, \mathcal{M} \quad \begin{array}{l} \text{selects } b_{*,t} = [b_{1,t}, \ldots, b_{n,t}] \\ \text{such that } \sum_{p=1}^{n} b_{p,t} = 1 \end{array} \tag{2.6}$$

The bandwidth fraction vector $b_{*,t}$ determines the fraction of bandwidth allocated to the respective camera in the network. The actual amount of bandwidth allocated to each camera is then determined by

$$B_{p,w} = b_{p,t} \cdot \mathcal{H} \tag{2.7}$$

where $\mathcal{H}$ is the amount of global bandwidth available to all cameras. The number of frames sent by the cameras in one allocation period is dependent on the configuration of the system and all frames in the allocation period have the same size of allocated bandwidth. If the current frame size of the camera is larger than the allocated bandwidth $s_{p,w} > B_{p,w}$ then the frame is dropped. On the other hand, $s_{p,w} \leq B_{p,w}$ indicates suboptimal utilization of the allocated bandwidth by camera c_p. Thus, ideally bandwidth allocation is approximately equal to the bandwidth allocated to the camera $s_{p,w} \approx B_{p,w}$. The manager recalculates the fraction of bandwidth allocated to each camera at every instance (periodically or event-based) it is invoked according to Equation (2.8)

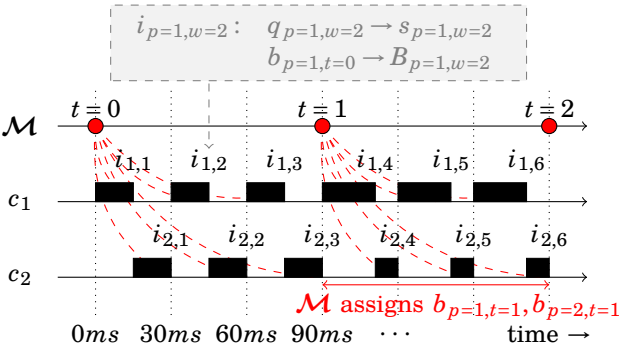


Figure 2.3 Example timeline of a Time-triggered manager [Nayak Seetanadi, Oliveira, Almeida, Årzén, and Maggio, 2018].

$$b_{p,t+1} = b_{p,t} + \varepsilon \cdot \left\{ -\lambda_{p,t} \cdot f_{p,t} + b_{p,t} \cdot \sum_{i=1}^{n} \left[\lambda_{i,t} \cdot f_{i,t} \right] \right\} \tag{2.8}$$

The matching function $f_{p,t}$ determines the match between the allocated bandwidth $B_{p,w}$ and the frame size $s_{p,w}$ of frame w for camera c_p. A simple choice for the matching function is the normalized error function from Equation (2.4). $\lambda_{p,t}$ determines the fraction of bandwidth adaptation performed by the network manager. A small value of $\lambda_{p,t}$ indicates that the network manager regulates the bandwidth of camera c_p with a lower priority and places a larger emphasis on the camera to regulate its frame size to avoid frame drops. $f_{p,t}$ and $\lambda_{p,t}$ are discussed in more detail in [Maggio, Bini, Chasparis, and Årzén, 2013].

Figure 2.3 shows an example timeline when $\mathcal{M}$ is triggered periodically with $\pi_{\mathcal{M}} = 3 \cdot \pi_{\text{alloc}} = 90ms$ and $\mathcal{C} = c_1, c_2$. During each π_{alloc}, each camera transmits three frames. Each bar on the timeline corresponds to $b_{p,w}$ when p is the id of camera c_p and w is the frame number.

Each camera is allocated equal amount of bandwidth when $t = 0$ ($b_{1,0} = b_{2,0}$). $\mathcal{M}$ allocates more bandwidth to camera c_1 at $t = 1$ ($b_{1,4} > b_{2,4}$) depending on the matching function of the cameras ($|f_1| > |f_2|$). $b_{1,\_}$ and $b_{2,\_}$ are recalculated every period, determining the actual bandwidth allocated, $B_{1,\_}$ and $B_{2,\_}$ respectively.

The time-triggered manager has two major drawbacks,

- **Choice of period $\pi_{\mathcal{M}}$:** Choosing a small value of $\pi_{\mathcal{M}}$ ensures that $\mathcal{M}$ recalculates the bandwidth allocation vector $b_{*,t}$ more often. This leads to faster adaptation to camera requirements but also leads to unnecessary adaptation during scenarios with no scene changes in the images captured by the cameras. The number of bandwidth recalculations performed by the manager is reduced by choosing a larger value of $\pi_{\mathcal{M}}$. However, it has the drawback of slow reactions to changes in camera bandwidth requirements causing dropped frames and bandwidth underutilization.

- **Resource allocation overhead:** The manager $\mathcal{M}$ recalculates the bandwidth at an allocation period $\pi_{\mathcal{M}}$ each time it is invoked. The vector $b_{*,t}$ is recalculated at each invocation according to Equations (2.6) and (2.8). The computational complexity of bandwidth recalculation is dependent on the number of cameras in the network leading to computation times that significantly impact the amount of time available for frame transmission.

These drawbacks are mitigated through the use of an event-based triggering scheme for the network manager. An event-based manager reallocates bandwidth to the cameras in the network only when required avoiding unnecessary manager interventions. The event-based manager intervenes when the number of cameras in the network changes and also when the value of normalized error of any camera in the network exceeds a triggering threshold τ_{thr}. τ_{thr} is a design choice that is dependent on the camera network and the scenes captured by the cameras. A small τ_{thr} forces the manager to intervene often even for small changes in the scenes being captured. On the contrary, a large τ_{thr} leads to fewer manager interventions placing an emphasis on camera adaptation.

2.2 Model Checking

Subsection 2.1 describes two triggering strategies for the manager, namely, time-triggered and event-triggered. Guarantees on the time-triggered manager are derived intrinsically based on work from [Chasparis, Maggio, Bini, and Årzén, 2016]. These safety guarantees do not necessarily hold for the event-triggered manager and they are obtained using model checking. Papers II, III and IV apply model checking to the event-triggered manager to provide guarantees on its behaviour. Model checking is the method of verifying functionality of a system that is modelled as a Finite State Machine (FSM).

Figure 2.4 shows the general flow of applying model checking to verify desired properties of a system. The system model and the system property to be verified are modelled using formal language and formal mathematics. We use Markov Decision Processes (MDP) to represent the different entities in the system, namely the manager and the cameras. The stochastic nature of image capture is modelled using probabilistic transitions and non-deterministic MDPs. System properties capture the different behaviours of the system for verification and are either qualitative or quantitative in nature. For example, some properties for the camera network are “Stability after Bandwidth allocation”, “Number of Frames Dropped”, “Number of Manager Interventions” and so on.

The model checking software indicates affirmative/negative property verification for qualitative properties, and the cost/reward incurred for quantitative properties. A property verification failure indicates that the model needs to be refined or the property constraints need to be relaxed. We use PRISM [Kwiatkowska, Norman, and Parker, 2011], a probabilistic model checker for formal modelling and analysis of stochastic and probabilistic systems. Probabilistic model checking is the formal technique for analyzing systems with

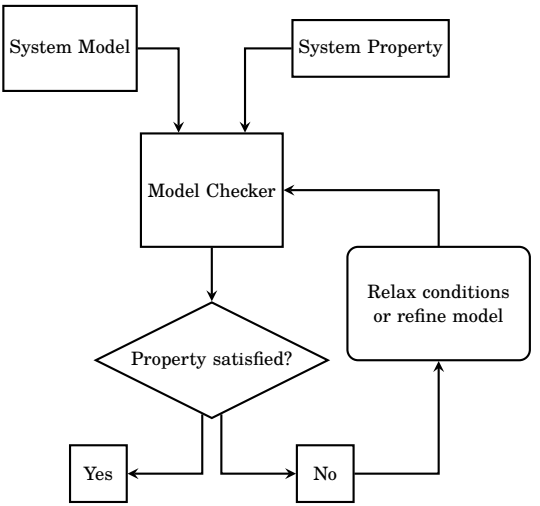


Figure 2.4 Model Checking Methodology

probabilistic and stochastic behaviours such as disturbances during image capture, randomized algorithms, communication networks with packet uncertainty, and other such dynamic systems.

System Models

The physical system being verified is modelled using the appropriate framework depending upon the relevant behaviour captured. Generally, systems are modelled as Markov chains consisting of states and transitions. Each state in the Markov chain captures the complete system in that state. All models in the camera network are modelled using Markov Decision Processes (MDPs). An MDP is an extension of Markov chains with the possibility of modelling discrete-time stochastic control processes through probabilistic transitions.

DEFINITION 1
An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a set of finite states, $\mathcal{A}$ is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a function that encodes the probability of transitioning from state s to state s' as a result of an action a, and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the choice of action a determines a transition from state s to state s'.

System Property

System properties describe the relevant behaviour of the system and are constructed using Probabilistic Computation Tree Logic (PCTL). Computational tree logic (CTL) is branching logic with a tree structure where different paths of the system are represented as individual branches. PCTL, an extension of CTL, is

used to construct both probabilistic and non-deterministic properties of the system model. Model checking softwares construct the whole state space of system model using computational trees for verification. PCTL properties are then interpreted to be true for either a state $s \in S$, or over a probabilistic path over the computational tree. PRISM supports a variety of operators for property construction including **P:** Probabilistic Operator, **S:** Steady State Operator, **R:** Reward Operator, **A:** For-All Operator, and **E:** There-Exists Operator. The following system properties are constructed for evaluation of the camera network using the relevant operators.

Qualitative Properties

- $\mathbf{P}_{\geq 1}[\mathsf{G}\ (\mathcal{H}_{\text{alloc}} = \mathcal{H})]$: The allocated bandwidth $\mathcal{H}_{alloc}$ is equal to the available bandwidth $\mathcal{H}$, globally with a probability of 1.
- $\mathbf{P}_{\geq 1}[\mathsf{F}\ (\mathsf{G}!(\mathsf{any\_change\_event}))]$: No changes, or no changes in the state transitions occur globally with a probability of 1.
- $\mathbf{P}_{\geq 1}[\mathsf{G}\ (\mathsf{used\_bw} = \mathsf{max\_bw})]$: The total amount of bandwidth used by the cameras is equal to the maximum amount of available bandwidth with a probability of 1.

Quantitative Properties

- $\mathbf{R}^{\mathsf{rm\_calls}}_{\mathsf{max/min=?}}\ [\mathsf{F\ end}]$: The maximum/minimum number of manager interventions during model checking. The variable _calls tracks the number of times the state `nm_calc_bw` is reached in the network manager MDP shown in Figure 2.5.
- $\mathbf{R}^{\mathsf{frames\_dropped}}_{\mathsf{max/min=?}}\ [\mathsf{F\ end}]$: The maximum/minimum number of frames dropped. The dropped frames are tracked by comparing the amount of bandwidth allocated and the frame size of the cameras and storing the value in variable `frames_dropped`.
- $\mathbf{R}^{\mathsf{cost}}_{\mathsf{max/min=?}}\ [\mathsf{F\ end}]$: The maximum/minimum cost accumulated by tracking a predefined cost function.

Applications of Model Checking

A variety of hardware and software systems have been successfully verified with model checking through a combination of different model and property types. Model checking has also been applied in industry owing to its ability to evaluate correctness of complex stochastic systems, see [Cimatti, 2001] for survey on industrial applications. Similarly, probabilistic model checking has been applied to verify different stochastic systems such as distributed algorithms [Kwiatkowska, Norman, and Parker, 2012; Lehmann and Rabin, 1981], network protocols [Duflot, Kwiatkowska, Norman, and Parker, 2006; Kwiatkowska, Norman, and Sproston, 2002] and biological systems [Heath, Kwiatkowska, Norman, Parker, and Tymchyshyn, 2006; Kwiatkowska, Norman, and Parker, 2008].

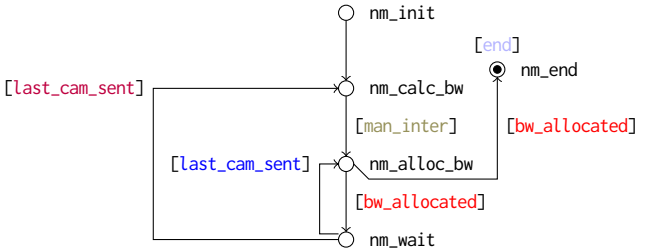


Figure 2.5 Network Manager, MDP Representation

2.3 Camera Network Models

The different entities of the camera network, namely the manager and the cameras, are built with independent system modules. The different entities of the network are synchronized by labels attached to the different transitions in the system modules. The modules capture the relevant behaviour of each of the entities as described in Section 2.

Network Manager

The network manager performs event-based resource allocation and is modelled identically in all instances, irrespective of the camera model in the network. Figure 2.5 shows a simplified MDP representation of the network manager as state-transition pairs. Each state in the MDP captures the current behaviour of the manager. The network manager MDP is completely deterministic and does not contain any probabilistic and non-deterministic transitions.

The state `nm_init` denotes the initial state of the network manager where the different properties of the manager are initialized. The network manager (re)calculates the amount of bandwidth allocation to the cameras in state `nm_calc_bw`. The state `nm_alloc_bw` indicates the bandwidth allocation by the network manager in that period. The label `[man_inter]` is used in the reward structure to calculate the number of manager interventions.

`nm_wait` denotes that the manager is waiting for cameras to transmit images and `nm_end` denotes the end of image transmission with a self-absorbing state. The end state is configured to be reachable after the transmission of a pre-determined number of frames. Labels `[bw_allocated]` and `[last_cam_sent]` indicate the completion of bandwidth allocation and completion of image transmission respectively. The labels are also used for synchronization of the various different modules during model checking.

Camera Models

The different camera models in the thesis describe three unique ways of representing camera behaviour described in Subsection 2.1. The cameras are modelled using deterministic, probabilistic, and non-deterministic MDPs.

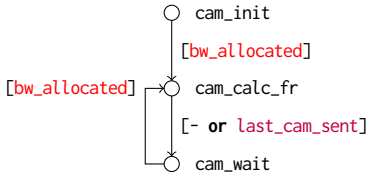


Figure 2.6 Deterministic Camera Model

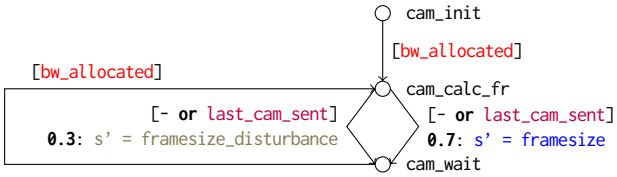


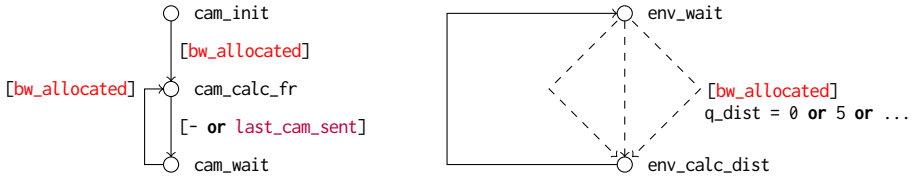
Figure 2.7 Probabilistic Camera Model

Deterministic MDPs model the camera as a completely deterministic state space. Deterministic models do not contain probabilistic or non-deterministic transitions between the different states in the state space. Figure 2.6 shows the state transition representation of a completely deterministic camera MDP. Each camera in the network is represented by an independent module with the same model. The model does not consider disturbances during image capture.

`cam_init` is the initial state of the camera. The camera transitions to the next state during the appropriate network manager transition. The transitions in the manager and the other camera modules are synchronized with the label [`bw_allocated`]. The camera calculates the resulting frame size without disturbances given by Equation 2.2 in the state `cam_calc_fr`. The label [`last_cam_sent`] is present only in the MDP of the last camera (all other cameras have no label, given by [`-`]) and indicates the transmission of the image by the final camera. The cameras wait for further bandwidth allocations in state `cam_calc_fr`. The cameras stop image transmissions when the manager MDP reaches the state `nm_end`.

Deterministic models are simple and small in nature but do not capture complete real life behaviour that exhibits uncertainty and stochasticity. Stochastic behaviour in real systems is modelled using probabilities and non-determinism in model checking.

Probabilistic models attach probabilities to transitions from one state to another while ensuring that all probabilities from a state sum to one. The models capture uncertain behaviour such as “The message is lost with a probability of 0.1", “The coin toss is heads with a probability of 0.5 and tails with a probability of 0.1" and “The system will fail with a probability of 0.001". The camera network model attaches a probability of 0.3 to the transition that causes disturbance during image capture and 0.7 to the transition with no disturbances.



(a) Camera

(b) Environment

Figure 2.8 Non-Deterministic Camera Model

Figure 2.7 shows the probabilistic model of a camera that models disturbances through probabilities on transitions. The disturbance is introduced into the generated frame size described in Equation (2.3). The uncertainty in image capture is modelled by the transitions from state `cam_calc_fr` to `cam_wait`. The two transitions correspond to two different equations. The transition on the right calculates the frame size without disturbances given by Equation (2.2) and is attached a probability of `0.7`. The transition on the left calculates the frame size with disturbances given by Equation (2.3) and is attached a probability of `0.3`. The two different transitions model the scenario in which disturbances occur less frequently in the cameras.

The addition of probabilities to the camera model increases the complexity and size of the resulting state space compared to the deterministic model. The larger state space leads to state space explosion and causes complications in property verification [Nayak Seetanadi, Årzén, and Maggio, 2019].

Non-deterministic MDPs are an alternative approach of modelling stochastic and uncertain behaviour in systems. Non-determinism is particularly useful in systems in which the probabilities of transition are not explicitly known. Stochastic systems with concurrent behaviour, consisting of an unknown adversary and similar systems are suitable for modelling with non-deterministic MDPs. In the non-deterministic camera model, an environment module inserts disturbances during image size calculations by varying the encoding quality.

Figure 2.8 shows the non-deterministic model of a camera together with an environment module. The camera model is identical to the deterministic camera with the same states, transitions and labels. The size of disturbances during frame size calculations are determined by the environment module and synchronized using the same label [`bw_allocated`]. The environment module randomly allocates a value to the disturbance of individual frames of each camera in the network leading to changes in the resulting frame sizes. The value of disturbance ranges from 0 (indicating a deterministic frame with no disturbances) to a value that determines the stochastic nature of the current scene. The different dashed lines in the figure denote the different transitions with unique disturbance values and all transitions are allocated a probability of 1. This forms

the non-deterministic choice capturing the disturbances that arise during image capture.

Models using non-determinism, similar to probabilities models, are more complex in their state space and larger in size compared to deterministic models. [Nayak Seetanadi, Årzén, and Maggio, 2019] showed that non-deterministic models scale better with respect to probabilities models. Non-deterministic models also lead to better property verification as they capture a varied behaviour of real systems. They also provide maximum and minimum values obtained during verification of quantitative properties leading to better evaluation of system performance.

Drawbacks of Model Checking

The largest drawback of model checking is the widely studied state space explosion problem [Clarke, Klieber, Nováček, and Zuliani, 2012]. State space explosion is caused by the exponential increase in the number of states describing a system. The large number of states is inevitable when describing complex systems with uncertainties in its behaviour. As model checking tools build and store the whole state space in memory, it causes memory issues during model building leading to failure. This restricts either the model complexity or the duration of system functionality modelled for verification.

Research into the state space explosion problem has developed different techniques for mitigating it through different approaches. Most of the approaches involve reducing the memory footprint of the state space either through symbolic model checking with binary decision diagrams [Baier, Clarke, Hartonas-Garmhausen, Kwiatkowska, and Ryan, 1997]. Similarly, counter example guided abstraction refinement builds only the appropriate states and bounded model checking applies SAT solvers to provide counter examples. Unfolding techniques search the constructed state space for property verification without considering all possible paths in concurrent systems modelling [McMillan and Probst, 1995] [Esparza and Heljanko, 2000]. These techniques are implemented inherently into the different model checking tools. In conjunction with advances in model checking tools, there are also specification guidelines in order to avoid state space explosion problems [Groote, Kouters, and Osaiweran, 2015].

3

Real-time Routing Network

This chapter provides an introduction to the real-time dual-delay routing networks investigated in papers V and VI. First, the chapter discusses the construction of the two different routing networks constructed. Then a short dive in reinforcement learning is provided, finally concluding with the description of the safe routing algorithm used for real-time packet routing.

A recent paper [Baruah, 2018] introduced a dual-delay model for communication in real-time networks consisting of computational nodes that are interconnected with directed edges. Each node x in the network has limited computational capacity and makes independent routing decisions, while each edge $e : (x \rightarrow y)$ from node x to node y in the network is characterized by two different transmission times:

- **Worst-case transmission time, c^W_{xy}:** The worst-case transmission time that the system is guaranteed to never violate even under extreme load. c^W_{xy} is obtained through worst case timing analysis before transmission of packets and is considered static.

- **Typical transmission time, $c^T_{xy} \in$ (0, c^W_{xy}]:** The typical transmission times capturing the current network behaviour and load. c^T_{xy} is pre-determined using knowledge about time-dependent network loads or explored dynamically through packet transmission.

The goal of real-time routing algorithms is to determine a route from source i to destination t that minimizes the total path transmission δ_{it} while also ensuring that δ_{it} is less than or equal to the pre-determined packet deadline D_F.

In general, routing algorithms can be divided into *static* and *dynamic* routing. In static routing the complete path for packet transmission is determined before the packet leaves the source node. This has low complexity but can lead to suboptimal transmission times in networks with varying c^T. In dynamic routing, routing decisions are taken at each node considering the current network load and delays already encountered and it generally leads to lower transmission times compared to static routing.

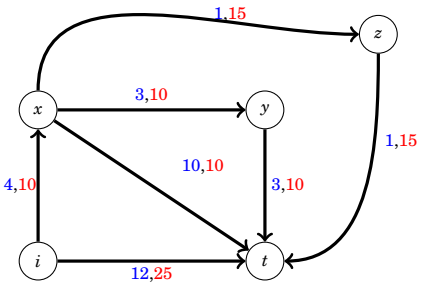


Figure 3.1 Routing network from [Nayak Seetanadi, Årzén, and Maggio, 2020]

This objective of minimizing delays while subject to constraints can be represented as a cost minimization problem. Prior work done in the field of cost minimizing for paths with multiple constraints can be divided into bi-criteria and multi-criteria optimization. Bi-criteria shortest path problems [Hansen, 1980; Chen and Nie, 2013] consist of edges with two cost parameters similar to the one considered in the thesis. These problems have also been extended to multi-criteria shortest path problems [Gandibleux, 2006] with varying non-linear cost functions. The solutions to multi-criteria problems generally assume the different costs to be unrelated to each other. Also research in this area is generally constricted to static optimization with non-varying cost. This is not a realistic assumption for a routing network with varying load and dynamic structure changes. This thesis solves the optimization problem using reinforcement learning with a modified exploration policy to ensure safety.

3.1 Real-Time Network Construction

The thesis considers two example real-time networks constructed with the dual-delay specifications to evaluate real-time routing algorithms. Each network is a graph $\mathcal{G}$ with independent nodes and multiple outgoing edges from each node in the network. Each edge $e : (x \rightarrow y)$ is a directed edge from node x to node y and is characterized by the two delays described above.

Routing network from [Nayak Seetanadi, Årzén, and Maggio, 2020]

Figure 3.1 shows a directed graph with *five* nodes and *seven* edges. The network is constructed with directed edges and no loops exist in the network. The typical delays over each edge are show in blue and the worst-case delays are shown in red. The example network was first introduced in the presentation of [Baruah, 2018] and implemented for evaluation of routing algorithms in papers [Nayak Seetanadi, Årzén, and Maggio, 2020] and [Nayak Seetanadi, 2020].

There exist *four* possible paths from source i to destination t for different end-to-end delay guarantees. The following table shows the worst-case delay and typical delays encountered on the paths when c_{xy}^T is static and known.

Path	Worst-case Delay	Typical Delay
$i \rightarrow t$	25	12
$i \rightarrow x \rightarrow t$	20	14
$i \rightarrow x \rightarrow y \rightarrow t$	30	10
$i \rightarrow x \rightarrow y \rightarrow t$	40	6

The path $(i \rightarrow x \rightarrow y \rightarrow t)$ has the typical delay of 10 and guaranteed worst-case delay of 30, whereas path $(i \rightarrow x \rightarrow z \rightarrow t)$ has the typical delay of six and a guaranteed delay of 40. In a completely deterministic network, i.e. where the actual transmission time δ_{xy} is equal to the typical transmission time c_{xy}^T over each edge $e : (x \rightarrow y)$, the choice of routing path is trivial and can be performed with the help of a similar routing table. The choice of routing path becomes complicated when the actual transmission times differ from the typical transmission times and is highly dependent on the given deadline D_F and the amount of delay already encountered by the packet on arrival at a node.

Routing Tree Network from [Nayak Seetanadi, 2020]

Figure 3.2 shows a tree network modified with more interconnections between the nodes. Compared to the previous network, the modified tree network consists of a larger number of feasible paths for packet routing capturing a realistic model of a modern routing network with larger number of nodes and connections. Each edge in the network is directed and is assigned a random value for typical transmission times c^T, and worst-case transmission times c^W. Paper VI uses the modified tree network to evaluate complexity of the different routing algorithms in the presence of a large number of nodes and edges.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning that works on the premise of an *agent* learning about the *environment*, to maximize a long-term *reward*. Figure 3.2 shows the general feedback loop through which an agent interacts with the environment. The real-time routing algorithms described in the thesis are based on reinforcement learning with few modifications to guarantee safe routing of packets.

The following basic terminology describes the different components of reinforcement learning algorithms:

- **State *s*:** The state s of the system is a complete description of the current status of the system. It contains all information about the current environ-

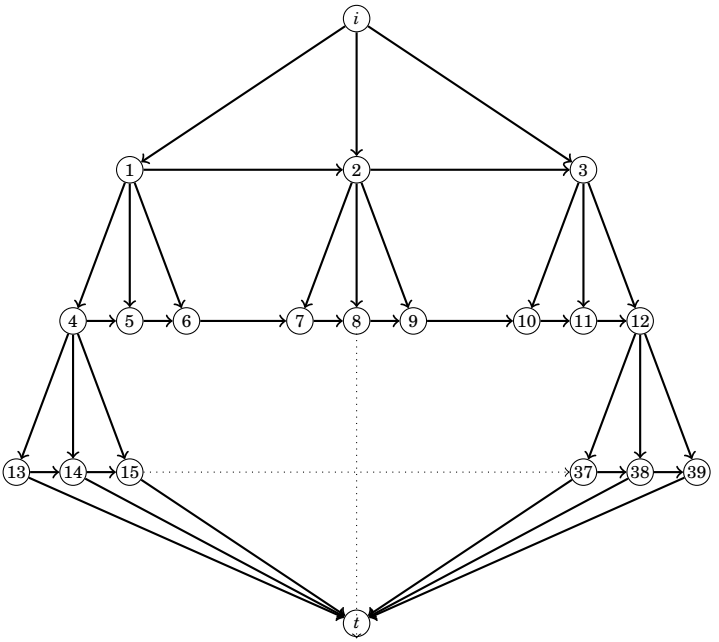


Figure 3.2 Modified Tree Network [Nayak Seetanadi, 2020]

ment that is observable by the agent. The term state space describes the current combination of states and actions that are known to the agent.

- **Action a:** The agent interacts with the environment through an action a to transition from the current state s to the next state s'. The complete set of valid actions is termed as the action space which can be either discrete or continuous depending upon the environment.

- **Policy π:** The agent chooses an action a from the feasible action space with the help of a policy π. The policy of an agent dictates the exploration of the environment by it and is generally based on a probability distribution.

The agent interacts with the environment through actions and manipulates the current state of the environment. The environment rewards actions performed by the agent either instantaneously or in a delayed manner. The agent executes multiple actions during one complete system simulation, called an *episode*, and aims to improve knowledge about the environment that is already obtained during subsequent episodes. Exploration of the different states is performed through a probabilistic distribution that dictates the action chosen from a state. The tradeoff between visiting states that consistently award large rewards and exploring states in search of higher rewards is well known and termed the exploration/exploitation tradeoff [Sutton and Barto, 2018].

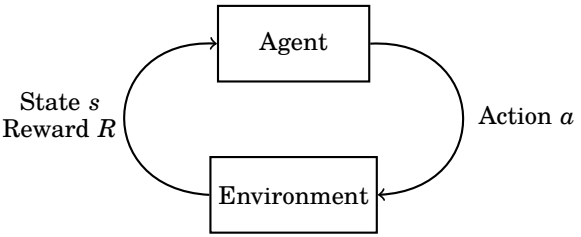


Figure 3.3 Reinforcement Learning Feedback Loop

The broad premise of reinforcement learning can be applied to a variety of topics such as:

- An agent learning to play the popular video game **Super Mario Bros**. The agent can move left, right and jump forming its action space. The environment rewards the agent positively for progressing further through the level and defeating enemies while penalizing the agent for loosing lives.

- An autonomous car driving through a maze to its destination. The agent is given positive rewards on how fast it reaches the destination, with a negative reward for crashing. The action space for the car is the amount of acceleration and direction of steering.

- Packet routing through a network. The agent transmits packets from source to destination through multiple possible paths with minimal delays. The action space for agent is the next edge for packet transmission an the agent obtains a positive reward inversely proportional to the packet delay.

RL has found recent success in the context of video games where it has achieved superhuman level of expertise [Silver et al., 2018; Vinyals et al., 2019]. Recent research attempts to emulate the success of RL algorithms on real-time systems with safety constraints. These safety constraints necessary for critical systems are opposite to the inherent risk of RL algorithms and lead to safety violations in stochastic systems. RL algorithms can be modified to reduce, or even eliminate risk and are termed as safe reinforcement learning algorithms. see [García and Fernández, 2015] for a survey on current research landscape of safe reinforcement learning. In general, RL algorithms are augmented to consider safety constraints through:

- Modifying the optimization criteria. Minimize the probability of reaching unsafe states by attaching a high cost.

- Exploration through safe actions. Mitigate risk by taking only safe actions during the exploration phase. Actions are determined to be safe through expert system knowledge.

Algorithm 1 Pre-Processing:

1: **for** each node x **do**
2: **for** each outgoing edge $(x \rightarrow y)$ **do**
3: $c_{xyt} = c^W_{xy} + \min(c_{y\_t})$

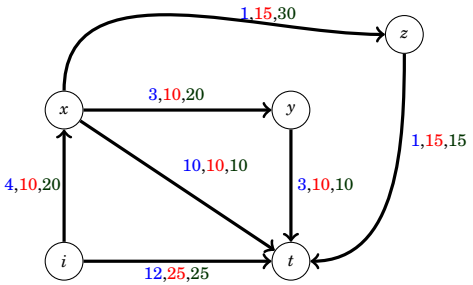


Figure 3.4 Network after pre-processing

This thesis applies a modified exploration policy to safely route packets through real-time communication networks. Safe RL routing algorithm developed in the thesis consists of three independent algorithms. A *Pre-processing* algorithm run during network initialization determines safe edges for packet transmission using worst-case delays. A *Node Logic* algorithm is run at each node at the arrival of each packet to determine the optimal edge for packet transmission that is also safe. Finally, the *Environment Reward* algorithm evaluates the optimality of the routing path on arrival of the packet at the destination.

Pre-processing Stage

Safe exploration in RL algorithms is performed with knowledge about the system to ensure safety. We run a pre-processing algorithm to obtain information on the minimum delay to destination t over each edge $e : (x \rightarrow y)$ given by c_{xyt}. Papers V and VI use a simple Dijkstra's shortest path algorithm [Dijkstra, 1959; Mehlhorn and Sanders, 2008] for pre-processing. For each outgoing edge $e : (x \rightarrow y)$ from each node x in the network, we add the worst-case delay over the edge, c^W_{xy}, to the minimum worst-case delay to destination from node y, c_{y_t} to obtain the minimum guaranteed worst-case delay from node x to destination t over the edge $e : (x \rightarrow y)$, c_{xyt}. The term $\min(c_{y\_t})$ denotes the least guaranteed worst-case delay over all outgoing edges of node y. The pre-processing algorithm is run inversely to the direction of packet transmission as preceding nodes depend on delay values of links from the following nodes. The values obtained after the pre-processing stage determine the safe bounds for packet transmission and avoid unsafe edges during exploration.

Figure 3.4 shows the real-time network from [Nayak Seetanadi, Årzén, and

Maggio, 2020] with c_{xyt} shown over each edge obtained after execution of the pre-processing algorithm. Algorithm 1 shows the pseudocode of the pre-processing algorithm that is executed on a global level during network initialization and network reconfiguration to obtain worst-case transmission times to destination, c_{xyt}.

Optimization phase

The optimization phase, also called value iteration, of RL estimates and updates the value function of a state and is an integral part of all RL algorithms. The function determines the value of being in the current state and the reward expected from the remainder of the episode. The same can be extended to estimate values of state action pairs and the reward expected after taking a particular action. RL algorithms constantly update value functions of states (or state action pairs) and capture the evolution of the environment over numerous episodes.

RL algorithms can be mainly divided into model-based and model-free algorithms, depending upon whether the agent has access to a model of the environment. Model-based algorithms make efficient use of the information obtained per episode from the environment and have been studied extensively in other research domains. The problem of finding optimal solutions for model-based problems has been studied in optimal control through the use of Dynamic Programming (DP) [Bertsekas, Bertsekas, Bertsekas, and Bertsekas, 1995] [Kirk, 2004]. DP is also used to solve MDPs and compute optimal policies given a perfect model of the environment. This requires complete knowledge about transition probabilities between the different states and the corresponding rewards obtained, which is not always possible in most real world problems.

Model-free algorithms, such as Monte Carlo methods and Temporal-Difference learning, are popular in reinforcement learning due to their simple implementation and the large number of episodes generally performed in RL. Monte Carlo (MC) methods are simple, model-free algorithms that use experience obtained through system sampling to execute optimal actions. These methods do not assume complete knowledge of the environment and learn through interactions with it. Value iteration in MC methods are performed after completion of one episode and this knowledge is used in the subsequent episodes.

Temporal-Difference (TD) learning is a model-free value iteration method that combines properties of both Monte-Carlo methods and dynamic programming. TD learning explores the environment through samples similar to MC methods. Compared to MC methods however, TD learning methods update state value estimates without waiting for the episode termination, resulting in faster learning. A unique version of TD learning where value iteration is performed after each step as opposed to at the end of the episode, is termed as TD(0) or one-step TD and is shown in Equation (3.1).

$$Q(s,a) = Q(s,a) + \alpha \cdot \left(\mathcal{R} + \gamma \cdot \max\left(Q(s',a')\right) - Q(s,a)\right) \tag{3.1}$$

States s and s' are the current state and next state respectively and a is the action taken from state s to transition to state s'. $Q(s,a)$ and $Q(s',a')$ are the Q-values for the current and next state-action pairs respectively. The term $\max(Q(s',a'))$ denotes the estimated maximum Q-value for all actions from state s'. α is the learning rate that determines the amount of old state information that is overwritten with new information obtained in the current episode. The discount factor γ determines the emphasis placed by the algorithm on future rewards. A small value of γ means the agent will only consider current rewards whereas a large value will make the agent try to obtain higher long-term rewards. $\mathcal{R}$ is the reward from the environment obtained from state s after taking action a.

This thesis applies *TD(0)* where values of state action pairs are updated immediately after packet transmission, while minimizing transmission of unnecessary messages. Each state in the routing algorithm is a duplex of the current node, x, and the amount of time remaining for packet transmission, $(D_F - \delta_{ix})$, where D_F is the pre-determined packet deadline and δ_{ix} is the time elapsed upon packet arrival at node x.

The learning agent is tuned to obtained large rewards and performs actions consistently that have previously resulted in minimal transmission times. New paths from a state are explored regularly to ensure that any new paths, or paths that were previously congested and result in lesser delays, are considered for future transmission. This continuous exploration ensures that the agent adapts to congestions in the network at expense of few packets having comparatively higher delays.

Exploration policy

The exploration policy of RL algorithms determines the method with which an agent chooses an action a from a given state s in order to maximize the obtained reward. The policy also explores unknown states in the environment in search of higher rewards. In a dynamic environment, the algorithm also explores states that previously returned poor rewards.

We use ϵ-greedy exploration algorithm in the thesis to determine the outgoing edge from each node on packet arrival. ϵ-greedy exploration provides a balance between exploration and exploitation in RL. The term ϵ in ϵ-greedy refers to the probabilities that determines the choice of action from a state. The policy chooses the best action, i.e. the action that has the maximum value associated with it, with a probability of $(1-\epsilon)$ in most episodes. The policy also explores new and sub-optimal actions with a small probability of ϵ.

The ϵ-greedy algorithm in the thesis applies a modified probability distribution to ensure safe exploration by attaching a probability of 0 to unsafe edges. This exploration policy ensures that new routing paths are explored regularly in search of links with lower delays resulting in load balancing while also ensuring that the paths chosen by the algorithm are safe.

Algorithm 2 shows the pseudocode of the ϵ-greedy exploration algorithm and *TD(0)* optimization for packet routing. The algorithm is run at each node

Algorithm 2 Node Logic (x)

1: **for** Every packet **do**
2: **if** x = source node i **then**
3: $D_x = D_F$ // Initialize the deadline
4: $\delta_{it} = 0$ // Initialize total delay for packet = 0
5: **for** each edge $(c \rightarrow y)$ **do**
6: **if** $c_{xyt} > D_x$ **then** // Edge is infeasible
7: $P(x|y) = 0$
8: **else if** $Q((x, D_x), y) = \max(Q((x, D_x), \_ \in \mathcal{F}))$ **then**
9: $P(x|y) = (1 - \epsilon)$
10: **else**
11: $P(x|y) = \epsilon/(size(\mathcal{F} - 1))$
12: Choose edge $(x \rightarrow y)$ with Probability P
13: Observe δ_{xy}
14: δ_{it} += δ_{xy}
15: $D_y = D_x - \delta_{xy}$
16: R = Environment Reward Function(x, δ_{it})
17: $Q((x, D_x), y)$ = Value iteration
18: **if** y = Destination node t **then**
19: DONE

x at the arrival of the packet. If the node is the source, the deadline is initialized to the final deadline of the packet, $D_x = D_F$, and the total path delay is initialized to 0, $\delta_{it} = 0$. The exploration phase of the algorithm is shown in Lines *five* to 12 and value iteration is shown in Lines 13 to 17.

The exploration phase of the algorithm evaluates the feasibility of each outgoing edge from node x, $(x \rightarrow y)$. If $c_{xyt} > D_x$ then the edge is unsafe and is assigned probability of 0 and is not chosen for packet transmission. The remaining safe edges form the set of feasible edges $\mathcal{F}$ for packet routing. The edge associated with the maximum Q value, $Q((x, D_x), y) = \max(Q((x, D_x), \_ \in \mathcal{F}))$, is assigned the highest probability $P(x|y) = (1 - \epsilon)$. The rest of feasible edges are assigned probabilities such that $P(x|y) = \epsilon/(size(\mathcal{F} - 1))$. The outgoing edge $(x \rightarrow y)$ is then chosen with probability P.

The value iteration phase evaluates the value of being in the current state and it is updated after receiving the reward from the environment. The state value update is performed in Line 17 of the node logic algorithm. The choice of the value iteration algorithm depends upon the different parameters of the network. A simple choice is updating values of the state action pairs using the TD(0) algorithm given by Equation (3.1).

Reward Function

The reward function is the score obtained by the agent to determine the net positive/negative effect of actions taken during the episode. Thus the objective of

Algorithm 3 Environment Reward Function(x, δ_{it})

```
1: Assigns the reward at the end of transmission
2: if Node x = Destination node t then
3:     R = D_F − δ_it
4: else
5:     R = 0
```

an agent is to maximize the total reward over multiple episodes. The agent also alters its exploration policy through the reward function due to low rewards obtained due to a particular action. The reward function for real-time routing is the total amount of time saved when the packet reaches its destination. The reward obtained reduces when a particular edge is congested, altering the exploration policy to choose other edges for packet transmission.

