Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Performance and Availability Assessment of a Highway Video Monitoring System with Stochastic Petri Nets

Carlos Victor Santana Brito

Número de Ordem PPGCC: M001

Picos-PI, Janeiro de 2025

Carlos Victor Santana Brito

Performance and Availability Assessment of a Highway Video Monitoring System with Stochastic Petri Nets

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Supervisor: Prof. Dr. Francisco Airton Pereira da Silva

Co-supervisor: Prof. Dr. Paulo Antonio Leal Rego

Picos-PI

Janeiro de 2025

FICHA CATALOGRÁFICA
Universidade Federal do Piauí
Sistema de Bibliotecas UFPI - SIBi/UFPI
Biblioteca Setorial do CCN

B862p Brito, Carlos Victor Santana.
Performance and availability assessment of a highway video monitoring system with stochastic Petri Rets / Carlos Victor Santana Brito. -- 2025.
99 f.

Dissertação (Mestrado) - Universidade Federal do Piauí. Centro de Ciências da Natureza. Programa de Pós-Graduação em Ciências da Computação, Picos, 2025.
"Orientador: Prof. Dr. Francisco Airton Pereira da Silva. Coorientador: Prof. Dr. Paulo Antonio Leal Rego".

1. Rede de Petri. 2. Vídeo-monitoramento. 3. Rede Petri - desempenho e manutenção. I. Silva, Francisco Airton Pereira da. II. Rego, Paulo Antonio Leal. II. Titulo.

CDD 005.4

Bibliotecária: Caryne Maria da Silva Gomes - CRB3/1461

Carlos Victor Santana Brito

Performance and Availability Assessment of a Highway Video Monitoring System with Stochastic Petri Nets

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Documento assinado digitalmente
gov.br
FRANCISCO AIRTON PEREIRA DA SILVA
Data: 25/03/2025 13:52:57-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Francisco Airton Pereira da Silva
Orientador

Documento assinado digitalmente
gov.br
PAULO ROMERO MARTINS MACIEL
Data: 03/04/2025 14:10:15-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Paulo Romero Martins Maciel
Membro da Banca

Documento assinado digitalmente
gov.br
PAULO ANTONIO LEAL REGO
Data: 26/03/2025 18:17:17-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Paulo Antonio Leal Rego
Co-orientador

Documento assinado digitalmente
gov.br
JULIANA OLIVEIRA DE CARVALHO
Data: 26/03/2025 15:48:45-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Juliana Oliveira de Carvalho
Membro da Banca

Documento assinado digitalmente
gov.br
GERALDO PEREIRA ROCHA FILHO
Data: 03/04/2025 09:41:18-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Geraldo Pereira Rocha Filho
Membro da Banca

Picos-PI
Janeiro de 2025

Resumo

Os crimes de trânsito são um dos muitos problemas recorrente em vários países do mundo. Crimes envolvendo roubo de carros, motocicletas e outros veículos aparecem ocasionalmente nas manchetes de jornais e programas de notícias. Atualmente é possível utilizar câmeras de monitoramento para detectar e rastrear veículos ilegais. No entanto, implementar um sistema desta natureza no mundo real requer recursos físicos e digitais abundantes. Neste contexto, esta dissertação apresenta modelos de redes de Petri estocásticas para avaliar a confiabilidade e eficiência de manutenção destes sistemas. Apresentamos três modelos de manutenção: um modelo reativo básico, um modelo reativo avançado com capacidade de auto-reparo e um modelo preventivo que utiliza rejuvenescimento de *software*. Nós avaliamos o impacto destas estratégias de manutenção na confiabilidade do sistema. Essa pesquisa inclui estudos de caso que demonstram a aplicação prática desses modelos, destacando seu potencial para melhorar a confiabilidade do sistema com tempo de inatividade mínimo. As conclusões sugerem que o planejamento estratégico da manutenção aumenta significativamente a confiabilidade dos sistemas de vigilância, contribuindo assim para a segurança e a eficiência operacional de ambientes urbanos inteligentes. Esta dissertação também propõe um modelo de desempenho em redes de Petri estocásticas para avaliar um sistema de vídeo-monitoramento através da detecção de placas em rodovias. O uso de redes de Petri gera maior precisão entre o modelo e a realidade, pois permite capturar comportamentos complexos, como concorrência e paralelismo. Os resultados mostram uma tendência de maior impacto nas métricas quando se varia a capacidade dos dois módulos iniciais do sistema e o número de câmeras utilizadas. Os estudos de casos propostos são um guia prático para administradores de sistema utilizarem o modelo.

Palavras-chaves: vídeo-monitoramento, desempenho, disponibilidade, manutenibilidade, rede de Petri, análise de sensibilidade;

Abstract

Traffic crimes are one of the many recurring problems in many countries around the world. Crimes involving theft of cars, motorcycles and other vehicles occasionally appear in newspaper headlines and news programs. It is currently possible to use surveillance cameras to detect and track illegal vehicles. However, implementing such a system in the real world requires abundant physical and digital resources. In this context, this dissertation presents stochastic Petri net models to evaluate the reliability and maintenance efficiency of these systems. We present three maintenance models: a basic reactive model, an advanced reactive model with auto-repair capabilities and a preventive model that uses software rejuvenation. We evaluate the impact of these maintenance strategies on system reliability. This research includes case studies that demonstrate the practical application of these models, highlighting their potential to improve system reliability with minimal downtime. The conclusions suggest that strategic maintenance planning significantly increases the reliability of surveillance systems, thus contributing to the safety and operational efficiency of smart urban environments. This dissertation also proposes a performance model in stochastic Petri nets to evaluate a video surveillance system through the detection of signs on highways. The use of Petri nets generates greater precision between the model and reality, as it allows capturing complex behaviors, such as concurrency and parallelism. The results show a tendency for greater impact on the metrics when the capacity of the two initial modules of the system and the number of cameras used vary. The proposed case studies are a practical guide for system administrators to use the model.

Keywords: video monitoring, performance, availability, maintainability, Petri net, sensitivity analysis.

List of Figures

List of Tables

List of abbreviations and acronyms

DoE	Design of Experiments
SPN	Stochastic Petri Net
PN	Petri Net
IoT	Internet of Things
VANET	Vehicular Ad Hoc Network
MRT	Mean Response Time
SLA	Service Level Agreement
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
FPS	Frames Per Second
CDF	Cumulative Distribution Function

Contents

1 Introduction

The urban security landscape within smart cities has undergone a significant transformation, driven by the rapid advancement of technological paradigms and a growing imperative for fortified security measures, along with efficient urban governance. Surveillance systems, characterized by a network of Closed-Circuit Television (CCTV) cameras and various sensors, have become an integral part of this transformation, serving not only to monitor extensive urban areas, but also to manage traffic flows and safeguard the population. The application of camera surveillance systems to monitor behavior in public spaces is fundamental to collective security (POGADADANDA et al., 2023). According to the website Statista (2023), the video surveillance camera market was valued at 35 billion US dollars in 2022, with forecasts for growth to more than 62 billion dollars by 2027. Modern surveillance systems have critical functionalities such as facial recognition, motion detection and activity tracking, which play essential roles in preventing potential disasters (CHEONG et al., 2019a; BABIYOLA et al., 2023). Despite its diverse applications – from crime prevention and traffic management to fire detection – continuous monitoring by human resources is expensive and requires 24-hour camera surveillance (SUDHEER et al., 2022; ELHARROUSS; ALMAADEED; AL-MAADEED, 2021). Therefore, there is a growing need for automated surveillance systems that offer more efficient and cost-effective monitoring.

A distinctive feature of contemporary urban surveillance systems is their synergistic integration with cutting-edge technologies such as Artificial Intelligence (AI) and deep learning algorithms. This integration facilitates the autonomous analysis of voluminous video data streams, enabling the discernment of behavioral patterns, the identification of anomalous activities and the recognition of specific incidents, thus avoiding the need for continuous human surveillance. The arrival of cloud and edge computing technologies (NGUYEN et al., 2021) has further increased the effectiveness of these systems by optimizing data processing capabilities, while the incorporation of blockchain technology ensures data integrity and security, whilst raising pertinent concerns in relation to privacy and data protection (NGUYEN et al., 2023).

1.1 Problem

The Internet of Things (IoT) is a communication paradigm that aims to make everyday technologies even more immersive and widespread (GONÇALVES et al., 2021). Smart surveillance systems supported by IoT can perform monitoring automatically, reducing the need for human intervention and improving overall performance (PAIKARAY;

GANDHI, 2022). Integrating automated analysis into video surveillance is an area that can be further explored (CHEONG et al., 2019b), providing significant benefits in terms of efficiency and security. Video surveillance is an integral component of modern urban security, and when combined with computational analysis (CHEONG et al., 2019b), it not only simplifies information collection but also optimizes data analysis. However, planning a high-performance monitoring system is a challenging task.

The demand for fast and continuous processing of information from multiple distributed cameras is a factor that adds significant complexity and costs, especially in equipment connectivity in the security sector (GONÇALVES et al., 2021). Running tests in real environments is generally not feasible, as more advanced surveillance systems are often expensive and must be deployed in large scale to cover a relatively large area (BRITO et al., 2023). Installing cameras, maintaining high-capacity servers, and implementing robust networks contribute to the system's total cost. Such factors imply significant challenges in terms of coordinating and synchronizing data between different cameras and servers, as well as the need for efficient algorithms to quickly identify vehicles of interest. Analytical models can be used to evaluate surveillance systems during the design phase, as testing in a real environment is often expensive (JIANG et al., 2022).

Another important aspect for monitoring systems is availability. Designing a highly available monitoring system is fraught with challenges, primarily due to the commonality of partial equipment failures that can compromise surveillance quality or cause total system breakdowns (GONÇALVES et al., 2021). Consequently, investigations into maximizing system availability are crucial. Surveillance systems' reliability is paramount as they often directly impact human safety. The combination of availability and reliability constitutes dependability, which is essential for any surveillance system. An important dimension of system availability is maintainability, defined as the ease of performing various maintenance activities (BOSCH; BENGTSSON, 2001). Integrating maintenance considerations into availability analyses illuminates the interplay between these domains. Another critical aspect is software aging, which refers to the gradual degradation of software performance over extended operational periods, potentially leading to system failures. Addressing this issue involves software rejuvenation techniques, including periodic preemptive restarts to maintain optimal performance (HUANG et al., 1995; SILVA et al., 2022). Analytical models can be used to evaluate surveillance systems during the design phase, as testing in a real environment is often expensive (JIANG et al., 2022).

Evaluating an urban monitoring system is a complex task that requires a lot of planning and efficiency. Each aspect of the system - performance and availability - requires a different type of analysis, and therefore an entirely different set of parameters. For performance, for example, the system administrator can vary capabilities, processing algorithms, number of cameras, frame rate, among others. While in availability the set of

parameters is focused on component failure and recovery time. Mainly in performance, it is noticeable that there is a greater number of parameters, as there are several metrics that can be measured, such as utilization, response time, discard rate, and others. Furthermore, planning in the real world can be quite costly if the system administrator wants to vary each parameter in order to find an optimal metric value. To mitigate costs arising from a practical evaluation of video monitoring systems, there is the possibility of using analytical models. Analytical models can estimate values for both types of metrics discussed here, without incurring financial costs for system administrators. Therefore, the main question of this research is: **Can we predict performance and availability metrics of video monitoring systems without prior real physical infrastructure using analytical models?**

1.2 Objetives

The general objective of this dissertation is to create models that can help administrators of computing infrastructures to plan and evaluate video monitoring systems before deployment. The models should consider aspects of performance and availability, providing models that capture the system's behavior and address details that add value to the analysis, such as maintenance of the system.

The specific objectives of this work are:

- Design a model to allow system performance analysis, which captures specific behaviors of a video monitoring system, as a *frames* control mechanism. It is also expected that the model will be used for capacity planning, varying queuing parameters, processing time, number of cameras, among others. Alongside the model there are also case studies that serve as a practical guide to its uses and potential;
  - A paper was published based on this specific objective:
    - Brito, Carlos, et al. "Performance evaluation of a video surveillance system using stochastic petri nets for license plate detection on highways."Journal of Reliable Intelligent Environments 10.4 (2024): 477-488.
- Design models that allow system availability analysis. The models address different maintenance strategies: reactive, preventive, and self-repair. Furthermore, the models consider the software aging aspect of the applicable components. Alongside the model there are also case studies that serve as a practical guide to its uses and potential, such as the impact of varying maintenance strategies;
  - A paper was written based on this specific objective and submitted to Computing journal (waiting result).

- Carry out a practical experiment to validate the presented models, performance and availability. Validation shows that the models are accurate and faithful to reality. Through statistical techniques, a practical analysis can determine the validity of the proposed models.

  – Experiments were carried out and successfully validated the models.

1.3 Out of Scope

Some subjects are outside the context of this work and, therefore, were not addressed in this work to keep it lean and objective. Some of these subjects are listed below:

- Learning techniques: the topic is one of the most common when looking for surveillance studies in general. Most of the works do research with the objective of finding the technique that provides greater accuracy, precision, *recall* and F1, which are common metrics in the field of machine learning. This work, however, explores the assessment of performance and availability that is less covered in the literature. It is possible to model the behavior of learning algorithms with analytical models, but this would result in a study completely unrelated to the one presented here. Therefore, we will focus only on the two aspects already mentioned, as they have a better affinity with the evaluation of distributed systems.;

- Energy and cost analysis: the use of computational resources such as cameras and servers normally enables the analysis of both energy and monetary costs. However, the focus of the work is purely on the proposed metrics, performance and availability. However, some metrics can enable this analysis by their users, such as the team calls per year metric, which in itself does not express cost, but can be used to calculate monetary costs;

- Security: security aspects were not considered in the modeling. Normally, modeling problems involving security is quite complex in terms of modeling. One of the main challenges is how to deal with invasions, a completely random event in a model that requires mathematical rates and distributions. Security modeling is not impossible, but instead of incorporating it into these models, it would be more appropriate to generate a model focused on this aspect. The failure time of a system is also a random event, but with a study it is possible to state with a certain degree of certainty that the system will eventually fail, but with attacks it would not be so simple. First, it is necessary to study whether it is likely that an attack will occur and then whether it is possible to calculate the average time between attacks given that it is an uncertain event. In other words, an electronic component will always

fail, but an attack will not always occur. Therefore, we chose not to explore this aspect, as it would culminate in another study unrelated to this one.;

- Data storage: data collected and processed by cameras is occasionally stored temporarily or permanently for later reference. However, as with the security aspect, it would be more prudent to generate a model focused only on this aspect, since it does not interact well with the aspects modeled here, speaking at the model level.

1.4 Work Structure

The remainder of this dissertation is organized as follows: Chapter 2 presents concepts necessary to understand this dissertation; Chapter 3 presents the related works; Chapter 4 shows the workflow used to write this work; Chapter 5 the modeled architecture. Chapter 6 presents the part of the dissertation that performs the availability analysis of a video monitoring system, including: related work, architecture, overview of the models, sensitivity analysis and case studies; Chapter 7 shows the performance part of the dissertation, including: related work, architecture, model overview, sensitivity analysis and case studies; Chapter 8 presents the validation of the models. Finally, Chapter 9 concludes this dissertation.

2 Background

This chapter presents essential concepts for understanding this work. Important information about Stochastic Petri Nets and some equations used to analyze the proposed model. Next, the functioning of the Design of Experiments (DoE) and how its results are generally interpreted are discussed. Furthermore, concepts about the Percentage Difference method are presented.

2.1 Stochastic Petri Nets

Petri Nets (PN) are graphical and mathematical modeling tools used to represent different types of systems, characterized by simultaneity, asynchrony, distribution, parallelism, non-determinism and/or stochastic processes (CHEN; HA, 2018). Offering a set of formalisms to abstract complex systems, PNs, as graphical tools, can be used as a visual communication aid similar to flowcharts, block diagrams and networks. Among the variations of PNs, the Generalized Stochastic Petri Nets (GSPN) stand out, which incorporate stochastic elements to represent systems in an even more precise way (MOLLOY, 1982; GERMAN, 2000; MARSAN et al., 1994; MACIEL, 2023b; MACIEL, 2023a). However, for simplicity, we will refer to these networks as SPNs.

SPNs can be identified as a type of directed graph divided into two parts, populated by three types of objects. These objects are places, transitions, and directed arcs that connect places to transitions and transitions to places (RODRIGUES et al., 2020). The timed transition follows a stochastic behavior, following a probability distribution function (MURATA, 1989). The immediate transition triggers when it is activated, without waiting for any specific period. The white circle symbolizes the places. Arcs are used to connect places to transitions. Inhibitor arcs block or allow the passage of tokens from one place to another. Furthermore, the token is assigned to a specific place (BRITO et al., 2021). Figure 1 shows the components of an SPN model.

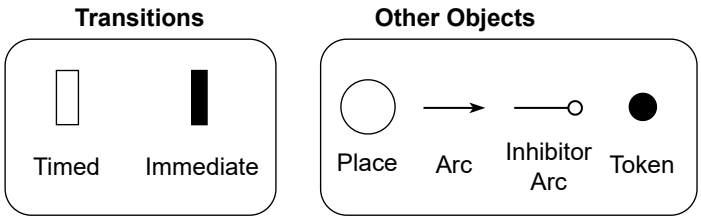


Figure 1 – SPNs Components.

SPN behavior is defined in terms of a token flow. Tokens are created and destroyed

according to transition fires (GERMAN, 2000). Immediate transitions represent instantaneous activities and have greater trigger priority than timed transitions. Such transitions can also contain a guard condition and the user can specify a different trigger priority among other immediate transitions. There are also guard functions in SPNs. Guard functions are Boolean expressions that control the triggering of a transition, declaring some condition in relation to the network marking. If a transition's guard function produces a true value, it is able to fire, otherwise the transition is disabled (MARSAN et al., 1998). The SPNs models were proposed with the aim of developing a tool that allowed the integration of formal description, proof of correctness and performance evaluation (SILVA et al., 2018).

2.2 Sensitivity Analysis

Sensitivity analysis measures the impact of specific input data on output data to identify weaknesses in computer systems. Subsequently, techniques are employed to improve these systems in various scenarios (CAMPOLONGO; TARANTOLA; SALTELLI, 1999). Systems designers often adopt sensitivity analyzes to evaluate how "sensitive" a metric is to changes in the model (SANTOS et al., 2021). There are many ways to perform sensitivity analysis, such as: regression analysis, perturbation analysis (PA), one-by-one variation, Monte Carlo simulation, parametric differential analysis, correlation analysis and percentage difference.

Distinguishing which method to use is a difficult step, making it necessary to establish which computational resources are available, and the characteristics of the problems addressed (CAMPOLONGO et al., 2004; PIANOSI et al., 2016). In a way, sensitivity analysis can provide necessary security and guide results within the perspective pre-established by system administrators. In this work, we apply a sensitivity analysis with DoE and Percentage Difference.

2.2.1 Design of Experiments

DoE corresponds to a set of statistical techniques that deepen the understanding of the product or process under study (KLEIJNEN, 1995). DoE is a powerful technique used to explore new processes and gain deeper insights into existing processes, followed by optimizing those processes to achieve world-class performance (ANTONY, 2014). In the specialized literature (FEITOSA et al., 2021; COSTA et al., 2016; SANTOS et al., 2021), we find categories of graphs generally used in experiments with the DoE approach. The factor effect graph, represented by bars arranged in descending order, highlights the relative impact of each factor. The higher the bar, the greater its impact, providing a clear view of the influences of each factor. Figure 2 presents an example of a factor effect graph. The graph shows three factors: A, B and C. Factor C has the greatest impact.

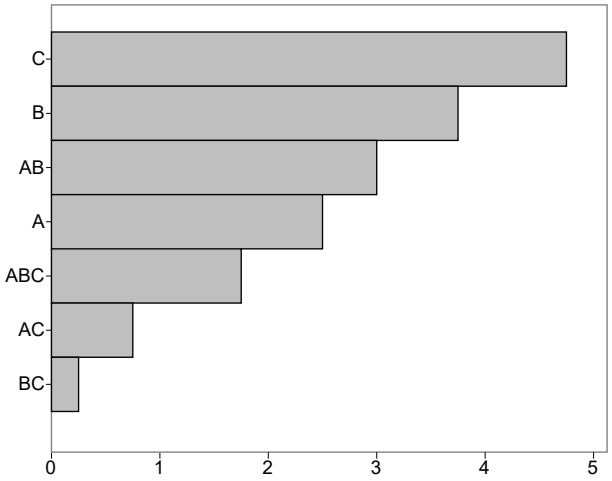


Figure 2 – Generic example of factor effect graph.

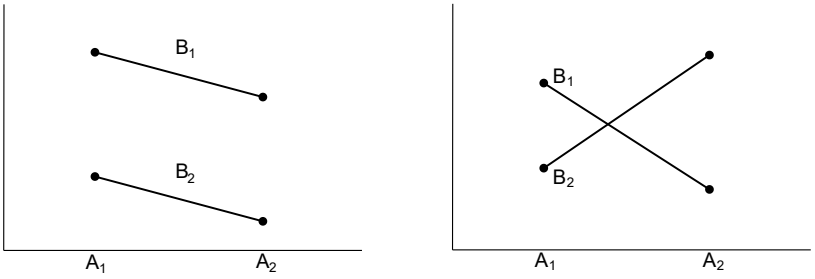
The factor effect graph allows us to identify which interaction of factors has the most significant effect on the optimization process or study design, indicating where attention should be focused. Understanding the magnitude of interactions between factors allows the selection of the best combination of measures, identifying patterns of cumulative or degrading effects between factors. The interaction between factors A and B can be calculated using Equation (2.1). The $E_{A,B(+1)}$ represents the effect of factor A on the high level of factor B, and $E_{A,B(-1)}$ represents the effect of factor A on the low level of factor B.

$$I_{A,B} = \frac{1}{2}(E_{A,B(+1)} - E_{A,B(-1)}) \tag{2.1}$$

Interaction graphs aim to identify interactions between factors. An interaction occurs when the influence of a given factor on the outcome is changed (amplified or reduced) by variation in the levels of another factor. If the lines on the graph are parallel, this indicates the absence of interaction between the factors. On the other hand, if the lines are not parallel, it is a sign of a significant interaction between the factors in question. Figure 3a represents an example in which there are no interactions between the factors, as the lines are parallel. Figure 3b exemplifies a case of interaction between factors, as the lines intersect. In this case, the change under a given metric for factor A at level A1 is greater than that at level A2. Changes in factor A levels for some given metric indicate a dependence of factor A on factor B levels.