Algorithm 3 shows the pseudocode of the reward function for the safe routing algorithm. The reward R is obtained after each edge traversal by the algorithm. If $x \neq t$, the packet is still in transmission and the destination is not reached. The episode continues and the packet is propagated to the next node. If $x = t$, the destination is reached and the reward is calculated. The positive reward is the amount of time saved during complete packet transmission. It is calculated as $R = D_F - \delta_{it}$ and awarded at the end of the episode.

4

Conclusion and Future Work

The thesis investigated the application of control techniques to improve performance and ensure correctness of surveillance cameras and real-time routing networks. This was accomplished through control techniques with co-application of model checking and reinforcement learning techniques. The thesis showed that application of model checking simplifies verification of the correctness and fairness of the event-based manager. The self-adaptive cameras in the network regulated their bandwidth utilization efficiently through simple PI controller minimizing error as opposed to complex models that rely on estimating the frame size of images to fit the allocated bandwidth. The properties of the camera network were constructed and verified using deterministic, probabilistic, and nondeterministic models built using Markov Decision Process (MDP). The thesis also party investigated strategy generation using model checking and contention between the different entities in the camera network model. Books [Clarke, Henzinger, Veith, and Bloem, 2018] and [Baier and Katoen, 2008] provide further information on nuances of model checking and the various modelling and property verification techniques.

The dual-delay model in the second part of the thesis implemented safe packet routing in real-time networks. The algorithms proposed in the thesis ensured safety by exploring only safe paths using expert information on the system obtained through a pre-processing algorithm. The routing algorithm is then run independently at each node to route packets on paths with minimal delays, while exploring sub-optimal paths in search of better rewards. This continuos exploration ensures that the routing algorithms react to time-varying delays and link congestions and perform automatic load balancing over different outgoing paths. The thesis showed the possibility of application of powerful reinforcement learning algorithms to time-sensitive critical systems using system knowledge. [Sutton and Barto, 2018] is the premier authority on reinforcement learning and provides further analysis of the TD learning algorithm used in the thesis. The book provides an introduction to construction of the RL environment and the different learning techniques applied to various problems.

The following areas show promise for future research and expansion on ideas presented in the thesis.

The self adaptive cameras in the surveillance network from the thesis are assigned equal priorities in the network. Their relative importance is then calculated using the error function and the bandwidth is allocated to them accordingly. Instantaneous changes in the scenes captured by the camera can lead to large changes in the frame sizes, causing frame drops. An interesting area for investigation is application of simple image processing with object and motion detection to predict the camera priorities and reduce frame drops due to bandwidth mismatch.

Paper I applies Structural Similarity Index (SSIM) [Wang, Bovik, Sheikh, and Simoncelli, 2004] to perform image analysis and compare the quality of encoded images. SSIM was not used in the following research due to its high computation overhead at the camera. This can be mitigated by applying recent advancements in edge computing for complex and fast image analysis at the cameras. This image analysis could make it possible for informed priority assignments to the cameras without complete dependence on the error function for bandwidth allocation.

Paper II explored strategy generation for the camera network through the application of model checking. The model checking tool was tasked with investigating a strategy to minimize cost that penalized manager interventions and dropped frames with a single cost. This could be expanded to generate different strategies during the different stages of operation. For example, attaching a higher cost for consecutive dropped frames as the camera would lose data for a longer period of time as opposed to singular dropped frames which would lead to choppy video but better information.

The camera models built for model checking in papers II, III and IV were restricted in their scope due to tool limitations. The complexity, number of cameras in the surveillance network, and the time horizon of property verification was low due to the state space explosion problem. Current research into application of machine learning for state space reduction and prediction is promising but was not accessible during the writing of the thesis. A possible future direction of research can utilize these new techniques to build complex non-deterministic models with larger time horizon for verification.

[Edpalm, Martins, Maggio, and Årzén, 2018] investigated frame size estimation when using H.264 video compression. The authors compared different estimation models including the linear model from the thesis. The linear camera model under performs other models as the model is more suitable for feedback-based frame size regulation as opposed to frame size estimation. An interesting extension is using the linear camera model for bandwidth regulation of video with inter-frame compression techniques such as H.264. This would also evaluate disturbance rejection and regulation of video bandwidth in the presence of embedded audio.

The dual-delay routing used in the thesis was introduced recently [Baruah, 2018] and shows promise in its application to various domains. Paper V explored

the application of the dual-delay model for safe routing using safe-reinforcement learning. Paper VI showed that the choice of routing algorithm is non-trivial and is dependent on network size and requirements. Both papers used NetworkX [Hagberg, Schult, and Swart, 2008], a graph emulator to implement the example networks without complete network emulation. It would be interesting to investigate the performance of the routing algorithms with full IP-stack on a network emulator or even a real network.

The dual-delay model is unique as previous research did not investigate dual-criteria optimization for two costs that are independent of each other. The dual-delay model can naturally represent traffic flows on highways and evaluate different routing algorithms of transportation networks. Reinforcement learning can then be applied to optimize traffic over highways and reduce congestions.

Bibliography

Baier, C., E. M. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan (1997). "Symbolic model checking for probabilistic processes". In: *Automata, Languages and Programming*, pp. 430–440.

Baier, C. and J.-P. Katoen (2008). *Principles of model checking*. MIT press. ISBN: 9780262026499.

Baruah, S. (2018). "Rapid routing with guaranteed delay bounds". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. Nashville, TN, USA. DOI: 10.1109/RTSS.2018.00012.

Bertsekas, D. P., D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas (1995). *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA.

Chasparis, G. C., M. Maggio, E. Bini, and K.-E. Årzén (2016). "Design and implementation of distributed resource management for time-sensitive applications". *Automatica* **64**, pp. 44–53. DOI: 10.1016/j.automatica.2015.09.015.

Chen, P. and Y. Nie (2013). "Bicriterion shortest path problem with a general nonadditive cost". *Transportation Research Part B: Methodological* **57**, pp. 419–435. DOI: 10.1016/j.trb.2013.05.008.

Cimatti, A. (2001). "Industrial applications of model checking". In: *Modeling and Verification of Parallel Processes: 4th Summer School, MOVEP 2000 Nantes, France, June 19–23, 2000 Revised Tutorial Lectures*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 153–168. DOI: 10.1007/3-540-45510-8_6.

Clarke, E. M., T. A. Henzinger, H. Veith, and R. Bloem, (Eds.) (2018). *Handbook of Model Checking*. Springer. ISBN: 978-3-319-10574-1.

Clarke, E. M., W. Klieber, M. Nováček, and P. Zuliani (2012). "Model checking and the state explosion problem". In: *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*. Berlin, Heidelberg, pp. 1–30.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**:1, pp. 269–271. DOI: 10.1007/BF01386390.

Ding, W. and B. Liu (1996). "Rate control of mpeg video coding and recording by rate-quantization modeling". *IEEE Transactions on Circuits and Systems for Video Technology* **6**:1, pp. 12–20. DOI: 10.1109/76.486416.

Duflot, M., M. Kwiatkowska, G. Norman, and D. Parker (2006). "A formal analysis of bluetooth device discovery". *International Journal on Software Tools for Technology Transfer* **8**:6, pp. 621–632. DOI: 10.1007/s10009-006-0014-x.

Edpalm, V., A. Martins, K.-E. Årzén, and M. Maggio (2018). "Camera Networks Dimensioning and Scheduling with Quasi Worst-Case Transmission Time". In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany, 17:1–17:22. ISBN: 978-3-95977-075-0.

Edpalm, V., A. Martins, M. Maggio, and K.-E. Årzén (2018). *H.264 Video Frame Size Estimation*. Technical Reports TFRT-7654. Department of Automatic Control, Lund Institute of Technology, Lund University.

Esparza, J. and K. Heljanko (2000). "A new unfolding approach to ltl model checking". In: Montanari, U. et al. (Eds.). *Automata, Languages and Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 475–486. ISBN: 978-3-540-45022-1.

Gandibleux, X. (2006). *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. International Series in Operations Research & Management Science. Springer US.

García, J. and F. Fernández (2015). "A comprehensive survey on safe reinforcement learning". *Journal on Machine Learning Research* **16**:1, pp. 1437–1480.

Groote, J. F., T. W. Kouters, and A. Osaiweran (2015). "Specification guidelines to avoid the state space explosion problem". *Software Testing, Verification and Reliability* **25**:1, pp. 4–33. DOI: https://doi.org/10.1002/stvr.1536.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). "Exploring network structure, dynamics, and function using networkx". In: Varoquaux, G. et al. (Eds.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, pp. 11–15. URL: https://www.researchgate.net/publication/236407765_Exploring_Network_Structure_Dynamics_and_Function_Using_NetworkX.

Hansen, P. (1980). "Bicriterion path problems". In: *Multiple Criteria Decision Making Theory and Application*. Springer, Berlin, Heidelberg, pp. 109–127. DOI: 10.1007/978-3-642-48782-8_9.

Heath, J., M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn (2006). "Probabilistic model checking of complex biological pathways". In: *Proc. Computational Methods in Systems Biology (CMSB'06)*. Vol. 4210. Springer, pp. 32–47. DOI: 10.1007/11885191_3.

Kirk, D. E. (2004). *Optimal control theory: an introduction*. Courier Corporation.

Kwiatkowska, M., G. Norman, and D. Parker (2008). "Using probabilistic model checking in systems biology". *ACM SIGMETRICS Performance Evaluation Review* **35**:4, pp. 14–21.

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

Kwiatkowska, M., G. Norman, and D. Parker (2012). "Probabilistic verification of herman's self-stabilisation algorithm". *Formal Aspects of Computing* **24**:4, pp. 661–670.

Kwiatkowska, M., G. Norman, and J. Sproston (2002). "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol". In: *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*. Vol. 2399. LNCS. Springer, pp. 169–187. DOI: 10.1007/3-540-45605-8_11.

Legay, A., B. Delahaye, and S. Bensalem (2010). "Statistical model checking: an overview". In: *International Conference on Runtime Verification*. LNCS 6418. Springer, pp. 122–135. DOI: 10.1007/978-3-642-16612-9_11.

Lehmann, D. and M. Rabin (1981). "On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract)". In: *Proc. 8th Annual ACM Symposium on Principles of Programming Languages (POPL'81)*, pp. 133–138. DOI: 10.1145/567532.567547.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". In: *Euromicro Conference on Real-Time Systems*. DOI: 10.1109/ECRTS.2013.17.

McMillan, K. L. and D. K. Probst (1995). "A technique of state space search based on unfolding". *Formal Methods in System Design* **6**:1, pp. 45–65. DOI: 10.1007/BF01384314.

Mehlhorn, K. and P. Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox*. 1st ed. Springer. ISBN: 9783540779773.

Nayak Seetanadi, G., K.-E. Årzén, and M. Maggio (2019). "Model checking a self-adaptive camera network with physical disturbances". In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. DOI: 10.1109/ICAC.2019.00021.

Nayak Seetanadi, G., J. Camara, L. Almeida, K.-E. Årzén, and M. Maggio (2017). "Event-driven bandwidth allocation with formal guarantees for camera networks". In: *2017 IEEE Real-Time Systems Symposium (RTSS)*. DOI: 10.1109/RTSS.2017.00030.

Nayak Seetanadi, G. (2020). "Adaptive routing for real-time networks with dynamic deadlines using safe reinforcement learning". In: *ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI)*. Under Review.

Nayak Seetanadi, G., K.-E. Årzén, and M. Maggio (2020). "Adaptive routing with guaranteed delay bounds using safe reinforcement learning". In: *Proceedings of the 28th International Conference on Real-Time Networks and Systems*. DOI: 10.1145/3394810.3394815.

Nayak Seetanadi, G., L. Oliveira, L. Almeida, K.-E. Årzén, and M. Maggio (2018). "Game-theoretic network bandwidth distribution for self-adaptive cameras". In: *SIGBED Review. Association for Computing Machinery*. DOI: 10.1145/3267419.3267424.

Silver, D. et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play". *Science* **362**:6419, pp. 1140–1144. DOI: 10.1126/science.aar6404.

Silvestre, J., L. Almeida, R. Marau, and P. Pedreiras (2007). "Dynamic qos management for multimedia real-time transmission in industrial environments". In: *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*. DOI: 10.1109/EFTA.2007.4416963.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press. ISBN: 9780262039246.

Vinyals Orioland Babuschkin, I. et al. (2019). "Grandmaster level in starcraft ii using multi-agent reinforcement learning". *Nature* **575**:7782, pp. 350–354. DOI: 10.1038/s41586-019-1724-z.

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). "Image quality assessment: from error visibility to structural similarity". *IEEE Transactions on Image Processing* **13**:4. DOI: 10.1109/TIP.2003.819861.

Paper I

Game-Theoretic Network Bandwidth Distribution for Self-Adaptive Cameras

Gautham Nayak Seetanadi Luis Oliveira Luis Almeida

Karl-Erik Arzén Martina Maggio

Abstract

Devices sharing a network compete for bandwidth, being able to transmit only a limited amount of data. This is for example the case with a network of cameras, that should record and transmit video streams to a monitor node for video surveillance. Adaptive cameras can reduce the quality of their video, thereby increasing the frame compression, to limit network congestion. In this paper, we exploit our experience with computing capacity allocation to design and implement a network bandwidth allocation strategy based on game theory, that accommodates multiple adaptive streams with convergence guarantees. We conduct some experiments with our implementation and discuss the results, together with some conclusions and future challenges.

1. Introduction

Nowadays, networked devices became commonplace, from surveillance cameras to industrial sensors and actuators, or even team of mobile robots. If these devices access the network in an on-demand fashion, sharing bandwidth may result in problems due to network congestion. On the contrary, when the number of devices is unchanged during execution and the communication pattern is very streamlined, network dimensioning should be enough to handle all the simultaneous transmissions in a timely manner.

Unfortunately, there are circumstances in which a proper dimensioning of the network capacity is either impossible, or too expensive. This can for example be the case with a network of cameras. Suppose we have a surveillance camera network, where a certain number of cameras are monitoring a given area. In general, if the designer allocates to each camera the maximum bandwidth they may require, there will be a significant bandwidth over-provisioning. A more efficient approach would be to allocate an average bandwidth so that the cameras can transmit their video streams with an average quality. However, if something is happening in one area – for example a lot of movements are detected by one camera – the bandwidth of the corresponding video stream may be increased, granting it better quality. At the same time, another area may be empty and therefore the corresponding bandwidth can be decreased to accommodate the higher quality stream. The camera network becomes therefore *adaptive* and is able to adjust to the characteristics of the execution environment.

Problem Statement. We consider a system composed of a set of cameras that send video streams to a monitoring node through a shared Ethernet network that supports virtual channels with bandwidth reservations. The cameras must respect their assigned bandwidth, therefore, they run some basic computation on the captured frames to determine the compression level that they should apply. The bandwidth in the network needs to be dynamically allocated at runtime by a monitoring node to accommodate the transient needs of the different cameras. The monitoring node needs to quickly redistribute the available bandwidth introducing as little additional overhead as possible, in particular concerning the transmission of additional information.

Contribution. We propose to achieve the mentioned low-overhead adaptation by decoupling the action of the *resource manager*, in charge of the network bandwidth distribution, and the cameras. Recently, the same strategy has been adopted for CPU allocation [Maggio, Bini, Chasparis, and Årzén, 2013], where the decoupling of adaptation at the application level and of the adjustment of the scheduling parameters has proven successful. The consequences of allocating CPU can be disruptive for the system but the action itself of allocating CPU itself is is a fairly "safe" operation – with respect to the amount of overhead introduced for the scheduling parameter change – on the contrary changing the channel size has a non-negligible additional overhead in terms of messages exchange, and some associated risks. In fact, if the amount of bandwidth is not enough to stream the frames, every other frame transmission is dropped and a significant

reduction in effective video frame rate takes place, with strong negative impact on quality. This paper presents the implementation of a resource manager, allocating network bandwidth to a network of self-adaptive cameras, preserving some guarantees on the transmission of the streams.

Related Work. The topic of self-adaptive cameras has already been investigated for a long time, particularly in the scope of video transmission over the Internet or in local area networks [Vandalore, Feng, Jain, and Fahmy, 2001; Communication, 2004; Rinner and Wolf, 2008]. Research has mainly focused on two complementary issues, video transmission and image compression. The former led to standard protocols such as Real-Time Transport Protocol (RTP), Real-Time Streaming Protocol (RTSP), Session Initiation Protocol (SIP) and their improvements, which 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 [Veeraraghavan and Weber, 2008] or using traffic prioritization with allocation of network resources in the nodes [Cao, Nguyen, and Nguyen, 2013].

On the other hand, image and video compression led to standards such as MJPEG, JPEG2000, MPEG-4, H.264 and more recently MPEG-H and H.265 that work by exploring redundant information within each image frame and in sequences of frames. However, these techniques frequently impose strong delays and thus a careful selection must be done for different application domains. While streaming of stored video can tolerate longer delays other domains impose more stringent limitations such as live streaming with augmented reality [Razavi, Fleury, and Ghanbari, 2008], surveillance [Genetec, 2010], industrial supervised multimedia control [Rinner and Wolf, 2008], multimedia embedded systems [Ramos, Panigrahi, and Dey, 2007], automated inspection [Kumar, 2008] and vehicle navigation [Lima and Victorino, 2016].

In these cases, image compression is frequently preferred to video compression for the lower latency incurred and lower memory and computing resource requirements. Nevertheless, any compression also incurs variability in transmission frame sizes that further complicates the matching with the instantaneous network conditions and has motivated substantial research into adaptive techniques [Rinner and Wolf, 2008; Ramos, Panigrahi, and Dey, 2007]. However, these works have essentially focused on adapting (single) streams to what the network provides, without protection against mutual interference. This protection can be achieved using network reservations (channels), as with Resource Reservation Protocol (RSVP) or lower layer real-time protocols. However, this has not been common due to high potential for poor network bandwidth efficiency. The work in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] 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 follow this line of work by improving over [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] in the way cameras adapt to their allocated bandwidth, namely using a PI feedback controller, and in the way the manager allocates bandwidth, using a game theoretic approach [Maggio, Bini, Chasparis, and

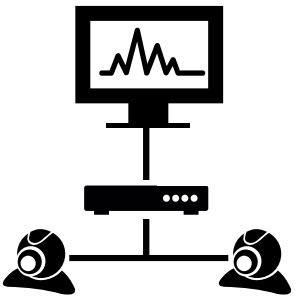


Figure 1. System architecture.

Årzén, 2013].

2. Model

The system is composed of a central node receiving video streams from a set of cameras, $\mathcal{C} = \{c_1, \ldots, c_n\}$, with cardinality n, $|\mathcal{C}| = n$. The central node also runs a resource manager $\mathcal{M}$, in charge of distributing the available network bandwidth $\mathcal{H}$. Figure 1 shows a system with a node that is in charge of being the network manager and the monitor for the cameras, a switch where bandwidth can be allocated and two cameras sharing the bandwidth.

2.1 The camera

This subsection describes the behavior of a camera c_p with $p \in \{1, \ldots, n\}$. The camera records a stream of frames. Each of these frames is encoded in an image, that is then sent to the central node via the network. The stream of images can be denoted with $I_p = \{i_{p,1}, \ldots, i_{p,m}\}$, where p is the camera identifier and m is the cardinality of the set of images (the longer the system runs, the more images each camera produces). Each element $i_{p,w}$ in the set, $w \in \{1, \ldots, m\}$, has the following characteristics.

The value of $q_{p,w}$ represents the *quality* used for the frame encoding. 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 in the encoding. The value of $s_{p,w}$ indicates the *size* of the encoded image. For each of the cameras, depending on the resolution used for the recording and on actual manufacturer parameters, the image size has a maximum and a minimum value, respectively denoted with $s_{p,\max}$ and $s_{p,\min}$, which we assume to be known. Finally, each camera transmits τ_p frames per second, a parameter in our implementation.

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, in principle, rather complex (see [Marau, Almeida, and Pedreiras, 2006; Silvestre-Blanes,

Almeida, Marau, and Pedreiras, 2011] for an early exponential model). It depends on many factors, including the complexity of the scene to encode, the sensor used by the camera manufacturer, the amount of light that reaches the sensor. In this work we approximate this relationship using the following affine model

$$s^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,\max} + \delta s_{p,w}, \tag{4.1}$$

where $\delta s_{p,w}$ represents a stochastic disturbance on the frame size which can be both positive and negative to capture more difficult or easier scenes to encode. This model is only a coarse-grained approximation of the camera behavior, the idea behind it being that, for control-purposes, the model needs to only capture the trend in the size behavior – increasing quality will more or less linearly increase the frame size, while decreasing quality will more or less linearly decrease the frame size. Linearity is also assumed in the model but not necessarily needed for the controller derivation, as the regulation adapts to the current operating conditions in an adaptive way using a normalized error as seen below. Assuming that a frame size has constraints, we then saturate the result to ensure that the actual size is between the minimum and the maximum size:

$$s_{p,w} = \max\{s_{p,\min}, \min\{s_{p,\max}, s^*_{p,w}\}\}. \tag{4.2}$$

The camera adapts its behavior, meaning that it automatically changes the quality $q_{p,w}$ to match the amount of network bandwidth that it can use. We denote with $B_{p,w}$ the amount of bandwidth that the p-th camera has available for the transmission of the w-th image at a given frame rate (the channel allows the transmission of a certain number of bytes per frame indicated with $B_{p,w}$). The camera then adjusts its quality parameter using an Adaptive Proportional and Integral (PI) controller

$$\begin{aligned} e_{p,w} &= \overbrace{\frac{B_{p,w-1} - s_{p,w-1}}{B_{p,w-1}}}^{\text{normalized error}} \\ q^*_{p,w} &= k_p \cdot e_{p,w} + k_i \cdot \sum_{t=1}^{w-1} e_{p,t} \end{aligned} \tag{4.3}$$

and saturating the result using the minimum and maximum quality values which are 15 and 85 respectively[1],

$$q_{p,w} = \max\{15, \min\{85, q^*_{p,w}\}\}. \tag{4.4}$$

The gains k_p and k_i are parameters of the controller inside the camera and determine how aggressive the adaptation is. Depending on their values, the system can be analyzed as a dynamical system and standard control theory can be used

[1] Ideally, the quality is a number between 1 and 100, but in our controller we impose saturations that are based on our prior experience with the equipment.

to prove that the value of $q_{p,w}$ converges to a specific value. Also, the same theory allows to prove that if the conditions of the scene change – for example including more artifacts that makes it more difficult to encode – the quality settles to a new value that allows the transmission of the frame, if such a quality value exists. We also implement an anti-windup mechanism in the controller, a standard practice for PI controllers [Åström and Hägglund, 1995].

2.2 The network manager

To determine how to distribute the network bandwidth we use the approach proposed in [Maggio, Bini, Chasparis, and Årzén, 2013] for CPU allocation and extend it to handle network bandwidth allocation. The network, particularly the link between the switch and the monitoring station, has a fixed capacity, which we denote $\mathcal{H}$. 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 in which the network manager is invoked, $\mathcal{M}$ selects a vector $b_{*,w}$, whose elements sum to one.

$$\begin{array}{ll} \forall\, t, \mathcal{M} & \text{selects } b_{*,t} = [b_{1,t}, \ldots, b_{n,t}] \\ & \text{such that } \sum_{p=1}^{n} b_{p,t} = 1 \end{array} \tag{4.5}$$

This means that each of the elements of the vector determines the fraction of the available bandwidth that is assigned to each video stream. Denoting the actual amount of bandwidth assigned by the resource manager to camera p at time t with $B_{p,t}$, this implies $B_{p,t} = b_{p,t} \cdot \mathcal{H}$. Knowing the frame rate τ_p used by camera p, then the bandwidth allocated by the network manager to that camera at time t in bytes per second ($B_{p,t}$) can be easily converted to the bandwidth (in bytes per frame) allocated to the transmission of frame w, ($B_{p,w}$) as follows: $B_{p,w} = \frac{B_{p,t}}{\tau_p}$.

The network manager is periodically triggered with period $\pi_{\mathcal{M}}$, a parameter in our implementation. Its first invocation, at time 0 equally divides the available bandwidth to the cameras. The following executions, happening at times $\{\pi_{\mathcal{M}}, 2\pi_{\mathcal{M}}, 3\pi_{\mathcal{M}}, \ldots\}$ assign the bandwidth based on the following relationship, from [Maggio, Bini, Chasparis, and Årzén, 2013], where the index t denotes the current time instant and $t+1$ the following one.

$$b_{p,t+1} = b_{p,t} + \varepsilon \cdot \{-\lambda_{p,t} \cdot f_{p,t} + b_{p,t} \cdot \sum_{i=1}^{n} [\lambda_{i,t} \cdot f_{i,t}]\} \tag{4.6}$$

Equation (4.6) introduces some terms. ε is a small constant and it is used to limit the change in bandwidth that is actuated for every step. Typical values are between 0.1 and 0.6. The choice of a suitable value for ε depends on the trade-off between the responsiveness of the controller (higher values making it converge faster, in principle, but also making it likely to have overshoots) and its robustness to disturbances (lower values delay convergence favoring a more stable behavior in the presence of transient disturbances). $\lambda_{p,t} \in (0,1)$ is a weight that denotes the fraction of adaptation that should be carried out by the resource

manager. A lower $\lambda_{p,t}$ value indicates that the resource manager is less willing to accommodate the needs of the p-th camera. The importance of this value lays in the relative difference between the values assigned to all the cameras. If all the cameras have an equal $\lambda_{p,t}$, the resource manager is not going to favor any of them. If one of the cameras has a higher value with respect to the others, the resource manager is "prioritizing" the needs of that camera over the others, and the changes will favor that specific camera. In the following, we will assume $\lambda_{p,t}$ to not change during execution, and use λ_p instead. A change in the value of λ_p has no impact in our analysis, and can be used to change the resource manager preference during runtime. $f_{p,t}$ is a function that we call the *matching function*, and 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. Denoting with w the index of the last transmitted frame at time t, and with t_w the time of transmission of the w-th frame, f_{p,t_w} determines a match between the quality $q_{p,w}$ and the bandwidth b_{p,t_w} available for the camera when the transmission of the w-th frame happens. For the analysis in [Maggio, Bini, Chasparis, and Årzén, 2013] to hold (therefore obtaining proof for the properties discussed in Section 2.3), the matching function should satisfy the following properties:

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

This basically means that the matching function should 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 should 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).

In our implementation we select the matching function to be a normalized version of the mismatch between the bandwidth allocated to the camera and the size of the frame produced by the camera, $e_{p,w}$ in Equation (4.3):

$$
f_{p,t_w} = \frac{B_{p,w} - s_{p,w}}{\underbrace{B_{p,w}}_{B_{p,w} = \frac{B_{p,t}}{\tau_p} = \frac{b_{p,t} \cdot \mathcal{H}}{\tau_p}}}. \tag{4.7}
$$

The so defined matching function automatically satisfies properties (P1a-c) and (P3a-b). If one assumes the disturbance $\delta s_{p,w}$ to be negligible, it is possible to

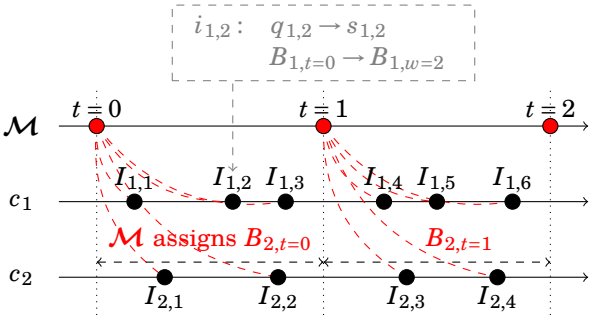


Figure 2. Example of system timeline.

use Equation (4.1) 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 2.1. In the following we will use $f_{p,t}$ to indicate the generic value of the matching function over time and f_{p,t_w} to indicate the precise value of the matching function computed for the frame w transmitted at time t_w. Figure 2 shows a timeline example. At time 0 the network manager decides the values of $b_{1,0}$ and $b_{2,0}$, which in turn assign a value to $B_{1,t=0}$ and $B_{2,t=0}$. Depending on the frame rates τ_1 and τ_2, the cameras have then a value for the amount of bandwidth that each frame should consume. In the case of the first camera, the choice of the resource manager determines $B_{1,w=1}$, $B_{1,w=2}$ and $B_{1,w=3}$. In the case of the second camera, only $B_{2,w=1}$ and $B_{2,w=2}$ are affected. At time $t=1$ the resource manager chooses a different allocation, affecting the frames in the next interval. For each frame, the cameras determine a quality, that in turn affects the frame size. For example, for $I_{1,2}$, the second image transmitted by the first camera, the quality $q_{1,2}$ determines the frame size $s_{1,2}$. Finally, the following quality $q_{1,3}$ is computed using the difference between the network bandwidth allocated to the frame $B_{1,w=2}$ and the size of the encoded frame $s_{1,2}$.

2.3 System's behavior

From a theoretical perspective, the resource allocation and camera adaptation scheme is not different from the CPU allocation and service level adjustment proposed in [Maggio, Bini, Chasparis, and Årzén, 2013; Chasparis, Maggio, Bini, and Årzén, 2016]. In both cases, there is one entity determining the resource allocation and other entities that can change their resource demands while being cooperative in trying to reach an agreeable resource distribution without unfairly favoring one entity. The behavior of the system has therefore been analyzed and some properties have been proven [Chasparis, Maggio, Bini, and Årzén, 2016]. 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, $\forall p$ such that $\lambda_p > 0, \forall t, b_{p,t} > 0$.

Balance. The balance property holds in case of overload conditions. The network

is overloaded when the capacity $\mathcal{H}$ is not enough to guarantee that all the cameras have a matching function greater or equal to zero $\forall p, \exists i, f_{p,i} \leq 0$, even when $\forall p, q_{p,\min}$. 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 unique 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 [Chasparis, Maggio, Bini, and Årzén, 2016, Theorem 4.1] and asynchronous [Chasparis, Maggio, Bini, and Årzén, 2016, Theorem 4.2] update.
Scalability. The average measured overhead for the computation of the bandwidth distribution in the resource manager is 2μs, for a network of two cameras. Despite this number being small, this is not the reason why we claim this approach has low overhead. One of the reasons why this resource allocation strategy is low-overhead 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 (4.6). This makes the system able to scale to a high number of cameras with limited impact.

3. Implementation and Setup

This section describes the underlying protocol which is used to transmit data, together with the acquisition and encoding of images. As shown in Figure 1, the implemented system consists of a network manager and monitor node, together with cameras connected over a Switched Ethernet local area network. The network manager oversees all activity on the network and implements the bandwidth allocation strategy described in Section 2.2. Since it acts as a monitor, it also receives the images transmitted by the connected cameras. It continually monitors the bandwidth consumption and apply bandwidth changes.

OpenCV. Each camera p captures an image w and encodes it using the given quality $q_{p,w}$ computed according to Equation (4.4) in in Section 2.1. The camera than transmits the image to the monitor node. We use OpenCV for image capture and modification, due to its pre-built open source libraries that implement different image processing functionality [Bradski, 2000]. Specifically, our implementation uses the `imencode` function, which takes an input image and encodes it using the `jpeg` format and a given compression ratio $c_{p,w}$, that we compute as $c_{p,w} = 100 - q_{p,w}$.

FTT-SE. To dynamically change the amount of bandwidth allocated, we need an underlying architecture that support bandwidth adaptation. For this, we use the Flexible Time Triggered (FTT) scheduling [Pedreiras and Almeida, 2003], which enforces adaptive hard reservations. In our implementation we use the Switched Ethernet (SE) implementation FTT-SE [Marau, Almeida, and Pedreiras, 2006]. We use the asynchronous communication scheme for the FTT-SE setup

and select a frame transmission period of 30ms, as done in prior work. 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 [Almeida et al., 2007].

Physical Setup. The following section describes experimental results obtained with our implementation. The three physical units used for the experiments form a multiple source single sink architecture. Each unit runs Fedora 21. The first unit has a Intel Core i7-4790, 8 core CPU with 32 GB RAM. It runs the network manager and monitor nodes, implemented as independent processes on the system. The other two units are two cameras, for which we use the commercial off-the-shelf cameras Logitech C270. The first camera was positioned to capture a scene with a lot of artifacts, like fast moving objects. The second camera captures a mostly static scene. We differentiate the scenes to simulate a scenario where cameras have different priorities and needs.

Implementation parameters. For the entire infrastructure the implementation parameters are the execution period $\pi_{\mathcal{M}}$, the total available network bandwidth $\mathcal{H}$ and the value of ε. For all of the experiments described in Section 4, $\pi_{\mathcal{M}}$ was set to 300 ms and ε was set to 0.15. In our setup, the available network bandwidth $\mathcal{H}$ is 4 Mbps. We deliberately set a low total bandwidth to stress the system and make sure that adaptation is needed. When two cameras are active, the amount of bandwidth is not enough to transmit the frames, unless the used compression is really high.

Assessment Criteria. We use three different criteria to assess the obtained solutions. The first criterion is the difference between the bandwidth allocated by the resource manager (`AllocBW`) and the one used by the cameras (`InstBW`). The second criteria is called Structural Similarity (SSIM) Index [Wang, Bovik, Sheikh, and Simoncelli, 2004]. The SSIM is a metric that represents the information loss from an original image to a transformed one. We use the SSIM to compare the original and encoded image, computing it offline to avoid runtime overhead. Finally, the third assessment criterion is the amount of frames dropped because the allocated bandwidth was not enough to transmit them. The camera captures an image and stores it in the buffer. During transmission if the camera is unable to transmit the whole frame in the allocated bandwidth, it is dropped. Notice that the system does not have enough bandwidth to transmit the full set of frames it records, therefore the optimal percentage of transmitted frames is not 100 (but varies depending on what the network bandwidth allows to achieve).

4. Experimental Validation

We conducted experiments to compare different allocation methodologies (ours and the state-of-the-art solution) and camera adaptation techniques. We recall that there are two adaptation levels: (a) the network manager decides how to dis-

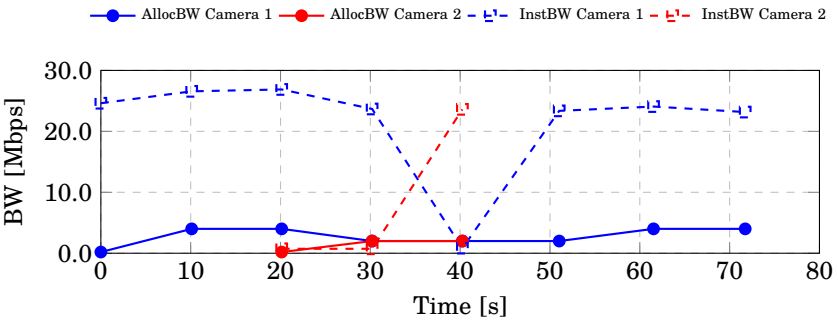


Figure 3. Results with Equal Distribution, without Camera Adaptation – Experiment 1.

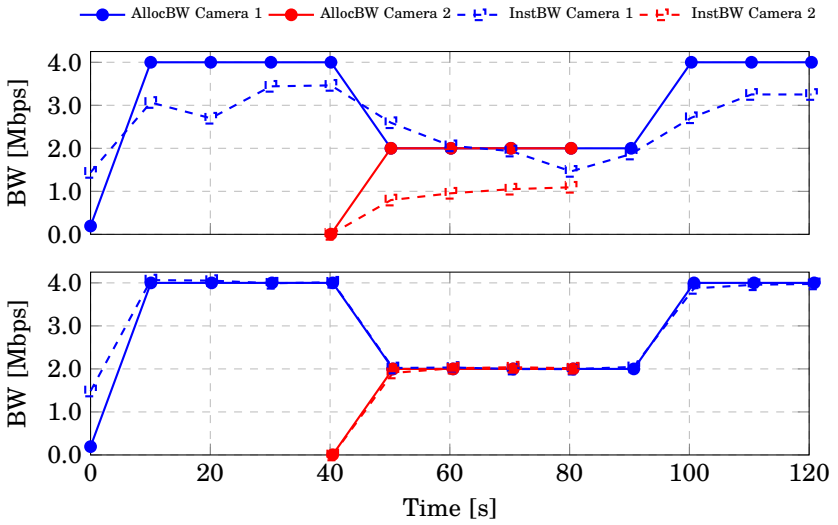


Figure 4. Results with Equal Distribution, and adaptation with [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] (top) and PI controller (bottom) – Experiment 2.

tribute bandwidth to the connected cameras, (b) the cameras adapt the encoding process to use the bandwidth.

Experiment 1: need to adjust. In this first experiment, the network manager distributes the bandwidth equally. The cameras do not apply any adaptation mechanism. We use this experiment as a baseline to test the system's operation. Figure 3 shows the experiment results. For the first 20 seconds, there is only one camera in the network, which receives all the bandwidth. A second camera joins

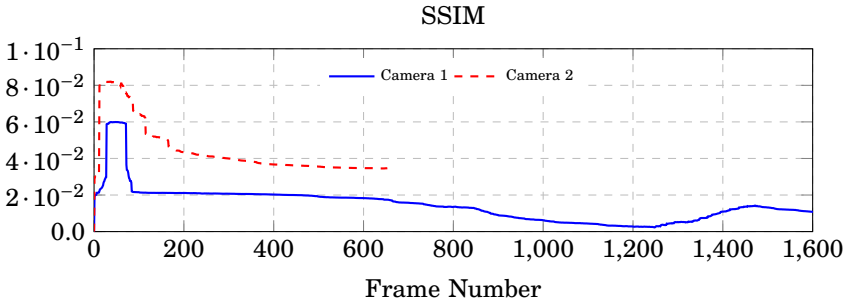


Figure 5. SSIM difference between PI controller and the strategy in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] – Experiment 2.

the network around 20 seconds after the start and is turned then off when the time is equal to 40 seconds. The manager reacts reducing the amount of bandwidth allocated to first camera and equally distributing network resources to the two cameras. The images produced by the cameras are too big and, in absence of any kind of adaptation, they are often not able to send the data – as reported in the first lines of Tables 1 and 2, only roughly 19% of the frames produced by Camera 1 and 12% of the frames produced by Camera 2 are transmitted.

Experiment 2: comparison with [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011]. The next experiment compares two alternatives for the camera adaptation strategies, using the same equal bandwidth allocation described for Experiment 1. As done for Experiment 1, Camera 1 joins the network immediately, while the second one enters at around 40 seconds from the start of the experiment. In both cases, the cameras attempt to match the size of the encoded images to the available bandwidth. Around 85 seconds from the experiment start, Camera 2 is shut down and releases its bandwidth, which is given to Camera 1.

In one case (the left plot in Figure 4), we use the model and adaptation strategy in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011], which fits the Variable Bit Rate (VBR) in the cameras to the given Constant Bit Rate (CBR) channel [Ding and Liu, 1996]. In the second case (the right plot in Figure 4), the camera uses the adaptive PI controller described in Section 2 with k_p is set to 10 and k_i is set to 1. We have tuned these parameters for the camera controller empirically according to standard control practice [Åström and Hägglund, 1995]. Compared to the model in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011], our camera is more efficient at using the bandwidth allocated by the network manager. Figure 5 shows the differences in the SSIM per frame. Both the cameras have a SSIM higher than 0. This indicates that quality of images captured in both the cameras is higher with the PI controller compared to the model in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011], making the PI controller a better choice. The amount of frames dropped is similar in both the runs, with the PI model allowing the cameras to transmit slightly more frames (another point in

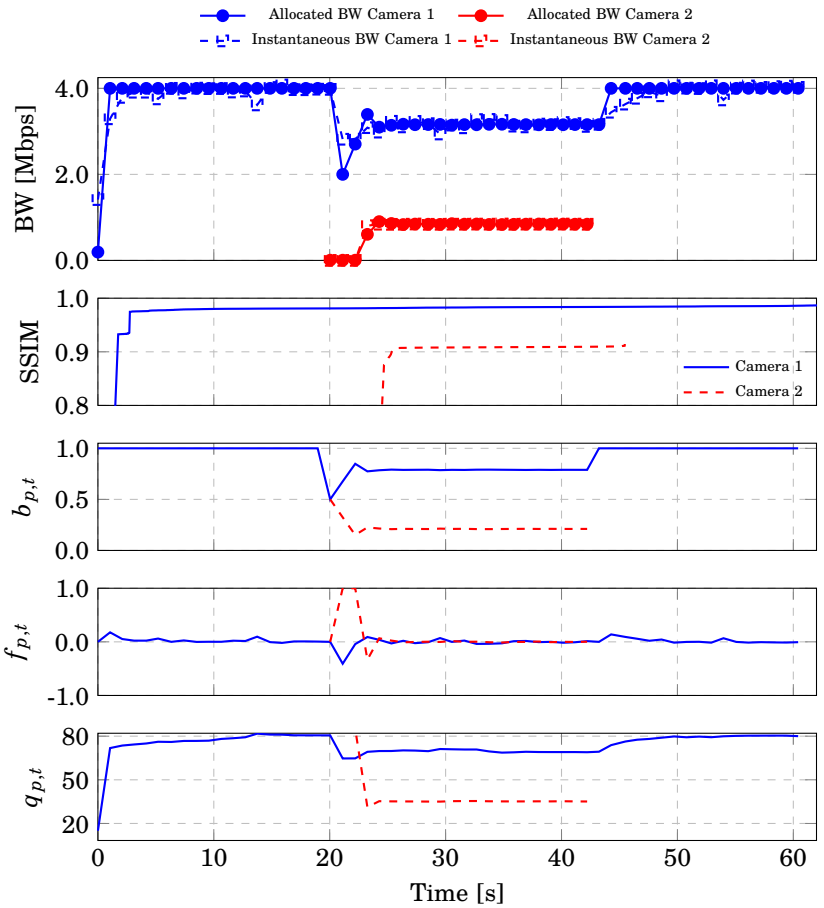


Figure 6. Results with Network manager and PI camera controller – Experiment 3.

favor) – see Tables 1 and 2.

Experiment 3: the full system. The last experiment incorporates the complete adaptation strategy. The network manager uses the game-theoretic approach to allocate bandwidth to the connected cameras and the cameras use the PI controller to ensure to fully take advantage of the given bandwidth. As a result, the frame that has a lot of artifacts (like Camera 1) and a very time-varying scene is given more network bandwidth. The resulting system is efficient in both allocation and utilization of the allocated bandwidth.

Figure 6 shows the allocation of bandwidth to the two cameras. The network bandwidth starts off by allocating most of the bandwidth available to Camera

Table 1. Statistics for Camera 1

Experiment No	No of Captured frames	No of Transmitted frames	Percentage of Transmitted Frames
1	1475	280	18.98
2 [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011]	3285	1641	49.95
2 [PI]	3667	1869	50.96
3	1735	805	46.39

Table 2. Statistics for Camera 2

Experiment No	No of Captured frames	No of Transmitted frames	Percentage of Transmitted Frames
1	517	62	11.99
2 [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011]	1005	593	59.00
2 [PI]	635	396	62.36
3	634	249	39.27

1. Around 25 seconds, Camera 2 is introduced. This causes the manager to reconfigure the network and allocate the bandwidth equally. Soon, the manager realizes that Camera 2 does not require as much bandwidth as Camera 1. Thus, the manager adjusts the allocation. Once Camera 2 leaves the network at around 62 seconds, Camera 1 receives the needed additional bandwidth. The Figure also shows the remaining properties of the experiment: the normalized bandwidth $b_{p,t}$ of the camera p at time t that the manager uses to calculate the amount of bandwidth to be allocated, the matching function, $f_{p,t}$ and the quality $q_{p,t}$ set by both cameras.

A negative value of $f_{p,t}$ indicates that the camera is starved and a positive value indicates that the camera has an abundance of bandwidth allocated. The optimal value of $f_{p,t}$ is zero. At every change in the network both the cameras react by changing their qualities and the resource manager by changing the allocation.

5. Conclusion

In this paper we have applied a CPU allocation strategy [Maggio, Bini, Chasparis, and Årzén, 2013] to the problem of network bandwidth allocation with a set of cameras competing for bandwidth. In this paper, we have shown that a resource manager acting periodically in the system is able to achieve some guarantees on

convergence, scalability and the general behavior of the system itself.

References

Almeida, L. et al. (2007). "Online qos adaptation with the flexible time-triggered (FTT) communication paradigm". In: Insup Lee Joseph Y-T. Leung, S. H. S. (Ed.). *Handbook of Real-Time and Embedded Systems*. CRC Press.

Åström, K. J. and T. Hägglund (1995). *PID Controllers: Theory, Design, and Tuning*. 2nd ed. Instrument Society of America, Research Triangle Park, NC.

Bradski, G. (2000). "The openCV library". *Doctor Dobbs Journal* **25**:11.

Cao, D. T., T. H. Nguyen, and L. G. Nguyen (2013). "Improving the video transmission quality over ip network". In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. DOI: 10.1109/ICUFN.2013.6614884.

Chasparis, G. C., M. Maggio, E. Bini, and K.-E. Årzén (2016). "Design and implementation of distributed resource management for time-sensitive applications". *Automatica* **64**, pp. 44–53. DOI: 10.1016/j.automatica.2015.09.015.

Communication, A. (2004). *White paper: digital video compression: review of the methodologies and standards to use for video transmission and storage*.

Ding, W. and B. Liu (1996). "Rate control of mpeg video coding and recording by rate-quantization modeling". *IEEE Transactions on Circuits and Systems for Video Technology* **6**:1, pp. 12–20. DOI: 10.1109/76.486416.

Genetec (2010). *White paper: three simple ways to optimize your bandwidth management in video surveillance*.

Kumar, A. (2008). "Computer-vision-based fabric defect detection: a survey". *IEEE Transactions on Industrial Electronics* **55**:1, pp. 348–363. DOI: 10.1109/TIE.1930.896476.

Lima, D. A. de and A. C. Victorino (2016). "A hybrid controller for vision-based navigation of autonomous vehicles in urban environments". *IEEE Transactions on Intelligent Transportation Systems* **17**:8, pp. 2310–2323. DOI: 10.1109/TITS.2016.2519329.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". In: *Euromicro Conference on Real-Time Systems*. DOI: 10.1109/ECRTS.2013.17.

Marau, R., L. Almeida, and P. Pedreiras (2006). "Enhancing real-time communication over cots Ethernet switches". In: *IEEE International Workshop on Factory Communication Systems*. DOI: 10.1109/WFCS.2006.1704170.

Pedreiras, P. and L. Almeida (2003). "The flexible time-triggered (FTT) paradigm: an approach to qos management in distributed real-time systems". In: *Proceedings International Parallel and Distributed Processing Symposium*. DOI: 10.1109/IPDPS.2003.1213243.

Ramos, N., D. Panigrahi, and S. Dey (2007). "Dynamic adaptation policies to improve quality of service of real-time multimedia applications in ieee 802.11e wlan networks". *Wirel. Netw.* **13**:4, pp. 511–535. DOI: 10.1007/s11276-006-9203-5.

Razavi, R., M. Fleury, and M. Ghanbari (2008). "Low-delay video control in a personal area network for augmented reality". *IET Image Processing* **2**:3. DOI: 10.1049/iet-ipr:20070183.

Rinner, B. and W. Wolf (2008). "An introduction to distributed smart cameras". *Proceedings of the IEEE* **96**:10. DOI: 10.1109/JPROC.2008.928742.

Silvestre-Blanes, J., L. Almeida, R. Marau, and P. Pedreiras (2011). "Online qos management for multimedia real-time transmission in industrial networks". *IEEE Transactions on Industrial Electronics* **58**:3. DOI: 10.1109/TIE.2010.2049711.

Vandalore, B., W.-c. Feng, R. Jain, and S. Fahmy (2001). "A survey of application layer techniques for adaptive streaming of multimedia". *Real-Time Imaging* **7**:3. DOI: 10.1006/rtim.2001.0224.

Veeraraghavan, V. and S. Weber (2008). "Fundamental tradeoffs in distributed algorithms for rate adaptive multimedia streams". *Comput. Netw.* **52**:6, pp. 1238–1251. DOI: 10.1016/j.comnet.2008.01.012.

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). "Image quality assessment: from error visibility to structural similarity". *IEEE Transactions on Image Processing* **13**:4. DOI: 10.1109/TIP.2003.819861.

Paper II

Event-Driven Bandwidth Allocation with Formal Guarantees for Camera Networks

Gautham Nayak Seetanadi Javier Cámara Luis Almeida

Karl-Erik Arzén Martina Maggio

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.

1. 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 [Pedreiras and Almeida, 2003; Almeida et al., 2007]. 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 [Heo and Abdelzaher, 2009].

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 [Seetanadi, Oliveira, Almeida, Arzen, and Maggio, 2017]. 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 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 [Chasparis, Maggio, Bini, and Årzén, 2016], 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.

2. 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, $\mathcal{C}_t = \{c_1, \ldots, c_n\}$, 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.

2.1 The camera

This subsection describes the behavior of the cameras where c_p with $p \in \{1, \ldots, n\}$ 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 $I_p = \{i_{p,1}, \ldots, i_{p,m}\}$, 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, $w \in \{1, \ldots, m\}$, has the following characteristics.

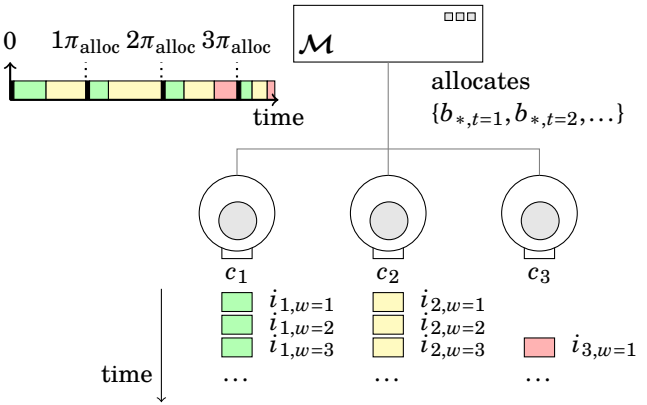


Figure 1. System architecture.

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 [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] 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

$$\hat{s}^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,\max} + \delta s_{p,w}, \tag{4.1}$$

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:

$$\hat{s}_{p,w} = \max\{s_{p,\min}, \min\{s_{p,\max}, \hat{s}^*_{p,w}\}\}. \tag{4.2}$$

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 [Wang, Chen, Huang,

Subramonian, Lu, and Gill, 2008][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 $B_{p,w}$.

$$\begin{aligned} e_{p,w} &= \overbrace{\frac{B_{p,w-1} - s_{p,w-1}}{B_{p,w-1}}}^{\text{normalized error}} \\ q^*_{p,w} &= k_{p,\text{Proportional}} \cdot e_{p,w} + k_{i,\text{Integral}} \cdot \sum_{t=1}^{w-1} e_{p,t} \end{aligned} \tag{4.3}$$

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],

$$q_{p,w} = \max\{q_{\min}, \min\{q_{\max}, q^*_{p,w}\}\}. \tag{4.4}$$

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.

2.2 The network manager

To determine how to distribute the network bandwidth we use the approach proposed in [Maggio, Bini, Chasparis, and Årzén, 2013] 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.

$$\forall t, \mathcal{M} \quad \text{selects} \quad b_{*,t} = [b_{1,t}, \cdots, b_{n,t}], \quad \text{such that} \sum_{p=1}^{n} b_{p,t} = 1 \tag{4.5}$$

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.

[1] Given the model in Equation (4.2), 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 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 $B_{p,w}$.

$$B_{p,w} = b_{p,t_{\mathcal{M},w}} \cdot \pi_{\text{alloc}} \cdot \mathcal{H} \tag{4.6}$$

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 [Maggio, Bini, Chasparis, and Årzén, 2013], where the index t denotes the current time instant and $t+1$ the following one.

$$b_{p,t+1} = b_{p,t} + \varepsilon \cdot \{-\lambda_{p,t} \cdot f_{p,t} + b_{p,t} \cdot \sum_{i=1}^{n} [\lambda_{i,t} \cdot f_{i,t}]\} \tag{4.7}$$

Equation (4.7) 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

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

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 [Maggio, Bini, Chasparis, and Årzén, 2013] 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 (4.3) as the matching function. For the analysis in [Maggio, Bini, Chasparis, and Årzén, 2013] to hold – which proves properties such as *starvation avoidance*, *balance*, *convergence*, and *stability*, discussed in Section 2.3 –, the matching function should satisfy the following properties:

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

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 (4.3) – 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 (4.1) 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 2.1. 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 the sequence of quality per frame chosen by the camera $\langle q_{p,w=0}, q_{p,w=1}, \ldots \rangle$. An example timeline can be seen in Appendix A.

2.3 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 [Maggio, Bini, Chasparis, and Årzén, 2013]. The behavior of the system has therefore been analyzed and some properties have been proven [Chas-

paris, Maggio, Bini, and Årzén, 2016]. 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) [Chasparis, Maggio, Bini, and Årzén, 2016, Theorem 4.1] and asynchronous updates (the cameras update their quality potentially at different times) [Chasparis, Maggio, Bini, and Årzén, 2016, 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 (4.7). 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.

3. Towards event-triggered activation

As briefly summarized in Section 2.3, 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.

3.1 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.

3.2 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

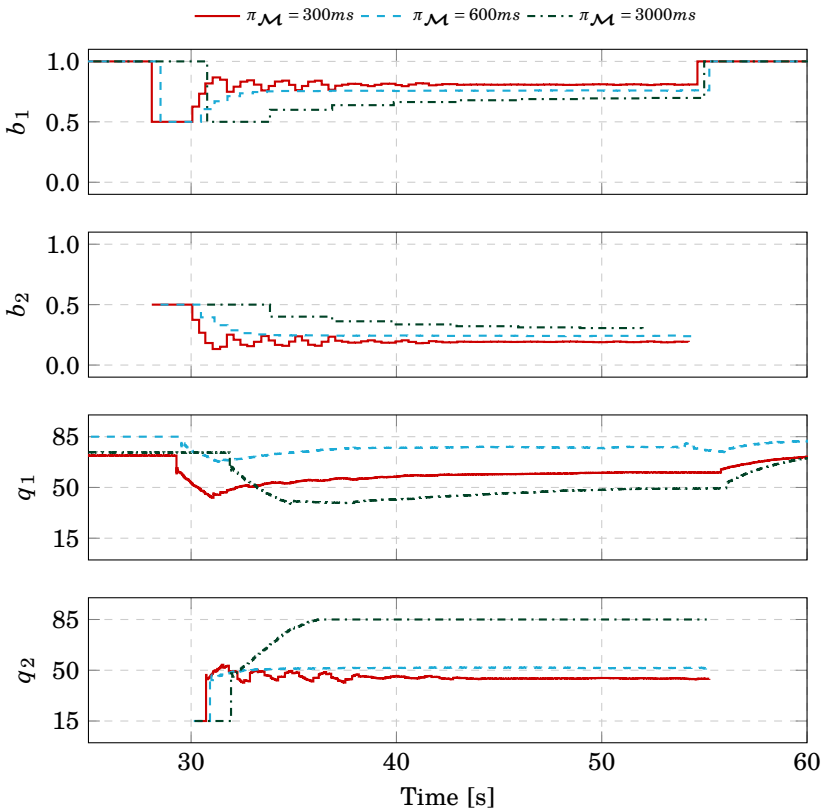


Figure 2. Time-triggered activation with different network manager periods: bandwidth allocation (b_1 and b_2) and image quality (q_1 and q_2).

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 (4.3), (ii) compute the new resource distribution vector $b_{*,t}$ according to Equation (4.7), 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 2.3, 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) [Maggio, Bini, Chasparis, and Årzén, 2013]. 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 3.1 demonstrated that a larger activation period is not always a viable alternative. This motivates our investigation of an event-triggered activation scheme.

4. 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

[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 ($t \bmod \pi_{\text{alloc}} = 0$).

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 (4.7).

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 (4.3). 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 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 [Heemels, Johansson, and Tabuada, 2012; Wang and Lemmon, 2011]. 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 2.3 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.

5. 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 5.1.

Then we introduce the model of the system in Section 5.2 and the properties we prove in Section 5.3. Finally, we discuss an optimal network manager activation strategy generated by our model checker in Section 5.4.

5.1 Background on model checking

Probabilistic Model Checking (PMC) [Kwiatkowska, Norman, and Parker, 2007] 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) [Andova, Hermanns, and Katoen, 2003], which state that some probability or reward meet some threshold.

An example of a probability-based PRCTL property is $P_{\geq 1}[G\ \mathcal{H}_{alloc} = \mathcal{H}]$, 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 $R^{r}_{min=?}[F\ \phi]$ 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 $R^{nmi}_{min=?}[F\ t = t_{max}]$, meaning "find a strategy that minimizes the number of total network manager interventions (captured in reward nmi) throughout a system execution period $(0, t_{max})$, *i.e.*, when time is equal to t_{max}."

In this section, we illustrate probabilistic modeling of the camera network system using the high-level language of the probabilistic model checker PRISM [Kwiatkowska, Norman, and Parker, 2011], in which DTMCs and MDPs can be expressed as processes formed by sets of commands like the following

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

[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 [Clarke and Emerson, 1982] or LTL [Gerth, Peled, Vardi, and Wolper, 1996].

[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.

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.

5.2 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 4.1.

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 in this case as a penalty for every dropped frame and for each network manager intervention (cf. Section 5.4).

```
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
```

[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.

```
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 4.1. 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).

5.3 Formal guarantees

With respect to the properties discussed in Section 2.3, *starvation avoidance* and *balance* depend on how Equation 4.7 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 $P_{\geq 1}[F(G!(any\_change\_event))]$, where any_change_event is defined in our formal model as the disjunction $nmchange \vee c1change \vee \ldots \vee cnchange$. 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 $max\_frames = 30$ (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 2.3, 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 $P_{\geq 1}[G\ used\_bw = max\_bw]$, 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 4.3, 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 $R^{nm\_calls}_{max=?}[F\ rounds = max\_frames]$, 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 4.3, line 11).

5.4 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 4.1, 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 $penalty\_frames \cdot (drop1 + \ldots + dropn)$ and $penalty\_intervention \cdot (nm\_interventions)$, where $dropx$ is the number of dropped frames for camera x at the end of the execution (Listing 4.2, 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 $R^{total\_cost}_{min=?}[F\ rounds = max\_frames]$, 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 $\langle best\_dont, best\_dont, best\_do, best\_do \rangle$ 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 5.3. 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 5.3.

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.

6. 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 [Pedreiras and Almeida, 2003], which enforces adaptive hard reservations. In our implementation we use the Switched Ethernet (SE) implementation FTT-SE [Marau, Almeida, and Pedreiras, 2006]. 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 [Almeida et al., 2007].

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

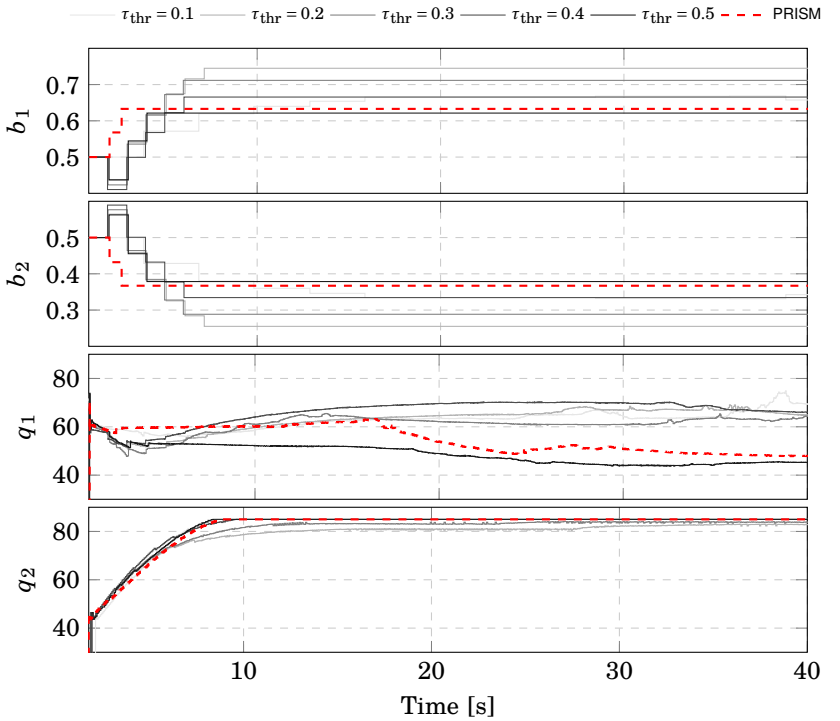


Figure 3. Event-triggered activation with $\tau_{thr} \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the optimal strategy synthesized by PRISM: bandwidth allocation (b_1 and b_2) and image quality (q_1 and q_2).

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 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 $\tau_{thr} = 0.3$ minimizes total cost. When the system runs for more time, however, the cameras will record images

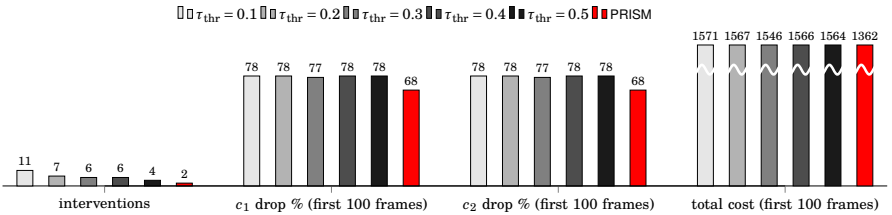


Figure 4. Aggregate metrics for the experiment: resource manager interventions, dropped frames and cost.

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.

7. 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 [Vandalore, Feng, Jain, and Fahmy, 2001; Communication, 2004; Rinner and Wolf, 2008; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008; Toka, Lajtha, Hosszu, Formanek, Géhberger, and Tapolcai, 2017; Zhang, Chowdhery, Bahl, Jamieson, and Banerjee, 2015], 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 [Veeraraghavan and Weber, 2008] or using traffic prioritization [Cao, Nguyen, and Nguyen, 2013].

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 [Razavi, Fleury, and Ghanbari, 2008], industrial supervised multimedia control [Rinner and Wolf, 2008], multimedia embedded systems [Ramos, Panigrahi, and Dey, 2007], automated inspection [Kumar, 2008] and vehicle navigation [Lima and Victorino, 2016] – 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 [Rinner and Wolf, 2008; Ramos, Panigrahi, and Dey, 2007; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008]. 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 [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] 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 [Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008]. 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 [Toka, Lajtha, Hosszu, Formanek, Géhberger, and Tapolcai, 2017] 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 [Zhang, Chowdhery, Bahl, Jamieson, and Banerjee, 2015], 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 [Nagaoka, Ito, Okano, and Kusumoto, 2011], 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 [Hanh and Van Hung, 2007]. 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 [Moreno, Cámara, Garlan, and Schmerl, 2015]. 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.

8. 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

Almeida, L. et al. (2007). "Online qos adaptation with the flexible time-triggered (FTT) communication paradigm". In: Insup Lee Joseph Y-T. Leung, S. H. S. (Ed.). *Handbook of Real-Time and Embedded Systems*. CRC Press.

Andova, S., H. Hermanns, and J.-P. Katoen (2003). "Discrete-time rewards model-checked". In: *1st International Workshop Formal Modeling and Analysis of Timed Systems*. DOI: 10.1007/978-3-540-40903-8_8.

Cao, D. T., T. H. Nguyen, and L. G. Nguyen (2013). "Improving the video transmission quality over ip network". In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. DOI: 10.1109/ICUFN.2013.6614884.

Chasparis, G. C., M. Maggio, E. Bini, and K.-E. Årzén (2016). "Design and implementation of distributed resource management for time-sensitive applications". *Automatica* **64**, pp. 44–53. DOI: 10.1016/j.automatica.2015.09.015.

Clarke, E. M. and E. A. Emerson (1982). "Design and synthesis of synchronization skeletons using branching-time temporal logic". In: *Logic of Programs*. DOI: 10.1007/BFb0025774.

Communication, A. (2004). *White paper: digital video compression: review of the methodologies and standards to use for video transmission and storage*.

Gerth, R., D. Peled, M. Vardi, and P. Wolper (1996). "Simple on-the-fly automatic verification of linear temporal logic". In: *International Symposium on Protocol Specification, Testing and Verification*. DOI: 10.1007/978-0-387-34892-6_1.

Hanh, T. and D. Van Hung (2007). *Verification of an Air-Traffic Control System with Probabilistic Real-time Model-checking*. Tech. rep. UNU-IIST United Nations University International Institute for Software Technology.

Heemels, W., K. Johansson, and P. Tabuada (2012). "An introduction to event-triggered and self-triggered control". In: *IEEE Conference on Decision and Control*.

Heo, J. and T. Abdelzaher (2009). "Adaptguard: guarding adaptive systems from instability". In: *6th ACM International Conference on Autonomic Computing*. DOI: 10.1145/1555228.1555256.

Kumar, A. (2008). "Computer-vision-based fabric defect detection: a survey". *IEEE Transactions on Industrial Electronics* **55**:1, pp. 348–363. DOI: 10.1109/TIE.1930.896476.

Kwiatkowska, M., G. Norman, and D. Parker (2007). "Stochastic model checking". In: *7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. DOI: 10.1007/978-3-540-72522-0_6.

Kwiatkowska, M., G. Norman, and D. Parker (2011). "PRISM 4.0: verification of probabilistic real-time systems". In: *23rd International Conference on Computer Aided Verification*. DOI: 10.1007/978-3-642-22110-1_47.

Lima, D. A. de and A. C. Victorino (2016). "A hybrid controller for vision-based navigation of autonomous vehicles in urban environments". *IEEE Transactions on Intelligent Transportation Systems* **17**:8, pp. 2310–2323. DOI: 10.1109/TITS.2016.2519329.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". In: *Euromicro Conference on Real-Time Systems*. DOI: 10.1109/ECRTS.2013.17.

Marau, R., L. Almeida, and P. Pedreiras (2006). "Enhancing real-time communication over cots Ethernet switches". In: *IEEE International Workshop on Factory Communication Systems*. DOI: 10.1109/WFCS.2006.1704170.

Moreno, G., J. Cámara, D. Garlan, and B. Schmerl (2015). "Proactive self-adaptation under uncertainty: a probabilistic model checking approach". In: *10th ACM/SIGSOFT Joint Meeting on Foundations of Software Engineering*. DOI: 10.1145/2786805.2786853.

Nagaoka, T., A. Ito, K. Okano, and S. Kusumoto (2011). "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. DOI: 10.1587/transinf.E94.D.958.

Pedreiras, P. and L. Almeida (2003). "The flexible time-triggered (FTT) paradigm: an approach to qos management in distributed real-time systems". In: *Proceedings International Parallel and Distributed Processing Symposium*. DOI: 10.1109/IPDPS.2003.1213243.

Ramos, N., D. Panigrahi, and S. Dey (2007). "Dynamic adaptation policies to improve quality of service of real-time multimedia applications in ieee 802.11e wlan networks". *Wirel. Netw.* **13**:4, pp. 511–535. DOI: 10.1007/s11276-006-9203-5.

Razavi, R., M. Fleury, and M. Ghanbari (2008). "Low-delay video control in a personal area network for augmented reality". *IET Image Processing* **2**:3. DOI: 10.1049/iet-ipr:20070183.

Rinner, B. and W. Wolf (2008). "An introduction to distributed smart cameras". *Proceedings of the IEEE* **96**:10. DOI: 10.1109/JPROC.2008.928742.

Seetanadi, G. N., L. Oliveira, L. Almeida, K.-E. Arzen, and M. Maggio (2017). "Game-theoretic network bandwidth distribution for self-adaptive cameras". In: *15th International Workshop on Real-Time Networks*. DOI: 10.1145/3267419.3267424.

Silvestre-Blanes, J., L. Almeida, R. Marau, and P. Pedreiras (2011). "Online qos management for multimedia real-time transmission in industrial networks". *IEEE Transactions on Industrial Electronics* **58**:3. DOI: 10.1109/TIE.2010.2049711.

Toka, L., A. Lajtha, É. Hosszu, B. Formanek, D. Géhberger, and J. Tapolcai (2017). "A resource-aware and time-critical IoT framework". In: *IEEE International Conference on Computer Communications INFOCOM*. DOI: 10.1109/INFOCOM.2017.8057143.

Vandalore, B., W.-c. Feng, R. Jain, and S. Fahmy (2001). "A survey of application layer techniques for adaptive streaming of multimedia". *Real-Time Imaging* **7**:3. DOI: 10.1006/rtim.2001.0224.

Veeraraghavan, V. and S. Weber (2008). "Fundamental tradeoffs in distributed algorithms for rate adaptive multimedia streams". *Comput. Netw.* **52**:6, pp. 1238–1251. DOI: 10.1016/j.comnet.2008.01.012.

Wang, X., M. Chen, H. M. Huang, V. Subramonian, C. Lu, and C. D. Gill (2008). "Control-based adaptive middleware for real-time image transmission over bandwidth-constrained networks". *IEEE Transactions on Parallel and Distributed Systems* **19**:6. DOI: 10.1109/TPDS.2008.41.

Wang, X. and M. Lemmon (2011). "Event-triggering in distributed networked control systems". *IEEE Transactions on Automatic Control* **56**:3. DOI: 10.1109/TAC.2010.2057951.

Zhang, T., A. Chowdhery, P. (Bahl, K. Jamieson, and S. Banerjee (2015). "The design and implementation of a wireless video surveillance system". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438. DOI: 10.1145/2789168.2790123.

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.

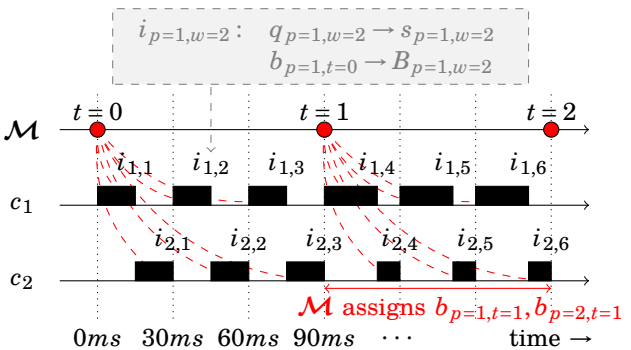


Figure 5. Example of system timeline.

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 $s_{1,2}$.

B. Model of camera and network manager behavior

Listing 4.2 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 (4.2), the update of the quality in the controller according to Equation (4.4), and the calculation of the normalized error, or matching function, as the term $e_{1,w}$ in Equation (4.3).