2.2.2 Percentage Difference

The percentage difference method was chosen to perform the sensitivity analysis of this work, being an approach that does not use a continuous domain for the parameter input values. This process is based on calculating the percentage difference, when varying an input parameter from its minimum value to its maximum value. It is necessary to



(a) Interaction graphs with low interaction between factors.

(b) Interaction graphs with high interaction between factors.

Figure 3 – Generic interaction graphs.

use the entire range of possible values for each parameter to calculate the parameter sensitivities (HOFFMAN; GARDNER, 1983).

Equation 2.2 shows how the sensitivity index is calculated using the percentage difference method. The expression $max\ \{Y(\theta)\}$ and $min\ \{Y(\theta)\}$ are, respectively, the maximum and minimum output values, measured when varying the θ parameter over a range of its n possible values of interest. If $Y\ (\theta)$ is known to vary monotonically, only the extreme values of θ (i.e., θ_1 and θ_n) can be used to calculate $max\ \{Y(\theta)\}$, $min\ \{Y(\theta)\}$ and as a result $S_\theta\{\mathrm{Y}\}$ (JúNIOR, 2016).

$$S_\theta\{Y\} = \frac{max\ \{Y(\theta)\} - min\ \{Y(\theta)\}}{max\ \{Y(\theta)\}} \tag{2.2}$$

2.3 Dependability

Dependability is a fundamental property of systems, encompassing attributes such as availability, reliability, maintainability, and maintenance support. It refers to the ability of a system to deliver its intended functionality without failures, interruptions, or unacceptable degradation over time (AVIZIENIS et al., 2004). This concept is critical in the design and operation of systems where failures can result in significant safety risks, financial losses, or operational disruptions, such as in healthcare, telecommunications, and industrial automation.

Availability, defined as the probability of a system or component being operational at a specific time, considering the alternation between operating and failure states (LAPRIE; AVIZIENIS; KOPETZ, 1992), is a cornerstone of dependability. This attribute encompasses reliability, maintainability, and maintenance support, and is widely used in systems engineering to evaluate the operational readiness of equipment and infrastructure. By assessing availability, one can determine a system's ability to perform its functions within a given time interval. The calculation of availability (A) often relies on parameters such as the Mean Time to Failure (MTTF) and the Mean Time to Repair (MTTR), which

represent the average time until a failure occurs and the average time required to restore operation, respectively. The steady-state availability is classically expressed as shown in Eq. 2.3.

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \tag{2.3}$$

MTTF, in particular, is calculated as the inverse of the failure rate (λ) in systems following the exponential distribution, a widely used model in reliability engineering. This metric represents the average operational lifetime of a system or component before failure (O'CONNOR; KLEYNER, 2011). The probability density function (PDF) associated with MTTF for the exponential distribution is expressed in Eq. 2.4, where (λ) is the constant failure rate, and t represents time. This formulation is crucial for modeling systems with a constant hazard rate, simplifying reliability analyses in practical scenarios.

$$f(t) = \lambda \exp(-\lambda t) \tag{2.4}$$

MTTR, in particular, quantifies the average time required to repair a system after a failure. For systems with multiple failure modes, this metric is calculated as shown in Eq. 2.5, where λ represents the failure rate for each mode, and t_r is the expected repair time for each mode (O'CONNOR; KLEYNER, 2011). The numerator $\Sigma(\lambda t_r)$ represents the total weighted repair time across all failure modes, while the denominator $\Sigma\lambda$ reflects the total failure rate. This formulation ensures that MTTR accurately captures the average repair time by considering both the frequency and duration of failures in the system. By integrating MTTF and MTTR, availability provides a holistic measure of a system's operational readiness, balancing reliability and maintainability.

$$\text{MTTR} = \frac{\Sigma(\lambda t_r)}{\Sigma\lambda} \tag{2.5}$$

Reliability is defined as the probability that a system will satisfactorily perform its specific purpose for a given period without experiencing a failure (TORRES et al., 2016; CARDELLINI et al., 2011). Modeling reliability in critical systems is essential due to the significant consequences that failures may impose on safety, costs, and performance. The reliability ($R(t)$) is expressed as 2.6.

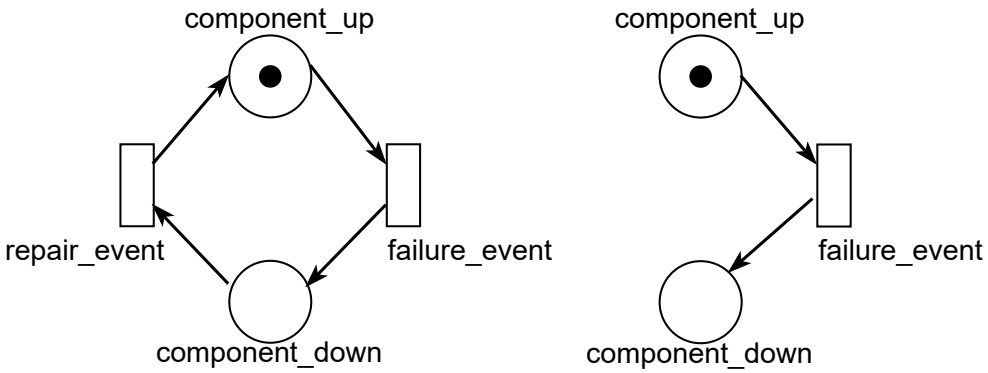
$$R(t) = P(T \geq t) \tag{2.6}$$

where T is the random variable representing the Time to Failure (TTF) of the system or a single component. Alternatively, reliability can be expressed as 2.7

$$R(t) = 1 - F(t) \tag{2.7}$$

where $F(t) = P(T < t)$ is the cumulative distribution function (CDF) that describes the probability of a failure occurring before time t.

Figure 4a presents a generic example of a SPN model for availability. If there is a token in *component_up*, it means the component is operational. The component enters a failed state when the *failure_event* transition is enabled, firing a token to the *component_down* place, which represents unavailability. This transition follows a stochastic process (generally modeled by an exponential distribution) parameterized by the MTTF. Conversely, the *repair_event* transition represents repair, defined by the MTTR. In this example, component availability corresponds to the probability of having at least one token in *component_up*.



(a) Generic availability SPN model. (b) Generic reliability SPN model.

Figure 4 – Example of an SPNs model to represent the availability and reliability of a generic component.

Figure 4b illustrates a generic model for reliability. Unlike the availability model, the reliability model excludes the *repair_event* component, representing a system where repair or maintenance is not considered. The remaining components follow the same flow, with failures leading to system downtime without recovery. In this context, the reliability model focuses on assessing the system's or component's ability to function continuously, emphasizing uninterrupted operation over a specified time frame.

2.4 Maintainability

The maintainability of a system is an extremely important factor for its availability and reliability. The study by Sas et al. (2021) shows that low levels of maintainability can indirectly affect the reliability of a system, since minor problems can accumulate and cause more significant failures over time. Maintainability is the area that studies the aspects

and impacts of maintaining a system on other metrics, such as availability and reliability, ensuring that the system operates continuously and efficiently. Software maintenance has long been known as one of the most expensive and resource-intensive phases of the software development process, as it involves ongoing activities that ensure that the system remains functional, up-to-date, and secure. In addition, the maintainability of a system also includes the ease with which updates and improvements can be implemented, which is crucial for the longevity of the system and user satisfaction (MARI; EILA, 2003).

System maintenance design is a challenging process, as it involves choosing the most appropriate strategies for each situation. There are several possible ways to plan and perform maintenance, whether reactive or preventive, depending on the objectives and conditions of the system. Reactive maintenance is when a defect is expected to appear in the system in order to be repaired, which can cause unexpected interruptions and higher correction costs. Preventive maintenance, on the other hand, seeks to use techniques to try to predict when defects will occur and to apply corrective measures as soon as possible, in order to avoid future problems and improve the efficiency of the system (ALSOLAI; ROPER, 2020). Preventive maintenance, in addition to seeking to minimize future problems, can also reduce long-term costs and increase system efficiency, making it an essential strategic approach for the longevity and performance of any technological system. Furthermore, the implementation of good maintenance practices can contribute to the sustainability of systems, extending their useful life and reducing the environmental impact resulting from the need for frequent replacements.

3 Related Works

This section presents related works that analyzed the performance of their proposals. Table 1 compares related work using six aspects: Context, Metrics, Microservices Architecture, Capacity Planning, Sensitivity, and Processing Pipeline.

Table 1 – Comparison of selected works.

Reference	Metric	Micro-service arch.	Capacity planning	Sensiti-vity	Maintain-ability	Context
(YOON et al., 2020)	Total processing time	✓	✗	✗	✗	Smart cities
(ALSABAAN et al., 2021)	Coverage percentage	✗	✗	✗	✗	Coverage guarantee
(SUN; ALJERI; BOUKERCHE, 2019)	Vehicle speed, traffic density	✗	✓	✗	✗	Traffic control
(MA et al., 2018)	Processing time, number of key frames	✓	✓	✗	✗	Storage
(KIM; JEONG, 2023)	Average detected time	✗	✓	✗	✗	Smart cities
(JOUBARI; OTHMAN; VÈQUE, 2022)	Congestion rate, travel time, waiting time	✗	✗	✗	✗	Traffic control
(ZHANG et al., 2022)	Throughput, latency	✓	✗	✗	✗	Real-time
(UGLI et al., 2023)	Accuracy, memory utilization	✓	✗	✗	✗	Deep Learning
(DING; CHEN; YU, 2022)	Root mean square error, recognition rate	✗	✗	✓	✗	Traffic control
(ZHIRNOV; LYAKHOV; KHOROV, 2019)	Delay, average bitrate, stall	✗	✗	✗	✗	Real-time

Continued on next page

Table 1 – continued from previous page

Reference	Metric	Micro-service arch.	Capacity planning	Sensitivity	Maintainability	Context
(SINQADU; SHIBESHI, 2020)	Latency, memory utilization, network utilization	✓	✓	✗	✗	IoT
(OLAYODE; TARTIBU; OKWU, 2021)	Root mean square error	✗	✗	✗	✗	Traffic control
(LÓPEZ et al., 2013)	Patrol task time, charge time	✗	✗	✗	✓	Robotics
(LIN; PHAM, 2022)	Expected loss cost ratio	✗	✗	✓	✓	Optimization
(TIAN et al., 2024)	Availability, maintenance cost	✗	✗	✓	✓	Optimization
(GONÇALVES et al., 2021)	Reliability, availability	✓	✓	✓	✗	Smart cities
(LINS et al., 2024)	Reliability, availability	✗	✓	✓	✗	Smart cities
This work	Mean response time, utilization, drop probability	✓	✓	✓	✓	Traffic control

Metric

The metrics used in a paper are the key to defining its main objective. Works that seek to evaluate the performance of an algorithm can use resource utilization and total processing time metrics. Some studies can evaluate their algorithm in more detail using machine learning classical metrics. The metrics of the works chosen here investigate the performance of their proposals in relation to resource usage and processing and network delays. In the field of intelligent monitoring, the trend observed when surveying work is the use of machine learning metrics to evaluate learning algorithms. This type of analysis is beyond the scope of this article, so works that focus on the performance of the proposed system were selected. However, it can be seen that there is also a good amount of work that also evaluates the performance of the system. This work focuses on carrying out an analysis more focused on the processing time of a video monitoring system. The metrics

chosen here evaluate response time, usage and discard rate, in addition to carrying out a study using a cumulative distributive function.

Microservice Architecture

The use of microservices is a type of architecture used in distributed systems today. Microservices allow different parts or modules of a system to be separated and distributed across one or more computers, in a way that each service has its unique responsibilities and characteristics. In short, each service has its own responsibility, context and independent database, without direct interference from another service. Among the advantages of microservices are scalability, fault tolerance, ease of maintenance, among others. This type of architecture is beneficial for modularized systems such as the one presented here, and in other selected proposals. Monitoring systems do not always need modules, as sometimes only one main program is needed, so the number of proposals that use and do not use microservices is equivalent.

Capacity Planning

Capacity planning concerns the potential of the work to help system administrators plan the system for later deployment. Typically, works that propose models and simulators tend to offer a solution more focused on capacity planning. The main focus of observed video monitoring work was to propose algorithms and proprietary solutions that aim to improve the efficiency of a proposed or existing system. Therefore, there are few works that focus on capacity planning. This work proposes the use of models to aid in capacity planning, as do some of the selected works.

Sensitivity

Sensitivity analysis seeks to evaluate the components and parameters of a system to verify which ones have the most impact on a given metric. In other words, which factors are most relevant to the occurrence of a cause. There are several types of sensitivity analysis such as percentage difference, DoE and cause and effect diagrams. However, here a broader concept will be considered, any effort made to identify critical components was considered as sensitivity analysis. That said, almost no article has carried out this type of analysis, as carried out in this article. This type of analysis is not so common in the context of video monitoring, and is more commonly seen in work that proposes models, such as this work.

Maintainability

This aspect observes whether the analyzed article explicitly considered maintenance aspects in the study carried out. Maintenance is an important aspect for systems when performing an availability analysis. Usually, what can be seen in the literature is that most availability studies use generic repair methods for the components of the presented architecture. A maintenance study can help increase the system's availability by evaluating different techniques and strategies that can optimize the repair time of a component, as shown in later sections. Few of the studies surveyed were concerned with studying maintainability aspects. This dissertation analyzes the impact of three different maintenance strategies on system availability.

Context

Most articles can be classified into three general contexts: real-time, optimization, smart cities and traffic control. Below each context is commented separately.

Real-time and Others. Two articles were classified in the real-time context (ZHANG et al., 2022; ZHIRNOV; LYAKHOV; KHOROV, 2019). Zhang et al. (2022) focuses on optimizing scheduling for accelerating inference and maintaining trustworthiness in video analytics results through edge devices and blockchain technology. Zhirnov, Lyakhov e Khorov (2019) proposes a mathematical model of adaptive bitrate control algorithms for video and web traffic evaluation. Other papers outside the three classifications were classified into specific categories (ALSABAAN et al., 2021; MA et al., 2018; UGLI et al., 2023; SINQADU; SHIBESHI, 2020; LÓPEZ et al., 2013). Alsabaan et al. (2021) proposes a distributed surveillance system using Positive Orthogonal Codes for full coverage guarantee in target areas. Ma et al. (2018) investigates video data characteristics and proposes a campus surveillance video storage system for university campus applications, enhancing query efficiency through key-frame indexing. Ugli et al. (2023) proposes a Cognitive Video Surveillance Management with long short-term memory model for efficient traffic management and public safety in smart cities. Sinqadu e Shibeshi (2020) evaluates traffic surveillance using iFogSim for IoT applications, and proposes a new model for traffic surveillance with improved performance. López et al. (2013) proposes a multi-robot surveillance application integrated with building automation system, featuring a fully autonomous system with automatic battery charging processes.

Optimization. Two papers were classified under the context of optimization (LIN; PHAM, 2022; TIAN et al., 2024). Lin e Pham (2022) discusses optimization methods for security surveillance systems and focus on inspection-maintenance strategies using Nelder-Mead method. Tian et al. (2024) proposes a maintenance optimization for balanced

systems with state-dependent inspection intervals and also propose a method that enhances system availability and minimizes maintenance costs efficiently.

Smart cities. Four papers were classified under the context of smart cities (YOON et al., 2020; KIM; JEONG, 2023; GONÇALVES et al., 2021; LINS et al., 2024). Yoon et al. (2020) proposes a Cloud-based UTOPIA Smart Video Surveillance for smart cities with MapReduce for big video data. Kim e Jeong (2023) evaluates surveillance system performance for smart cities with multiple cameras aiming to find a cost-effective and reliable surveillance system for smart cities. Gonçalves et al. (2021) evaluates the dependability of surveillance systems in smart cities using stochastic models. Lins et al. (2024) proposes a stochastic modeling approach that enhances reliability and availability in drone-based surveillance systems.

Traffic control. Five works were identified in the context of traffic control, including this dissertation (SUN; ALJERI; BOUKERCHE, 2019; JOUBARI; OTHMAN; VÈQUE, 2022; DING; CHEN; YU, 2022; OLAYODE; TARTIBU; OKWU, 2021). Sun, Aljeri e Boukerche (2019) proposes a queueing model-assisted traffic density estimation scheme for vehicular edge computing in dynamic vehicular networks. Joubari, Othman e Vèque (2022) focuses on urban traffic modeling using queuing theory. They study traffic behavior using queuing theory and Markov chains, and develop a stochastic mobility model for urban traffic environments. Ding, Chen e Yu (2022) proposes a Markov chain-based platoon recognition model in mixed traffic with human-driven and connected and autonomous vehicles. Olayode, Tartibu e Okwu (2021) compares Markov Chain Model and Artificial Neural Network for traffic prediction at signalized road intersections. Finally, this dissertation is also framed as traffic control. Work in this category aims to predict or recognize vehicle volumes. This chapter also proposes an architecture for recognizing vehicles in traffic. However, the model proposed here considers broader system performance criteria, such as its response time. Other works have a greater focus on the performance of their improvement proposal, which are algorithms and frameworks. Therefore, the proposed model in this chapter emerged to take advantage of the gap in articles that deal with performance comprehensively in this context.

4 Methodology

The main objective of this work is to develop models to evaluate the performance and availability of a video monitoring system. Figure 5 presents a flowchart that summarizes the strategy used in this work. The presented methodology was used to generate both performance and availability models.

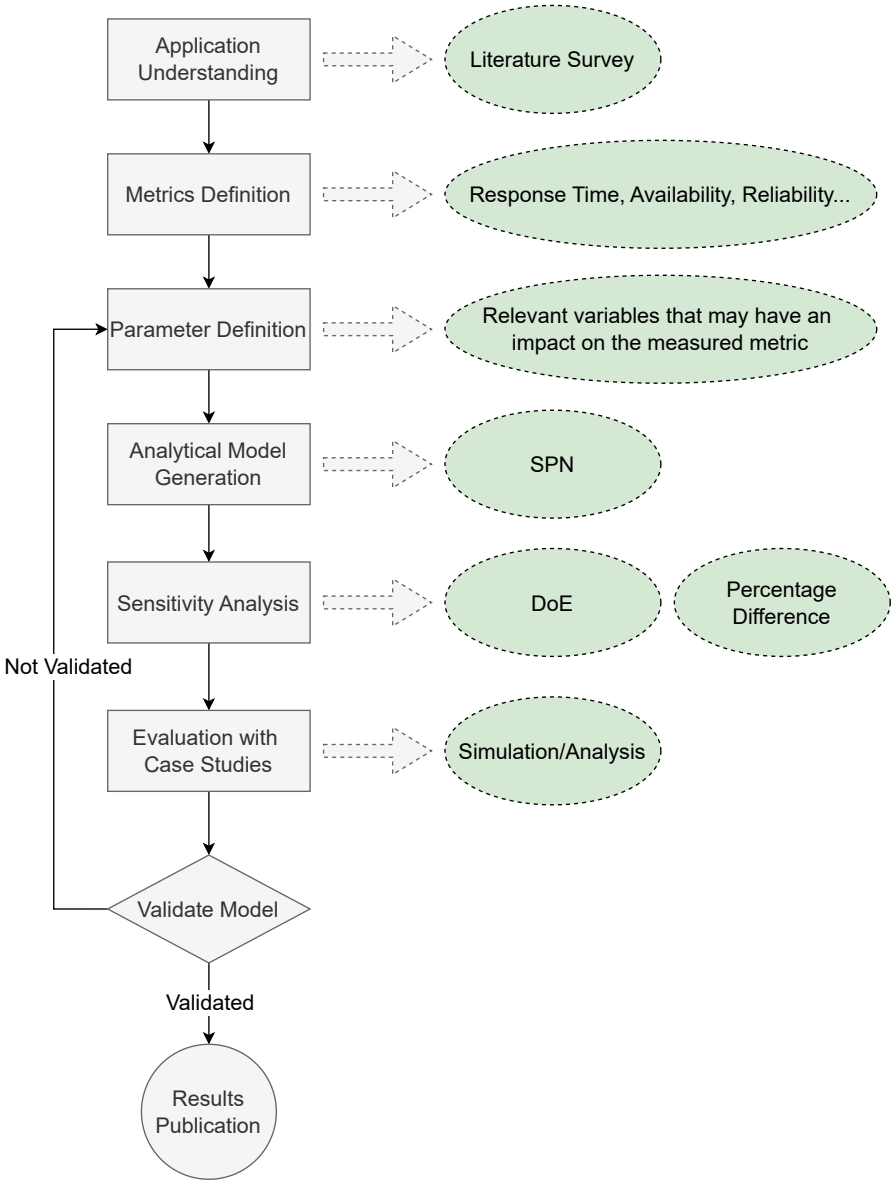


Figure 5 – Work development methodology.

- **Application Understanding:** Understanding how the application operates is

fundamental, and it is necessary to establish the number and type of components involved, in addition to carrying out a survey of the literature about the context to obtain a better understanding.

- **Metrics Definition:** It is important to identify which metrics are relevant, keeping in mind the information that the model can offer to evaluate both the performance and availability of the system. In this work, the metrics selected were: system response time for performance, in addition to complementary metrics, such as utilization and discard rate; system availability; and the rate of calls per year, which will be important to evaluate the maintenance aspect.

- **Parameter Definition:** The parameters that will be inserted into the model are defined. These parameters define the behavior and resource capabilities for the components. In this work, the parameters selected were: the processing times of the components for the performance model; and failure times for each component for the availability model.

- **Analytical Model Generation:** Models are developed at this stage. Each model is built considering the metrics and parameters that were defined in the previous phases. Resulting in a performance model, and five availability area models. The choice of Stochastic Petri Nets occurs because they offer the necessary and sufficient tooling to describe the various behaviors and specifications of the evaluated system.