```
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 (4.2)
10 formula update_q1 = ... // control action according to equation (4.3) and (4.4)
11 formula f1 = ... // matching function e_{1,w} according to equation (4.3)
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);
25       // check threshold
26 endmodule
```

Listing 4.2. 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 5.3), 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 (4.7)
2  formula update_bw2 = ... // update bw according to equation (4.7)
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 4.3. Network manager module description

The code for the network manager (Listing 4.3) 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).

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 2 and 3, one of the advantages of using the allocation policy proposed in [Maggio, Bini, Chasparis, and Årzén, 2013] 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 (4.7), (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.

D. Additional results

This appendix presents additional experiments that assess the performance of

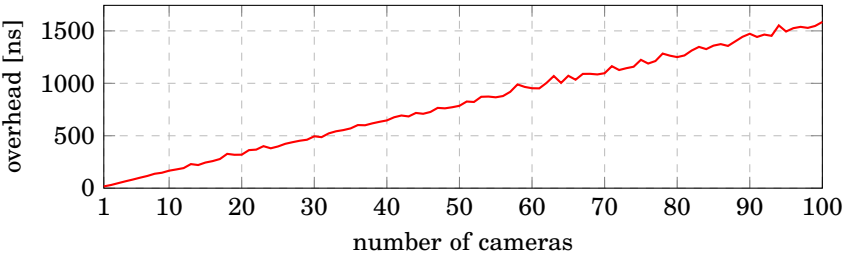


Figure 6. Average overhead measured for the computation of the network allocation.

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 2, and with the event-triggered network manager described in Section 4, 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 5 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 1 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 5.4. 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.

To confirm the validity of these results, we executed each experiment 10

Table 1. 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

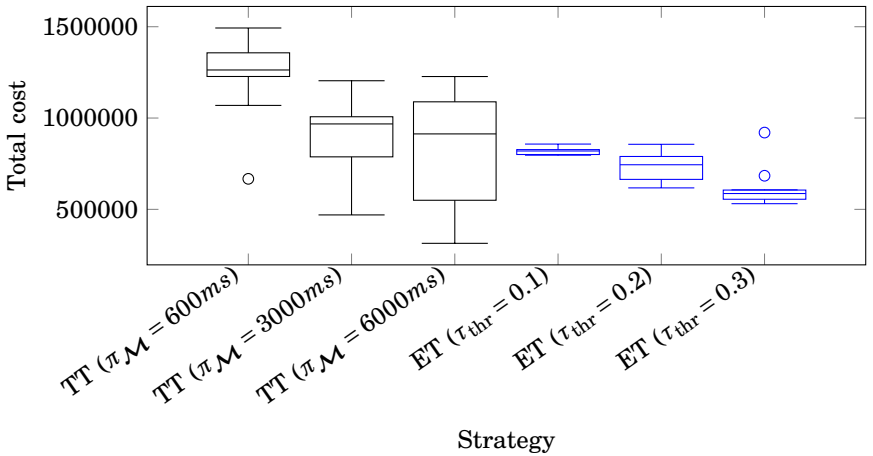


Figure 7. Total cost box plot for 10 1-hour long executions traces.

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 1. 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.

Paper III

Control-Based Event-Driven Bandwidth Allocation Scheme for Video-Surveillance Systems

Gautham Nayak Seetanadi Karl-Erik Årzén Martina Maggio

Abstract

Modern computer systems consist of large number of entities connected through a shared resource. One such system is a video surveillance network consisting of a set of cameras and a network manager. The surveillance cameras capture a stream of images and transmit them to the manager over a shared constrained network. The central manager allocates bandwidth to the cameras in a fair manner using a threshold based game-theoretic approach. The cameras regulate their bandwidth using a *quality* factor to fit sizes of the generated frames to the bandwidth allocated to the manager. The presence of these multiple control loops that interact with each other leads to complexity in providing performance and safety guarantees.

Our previous work [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] explored performance of the event-based manager using model checking to verify relevant properties. However, the paper only evaluated linear models of camera behavior without the presence of disturbance that arise during image capture and encoding. In this paper we build on our previous work by verifying complex camera models that capture uncertainties during image capture. We model the uncertainties using probabilistic Markov Decision Processes (MDPs) and verify relevant properties of the system. We also evaluate system performance for different system parameters with varying triggering thresholds, showing the advantage of model checking for safe and informed parameter selection. Finally, we evaluate the effect of varying thresholds on manager interventions by capturing images on a commercial off-the-shelf (COTS) camera test-bed.

Journal Manuscript under Review.

1. Introduction

Large number of interconnected devices in complex computational infrastructures has increased the need to share common system resources (e.g., CPU, network bandwidth, storage). Sharing resources leads to contention within the connected devices for access to the constrained resource. In these constrained dynamical systems, performance problems are mitigated mainly through two methods. The first is through over-provisioning of the constrained resource. This however is not an economical solution due to resource under-utilization. Over-provisioning is also not possible in systems without accurate knowledge about resource requirements. The second preferred method is by employing entities that can adapt their utilization to use only a fraction of the constrained resource. But this scenario where the entities adjust their requirements in the presence of multiple independent loops can cause disruptive effects and lead to system instability [Heo and Abdelzaher, 2009].

This paper considers a video-surveillance system that is composed of a network manager and cameras using a shared communication channel. The cameras capture a stream of images and encode each captured image using a quality factor which in turn regulates its bandwidth utilization. The frame is then transmitted over the network to a central node, the network manager.

The network manager is a decision node that allocates a fraction of total available bandwidth to all the cameras connected in the network. The manager ensures that the bandwidth allocated to the cameras in a fair manner based upon the errors between the bandwidth allocated by it and the actual bandwidth utilization by the cameras. The amount of allocated bandwidth allocated is calculated either periodically or only when required due to larger changes in the images captured by the cameras.

The periodically triggered manager is efficient and predictable, but has the drawback of unnecessary communication and non-trivial triggering period selection. If the period is too large then the manager is slow to react to changes in the camera bandwidth requirements. On the other hand a small period causes large communication overhead due to the number of control messages sent to the cameras. The event based manager is triggered based on events related to the scene being captured by the cameras. This minimizes network reconfiguration, reducing calculation overhead but complicates guarantees provided on the behavior of the system.

The cameras in the network are adaptive and regulate their bandwidth utilization by encoding the images. Images captured directly by the camera are large in size and unsuitable for direct transmission. The size of images after encoding them is dependent on the encoding technique used and the amount of compression decided by the quality factor. Image quality denotes the amount of information retained in the image after encoding with a higher quality generally leading to a clearer image but with a large frame size. The resulting frame size is also dependent on the artifacts in the images captured, arising due to lighting, movement, nature among others [Edpalm, Martins, Maggio, and Årzén, 2018] [Ed-

palm, Martins, Årzén, and Maggio, 2018] [Ding and Liu, 1996], leading to different sizes for images encoded with the same quality factor.

The co-presence of these multiple, event-based and time-triggered control strategies can affect the performance and stability of the camera system. Model checking has shown promise in verification of such complex systems with entities displaying stochastic behaviors. It has already been applied successfully to validate and prove relevant properties of real-time systems [Nagaoka, Ito, Okano, and Kusumoto, 2011; Hanh and Van Hung, 2007]. Applying it to the camera network model allows us to prove properties like stability, convergence, number of manager interventions and number of dropped frames [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017].

We use a probabilistic model checker PRISM [Kwiatkowska, Norman, and Parker, 2011], to verify different properties of the system by including probabilistic disturbances in the camera model. The complex models used for verification lead to state space explosion problem [Valmari, 1998] due to the presence of complex models. We also show that using Statistical Model Checking (SMC) helps to reduce the impact of state-space explosion at the expense of relaxed guarantees on system performance.

The contributions of this paper are as follows:

- Realistic camera model implementation by incorporating dynamics that arise during image capture
- Comparison of system behavior with varying camera parameters for model with disturbances
- Evaluation of statistical model checking of the camera network with relaxed guarantees
- Expanded experiments on the real camera network with larger number of cameras for larger number of triggering thresholds

This paper expands on the results from [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] in several directions, both from the model checking (theoretical) perspective and from the practical (implementation) standpoint. On the theoretical side, We verify properties of probabilistic camera models using both classical model checking and statistical model checking. On the experimental evaluation side, we analyze system performance for a larger number of triggering thresholds of the event based manager. Finally, the paper also analyses the effect of threshold selection on the number of manager interventions and frames dropped by the cameras.

Paper Organization: First we introduce the camera network architecture in Section 2 and discuss the behavior of the cameras. The section also describes the two different triggering strategies of the network manager. Section 3 gives an introduction to model checking and describes the different types of model verification. Section 4 describes the implementation of the camera network in PRISM [Kwiatkowska, Norman, and Parker, 2011], the model checking tool used

in this paper. Sections 5 and 6 discuss the results obtained using model checking and the camera test-bed respectively. Finally, we discuss related work in 7 and conclude the paper in Section 8.

2. System Model

This section describes the architecture and modeling of the camera surveillance network. We also introduce the notation used in the remainder of the paper.

2.1 System Architecture

The system consists of a set of cameras connected to a central manager via Ethernet networking. At each time instant t, we define a set of active cameras $\mathcal{C}_t = \{c_1, \ldots, c_n\}$ where n is the number of active cameras. Each camera in the network captures a stream of images, encodes, and transmits them to the network manager. The camera encodes each image ensuring that the resulting frame size is less than the bandwidth allocated by the network manager. The manager allocates bandwidth to all the cameras in the network at each instant it is invoked. Bandwidth allocation to all the cameras is the network is recalculated by evaluating the camera requirements and its relative error compared to other cameras in the network.

The architecture enables the use of two independent strategies to achieve a good match between the resource allocated to each camera and their resource consumption:.

- **Image fitting at camera p:** To fit a frame w to the allocated bandwidth, $B_{p,w}$, by choosing the appropriate image quality, $q_{p,w}$.

- **Bandwidth allocation at the manager:** The amount of bandwidth to allocate to each of the cameras while ensuring the bandwidth does not exceed the total amount of available bandwidth $\mathcal{H}$.

Figure 1 shows an example network with 4 cameras, (c_1, c_2, c_3, c_4) connected to a network manager through a router. The unidirectional blue arrows denote the control signals sent from the manager to the cameras. The control signals contain information about the amount of bandwidth allocated to each cameras and their appropriate communication slot. The black arrows denote images sent from the cameras to the network manager over the shared Ethernet connection.

The manager enforces bandwidth restrictions for each camera using flexible time-triggered Switched Ethernet (FTT-SE) [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011]. FTT-SE dynamically allocates Ethernet bandwidth to all the cameras independently and frames that are larger in size compared to the instantaneous allocated bandwidth are dropped.

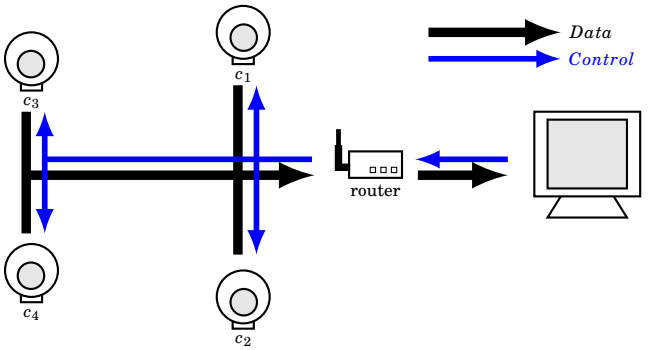


Figure 1. System Overview

2.2 Camera Behaviour

The cameras in the network capture a set of frames and encode them using the chosen encoding technique (In this paper we use Motion JPEG(MJPEG)). The images are then transmitted to the manager over a common Ethernet connection during each transmission period. Figure 2 shows an original image and an image after encoding. The original frames captured by the camera consist of redundant information and are unsuitable for direct transmission. Most encoding techniques use a quality factor $q_{p,w}$ to encode images and reduce the amount of redundant information contained. Higher values of $q_{p,w}$ leads to frames of larger sizes with fewer artifacts. Motion JPEG (MJPEG) generally has quality factor bounded such that $q_{p,w} \in (0,100]$.

The camera generates a frame of size $s^*_{p,w}$ as shown in Equation 4.1. $s_{p,\max}$ is the estimated maximum frame size that is dependent on the number of pixels in the image. The frame size is then saturated to set global limits for the maximum and minimum frame sizes. $\delta s_{p,w}$ indicates the stochastic disturbance that occurs due to changes in the scene being captured such as movement, illumination, number of people in frame to name a few.

$$\hat{s}^*_{p,w} = 0.01 \cdot q_{p,w} \cdot s_{p,\max} + \delta s_{p,w}, \tag{4.1}$$

$$\hat{s}_{p,w} = \max\{s_{p,\min}, \min\{s_{p,\max}, \hat{s}^*_{p,w}\}\}. \tag{4.2}$$

The camera frame size controller is synthesized using Equation 4.2. The frame size $s_{p,w}$ is determined by regulating the quality $q_{p,w}$ with a simple PI controller. The controller minimizes the error $e_{p,w}$, between the allocated bandwidth $B_{p,w}$, and the resulting frame-size $s_{p,w}$ as shown in Equation 4.3. $e_{p,w}$ is then normalized, limiting the error between -1 and 1 in most operating conditions.

This control design achieves a zero-steady state error (when $\delta = 0$) ensuring that the size of the encoded frame is equal to the amount of bandwidth allocated by the manager. The quality parameter $q_{p,w}$ is the control signal for the camera

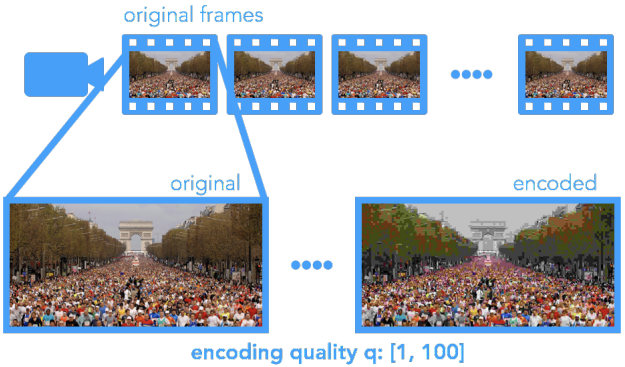


Figure 2. Camera Encoding

controller and it roughly corresponds to a compression level of the frame. The values of k_p and k_i are the proportional and integral gains of the camera controller respectively.

$$e_{p,w} = \frac{\overbrace{B_{p,w-1} - s_{p,w-1}}^{\text{normalized error}}}{B_{p,w-1}} \tag{4.3}$$

$$q^*_{p,w} = k_p \cdot e_{p,w} + k_i \cdot \sum_{t=1}^{w-1} e_{p,t} \tag{4.4}$$

$$q_{p,w} = \max\{q_{\min}, \min\{q_{\max}, q^*_{p,w}\}\}. \tag{4.5}$$

2.3 Manager Behaviour

The network manager $\mathcal{M}$ acts on a global scale deciding the optimal bandwidth to allocate to the cameras $\mathcal{C}_t$ connected in the network. The manager also ensures that the total allocated bandwidth does not exceed the fixed capacity $\mathcal{H}$ of the network. $\mathcal{M}$ selects the bandwidth allocation vector $b_{*,w}$ at each instant t that it is invoked such that,

$$\begin{aligned} \forall\, t, \mathcal{M} \quad & \text{selects } b_{*,t} = [b_{1,t}, \ldots, b_{n,t}] \\ & \text{such that } \textstyle\sum_{p=1}^{n} b_{p,t} = 1 \end{aligned} \tag{4.6}$$

Each element of $b_{*,t}$ determines the fraction of the bandwidth assigned to the respective camera in the network. The bandwidth assignment is done periodically during each allocation period.

Equations 4.7 and 4.8 form the core of calculations in the manager. $B_{p,w}$ denotes the amount of bandwidth allocated to camera p for frame w which is a function of the $b_{p,t_{\mathcal{M},w}}$, the fraction of bandwidth (between 0 and 1).

Table 1. Explanation of camera terms

Term	Explanation
c_p	Camera number p
$s_{p,w}$	Size of frame w captured by camera p
$q_{p,w}$	Quality of frame w captured by camera p
$\delta s_{p,w}$	Stochastic disturbance on frame w
$e_{p,w}$	Error during tx of frame w of camera p
$B_{p,w}$	Amount of bandwidth allocated to camera p for frame w
k_p, k_i	Proportional and Integral gains of the controller in the camera

$$B_{p,w} = b_{p,t_{\mathcal{M},w}} \cdot \pi_{\text{alloc}} \cdot \mathcal{H} \tag{4.7}$$

If the frame size is greater than the amount of bandwidth allocated, i.e. $s_{p,w} > B_{p,w}$, the frame is dropped as it is outdated and the corresponding transmission slot is lost.

$$b_{p,t+1} = b_{p,t} + \varepsilon \cdot \{-\lambda_{p,t} \cdot f_{p,t} + b_{p,t} \cdot \sum_{i=1}^{n} [\lambda_{i,t} \cdot f_{i,t}]\} \tag{4.8}$$

The fraction of bandwidth allocated to each camera is calculated using Equation 4.8. $\lambda_{p,t} \in [0,1]$ denotes the relative importance of camera c_p in the network and the extent of bandwidth reallocation by the manager. ε is a constant that limits the change in bandwidth trading off between the responsiveness of the manager and its robustness to disturbances.

The matching function $f_{p,t}$ determines a match between the frame size $s_{p,w}$ and the allocated bandwidth $B_{p,w}$ for the w-th frame of camera p. In the current implementation the matching function is chosen to be the same as the normalized error used in the camera given by Equation 4.3. In order for the system to adhere to properties such as *starvation avoidance, balance, convergence* and *stability*, the matching function has to 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 $b_{p,w} \leq b_{p,t_{w-1}}$.

Table 2. Explanation of manager terms

Term	Explanation
$\mathcal{M}$	Manager
$B_{p,w}$	Bandwidth allocated to camera p for the transmission of the w-th frame
$b_{p,t}$	Fraction of bandwidth calculated for camera p at time t (between 0 and 1)
π_{alloc}	Allocation period
$\mathcal{H}$	Global available bandwidth
ε	Scaling factor that decides the aggressiveness of the manager
$\lambda_{p,t}$	Decides the importance of camera in the network [Maggio, Bini, Chasparis, and Årzén, 2013]
$f_{p,t}$	Error of camera p at time t
τ_{thr}	Threshold that triggers the manager

The manager is triggered with either a time-triggered or an event-based strategy. The time-triggered manager is activated periodically with period $\pi_{\mathcal{M}}$, a multiple of π_{alloc}, the allocation period. At time 0 the manager divides the available bandwidth among the cameras equally. 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 relationship from Equation 4.8. This triggering strategy has some drawbacks. Triggering the manager has a computational overhead with linear complexity on the number of cameras. The choice of $\pi_{\mathcal{M}}$ is also non-trivial. Choosing a small value of $\pi_{\mathcal{M}}$ invokes the manager too frequently causing constant bandwidth reallocation. On the other hand, a large value leads to slow reactions to changes in the scenes captured by the cameras.

An event-based manager reallocates bandwidth only when necessary, avoiding frequent bandwidth recalculations [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017]. We formulate the following rules to ensure the event-based manager reacts to changes in the network.

- $\mathcal{C}_{t^-} \neq \mathcal{C}_t$. When the number of cameras in the network changes. The manager starts by allocating equal amount of bandwidth to all the cameras and then proceeds to allocate bandwidth depending on the threshold and calculates it using on Equation 4.8.

- $(\exists c_p \text{ s.t. } \tau_{\text{thr}} < |e_{p,w_t}|) \wedge (t \bmod \pi_{\text{alloc}} = 0)$. When all the cameras have finished transmission and a camera has an error greater than the absolute value of the threshold τ_{thr}.

The system allows cameras to dynamically join and leave the network. For convenience Tables 1 and 2 give the summary of terms used in this paper.

3. Model Checking

This section gives an introduction to model checking. First we discuss the modeling checking methodology and describe the two camera models constructed. Then we define the different camera properties constructed for evaluation and model parameters used for system evaluation.

3.1 Model Checking Introduction and Motivation

Cyber-physical systems and communication networks are growing in complexity. This rapid growth also increases the presence of errors and makes the reliability of these systems a critical issue. Model checking has shown promise in detecting errors and providing guarantees in large scale cyber-physical systems.

Traditionally, software testing has been the dominant methodology to verify software systems. However it has the drawbacks of being a time consuming, complex, and costly procedure. In some cases testing is not feasible due to the inability to cover all test cases. In general, model checking is a more powerful tool compared to testing as it uses a model and provides more useful information about system behavior.

Hardware systems have high manufacturing costs, and therefore it is crucial to catch design errors early. Hardware correctness is generally verified using emulation and simulation. Similar to software verification, exploring all the possible input combinations and verifying all outputs is often not feasible. Model checking can be used in both these scenarios.

Model checking is used to verify certain properties of a system model. In model based verification, the models are described in a mathematically precise way. Systems are modeled as Markov chains and verified using existing model checking and verification tools, some of which are **PRISM** [Kwiatkowska, Norman, and Parker, 2011], **SPIN** [Holzmann, 2003], **UPPAAL** [Larsen, Pettersson, and Yi, 1995].

Model checking also makes it possible to generate strategies for optimal operation of the modeled system while subjected to required constraints. This strategy of operation depends on precise modeling of the system. It is therefore important to eliminate any ambiguities or incompleteness in the model but include non-determinism and a characterization of the uncertainty.

Figure 3 shows the general work-flow of applying model checking for property verification. The *system model* represents the functioning of the system as an MDP consisting of states and transitions. *Sytem property* is the required behaviour of the system that is evaluated. The surveillance camera network *model* and *properties* are explored in sections 3.2 and 3.3 respectively.

Probabilistic Model Checking (PMC) involves modeling of systems with stochastic behavior and the verification of properties of such systems. In PMC, systems are modeled as state-transition systems where transitions between the different states are decided by probabilities. Discrete-Time Markov chains (DTMC), Markov Decision Processes (MDP) and Probabilistic Timed Automata (PTA) are some of the models that support probabilistic transitions. The prop-

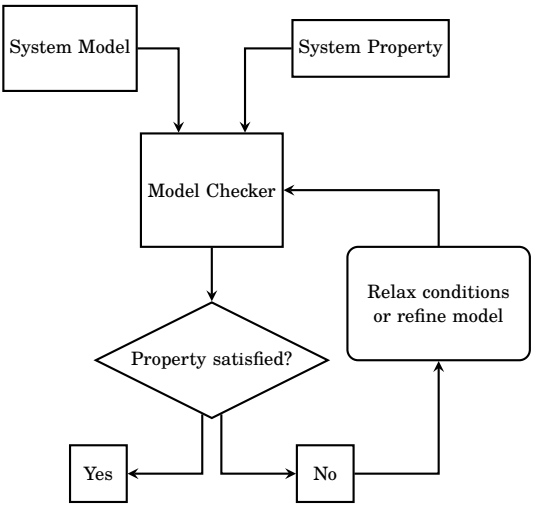


Figure 3. Model Checking Methodology

erties to be verified are expressed using probabilistic temporal logic, such as probabilistic reward computation-tree logic (PRCTL) [Andova, Hermanns, and Katoen, 2003]. The camera surveillance network is modeled using Markov Decision Processes (MDP).

.

A MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a set of finite states, $\mathcal{A}$ is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a function that encodes the probability of transitioning from state s to state s' as a result of an action a, and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the choice of action a determines a transition from state s to state s'.

3.2 Deterministic vs Probabilistic Models

Deterministic models are simple system representations that do not contain any probabilities on transitions between the different states in the system model. This makes property verification simple and allows larger number of sub-systems in the system model. However, these simple models does not cover realistic behaviors of a system. Systems with unreliable or uncertain behavior require modeling of probabilities between the different transitions in the model. Probabilistic models are used for verifying randomised algorithms like coin tossing experiments and model-based performance evaluation in failure prone systems.

Uncertainties in the camera network model arise during image capture as shown in Equation 4.1. Previous work [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] considered a linear relationship was considered between quality and frame size and did not consider the disturbance δ during system verification.

3.3 Property Verification

The characteristics of the system that is being verified are modeled as properties using formal language. The properties are modeled using precise mathematical equations and check for the presence of appropriate states and paths in the state-space. The different properties are classified into Qualitative and Quantitative properties.

Qualitative properties evaluate definite state reachability or full probability of the property. For example, $\mathsf{P}_{\geq 1}[\mathsf{G}\ \mathcal{H}_{\mathsf{alloc}} = \mathcal{H}]$, where $\mathcal{H}$ indicates the total amount of bandwidth available and $\mathcal{H}_{alloc}$ indicates the total amount of bandwidth allocated to all the cameras connected in the network. G denotes that the property is checked globally. The property checks for complete bandwidth allocation with a probability of 1.

Quantitative properties evaluate whether the system reaches a state or exceeds a property with a threshold. They are framed similar to qualitative properties with a relaxed probability requirement. Considering the same PRTCL property as above, $\mathsf{P}_{\geq \frac{2}{3}}[\mathsf{G}\ \mathcal{H}_{\mathsf{alloc}} = \mathcal{H}]$ relaxes the previous strict probability to $\frac{2}{3}$. The relaxed bound simplifies property verification and accommodates larger model sizes.

3.4 Classical vs Statistical Model Checking

Model checking tools construct the whole state-space by building states and transitions, and storing it in memory. These tools have an upper limit on memory usage to limit runoff programs, restricting the size and complexity of system models. Classical model checking builds the whole state-space irrespective of the presence(or absence) of probabilities in transitions. Probabilistic models capture complex behaviors of the system but are larger in size in comparison to deterministic models. This state-space explosion causes memory issues during verification of complex system models.

Statistical model checking (SMC) [David, Larsen, Legay, Mikučionis, and Wang, 2011] performs Monte Carlo simulations of the system with stochastic behaviors. SMC simulates the system for a finite number of runs and applies techniques from the area of statistics to evaluates it for property verification (or failure). These Monte Carlo simulations of the system provide estimates on the probabilistic measure on executions. SMC provides relaxed bound on system performance useful for verification of large complex systems, where classical model checking fails due to memory limitations [David, Larsen, Legay, Mikučionis, and Poulsen, 2015].

3.5 Strategy generation

Strategy generation in a probabilistic system selects an optimal action or state transitions to resolve non deterministic choices and maximise rewards (or minimise cost). PRTCL reward minimization operator $\mathsf{R}^{r}_{\mathsf{min}=?}$ selects a strategy that resolves r where r is the reward.

For example, $R^{r}_{min=?}[F\ \phi]$ minimises the total reward r along paths that leads to state satisfy the state formula ϕ. In the camera network, some relevant strategies are 'minimise the number of manager interventions', 'minimise the total number of dropped frames', 'minimise a defined cost' or maximise the bandwidth utilization'. Strategy generation is considered to be out of scope of this paper.

4. System Implementation

This section describes the modeling of the camera surveillance system and disturbances during image capture using PRISM model checker. We also describe the construction of different camera properties and model parameters used for model checking.

4.1 System Modelling

The different entities of the system are modeled as a set of modules in PRISM. In the camera network model, the different modules are namely the manager, the scheduler and the cameras. The cameras are modeled identically to each other but have their own independent modules. The scheduler is introduced into the camera model to determine the order in which the the different modules in the system are invoked. This module is necessary as the network manager is modeled to act only when required in an event-based manner. The camera modules are scheduled in a turn based manner and then the scheduler evaluates the necessity of invoking the event-based manager. An example schedule is show below.

$$
\begin{aligned}
[Turn1]scheduler &\rightarrow manager \rightarrow cam_1 \rightarrow cam_2 \rightarrow ....cam_n \\
[Turn2]scheduler &\rightarrow manager \rightarrow cam_1 \rightarrow cam_2 \rightarrow ....cam_n \\
[Turn3]scheduler &\rightarrow cam_1 \rightarrow cam_2 \rightarrow ....cam_n \\
&.... \\
&.... \\
&.... \\
[TurnX]scheduler &\rightarrow manager \rightarrow cam_1 \rightarrow cam_2 \rightarrow ....cam_n
\end{aligned}
$$

The scheduler invokes the manager and all the cameras in order during Turns 1 and 2. Turn 3 shows a scenario when the cameras have reached equilibrium (the frame sizes match the allocated bandwidths) and the manager is not invoked. It is invoked again during Turn X when the allocated bandwidth requires recalculation.

4.2 Disturbance Modelling

Disturbances occur in the camera due to changes in the scene captured by the camera. Equation 4.1 models the behavior of a camera with disturbances where δ denotes the disturbance. This uncertainty is modeled in PRISM using probabilities on the different transitions. The camera module is composed of two different formulas for calculation of the frame size. The first formula considers the value of δ to be 0. The second formula includes δ during frame size calculations and results in a frame of larger size. The current model assigns a probability of 0.7 to the linear formula and 0.3 to the formula for frame size with disturbances to ensure that the total probability is 1.

4.3 Properties

Model checking involves verifying relevant properties of a given system model. These properties are generally the desired/undesired behaviors of the model. Real life requirements are converted to formal language and input into the model checker as shown in figure 3. The following properties for the camera model are verified

1. **Convergence:**

$$P_{min} =?[F(G!(any\_change\_event))]$$

   Convergence is defined as "nothing happens in the model after some time" for model checking. The PRCTL formula above returns the minimum probability P_{min} of the model finally F, globally G such that no changes occur in the system !(any_change_event). !(any_change_event) is a variable that records changes in the bandwidth allocations and can be used as a measure to gauge if the system has converged.

2. **Number of Resource Manager Interventions:**

$$R_{max}^{rm\_calls} =?[F(rounds = max\_rounds)]$$

   This property is used to quantify the number of manager interventions during one verification run of the model. The reward operator $R_{max}^{rm\_calls}$ here counts the maximum number of resource manager calls rm_calls finally F after a certain number of rounds max_rounds have passed.

3. **Number of dropped frames :**

$$R_{max}^{dropped\_frames} =?[F(rounds = max\_rounds)]$$

   Similar to the previous property, this property quantifies the number of dropped frames. The reward operator $R_{max}^{dropped\_frames}$ counts the maximum number of dropped frames dropped_frames, finally F after a certain number of rounds max_rounds have passed.

4. **Total cost:**

$$R_{max}^{total\_cost} =?[F(rounds = max\_rounds)]$$

total_cost calculates the cost incurred by a triggering policy i.e chosen threshold. The property is tuned differently depending upon actions that are penalizing to a system. For example dropping of frames, manager interventions, or addition/deletion of new cameras affect the camera network differently dependent upon the configuration of the system. In this paper we penalise dropped frames and manager interventions by 10 and 1 respectively. In PRISM this can be defined as $total\_cost = 10 \cdot dropped\_frames + 1 \cdot rm\_calls$.

4.4 Model Parameters

The performance of the system is evaluated with respect to different parameter values using model checking. This enables faster parameter evaluation and tuning with varying disturbances in the model. For the camera system the following parameters are considered,

ϵ is the scaling factor that denotes the aggressiveness of the system. It is a trade off between responsiveness and stability of the manager. Higher values of ϵ increase convergence speed of the system but may cause overshoots, whereas small values of ϵ result in a slower but more stable behavior during network reconfigurations. In the current model, ϵ is set to 0.5.

$\lambda \in (0,1)$ denotes the importance of the camera in the network. When $\lambda = 0$, the network manager does not reallocate bandwidth to the camera and the onus is on the camera to adjust its bandwidth without manager interventions. When $\lambda = 1$, the network manager adjusts the bandwidth allocated to the camera at a higher rate with no quality changes expected from the camera. This methodology grants the architecture the ability to incorporate multiple types of cameras in the same network. For example, if camera c_p is incapable of changing it encoding quality, then λ_p is set to 1.

The triggering threshold $\tau_{thr} \in (0.1, 0.9)$, determines the activation of the manager for network re-configurations. The choice of τ_{thr} is non-trivial and is discussed in detail in [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017]. Small values of τ_{thr} lead to quicker manager activation but cause higher number of network re-configurations. Large values of τ_{thr} reduce the number of manager interventions and network re-configurations at the cost of information loss.

$k_{k,p}$ and $k_{i,p}$ are the proportional and integral gains of camera p that determine the aggressiveness with which the camera regulates its quality. Large controller gains generally lead to faster camera adaptations and larger quality changes.

τ_{thr} and $k_{i,p}$ are analyzed in more detail in the following sections.

5. Verification Results

This section discusses the verification results for the camera models built in PRISM. The section also shows the state-space evolution and property verification results for the two different types of the model.

Model I: The first model of the camera system describes the system with no probabilities. The camera modules use a linear equation to derive the frame size $\hat{s}_{p,w}$ given a quality $q_{p,w}$. Equation 4.1 with $\delta = 0$ disregards the effect of disturbance during image capture. This simplification in frame size calculation results in a smaller state-space that enables performance evaluation of a larger number of cameras. Model II also provides stronger property verification guarantees using classical model checking compared to Model II.

Model II: The second model of the camera network incorporates disturbances during image capture as probabilistic transitions. The probabilities are added during the frame size calculation as shown in Equation 4.1. A realistic model of the camera model is realized using two independent equations. The first equation is linear and already described in Model I. A high probability is attached to this equation. The second equation generates a larger frame size $\hat{s}_{p,w}$ for a given quality $q_{p,w}$ compared to the linear equation. A lower probability is attached to this equation. The probability distribution models the scenario in which disturbances occur in the generated frames.

5.1 Classical model checking results

Table 3 shows the evolution of the size of camera state-space. The camera and manager modules are modeled in minimal number of states while capturing the relevant behavior of the physical camera network. The system also imposes an upper limit on the number of frames transmitted to the manager to ensure convergence of property verification. Without the limit, the state space grows infinitely and makes model checking impossible due its infinite size.

Model I without probabilities grows at a lower rate compared to the model with probabilities, Model II. PRISM is unable to build the model of the camera network with probabilities due to memory limitations for Model II with number of cameras larger than 5.

Deterministic Model Results The subsection analyses the impact of controller gain k_i on the performance of the camera model without disturbances. Figure 4 shows the effect of k_i on the percentage of frames dropped by the manager due to bandwidth restriction violations. The plot on the left shows the percentage when controller gain k_i is 5 and the right plot when k_i is 25, for an increasing number of cameras. There is a very low percentage of dropped frames when the number of cameras in the network is 2, 4, 5 and 10 due to floating point calculations leading to quicker convergence of bandwidth calculations. The frames are dropped mostly during network initialization and initial regulation of camera quality. This is transient phase of the camera network as the total amount of available bandwidth is lower than the ideal bandwidth required for each cam-

Table 3. State space size evolution

Number of Cameras	Model I	Model II
2	123	309
3	157	5537
4	183	9297
5	213	54561
6	247	N/A
7	277	N/A
8	307	N/A
9	337	N/A
10	363	N/A

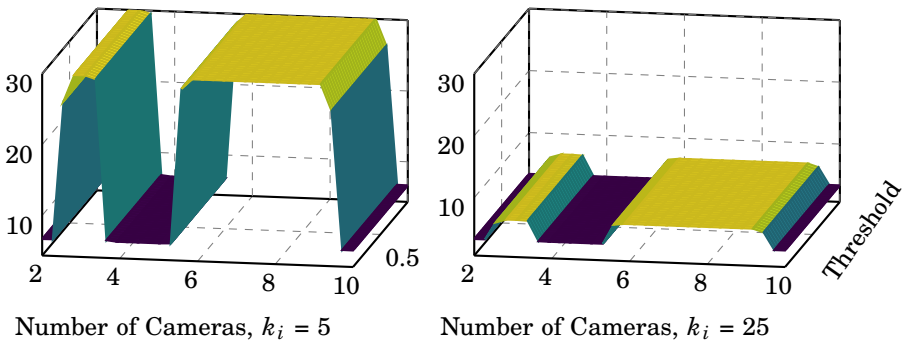


Figure 4. Percentage of Dropped Frames for Model I

era. This causes quality q_p regulation in camera frame sizes. If the current frame size, $s_{p,w}$ is greater than the amount of bandwidth $B_{p,w}$ allocated by the manager, then the frame is dropped. Larger k_i leads to faster bandwidth adaptation and convergence, leading to fewer dropped frames.

Probabilistic Model Results Adding probabilities to the model increases the size of its state-space as shown in Table 3. This increases the complexity of property verification due to the large state-space. PRISM is unable to build models with large number of cameras due to memory issues resulting in a reduced number of cameras to 5. Figure 5 shows the percentage of dropped frames for two values of controller gain k_i. Having a low value of k_i, causes a frame drop greater than 75% while using a gain of higher value reduces the dropped frames. It is also seen that the amount of dropped frames is higher for Model II compared to Model I. This is due to the presence of dynamics in the model giving a more complex analysis of the system. Using a probabilistic model in combination with

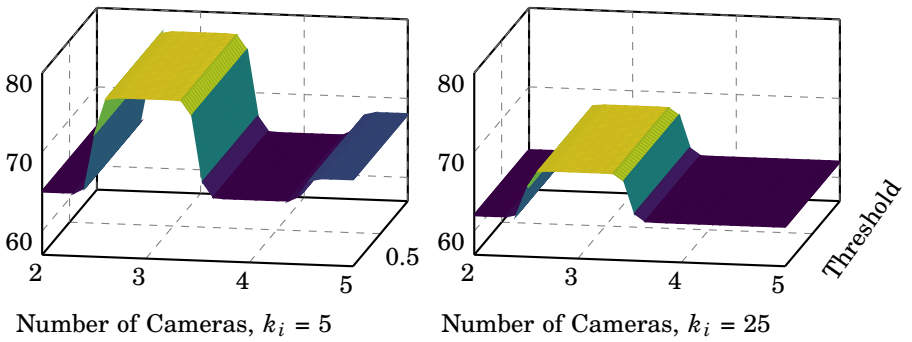


Figure 5. Percentage of Dropped Frames for Model II

model checking enables choosing controller parameters using more detailed information. There is also faster convergence for a particular number of cameras as seen in the previous experiment.

5.2 Statistical Model Checking Results

Statistical model checking allows verification of an increased number of camera modules with disturbances using model II. Compared to classical model checking, statistical model checking is able to build state-space of the camera network with 20 cameras instead of 5. Figure 6 shows the effect of threshold, τ_{thr} on the percentage of dropped frames by the cameras. The percentage of dropped frames increases for higher values of τ_{thr} and lower values of k_i as the camera adaptation is slower and the manager does not intervene for small disturbances.

Higher values of k_i leads to reduced number of dropped frames for all thresholds due to faster camera adaptations. Increasing τ_{thr} does not increase the number of dropped frames as the cameras regulate their bandwidth utilization quicker. There is faster convergence for values 4, 5, 10 and 20 due to divisibility and model building convergence. We do not apply SMC to the linear camera model, Model I, as the state-space of the model is smaller in size and classical model checking provides more robust bounds on property verification.

6. Experimental Results

This section describes the experimental results obtained using the camera testbed. First, we provide context for the experiment by describing results from our previous work [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017]. Next we discuss costs obtained for varying triggering thresholds τ_{thr} with experiments conducted to evaluate the number of dropped frames and manager interventions.

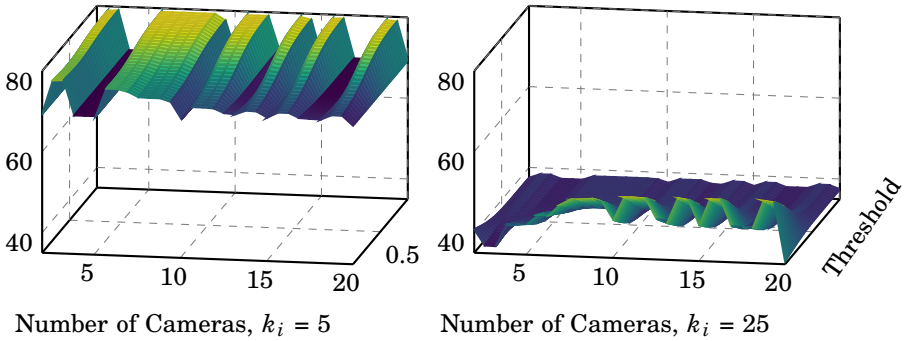


Figure 6. Percentage of Dropped Frames for Model II using SMC

6.1 Experimental Setup

The camera network is composed of multiple cameras and a network manager connected through local Ethernet. The multiple units of the network are implemented on a PC with Intel core i7-4790 8 core CPU with 32 GB RAM. The current implementation of the network consists of three off-the-shelf Logitech C270 cameras.

The experiment has the following setup:

- Proportional Gain: $k_{p,1} = k_{p,2} = 10$
- Integral Gain: $k_{i,1} = k_{i,2} = 0.5$
- Minimum Quality: $q_{1,\min} = q_{2,\min} = 15$
- Maximum Quality: $q_{1,\max} = q_{2,\max} = 85$
- Camera Importance: $\lambda_1 = 0.7$, $\lambda_2 = 0.3$
- Scaling Factor: $\varepsilon = 0.4$
- Allocation Period: $\pi_{\text{alloc}} = 30\,\text{ms}$
- Total Network Bandwidth: $\mathcal{H} = 4\,\text{Mbps}$.

The total network bandwidth is constrained to ensure camera adaptation.

6.2 Results

Previous work [Seetanadi, Oliveira, Almeida, Arzen, and Maggio, 2017] describes the results for camera adaptation with time-triggered manager. Similarly, [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] investigated the implementation of a event-triggered manager. This paper builds on the previous experimental results by examining the operation of the event-triggered manager for a larger number of threshold values.

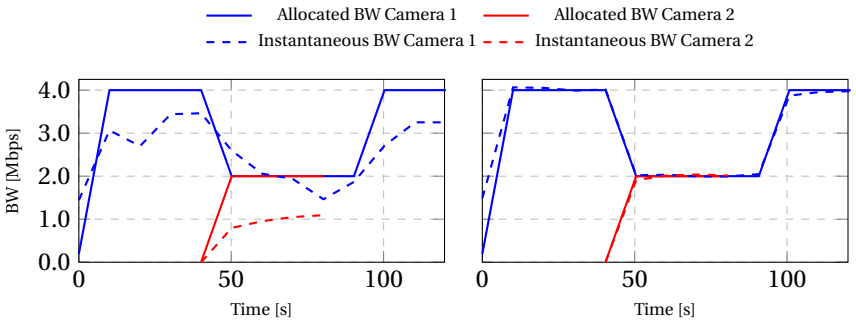


Figure 7. Results with Equal Distribution, and adaptation with [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] (left plots) and PI controller (right plots) [Seetanadi, Oliveira, Almeida, Arzen, and Maggio, 2017]

Recall that there are two control strategies:

- The network manager, that decides the amount of bandwidth to be allocated to each of the cameras
- The camera, that adapts its quality to fit the frame to the allocated bandwidth

[Seetanadi, Oliveira, Almeida, Arzen, and Maggio, 2017] described the camera adaptation using control theory and also showed the advantage of cooperation between the network manager and the cameras work to ensure optimal network utilization. [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] improves on the work by showing a need for triggering the manager only when required and the non-trivial decision involving the optimal selection of the triggering threshold, τ_{thr}.

6.3 Time Triggered Manager

In this experiment, the manager allocates an equal amount of bandwidth to the two cameras in the network to evaluate camera adaptation. The network manager allocates bandwidth to the cameras periodically with game-theoretic bandwidth allocation.

Figure 7 shows the comparison between two adaptation schemes, one from [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] and the adaptive PI controller from previous work [Seetanadi, Oliveira, Almeida, Arzen, and Maggio, 2017]. Only camera 1 is present initially up to time 40 seconds, then camera 2 joins the network. Camera 2 leaves the network at around time 80 seconds.

The left plot shows the adaptation strategy from [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] with a frame size estimation model to fit image sizes to the allocated bandwidth. The right plot shows the camera with a tuned PI controller. The parameters of the camera controller are tuned empirically according

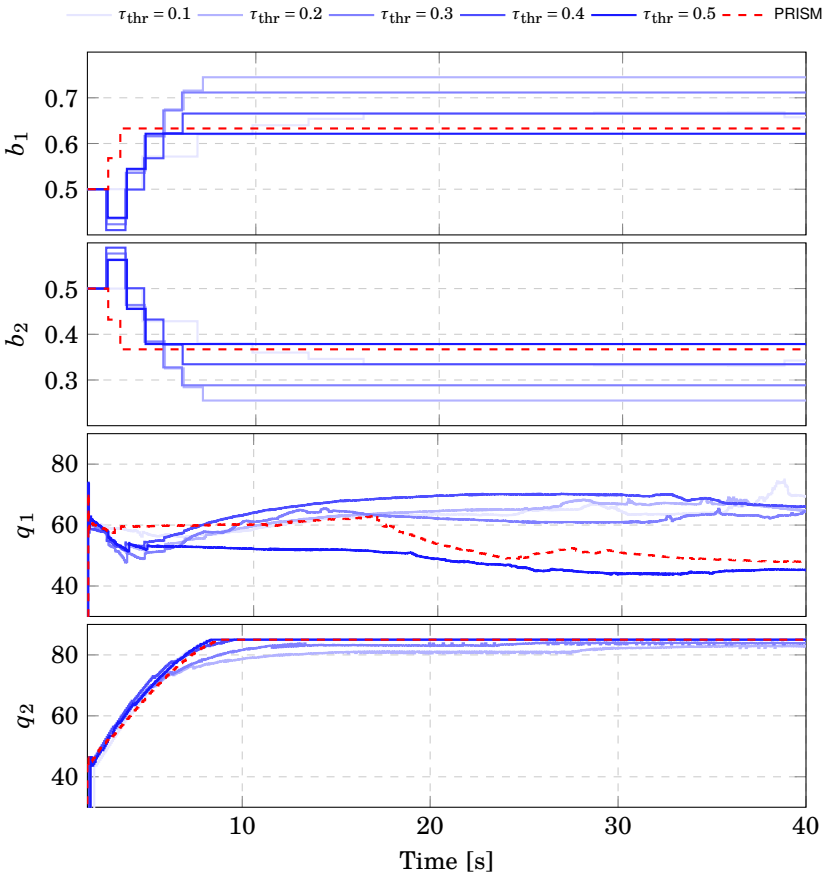


Figure 8. Event-triggered activation with $\tau_{thr} \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the optimal strategy synthesised by PRISM: bandwidth allocation (b_1 and b_2) and image quality (q_1 and q_2) [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017].

to standard control practices [Åström and Hägglund, 1995]. The PI controller is more efficient at using the bandwidth allocated by the network manager compared to the model from [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011]. The combination of time-triggered bandwidth allocation and camera adaptation using PI controller is efficient in bandwidth allocation and its utilization by the cameras.

6.4 Event-triggered Manager

The event-triggered manager is designed to be more efficient compared to the time-triggered manager. It is only triggered when the absolute error in the cam-

eras exceeds τ_{thr}.

Figure 8 shows the event-triggered manager triggered at different thresholds. Both cameras are initialized in the network simultaneously at time 0. Camera 1 captures a scene that contains a large number of artifacts and thus requires a larger allocation of the bandwidth. Camera 2 captures a simple scene with fewer movements and artifacts requiring a comparatively smaller bandwidth allocation. The manager allocates a larger amount of bandwidth to camera 1 and lower amount of bandwidth to camera 2 as seen in the figure. Thus, selecting the value of τ_{thr} becomes a crucial design choice to optimize performance of the camera surveillance network.

6.5 Choice of Triggering Threshold, τ_{thr}

The cost function chosen in the paper is defined as shown in Equation 6.5, where man_int is the number of manager interventions and $drop1$, $drop2$ and $drop3$ are the number of dropped frames of camera 1,2 and 3 respectively.

$$cost = 1 \cdot man\_int + 10 \cdot \sum(drop1 + drop2 + drop3)$$

Figure 9 shows the cost for the camera network with various triggering thresholds τ_{thr}. The plot shows the results of 1 hour long experiments performed 10 times independently. The cost is large for both low and high values of τ_{thr}. For low values of τ_{thr}, the cost is high due to the continuous reconfiguration of the network accumulating costs for both manager interventions and dropped frames.

Lower values of τ_{thr} lead to frequent network reconfigurations by the network manager. Small disturbances in the the camera scenes invoke manager interventions and lead to frame drops during network reconfigurations. Higher values of τ_{thr} increase costs and variance. There are fewer network reconfigurations even during large disturbances in scenes captured by the cameras. This places more emphasis on the cameras to regulate their bandwidth usage, causing large number of frame drops. For $\tau_{thr} = [0.3, 0.6]$, the cost is lower compared to other values of τ_{thr}. These values indicate a good trade-off between high costs due to frequent interventions and high costs due to dropped frames.

Figure 10 shows the trend of number of manager interventions for increasing values of τ_{thr}. It shows a 5 point smoothed average of the number of manager interventions. The number of manager interventions reduces as τ_{thr} increases as there are fewer network re-configurations and the manager is not invoked for small disturbances in the camera images. Higher values of τ_{thr} lead to fewer manager interventions with no network reconfigurations even during large changes in camera scenes.

Figure 11 shows a 5 point average of the number of dropped frames versus threshold τ_{thr}. The number of dropped frames increases for higher values of τ_{thr} due to lower number of manager interventions placing emphasis on quicker camera adaptations. These results show that the tuning of the cost function is

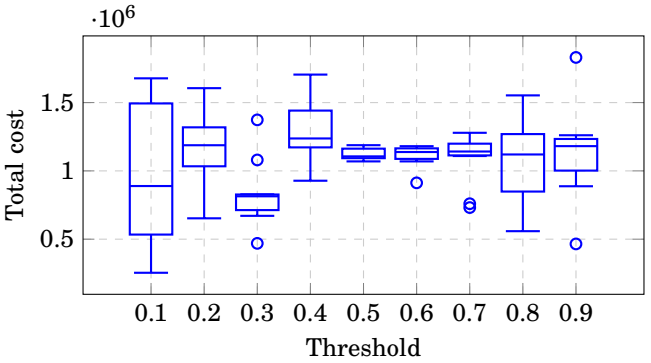


Figure 9. Total cost box plot for 10 1-hour long executions traces

non-trivial and can be varied by choosing the appropriate weights depending upon the behavior of the camera network.

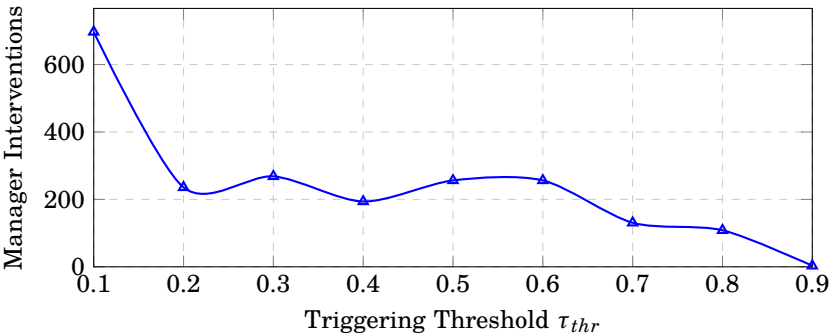


Figure 10. Manager Interventions

7. Related Work

Video transmission using self-adaptive cameras have been investigated in scenarios where images are transmitted over the internet or local area networks with focus on image compression [Vandalore, Feng, Jain, and Fahmy, 2001; Communication, 2004; Rinner and Wolf, 2008; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008; Toka, Lajtha, Hosszu, Formanek, Géhberger, and Tapolcai, 2017; Zhang, Chowdhery, Bahl, Jamieson, and Banerjee, 2015]. Research on video transmission has led to transmission protocols such as RTP, RTSP, SIP and improvements upon them which measure key network parameters, such as bandwidth usage, packet loss and round-trip delays. These help to cope with network

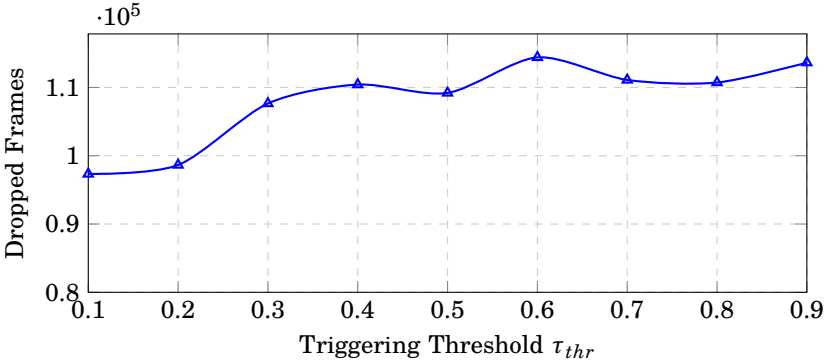


Figure 11. Dropped Frames

load conditions, controlling the load submitted to the network [Veeraraghavan and Weber, 2008] or using traffic prioritization [Cao, Nguyen, and Nguyen, 2013]. Research into image compression led to standards to reduce bandwidth utilization like MJPEG, JPEG2000, MPEG-4, H.264, MPEG-H and H.265 that remove redundant information in sequence of frames.

Some research domains impose limitations on the acceptable delays in the system. for eg. augmented reality [Razavi, Fleury, and Ghanbari, 2008], industrial supervised multimedia control [Rinner and Wolf, 2008], multimedia embedded systems [Ramos, Panigrahi, and Dey, 2007], automated inspection [Kumar, 2008] and vehicle navigation [Lima and Victorino, 2016]. Video surveillance is similar. Image compression is frequently preferred to video compression. This is due to lower latency incurred and low memory and computing requirements during image compression compared to video compression. The variability that occurs during compression complicates fitting the compressed image to the instantaneous network conditions. This has motivated substantial research into adaptive techniques [Rinner and Wolf, 2008; Ramos, Panigrahi, and Dey, 2007; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008]. These works focus mainly on the adaptation of streams to what the network provides. Not on the scheduling on the network. A way of achieving this network reservation is through the use of channels. This is addressed in [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] which addressed the problem using adaptive network channels using a global network manager that tracks camera bandwidth usage. Using control theory for quality adaptation used in the paper is similar to [Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008]. Our work adds to the camera adaptation by implementing a network bandwidth distribution strategy. Also explored in this paper is the use of classical and statistical model checking to verify desirable properties.

Although better performance can be obtained using domain knowledge to optimise bandwidth allocation [Toka, Lajtha, Hosszu, Formanek, Géhberger, and Tapolcai, 2017], this paper assumes no such knowledge. Also investigated is the

usage of cameras that transmit images using wireless networks [Zhang, Chowdhery, Bahl, Jamieson, and Banerjee, 2015] which highlights the need for adaptation. In this case it increases the amount of processing at the node. Compared to the work, this paper also provides formal guarantees on the system obtained via model checking.

Model checking has been applied to analyse complex system in recent times. [Hamman, Hopkinson, and Fadul, 2017; Yüksel, Zhu, Nielson, Huang, and Nielson, 2012] show the application of model checking to smart grids using different model checking tools. Model checking has also been used for finding errors in cloud applications [Tian, Zhang, Zhou, and Zhong, 2016].

This paper uses PRISM [Kwiatkowska, Norman, and Parker, 2011] for the camera network analysis. Using the support available for probabilistic models and statistical model checking, PRISM has been used for verification of a wide-variety of systems. For e.g. randomised distributed algorithms [Kwiatkowska, Norman, and Segala, 2001], software with probabilistic transitions [Kattenbelt, Kwiatkowska, Norman, and Parker, 2009], designs for applications using nanotechnology [Norman, Parker, Kwiatkowska, and Shukla, 2005], communication protocols [Duflot, Kwiatkowska, Norman, and Parker, 2006], self-adaptive software [Calinescu, Ghezzi, Kwiatkowska, and Mirandola, 2012], security [Basagiannis, Katsaros, Pombortsis, and Alexiou, 2009] and anonymous protocols [Shmatikov, 2002].

PMC has been applied to verify properties of a video streaming system in [Nagaoka, Ito, Okano, and Kusumoto, 2011]. In this work, high-fidelity simulations have been abstracted into probabilistic higher-level models. PMC has also been used in verification of safety and timeliness properties in air traffic control systems [Hanh and Van Hung, 2007]. Contrary to all the mentioned papers, this paper uses PMC to verify a class of properties that is typically related with control-theoretical guarantees.

8. Conclusion

Model checking has shown great promise in verification of complex systems. Compared to testing, model checking is more exhaustive and robust. The drawback with model checking is state space explosion and translation of relevant properties of the system into formal language. The work in this paper addresses both of these concerns.

We applied model checking to a camera surveillance network with two independent strategies for bandwidth adaptation. We built two models of the adaptive cameras in the network, a linear model from [Seetanadi, Camara, Almeida, Årzén, and Maggio, 2017] that did not consider disturbances during image encoding, and a second model that incorporated disturbances using probabilistic transitions in the model. Application of classical model checking techniques allowed for more informed choice of control parameters and triggering thresholds for the camera models with and without disturbances. The camera model con-

sisting of probabilistic disturbances led to state-space explosion due to memory issues. We showed that using SMC, the state-space explosion can be mitigated by performing property verification of complex models at the expense of relaxed bounds. Although the obtained bounds are not as robust as classical model checking, we obtained useful information on system performance and parameter selection. Finally, we evaluated the effect of different triggering thresholds on the number of manager interventions and dropped frames.

References

Andova, S., H. Hermanns, and J.-P. Katoen (2003). "Discrete-time rewards model-checked". In: *1st International Workshop Formal Modeling and Analysis of Timed Systems*. DOI: 10.1007/978-3-540-40903-8_8.

Åström, K. J. and T. Hägglund (1995). *PID Controllers: Theory, Design, and Tuning*. 2nd ed. Instrument Society of America, Research Triangle Park, NC.

Basagiannis, S., P. Katsaros, A. Pombortsis, and N. Alexiou (2009). "Probabilistic model checking for the quantification of dos security threats". *Computers & Security* **28**:6, pp. 450–465. ISSN: 0167-4048.

Calinescu, R., C. Ghezzi, M. Kwiatkowska, and R. Mirandola (2012). "Self-adaptive software needs quantitative verification at runtime". *Commun. ACM* **55**:9, pp. 69–77. ISSN: 0001-0782.

Cao, D. T., T. H. Nguyen, and L. G. Nguyen (2013). "Improving the video transmission quality over ip network". In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. DOI: 10.1109/ICUFN.2013.6614884.

Communication, A. (2004). *White paper: digital video compression: review of the methodologies and standards to use for video transmission and storage*.

David, A., K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen (2015). "Uppaal smc tutorial". *International Journal on Software Tools for Technology Transfer* **17**:4, pp. 397–415. DOI: 10.1007/s10009-014-0361-y.

David, A., K. G. Larsen, A. Legay, M. Mikučionis, and Z. Wang (2011). "Time for statistical model checking of real-time systems". In: *Computer Aided Verification*. Vol. 6806. LNCS. Springer, pp. 349–355. ISBN: 978-3-642-22110-1. DOI: 10.1007/978-3-642-22110-1_27.

Ding, W. and B. Liu (1996). "Rate control of mpeg video coding and recording by rate-quantization modeling". *IEEE Transactions on Circuits and Systems for Video Technology* **6**:1, pp. 12–20. DOI: 10.1109/76.486416.

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

Edpalm, V., A. Martins, K.-E. Årzén, and M. Maggio (2018). "Camera Networks Dimensioning and Scheduling with Quasi Worst-Case Transmission Time". In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany, 17:1–17:22. ISBN: 978-3-95977-075-0.

Edpalm, V., A. Martins, M. Maggio, and K.-E. Årzén (2018). *H.264 Video Frame Size Estimation*. Technical Reports TFRT-7654. Department of Automatic Control, Lund Institute of Technology, Lund University.

Larsen, K. G., P. Pettersson, and W. Yi (1995). "Model-checking for real-time systems". In: *Proc. of Fundamentals of Computation Theory*. Vol. 965. LNCS, pp. 62–88. DOI: 10.1007/3-540-60249-6_41.

Hamman, S. T., K. M. Hopkinson, and J. E. Fadul (2017). "A model checking approach to testing the reliability of smart grid protection systems". *IEEE Transactions on Power Delivery* **32**:6, pp. 2408–2415. DOI: 10.1109/TPWRD.2016.2635480.

Hanh, T. and D. Van Hung (2007). *Verification of an Air-Traffic Control System with Probabilistic Real-time Model-checking*. Tech. rep. UNU-IIST United Nations University International Institute for Software Technology.

Heo, J. and T. Abdelzaher (2009). "Adaptguard: guarding adaptive systems from instability". In: *6th ACM International Conference on Autonomic Computing*. DOI: 10.1145/1555228.1555256.

Holzmann, G. (2003). *Spin Model Checker, the: Primer and Reference Manual*. First. Addison-Wesley Professional. ISBN: 0321228626.

Kattenbelt, M., M. Kwiatkowska, G. Norman, and D. Parker (2009). "Abstraction refinement for probabilistic software". In: *Verification, Model Checking, and Abstract Interpretation*. Berlin, Heidelberg, pp. 182–197.

Kumar, A. (2008). "Computer-vision-based fabric defect detection: a survey". *IEEE Transactions on Industrial Electronics* **55**:1, pp. 348–363. DOI: 10.1109/TIE.1930.896476.

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

Kwiatkowska, M. Z., G. Norman, and R. Segala (2001). "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. London, UK, UK, pp. 194–206. ISBN: 3-540-42345-1.

Lima, D. A. de and A. C. Victorino (2016). "A hybrid controller for vision-based navigation of autonomous vehicles in urban environments". *IEEE Transactions on Intelligent Transportation Systems* **17**:8, pp. 2310–2323. DOI: 10.1109/TITS.2016.2519329.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". In: *Euromicro Conference on Real-Time Systems*. DOI: 10.1109/ECRTS.2013.17.

Nagaoka, T., A. Ito, K. Okano, and S. Kusumoto (2011). "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. DOI: 10.1587/transinf.E94.D.958.

Norman, G., D. Parker, M. Kwiatkowska, and S. Shukla (2005). "Evaluating the reliability of nand multiplexing with prism". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **24**:10, pp. 1629–1637. ISSN: 0278-0070.

Ramos, N., D. Panigrahi, and S. Dey (2007). "Dynamic adaptation policies to improve quality of service of real-time multimedia applications in ieee 802.11e wlan networks". *Wirel. Netw.* **13**:4, pp. 511–535. DOI: 10.1007/s11276-006-9203-5.

Razavi, R., M. Fleury, and M. Ghanbari (2008). "Low-delay video control in a personal area network for augmented reality". *IET Image Processing* **2**:3. DOI: 10.1049/iet-ipr:20070183.

Rinner, B. and W. Wolf (2008). "An introduction to distributed smart cameras". *Proceedings of the IEEE* **96**:10. DOI: 10.1109/JPROC.2008.928742.

Seetanadi, G. N., J. Camara, L. Almeida, K.-E. Årzén, and M. Maggio (2017). "Event-driven bandwidth allocation with formal guarantees for camera networks". In: *RTSS, Real-Time Systems Symposium*. Paris, France. DOI: 10.1109/RTSS.2017.00030.

Seetanadi, G. N., L. Oliveira, L. Almeida, K.-E. Arzen, and M. Maggio (2017). "Game-theoretic network bandwidth distribution for self-adaptive cameras". In: *15th International Workshop on Real-Time Networks*. DOI: 10.1145/3267419.3267424.

Shmatikov, V. (2002). "Probabilistic analysis of anonymity". In: *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pp. 119–128.

Silvestre-Blanes, J., L. Almeida, R. Marau, and P. Pedreiras (2011). "Online qos management for multimedia real-time transmission in industrial networks". *IEEE Transactions on Industrial Electronics* **58**:3. DOI: 10.1109/TIE.2010.2049711.

Tian, T., Y. Zhang, Q. Zhou, and P. Zhong (2016). "Modelx: using model checking to find design errors of cloud applications". In: *2016 IEEE International Conference on Computer and Information Technology (CIT)*, pp. 607–610. DOI: 10.1109/CIT.2016.66.

Toka, L., A. Lajtha, É. Hosszu, B. Formanek, D. Géhberger, and J. Tapolcai (2017). "A resource-aware and time-critical IoT framework". In: *IEEE International Conference on Computer Communications INFOCOM*. DOI: 10.1109/INFOCOM.2017.8057143.

Valmari, A. (1998). "The state explosion problem". In: *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*. Springer, pp. 429–528. DOI: 10.1007/3-540-65306-6_21.

Vandalore, B., W.-c. Feng, R. Jain, and S. Fahmy (2001). "A survey of application layer techniques for adaptive streaming of multimedia". *Real-Time Imaging* **7**:3. DOI: 10.1006/rtim.2001.0224.

Veeraraghavan, V. and S. Weber (2008). "Fundamental tradeoffs in distributed algorithms for rate adaptive multimedia streams". *Comput. Netw.* **52**:6, pp. 1238–1251. DOI: 10.1016/j.comnet.2008.01.012.

Wang, X., M. Chen, H. M. Huang, V. Subramonian, C. Lu, and C. D. Gill (2008). "Control-based adaptive middleware for real-time image transmission over bandwidth-constrained networks". *IEEE Transactions on Parallel and Distributed Systems* **19**:6. DOI: 10.1109/TPDS.2008.41.

Yüksel, E., H. Zhu, H. R. Nielson, H. Huang, and F. Nielson (2012). "Modelling and analysis of smart grid: a stochastic model checking case study". In: *2012 Sixth International Symposium on Theoretical Aspects of Software Engineering*, pp. 25–32. DOI: 10.1109/TASE.2012.44.

Zhang, T., A. Chowdhery, P. (Bahl, K. Jamieson, and S. Banerjee (2015). "The design and implementation of a wireless video surveillance system". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438. DOI: 10.1145/2789168.2790123.

Paper IV

Model Checking a Self-Adaptive Camera Network with Physical Disturbances

Gautham Nayak Seetanadi Karl-Erik Årzén Martina Maggio

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.

1. 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 [Edpalm, Martins, Årzén, and Maggio, 2018]. Even when images are transmitted as simple JPEG frames, the encoded scene makes a difference in the image sizes [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017]. This motivates the need to include some considerations about the physics in the network bandwidth distribution protocols. However, due to the unpredictable and ever-changing 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 [Almeida et al., 2007]. Transmission protocols, like the Flexible Time Triggered for Switched Ethernet (FTT-SE) [Pedreiras and Almeida, 2003], guarantee the non-violation of the assigned bandwidth. In a recent paper [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017], 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 [Heo and Abdelzaher, 2009], so we needed formal guar-

antees that this would not happen in this case. We therefore resorted to model checking, and verified properties like the transmission of frames [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017]. 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 [Clarke, Klieber, Nováček, and Zuliani, 2012]. This paper discusses the problem we encountered and how we changed our model to 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017].
- 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 [Chen, Forejt, Kwiatkowska, Parker, and Simaitis, 2013].

The remainder of this paper is organized as follows. Section 2 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 3 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 4 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 5 gives an overview of related work and Section 6 concludes the paper.

2. 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., $\mathcal{H} = 4\,[Mb/s]$.

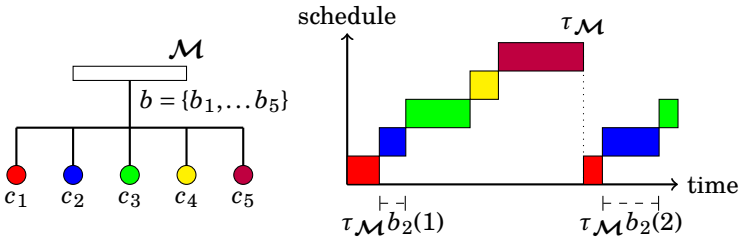


Figure 1. System Overview

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 [Maggio, Bini, Chasparis, and Årzén, 2013] adapted to network bandwidth in [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017], 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 [Varma, 2015]. 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 [Hellerstein, Diao, Parekh, and Tilbury, 2005; Varma, 2015].

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.

$$s_{i,w} = 0.01 \cdot q_{i,w} \cdot s_{i,\max} + \delta s_{i,w} \tag{4.1}$$

The calculated frame size is then saturated between pre-determined values of minimum and maximum allowable frame sizes as shown in (4.2).

$$s_{i,w} = \max\{s_{i,\min}, \min\{s_{i,\max}, s^*_{i,w}\}\}. \tag{4.2}$$

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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017], 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.

3. 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 [Kwiatkowska, Norman, and Parker, 2011], 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017], 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.

3.1 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,* $Pr : S \times A \rightarrow S$ *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 [Clarke, Henzinger, Veith, and Bloem, 2018]. We use MDPs to design the deterministic, probabilistic, and nondeterministic 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 [Kwiatkowska, 2016]. 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.

3.2 Modeling the Problem

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

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 4.1 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.

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

Listing 4.1. Network Manager, PRISM Code

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 4.1 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.

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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017]. 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 4.1 and n copies of a single camera model, depending on the number of cameras in the network. Listing 4.2

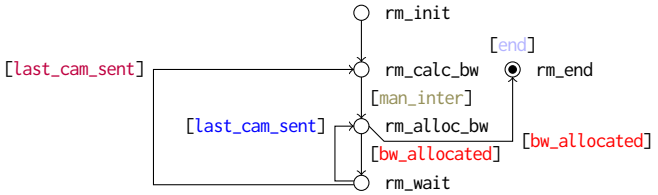


Figure 2. Network Manager, MDP Representation

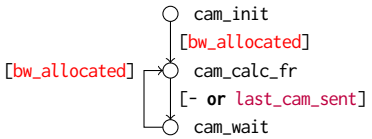


Figure 3. Camera Deterministic Model

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 (4.1), 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.

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 4.3. 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017] – and shown in Equation (4.1). With respect to prior work, however, it has been shown that the disturbance acting on the frame size would affect the