- **Sensitivity Analysis:** Using sensitivity analysis techniques allows the system designer to understand which components are most critical. The analyzes used here were the DoE and the percentage difference analysis. The percentage difference was used in the context of availability and the DoE was used in the context of performance.

- **Evaluation with Case Studies** Case studies were created to verify the models' ability to generate results. The case studies were designed to also serve as a practical guide on how the user of the model can use it in various analyses.

- **Validate Model:** For the validation process, the two aspects modeled in this dissertation were used. The validation process consists of carrying out measurements of the modeled system in reality and comparing the results with those generated by the model simulation. If the results are statistically similar, we say that a model is validated.

- **Results Publication:** After completing the entire study process and generation of the model, the results found are then disseminated through the submission of articles and the writing of the dissertation.

5 Architecture

This section presents the proposed architecture for the highway monitoring system, which aims to identify vehicle license plates for policing purposes. Figure 6 illustrates this architecture. The chosen architecture is used at the Federal University of Ceará in a project with the support of teachers and students in partnership with the local government. The system operates as follows. Initially, each camera (1) in the system monitors a specific section of the highway and transmits the frames captured via the network, which can be wired or 5G (2). The cell tower's role is critical, acting as a relay point for data transmission from cameras, with an anticipation towards the adoption of 5G technologies. These technologies promise enhanced speed, capacity, reduced latency, and improved support for the IoT (COLLINS et al., 2018). This data is received by a machine that houses three microservices dedicated to processing frames.

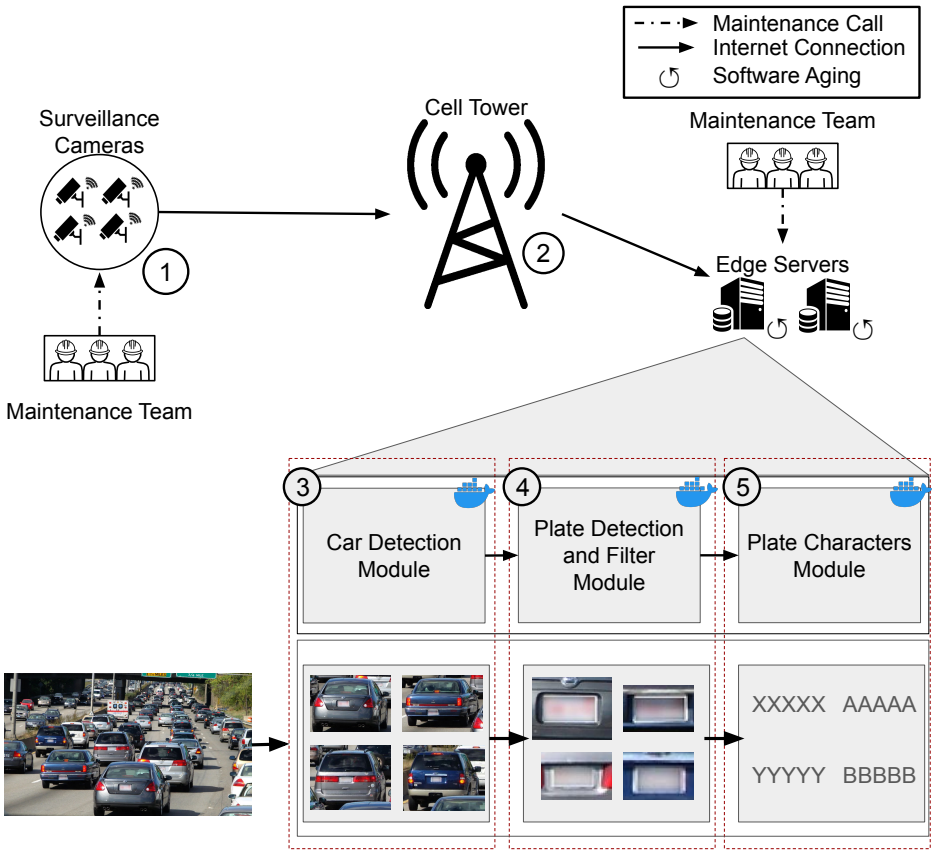


Figure 6 – Video monitoring architecture proposal.

The first part in the data flow is the arrival of frames to the Car Detection module. This module performs the third step (3) by analyzing each frame to check for the presence

of cars. If cars are detected, the module isolates them, generating individual subframes for each vehicle. These subframes are then forwarded to the Plate Detection and Filtering Module for further analysis. This module performs the fourth step (4), which has two main functions. The first is to highlight the license plates of detected cars. If a car's license plate is illegible or not properly framed, that car will be discarded. The second function is to apply filters necessary for the final processing of the license plate. If, for any reason, filtering is not possible, the plate will be discarded. Finally, in the fifth and final step, the Character Extraction Module (5) receives the already filtered plates for processing. This module is responsible for extracting characters from license plates for use in policing and vehicle tracking operations. The system design considers the edge servers' susceptibility to aging, necessitating maintenance protocols for both cameras and servers to ensure their optimal functioning. In the event of a failure in either the cameras or the servers, a maintenance team is dispatched to perform the required repairs, emphasizing the architecture's focus on maintainability and resilience.

In this research, the evaluation of availability encompasses the examination of three distinct maintenance strategies to determine their influence on the system's availability and reliability. Three maintenance routines were chosen for this dissertation: reactive, preventive and auto-repair. Reactive maintenance is characterized by the procedure in which a component must fail before the initiation of a maintenance request. Initially, the system operates under normal conditions until a failure occurs in one of its components. This failed component remains non-operational until its malfunction is recognized by an administrator. Upon detection of the failure, the system administrator is responsible for notifying the maintenance team. Following the notification, the maintenance team is dispatched to the site of the failed component to perform necessary repairs. After the completion of the repair work, the maintenance team departs from the site, allowing the system to resume its standard operational flow from the beginning. The reactive maintenance strategy is noted for its general applicability across various architectural components, distinguishing it as a more universal approach compared to the subsequent strategies detailed in this study.

This surveillance system employs a comprehensive monitoring strategy that encompasses both manual and automated mechanisms to ensure continuous operational integrity. System users and administrators are instrumental in overseeing the operational status, promptly identifying and addressing any anomalies. Complementing this hands-on monitoring, we incorporate advanced automated fault detection systems. These systems utilize sophisticated algorithms to continuously analyze data, enabling them to detect potential issues before they escalate into system failures. Additionally, for critical components such as edge servers, the architecture includes self-repair routines. These routines autonomously address failures, facilitating seamless system operation with minimal downtime. This integrative approach ensures robust monitoring and maintenance, enhancing system reliability

and efficiency.

The preventive maintenance routine shares similarities with reactive maintenance, but it uniquely integrates scheduled maintenance interventions at predetermined intervals. Initially, the system operates under a reactive maintenance regime. At set times, the system transitions into a phase of scheduled preventive maintenance. During these periods, the maintenance team is dispatched to the active component, their visit having been planned in advance. Upon arrival, the team initiates maintenance procedures, necessitating a temporary cessation of the component's operation. Once the maintenance activities are concluded, the team departs from the location, and the component reverts to its standard reactive maintenance protocol. This cycle continues, with the component alternating between reactive maintenance and scheduled preventive interventions, ensuring ongoing system reliability and availability.

The auto-repair approach aims at addressing failures through the autonomous restart of components, targeting issues stemming from software anomalies rather than hardware malfunctions. The procedure begins with a monitoring component vigilantly overseeing the system's operational status. The monitoring component here can be represented by any generic monitoring mechanism, such as a watchdog or a dedicated hardware. Upon detecting a malfunction, the monitor attempts to rectify the issue by initiating a restart of the affected component. If this restart proves successful, the component is restored to its monitored state, ensuring continued system functionality. Conversely, if the restart fails to resolve the problem, the component transitions into a reactive maintenance mode for more thorough intervention. Additionally, should the monitoring component itself encounter a failure, the system similarly defaults to the reactive maintenance protocol. In such instances, the repair process not only addresses the initial fault but also encompasses the restoration of the monitoring component, thereby maintaining the system's resilience and operational integrity.

While standby techniques such as cold, warm, and hot configurations offer robust solutions for ensuring system dependability during failures by duplicating critical components, they also lead to increased acquisition and operational costs. Our research, therefore, focuses on maintenance techniques that optimize component availability and performance. By doing so, we aim to provide insights into alternative strategies that may offer more sustainable cost efficiencies while still maintaining high levels of system reliability.

6 Availability Analysis

This chapter presents the availability architecture, the availability model, the results obtained through its sensitivity analysis via percentual difference; and stochastic analysis performed using appropriate modeling tool, through planned case studies, and the related works. The availability of surveillance systems is a critical aspect that ensures continuous monitoring and security in urban environments. This chapter delves into the architecture and maintenance strategies of a proposed surveillance system, focusing on its ability to remain operational despite potential failures. By examining different maintenance models, we aim to identify the most effective strategies to enhance system reliability and minimize downtime.

The architecture of the surveillance system includes key components such as surveillance cameras, edge servers, and maintenance teams. These components work together to capture, process, and store visual data, which is crucial for monitoring urban areas. The system's design also considers the impact of software aging and the necessity for regular maintenance to prevent failures and ensure optimal performance. To evaluate the system's availability, we employ stochastic Petri net models that simulate various maintenance scenarios. These models help in understanding the effects of reactive, preventive, and auto-repair maintenance strategies on the system's reliability. Through sensitivity analysis and case studies, we demonstrate the practical applications of these models, providing valuable observations for system administrators to enhance the dependability of urban surveillance systems.

6.1 Models Overview

This section elucidates the foundational model, conducts a sensitivity analysis of this base model, details the maintenance models for the strategies discussed earlier, and outlines a model for assessing system reliability.

6.1.1 Base Model

The foundational structure of the proposed architecture is depicted in Figure 7. This model comprises four primary elements, each symbolizing a component of the architectural framework. Sequentially from left to right, the initial element signifies the surveillance cameras within the system. The quantity of tokens in `C_U` mirrors the count of operational cameras at any given instance, with C denoting the system's total camera ensemble. The transition from an operational to a non-operational state for a camera is triggered by

the activation of C_MTTF, resulting in the transfer of a token from C_U to C_D—the latter representing the non-operational state of this component. Conversely, the reactivation of each camera from its dormant state back to functionality is facilitated by C_MTTR, which reallocates a token from C_D back to C_U. Hence, C_MTTF and C_MTTR serve to delineate the mean time to failure and mean time to recovery for the cameras, respectively. The remaining elements of the model—representing the network, the cell tower, and the edge servers—adhere to a similar operational paradigm as that of the camera component.

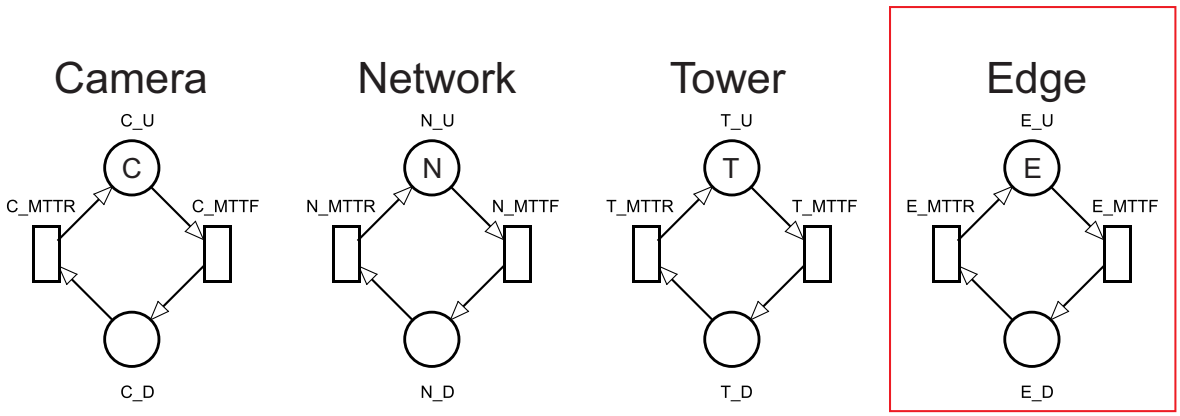


Figure 7 – Base model.

Within the framework of a SPN model, the symbol P denotes probability, while # signifies the count of tokens present in a specified location. The formula to ascertain the availability of the base model is articulated through Equation 6.1, which calculates the likelihood of all components within the base model functioning concurrently. Subsequent sections will delve into a sensitivity analysis of this base model, followed by the introduction of models developed from the foundational model and insights gleaned from the sensitivity analysis.

$$A = P\{(\#E\_U > 0)AND(\#C\_U = N)AND(\#T\_U > 0)AND(\#N\_U > 0)\} \quad (6.1)$$

6.1.2 Sensitivity Analysis

This segment unveils the outcomes derived from conducting a sensitivity analysis on the base model. The analysis facilitates the production of indices for each parameter within the system, enabling the evaluation of their respective impacts on the ultimate metric. For this study, the method of percentage difference analysis has been employed to discern and corroborate the most crucial components of the system. These pivotal components have subsequently been adjusted to formulate new models. The findings from the sensitivity analysis are encapsulated in Table 2.

Table 2 – Sensitivity Result

Parameter	Sensitivity Index
C_MTTF	4.49×10^{-2}
C_MTTR	1.39×10^{-2}
E_MTTF	2.78×10^{-3}
E_MTTR	2.09×10^{-3}
N_MTTF	1.92×10^{-4}
N_MTTR	1.44×10^{-4}
T_MTTF	4.44×10^{-5}
T_MTTR	3.33×10^{-5}

The sensitivity analysis revealed that cameras possess the highest sensitivity indices within the system, highlighting their operational and repair durations as significantly influential on the system's availability compared to other components. Subsequently, the edge server emerged as the second most critical element, with its failure and recovery times ranking third and fourth in terms of importance. The network was identified as the third key component, with its operational metrics positioned fifth and sixth. Conversely, the cell tower was deemed the least impactful component, with its performance metrics ranking seventh and eighth. The prominence of cameras within the model is attributed not only to their marginally shorter failure times but also to their numerical prevalence within the system. The edge server stands as the second most pivotal component, given its relatively brief failure duration among the system elements. The network and cell tower exert minimal influence on the system's overall performance, owing to their longer failure times and their smaller numbers in comparison to cameras and servers. These insights not only highlight the most critical components within the system but also affirm the accuracy of the modeling approach employed in subsequent sections.

6.1.3 Maintenance Models

This segment delineates the maintenance models developed for the base model. Figure 8 introduces a model that implements reactive maintenance for both the edge component and the cameras. Within the camera component, the protocol adheres to the traditional reactive maintenance process. The variables C and CT are utilized to specify the quantity of cameras and the corresponding maintenance teams, respectively. Given their negligible impact as established by the sensitivity analysis, the cell tower and network components remain unaltered in this model. Additionally, these components are often managed by external entities, rendering their maintenance beyond the purview of system administrators. The variables T and N enable adjustments to the number of cell towers and networks, respectively. The Server 1 component symbolizes an edge server subject to a standard reactive maintenance schedule, with the variable M allowing for the modification of the number of server maintenance teams. The inclusion of multiple servers within the

system is facilitated by replicating the Server component 1 S times, where S denotes the total number of servers.

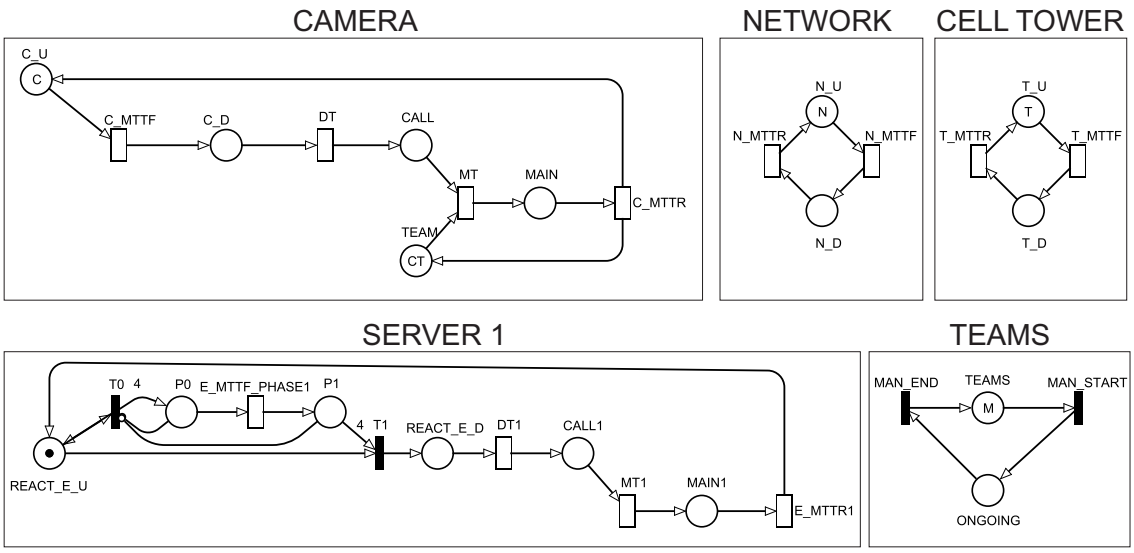


Figure 8 – Reactive maintenance model.

Figure 9 delineates the Camera component with enhanced clarity. The state of camera operation is symbolized by the place `C_U`, whereas the place `C_D` signifies the cameras' non-operational state. The transition between active and inactive states is facilitated through the activation of the transition `C_MTTF`, indicative of the mean time until failure of the component. Upon failure detection, the transition `DT` is activated, transferring tokens from `C_D` to `CALL`. The place `CALL` denotes the recognition of failure and the subsequent notification of the maintenance team. The maintenance team's readiness is represented by the place `TEAM`. The transition `MT` encapsulates the duration required for the maintenance team to reach the malfunctioning camera. The activation of `MT` transition results in the consumption of tokens from both `CALL` and `TEAM` places, reallocating them to the place `MAIN`, which illustrates the repair phase for the cameras. The mean duration for camera repair is represented by the transition `C_MTTR`. Triggering `C_MTTR` transition facilitates the restoration of a token to the `TEAM` place and the return of a camera to operational status by moving a token back to `C_U`. Table 3 shows the server semantics for every timed transition of this model.

Figure 10 provides a detailed view of the server component. The place `REACT_E_U` indicates the operational status of the edge server, while `REACT_E_D` signifies the server's non-operational, or downtime, state. The transition between these states occurs upon the activation of transition `T1`, which is designated to represent the mean time until the server experiences a failure. Once the period allocated for failure detection has passed,

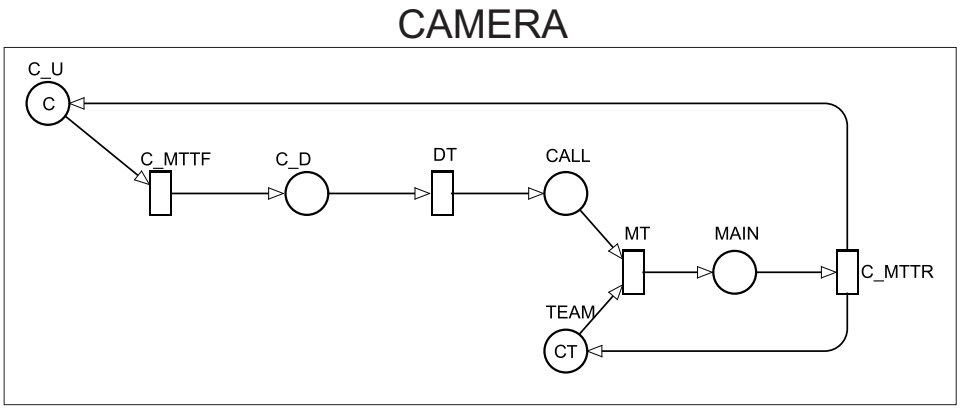


Figure 9 – Reactive maintenance model - Cameras.

Table 3 – Reactive Model Server Semantics

Transition	Semantic
C_MTTF	Infinite Server
DT	Infinite Server
MT	Infinite Server
C_MTTR	Infinite Server
N_MTTF	Infinite Server
N_MTTR	Infinite Server
T_MTTF	Infinite Server
T_MTTR	Infinite Server
E_MTTF_PHASE1	Single Server
DT1	Infinite Server
MT1	Infinite Server
E_MTTR1	Infinite Server

the transition `DT1` is triggered, effecting the removal of a token from `REACT_E_D` and its placement into `CALL1`. This action into `CALL1` delineates the moment of failure recognition and the subsequent summoning of the maintenance team for repair activities.

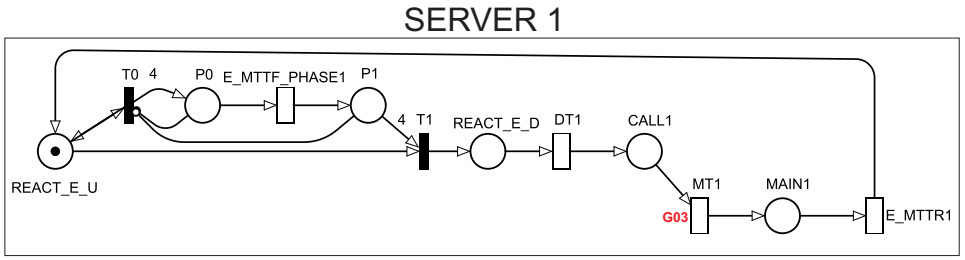


Figure 10 – Reactive maintenance model - Edge Server.

The journey of the maintenance team to the server is encapsulated within the transition `MT1`, which quantifies the time required for the team to arrive at the server's location. The activation of `MT1` results in the relocation of a token from `CALL1` to `MAIN1`, symbolizing the commencement of repair activities. This transition is contingent upon

the availability of maintenance teams, as indicated by tokens present in the TEAMS place, which will be discussed further. MAIN1 signifies the phase where the server is under repair. The transition E_MTTR1 delineates the mean duration necessary for server restoration. Triggering E_MTTR1 transfers a token from MAIN1 back to REACT_E_U, thereby reinstating the server to operational status.

The server's susceptibility to software aging is modeled through interactions between REACT_E_U and transition T1. Upon the server's initiation, four tokens are positioned within P0. The movement of tokens between P0 and P1 illustrates the software aging process. Place P4 monitors the progression of aging, with the accumulation of four tokens in P4 triggering the aging fault through activation of T1. The transition E_MTTF_PHASE1 specifies the average duration of each aging phase for server 1, with the aggregate of these durations representing the total aging period for the server.

Figure 11 displays the mechanism overseeing the allocation of maintenance teams to the server. The TEAMS place holds the total number of available maintenance teams for the edge server, whereas ONGOING reflects the count of teams presently engaged in maintenance tasks. The condition G01, integrated into transition MAN_END, is activated when the number of teams in ONGOING surpasses ongoing maintenance activities. Conversely, condition G02, associated with transition MAN_START, is triggered when the occurrence of maintenance exceeds the teams denoted in ONGOING. Accordingly, the introduction of additional servers is governed by conditions G01 and G02. Condition G03 collaborates with G01 and G02 to ensure maintenance commencement is dependent on team availability. Table 4 provides an exemplification of a guard condition applied within the model.

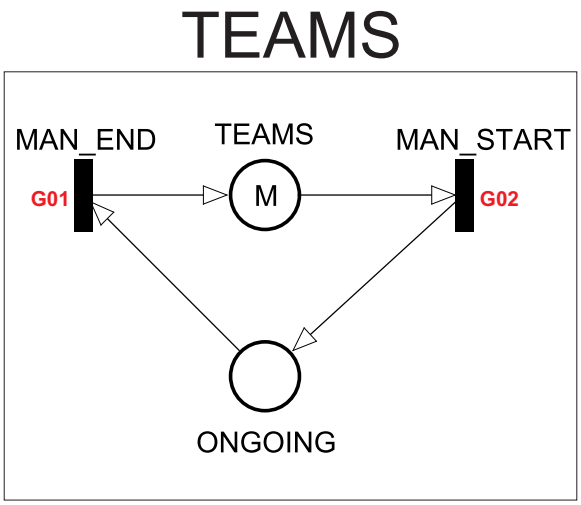


Figure 11 – Reactive maintenance model - Edge Teams.

Figure 12 depicts the base model augmented with both preventive and reactive maintenance strategies. The operational dynamics of this model are largely consistent with those outlined in Figure 8. The preventive model seeks to enhance system availability

Table 4 – Example of guard conditions for control of maintenance teams.

Label	Condition
G01	(#ONGOING) >(#MAIN1)
G02	(#ONGOING) <(#MAIN1)
G03	#TEAMS >0

by integrating a software rejuvenation protocol into the server operations, deemed here as preventive maintenance. The orchestration of preventive maintenance is facilitated by a discrete component represented at the model's lower segment, referred to as the clock. The `CLOCK` place signifies the server's progression towards a state necessitating preventive maintenance. Activation of the `START` transition initiates the rejuvenation process, with the `MAIN_START` place indicating the commencement of preventive maintenance activities. The `START` transition may employ either a deterministic or an Erlang distribution, the latter of which renders the model analytically tractable by approximating a deterministic behavior. Table 5 shows the server semantics for the new timed transition for this part of the model.

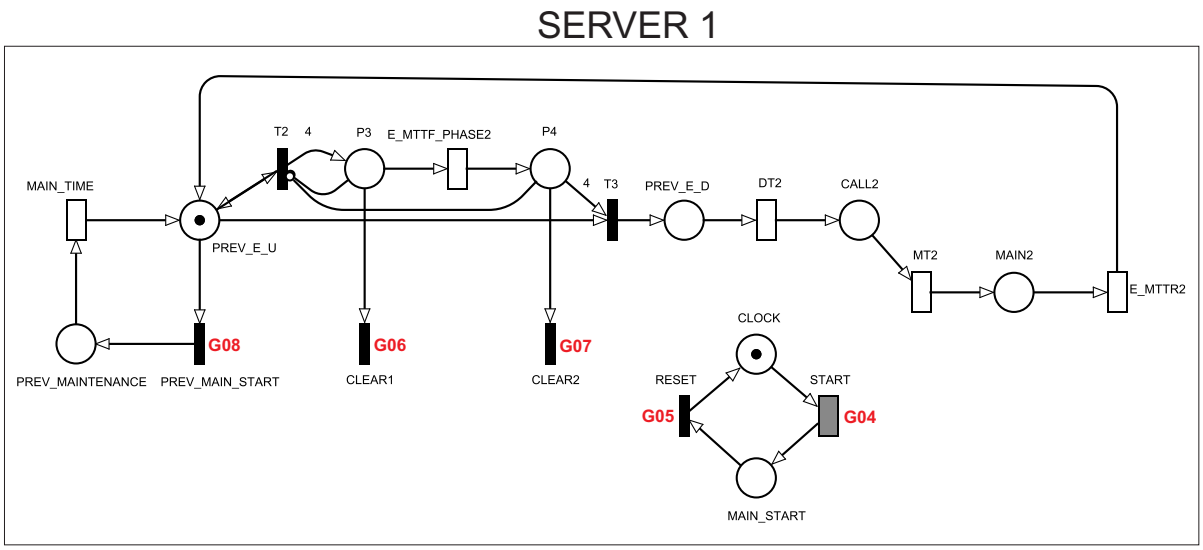


Figure 12 – Preventive-reactive maintenance model.

Table 5 – Preventive Reactive Model New Server Semantics

Transition	Semantic
E_MTTF_PHASE2	Single Server
DT2	Infinite Server
MT2	Infinite Server
E_MTTR2	Infinite Server
START	Single Server
MAIN_TIME	Single Server

Software rejuvenation is implemented in our system to proactively manage the gradual degradation of software performance and prevent unexpected failures. This technique involves periodically halting the software to clear its runtime state and then restarting it, which refreshes the system's resources. By resetting the software to an initial "young"state, we mitigate the risk of errors that typically accumulate over time due to memory leaks, data corruption, or resource exhaustion. The rejuvenation process thus directly contributes to enhancing system reliability by reducing the likelihood of system crashes and performance bottlenecks. Additionally, strategic scheduling of rejuvenation intervals based on system workload and performance data ensures minimal disruption, thereby maintaining high system availability. This preventative approach is crucial for systems requiring high dependability, ensuring they remain robust against both gradual degradation and abrupt malfunctions (VINÍCIUS et al., 2022; SILVA et al., 2022).

On the leftmost side of the model, the `PREV_MAIN_START` transition starts server preventive maintenance and is enabled when a token is in place `MAIN_START`. The `PREV_MAINTENANCE` place indicates that the server is undergoing preventive maintenance and enables the `RESET` transition that resets the clock component. The `MAIN_TIME` transition indicates the time needed to do the preventive maintenance on the server, which should normally be less than the reactive maintenance time. After the preventive maintenance, the server becomes active again. The transitions `CLEAR1` and `CLEAR2` are used to reset server aging by consuming the tokens in `P3` and `P4`. Both transitions are enabled once preventive maintenance is started. All behaviors described above are portrayed in the model by the guard conditions G04, G05, G06, G07, and G08. Table 6 shows examples of guard conditions that guarantee the correct functioning of the model.

Table 6 – Examples of guard conditions for maintenance and preventive maintenance control.

Label	Condition
G04	#PREV_E_U >0
G05	#PREV_MAINTENANCE >0
G06	(#PREV_MAINTENANCE >0) AND (#P3 >0)
G07	(#PREV_MAINTENANCE >0) AND (#P4 >0)
G08	#MAIN_START >0

Figure 13 illustrates the base model augmented with reactive maintenance and the incorporation of an auto-repair mechanism. This model shares substantial similarities with the previously described reactive model. Situated in the figure's lower-left quadrant is a component designated as the monitor, tasked with overseeing system operations and initiating corrective measures upon server malfunction. The place `MON_U` denotes the monitor in its operational state, whereas `MON_D` indicates a scenario where the monitor is non-functional. Transitions between the operational and non-operational states of the monitor are governed by the activation of transition `MON_MTTF`, which calculates the mean

duration until the monitor's failure, and transition MON_MTTR, specifying the average time required for monitor restoration. Table 7 shows the server semantics for the new timed transition for this part of the model.

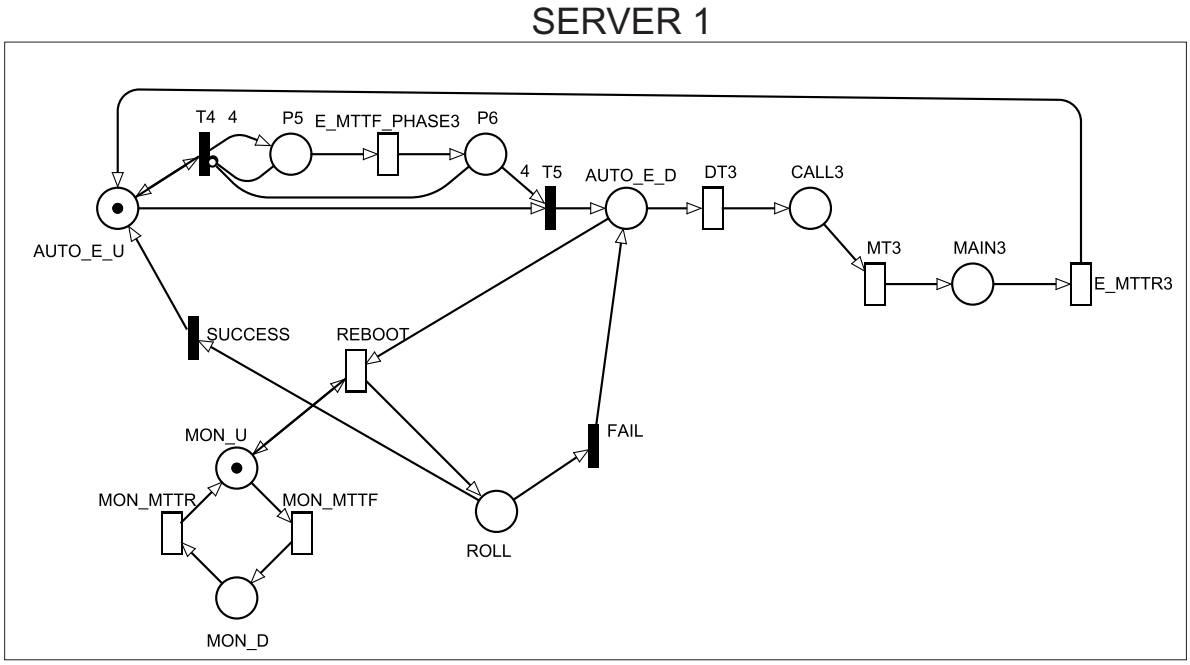


Figure 13 – Auto-repair maintenance model.

Table 7 – Auto-repair Model New Server Semantics

Transition	Semantic
E_MTTF_PHASE3	Single Server
DT3	Infinite Server
MT3	Infinite Server
E_MTTR3	Infinite Server
REBOOT	Infinite Server
MON_MTTF	Infinite Server
MON_MTTR	Infinite Server

Upon detecting tokens within AUTO_E_D, the monitor initiates an attempt to reboot the server. This action is quantified by the REBOOT transition, which delineates the average duration required for server restart. The transitions REBOOT and DT are designed to execute concurrently, reflecting the intent for both the reboot and failure detection processes to proceed in tandem. For this strategy to be viable, the server's restart period must be less than the time taken to detect a failure. Activation of the REBOOT transition results in the consumption of a token from AUTO_E_D and the subsequent allocation of a token to ROLL. This intermediate ROLL place serves to verify whether the reboot has effectively rectified the issue. Should the reboot fail to solve the problem, the FAIL transition is engaged.

Conversely, if the reboot is successful, the `SUCCESS` transition is invoked. The engagement of `FAIL` transition moves a token from `ROLL` back to `AUTO_E_D`, whereas the activation of `SUCCESS` transition transfers a token from `ROLL` to `AUTO_E_U`. The likelihood of either `SUCCESS` or `FAIL` transitions being activated is determined by their assigned probabilities.

For this study, selected metrics include availability, annual call volume, call failure probability, and system reliability, offering a multifaceted exploration of the models' performance characteristics. Availability across all models is articulated through Equation 6.2, representing the likelihood of concurrent operational status for the edge server, cell tower, network, and cameras. A Service Level Agreement (SLA) has been specified to establish the required operational threshold for the cameras, formulated by Equation 6.3. Here, N represents the total camera count within the system, and L denotes the requisite availability percentage, with L being a fractional value ranging from 0 to 1.

$$A = P\{(\#E\_U > 0)AND(\#C\_U >= SLA)AND(\#T\_U > 0)AND(\#N\_U > 0)\} \tag{6.2}$$

$$SLA = N \times L \tag{6.3}$$

Equation 6.4 delineates the $CallsPerYear$ metric, which estimates the annual volume of maintenance calls for a specific component. This metric is calculated by taking the expected number of tokens in the `CALL` place for either a camera or server, denoted as N, dividing this by the mean travel time (`MTT`) for the maintenance team to reach the component, and then multiplying the result by 8760, the total hours in a year. This approach quantifies the frequency of maintenance interventions required over the course of a year for each component.

Equation 6.5 introduces the $Probability\_of\_Unattended\_Calls$ metric, which assesses the likelihood that a maintenance request for a component goes unattended due to the unavailability of a maintenance team. This metric is calculated based on the probability of having tokens in the `CALL` place for the camera or server (N) while concurrently having no tokens in the corresponding maintenance team's place, indicating a lack of immediate maintenance support availability. Although the example metrics provided are applied to a scenario involving a singular server, the inclusion of additional servers necessitates their integration into the metrics calculation to accurately reflect the system's maintenance demands.

$$CallsPerYear = (E\{\#CALL_N\}/MT_N) \times 8760 \tag{6.4}$$

$$Probability\_of\_Unattended\_Calls = P\{(\#CALL_N > 0)AND(\#TEAMS = 0)\} \tag{6.5}$$

6.1.4 Reliability Model

This segment introduces the reliability model applicable to the architecture under consideration. Figure 14 showcases the reliability model, focusing on the reactive maintenance routine. It is important to note that models for other maintenance routines would exhibit analogous operational principles, thereby yielding similar outcomes. In the context of Petri nets, reliability models are derived from availability models with a key modification: a component is rendered inactive following its initial failure. This operational shift is signified by the `CONF` place, which acts as a reliability trigger within the model. The presence of a token in `CONF` configures the model to operate in reliability mode, whereas the absence of tokens switches the model to availability mode. When in reliability mode, the model inactivates transitions identified in red. This action prevents the execution of repair processes for components, thereby maintaining them in a state of inactivity.

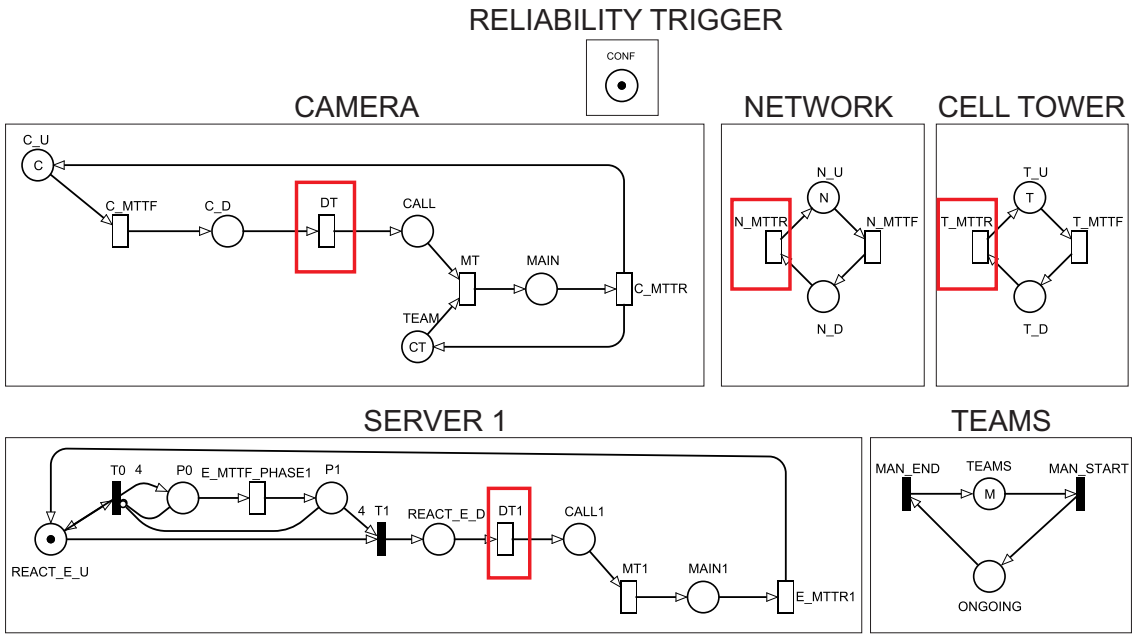


Figure 14 – Reliability Model.

Equation 6.6 gives the system's reliability. The metric is given by one minus unavailability, in other words, the probability that any system component will fail.

$$Reliability = 1 - P\{(\#SERVER\_U = 0)OR(\#CAM\_U < SLA)OR (\#N\_U = 0)OR(\#T\_U = 0)\} \quad (6.6)$$

6.2 Case Studies

In this section, we elucidate the findings derived from the examination of the models introduced in Section 6.1. Subsequent subsections delve into the availability and reliability analyses of all the models discussed. The experimental outcomes were procured via stationary analysis for assessing availability and transient analysis for evaluating reliability. The Mercury Tool, version 5.0.1 (PINHEIRO et al., 2021), served as the computational resource for modeling and result generation. Table 8 delineates the parameters employed for each system component, with the specified values sourced from previous scholarly contributions in this domain (MELO et al., 2018; GONÇALVES et al., 2021). Additional parameters, such as the interval between server updates and the time to detect failures, were determined based on empirical considerations, adhering to a timeframe quantified in minutes.

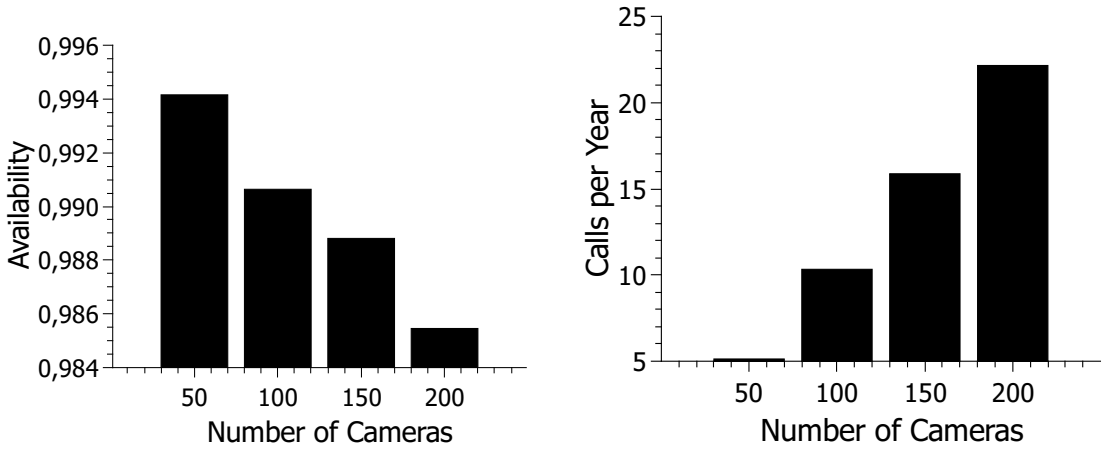
Table 8 – Model Parameters

Component	MTTF (h)	MTTR (h)
Edge Server	4765.06	0.96
Network	83220.0	12.0
Cell Tower	150000.0	5.0
Camera	3603.95	5.0

6.2.1 Case Study 1 - Cameras

This subsection presents an analysis focused on how varying the quantity of cameras influences system availability and the annual call volume required by the maintenance team. Figure 15a illustrates the availability outcomes associated with different camera quantities alongside a server. The configuration incorporating 50 cameras exhibited the highest system availability, registering at approximately 99.40%. Expanding the camera count to 100 resulted in a slight reduction in availability, approximately 0.30%. With the inclusion of 150 cameras, the system's availability further declined to around 98.90%, and with 200 cameras, it slightly dipped to 98.85%. It is evident that an increase in the number of cameras correlates with a marginal decline in system availability, tallying a total decrease of about 0.55% from 50 to 200 cameras. This trend suggests that integrating additional cameras introduces more potential points of failure, thereby diminishing the overall system availability. Notably, the decline in availability does not follow a linear trajectory;

Thus, the model elucidates a scenario in which the augmentation of cameras does not precipitate a significant erosion in availability. In practical terms, this implies the system could feasibly support up to 150 cameras without incurring substantial availability compromises compared to a setup with 100 cameras.



(a) Availability varying the number of cameras. (b) Calls per year varying the number of cameras.

Figure 15 – Results of the impact of camera variation on the evaluated metrics.

Figure 15b illustrates the annual frequency of service requests directed towards the maintenance team, correlating with the quantity of cameras managed by a single server. An examination of the data reveals that a setup consisting of 50 cameras necessitated merely five service requests within a year, whereas the implementation of 100 cameras led to 11 requests. The frequency of service requests continued to escalate, reaching 16 with 150 cameras, and peaking at 23 with 200 cameras. The range of service requests observed annually spanned from 5 to 23, corresponding to the setups with 50 and 200 cameras, respectively. This pattern suggests a proportional increase in service requests, approximately 5 to 6 additional requests per annum for every increment of 50 cameras. However, it is important to note the potential for a non-linear escalation in service requests for configurations exceeding 200 cameras, underscoring the criticality of such analytical evaluations. The direct correlation between the augmentation of camera numbers and the heightened probability of operational failures highlights an imperative trade-off: while an expanded surveillance apparatus can potentially enhance security measures, it also necessitates more frequent maintenance interventions to ensure optimal functionality and availability. Thus, a strategic calibration of camera deployment in tandem with maintenance readiness is essential to maximize security benefits while maintaining system reliability.