```
formula framesize = ... // compute frame size from equation (4.1)
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 4.2. Camera Deterministic Model, PRISM Code

```
formula framesize = ... // compute frame size from equation (4.3)
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 4.3. Camera Probabilistic Model, PRISM Code

camera quality as a multiplicative term [Edpalm, Martins, Årzén, and Maggio, 2018]. Rather than modeling the disturbance as shown in [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017], 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 stochas-

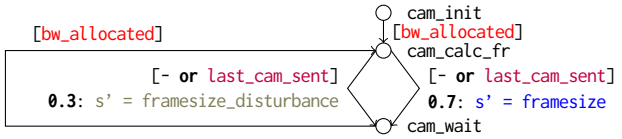
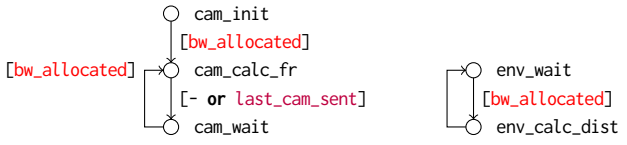


Figure 4. Camera Probabilistic Model



(a) Camera (b) Environment

Figure 5. Camera Nondeterministic Model

tic 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:

$$s_{i,w} = \begin{cases} (0.7) & 0.01 \cdot q_{i,w} \cdot s_{i,\max} \\ (0.3) & 0.01 \cdot (q_{i,w} + \delta q_{i,w}) \cdot s_{i,\max} \end{cases} \tag{4.3}$$

The first line in Equation (4.3) 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 [Edpalm, Martins, Årzén, and Maggio, 2018], and the values we used were found using the testbed developed for [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017].

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 4.1.

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

```
formula framesize_disturbance = ... // from equation (4.4)
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 4.4. Camera Nondeterministic Model, PRISM Code

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 4.4. 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.

$$s_{i,w} = 0.01 \cdot (q_{i,w} + \delta q_{i,w}) \cdot s_{i,\max} \tag{4.4}$$

We use Equation (4.4) to model the stochasticity of the image capture. The equation is the same as the second line in Equation (4.3), 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 4.1, that includes verification results.

3.3 Stochastic Games

PRISM-Games [Chen, Forejt, Kwiatkowska, Parker, and Simaitis, 2013] 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 4.3.

3.4 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 4.5 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 4.6 shows the properties that we use to evaluate the three different MDP models: the deterministic one presented in Section 3.2, the probabilistic

```
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 4.5. 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 4.6. Properties of MDP models

one presented in Section 3.2, and the nondeterministic one discussed in Section 3.2.The identifier **R** indicates a *reward* of the specified type. Rewards are used in PRISM to encode quantitative verification. The term reward indicates a positive quantity, but can be used to quantify costs as well [Kwiatkowska, Norman, and Parker, 2011]. 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 4.1 describes our results.

Similarly, Listing 4.7 shows the properties that we use to evaluate the cooperative vs competitive behavior of the system using SMGs. The properties of Listing 4.6 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 4.3 describes the results we obtained for these properties.

4. 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 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017]. The other parameters (e.g., the period of the resource manager) were set according to the previous experiments [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017][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.

4.1 Scalability

Here, we evaluate how the three proposed models scale with the number of cameras, i.e., with the size of the problem. Table 1 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio,

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

Table 1. 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

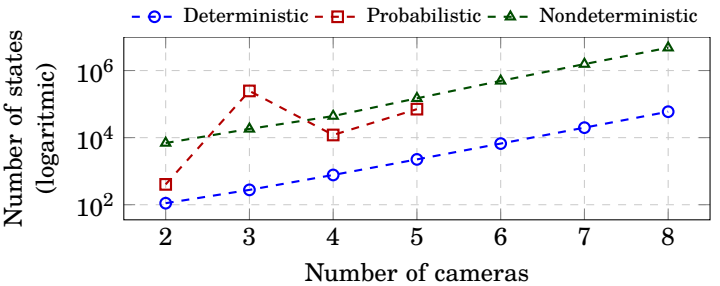


Figure 6. Scalability Analysis: State Space Growth Plot

2017], 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 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 band-

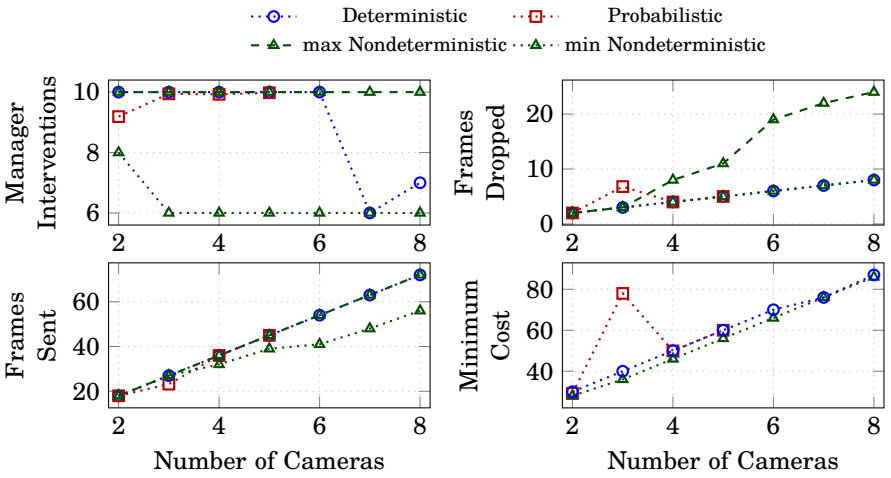


Figure 7. Property verification for Deterministic, Probabilistic, and Nondeterministic Model

width.

4.2 Property verification

In this section, we compare the results we obtain when we verify the properties described in Section 3.4 (Listing 4.6). 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 respected.

Minimum Cost: The final graph shows the minimum cost (where the cost is computed according to the formula introduced in Section 3.4, 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).

4.3 Collaborative vs. Competitive

This section provides a deeper analysis of the nondeterministic model presented in Section 3.2 (as it offers a realistic representation of the disturbance and better

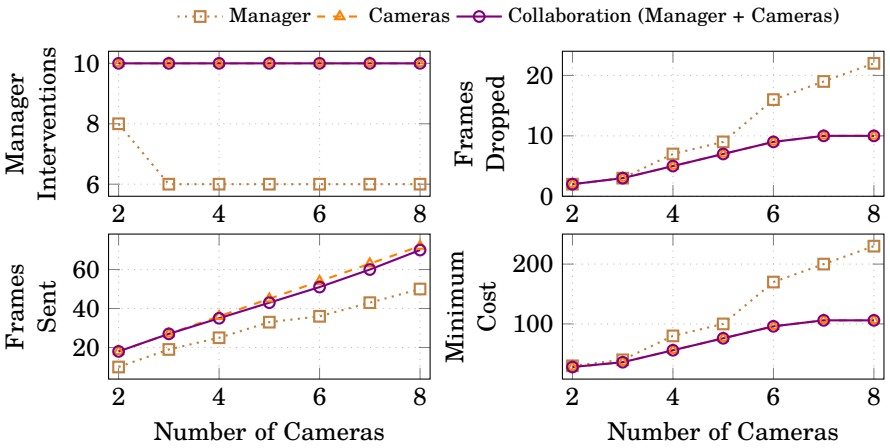


Figure 8. Cooperative vs. Competitive Optimization

scalability properties). We define the problem as a Stochastic Game, as shown in Section 3.3. We use PRISM-Games [Chen, Forejt, Kwiatkowska, Parker, and Simaitis, 2013] to compare the system's behavior when the objective is to minimize or maximize a property (among the ones defined in Section 3.4, Listing 4.7 — 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.

5. 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 [Vandalore, Feng, Jain, and Fahmy, 2001; Communication, 2004; Rinner and Wolf, 2008; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008; Toka, Lajtha, Hosszu, Formanek, Géhberger, and Tapolcai, 2017; Zhang, Chowdhery, Bahl, Jamieson, and Banerjee, 2015; Cao, Nguyen, and Nguyen, 2013], leading to alternative protocols and improvements. They typically 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 [Rinner and Wolf, 2008; Ramos, Panigrahi, and Dey, 2007; Wang, Chen, Huang, Subramonian, Lu, and Gill, 2008], without considering network scheduling. In the scheduling domain, [Silvestre-Blanes, Almeida, Marau, and Pedreiras, 2011] 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 [Seetanadi, Cámara, Almeida, Årzén, and Maggio, 2017] 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 [Fainekos, Girard, and Pappas, 2006]. 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 [Clarke, Henzinger, Veith, and Bloem, 2018] to verify desirable properties of the complex system composed of cameras and network manager. We used the PRISM model checker [Kwiatkowska, Norman, and Parker, 2011]. In the literature, PRISM has been used for the verification of properties of a wide-variety of systems [Kwiatkowska, 2016], among which we find: randomized distributed algorithms [Kwiatkowska, Norman, and Segala, 2001], probabilistic software [Kattenbelt, Kwiatkowska, Norman, and Parker, 2009], nontechnology designs [Norman, Parker, Kwiatkowska, and Shukla, 2005], communication protocols [Duflot, Kwiatkowska, Norman, and Parker, 2006], self-adaptive software [Calinescu, Ghezzi, Kwiatkowska, and Mirandola, 2012], security [Basagiannis, Katsaros, Pombortsis, and Alexiou, 2009], anonymous protocols [Shmatikov, 2002]. Specifically, we use Probabilistic Model Checking (PMC). PMC was instrumental in the verification of properties of a video streaming service in [Nagaoka, Ito, Okano, and Kusumoto, 2011]. 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 [Hanh and Van Hung, 2007]. 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 [Shapley, 1953; Chatterjee and Ibsen-Jensen, 2015], and partial-observing games [Chatterjee, Doyen, Nain, and Vardi, 2014; Chatterjee and Doyen, 2012]. We modeled our system as a turn-based game [Condon, 1992].

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 [Simaitis, 2014], UAV mission planning [Feng, Wiltsche, Humphrey, and Topcu, 2015], pan-tilt zoom cameras [Ozay, Topcu, Murray, and Wongpiromsarn, 2011], aircraft power distribution [Basset, Kwiatkowska, Topcu, and Wiltsche, 2015] and self-adaptive architectures [Cámara, Garlan, Schmerl, and Pandey, 2015]. 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 [Zeng, Natale, Giusto, and Sangiovanni-Vincentelli, 2009]. This work focused more on the timing analysis which we do not pursue in our work.

6. 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

Almeida, L. et al. (2007). "Online qos adaptation with the flexible time-triggered (FTT) communication paradigm". In: Insup Lee Joseph Y-T. Leung, S. H. S. (Ed.). *Handbook of Real-Time and Embedded Systems*. CRC Press.

Basagiannis, S., P. Katsaros, A. Pombortsis, and N. Alexiou (2009). "Probabilistic model checking for the quantification of dos security threats". *Computers & Security* **28**:6, pp. 450–465. ISSN: 0167-4048.

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

Calinescu, R., C. Ghezzi, M. Kwiatkowska, and R. Mirandola (2012). "Self-adaptive software needs quantitative verification at runtime". *Commun. ACM* **55**:9, pp. 69–77. ISSN: 0001-0782.

Cámara, J., D. Garlan, B. Schmerl, and A. Pandey (2015). "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. Salamanca, Spain, pp. 428–435. ISBN: 978-1-4503-3196-8.

Cao, D. T., T. H. Nguyen, and L. G. Nguyen (2013). "Improving the video transmission quality over ip network". In: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. DOI: 10.1109/ICUFN.2013.6614884.

Chatterjee, K. and L. Doyen (2012). "Partial-observation stochastic games: how to win when belief fails". In: *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pp. 175–184.

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

Chatterjee, K. and R. Ibsen-Jensen (2015). "Qualitative analysis of concurrent mean-payoff games". *Inf. Comput.* **242**:C, pp. 2–24. ISSN: 0890-5401.

Chen, T., V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis (2013). "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)*. Vol. 7795. LNCS, pp. 185–191.

Clarke, E. M., T. A. Henzinger, H. Veith, and R. Bloem, (Eds.) (2018). *Handbook of Model Checking*. Springer. ISBN: 978-3-319-10574-1.

Clarke, E. M., W. Klieber, M. Nováček, and P. Zuliani (2012). "Model checking and the state explosion problem". In: *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*. Berlin, Heidelberg, pp. 1–30.

Communication, A. (2004). *White paper: digital video compression: review of the methodologies and standards to use for video transmission and storage*.

Condon, A. (1992). "The complexity of stochastic games". *Information and Computation* **96**:2, pp. 203–224. ISSN: 0890-5401.

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

Edpalm, V., A. Martins, K.-E. Årzén, and M. Maggio (2018). "Camera Networks Dimensioning and Scheduling with Quasi Worst-Case Transmission Time". In: *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Vol. 106. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany, 17:1–17:22. ISBN: 978-3-95977-075-0.

Fainekos, G. E., A. Girard, and G. J. Pappas (2006). "Temporal logic verification using simulation". In: *Formal Modeling and Analysis of Timed Systems*. Berlin, Heidelberg, pp. 171–186.

Feng, L., C. Wiltsche, L. Humphrey, and U. Topcu (2015). "Controller synthesis for autonomous systems interacting with human operators". In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. ICCPS '15. Seattle, Washington, pp. 70–79. ISBN: 978-1-4503-3455-6.

Hanh, T. and D. Van Hung (2007). *Verification of an Air-Traffic Control System with Probabilistic Real-time Model-checking*. Tech. rep. UNU-IIST United Nations University International Institute for Software Technology.

Hellerstein, J. L., Y. Diao, S. Parekh, and D. M. Tilbury (2005). "Control engineering for computing systems - industry experience and research challenges". *IEEE Control Systems Magazine* **25**:6, pp. 56–68. ISSN: 1066-033X.

Heo, J. and T. Abdelzaher (2009). "Adaptguard: guarding adaptive systems from instability". In: *Proceedings of the 6th International Conference on Autonomic Computing*. ICAC '09. ACM, Barcelona, Spain, pp. 77–86. ISBN: 978-1-60558-564-2. DOI: 10.1145/1555228.1555256. URL: http://doi.acm.org/10.1145/1555228.1555256.

Kattenbelt, M., M. Kwiatkowska, G. Norman, and D. Parker (2009). "Abstraction refinement for probabilistic software". In: *Verification, Model Checking, and Abstract Interpretation*. Berlin, Heidelberg, pp. 182–197.

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

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

Kwiatkowska, M. Z., G. Norman, and R. Segala (2001). "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. London, UK, UK, pp. 194–206. ISBN: 3-540-42345-1.

Maggio, M., E. Bini, G. Chasparis, and K.-E. Årzén (2013). "A game-theoretic resource manager for rt applications". In: *Euromicro Conference on Real-Time Systems*. DOI: 10.1109/ECRTS.2013.17.

Nagaoka, T., A. Ito, K. Okano, and S. Kusumoto (2011). "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. DOI: 10.1587/transinf.E94.D.958.

Norman, G., D. Parker, M. Kwiatkowska, and S. Shukla (2005). "Evaluating the reliability of nand multiplexing with prism". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **24**:10, pp. 1629–1637. ISSN: 0278-0070.

Ozay, N., U. Topcu, R. M. Murray, and T. Wongpiromsarn (2011). "Distributed synthesis of control protocols for smart camera networks". In: *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pp. 45–54.

Pedreiras, P. and L. Almeida (2003). "The flexible time-triggered (FTT) paradigm: an approach to qos management in distributed real-time systems". In: *Proceedings International Parallel and Distributed Processing Symposium*. DOI: 10.1109/IPDPS.2003.1213243.

Ramos, N., D. Panigrahi, and S. Dey (2007). "Dynamic adaptation policies to improve quality of service of real-time multimedia applications in ieee 802.11e wlan networks". *Wirel. Netw.* **13**:4, pp. 511–535. DOI: 10.1007/s11276-006-9203-5.

Rinner, B. and W. Wolf (2008). "An introduction to distributed smart cameras". *Proceedings of the IEEE* **96**:10. DOI: 10.1109/JPROC.2008.928742.

Seetanadi, G. N., J. Cámara, L. Almeida, K.-E. Årzén, and M. Maggio (2017). "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*, pp. 243–254.

Shapley, L. S. (1953). "Stochastic games". *Proceedings of the National Academy of Sciences* **39**:10, pp. 1095–1100. ISSN: 0027-8424.

Shmatikov, V. (2002). "Probabilistic analysis of anonymity". In: *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pp. 119–128.

Silvestre-Blanes, J., L. Almeida, R. Marau, and P. Pedreiras (2011). "Online qos management for multimedia real-time transmission in industrial networks". *IEEE Transactions on Industrial Electronics* **58**:3. DOI: 10.1109/TIE.2010.2049711.

Simaitis, A. (2014). *Automatic Verification of Competitive Stochastic Systems*. Department of Computer Science, University of Oxford.

Toka, L., A. Lajtha, É. Hosszu, B. Formanek, D. Géhberger, and J. Tapolcai (2017). "A resource-aware and time-critical IoT framework". In: *IEEE International Conference on Computer Communications INFOCOM*. DOI: 10.1109/INFOCOM.2017.8057143.

Vandalore, B., W.-c. Feng, R. Jain, and S. Fahmy (2001). "A survey of application layer techniques for adaptive streaming of multimedia". *Real-Time Imaging* **7**:3. DOI: 10.1006/rtim.2001.0224.

Varma, S. (2015). *Internet Congestion Control*. 1st. San Francisco, CA, USA. ISBN: 0128035838, 9780128035832.

Wang, X., M. Chen, H. M. Huang, V. Subramonian, C. Lu, and C. D. Gill (2008). "Control-based adaptive middleware for real-time image transmission over bandwidth-constrained networks". *IEEE Transactions on Parallel and Distributed Systems* **19**:6. DOI: 10.1109/TPDS.2008.41.

Zeng, H., M. D. Natale, P. Giusto, and A. Sangiovanni-Vincentelli (2009). "Stochastic analysis of can-based real-time automotive systems". *IEEE Transactions on Industrial Informatics* **5**:4, pp. 388–401. ISSN: 1551-3203.

Zhang, T., A. Chowdhery, P. (Bahl, K. Jamieson, and S. Banerjee (2015). "The design and implementation of a wireless video surveillance system". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438. DOI: 10.1145/2789168.2790123.

```
// 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 4.7. Properties of SMG models

Paper V

Adaptive Routing with Guaranteed Delay Bounds using Safe Reinforcement Learning

Gautham Nayak Seetanadi Karl-Erik Årzén Martina Maggio

Abstract

Time-critical networks require strict delay bounds on the transmission time of packets from source to destination. Routes for transmissions are usually statically determined, using knowledge about worst-case transmission times between nodes. This is generally a conservative method, that guarantees transmission times but does not provide any optimization for the typical case. In real networks, the typical delays vary from those considered during static route planning. The challenge in such a scenario is to minimize the total delay from a source to a destination node, while adhering to the timing constraints. For known typical and worst-case delays, an algorithm was presented to (statically) determine the policy to be followed during the packet transmission in terms of edge choices. In this paper we relax the assumption of knowing the typical delay, and we assume only worst-case bounds are available. We present a reinforcement learning solution to obtain optimal routing paths from a source to a destination when the typical transmission time is stochastic and unknown. Our reinforcement learning policy is based on the observation of the state-space during each packet transmission and on adaptation for future packets to congestion and unpredictable circumstances in the network. We ensure that our policy only makes safe routing decisions, thus never violating pre-determined timing constraints. We conduct experiments to evaluate the routing in a congested network and in a network where the typical delays have a large variance. Finally, we analyze the application of the algorithm to large randomly generated networks.

1. Introduction

This paper describes an application of reinforcement learning to the problem of routing in networks where each edge can be represented by a very conservative upper bound on the delay to traverse it, but the typical delay experienced when traversing edges can differ from its upper bound.

Context. A recent paper [Baruah, 2018] introduced the problem of determining routes in graphs in which the delays across edges are characterized by both conservative upper bounds and typical values. The problem solved in the paper is to determine a route that from an initial source node i traverses edges of the graph and reaches a destination node t minimizing the delay under typical circumstances and preserving guarantees of not exceeding a total budget for the transmission, denoted with D_F. Compared to *static* routing, in which a decision on the entire route that the transmitted packet should follow, is taken prior to setting out from the source, the paper introduces *adaptive* routing, in which the decision on which edge to traverse next in the graph is taken online, based on information about the elapsed transmission time. Adaptive routing is in general capable of achieving smaller typical delays compared to static routing. This is because adaptive routing uses knowledge of the delay that was experienced across already traversed edges to aid future decisions.

The adaptive routing technique presented in [Baruah, 2018] is based on the construction of tables that at each vertex determine which outgoing edge should be taken for intervals of experienced delays. In the presented solution, these tables are built once and then the complexity of selecting the outgoing edge from the current vertex only depends on the number of rows that each of these tables has, which in turn depends on the topology of the network and on the number of potential time intervals.

In our work, we use *safe* reinforcement learning to combine optimal path finding with safe state-space exploration. This reduces the size of the tables stored at each vertex. The de-centralized approach allows each vertex to make routing decisions irrespective of the future delays while respecting end-to-end delay constraints.

Notation. In the remainder of this paper, we use the following notation. We consider a graph G, where the vertices represent nodes and the edges represent a direct path between two nodes. We denote with $\mathcal{V}$ the set of vertices, and with $\mathcal{E}$ the set of edges of G. We use the notation $e : (x \rightarrow y)$ to indicate edge e, connecting node x with node y. Each edge is characterized by two numbers, a worst-case transmission time c^W_{xy} and a typical experienced delay c^T_{xy}, dropping the arrow for simplicity. We use c_{xyt} to denote the minimum worst case delay from the node x to the destination t when choosing edge $(x \rightarrow y)$.

Problem Statement. Our problem is to determine a policy to route messages from a source vertex $i \in \mathcal{V}$ to a destination vertex $t \in \mathcal{V}$, to minimize (typical) transmission time for the message, and ensure that the total delay experienced

by the message, τ does not exceed a specified final deadline value, D_F, $\tau \leq D_F$. We aim at solving this problem in a de-centralised way despite changes in the typical experienced delays that can happen during run-time.

Contribution. In this paper, we argue that estimates of typical delays are unlikely to be available prior to exploration of the network behavior and these typical delays are in general time-varying. To address this problem in adaptive network, we develop an algorithm based on *reinforcement learning* (RL) to automatically (and safely) perform adaptive routing. The complexity of computing the outgoing edge for each vertex only depends on the number of outgoing edges from the vertex. In general, this does not exceed the complexity of the adaptive routing technique proposed in [Baruah, 2018], where tables could have repeated entries (the same outgoing edge can be selected for different intervals of experienced delay).

Our proposal automatically adjusts to events happening in the network (like a link suddenly becoming less reliable or congested). In this paper we show that the policy identified by the reinforcement learning algorithm exposes *stochastic convergence* to the optimal policy identified by the algorithm presented in [Baruah, 2018] when typical delays are experienced. We also show that when typical delays are represented by probability distributions the policy learns to take the variance of the delays into account. Finally we show that our algorithm works to minimize transmission times even in large networks.

Compared to [Baruah, 2018], we show that our work has low computational and storage complexity while improving network adaptation to time-varying typical delays. Compared to classical reinforcement learning (RL), our work guarantees safety bounds. We show that with proper domain knowledge, powerful reinforcement learning techniques can be used for time critical real-time applications.

Paper Organization. The remainder of this paper is organized as follows. In Section 2 we present our algorithm, explaining the necessary background on reinforcement learning techniques and our choices. Section 3 discusses an experimental evaluation showing the shortcomings of previous contributions when typical delays are not fixed and how our algorithm overcomes them. We also show that the identified policy in our case stochastically converges to the optimal routing technique proposed in [Baruah, 2018]. Section 4 discusses related work and Section 5 concludes the paper.

2. Algorithm

As in previous work [Baruah, 2018], we distinguish between a *pre-processing* phase and a *run-time* phase. The pre-processing phase is used for the definition of the safe bounds, i.e., to ensure that information about the worst-case paths is propagated to each node. The run-time phase is used for the execution of the reinforcement learning algorithm [Sutton and Barto, 2018]. During run-time the

system explores safe routes in order to build and update a policy that determines the best action to take depending on the currently experienced packet delay.

2.1 Pre-processing phase

The aim of the pre-processing phase is to determine the possible worst-case delays experienced by a packet in the network. More precisely, given an edge $e : (x \rightarrow y)$, we need to compute a safe bound for the worst-case delay to the destination t experienced by a packet that exits x via e. This can be done simply by determining the minimum worst-case delay to the destination from vertex y, c_{yt}, and then adding the worst-case delay of the edge e, c^W_{xy} . Assuming that the worst-case delays are positive numbers, we can simply use Dijkstra's shortest path algorithm [Dijkstra, 1959; Mehlhorn and Sanders, 2008] to determine these values. This gives us the delay bound for which transmission can be guaranteed for each edge.

Figure 1 shows an example[1] of a network and the corresponding generated state space $\mathcal{S}$. On the top left side, we show the network and the delays over each edge. The typical delays, c^T_{xy} are denoted in blue and the the worst-case delays, c^W_{xy} are shown in red. Below that we show the network computed by the application of the Dijkstra's shortest path algorithm for each edge. The worst-case delay to the destination t over each edge, c_{xyt} is shown in green. On the right side, we show the state space. For clarity, the figure does not display the edges between nodes. The state space is explored in detail in the following subsection.

We argue that this pre-processing phase is necessary regardless of the choice of run-time algorithm, to determine safe bounds for the network and avoid choosing an edge that may lead to violation of the worst-case delay constraint.

2.2 Run-time phase

During the run-time phase, a policy to select a path to send a packet from origin i to destination t is determined. This is accomplished using reinforcement learning. In particular, we model our problem using a Markov Decision Process (MDP).

An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is a set of finite states, $\mathcal{A}$ is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a function that encodes the probability of transitioning from state s to state s' as a result of an action a, and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the choice of action a determines a transition from state s to state s'. We use actions to encode the selection of an outgoing edge from a vertex.

State Space. In our state, we encode both the current vertex we are in and the elapsed time from the beginning of the packet transmission (in time units). The set of possible states $\mathcal{S}$ is then the Cartesian product between the set of vertices $\mathcal{V}$ and the set of natural numbers that are less than the deadline D. Denoting with

[1] The example is based on the presentation of [Baruah, 2018], where it was used to illustrate the problem.

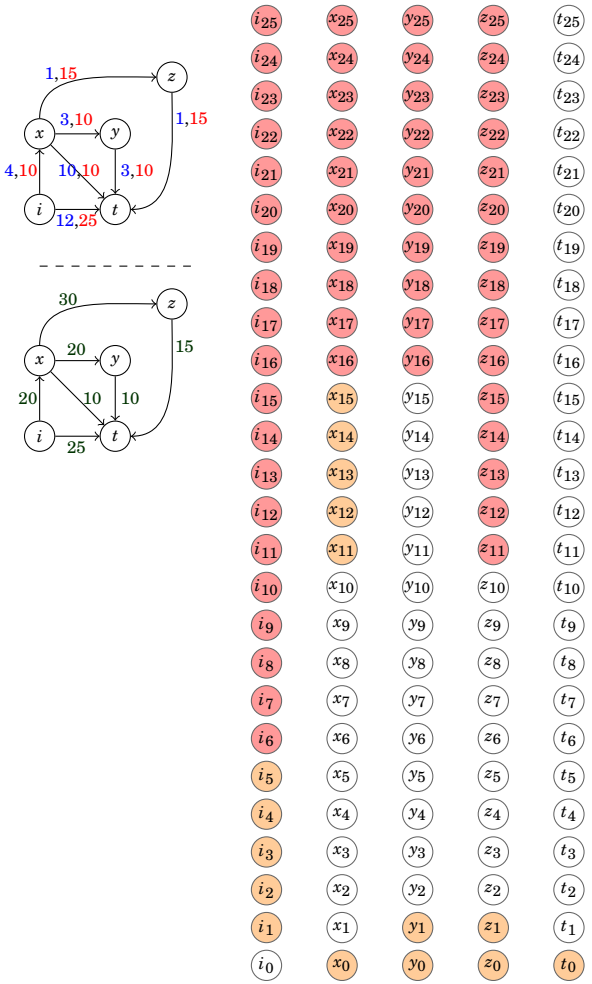


Figure 1. Example of graph and corresponding state space for the reinforcement learning problem formulation.

N the set of integer values that satisfy the deadline constraint, i.e., $N = \{n \in \mathbb{N} \mid 0 \leq n \leq D\}$ then the set of possible states is

$$\mathcal{S} = \mathcal{V} \times N = \{(v, n) \mid v \in \mathcal{V} \wedge n \in N\}.$$

We use the compact notation v_n to denote the state (v, n).

In the graph from figure 1, the initial source node i is connected to the destination node t with an edge having typical transmission time 12 (in blue) and worst-case transmission time 25 (in red). Alternatively, node i is connected to

node x with a typical transmission time of 4 time units and a maximum delay of 10 time units. The maximum admissible delay D_F is 25 time units.

On top of the state space, we want to enforce the constraint that we only explore safe paths, i.e., paths that cannot lead to a violation of the worst-case transmission time requirement D_F. For example, in node x choosing the edge $(x \rightarrow t)$ leads us to the destination node in 10 time units in the worst case, choosing the edge $(x \rightarrow y)$ leads us to the destination in 20 time units in the worst case, and choosing $(x \rightarrow z)$ has a worst case delay of 30 time units. Therefore, it is not safe in the example to be in state x_{20}, as there is no path to the destination that allows us to *certainly* reach t in less than D_F time units. We mark the unsafe states in red.

Finally, not all the nodes of the state space $\mathcal{S}$ are reachable. For example, node t_0 is clearly not reachable as there is no way to cross the path from the source node to the destination node in zero time. However, node t_1 is potentially reachable from the source when the edge $(i \rightarrow t)$ is chosen, if the experienced delay is one time unit. Furthermore, we assume that the packets are not stalled, which means for example that node i_1 is not reachable. We mark the unreachable nodes with orange in Figure 1.

There are still some edges that we should eliminate among the choices. For example, if we are in state x_{10}, with $D_F = 25$, we need to impede the choice of the edge $(x \rightarrow y)$, as the worst case delay for the single possible path is higher than the 15 time units that are left. We do this by limiting the actions that the policy can choose in each state.

Reinforcement Learning. Once we limit the set of possible actions to ensure we will meet the worst case deadline, we use a reinforcement learning policy to decide which outgoing edge – action $a \in A$ – to take from a state $s = v_n$. Generally speaking, decisions based on reinforcement learning are constructed using feedback to *reward* successful actions taken to explore the state space [Sutton and Barto, 2018]. This decision making process, or *policy*, leads to a formal definition of the value of being in the current state and how to select the action that is taken in each state. The goal of this decision making process is to take optimal actions so as to maximize the reward obtained. Using the reinforcement learning terminology, we denote the transmission of a message from source to destination with the word *episode*. We also define the reward received in each state as the total amount of time saved while traversing the path to the destination. This reward is calculated at the end of each episode and then propagated to the other nodes.

TD Learning. A popular reinforcement algorithm is the Temporal-Difference (TD) learning [Sutton and Barto, 2018] algorithm. TD learning gained popularity in TD-Gammon, a program that learned to play backgammon at the level of expert human players [Tesauro, 1995].

It is a model-free learning method which learns by sampling the environment and determines an estimate of how good it is for the algorithm to perform an action from a state, i.e., a value function taking action a in state s, usually identified as $Q(s,a)$. This learning method is coupled with an exploration algorithm,

which in our case is the ε-greedy algorithm. Similar to Monte-Carlo (MC) methods, TD learning samples the environment and learns from it, by updating the value function. While MC methods update the value $Q(s)$ of each state s in the current episode at the conclusion of the episode, TD learning adjusts its estimate to match later predictions about the future before the final outcome is known.

Using MC would be impractical for our algorithm as it would generate a lot of messages in the network to transmit reward information at the conclusion of the message transmission. On the contrary, the one step TD algorithm allows us to make routing decision and value iteration based only on $Q(s,a)$ and $Q(s',a')$, i.e., the Q-value for the current and next state. The next state Q-value can be appended to the acknowledgement of the reception of a message, therefore making this choice practical from an implementation standpoint. The value update policy is computed as

$$Q(s,a) = Q(s,a) + \alpha \cdot (\mathcal{R} + \max(\gamma\, Q(s',a')) - Q(s,a)) \tag{4.1}$$

where s and s' are the current and next states respectively, a' is the action with the highest Q-value for state s', α is the learning rate with which we overwrite old information with new, $\mathcal{R}$ is the reward obtained during the transition from state s to state s', and γ is the discount factor that captures uncertainty of $Q(s',a')$.

Exploration Policy. ε-greedy exploration ensures that the system chooses the edge that was identified as best for the transmission of most packets while at the same time exploring other edges in search of a path that has a higher reward. The ε-greedy policy chooses one action from the vector of feasible actions $\mathcal{A}$. Specifically this action a is either the one that has the maximum value of $Q(s,a)$ or a random action that explores the state space. The policy explores the state space with a probability ε and take the action with the maximum estimated reward with a probability of $(1-\varepsilon)$. In principle, ε is chosen to be a small number, such that most of the time the policy exploits knowledge about the current state. The continuous exploration of the state space, ensures (in a probabilistic sense) the detection of paths with higher rewards and resilience to changes.

In order to guarantee that the total transmission time never violates the set deadline D_F, we modify the ε-greedy policy to perform *safe* reinforcement learning. Safe reinforcement learning is the process of learning actions that maximize the rewards while ensuring reasonable system performance and/or respect safety constraint [García and Fernández, 2015]. Safe reinforcement learning can be broadly divided into two categories:

- *Modified Optimization Criterion*: In this case, the concept of risk is introduced into the optimization process. The policy is determined with explicit knowledge about the risk involved in taking specific actions.
- *Safe Exploration Process*: In this second case, the exploration process is modified to avoid exploratory actions that could have harmful consequences. This is the alternative taken in this work, where we impede certain actions from being taken.

In previous work, this knowledge has been used mostly in scenarios where a human teacher demonstrates a safe path [Driessens and Džeroski, 2004; Smart and Kaelbling, 2000]. In this paper, the modified ε-greedy policy ensures a safe exploration process through incorporating external knowledge obtained in the preprocessing phase. The exploration policy then allows for small explorations that deviate from the optimal policy.

A popular variation of this exploration algorithm is the decaying ε-greedy. The decaying algorithm reduces ϵ, thus minimizing the exploration during routing of later packets. This reduces the delay over the network as most of the exploration is required during the transmission of the first few packets. Once the value of most of state action pairs $Q(s,a)$ of the MDP are known, the need for exploration reduces. Tuning the decay function according to the size of the network is considered out of scope of this paper.

The paper investigates both exploration policies in Section 3. We use the decaying ε-greedy policy in Experiment 3.1 as a static network is considered. In the following experiments, we use the non-decay ε-greedy due to probabilistic distributions of delays and dynamic networks with congestion.

2.3 Algorithm

Algorithms 4, 5 and 6 show the pseudo-code of our algorithm. Algorithm 4 is executed globally to obtain worst-case transmission times c_{it} during network initialization and reconfiguration(node addition or removal). Algorithm 5 executed at the node can either be run globally or at a local level depending upon the network configuration. Monte-Carlo methods require knowledge of the entire state-space and thus are practical for small networks. Algorithm 5 describes one step TD-learning making routing decision and value iteration based only on $Q(s,a)$ and $Q(s',a')$, i.e., the Q-value for the current and next state allowing for distributed decision making.

Consider the routing problem from source i to destination t. As described above, the algorithm is split into pre-processing and run-time phases. The preprocessing phase calculates the shortest path to the destination using Dijkstra's algorithm [Dijkstra, 1959]. The minimum worst case transmission time c_{it} to the destination t over the edge $(i \rightarrow x)$ is obtained by adding the worst case transmission time over the edge c^W_{ix} and the minimum worst case transmission time c_{x_t} from v to destination t over all outgoing edges of x. The values of all state-action pairs are initialized to 0.

Algorithm 5 is run at each node u when a packet arrives. If u = source node, the deadline $D_u = D_F$ is set to the final deadline. For the other nodes, the deadline is determined online. For each edge $(u \rightarrow v)$ from u, if $c_{uvt} > D_u$, the edge is infeasible and $P(u|v) = 0$. This ensures that deadlines are never violated. From the feasible edges $(\mathcal{F})$, the edge that corresponds to the action with maximum value, v such that $Q(u,v*) = \max(Q(u, v \in \mathcal{F}))$ has the highest probability $P(u|v*) = (1-\epsilon)$. The remaining feasible edges are assigned $P(u|v \setminus \{v*\}) = \epsilon/(\text{size}(\mathcal{F}-1))$. The next node is decided according to the probability P. The actual transmission time over the edge is δ_{ux}. This is subtracted from the dead-

Algorithm 4 Pre-Processing:

```
1: for each node u do
2:     for each edge (u → v) do
3:         // Delay bounds as described in Section 2.1
4:         c_uvt = c^W_uv + min(c_v_t)
5:         // Initialise the Q values to 0
6:         Q(u, v) = 0
```

line for the node D_u to obtain D_x, the deadline for the next node x. No reward is awarded in the middle of the episode and the value of the state action pair, $Q(u,v)$ is calculated using (4.1).

The reward R after each edge traversal is obtained as shown in Algorithm 6. If $v \neq t$, the destination is not reached and the episode continues by taking the node v as the current node. If $v = t$, the episode is complete. The reward, R is calculated as $R = D_F - \delta_{it}$ where D_F is the pre-determined deadline and δ_{it} is the total transmission delay over the whole path from source i to destination t.

The back-propagation of calculated reward is inherent in the value iteration as shown in Equation 4.1. As seen, the value iteration is dependent on the maximum Q-value of the next node. This can be sent to the sending node after each successful packet transmission depending on the protocol used. For example, If Transmission control protocol (TCP) is used, the acknowledgement messages could be extended to include the Q-value. In our implementation, we simply transmit the value back to the origin node.

2.4 An Example Episode

Using the same example as above with deadline $D_F = 25$, we begin the episode at node i. The feasible available edges are $[(i \rightarrow x),(i \rightarrow t)]$ as they both satisfy the constraint $c_{it} \leq D_i$ where D_i is the deadline at node i. As this is the first episode, there exists no information of the value of the two edges. Both edges have the same probability and one of them is chosen at random. If the edge chosen is $(i \rightarrow x)$ and the observed transmission time is $\delta_{ix} = 4$ then the new deadline D_x is $D_i - \delta_{ix} = 21$. The feasible edges from node x are $[(x \rightarrow y),(x \rightarrow t)]$. The value of the state action pair $Q(i,x)$ is calculated using Equation (4.1). As the value at the next node x is 0 due to no prior information, another random selection of the edge is made. If $(x \rightarrow t)$ is traversed with $\delta_{xt} = 8$, the destination node is reached and the episode is terminated. The reward for the state is calculated as $R = D_F - \delta_{it}$ where δ_{it} is the total transmission time of the packet. During the following transmissions, the algorithm makes use of this state-action value function to make a more informed decision. As we use ϵ-greedy algorithm for exploration, all state-action pairs are eventually explored and their value is calculated. This combination of TD learning and safe exploration leads to small transmission times while respecting time constraints.

Algorithm 5 Node Logic (u)

for Every packet **do**
  if u = source node i **then**
    $D_u = D_F$ // Initialise the deadline
    $\delta_{it} = 0$ // Initialise total delay for packet = 0
  for each edge $(u \rightarrow v)$ **do**
    if $c_{uvt} > D_u$ **then** // Edge is infeasible
      $P(u|v) = 0$
    else if $Q(u,v) = max(Q(u, a \in A))$ **then**
      $P(u|v) = (1-\epsilon)$
    else
      $P(u|v) = \epsilon/(size(\mathcal{F}-1))$
  Choose edge $(u \rightarrow v)$ with P
  Observe δ_{uv}
  δ_{it} += δ_{uv}
  $D_v = D_u - \delta_{uv}$
  R = Environment Reward Function(v, δ_{it})
  $Q(u,v)$ = Value iteration from Equation (4.1)
  if $v = t$ **then**
    DONE

Algorithm 6 Environment Reward Function(v, δ_{it})

Assigns the reward at the end of transmission
if $v = t$ **then**
  $R = D_F - \delta_{it}$
else
  $R = 0$

3. Evaluation

In this section, we will discuss the behavior and performance of the algorithm presented in Section 2, by applying it to the example shown in Figure 1 in different network conditions and to large-scale networks.

We build the network using Python and the NetworkX [Hagberg, Schult, and Swart, 2008] package. NetworkX allows us to build Directed Acyclic Graphs (DAGs) and to provide information about nodes and edges of those DAGs. In building the example of Figure 1, we annotate the graph using for each edge $(x \rightarrow y)$ the worst case delay as a weight. We also include the typical delay c^T_{xy} for each edge as an annotation in the graph, but we hide this information from the algorithm and only use it to determine the edge traversing time. The preprocessing phase calculates the worst case delay from each node to the destination. This initial phase is executed once during the creation of the network. Once the network is created, our reinforcement algorithm is executed for every packet.

Table 1. Optimal Path for Different Deadlines

D_F	Optimal Path	Delays [Baruah, 2018]	Average Delays (1000 episodes)
15	Infeasible	-	-
20	{i,x,t}	14	14
25	{i,x,y,t}	10	10.24
30	{i,x,y,t}	10	10.22
35	{i,x,z,t}	6	6.64
40	{i,x,z,t}	6	6.55

The experiment presented in Section 3.1 shows a comparison between the results obtained with our algorithm and the optimal choices. It highlights that the transmission times experienced using our algorithm converge to the optimal numbers obtained with the technique presented in [Baruah, 2018].

However, the algorithm presented in [Baruah, 2018] does not adapt to online changes. We show how our algorithm behaves in this case in the experiment presented in Section 3.2. In this case, a link gets congested, and its traversal time then becomes equal to the worst case. We show how our algorithm is able to dynamically adapt and converge to a new optimal policy.

However, the real contribution of this paper is shown when edge traversal times are unknown and vary over time. We show this in the experiments presented in Section 3.3. For the rest of the experiments, we assume that the edges traversal times behave according to a given probability distribution. This is similar to a real network where the delay for a packet is varied at every time instance. In particular, we investigate both a truncated normal distribution and a uniform distribution. With truncated normal distribution we mean a probability distribution that is a normal distribution, but is cut at zero (to avoid negative traversal times) and at the worst case traversal times (to ensure that the constraint is satisfied).

Finally the experiment presented in Section 3.5 shows the scalability of our algorithm and how it behaves when applied to an networks with an increased number of nodes.

3.1 Experiment 1: Convergence to the Optimal Route

To check that our reinforcement algorithm identifies the optimal route from the source to the destination node, we compare the length of optimal path determined by the RL algorithm after 1000 episodes, with the traversal time of the optimal path computed using the algorithm from [Baruah, 2018]. Both algorithms are applied to the network shown in Figure 1.

The delays for traversing an edge are set to the typical delays, thus giving us the possibility to verify convergence in nominal conditions. For the RL, the delay to traverse an edge becomes available only after the destination node of the edge is reached.

Table 1 shows the results we obtained for different values of D_F. When D_F = 15, it is impossible to find a solution that guarantees the worst case transmission delay, and both our algorithm and the optimal one presented in [Baruah, 2018] are able to identify that using Dijkstra's algorithm on the worst case transmission times. For the other deadlines, we show the delays obtained with the optimal algorithm and the average delay obtained in 1000 transmissions using our algorithm. In all cases, the delays experienced by packets using algorithm are very close to the optimal delays. The slight variations are due to the exploration that is built in our algorithm, and specifically to the exploring nature of the ϵ-greedy policy. For all the values of D_F and in all episodes, the deadline is never violated.

Figure 2(a) shows the evolution of the transmission delays as the number of packets sent increases. Rapid routing[Baruah, 2018] and safe RL converge to the same path and experience the similar transmission times. Classical RL is consistently able to have low transmission times at the expense of taking unsafe paths. This causes deadline violations in networks with varying transmission times are seen in section 3.4.

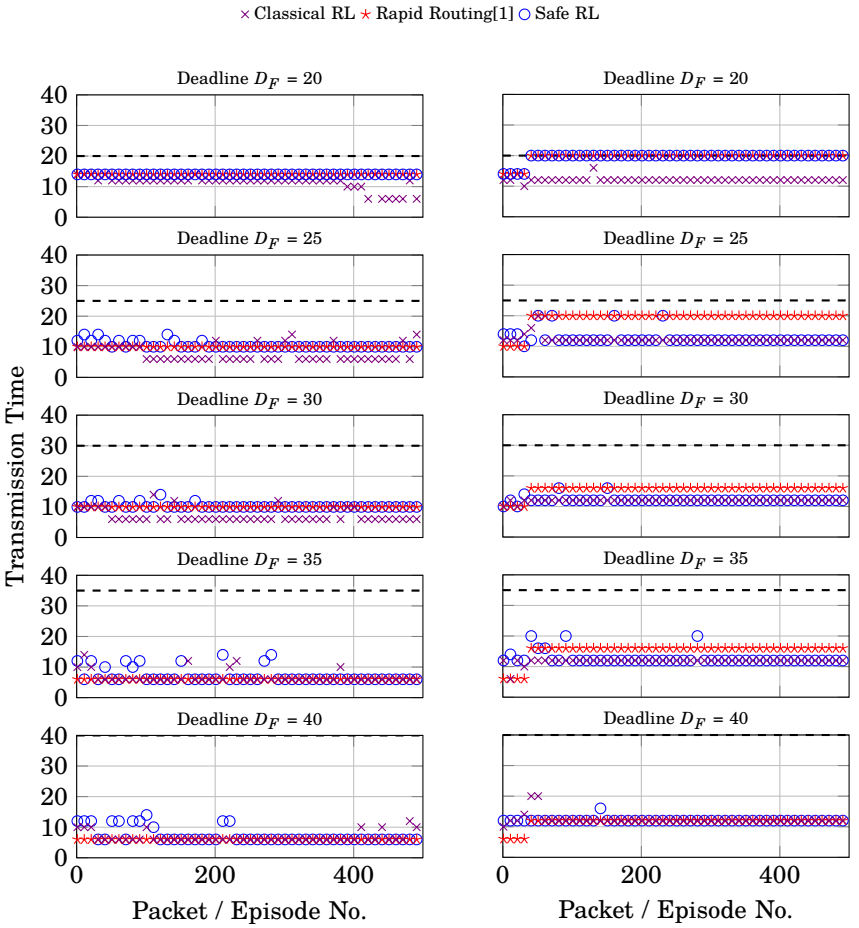
Except for $D_F = 20$, in all the other plots there is an initial exploration phase, in which the safe RL algorithm is exploring alternative routes, to find the optimal path. There is no exploration when $D = 20$ as the only feasible path in the network is $(i \rightarrow x \rightarrow t)$. Therefore the total path delay is always equal to 14. As mentioned before, the algorithm explores new routes with a probability that decays over time, which shows how the algorithm settles for a given route in the static case.

3.2 Experiment 2: Adaptation to Edge Congestion

In Experiment 2 we analyze the capability of the algorithm to adapt to new circumstances. In this specific case, we see how the algorithm reacts to the congestion of one of the edges. In the network we used before, we artificially introduce congestion on the edge $(i \rightarrow x)$. The delay δ_{ix} to traverse this edge increases from 4 to 10 time units, after the transmission of 40 packets. For the rest of the experiment, this edge remains congested.

Figure 2(b) shows the transmission times for different deadlines D_F. Due to the congestion, there is a need to adapt.

Both classical and safe RL adapt to the congestion and take a different path to the destination. For $D_F = 20$, classical RL again chooses a path that violates safety guarantees. Rapid Routing is at a disadvantage as the routing tables generated in the pre-processing stage are not sufficient to ensure adaptation. The delay increases (see the values the algorithm converges to compared to the values shown in Figure 2) and the optimal path changes. The congested edge is the first edge traversed for all previously determined optimal paths, therefore the algorithm has to determine if there is a better path using exploration. Eventually, the rewards are propagated and the algorithm adapts. In all cases except for $D_F = 20$, the algorithm converges to the path $(i \rightarrow t)$ with a total delay of 12. In the case of $D_F = 20$, the only feasible path is $(i \rightarrow x \rightarrow t)$ and the total transmission time on the path increases from 14 to 20 due to the experienced congestion.



(a) No Variance (b) Congested Network

Figure 2. Smoothed Total Delay for Experiments 3.1 and 3.2.

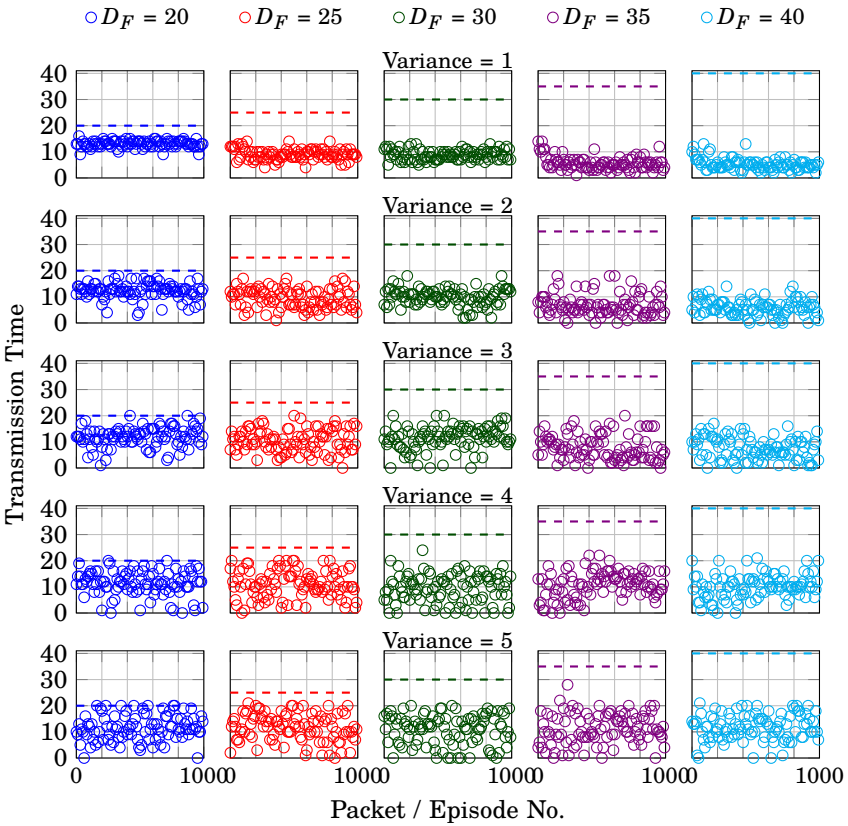


Figure 3. Transmission Time with Truncated Normally Distributed Edge Traversing Times.

The ability to adapt to current conditions is one of the motivations for using our algorithm rather than a static adaptive choice.

3.3 Experiment 3: Probabilistic Traversal Times

In this section, we discuss the convergence of our algorithm when the delay times for transmitting over the edges $(x \rightarrow y)$ are random variables δ_{xy} drawn from probability distributions. In this experiment we model the typical delay times using truncated normal and uniform distributions. Figure 3 shows the total path delays when the traversal time for an edge is distributed as a truncated normal distribution with mean value c^T_{xy} and different variance values. Each column in the figure shows a different value of D_F and each row a different variance, from 1 to 5. We denote the used probability distribution as truncated normal, because we cut the probability distribution below 0 and above the worst case c^W_{xy} to ensure

Table 2. Average Delays(1000 Episodes) in Experiments 3.1, 3.2, and 3.3

D_F	No Variance	Congested Network	Truncated Normal Distribution	Uniform Distribution around c^T	Uniform Distribution $[0, c^W]$
20	14.00	19.76	[12.80, 12.26, 12.02]	[13, 12.6, 12.49]	9.14
25	10.30	12.42	[9.15, 10.20 ,10.63]	[13.01, 12.69, 12.49]	9.84
30	10.27	12.17	[8.92, 10.01, 10.23]	[8.89, 9.04, 10.62]	10.28
35	06.94	12.42	[5.81, 7.15, 8.60]	[5.51, 5.95, 6.95]	9.98
40	06.84	12.32	[6.08, 6.945, 8.556]	[5.59, 5.81, 6.42]	10.12

that the traversal times respect the constraints of our problem ($0 \leq \delta_{xy} \leq C^W_{xy}$).

As the variance increases, the total delay also increases on average, as expected. One take away message is the deadline D_F is never violated, showing that the algorithm behaves correctly. Also, when the variance increases, different paths are explored as the Q values for the nodes tend to be closer to one another – rather than experiencing one best path, the variance blurs the differences between the paths and makes it more important and rewarding to distribute packets on different edges. In particular, looking at the case with $D_F = 35$, the case with variance 1 and 2 converges (primarily) to one specific path. The case with variance 3 and 4 explore different paths and reach different conclusions on the optimality of the chosen policy. The case with variance 5 tends to converge to a different optimal policy. This seems to suggest that the presence of high variance is another reason to use an adaptive policy rather than an optimal predetermined choice.

Similarly Figure 4 shows the results we obtain when δ_{xy} is drawn from a uniform distribution. Specifically, for each edge $(x \rightarrow y)$, $(0 \leq \delta_{xy} \leq c^W_{xy})$ is enforced and the extremes of the uniform distribution are chosen to be centered about the typical delay value with a varying interval length. Similar to the truncated normal distribution case, the network adapts and our algorithm ensures that the deadlines are never violated. Compared to the truncated normally distributed case, the uniform distribution seems to have a better effect on finding optimal routes and sticking to these routes.

3.4 Experiment 4: Uniformly Distributed Worst Case Traversal Times

We perform an experiment with a uniform distribution in which we select the extremes of the distribution as 0 and the worst case transmission delay (completely disregarding the information about the typical transmission time). In this case, the interval length creates much more variation in the typical traversal times. The results for this run are shown in Figure 5.

We compare the transmission times obtained using our safe reinforcement learning approach to the ones using classical RL and the rapid routing algorithm from [Baruah, 2018]. Classical RL algorithms are only concerned with maximizing the obtained reward, thus lead to deadline violations as seen in the figure. The violations occur due to the exploration of unsafe edges. The algorithm from[Baruah, 2018] does not violate deadlines but the typical transmission times

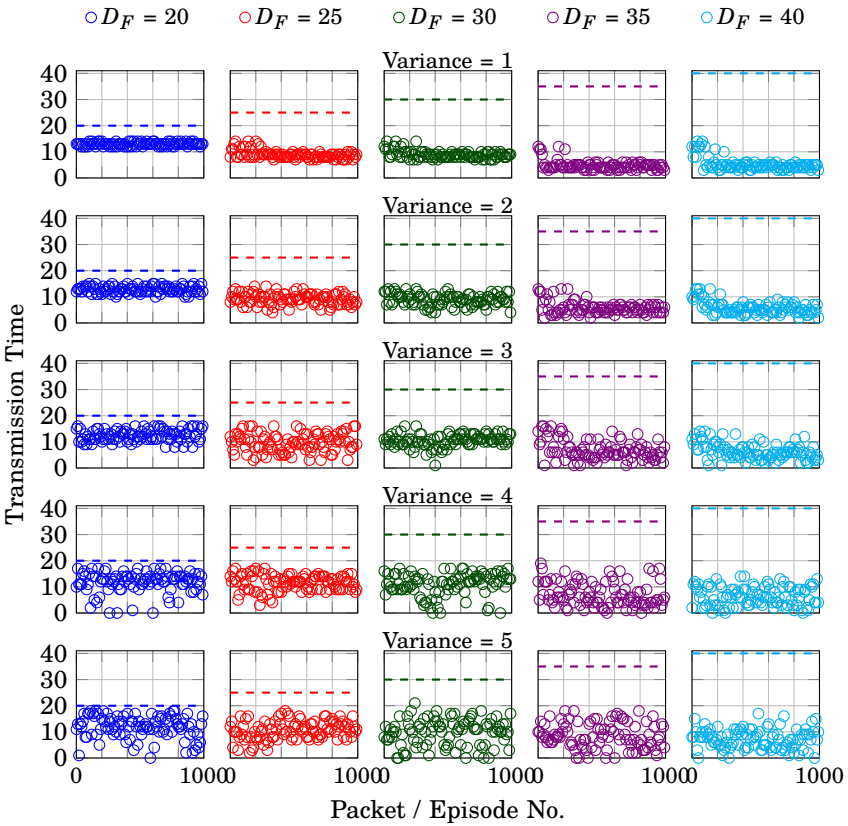


Figure 4. Transmission Time with Uniformly Distributed Edge Traversing Times.

are higher compared to our safe reinforcement learning approach. This is because the routing tables are built only once during network creation. The routing is not adaptive to the changing environment and routes messages according to the pre-built routing tables. One way of compensating would be to rebuild the routing tables for [Baruah, 2018] for every packet. This would be highly compute intensive and impractical as seen in experiment 3.5

Our algorithm is shown to work in meeting the deadlines D_F and also discovering new potential paths as the traversal times over the initial links in the path vary. Table 2 summarizes the delays experienced in the network for the three algorithms.

3.5 Experiment 5: Large Networks

In this last set of experiments we investigate the scalability of our algorithm. We create random networks, with a large number of nodes n. We ensure the network

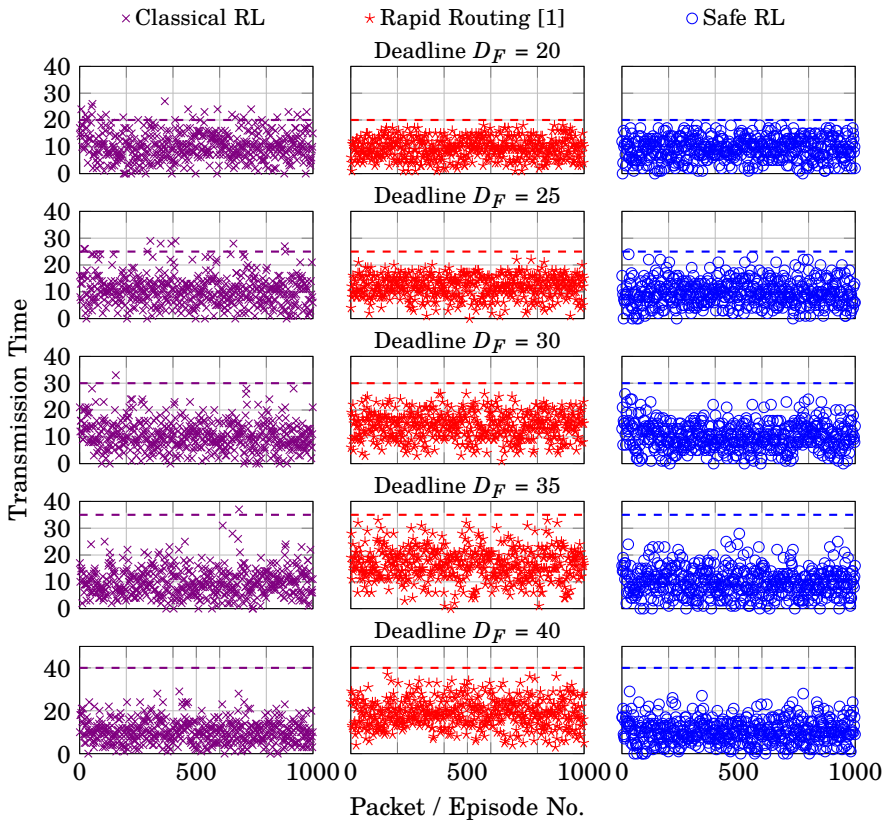


Figure 5. Transmission Time with Uniformly Distributed Edge Traversing Times for Large Intervals.

includes one edge $(0 \to 1)$ from node 0, the initial node, and one edge $(n-2 \to n-1)$ that reaches the final node $n-1$. We randomize all the other edges present in the network. We only add edges from a node with a lower index node to a higher one while ensuring no loops are created in the networks. Generally the networks generated consist of nodes with a large number of outgoing edges.