Figure 16 provides an analytical overview of the interplay between the number of cameras and maintenance teams through a combinatorial approach. This simulation aimed to assess the likelihood of service requests going unattended within specified operational parameters—namely, a detection time of five minutes and a team's response time of one

hour. The variable range for this analysis included camera quantities from 100 to 1000 and team configurations from 1 to 10. The number of people in a team has been abstracted, but it should be considered that the team has as many people as are necessary for maintenance. The results indicate a threshold wherein a configuration of ten teams effectively reduces the probability of unattended service requests to nearly 0% for up to 400 cameras. Beyond this threshold, the introduction of additional teams or the adoption of more efficient response strategies becomes necessary. The analysis further suggests alternative configurations under varying operational tolerances—for instance, accommodating up to a 15% likelihood of unattended requests with either 600 cameras managed by nine teams or 400 cameras with two teams. Notably, the efficacy of increasing team numbers plateaus with setups exceeding 800 cameras, thereby indicating no substantial improvement in response rates. This insight underlines the importance of strategic planning in surveillance system design, balancing the scale of surveillance capabilities with the operational efficiency of maintenance teams to ensure prompt and reliable service.

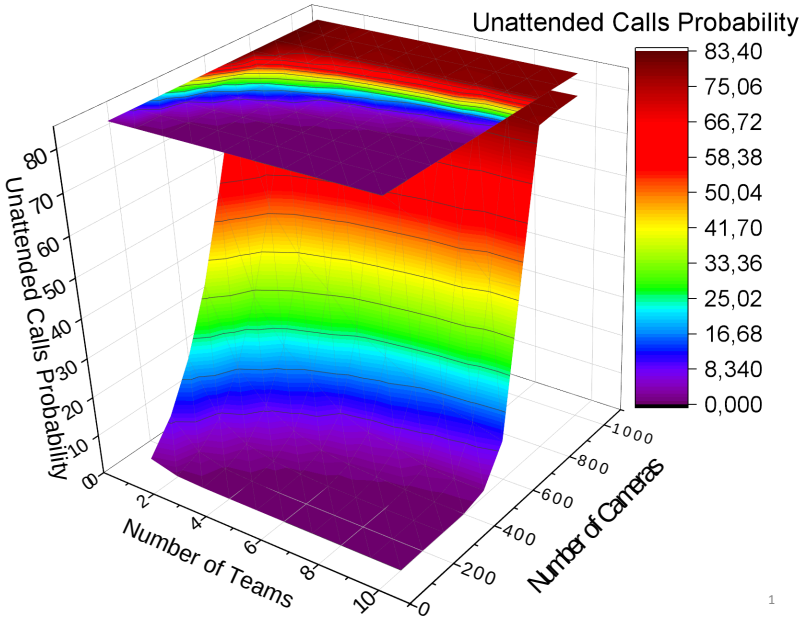


Figure 16 – Probability of a maintenance call not being answered for different numbers of cameras and teams.

Figure 17 delineates a combinatorial investigation focusing on the interrelation between the quantity of surveillance cameras and the assembly of maintenance teams, with the objective of discerning their collective influence on the overarching system availability. This analysis was conducted under the premise of adhering to a SLA stipulating a camera availability benchmark of 99%. Although the variability of outcomes in this study presented a narrower scope compared to the analysis of non-attendance probabilities, valuable insights were nonetheless gleaned. The findings from this examination reveal that within the ambit of the specified SLA, the deployment of merely three maintenance teams can significantly

alter the availability metrics for a system encompassing between four hundred to eight hundred cameras. This outcome elucidates the pivotal role of maintenance team quantity in achieving and maintaining desired levels of system availability within the defined camera range. The choice of a 99% SLA threshold was intentionally selected to emulate conditions of a more stringent operational environment. However, it is posited that adjusting the SLA to less rigorous standards may yield more nuanced and potentially enlightening observations regarding system behavior under varied operational exigencies.

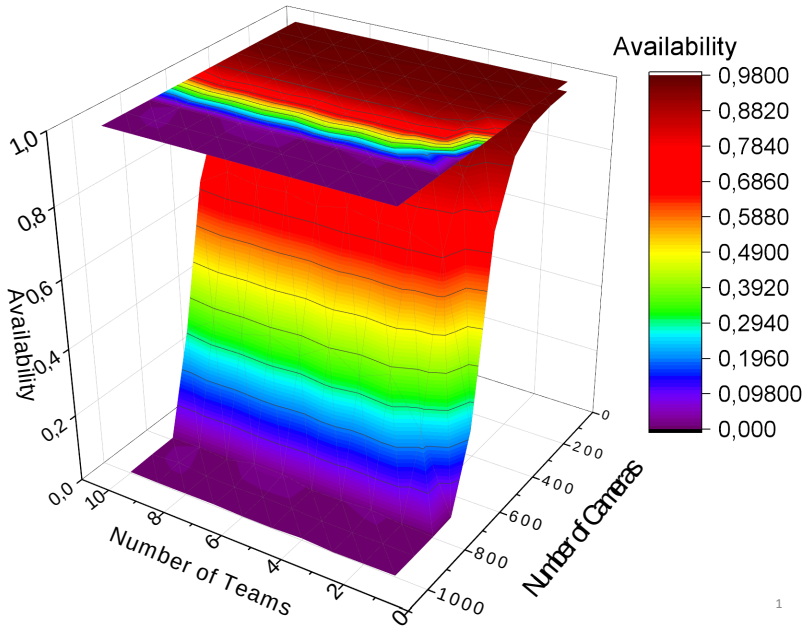
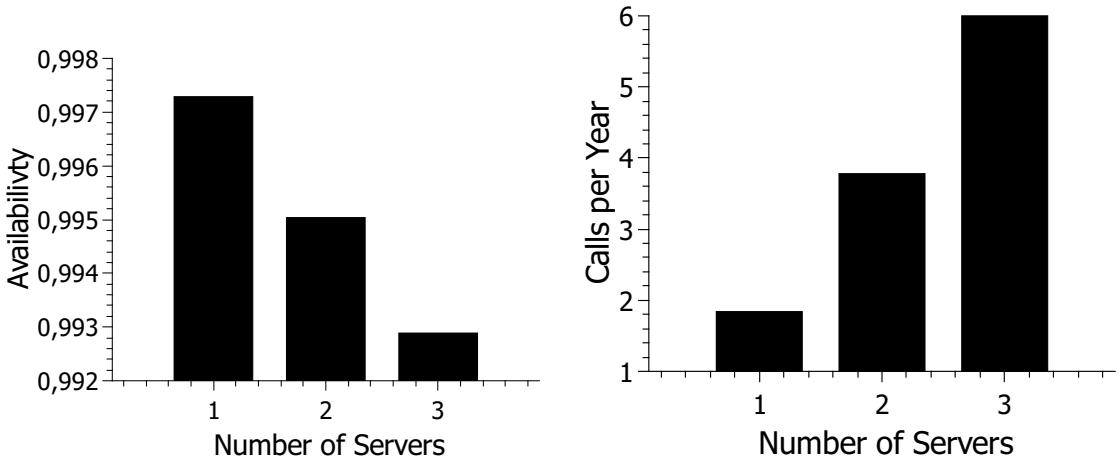


Figure 17 – General availability for different numbers of cameras and teams.

6.2.2 Case Study 2 - Server

This subsection elaborates on the outcomes derived from manipulating the number of servers on relation to system availability and the annual frequency of maintenance calls. The scope of this investigation was limited to scenarios in which the number of cameras was consistently fixed at five units. Figure 18a elucidates the availability metrics observed upon adjusting the server quantity. Initial observations conducted with a single server indicate a system availability of approximately 99.75%. Introducing an additional server reduced the availability slightly to about 99.52%, and the inclusion of a third server further diminished availability to approximately 99.32%. This decremental trend, spanning from one to three servers, amounted to a total availability reduction of roughly 0.43%. Analogous to the findings related to camera increments, this experiment corroborates that escalating server numbers inversely impacts system availability, attributable to the heightened likelihood of server failures. Contrary to the camera-centric scenario, the server variation exhibited a more predictable linear decline in system availability, yet it consistently remained above the 99% threshold. Nonetheless, the comparative analysis suggests that camera adjustments are

more conducive to preserving system availability than server modifications. This observed discrepancy in server variation impact is attributed to the operational reality that fewer servers are necessitated for managing camera clusters, coupled with the fact that servers, unlike cameras, are associated with significant acquisition expenses.



(a) Availability varying the number of servers. (b) Calls per year varying the number of servers.

Figure 18 – Results of the impact of server variation on the evaluated metrics.

Figure 18b delineates the annual frequency of service calls to the maintenance team, contingent on the number of servers deployed. Within the framework of this investigation, the employment of a solitary server necessitated a single maintenance call over the course of a year. This frequency escalated to approximately four calls with the integration of two servers, and further to six calls upon the addition of a third server. The incrementation from one to three servers resulted in a variation of four additional calls annually. Contrasting with the fluctuations observed in the camera-centric scenario, in which the increase approximated two calls per additional scenario, this pattern suggests that the inclusion of each server could predictably lead to an increase of approximately two maintenance calls per annum. Notably, the magnitude of calls attributed to server variations was substantially less than that observed in the camera variation scenario of Case Study 1, by more than a threefold difference. Nevertheless, increasing the number of servers has been identified as a strategic measure to mitigate maintenance demands, attributed to the relatively lower failure rates of servers compared to cameras. Consequently, optimizing the number of servers emerges as a pivotal strategy to reduce annual maintenance requirement.

Figure 19 unveils a combinatorial assessment focusing on two critical parameters: the mean server repair time and the server failure detection time, with the experiment maintaining a constant count of five cameras and a single server. The analysis spans an average repair time ranging from 5 to 25 hours, alongside a failure detection timeframe of 0.5 to 2.5 hours. This evaluative scenario aims to delineate the spectrum of operational choices available to an organization, thereby facilitating the selection of the most cost-

efficient alternative. Illustratively, an organization might deliberate between adopting a fault-detection system with a 1.5-hour response time coupled with a 10-hour repair window, versus a system with a 1-hour detection time and a 15-hour repair timeframe. The comparative advantage in terms of system availability positions the former option as more advantageous for the contracting entity. Additionally, in circumstances necessitating a minimum 99.5% system availability with a server repair duration pegged at 20 hours, the analysis infers that the failure detection interval must not exceed 1 hour to uphold the stipulated availability threshold.

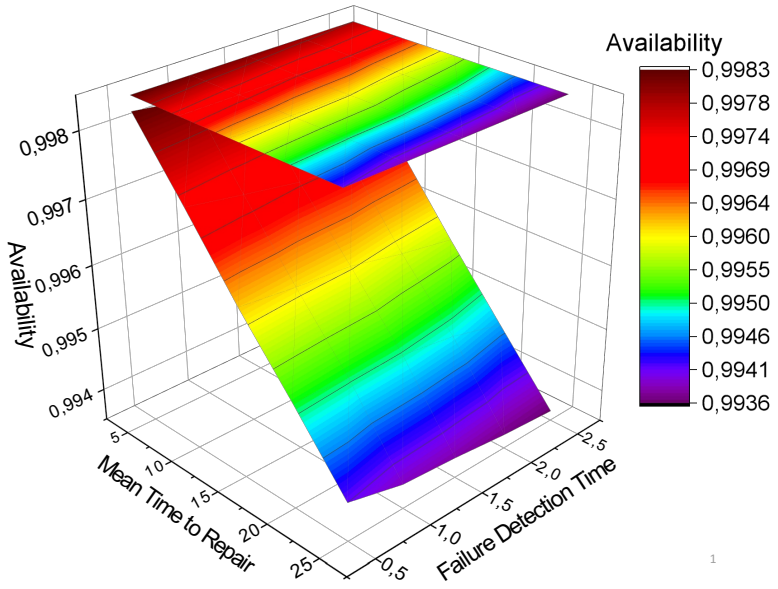


Figure 19 – Availability varying server repair time and team movement time.

6.2.3 Case Study 3 - Maintenance Strategies

This section elucidates the outcomes of altering maintenance strategies, particularly focusing on the intervals between rejuvenation periods. Figure 20 catalogs the variations in system availability as the intervals of rejuvenation are extended by 1000-hour increments. The span of availability observed ranges from approximately 99.95% to 99.82%. Initially, a marginal increase in availability is noted in the first interval, subsequently followed by a pronounced exponential decline. This pattern suggests that protracted intervals between rejuvenations tend to exacerbate the risk of system failures, as the machinery or system components are left unattended for extended durations, thereby increasing their susceptibility to breakdowns. Conversely, the initial increase from 99.92% to 99.95% in availability indicates that there exists an optimal interval for rejuvenation that can enhance system reliability. Overly frequent rejuvenations may disrupt operational continuity, while excessively infrequent ones may lead to premature component failure. Thus, identifying and implementing an ideal rejuvenation frequency emerges as a pivotal strategy for augmenting

system availability, underscoring the delicate balance required to optimize maintenance schedules for enhanced system performance.

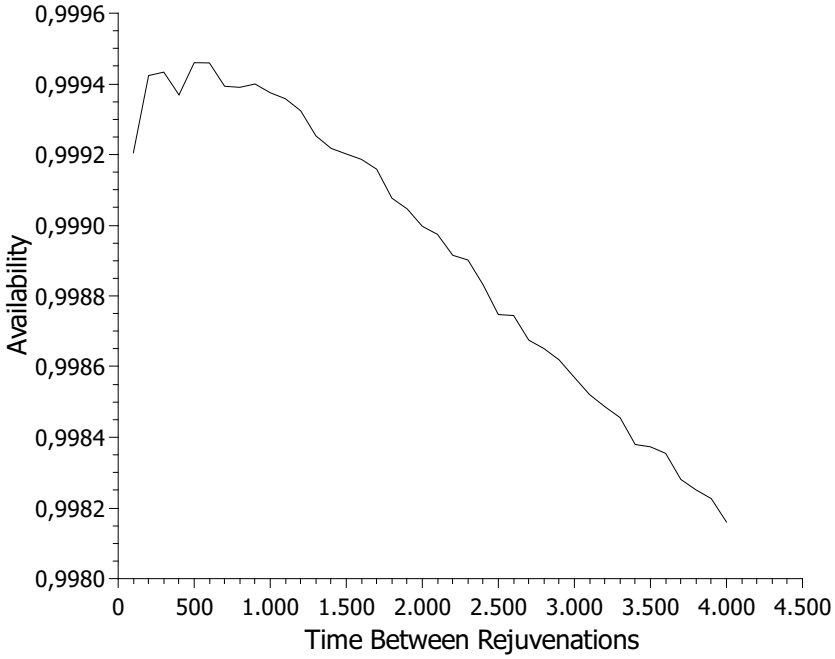


Figure 20 – Availability, varying interval between rejuvenation.

Figure 21 delineates the comparative effectiveness of three distinct maintenance strategies on system availability, with a fixed configuration of five cameras. Among the evaluated models, the reactive maintenance strategy exhibited the lowest system availability, achieving a figure close to 99.72%. In contrast, the self-repair strategy demonstrated a slightly enhanced availability, registering approximately 99.83%, marking an improvement of about 0.11% over the reactive approach. The preventive maintenance strategy, however, surpassed both by attaining an availability rate of roughly 99.94%, thus reflecting a significant increment of approximately 0.22% in comparison to the reactive model. This analysis underscores the efficacy of the preventive strategy, particularly when it involves software rejuvenation at judiciously selected intervals, as a superior method for bolstering system availability, as shown in Figure 20. Despite the commendable performance of the self-repair strategy in mitigating system failures, the application of preemptive maintenance tactics, aimed at regular system rejuvenation, emerges as the most advantageous approach for enhancing overall system reliability.

6.2.4 Case Study 4 - Reliability

This section presents the findings related to the reliability metrics assessed across different models, with a particular focus on the impact of camera reliability on the overall system. Figure 22 provides an overview of the impact of variations in camera reliability

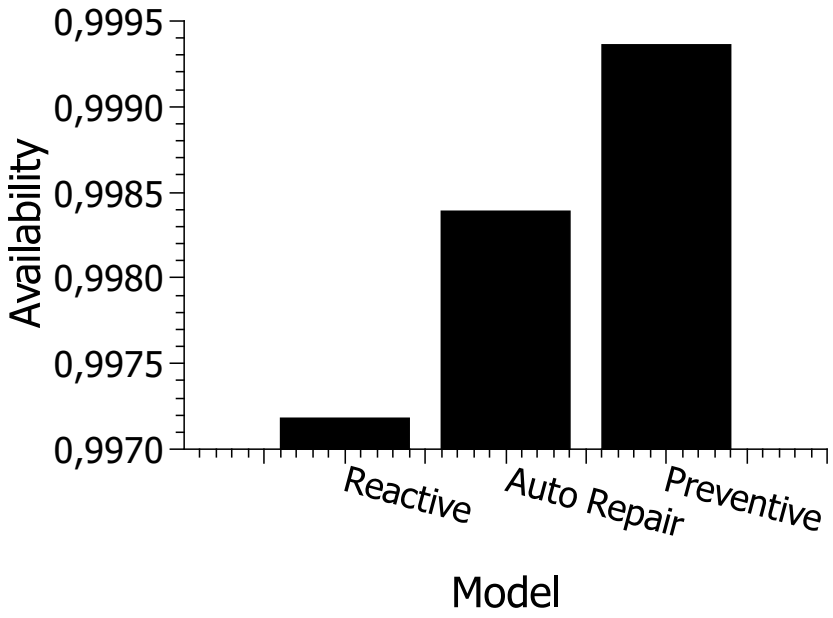


Figure 21 – Availability considering the three maintenance strategies presented.

affect system reliability. The decision to focus on cameras was predicated on their significant influence on the system's sensitivity to failures. In this analysis, the Mean Time to Failure (MTTF) of cameras was adjusted across three distinct intervals: a reduction of 50% from the base value, the base value itself (3603.95 hours), and an increase of 50% above the base value. For the purpose of this examination, both cameras and servers were configured to operate under a reactive maintenance framework. It is inherent to the nature of this experiment that reliability metrics exhibit a decline over time, reflecting the gradual wear and degradation of system components.

MTTF of the camera is reduced by 50%, the observed reliability falls below the benchmarks set in the alternative scenarios. The base MTTF scenario for camera reliability was intermediate, exhibiting higher reliability than the reduced MTTF scenario but lower than the enhanced MTTF scenario. The scenario with an increase of 50% in camera MTTF demonstrated the highest reliability across all variations, consistently outperforming the other scenarios throughout the duration of the experiment. All variations converged towards the x-axis when the experiment exceeded 6000 hours, indicating a decline in reliability over time. This experiment underscores the feasibility of selecting an optimal camera within a given budgetary constraint, provided it satisfies predefined reliability criteria. For instance, should the requirement stipulate a camera that maintains at least 20% system reliability after 500 hours, the base MTTF camera would suffice without necessitating additional expenditure on higher-priced options. Conversely, should the system reliability need to be maintained at 20% after 1000 hours, investment in a camera with a 50% increased MTTF becomes imperative.

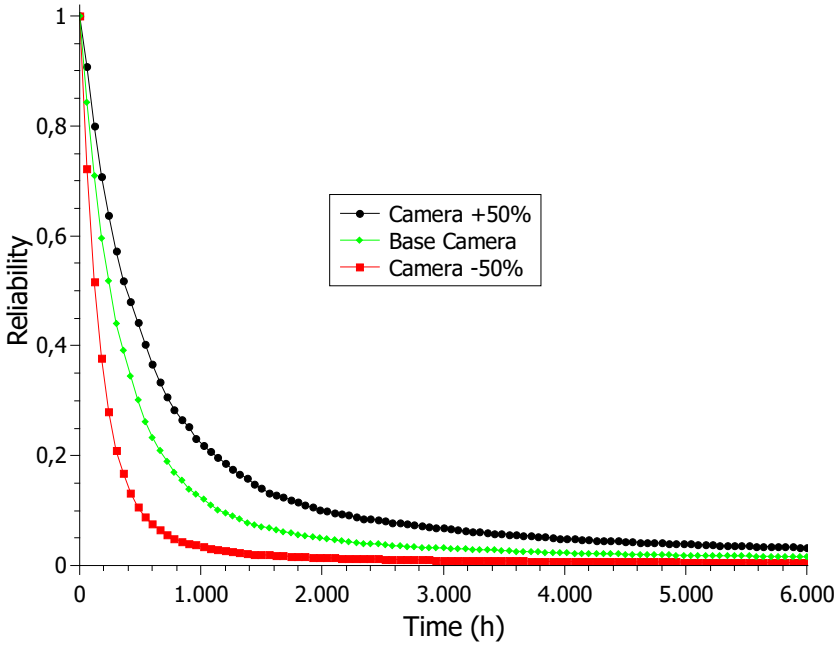


Figure 22 – System reliability by varying camera reliability.

7 Performance Analysis

This chapter presents the performance architecture, the performance model, the results obtained through its sensitivity analysis via DoE; and stochastic analysis performed using appropriate modeling tool, through planned case studies, and the related works.

This chapter outlines the architecture of the video monitoring system, focusing on its components and their interactions. The architecture includes surveillance cameras, edge servers, and the network infrastructure. The system is designed to handle large volumes of video data efficiently, leveraging edge computing to process data closer to the source, reducing latency and improving performance. This chapter also details the development of a performance model using Stochastic Petri Nets (SPNs). This model captures the behavior of the video monitoring system, including the processing times, queuing mechanisms, and the interactions between different components. The model is used to simulate various scenarios and analyze the system's performance under different conditions.

Several performance metrics are defined to evaluate the system, such as response time, utilization, and discard probability. These metrics help in understanding how well the system performs and identifying potential bottlenecks. The chapter also discusses the importance of these metrics in ensuring the system meets the required performance standards. Multiple case studies are presented to serve as a practical guide of the model. These case studies involve varying the number of cameras, traffic intensity, and other parameters to observe their impact on the system's performance. The results from these case studies provide insights into how different factors affect the system and help in optimizing its performance. This chapter provides a comprehensive analysis of the performance aspects of the video monitoring system, offering valuable observations for system administrators to plan and optimize their infrastructure effectively.

7.1 Model

This section presents the model developed considering the peculiarities of the proposed architecture. Figure 23 presents the model. The model is divided into four parts as well as the architecture, namely: (1) Admission, (2) Module A - Car Detection, (3) Module B - Plate Detection and Filtering, and (4) Module C - Extraction of Characters. Module A is responsible for identifying where the cars arriving in each frame are and cutting them into subframes that contain one car each. If there are no cars, the frame is discarded. Module B receives each subframe from the previous module and highlights the

license plate that was found. If there is no visible license plate, the subframe is discarded. Still in module B, a filter is applied to the subframe. If the license plate is not suitable for use in character extraction, the subframe is discarded. Finally, module C receives the subframe with the filter applied and extracts the characters with an Optical Character Recognition (OCR) algorithm.

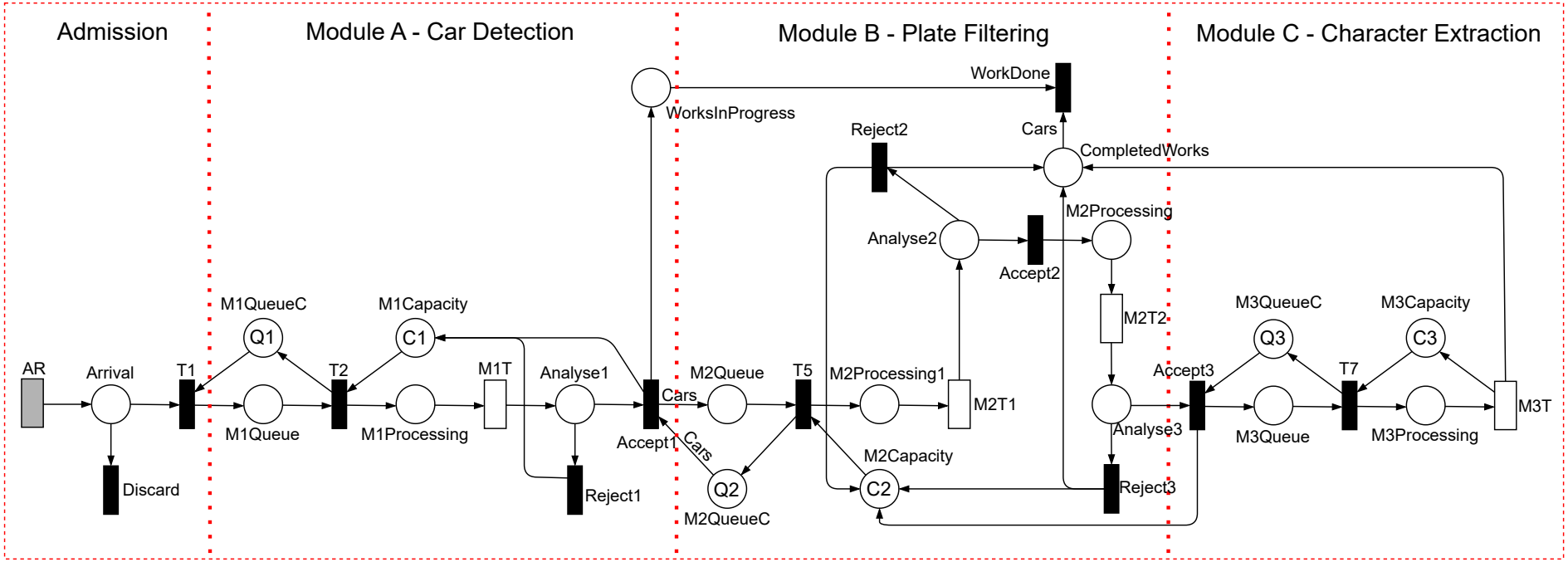


Figure 23 – Model developed based on the proposed architecture, considering specific modules and behaviors.

The first part of the system, admission, represents the entry of frames into the system, where AD is a deterministic transition that indicates the interval between the arrival of frames. The Arrival place stores the frames that arrive in the system, and if there is no capacity in the initial module, the frame is discarded by triggering the Discard transition. The second part of the system is made up of the car detection module. The triggering of the T1 transition indicates the entry of a frame into module A and, consequently, into the system. As long as there is a token in place M1QueueC, the module has queuing capability, and requests will be queued in place M1Queue. If there is processing capacity for the module in place M1Capacity, the transition T2 will be triggered, and the queued requests will be deposited in place M1Processing. The frame processing time for module A is defined by the M1T timed transition. When frame is processed, a token is taken from M1Processing and deposited in place Analyse1. The place Analyse1 represents the state in which a frame was processed and will be analyzed to check whether or not it has visible cars. If the frame does not contain cars, the Reject1 transition is triggered, and the frame is discarded. If the frame has visible cars, the Accept1 transition is triggered, and the frame is accepted for processing in the next component. Both Accept1 and Reject1 transitions are triggered according to the weights determined for them.

The system's third part comprises a plaque detection and filtering module. When the Accept1 transition is triggered, three actions occur: a token is deposited in place WorksInProgress, a quantity Cars of tokens is deposited in place M2Queue, and the same amount is reduced capacity of place M2QueueC. The place WorksInProgress represents how many frames are currently being processed in the system in modules B and C. This place is used to assist in the preparation of system metrics. The variable Cars denotes the number of cars found in a single frame, that is, they are subframes generated from the main frame. The places M2Queue and M2QueueC represent module B's queuing and queuing capacity, respectively. If processing capacity exists, the subframes will be sent to the processing queue by triggering the transition T5.

The places M2Processing1 and M2Capacity represent module B's first processing and processing capacity, respectively. The module's first processing is the time needed to detect the plates in each subframe, represented by the timed transition M2T1. After triggering the M2T1 transition, the frame goes through the second analysis in the system, represented by the place Analyse2. If subframe does not contain a plate, the transition Reject2 is triggered, and frame will be discarded and deposited in place CompletedWorks. If a sign is visible in the frame, the Accept2 transition is triggered. The transition that will be triggered is determined by the weights assigned to each one. The place CompletedWorks indicates how many subframes were processed by modules B and C, whether they were accepted or discarded. If the number of completed jobs is equal to the number of subframes generated by a frame, determined by the variable Cars, a request is considered to have been completed. This way, a token is removed from WorksInProgress, and Cars tokens

are removed from `CompletedWorks` by firing the `WorkDone` transition.

The triggering of the `Accept2` transition indicates that the subframe has a visible plate. The place `M2Processing2` represents that a subframe is undergoing the second processing of module B, the filter application. The time needed to apply the filter to subframe is represented by the timed transition `M2T2`. After applying the filter, a token will be deposited in place `Analyse3` by triggering the transition `M2T2`. The place `Analyse3` represents the system's third and final frame analysis. If the subframe becomes unsuitable for analysis, or if it is not possible to apply the filter for adverse reasons, it will be discarded by triggering the transition `Reject3`, which returns processing capacity to the module and deposits the subframe in place `CompletedWorks`. If the filter is applied successfully, the subframe passes to the next component by triggering the `Accept3` transition. Both transitions contain weights that determine their firing rate.

The fourth part of the system comprises the plate character extraction module. After the transition `Accept3` fires, a token will be consumed from place `M3QueueC` and will be deposited into place `M3Queue`. The places `M3Queue` and `M3QueueC` represent the queue and queuing capacity of subframes of the C module. If processing capacity exists, subframe leaves the queue and enters processing by triggering the transition `T7`. The places `M3Capacity` and `M3Processing` represent the processing capacity and number of items being processed in the C module, respectively. The time required for processing in the C module (character extraction) is represented by the timed transition `M3T`. Triggering the transition `M3T` consumes a token from place `M3Processing`, returns capacity to the module, and deposits a token at place `CompletedWorks`, indicating that subframe was processed successfully.

7.1.1 Metrics

in this work, we consider three metrics: probability of discard, mean response time, and utilization. Equation 7.1 presents the metric used to calculate the system's discard probability. The $Probability\_of\_Discard$ is defined by the probability of no queuing capacity in any of the modules. The metric generates a value between 0 and 1, so by multiplying by one hundred, we obtain a percentage value. Equation 7.2 presents an equation that helps calculate the mean response time. The *Reqs* metric counts the average number of requests in the system, where $E\{\#P\}$ represents the statistical expectation of the number of tokens in a place P. Equation 7.3 presents the metric used to calculate the mean response time. The $Mean\_Response\_Time$ metric is derived from Little's law (JAIN, 1990). The law indicates that the $Mean\_Response\_Time$ is given by the average number of requests in the system, divided by the arrival rate of requests in the system, the AR. The arrival rate is the inverse of the interarrival time. We abbreviate the time between arrivals here as AD. The obtained value is divided by the complement

of the probability of discard, represented as $1 - Probability\_of\_Discard$, to eliminate the impact of discarding on the results. Finally, Equation 7.4 presents the metric used to obtain the utilization of a component. The *Utilization* is given by dividing the number of items being processed in module X, divided by its processing capacity. Like the discard metric, the utilization metric must be multiplied by one hundred to obtain percentage values.

$$\begin{aligned} Probability\_of\_Discard =& P\{((\#M1QueueC = 0))OR((\#M2QueueC = 0)) \\ & OR((\#M3QueueC = 0))\} \times 100 \end{aligned} \tag{7.1}$$

$$\begin{aligned} Reqs =& ((E\{M1QueueC\}) + (E\{M1Processing\}) + (E\{Analyse1\}) \\ & + (E\{WorksInProgress\})) \end{aligned} \tag{7.2}$$

$$Mean\_Response\_Time = \frac{(Reqs \times AD)}{1 - Probabilidade\_de\_Descarte} \tag{7.3}$$

$$Utilization = \frac{(E\{MXProcessing\})}{CAPACITYX} \times 100 \tag{7.4}$$

7.1.2 Design of Experiments

This section shows the results of the model's sensitivity analysis using the DoE technique. Table 9 shows the factors used in the DoE and their adopted levels, while Table 10 shows all combinations between factors and their respective levels. The first three factors, C1, C2, and C3, refer to the processing capacity of each system module, A, B, and C, respectively. Capacities varied between 4 and 16 to simulate the number of cores available for each module. The AD factor was used here to vary the number of cameras in the system, where there was a variation between 1 and 3 60-fps cameras, represented by 16.6666 ms and 5.5555 ms, respectively. These two times were obtained by calculating the inverse of the framerate, from which we obtain the interval between frames Finally, the Cars factor adjusts the traffic intensity, varying the number of cars per frame from 1 to 5.

Table 9 – Design of Experiments.

Factor name	**Low Setting**	**High Setting**
C1	4.0	16.0
C2	4.0	16.0
C3	4.0	16.0
AD	5.55555	16.66666
Cars	1.0	5.0

Table 10 – Combination Table.

C1	C2	C3	AD	Cars	MRT
4.00	4.00	4.00	5.55	1.00	37147.00
4.00	4.00	4.00	5.55	5.00	40458.27
4.00	4.00	4.00	16.66	1.00	40934.98
4.00	4.00	4.00	16.66	5.00	44263.29
4.00	4.00	16.00	5.55	1.00	36964.35
4.00	4.00	16.00	5.55	5.00	40977.95
4.00	4.00	16.00	16.66	1.00	40485.42
4.00	4.00	16.00	16.66	5.00	44918.20
4.00	16.00	4.00	5.55	1.00	37757.41
4.00	16.00	4.00	5.55	5.00	29144.51
4.00	16.00	4.00	16.66	1.00	39456.51
4.00	16.00	4.00	16.66	5.00	27838.26
4.00	16.00	16.00	5.55	1.00	38213.14
4.00	16.00	16.00	5.55	5.00	29815.73
4.00	16.00	16.00	16.66	1.00	39649.25
4.00	16.00	16.00	16.66	5.00	27271.52
16.00	4.00	4.00	5.55	1.00	15192.19
16.00	4.00	4.00	5.55	5.00	13651.94
16.00	4.00	4.00	16.66	1.00	4635.67
16.00	4.00	4.00	16.66	5.00	4865.48
16.00	4.00	16.00	5.55	1.00	15269.23
16.00	4.00	16.00	5.55	5.00	13584.99
16.00	4.00	16.00	16.66	1.00	2732.21
16.00	4.00	16.00	16.66	5.00	4755.87
16.00	16.00	4.00	5.55	1.00	8855.87
16.00	16.00	4.00	5.55	5.00	10184.78
16.00	16.00	4.00	16.66	1.00	249.55
16.00	16.00	4.00	16.66	5.00	3076.53
16.00	16.00	16.00	5.55	1.00	8831.25
16.00	16.00	16.00	5.55	5.00	10048.44
16.00	16.00	16.00	16.66	1.00	250.79
16.00	16.00	16.00	16.66	5.00	3223.17

Figure 24 shows the effect factor graph for the presented model, considering the MRT metric. The graph shows which factors have the most impact on the model results and the most relevant combinations of factors. Factor C1 was the most relevant for the system's response time. The capacity of the first service is more relevant because, depending on it, more or fewer requests enter the system. In other words, if more requests enter the system, the response time also increases. The combination of factors C1 and AD proved to be the most relevant, as together, these factors regulate the number of requests that enter the system. The graph shows information about the factors' absolute impact but does not show which level increases or reduces the MRT. Therefore, below are some graphs of combinations of factors.

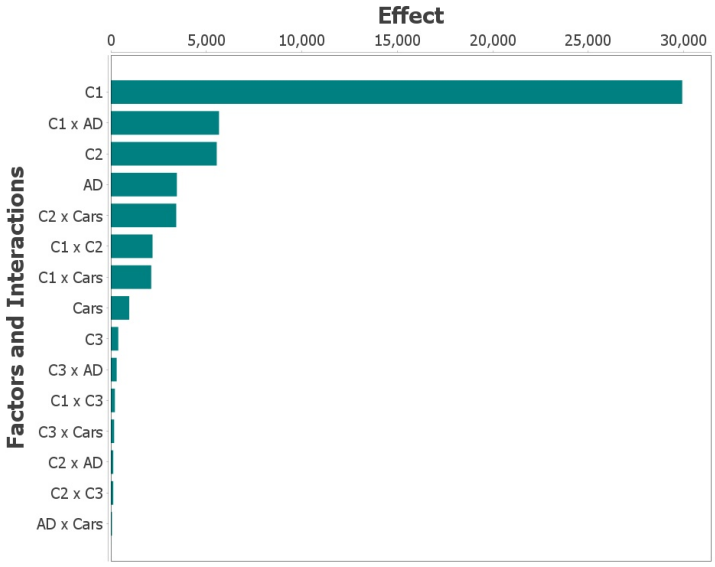


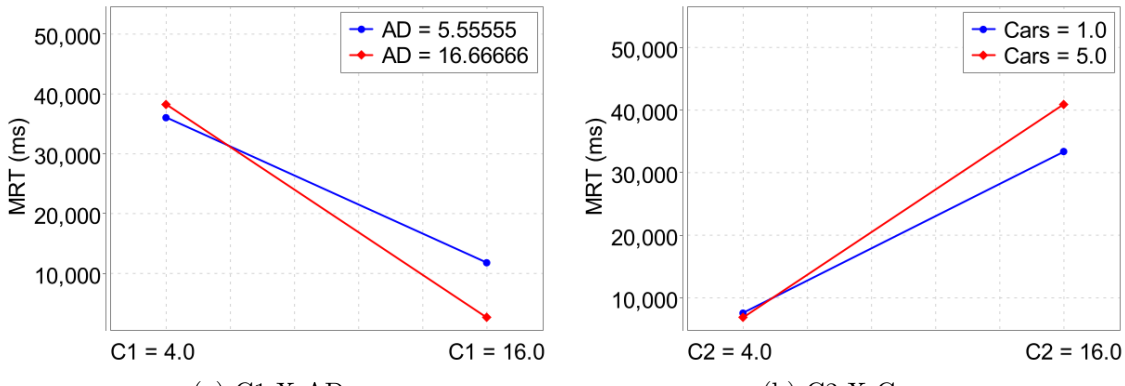
Figure 24 – Factor impact graph.

Figure 25 shows the impact of the four most relevant combinations according to the effect factor graph. The graphs were organized from the most relevant interaction to the least relevant. Figure 25a shows the interaction between factors C1 and AD. The times are very similar for C1, equal to four, with a slightly lower response time with 3 cameras. With a greater capacity, the metric value reduces by around ten seconds, emphasizing the use of a camera. Thus, if module A's capacity is around four, the number of cameras does not make much difference. However, if the capacity is around sixteen, it is recommended that only one camera be used to avoid overload.

Figure 25b shows the interaction between factors C2 and Cars. The results show that with low capacity, the system presents little difference in the number of cars per frame. However, both cases' response time greatly increases when C2 reaches sixteen capacity. Increasing capacity consequently increases the number of requests the system can handle, so we have an increase. The increase amounts to about thirty-two seconds with one car and forty seconds with five cars.

Figure 25c shows the interaction between factors C1 and C2. The results show that, in general, using more capacity in module B is more interesting for response time. The MRT presents values above thirty seconds for any capacity above four in module B. The response time drops a lot when module A increases capacity. The time is below ten seconds for both module B capacities of four and sixteen. Therefore, it is always advantageous to allocate more capacity to module A. It is also interesting to note that although C2 has a smaller impact, the degree of descent with capacity four appears greater than that with capacity sixteen.

Figure 25d shows the interaction between factors C1 and Cars. The results for C1



(a) C1 X AD. (b) C2 X Cars.

(c) C1 X C2 (d) C1 X Cars

Figure 25 – Interaction between factors.

equal to four are similar when varying the number of cars per frame. The difference is around four seconds, with the MRT being shorter for five cars. The behavior of the MRT being smaller when there are more cars or cameras is associated with the excessive discard at the entrance and throughout the system, as shown in the results section. When C1 is set to ten, both MRTs reduce to around ten seconds; this time, the scenario with one camera shows a greater downward trend than the scenario with five.

7.2 Case Studies

In this section, the results generated with the developed model are presented. Due to the complexity of the model, the experiments were carried out using stationary simulations with an approximate error of 2%. The Mercury tool (version 5.0.1) [1] was used to generate the model and consequently its results (SILVA et al., 2015). Table 11 presents the general values used in the components for the experiments. Capacities can be defined using the C1, C2, and C3 settings for modules A, B, and C, respectively. The queue capacities of the modules are given by the definitions Q1, Q2, and Q3, which are

[1] https://www.modcs.org/

the queue capacities of modules A, B, and C, respectively. The acceptance probabilities must be used in the transitions Accept1, Accept2, and Accept3, being the acceptance of modules A, B (detection), and B (filtering), respectively. The discard probabilities must be used in the Reject1, Reject2, and Reject3 transitions, which are the discard of modules A, B (detection), and B (filtering), respectively. The `AR` transition is the only one of the single server type; the rest of the transitions are of the infinite server type. Some of the experiments required different capacity values to generate relevant results; therefore, if the experiment does not specify a change, it must be assumed that these are the values. The time and probability values for each module were obtained by conducting practical experiments. The following subsections present three case studies that show the model's potential and serve as a practical guide for system administrators' future use.

Table 11 – Parameters used in the model.

Component	Capacity	Time	Probability of Acceptance|Discard (%)
Module A	8	166,14 ms	50|50
Module B - Detection	8	145,56 ms	60|40
Module B - Filter		20,0 ms	50|50
Module C	4	102,12 ms	-
Module Queue	1500	-	-

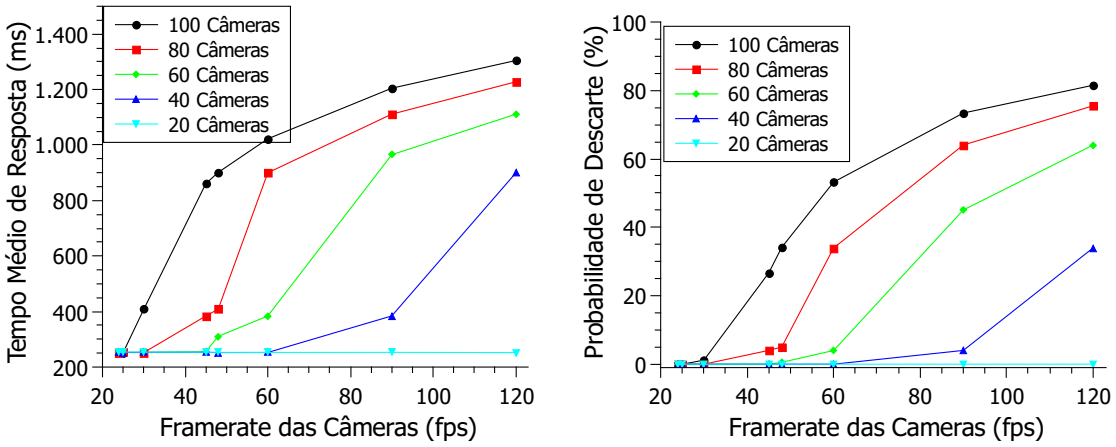
The values presented in Table 11 were obtained through a practical experiment. The experiment consisted of collecting performance data from the real system that was modeled in this article. The system was monitored for 10 minutes at 5 random times of the day. The system calculated each parameter based on the total number of requests processed during the monitoring time. Each module ran on a different instance of type *t2.small* from Amazon EC2[2], with 1 VCPU, 2 GB of RAM. The following subsections present three case studies that show the potential of the model and serve as a practical guide for future uses by system administrators.

7.2.1 Case Study 1 - Number of Cameras

This section presents the results of varying camera settings for system response time. The configurations varied by changing the number of cameras used and their framerate. The number of cameras varied between twelve and one hundred, and this variation is possible by dividing the time between arrivals and the number of desired cameras and inserting the resulting value in the `AR` transition. In this case study, the system discard result proved relevant to understanding the results. Therefore, a discard chart was also included in this section. The number of machines in each layer was adjusted to 80, this was done by multiplying the processing capacity (C1, C2 and C3) of each layer by 80. Figure 26 shows the result of varying cameras for the MRT and system discard. For this

[2] https://aws.amazon.com/pt/ec2/

experiment, empirically, framerates were chosen between 24 and 120 frames per second (fps).



(a) Mean response time.

(b) System's probability of discard.