For every edge in the network, $(x \to y)$, we randomly extract a value for the typical delay $c^T_{xy} \in (0,10]$ and for the worst case delay $c^W_{xy} \in [10,30]$. We ensure that there exists a path from every node x to the destination node n. Networks that do not satisfy the condition are discarded.

The pre-processing phase is applied as described in Section 2 to all networks to get the shortest guaranteed total delay to the destination node n. The deadline for the randomly generated networks is chosen to be the $1.2 \cdot c_{it}$ from the initial

Table 3. Average Delays(1000 Episodes) in Section 3.4

D_F	Classical RL	Rapid Routing	Safe RL
20	10.32	9.609	9.135
25	10.19	11.98	9.841
30	10.10	14.10	10.283
35	10.23	16.43	9.99
40	9.98	18.593	10.193

node i to the destination node t.

Figure 6 shows the average delay time and the deadline set for increasing number of nodes in the randomly generated network. We send 1000 packets over the network from source to destination and record the total delay. The algorithm works well finding paths to the destination minimizing total delays while ensuring no deadline violations, even for large networks. Classical RL has higher transmission times generally because of the large number of outgoing edges from each node and deadlines are violated during exploration. Rapid routing [Baruah, 2018] has low transmission times due to the routing tables built in the pre-processing stage. However this method has very high computational complexity and the routing tables have to be recalculated for every change in δ.

Figure 7 shows the computational time for the transmission of 1000 packets over the networks. We compare the performance of classical RL and rapid routing algorithm from [Baruah, 2018] with our safe reinforcement learning. Classical RL is the least computationally complex as it has no pre-processing stage. Our safe learning approach is a magnitude less computationally intense compared to rapid routing. In safe RL the pre-processing has to be only run once during network creation. In case of node addition, the recalculation is minimal. Comparatively, rapid routing has large routing tables have to be constructed and evaluated as described in [Baruah, 2018].

4. Related Work

In this section, we discuss prior research related to our work.

The problem of optimal paths in a stochastic network has been studied previously in [Polychronopoulos, 1992], [Loui, 1983], and [Bertsekas and Tsitsiklis, 1991]. These prior works consider the stochastic and dynamic shortest distance problems in a weighted network and minimize a cost function to obtain the optimal path. The weights in this case can be considered to be equal to the delays in the network. Our work builds on these by guaranteeing an upper bound on the delay while minimizing the weight. We also assume no prior knowledge on the distributions and also take into account the dynamic nature of real networks.

Networking systems are of great interest for the application of machine learning algorithms [Peshkin and Savova, 2007]. Actor-critic method [Tong and Brown,

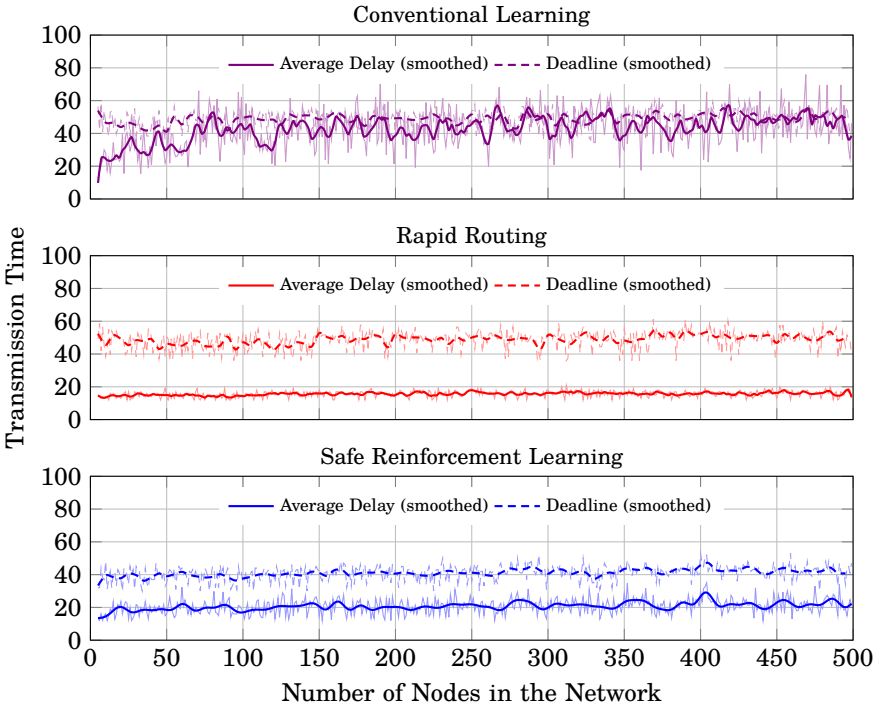


Figure 6. Delays for increasing nodes in network.

2002] and value function methods based on gain scheduling method [Carlström, 2000] have investing in depth on using modern computing advances for better adaptive routing. Similar to these works, we also use reinforcement learning but perform safe exploration to respect delay constraints.

Using external knowledge for safe reinforcement learning is a popular method as it enhances learning by using prior information (derived from human supervisor or otherwise) and prohibits exploration of unsafe paths. This can be broadly distinguished into three methods [García and Fernández, 2015]. *Providing initial knowledge* can mitigate exploration problems using bootstrapping as shown in [Driessens and Džeroski, 2004]. A similar methodology of restricting exploration space is to have a *finite set of demonstrations* to discover the state space. These methods have been applied mainly to physical systems as shown in [Tang, Singh, Goehausen, and Abbeel, 2010] and [Abbeel, Coates, and Ng, 2010]. Finally *Providing advice* uses a teacher to assist during the learning process. This can either be a teacher offering advice when the learner considers necessary as shown in [A. Clouse and E. Utgoff, 1992] and [Garcia and Fernandez, 2014]. Similarly, the teacher can offer advice whenever it deems necessary as shown in [Vidal,

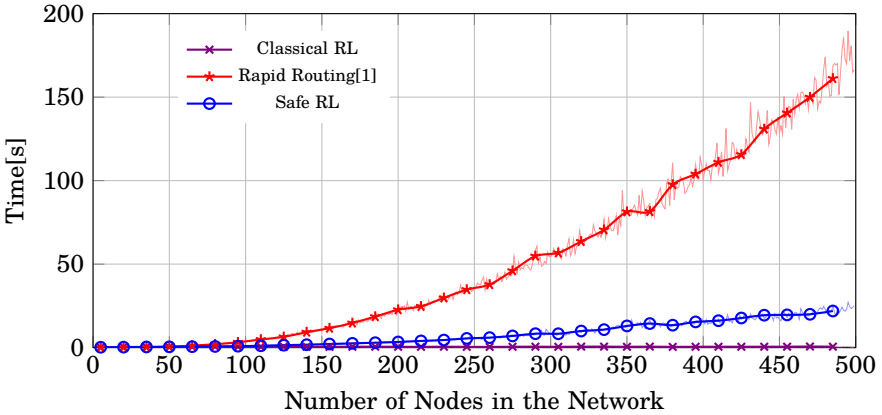


Figure 7. Computational times for increasing the number of nodes in the network.

Rodríguez, González, and Regueiro, 2013], [Thomaz and Breazeal, 2006]. These methods are not very effective in our problem as we consider a decentralized approach with each node making independent decisions.

5. Conclusion

In this paper we have applied reinforcement learning to the problem of routing over real-time networks. To guarantee packet transmission within a predetermined timing constraint, we augment the exploration of the state-space to only explore paths that are safe. This allows for small delays over the network. The constant evaluation of the state-space and exploration of new paths make the algorithm resilient to changes in the network due to, e.g., congestion, link failures. This also allows us to route packets over new paths which might not be realized to be safe in the case of static routing. The decentralized approach used here allows each node to make routing decisions based only on the current observed packet delay and the value function of the next node reducing the amount of information needed at each node.

We verified the stochastic convergence of the algorithm to the optimal path and performed experiments to verify the resilience of the reinforcement learning algorithm. Compared to classical RL we show that our algorithm is robust and never violates set deadlines. While compared to previous research, our algorithm is shown to be more adaptive to transmission time variations while having reduced computational complexity.

References

A. Clouse, J. and P. E. Utgoff (1992). "A teaching method for reinforcement learning". In: *Proceedings of the Ninth International Conference on Machine Learning*, pp. 92–110. DOI: 10.1016/B978-1-55860-247-2.50017-6.

Abbeel, P., A. Coates, and A. Y. Ng (2010). "Autonomous helicopter aerobatics through apprenticeship learning". *Int. J. Rob. Res.* **29**:13, pp. 1608–1639. DOI: 10.1177/0278364910371999.

Baruah, S. (2018). "Rapid routing with guaranteed delay bounds". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. Nashville, TN, USA. DOI: 10.1109/RTSS.2018.00012.

Bertsekas, D. P. and J. N. Tsitsiklis (1991). "An analysis of stochastic shortest path problems". *Math. Oper. Res.* **16**:3, pp. 580–595. DOI: 10.1287/moor.16.3.580.

Carlström, J. (2000). "Decomposition of reinforcement learning for admission control of self-similar call arrival processes". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. MIT Press, Denver, CO, pp. 989–995. URL: http://dl.acm.org/citation.cfm?id=3008751.3008895.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**:1, pp. 269–271. DOI: 10.1007/BF01386390.

Driessens, K. and S. Džeroski (2004). "Integrating guidance into relational reinforcement learning". *Machine Learning* **57**:3, pp. 271–304. DOI: 10.1023/B:MACH.0000039779.47329.3a.

Garcia, J. and F. Fernandez (2014). "Safe exploration of state and action spaces in reinforcement learning". *CoRR* **abs/1402.0560**. arXiv: 1402.0560. URL: http://arxiv.org/abs/1402.0560.

García, J. and F. Fernández (2015). "A comprehensive survey on safe reinforcement learning". *Journal on Machine Learning Research* **16**:42, pp. 1437–1480. URL: http://jmlr.org/papers/v16/garcia15a.html.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). "Exploring network structure, dynamics, and function using networkx". In: Varoquaux, G. et al. (Eds.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, pp. 11–15. URL: https://www.researchgate.net/publication/236407765_Exploring_Network_Structure_Dynamics_and_Function_Using_NetworkX.

Loui, R. P. (1983). "Optimal paths in graphs with stochastic or multidimensional weights". *Commun. ACM* **26**:9, pp. 670–676. DOI: 10.1145/358172.358406.

Mehlhorn, K. and P. Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox*. 1st ed. Springer. ISBN: 9783540779773.

Peshkin, L. and V. Savova (2007). "Reinforcement learning for adaptive routing". *CoRR* **abs/cs/0703138**. arXiv: cs/0703138. URL: http://arxiv.org/abs/cs/0703138.

Polychronopoulos, G. H. (1992). *Stochastic and Dynamic Shortest Distance Problems*. PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA.

Smart, W. D. and L. P. Kaelbling (2000). "Practical reinforcement learning in continuous spaces". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00, pp. 903–910. URL: http://dl.acm.org/citation.cfm?id=645529.657958.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press. ISBN: 9780262039246.

Tang, J., A. Singh, N. Goehausen, and P. Abbeel (2010). "Parameterized maneuver learning for autonomous helicopter flight". In: *2010 IEEE International Conference on Robotics and Automation*, pp. 1142–1148. DOI: 10.1109/ROBOT.2010.5509832.

Tesauro, G. (1995). "Temporal difference learning and td-gammon". *Commun. ACM* **38**:3, pp. 58–68. DOI: 10.1145/203330.203343.

Thomaz, A. L. and C. Breazeal (2006). "Reinforcement learning with human teachers: evidence of feedback and guidance with implications for learning performance". In: *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*. AAAI'06. Boston, Massachusetts, pp. 1000–1005. ISBN: 978-1-57735-281-5. URL: http://dl.acm.org/citation.cfm?id=1597538.1597696.

Tong, H. and T. X. Brown (2002). "Reinforcement learning for call admission control and routing under quality of service constraints in multimedia networks". *Machine Learning* **49**:2, pp. 111–139. DOI: 10.1023/A:1017924227920.

Vidal, P. Q., R. I. Rodríguez, M. Á. R. González, and C. V. Regueiro (2013). "Learning on real robots from experience and simple user feedback". *Journal of Physical Agents* **7**:1, pp. 57–65. DOI: 10.14198/JoPha.2013.7.1.08.

A. Routing Path analysis

This appendix contains more experiments that analyse the paths taken by the algorithm.

Figure 8 shows the best path determined by the algorithm and the path taken during the transmission of packets through the network for different deadlines and for increasing uniform variance.

The first column shows the best and chosen paths when $D_F = 20$. For low variances, the only possible path is $\{i, x, t\}$. This ensures that the deadline is not violated. For higher variances in the transmission times, new paths are available for transmission of the packets. For Variance = 4, new path $\{i, x, y, t\}$ is feasible and is explored around packet number 200. The algorithm determines that this is the best path ($\{c^T_{xy} + c^T_{yt}\} \lessapprox c^T_{xt}$) for transmission. Even though the algorithm discovers a better path, it is not always feasible due to the variance in the transmission

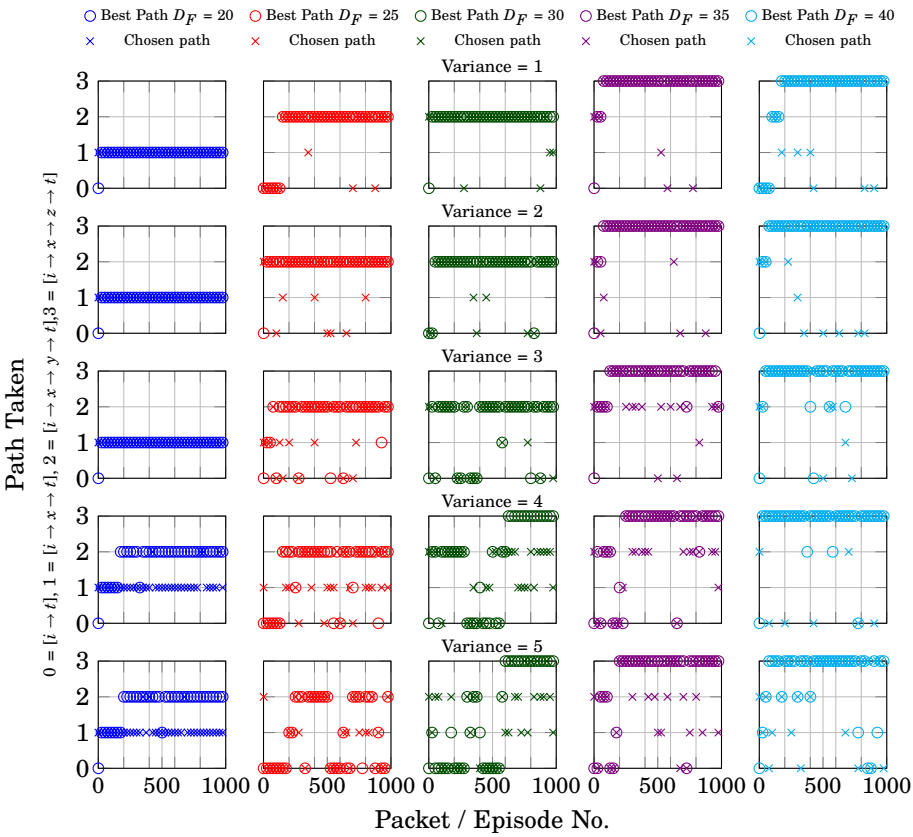


Figure 8. Best Path and Chosen Path with Uniformly Distributed Edge Traversing Times.

Table 4. Algorithm Properties Comparison

	Classical RL	Rapid Routing[1]	Safe RL
Safety Guarantees	No	Yes	Yes
Pre-Processing Stage	No	Need to be run for every δ change	Run only during structural changes
Exploration	ϵ-greedy	-	Safe ϵ-greedy
Routing	Based on probabilities	Routing table dependent	Based on probabilities
Storage at each node	One Q-value for each outgoing edge	Static tables	One Q-value for each outgoing edge

times. The path $(x \rightarrow y)$ is only feasible if $(i \rightarrow t)$ is traversed with $\delta_{it} = 0$[2], as this guarantees that the deadline will not be violated if $(x \rightarrow y)$ is traversed $(c_{xt} = 20)$. A similar phenomenon is observed when $D_F = 30$, enabling the traversal of the path $\{i, x, z, t\}$ leading to small transmission times for most packets.

In all cases, higher variance increases the uncertainty in path selection while making it possible to explore new paths that could potentially lead to shorter delays. This highlights the need for a dynamic routing strategy in real networks.

B. Algorithm Comparisons

Table 4 shows general characteristics of safe RL compared to the previous works. Comparing the three algorithms shows some similarities but also the major areas in which the algirthms differ.

[2] $\delta = 0$, is not feasible in real networks. We enable 0 transmission times here for ease of explanation

Paper VI

Adaptive Routing for Real-Time Networks with Dynamic Deadlines using Safe Reinforcement Learning

Gautham Nayak Seetanadi

Abstract

A great influx of available computing resources has given rise to a large number of smart devices in everyday life. These smart devices perform a variety of tasks from sensing to computation while communicating with each other to solve complex problems. This combination of devices are increasingly used for time-critical real-time applications and require guarantees on their end-to-end communication delays to ensure system correctness. We consider a large network of interconnected nodes with the goal of reaching a destination node t, from a given source node i, while ensuring that the total delay is less than a pre-determined deadline D_F. Each node in the network is an independent computational node that decides the outgoing edge. Each edge in the network is characterized by two types of delays. A worst case upper bound delay that is never violated between successive transmissions, and a typical delay that captures the current load of the network. This dual-delay model captures the behavior of networks with time-varying while also require end-to-end guarantees.

In this paper, we transmit packets through the dual-delay network from source i to destination t using *safe* reinforcement learning (RL). Our reinforcement learning based packet routing policy uses a pre-processing algorithm to ensure packet transmission only over safe paths. We build on our previous work by proposing algorithms for safe network topography changes through node additions and deletions. We evaluate the performance of our safe RL algorithm on two different networks with varying deadlines by comparing the performance of our algorithm with classical RL based routing and previous work on dynamic routing. We also discuss the viability of the different routing algorithms and their application to different network conditions.

Conference Manuscript under Review.

1. Introduction

Internet-of-Things(IoT) devices and applications are dependent on reliable and timely data transmission. This is essential with increased use of edge devices for latency sensitive real-time applications [Shi, Cao, Zhang, Li, and Xu, 2016]. Edge devices perform sensing and transmit information to a more capable computation center in a timely fashion. This time-criticality is important especially for robots and control computations performed in the cloud. Thus real-time networks require robust guarantees on end-to-end transmission times of packets under varying network load in the presence of multiple paths for data transmission.

Recent papers [Baruah, 2018], [Nayak Seetanadi, Årzén, and Maggio, 2020] represent real-time networks as a collection of nodes and edges. The nodes/edge devices in the network have minimal computational capability and decide the outgoing edge for transmission on arrival of a packet. The delays over each link vary over time as the nodes are mobile [1].

The edges in network are thus characterized with two types of delays:

- c^W: Worst case delays obtained by performing worst case timing analysis of the edge.
- $c^T \in (0, c^W]$: Typical delays that vary depending on the current network load.

This dual-delay model captures the behavior of a variety of IoT networks. Consider for example an autonomous car that has to navigate between various points of interest in traffic. Worst case delays, c^W are used to provide end-to-end delays on navigation time of the car. These worst case guarantees do not consider the current traffic conditions leading to conservative delays. Similarly, An edge device has to transmit a compute heavy job to a server with the choice of multiple paths to the destination. The job timing guarantees are given considering links that have better c^W leading to conservative transmission times. We consider packet transmission as the goal for the rest of the paper for simplicity.

The problem is to travel from a specified source node i to a specified destination node t in the network represented by the two-delay model. The chosen path minimizes typical delays while guaranteeing that the end-to-end transmission time δ_{it} is lower than the pre-specified end-to-end deadline D_F. Such network optimization can be achieved by either using static or dynamic routing. Static routing involves making decisions on the entire path of transmission at the source whereas dynamic routing makes decisions at each node considering previous delays encountered. Dynamic routing is shown to be the optimal choice for routing even with the drawback of higher computational needs [Polychronopoulos, 1992]. The high computational requirement is mitigated by advances in compute capacity of edge nodes.

[1] We assume connections never fail and are return worst case delays in extreme conditions.

Previously, dynamic routing problem has been solved for real-time networks though building dynamic routing tables at each node[Baruah, 2018]. This approach provides safety guarantees but the static routing tables do not adapt to changes in typical delays over links. In our work, we relax the assumption that typical delays are known a priori to transmission. Packets are routed through the network using *safe* reinforcement learning. This is similar to the approach in [Nayak Seetanadi, Årzén, and Maggio, 2020] where optimal paths are realized through safe state-space exploration.

Contributions of the paper:

- In comparison to our previous work [Nayak Seetanadi, Årzén, and Maggio, 2020], we
  - Give analytical safety guarantees on end to end delays for all packets
  - Propose algorithms for safe node additions and deletions
  - Adapt to variations in deadlines D_F
- In comparison to the work done by authors in [Baruah, 2018], we
  - Propose algorithms for safe node additions and deletions
  - Adapt to variations in deadlines D_F
  - Adapt to variations in c^T

1.1 Notation

We use the following notations for the remainder of the paper. We consider a directed graph $G = (\mathcal{N}, E)$ where nodes $\mathcal{N}$ are points of interest given in order as $(N_1.....N_n.....N_{\mathcal{N}})$. The source and destination nodes are denoted as N_i and N_t respectively for simplicity. Each link $(N_{n-1} \rightarrow N_n) \in E$ between two nodes N_{n-1} and N_n has the following properties,

- $c^T_{(n-1)n}$: Typical delay
- $c^W_{(n-1)n}$: Worst case delay
- $c_{(n-1)nt}$: Minimum worst case delay to destination t from node N_{n-1} guaranteed via edge $(N_{n-1} \rightarrow N_n)$. This is obtained after pre-processing as explained in Section 3

Similarly $\min(c_{n\_t})$ denotes the minimum guaranteeable worst case transmission time to destination t over all outgoing links from node N_n. In addition to this we denote the actual transmission time over the link $(N_{n-1} \rightarrow N_n)$ as $\delta_{(n-1)n}$. δ_{it} denotes the total transmission time for the packet from source i to destination t. D_F denotes the pre-determined final deadline for the packet and D_n denotes the deadline for each packet at node N_n.

We use *transmission time* and *delay* interchangeably through the paper.

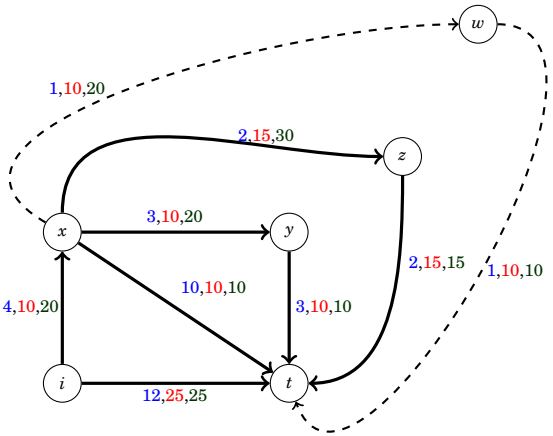


Figure 1. Case study network as described in [Nayak Seetanadi, Årzén, and Maggio, 2020] and presentation of [Baruah, 2018]

1.2 Outline of the paper

The remainder of the paper is organized as follows. In section 2 we present two models of networks built to analyse our algorithm, giving an introduction to reinforcement learning techniques and state-space evolution. Section 3 presents our algorithms for safe node addition and deletion. Section 4 provides analytical safety guarantees on the behavior of our algorithm. Section 5 discusses an experimental evaluation comparing our algorithm to related work. Section 6 gives a short discussion on implementation aspects of our algorithm. Section 7 discusses related work and Section 8 concludes the paper.

2. Model

In this section we describe the two different networks built to analyse our algorithm. Next we give a short introduction to reinforcement learning and explain the exploration-exploitation trade-off. Then we describe the evolution of the state space during exploration by reinforcement learning.

2.1 Network Models

Figure 1 shows an example network consisting of directed edges with typical delays $c^T_{(n-1)n}$ denoted in blue and the worst case delays $c^W_{(n-1)n}$ denoted in red for each edge $(N_{n-1} \rightarrow N_n)$. The model was introduced by the authors during the

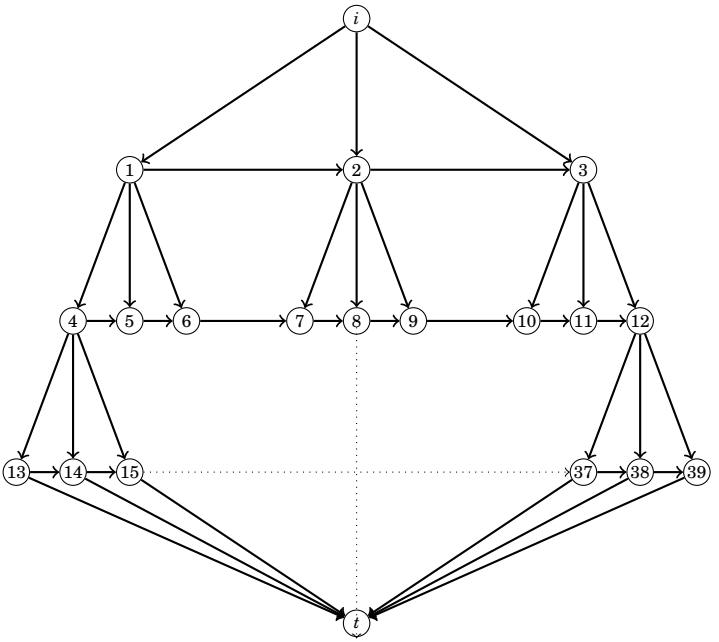


Figure 2. Modified Tree Network

presentation of [Baruah, 2018] and used to compare routing algorithms in [Nayak Seetanadi, Årzén, and Maggio, 2020]. The network is simple with few nodes but it captures the decision uncertainty in dynamic routing. For large deadlines D_F, links with a large difference in c^T and c^W (small c^T and large c^W) are ideal for packet transmission due to large c_{t} to destination t. Similarly for small D_F, links with smaller difference are ideal. The presence of these different links leads to complexity in routing decisions for varying packet deadlines.

Node w is a mobile node dynamically added to the network during experiments to evaluate dynamic network configurations.

Figure 2 shows a modified tree network with multiple paths from source i to destination t. The tree network is dense in nature with multiple possible paths from each node ensuring there are no loops present. The large network is representative of a complex network with multiple paths to destination. The edge delays are chosen pseudo-randomly with **either**

- Low c^T and high c^W ($c^T \in [1,5]$, $c^W \in [10,15]$). **or**
- Similar c^T and c^W ($c^T \in [10,20], c^W \in [10,25] \ni c^T \leq c^W$).

2.2 Markov Decision Process

The goal of efficient routing of packets from source i to destination t with minimal delays is accomplished using reinforcement learning(RL) in this paper. Most RL problems model their state-space using Markov Decision Processes (MDPs). A MDP is defined as a 4-tuple $(\mathcal{S}, \mathcal{A}, P_a, R_a)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is set of actions, $P_a : (s, a, s') \rightarrow \{0 \leq p \leq 1\}$ is the probability of choosing an action a from state s resulting in the state s', R_a is the instantaneous or delayed reward obtained after performing action a.

A modified MDP for the safe routing problem is defined as a 4-tuple $(s \in \mathcal{S}, a \in \mathcal{A}, P_a, R_a)$ where,

- $s \in \mathcal{S}$: State s belonging to the finite state space S. In the network model, we encode the current node N_n and current deadline D_n as individual states in the state-space. The set of possible states S is the Cartesian product between the set of adjacent nodes N_{n+1} from node N_n, $\mathcal{V}_{n,n+1}$ and the set of integers $\mathbb{Z}$ less than the deadline D_F. The set of possible states at each node N_n is then given by,

$$\mathcal{S}_n = \mathcal{V}_{n,n+1} \times D_F = \{(v, n) | v \in \mathcal{V}_{n,n+1} \wedge n \in \mathbb{Z}\}$$

- $a \in \mathcal{A}$: Action a belonging to the finite action space $\mathcal{A}$. For the routing problem, $\mathcal{A}$ are the outgoing edges from a node.
- $P_{a \in A}$: Probability of choosing an action, the outgoing edge a.
- $R_{a \in A}$: Reward obtained after action a. RL reward are either instantaneous, to evaluate optimality of each edge, or obtained at the end of packet transmission to evaluate optimality of the entire path. In this paper we have implemented delayed reward assignment to minimize computation at nodes.

2.3 Reinforcement Learning

Safe RL has a smaller state-space compared to classical RL as unsafe states are not explored. The constructed safe state-space is used to evaluate routing paths with lower delays. RL is the learning technique mapping current system state to actions, maximizing a reward. The learning agent is not programmed with particular actions to execute, it rather discovers the best action by performing them. The RL agent performs multiple episodes to evaluate actions by starting over numerous times. This is ideal for routing networks with the presence of constant packet transmissions and changing network conditions.

In the routing network, the RL agent decides the outgoing edge from each node in the network. This decision process is called *policy* and the continuous evaluation of state-action pairs leads to optimization of the routing policy.

2.4 Policy

The policy dictates value iteration of the states in the system. The choice of policy depends on various parameters of the routing problem such as, the time horizon

for obtaining the, node computation power, node storage capacity and so on. In our implementation we aim for minimal computation at each node and reward calculation at the final node propagated to the other nodes using temporal difference(TD) learning.

TD learning has emerged as a popular RL algorithm owing to its success with superhuman level performance in Backgammon, Chess and various Atari games [Tesauro, 1995] [Badia et al., 2020]. TD learning updates it estimates without waiting for packet to arrive at the destination leading to faster convergence. We calculate the reward at the destination t to minimize computation and back propagate the reward through value iteration. The value iteration function is given by

$$Q(s,a) = Q(s,a) + \alpha \cdot (\mathcal{R} + \max(\gamma\, Q(s',a')) - Q(s,a)) \tag{4.1}$$

where $Q(s,a)$ is the value of being in state s and taking action a. $\mathcal{R}$ is the reward obtained during the state transition and $\max(Q(s',a'))$ is the maximum reward obtained previously from the next state s'.

Learning Rate, α α, Also called *step-size* determines the amount of old learning overwritten. It is tuned depending of the probability of variations in the transmission times, δ. If $\alpha = 0$, the node makes routing decisions depending only upon the previous available information. Setting $\alpha = 1$, overwrites old information completely.

Discount Factor, γ γ captures the uncertainty in future rewards obtained on choosing action a. If $\gamma = 0$, the node is biased towards short term rewards minimizing delays to the next edge. Small γ evaluates edge decisions without taking into account the optimality of the whole path to destination. $\gamma = 1$ makes routing decision based on the whole path chosen for packet transmission.

2.5 Exploration

ϵ-greedy exploration policy ensures that the system chooses the outgoing edge for packet transmission from the vector of outgoing edges based on probabilities. ϵ-greedy algorithm chooses the edge identified as the best edge for transmission with a probability $(1-\epsilon)$. The policy explores other edges in search of a path that returns higher rewards with a probability ϵ. The probability of choosing an outgoing edge $(n+1) \in \mathcal{A}$ from node N_n to N_{n+1} for is given by the function

$$f(P) = \begin{cases} 0 & \text{if edge is unsafe} \\ 1-\epsilon & \text{if best edge} \in \mathcal{F} \\ \epsilon/(size(\mathcal{F}-1)) & \text{Otherwise} \end{cases} \tag{4.2}$$

An edge $(N_n \rightarrow N_{(n+1)})$ is unsafe if $c_{n(n+1)t} > D_n$. The best safe edge from a node N_n is one with the corresponding maximum Q value. Q value, $Q((n,D_n),n+1)$ is the value that determines the optimality of choosing the edge $(N_n \rightarrow N_{(n+1)})$ from node N_n with deadline D_n. The other feasible edges are explored with probability ϵ.

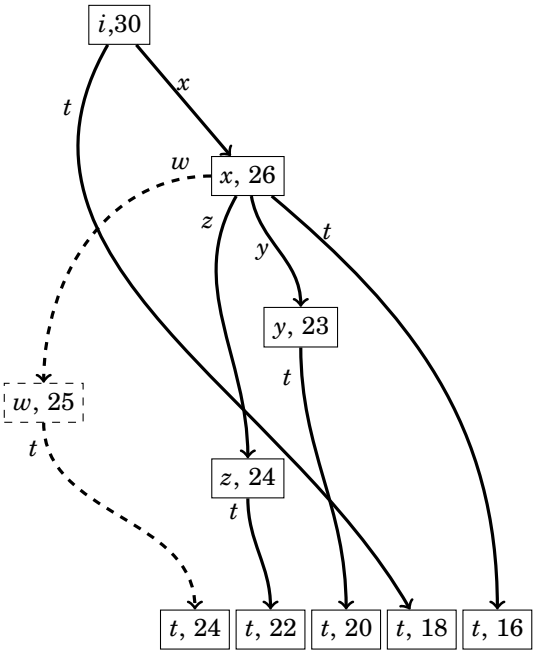


Figure 3. State-space evolution with $D_F = 30$ and $\delta_{n(n+1)} = c^T_{n(n+1)}$. Dashed lines show the state-space after node w addition.

Choosing a small ϵ ensures that known information is *exploited* for maximum reward whereas a large ϵ, *explores* newer paths in search of higher reward. The choice between maximizing known rewards and exploring unknown paths is known as the exploration-exploitation trade-off and is an inherent part of reinforcement learning [Sutton and Barto, 2018] [Habib, Arafat, and Moh, 2019] [Tesauro, 1995].

2.6 Reduced State Space

Q-learning uses state-action pairs to evaluate the value of being in a current state of the system. Each node consists of its own state-action pairs that are explored to find routing paths that result in reduced packet delays. The states in our system consist of both the current vertex and the time elapsed from packet transmission. The set of possible states $\mathcal{S}$ at node n is the Cartesian product of $\mathcal{V}_{n,n+1}$, the set of outgoing vertices and $N \leq D_n$, the set of Natural numbers less than or equal to the deadline at the current node n. This modified formulation of the state-space allows for decentralised state-space exploration in comparison to [Nayak Seetanadi, Årzén, and Maggio, 2020].

Figure 3 shows the evolution of the state-space with $D_F = 30$ and $\delta_{n(n+1)} = c^T_{n(n+1)}$ for the network from Figure 1. Each block in the figure shows states (n, D_n)

Algorithm 7 Pre-Processing:

1: **for** each node N_n **do**
2: **for** each edge $(N_n \rightarrow N_{n+1})$ **do**
3: // Dijkstra's weighted shortest path to destination t
4: $c_{n(n+1)t} = c^W_{n(n+1)} + \min(c_{(n+1)\_t})$

and each outgoing edge denotes the corresponding chosen edge $N_{(n+1)}$.

The different branches of the state-space are explored using Q-learning during packet transmission. We denote each packet transmission as an episode and each episode terminates when the destination node t is reached. RL uses continuous learning to evaluate the optimality of the transmission path. Each outgoing arrow in the state space shows the chosen edge and connects the current state to the next state. For eg, Choosing edge t from state $s = (i,30)$ results in state $s' = (t,18)$ given $c^T_{it} = 12$ as seen in Figure 1. This constitutes one episode of RL as the packet has reached the destination. The reward obtained for the episode is $\mathcal{R} = D_F - 12 = 18$. The action pair $((i,30),t)$ is updated to the value 18. The remaining state-space is built through exploration during subsequent packet transmissions.

The new states of the system are explored dynamically after node addition due to continuous exploration inherent to RL. In the figure, state $(w,25)$ is explored from state $(x,26)$ as $c_{xwt} > 26$. The example state-space is relatively small due to static transmission times. Variations in D_F and δ result in a large state-space. The *safe* state-space is subset of the largest state-space due to the presence of infeasible states and unsafe states that are not explored. For eg. When $D_F = 20$ and $\delta = c^T$, path $(i \rightarrow t)$ is always infeasible and never explored. Similarly, $(x \rightarrow z \rightarrow t)$ and $(x \rightarrow w)$ are also infeasible due to the restrictive deadline resulting in a smaller state-space.

3. Algorithm

In this section we discuss the underlying algorithm for safe reinforcement learning. First, in section 3.1 we give a short introduction to the algorithm already presented by the authors in [Nayak Seetanadi, Årzén, and Maggio, 2020]. Then we augment their algorithm by including rules for node addition and deletion as shown in section 3.2.

3.1 Routing algorithm from [Nayak Seetanadi, Årzén, and Maggio, 2020]

Our safe RL algorithm consists of a pre-processing algorithm to obtain safety guarantees on packet delays to the destination. Algorithm 7 shows the pseudocode of the pre-processing algorithm. It is executed globally to obtain worst-case transmission $c_{(n-1)nt}$ for each link $(N_{(n-1)} \rightarrow N_n)$ during network initialization. The algorithm is based on Dijkstra's shortest algorithm [Dijkstra, 1959] for weighted graphs and has low complexity.

Algorithm 8 Node Logic (N_n)

1: **for** Every packet **do**
2: **if** N_n = source node i **then**
3: $D_n = D_F$ // Initialise the deadline
4: $\delta_{it} = 0$ // Initialise total delay for packet = 0
5: **for** each edge $(N_n \rightarrow N_{n+1})$ **do**
6: **if** $c_{uvt} > D_u$ **then** // Edge is infeasible
7: $P(n|n+1) = 0$
8: **else if** $Q((n,D_n),n+1) = max(Q((n,D_n),\_)$ **then**
9: $P(n|n+1) = (1-\epsilon)$
10: **else**
11: $P(n|n+1) = \epsilon/(size(\mathcal{F}-1))$
12: Choose edge $(n \rightarrow n+1)$ with P
13: Observe $\delta_{n(n+1)}$
14: δ_{it} += $\delta_{n(n+1)}$
15: $D_n + 1 = D_n - \delta_{n(n+1)}$
16: R = Environment Reward Function$(n, \delta_{n(n+1)})$
17: $Q((n,D_n),n+1)$ = Value iteration from Equation (4.1)
18: **if** $N_{n+1} = t$ **then**
19: DONE

Algorithm 9 Environment Reward Function(N_{n+1}, δ_{it})

1: Assigns the reward at the end of transmission
2: **if** $N_{n+1} = t$ **then**
3: $R = D_F - \delta_{it}$
4: **else**
5: $R = 0$

Algorithm 8 is executed at each node during the arrival of packet. If N_n = source node, the deadline is set to $D_n = D_F$, the final deadline. For the other nodes in the network, the deadline is set online taking the time traversed into account. The choice of outgoing edge is determined by the probability function from Equation 4.2 as shown in lines 6 to 11. The actual transmission time encountered, δ over the edge is added to the total transmission time over the path δ_{it}. δ is also subtracted from the deadline at node N_n and forms the deadline for the next node N_{n+1}. The obtained reward is used to update the value $Q((n,D_n)n+1)$ of the state-action pair.