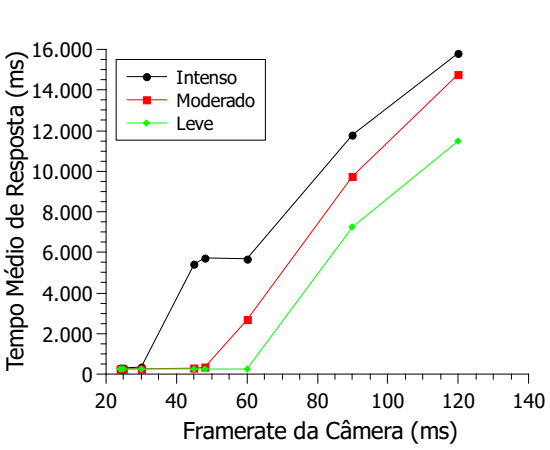
Figure 26 – Variation of the number of cameras.

Figure 26a shows the result of varying cameras for the system's MRT. The result shows a clear increase in MRT along with the increase in the number of cameras and framerate. Starting with twenty cameras, the system can respond ably to cameras at approximately 120-fps. With forty cameras, the system presents reasonable times for cameras of up to 90-fps; and presents a moderate time (eighty milliseconds) for 120-fps cameras. The results for sixty cameras show that the system maintains reasonable times up to a framerate of 60-fps. For eighty cameras, the system has shorter times, up to four hundred milliseconds, when using cameras at up to 48-fps. Finally, with one hundred cameras, the system presented the lowest values, around two hundred milliseconds, for 24 and 25-fps cameras, which are more common in this type of application, but it also showed an acceptable result of around four hundred milliseconds with 30-fps cameras.

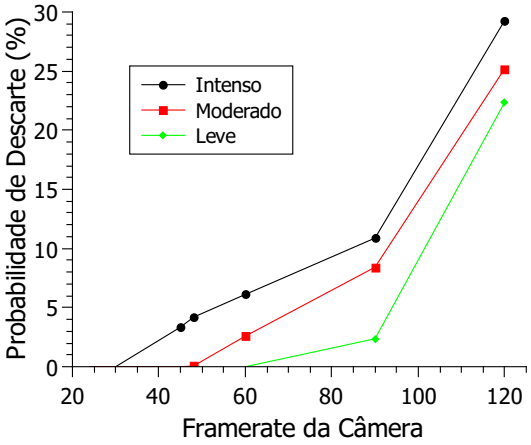
Figure 26b shows the probability of discard of the system as the cameras vary. The general results show that the system is stressed to the point of discarding 80% of requests with a framerate of 120-fps, except for the twenty camera scenario, which did not present any discard at any time. With forty cameras, the system can maintain a low probability of discard up to around 90-fps framerate. For the scenario with sixty cameras, the system only starts discarding packages with cameras from 60-fps onwards, intensifying from 90-fps onwards. The scenario with eighty cameras shows a low discard tendency with cameras up to 48-fps and intensifies after 60-fps. Finally, the scenario with one hundred cameras presents low discard with cameras up to 30-fps, and begins to intensify after 45-fps. From what can be seen, when the system starts to show even minimal discards, response times increase proportionally, so much so that the two graphs are very similar visually.

7.2.2 Case Study 2 - Traffic Intensity

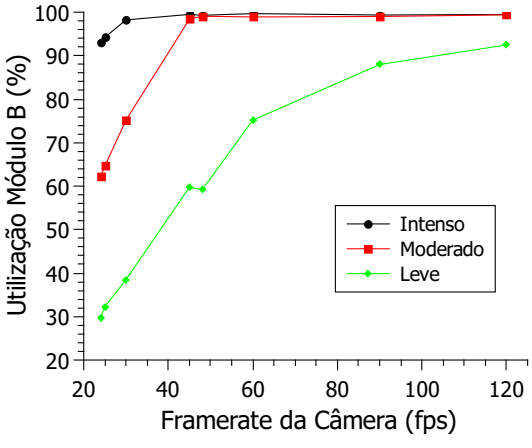
This section presents the results of the variation in traffic intensity and its impact on the system's response time. For this experiment, it was clear that the results of the system's discard and utilization of module B were relevant to understanding the results of the MRT. Therefore, graphs for the two metrics were also included in this section. Furthermore, a graph showing the cumulative distribution function (CDF) was also included because it is relevant to the proposed scenario. For this experiment, the capacity of module A was set at 12, and the capacity of module B was set at 6. Figure 27 shows the result of the variation in traffic intensity for the mean response time, system's discard, utilization of module B, and CDF. For this experiment, empirically, framerates were chosen between 24 and 120-fps. The number of cameras was set to 1 for this simulation. The number of cars in frames was set to 1, 2, and 3 for low, moderate, and intense traffic, respectively. As the purpose of this experiment is to observe the impact of traffic on metrics, low values for the number of cameras were used and only one machine was considered in each layer.



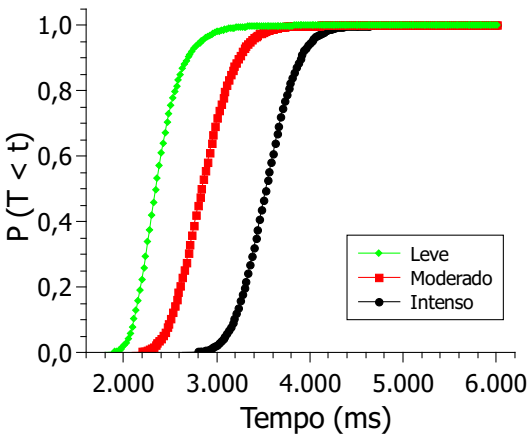
(a) Mean response time.



(b) System's probability of discard.



(c) Module B Utilization.



(d) CDF.

Figure 27 – Impact of variation in traffic intensity on system performance.

Figure 27a shows the impact of traffic intensity on the MRT system. The results show a noticeable increase in MRT as traffic intensifies. In low traffic, the MRT can maintain around 300 milliseconds with a camera at up to 60-fps and increases to eleven seconds with a camera at 120-fps. In moderate traffic, a camera with up to 48-fps can maintain the MRT in the range of 330 milliseconds. From 90-fps, the MRT exceeds ten seconds and reaches almost 15 seconds with 120-fps. Finally, a camera with up to 30-fps can maintain the MRT at around 350 milliseconds in intense traffic. The time varies between five seconds at 40-fps and around sixteen seconds at 120-fps due to the high utilization of module B and the high probability of discard. In general, this result can be used for planning camera locations. For example, a camera with up to 60-fps can be placed on a lightly traveled road, but it is only recommended to use cameras with up to 30-fps in areas moved so as not to overload the system.

Figure 27b presents the result of the traffic variation for the system's discard probability. The discard chart serves as a complement to the MRT chart. As traffic intensifies, it can be seen that system discard increases in all scenarios. In the low traffic scenario, the discard probability increases from 60-fps of framerate when the MRT grows in the previously mentioned graph. Likewise, the discard becomes noticeable with framerates from 48-fps for the moderate traffic scenario. Finally, the behavior is repeated with intense traffic, with an increase in discards from 30-fps onwards. The module B utilization chart also helps to understand the results better.

Figure 27c shows the impact of changing traffic intensity on the utilization of module B. Module B is important for this experiment, as a frame is divided into N subframes, so it receives a greater load. With low traffic, the system's MRT increases as module B exceeds 70% utilization and consequently begins to discard requests. With moderate traffic, response time increases from 48-fps, and even though the discard is very low at this point, module B begins to become stressed. Likewise, in intense traffic, the drop is low after 30-fps, but as the module is stressed in this situation, the MRT greatly increases. This set of graphs is important because it helps you plan the location of the cameras and understand the system's current workload.

Finally, Figure 27d presents the results for the CDF. The results were obtained through some adjustments to the model to make it absorbent. Therefore, the experiment was carried out to verify the mean time the system takes to process 100 frames using a 60-fps camera, equivalent to 1600 milliseconds of video. The results show the probability that all frames have been processed due to time. The result with low traffic shows values above 90% from 2700 milliseconds onwards and reaches 100% from 3850 milliseconds onwards. The result for moderate traffic reaches 90% probability from 3250 milliseconds and shows 100% probability from 4130 milliseconds. Finally, the result for intense traffic showed 90% and 100% probability from 3920 and 4680 milliseconds respectively.

7.2.3 Case Study 3 - Number of Cameras X Module A Capacity

This section presents the result of the variation in traffic intensity, along with the number of cameras for the system response time. For this experiment, the capacity of module B was set at 12. Figure 28 shows the result of the variation in module A capacity and the number of cameras for the system's mean response time. The choice of the two parameters is based on the DoE result, where the most impactful combination of factors was used for this case study. For this experiment, the framerate of the cameras was set at 25-fps. The number of cameras for this experiment was set to 1, 2, 3, 4, and 5. The module A capacity was set to 8, 10, 12, 14 and 16. This case study aims to study the impact of the number of cameras on the capacity of module A; Therefore, a low number of cameras was considered to find out how much each machine can support.

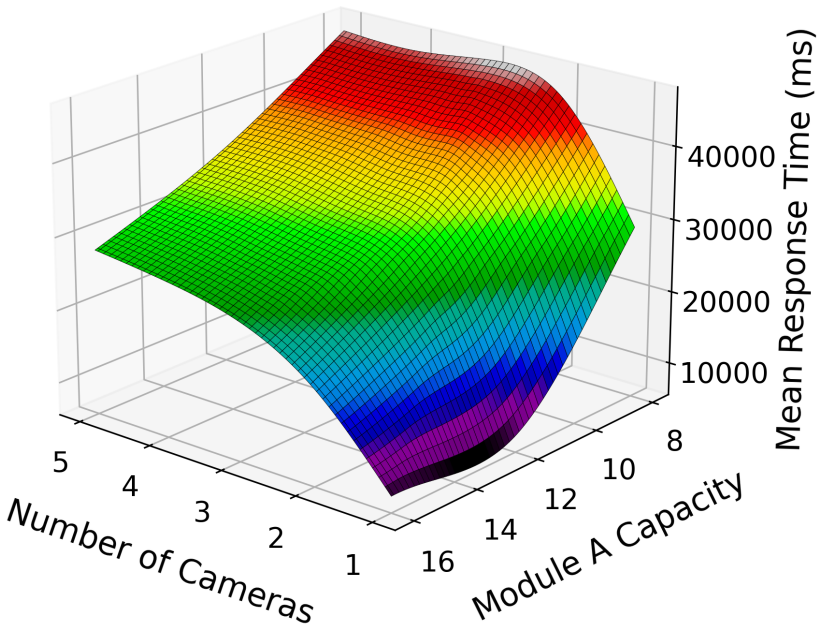


Figure 28 – Simultaneous variation in the number of cameras and module A capacity.

The graph shows a growth trend as cameras are added to the system, and the same trend is observed when module A's capacity is reduced. With only one camera, the MRT remains for around 7 seconds and is maintained with up to 12 capacity cores. When the capacity is reduced to 8, the time is close to 30 seconds. The scenario with two cameras presents around 18 seconds of MRT, and as the capacity of module A is reduced, the MRT reaches 44 seconds with capacity 8. With three cameras, the MRT varies between 24 and 47 seconds, with capacities of 16 and 8, respectively. The scenarios with four and five cameras behave very similarly. With a capacity of 16, both present an MRT of 26 and

28 seconds, respectively. Finally, both scenarios stagnate similarly to the three-camera scenario, with an average of 46 seconds.

The graph displays useful information for the evaluator that allows the system to understand behavior in a given circumstance. For example, in this case study, the graph shows that when module A has 8 capacity, the MRT no longer grows when 3 or more cameras are used. The same occurs for the scenario in which module A has a capacity of 16 when using more than one camera, which increases MRT by around 2 seconds per additional camera. Finally, the result helps to understand why these two components were the most impactful. As they are the system's entry points, if the capacity is very low, the component may not pass enough requests to the following modules, becoming a point bottleneck.

8 Model Validation

Neste capítulo serão apresentados os experimentos realizados para validar os modelos propostos nesta dissertação.

8.1 Availability Validation

The results obtained from the proposed models are expected to be similar to those computed in the entire system. In this context, validation techniques were applied to validate the model's three main aspects: assumptions, values, and distributions of input parameters and output values (JAIN, 1991). Dependability models can be validated by comparing the results of measurements collected from a real environment with those generated by the model (LAVENBERG, 1983). However, obtaining data from a real system is challenging since failures are unpredictable and can take a long time to observe (SCHROEDER; GIBSON, 2007). In any case, it is possible to inject faults into the system and observe its behavior. This technique is commonly used to verify the system's reliability.

Fault injection techniques can be divided into hardware-implemented fault injection, software-implemented fault injection, and simulated fault injection (WANG et al., 2019). Hardware-implemented fault injection can cause irreparable damage to system hardware. The fault injection implemented via simulation is considered complex and difficult to operate. Software-implemented fault injection does not require expensive additional equipment and does not cause damage to system hardware. This study uses software fault injection to validate the proposed reactive maintenance model, applying the same input conditions and calculating the respective confidence intervals.

Figure 29, based on (ROMERO, 2023), presents the steps to define the technique for calculating the Confidence Interval. The first step corresponds to analyzing whether the sample follows a normal distribution. If this is the case and the standard deviation of the sample is known, the analytic expression for the normal distribution can be used. Otherwise, the t-student distribution must be used. Assuming that the sample is not normally distributed and the distribution of the sample is known, the appropriate confidence interval analytic formula should be used. Otherwise, the semi-parametric bootstrap is used. If the sample does not follow a normal distribution and the proper distribution is unknown, one can use the n-parametric bootstrap or the central limit theorem.

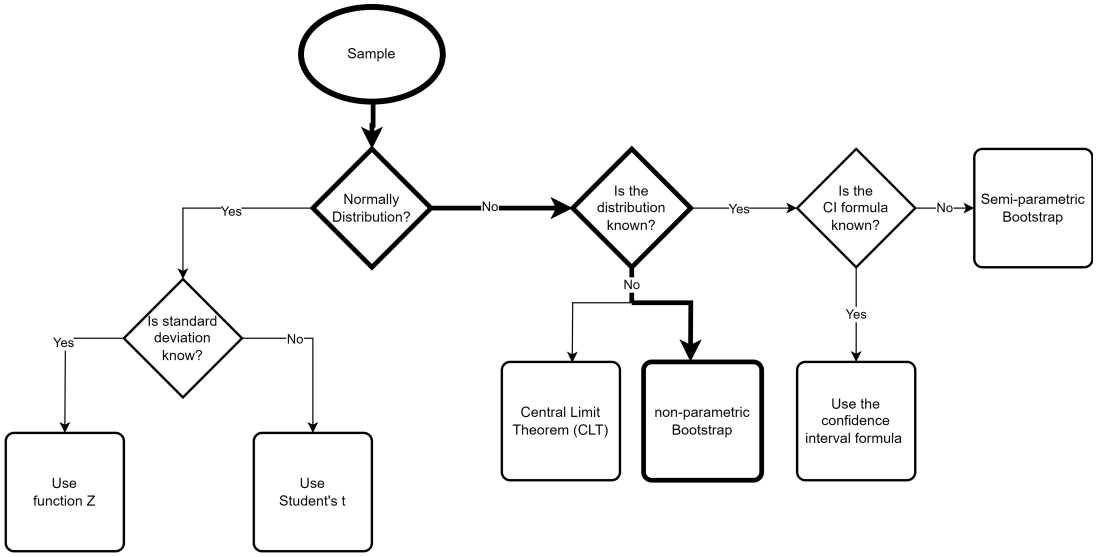


Figure 29 – Statistical inference

8.1.1 Validation Setup

This study aims to demonstrate the procedures used to validate the availability obtained from the proposed SPN model (Reactive Maintenance). A base environment was considered to carry out the experiments, consisting of a camera and a server.The mean failure (MTTF) and repair times (MTTR) used were obtained from (PEREIRA et al., 2022) and (BRITO et al., 2020). We assume that the mean failure and repair times are exponentially distributed, and thus, the failure and repair rates are the inverse of MTTF and MTTR. When considering the aging of the application that occurred on the server, the failure time adopted for this component is applied according to the Erlang distribution. Table 12 shows the adopted values for the MTTFs and MTTRs in hours.

Table 12 – MTTF and MTTR

	MTTF(h)	MTTR(h)
Camera	8760	1.67
Server	4765	10

We have stressed the system by accelerating the MTTF of the components using the value 1,000 as an acceleration factor. A fault injector was also developed, responsible for inserting faults in each component, listing the components that will fail, and monitoring these devices. The fault injector creates a thread for each of the components in the system, which are responsible for injecting the faults. This injector was run on a microcomputer with an AMD CPU at 3.2 GHz, 8 GB of RAM, and 500 GB of storage.

8.1.2 Validation Results

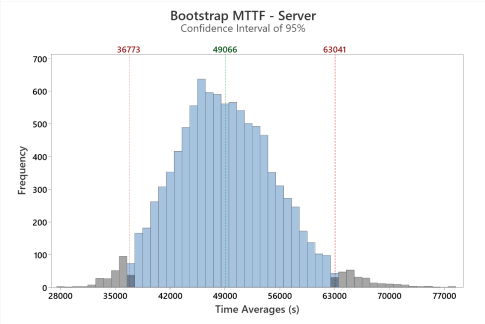
Table 13 shows the values in hours used in the fault injector. In this way, the threads generate a random number following the time distribution adopted for each component and wait to inject a fault. Another thread monitors whether the system is available or unavailable. If the system is active, the injector saves the status as "UP" in a text file; if the system is inactive, it saves the status as "DOWN". This system status check monitoring is done every 10 seconds. Two experiments were performed, one for the server and the other for the camera. Each experiment carried out lasted 166 hours, and a document was obtained with the periods in which the components were working ("UPs") or not responding ("DOWNs"). Therefore, it was possible to calculate the failure (MTTF) and repair times (MTTR). For example, if before a "DOWN" there are 70 "UPs". The activity time was 700 seconds, 70x10s (interval of obtaining the samples). Therefore, by transforming the "UPs" and "DOWNs" into MTTFs and MTTRs, one can calculate the actual availability of the system and also compute the 95% confidence interval.

In the collected data sample, it was impossible to determine the theoretical distribution, so statistical inferences were adopted, as shown in Figure 29. The technique used was the n-parametric bootstrap to calculate the confidence interval and generate the n resampling of size m (ROMERO, 2023). These resamples are obtained through random selections with replacement of the original sample. So we have n values of the statistics of interest, each coming from a resampling. These values are then sorted in ascending order $\widehat{\theta}_1, \widehat{\theta}_2, ..., \widehat{\theta}_i, ..., \widehat{\theta}_{n-1}, \widehat{\theta}_n$, where $\widehat{\theta}_1$ is the smallest value and $\widehat{\theta}_n$ is the largest value. After that, the value $\widehat{\theta}_{\alpha/2\times 100\%}$ at the $\alpha/2 \times 100\%$ percentile corresponds to the lower bound, and the value $\widehat{\theta}_{1-\alpha/2\times 100\%}$ at the $(1-\alpha/2) \times 100\%$ percentile matches the upper bound (ROMERO, 2023).

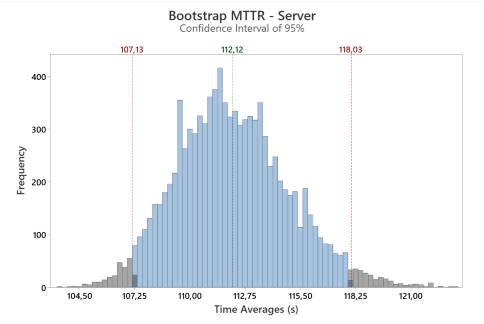
About 10,000 new samples of "UPs" and "DOWNs" blocks were generated using the bootstrap technique. The average was calculated for each resampling and these averages were ordered in ascending order, where the lowest 250th and the highest 250th of these 10,000 samples are the "percentiles" for the 95% confidence interval. Figures 30a and 30b present the confidence intervals for the MTTF and MTTR of the server, respectively. Figures 31a and 31b present the confidence intervals referring to the camera. Availability was computed from Equation (8.1). Notably, the correct interpretation of 95% confidence depends on understanding the probability frequency. This means that if our experiment is run repeatedly, the server and camera availability will be 95% of the time in these calculated confidence intervals in the long run.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{8.1}$$

Table 14 compares the results obtained from the model and those quantified in

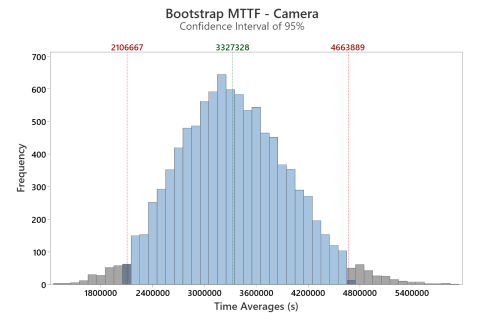


(a) Bootstrap MTTF - Server.

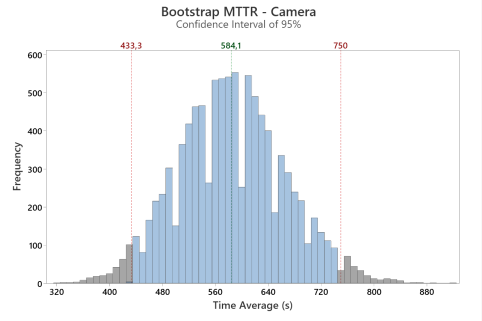


(b) Bootstrap MTTR - Server.

Figure 30 – Bootstrap - Server.



(a) Bootstrap MTTF - Camera.



(b) Bootstrap MTTR - Camera.

Figure 31 – Bootstrap - Camera.

Table 13 – MTTF and MTTR used in fault injector

	MTTF(h)	MTTR(h)
Camera	8.76	1.67
Server	4.76	10

the whole system. The respective confidence intervals were also computed for the results obtained in the whole system. The values obtained from the model are within the confidence interval calculated from the measurements performed on the entire system. Thus, the proposed models were validated. So, our model reliably represents the analyzed components (camera and server).

Table 14 – Results for Availability. CI: Confidence Interval.