Algorithm 9 shows the reward $\mathcal{R}$ obtained after edge traversal. If the destination node t is not reached then RL episode continues. If the destination node is reached, $\mathcal{R}$ is calculated as $R = D_F - \delta_{it}$ with D_F is the final deadline for the packet and δ_{it} is the transmission time over the whole path from source i to destination t. The reward is back-propagated inherently to all nodes though value

Algorithm 10 Node addition(N_n)

1: **for** all downstream links ($N_n \rightarrow N_{n+1}$) **do**
2: $c_{n(n+1)t} = c^W_{n(n+1)} + \min(c_{(n+1)\_t})$
3: **for** all upstream links ($N_{n-1} \rightarrow N_n$) **do**
4: **Minimum delay to destination**(N_{n-1}) //Algorithm 11

iteration from Equation 4.1.

3.2 Node Addition

When a new node, N_n is added to the network, the links are added to the appropriate upstream($N_{n-1} \rightarrow N_n$) and downstream nodes($N_n \rightarrow N_{n+1}$). Downstream links are directed from the added node to the nodes already present. Upstream links connect from the already present nodes to the newly added node. Typical transmission times for safe edges in both directions of the added node N_n, $c^T_{(n-1)n}$ and $c^T_{n(n+1)}$ are explored by the algorithm during packet transmission.

Worst case transmission times for downstream links, $c^W_{n(n+1)}$ over each link are local knowledge to the added node N_n and are used in the calculation of worst case times to destination $c_{n(n+1)t}$. $c^W_{(n-1)n}$ for upstream links may impact the worst case transmission time to destination of upstream nodes $c_{(n+1)nt}$.

Algorithm 10 show the pseudocode for node addition. We calculate $c_{n(n+1)t}$ for the downstream links with $\min(c_{(n+1)\_t})$, the worst case guaranteeable time to destination t. For the upstream links we similarly calculate $c_{(n-1)nt}$ for the upstream nodes using $\min(c_{n\_t})$. We verify the shortest path to the destination for the upstream node (N_{n-1} as shown in Algorithm 11.

Algorithm 11 Minimum delay to destination(N_n)

1: old $c_{n\_t} = c_{n\_t}$
2: $c_{n(n+1)t} = c^W_{n(n+1)} + \min(c_{(n+1)\_t})$
3: **if** $c_{n(n+1)t} <$ old c_{n_t} **then** // New delay to t
4: **for** All upstream links ($N_{n-1} \rightarrow N_n$) **do**
5: **Min delay to destination**(N_{n-1})

3.3 Node z Deletion

Any node deletion when the node has the packet will lead to inevitable packet loss and deadline violation. Algorithm 12 shows the pseudocode for deletion of nodes assuming no packets are in transmission. We again differentiate between downstream and upstream links as described in section 3.2. Downstream links do not affect the performance of the algorithm and are safely deleted. Deletion of upstream links ($N_{n-1} \rightarrow N_n$) affects minimum guarantees to the destination. The value of $c_{(n-1)\_t}$ is recalculated and propagated upstream to the appropriate nodes.

Algorithm 12 Node Deletion(N_n)

1: **for** all upstream links ($N_{n-1} \rightarrow N_n$) **do**
2: old $c_{(n-1)\_t} = c_{(n-1)\_t}$ // Save value to check for optimality
3: $P(n-1|n) = 0$
4: Recalculate min($c_{(n-1)\_t}$) without deleted link
5: **if** min($c_{(n-1)\_t}$) > old $c_{(n-1)\_t}$ **then**
6: Recalculate min($c_{(n-1)\_t}$) for upstream nodes

4. Proof

4.1 Proof of Safety

Consider a chosen path from source to destination with $\mathcal{V}$ number of nodes in the following order, $N_1 ....N_{\mathcal{V}}$. Each link ($N_{n-1} \rightarrow N_n$) chosen during transmission has the following transmission times,

- $c^T_{(n-1)n}$: Typical transmission times.
- $c^W_{(n-1)n}$: Worst case transmission time.
- $c_{(n-1)nt}$: Worst case transmission time to destination.

Additionally we denote the actual transmission time over each link ($N_{n-1} \rightarrow N_n$) to be $\delta_{(n-1)n}$. The deadline at each node N_n is denoted as D_n and the deadline at the beginning of each packet transmission at origin as D_F. The deadline at each node N_n is be given by,

$$\begin{aligned} D_2 &= D_1 - \delta_{12} \\ D_3 &= D_2 - \delta_{23} \\ &\cdot \\ &\cdot \\ D_{\mathcal{V}-1} &= D_{\mathcal{V}-2} - \delta_{(\mathcal{V}-2)(\mathcal{V}-1)} \\ D_{\mathcal{V}} &= D_{\mathcal{V}-1} - \delta_{(\mathcal{V}-1)\mathcal{V}} \end{aligned} \tag{4.3}$$

Using the above equations, we can write the deadline at the final node $N_{\mathcal{V}}$ as

$$D_{\mathcal{V}} = D_1 - \{\delta_{12} + \delta_{23} + ... + \delta_{(\mathcal{V}-1)\mathcal{V}}\} \tag{4.4}$$

where d_1 is the deadline at the initial node, or the final deadline D_F of the packet.

$$D_{\mathcal{V}} = D_F - \{\delta_{12} + \delta_{23} + ... + \delta_{(\mathcal{V}-1)\mathcal{V}}\} \tag{4.5}$$

According to the model, we always assume that $\delta_{(n-1)n} \leq c^W_{(n-1)n}$ holds for each edge ($N_{n-1} \rightarrow N_n$). Thus equation 4.5 is rewritten as,

$$d_{\mathcal{V}} + \{c^W_{12} + c^W_{23} + ... + c^W_{(\mathcal{V}-1)\mathcal{V}}\} \leq D_F \tag{4.6}$$

Consider the probabilistic Equation 4.2 that decides edge selection. An safe edge is characterised as $c_{(n-1)n} \leq d_{(n-1)}$. Comparing this and Equation 4.3,

$$\begin{aligned} d_1 &\geq c^W_{12} + \min(c_{2\_t}) \\ d_2 + \delta_{21} &\geq c^W_{12} + \min(c_{2\_t}) \\ d_2 &\geq c^W_{12} + \min(c_{2\_t}) - \delta_{21} \\ d_2 &\geq \min(c_{2\_t}) \end{aligned} \tag{4.7}$$

Thus the algorithm is safe at each node in addition to the overall safety as shown above.

5. Experimental results

In this section we evaluate the performance of our algorithm by comparing our work to Rapid Routing [Baruah, 2018], Classical RL [Sutton and Barto, 2018] and Safe RL [Nayak Seetanadi, Årzén, and Maggio, 2020]. Experiments 1 to 4 are performed on the network shown in Figure 1. Experiment 1 evaluates our algorithm with static deadlines D_F to emphasize the importance of algorithm for node additions. Experiment 2,3 and 4 are performed with varying deadlines D_F. Experiments 5 and 6 are performed on the modified tree network from Figure 2 to show the advantage of using our algorithm in large networks. Table 4 shows average transmission times and number of deadline violations for all experiments.

We represent the networks using directed acyclic graphs (DAGs), built using NetworkX package [Hagberg, Schult, and Swart, 2008] for python. Each link in the network is encoded with the two delays c^T and c^W as weights. The typical delay, c^T is hidden from the algorithm and only realised through exploration. The value of $c_{n(n+1)t}$ for each link $(N_n \rightarrow N_{n+1})$ in the network is obtained with the pre-processing algorithm.

5.1 Experiment 1

We transmit 1000 packets with $D_F \in \{20, 25, 30, 25, 40\}$ from source i to destination t to ensure safety and correctness of our algorithm. Figure 4 shows the total path delay of packets for various deadlines. Node w is added to the network after the transmission of 100 packets to supported algorithms(our work and classical RL) as described in Algorithm 10. No deadlines are violated by any algorithms due to constant delays over all links and constant deadlines. There are also no deadline violations after the addition of node w.

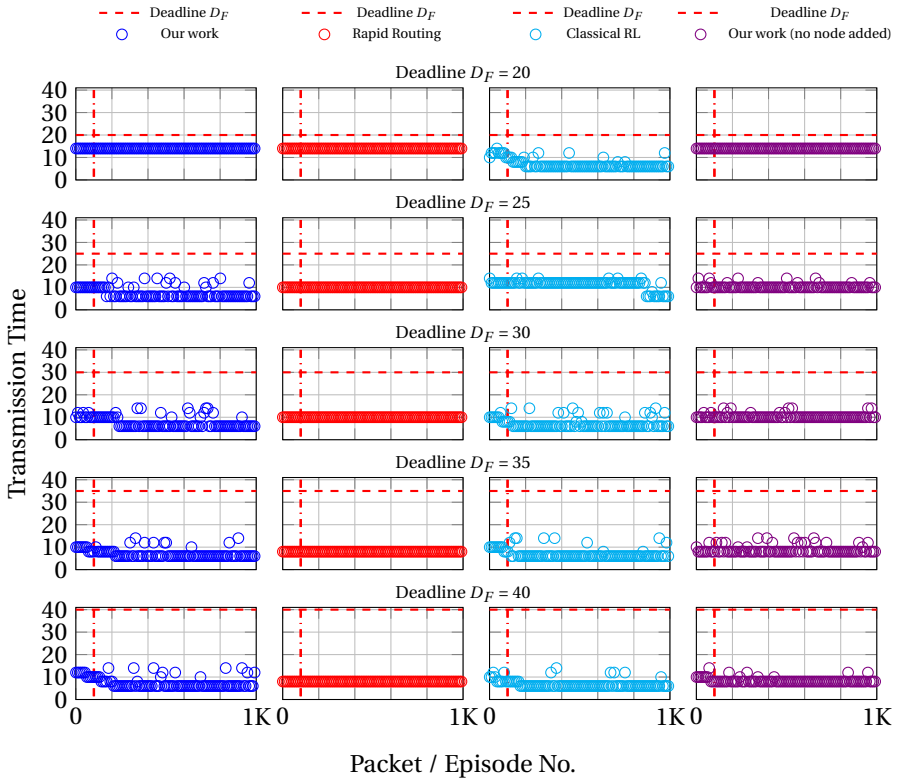


Figure 4. Experiment 1: Transmission times with $D_F \in \{20, 25, 30, 25, 40\}$ and $\delta = c^T$

Rapid routing [Baruah, 2018] transmits packets through the network using pre-built routing tables leading to lower total delays. The routing tables do not require recalculation as the link delays are static. Algorithms based on RL obtain actual link delays through exploration and thus have slower convergence. The exploration phase of RL based algorithms leads to higher delays for some transmitted packets. However this exploration phase is crucial to ensure the algorithms react to network changes.

When $D_F = 20$, $(i \rightarrow x \rightarrow t)$ is the only path with no safety violations. Most algorithms transmit packet only through this path to guarantee safety at each node. Classical RL does not provide any safety guarantees and converges to lower path delays as it chooses unsafe path for transmissions. We also evaluate our algorithm without node addition for delay comparison with Rapid Routing and Safe RL, as the algorithms do not support dynamic network changes.

Table 1 shows Q-tables at the end of state-space exploration. The columns

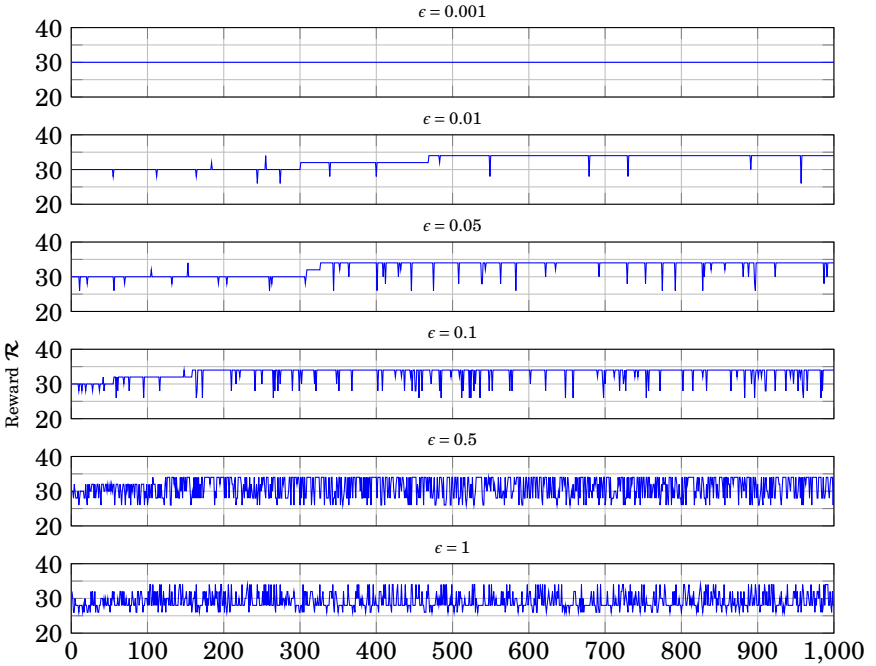


Figure 5. Experiment 1: Rewards obtained for varying ϵ with $D_F = 40$

represent the current node N_n and the rows represent the deadline D_n at the node. Each cell forms the state-action pairs and the current system. The value of each cell shows the action/next node N_{n+1} that returns the most value for the state-action pair. For eg. when $D_F = D_i = 20$, edge $(i \rightarrow x)$ returns the highest reward (the best path when $D_F = 20$ is $(i \rightarrow x \rightarrow t)$). The state-space in Experiment 1 is relatively small in size as D_F and δ are constants.

Choice of ϵ ϵ dictates the rate of exploration of new paths and. The choice of ϵ is dependent on the structure and behavior of the routing network. Figure 5 shows rewards obtained for varying values of ϵ for the network from Figure 1. We evaluate our algorithm with $D_F = 40$ to mark all paths as safe and set $\delta = c^T$.

Setting ϵ to a very small value leads to rewards with low variance with the drawback of lower dynamicity. On the other hand, large values of ϵ lead to convergence problems due to too many exploring paths. We choose $\epsilon = 0.1$ to minimize transmission times ensuring that new paths are regularly explored.

5.2 Experiment 2

In Experiment 2, we set random deadlines with $D_F \in [20, 40]$ for each packet transmission. We transmit 10000 packets over the network from Figure 1 for full state-space exploration. We set $\delta_{n-1,n} = c^T_{n-1,n}$ ensuring that this information

Table 1. Q-value tables after state-space exploration in Experiment 1

Time Rem	i	x	y	z	w
16	-	t	-	-	-
18	-	-	t	-	-
20	x	-	-	-	t
21	-	w	-	-	-
23	-	-	t	-	-
25	x	-	-	-	t
26	-	w	-	-	-
28	-	-	t	-	-
29	-	-	-	t	-
30	x	-	-	-	t
31	-	w	-	-	-
33	-	-	t	-	-
34	-	-	-	t	-
35	x	-	-	-	t
36	-	w	-	-	-
40	x	-	-	-	-

is unknown to the algorithm and is realised through exploration. We also set $\alpha = \gamma = 1$ as δ and the rewards obtained are deterministic. Figure 6 shows total transmission time, δ_{it} for the first 1000 random deadlines. Our algorithm never violates deadlines for any of the packets. If $D_F = constant$ for all packets, this information can be exploited to reduce the size of routing table as seen in the previous experiment.

Table 2 shows the Q-values of a fully explored state-space after Experiment 2 before the addition of the additional node. The state-space is larger compared to the state-space of Experiment 1. Each node in the network encounters a larger number of deadlines due to random deadlines and needs more packet transmissions to fully explore the state-space. Table 3 shows the change in state-space after the addition of node w to the network and transmission of additional packets. The algorithm explores the new paths and allocates a higher reward to path $(i \rightarrow x \rightarrow w \rightarrow t)$ (as $c^T_{xy} > c^T_{xz} > c^T_{xw}$).

5.3 Experiment 3

Figure 6 shows the typical transmission times for 1000 packets when $\delta = U(c^T)$, uniform distribution with variance = 5 in network 1. There are no deadline violations by our algorithm and leads to low delays. Table 4 shows the average delays for all routing algorithms. Rapid Routing has higher delays as it relies on pre-built routing tables and is unable to adapt to varying c^T. This can be mitigated by rerunning the pre-processing algorithm but is computationally heavy [Nayak Seetanadi, Årzén, and Maggio, 2020]. The three RL based algorithms have lower

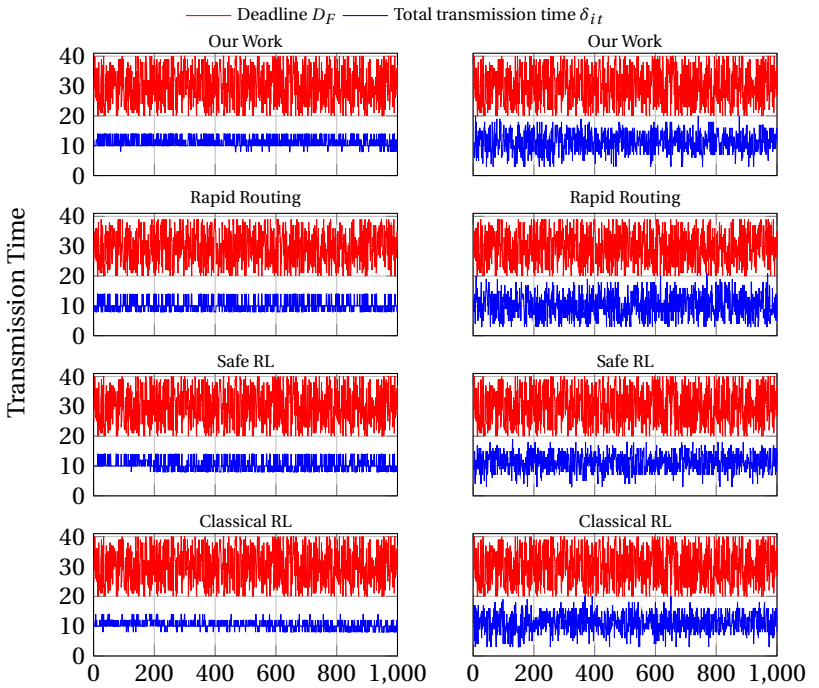


Figure 6. Experiment 2: Transmission times with $D_F \in [20,40]$ and $\delta = c^T$ (left) Experiment 3: Transmission times with $D_F \in [20,40]$ and $\delta = U(c^T)$, variance = 5(right)

transmission times and adapt dynamically to network variations. Classical RL has 14 deadline violations from the transmission of 10000 packets as thus is not suitable for providing safety guarantees.

5.4 Experiment 4

Experiment 4 evaluates the performance of all algorithms under stress. We set $\delta_{n-1,n} \in (0, c^W_{n-1,n}]$, a uniform distribution with extremes of the distribution as 0 and the worst case transmission time. The choice of interval length creates large variations in actual traversal times δ. The transmission times of first 1000 packets are shown in Figure 7. We compare transmission times for the different algorithms. Classical RL algorithms are only concerned with maximizing rewards and lead to 144 deadline violations at the end of 10000 packet transmissions. Safe RL has lower average transmission times due to the smaller state-space compared to our work. This smaller state-space leads to faster convergence and better efficiency in networks will small number of nodes. Rapid Routing has the highest average transmission times highlighting the drawback of using static tables for

Table 2. Q-value tables after state-space exploration in Experiment 2 before addition of node w

Time Rem	i	x	y	z	w
16	-	t	-	-	-
17	-	t	t	-	-
18	-	t	t	-	-
19	-	t	t	-	-
20	x	y	t	-	-
21	x	y	t	-	-
22	x	y	t	-	-
23	x	y	t	-	-
24	x	y	t	-	-
25	x	y	t	-	-
26	x	y	t	-	-
27	x	y	t	-	-
28	x	y	t	t	-
29	x	y	t	t	-
30	x	z	t	t	-
31	x	z	t	t	-
32	x	z	t	t	-
33	x	z	t	t	-
34	x	z	-	t	-
35	x	z	-	-	-
36	x	z	-	-	-
37	x	-	-	-	-
38	x	-	-	-	-
39	x	-	-	-	-
40	x	-	-	-	-

routing.

5.5 Experiment 5

Experiments 1 to 4 showed that our algorithm performs relatively well compared to the state of the art algorithms. The drawback of our algorithm is slow convergence due to the large state-space to be explored. The experiments showed a large state space is not required to efficiently route packets through the example network shown in Figure 1.

To analyse algorithm performance on large networks we route packets from source i to destination t in the modified tree network shown in 2 for experiments 5 and 6. We use Dijkstra's shortest path algorithm [Dijkstra, 1959] to calculate the shortest guaranteeable time to destination t. We set random deadlines such that $D_F \in [1.2 * \text{shortest path}, 2 * \text{shortest path}]$.

Table 3. Q-value tables after state-space exploration in Experiment 2 after addition of node w

Time Rem	i	x	y	z	w
16	-	t	-	-	-
17	-	t	t	-	-
18	-	t	t	-	-
19	-	t	t	-	t
20	x	w	t	-	t
21	x	w	t	-	t
22	x	w	t	-	t
23	x	w	t	-	t
24	x	w	t	-	t
25	x	w	t	-	t
26	x	w	t	-	t
27	x	w	t	-	t
28	x	w	t	t	t
29	x	w	t	t	t
30	x	w	t	t	t
31	x	w	t	t	t
32	x	w	t	t	t
33	x	w	t	t	t
34	x	w	-	t	t
35	x	w	-	-	t
36	x	w	-	-	-
37	x	-	-	-	-
38	x	-	-	-	-
39	x	-	-	-	-
40	x	-	-	-	-

We analyse packet delays and rewards obtained with $\delta_{(n-1)n} = c^T_{(n-1)n}$ for 50000 packets. Figure 8 shows packet transmission times of the first 10000 packets. Our algorithm has high transmission times initially during state-space exploration then the algorithm convergences to the optimal path during later transmissions. Safe RL has faster convergence due to the smaller state space. The average delays for safe RL is higher compared to our algorithm due to varying D_F. The large number of paths to t cause very high initial transmission times in classical RL leading to deadline violations. Rapid routing has equal transmission times for all packets due choice of deadline D_F. The small interval of deadlines leads to static efficient path being chosen.

Figure 9 shows the convergence of maximum rewards obtained for the three RL based algorithms. The convergence rate is directly related to the size of the state-space of the algorithm. Safe RL has the fastest convergence as its state-

Table 4. Average transmission times and Deadline Misses

Experiment	**Rapid Routing**	**Classical RL**	**Safe-RL**	**Modified Safe-RL**	**Modified Safe-RL (No node add)**
(Epsiodes)	Avg TX Times	Avg TX Times	Avg TX Times	Avg TX Times	Avg TX Times
Exp1($D_F = 40$) (10000)	10.0000	7.9328	10.5706	8.7068	10.0336
Exp2 (10000)	10.1440	10.8257	10.6416	10.9499	-
Exp3 (10000)	12.9917	11.3529	10.6846	11.0054	-
Exp4 (10000)	15.1520	13.3529	10.6636	11.0054	-
Exp5 (50000)	11.0000	17.6763	18.2826	17.5672	-
Exp6 (50000)	22.1172	18.2935	19.6926	18.3644	-
(Epsiodes)	D_F Misses	D_F Misses	D_F Misses	D_F Misses	D_F Misses
Exp1($D_F = 40$) (10000)	0	0	0	0	0
Exp2 (10000)	0	0	0	0	-
Exp3 (10000)	0	14	0	0	-
Exp4 (10000)	0	122	0	0	-
Exp5 (50000)	0	22	0	0	-
Exp6 (50000)	0	42	0	0	-

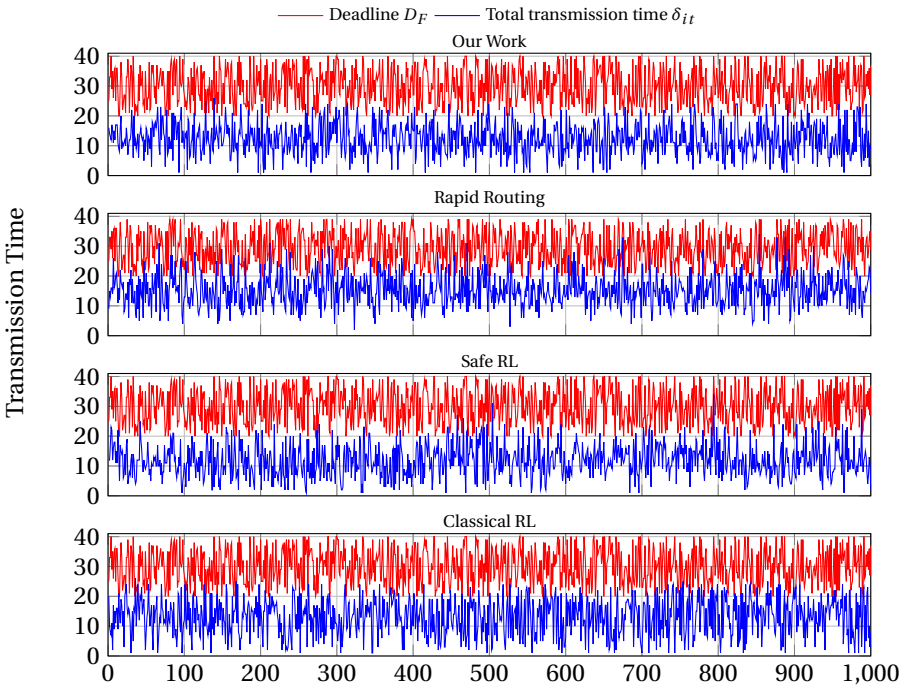


Figure 7. Experiment 4: Transmission times with $D_F \in [20,40]$ and $\delta \in (0, c^W]$

space is small compared to our algorithm. Classical RL is slow to converge as it explores all paths in the network disregarding safety.

5.6 Experiment 6

In Experiment 6, we set $\delta \in (0, c^W_{(n-1)n}]$ and simulate the system under very high load for 50000 packets. The other parameters are set the same as in the previous experiment.

Figure 8 shows the transmission times for varying deadlines. Rapid Routing has very high average transmission times due to the pre-built routing tables but has no deadline violations. Classical RL has the lowest average transmission times but suffers from deadline violations. Our algorithm performs better than Safe RL as our state-space captures more information about the current state of the system. The better performance comes at a cost of larger state-space leading to slower convergence to the optimal path as seen in Figure 9. This is a necessary trade off as real-time networks have packets with varying deadlines in a dynamic environment.

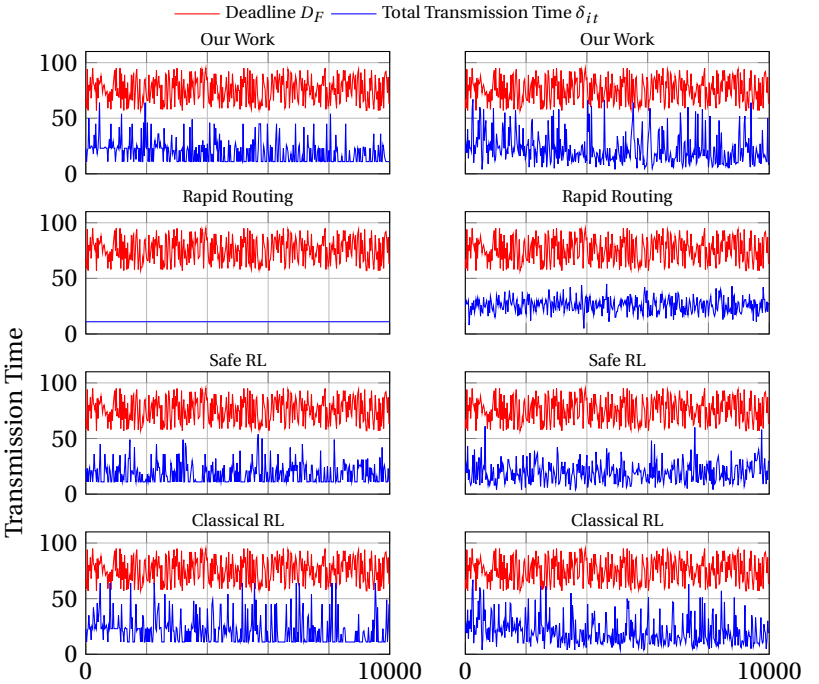


Figure 8. Experiment 5: transmission times for modified tree network with $\delta = c^T$(left). Experiment 6: Transmission times for modified tree network with $\delta \in (0, c^W]$(right)

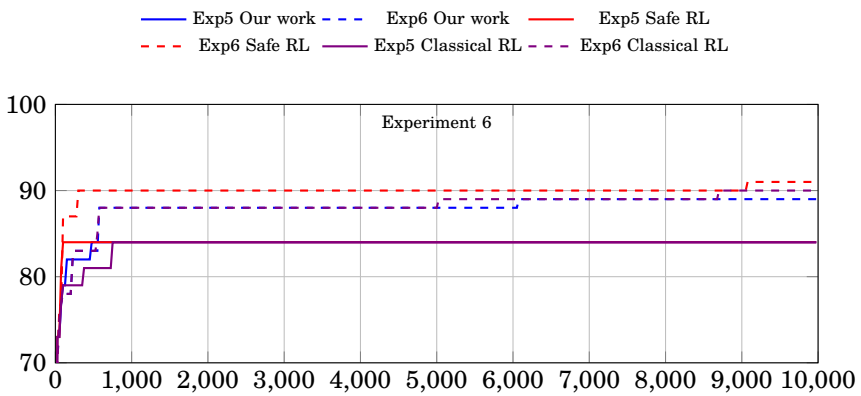


Figure 9. Maximum Rewards obtained over time

6. Discussion

In this section we discuss some of the aspects of practical implementation of our algorithm. Then we explore how the different algorithms analysed are suitable to different real-time networks.

6.1 Implementation Aspects

Variations in c^W Even with robust worst case timing analysis, it might not be feasible to guarantee c^W for each link. Changes in $c^W_{(n-1)n}$ can be adapted by running Algorithm 10 at N_n treating the variation as a node addition. We assume that no packets were transmitted during the algorithm to guarantee safety.

Presence of loops Deadlines cannot be guaranteed in the presence of loops in the network. This is due to the possibility of transmitting packets over a section of path multiple times during path exploration. Loop removal is widely studied in SDN literature and can be similarly applied in conjunction with our algorithm.

Multiple sources and Bi-directional links In our current work, we use a single source node. The work can be extended to multiple source nodes as the routing decisions are only based on current node and remaining time. In case of bi-directional edges, two routing tables would be needed (one for each direction).

Multiple packets In the presence of multiple packets over a link $(N_{n-1} \rightarrow N_n)$ the value of $c^T_{(n-1)n}$ increases. Our algorithm guarantees safety as the model guarantees $\delta_{(n-1)n} \leq c^W_{(n-1)n}$.

6.2 Choice of Algorithm

The choice of algorithm for packet routing is very dependent on the network parameters as seen from the result section. The size of the network, variance in parameters and number of parameters all play a major role in the choice of the algorithm.

Rapid Routing [Baruah, 2018] Rapid routing is suitable for routing packets through a static real-time networks with moderate variance in c^W in links comparatively. It has most of complexity in the pre-processing algorithm. The pre-processing algorithm is run only once in a static network with $\delta = c^T$ and routing is based on the constructed routing table. Variations in c^T would make Rapid Routing suboptimal. For optimality, the pre-processing has to execute again leading to large computational time [Nayak Seetanadi, Årzén, and Maggio, 2020]. Large variations in c^W between the different links leads to the construction of large routing tables at each node.

Classical RL [Sutton and Barto, 2018] Classical has low computational and is analysed extensively for packet routing. Classical RL is however unsuitable for real time networks as unsafe edges are chosen during exploration leading to deadline violations.

Table 5. Comparision of the different routing algorithms

	Rapid Routing [Baruah, 2018]	**Classical RL [Sutton and Barto, 2018]**	**Safe-RL [Nayak Seetanadi, Årzén, and Maggio, 2020]**
Routing Table	Built with pre-processing	Built with exploration	Built with exploration
State Space	-	Current node, Time remaining	Current node
Node addition and deletion	Rerun pre-processing	Yes	No
Adapt to c^T changes	Rerun pre-processing	Dynamic, through exploration	Dynamic, through exploration
Safety Guarantees	Yes	No	Stochastic convergence

Safe RL [Nayak Seetanadi, Årzén, and Maggio, 2020] Safe RL is most efficient for real-time networks that either have a small number of nodes or large networks with static deadlines. It has the advantage of small state-space size due to its choice of state leading to faster convergence of the algorithm.

Our Work Our work has the drawback of slow convergence due to the large state-space that is explored initially during network creation. On the other hand our algorithm performs better with more packets transmitted with different values of D_F. The algorithm also performs well in networks with variance in the different parameters of the network, which is expected in real-networks. Compared to the other algorithm, we also support safe node additions and deletions dynamically in the network.

Table 5 summarizes the different algorithms explored in the paper.

7. Related work

In this section we will discuss some prior research related to our work. The most recent algorithms [Nayak Seetanadi, Årzén, and Maggio, 2020], [Baruah, 2018] most related to the work presented in this paper is analysed in Section 5.

Previous research can widely be divided into three categories

- Optimal path finding in graphs with two or more cost function
- Reinforcement learning based routing methods
- Safe reinforcement learning methods

Shortest path problems are among the most widely studied problems in network optimization [Bertsekas, 1991], [Ahuja, Magnanti, and Orlin, 1988] and [Schrijver, 2003]. However these algorithms consider a single cost minimization and are suitable for worst case timing analysis. The problem of determining paths with multiple cost functions has been studied as bicriteria [Hansen, 1980], [Hamacher, Ruzika, and Tjandra, 2006] and multigraph [Martins, 1984], [Loui, 1983] cost optimization. These multiple cost shortest-path algorithm use interdependent costs where it is not possible to decrease some cost at the expense of others. This is not applicable to our problem of edges with two independent costs.

Dynamic routing in reinforcement learning has been a popular research area due to the large number of packet transmissions facilitating learning. See [Habib, Arafat, and Moh, 2019] and [Al-Rawi, Ng, and Yau, 2013] for surveys on routing using reinforcement learning. RL based routing do not provide guarantees on delays which leads to deadline violations during exploration.

Reinforcement learning based methods provide safety mainly using two methods. Using modified optimization criteria [Davidson and Schmidt, 1992] and [Ozan, Baskan, Haldenbilen, and Ceylan, 2015], the concept of risk is introduced into the optimization process. The learning policy is then updated considering this modified optimization method for safe learning.

In our work we use the second method, ensuring modified exploration to ensure safety. Safe exploration restricts exploratory actions using expert knowledge about the system [Moldovan and Abbeel, 2012], [Hans, Schneegaß, Schäfer, and Udluft, 2008] and [Sui, Gotovos, Burdick, and Krause, 2015]. We build on these works by ensuring safe routing for real-time networks.

8. Conclusion

In this paper we have studied the problem of packet routing over a recently proposed model for routing applicable for IoT networks. The model consists of links with dual-delays where the encountered delays are different from the worst case delays. To guarantee packet transmission within a pre-determined constraint, we apply safe reinforcement learning to find optimal paths ensuring traversal of only safe edges. The constant safe exploration ensures that the algorithm is dynamic and robust to changes in deadlines in conjunction with the large state-space that better captures behavior of the network. We also showed the robustness of our algorithm to node additions and deletions due to mobility in IoT networks. This dynamicity of our algorithm allows us to route over new paths through node additions that lead to lower transmission times.

We provided safety guarantees for our algorithm and verified it by packet transmission through two different network models with varying network properties. Compared to classical RL we provided robust safety guarantees for our algorithm. We also discussed the different state of the art routing algorithms different real-time networks and their applicability.

References

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1988). "Network flows".

Badia, A. P. et al. (2020). *Agent57: outperforming the atari human benchmark*. arXiv: 2003.13350 [cs.LG].

Baruah, S. (2018). "Rapid routing with guaranteed delay bounds". In: *2018 IEEE Real-Time Systems Symposium (RTSS)*. Nashville, TN, USA. DOI: 10.1109/RTSS.2018.00012.

Bertsekas, D. P. (1991). *Linear network optimization*.

Davidson, J. B. and D. K. Schmidt (1992). *Modified optimal control pilot model for computer-aided design and analysis*. Tech. rep. URL: https://ntrs.nasa.gov/citations/19930002271.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". *Numerische Mathematik* **1**:1, pp. 269–271. DOI: 10.1007/BF01386390.

Habib, M. A., M. Y. Arafat, and S. Moh (2019). "Routing protocols based on reinforcement learning for wireless sensor networks: a comparative study". *Journal of Advanced Research in Dynamical and Control Systems*, pp. 427–435.

Hagberg, A. A., D. A. Schult, and P. J. Swart (2008). "Exploring network structure, dynamics, and function using networkx". In: Varoquaux, G. et al. (Eds.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, pp. 11–15. URL: https://www.researchgate.net/publication/236407765_Exploring_Network_Structure_Dynamics_and_Function_Using_NetworkX.

Hamacher, H. W., S. Ruzika, and S. A. Tjandra (2006). "Algorithms for time-dependent bicriteria shortest path problems". *Discrete Optimization* **3**:3. Graphs and Combinatorial Optimization, pp. 238–254. DOI: 10.1016/j.disopt.2006.05.006.

Hans, A., D. Schneegaß, A. M. Schäfer, and S. Udluft (2008). "Safe exploration for reinforcement learning." In: *ESANN2008*, pp. 143–148. URL: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2008-36.pdf.

Hansen, P. (1980). "Bicriterion path problems". In: Fandel, G. et al. (Eds.). *Multiple Criteria Decision Making Theory and Application*. Springer, pp. 109–127. DOI: 10.1007/978-3-642-48782-8_9.

Loui, R. P. (1983). "Optimal paths in graphs with stochastic or multidimensional weights". *Communications of the ACM* **26**:9, pp. 670–676. DOI: 10.1145/358172.358406.

Martins, E. Q. V. (1984). "On a multicriteria shortest path problem". *European Journal of Operational Research* **16**:2, pp. 236–245. DOI: 10.1016/0377-2217(84)90077-8.

Moldovan, T. M. and P. Abbeel (2012). "Safe exploration in markov decision processes". *arXiv preprint arXiv:1205.4810*.

Nayak Seetanadi, G., K.-E. Årzén, and M. Maggio (2020). "Adaptive routing with guaranteed delay bounds using safe reinforcement learning". In: *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, pp. 149–160. DOI: 10.1145/3394810.3394815.

Ozan, C., O. Baskan, S. Haldenbilen, and H. Ceylan (2015). "A modified reinforcement learning algorithm for solving coordinated signalized networks". *Transportation Research Part C: Emerging Technologies* **54**, pp. 40–55. DOI: 10.1016/j.trc.2015.03.010.

Polychronopoulos, G. H. (1992). *Stochastic and dynamic shortest distance problems*. PhD thesis. Massachusetts Institute of Technology.

Al-Rawi, H., M. Ng, and K.-L. Yau (2013). "Application of reinforcement learning to routing in distributed wireless networks: a review". *Artificial Intelligence Review* **43**. DOI: 10.1007/s10462-012-9383-6.

Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media. ISBN: 978-3-540-44389-6.

Shi, W., J. Cao, Q. Zhang, Y. Li, and L. Xu (2016). "Edge computing: vision and challenges". *IEEE Internet of Things Journal* **3**:5, pp. 637–646. DOI: 10.1109/JIOT.2016.2579198.

Sui, Y., A. Gotovos, J. Burdick, and A. Krause (2015). "Safe exploration for optimization with Gaussian processes". In: *International Conference on Machine Learning*, pp. 997–1005.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An Introduction*. Adaptive computation and machine learning. MIT Press. ISBN: 9780262039246.

Tesauro, G. (1995). "Temporal difference learning and td-gammon". *Commun. ACM* **38**:3, pp. 58–68. DOI: 10.1145/203330.203343.