	Model	Real System [CI]
Camera	0.999799889	0.999824485 [0.999794362 ; 0.999839216]
Server	0.997696751	0.997720124 [0.997095184 ; 0.998131225]

8.2 Performance Validation

An SPN model represents the simplification of a real-world concept or problem. These models allow the simulation of various situations related to the problem under analysis, as well as the estimation of expected behavior through metrics. The data obtained are used as a basis for decision-making, whether regarding the development of a new solution or the adaptation of the current solution. Response time and cost of implementing the solution are factors that influence the decision. Some solutions may use cloud services, which can be expensive but allow adjustments to reduce costs. However, other solutions require the acquisition and implementation of physical infrastructure.

In large-scale projects, the cost of implementation can be prohibitive. Even with data acquired from simulations, there is a need to validate the information obtained to mitigate the risks and costs involved. Implementing a solution on a small scale involves lower costs, while allowing comparison with the metrics obtained and validating the model. Building a simplified model is also another possibility, as it allows validation with few software and hardware resources. Thus, validation consists of creating an experiment to determine whether an SPN model minimally represents the expected behavior of a real or simulated system.

Figure 32 presents the flowchart that describes the strategy used in this validation to ensure that the experiment is correctly validated:

- Setting up the experiment according to the original model: This step consists of configuring the experiment according to the model's base architecture. The experiment will reflect information from the model, such as queue sizes, processing capacities, service times, among others. This step also involves configuring computers and virtual machines, and the model will eventually become more complex, which will make it impossible to run the experiment. Therefore, it is necessary to simplify the proposed model.

- Simplifying the model: If it is found that it is not possible to generate the experiment from the original model, a new model is derived that encapsulates some characteristics, but maintains the essence of the architecture. In this way, the number of replicable components presented in the scenario and in the original model is reduced. Due to the restriction of computational resources, simplifying the model will allow the experiment to reflect components, quantity and processing parameters. From a statistical point of view, validating a part of the model will also validate the whole.

- Clock Synchronization: Physical or virtual machines need to be synchronized to minimize errors in MRT calculation. Virtual machines can be synchronized using a Network Time Protocol (NTP) tool (MILLS, 1991). Virtual machines hosted on

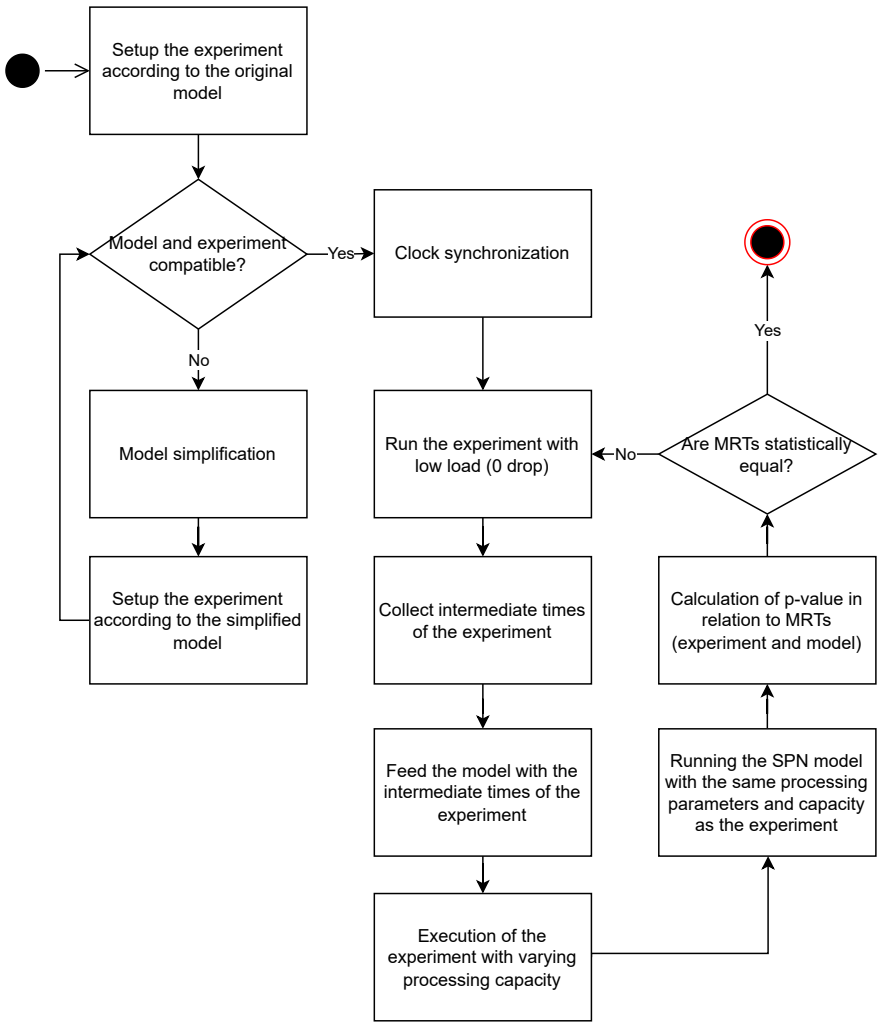


Figure 32 – Validation Flow

cloud services use the synchronization mechanisms provided by the services. Amazon Web Services (AWS) provides the local Amazon Time Synchronization Service [1], which uses NTP.

- Running the experiment with low load (0 drops): This step evaluates whether communication between components occurs as expected. Requests are sent with a low arrival rate, as this minimizes queue time, as well as data or event loss (0 drops).

- Collection of intermediate times of the experiment: The execution of the experiment collects the External Intermediate Times (EIT) between the components. An intermediate time is the difference between the time the request leaves one component and arrives at another component. The Internal Intermediate Times (IIT) of the component correspond to the internal processes prior to the actual processing.

[1] https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/configure-ec2-ntp.html

- Feeding the SPN model with the intermediate times of the experiment: The EITs and IITs obtained will feed the model, whether the original or the simplified one. The IITs are part of the system response time, even if they are low values that should be discarded in the context of the model. High IIT values may indicate a bottleneck in the model or in the experiment. In this case, it is important to reevaluate the "Model simplification"step.

- Execution of the experiment with varying processing capacity: Each component is configured with up to four parallel processing instances, according to the SPN model. In the experiment, the target component will have a variation of 1 and 4 processing instances. In this step, the experiment is executed with high load, where the MRT and Standard Deviation are collected. The experiment performs four sending cycles, so that in each cycle the processing capacity is increased in the target component.

- Execution of the SPN model with the same processing parameters and capacity of the experiment: The simplified model is configured to be executed under the same load conditions (arrival rate). In this step, the MRT and Standard Deviation corresponding to each variation in processing capacity are also collected.

- Calculation of the p-value in relation to the MRTs (experiment and model): From the data obtained from the simulation and the experiment, the p-value is calculated between the MRTs in each scenario, so that this statistical confidence is greater than/equal to 0.05.

- Statistical verification of MRTs: If it is verified that the MRTs are statistically equal, the process is successfully completed. Otherwise, the process returns to the stage of collecting the intermediate times of the experiment to make new adjustments to the model.

8.2.1 Model

Figure 33 presents the simplified SPN model, which was derived from the base model in Figure 23. Unlike the base model, the simplified model unifies the two frame processing services of the second component to resemble a system with processing in the cloud. A deterministic transition was used in the admission, since the validation process is the product of the comparison with absolute numbers defined in the experiment, that is, it does not take into account the statistical distribution of values.

According to the flowchart in Figure 32, the experiment was performed with the aim of finding significant variations in the MRT. The calibration of the experiment involves adjustments to several parameters such as queue size, number of processing elements and processing times in each component. It is worth noting that the setup of the experiment

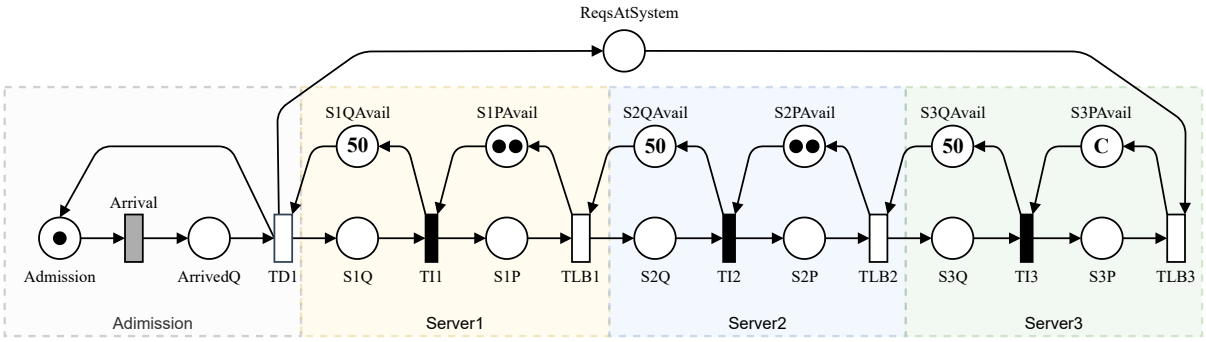


Figure 33 – SPN Model of Validation

involves the prior structuring of physical and virtual equipment, communication network and any other necessary resources. After this calibration, the collected times were fed into the simplified model. The model simulations collect the MRT values. The data collected from the experiment and simulation are subjected to statistical verification. The flow was repeated until the values in Table 15 were defined.

Table 15 – Main timed transitions used in the SPN model in Figure 33

Element	Description	Value	Server Type
Arrival	Time interval between arrival of requests	300 ms	Single Server
TD1	Time associated with transmitting the admission to Server 1	1 ms	Single Server
TLB1	Time associated with processing a request on Server1	100 ms	Single Server
TLB2	Time associated with processing a request on Server2	100 ms	Single Server
TLB3	Time associated with processing a request on Server3	3000 ms	Single Server

8.2.2 Experiment Environment

The experiment environment was set up on the AWS cloud platform [2]. The environment consists of four Amazon Elastic Compute Cloud (EC2) [3] virtual machines, of type t2.xlarge, which have 4 vcpu. The EC2 instances use the Ubuntu operating system, where the other characteristics of an instance are listed in Table 16. Figure 34 shows the architecture of the experiment that used an Amazon Virtual Private Cloud (VPC) [4] environment, encompassing all EC2 instances. Using a single environment makes it easier to manage resources, reduces latency, and minimizes the risk of incompatibility. An EC2 instance hosts the Source.jar application, which is responsible for transmitting requests. The other instances each host an application and a LoadBalancer.jar, where the first receives the request that follows in a chain to the last LoadBalancer. After the last node in the chain processes the request, the response is returned directly to the Source, which calculates the MRT metric.

[2] https://aws.amazon.com/pt/what-is-aws/
[3] https://aws.amazon.com/pt/ec2/
[4] https://aws.amazon.com/pt/vpc/

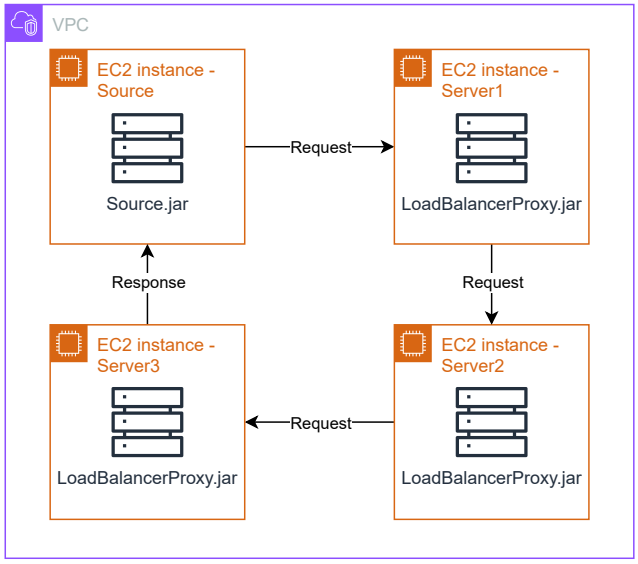


Figure 34 – AWS Environment

Table 16 – Experiment setup in AWS environment

Element	Description
Operating System	Ubuntu 24.04.1 LTS
Memory	15Gi
CPU	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz

8.2.3 Validation Tool

The validation experiment used the PASID Validator software. The software project was developed using Java language version 8, but has already been updated to Java version 11. This tool was designed in the PASID laboratory, where its objective is to certify the results from SPN models. The PASID Validator allows you to configure execution and network communication parameters, according to the SPN model under evaluation.

The PASID Validator consists of two executable artifacts, Source.jar and LoadBalancerProxy.jar, which are placed in the experiment's network environment. The Source.jar is responsible for sending requests, receiving the response and calculating the metrics at the end of the experiment. The LoadBalancerProxy receives the requests, queues them and distributes them among its ServiceProxy. Each ServiceProxy is responsible for processing a request and forwarding it to the next LoadBalancer. The last LoadBalancer in the chain processes the request (via ServiceProxy) and returns the response to the Source. The artifacts (Source and LoadBalancerProxy) communicate via TCP/IP sockets. The addresses and ports of each artifact are configured during the experiment setup. Each artifact knows the address and port of the next element in the chain. The use of sockets allows direct and continuous communication between the elements of the experiment,

ensuring reliable communication.

The Source sends requests at a time interval defined during setup. Requests are sent to the first LoadBalancerProxy, which distributes them to the chosen ServiceProxy through a circular queue algorithm. When the admission queue is full, incoming requests are discarded. The ServiceProxy performs a pseudo-processing of the request, which is done by means of a pause by calling the Thread.sleep() command. The waiting time is given by the average service time, which is specified during configuration. When it finishes, the ServiceProxy forwards them to the next element in the network. The other LoadBalancerProxy work similarly, except for the last one, which returns the response to the Source. This calculates the response time of the request, which will be used to calculate the MRT of the experiment. The timestamp of departure from one machine and arrival at the next machine is concatenated to the body of the request. This strategy allows you to understand the route of the request and identify possible bottlenecks.

8.2.4 Validation Scenario

In this scenario, the LoadBalancerProxy were configured according to Table 17. It should be noted that the term ServiceProx is an abstraction to represent a processing element in the capacity of the LoadBalancerProxy. According to the context under study, this element can mean a core, thread, container, among other elements. Likewise, the LoadBalancerProxy can represent a server, a balancer, a container orchestrator, among others. The queue discard strategy is drop when reaching the established limit for the queue size, which is 50 in each load balancer. The queue strategy used in the load balancers is First Come, First Served (FCFS). In each configuration, the number of services in the last load balancer is incremented by 1. In each configuration, 500 requests are sent, where the response time of each message will be used to calculate the MRT and the standard deviation of the configuration. The SPN model was simulated with an error rate of 2%, where the MRT and standard deviation were collected for statistical analysis.

Table 17 – Parameters for validation

Configuration	LoadBalancer1	LoadBalancer2	LoadBalancer3
1	2 Services	2 Services	1 Service
2	2 Services	2 Services	2 Services
3	2 Services	2 Services	3 Services
4	2 Services	2 Services	4 Services

8.2.5 Results

The graph in Figure 35 shows the comparison between the MRTs of the experiment and the SPN model. In each configuration, the values obtained in the experiment are

similar to those obtained in the model, following a decreasing trend as the number of services increases. It can be observed that the values of the experiment present greater variation in relation to the model, which is evidenced by the wide range of the standard deviation. This range of variation also decreases with the increase of services.

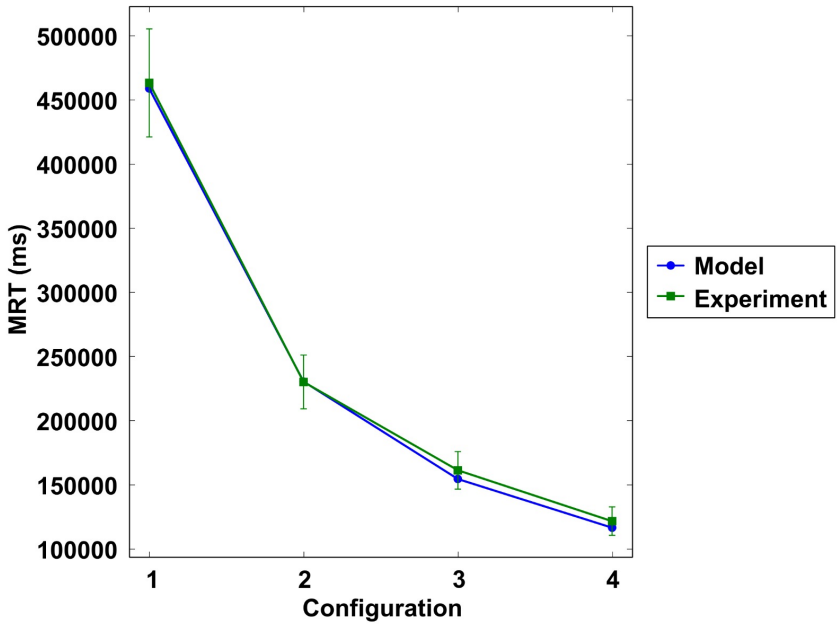


Figure 35 – Performance Validation Result

We compared the MRTs of the model and experiment by applying the one-sample T-Test. The T-Test is a type of statistical test used to compare the means of two groups; it can be used when the samples satisfy the conditions of normality, equal variance, and independence (KIM, 2015). The statistical results are listed in Table 18. In each experiment configuration, 500 requests were sent, resulting in the calculation of the MRT and standard deviation. The model resulted in a single value of the metric, as well as its standard deviation. All samples presented a normal distribution.

Table 18 – Result Test-T

Conf	MRT Model	SD Model	MRT Experim.	SD Experim.	t Value	p Value
1	459102.48	1521.20	463489.53	42135.41	-0.47	0.67
2	230696.34	748.49	230384.88	20944.08	0.07	0.95
3	154770.21	502.23	161477.93	14679.81	-2.04	0.11
4	116795.30	382.33	121858.38	11078.03	-2.04	0.11

The p-value is greater than 0.05 in all analyzed configurations. In these cases, we consider that there is no significant difference between the two sample groups and, therefore, we cannot refute the null hypothesis of equality in the analyzed cases, with 95% confidence. Considering the model as a reference, the analysis suggests that the experiment is capable of reproducing the reference values with reasonable accuracy, but

presents challenges in terms of variability. The main focus for improvements should be on reducing experimental variability and calibrating to smaller values.

Finally, this validation concludes that the results generated by the model are statistically equivalent to the experiment. The simplified model reflects the real environment, which also validates the base model. The result ratifies the use of the SPN model to plan and evaluate the architecture proposed in this work.

9 Conclusion

This dissertation designed models of stochastic Petri nets for a video monitoring architecture using computational resources at the network's edge. The models are intended to help system administrators plan the architecture before deployment. The architecture consists of one or more cameras that collect video data and send it to a set of microservices that process the visual data. The models consider several factors that influence the final availability and performance of the system. Availability, performance and reliability metrics were used to analyze the models. In addition, the paper also investigated context-specific metrics such as missed call probability and staff call rate. This paper also investigated the impact of maintenance routines on the chosen metrics. Through the use of Stochastic Petri Nets (SPN) models, we evaluated and compared reactive, autonomous, and preventive maintenance approaches, providing a comprehensive dependability analysis that underscores the pivotal role of maintenance in enhancing system reliability and efficiency. By conducting a detailed sensitivity analysis, we identified critical components that most significantly impact system availability, enabling us to formulate targeted maintenance strategies and optimize resource allocation.

Performance results show that various factors, such as the number of cameras, framerate, and traffic intensity, can significantly impact system performance. The results also show that, in general, it is recommended to use cameras up to 25-fps, as they have a lower response time, considering the subsequent framerates. The more intense the traffic on the road, the lower the camera's framerate must be to avoid overloading the system. Finally, the third case study highlighted the importance of the first module for the system's MRT. The first module (A) represents the entry point, and when it lacks processing resources, it tends to become a bottleneck for the system, as it is the entry point for requests. The following modules also need to be adjusted, but as they have multiple discard points, they are more difficult to become bottlenecks. The general results of the experiments suggest that if it is not possible to change the framerate of the cameras, it is interesting to reduce the number of them, which will vary depending on each case.

Availability results show that preventive maintenance, particularly when incorporating scheduled rejuvenation techniques, leads to substantial improvements in system availability compared to reactive and autonomous approaches. System reliability results show that camera reliability is one of the key factors for overall system reliability. The maintenance models demonstrated the impact that some routines could have on the final availability of the system. The case studies understood and addressed most aspects of the presented models.

In future work, we intend to add some new mechanisms to the model to consider some optimization aspects of the system architecture, such as the immediate discarding of static frames in moments of slow traffic, such as near traffic lights. The system currently receives all the frames sent by the cameras and processes them in an isonomic manner, so it is interesting that the system is optimized so as not to process frames that all vehicles have already processed in previous frames, as happens in times of traffic congestion. This way, the system would consume less resources which subsequently results in cost reduction. Furthermore, the study also plans to include modifications to the model to be able to represent the monitoring of multiple parts of a city using local networks in different monitoring sections. The model represents a single system that monitors large areas, and therefore can easily be overwhelmed. Therefore, it is interesting to study possible ways of distributing it in different areas in local networks that monitor different car lanes and how this can be modeled.

References

ALSABAAN, M. et al. A distributed surveillance system with full coverage guarantee using positive orthogonal codes. *IEEE Access*, IEEE, v. 9, p. 16837–16848, 2021. 17, 20

ALSOLAI, H.; ROPER, M. A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, Elsevier, v. 119, p. 106214, 2020. 15

ANTONY, J. *Design of experiments for engineers and scientists*. [S.l.]: Elsevier, 2014. 10

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. 12

BABIYOLA, A. et al. A hybrid learning frame work for recognition abnormal events intended from surveillance videos. *Journal of Intelligent & Fuzzy Systems*, IOS Press, n. Preprint, p. 1–14, 2023. 3

BOSCH, J.; BENGTSSON, P. Assessing optimal software architecture maintainability. In: IEEE. *Proceedings Fifth European Conference on Software Maintenance and Reengineering*. [S.l.], 2001. p. 168–175. 4

BRITO, C. et al. Dependability models for computer-assisted surveillance systems planning: A strategy based on multiple maintenance alternatives. 2023. 4

BRITO, C. et al. Avaliação de disponibilidade de uma arquitetura de computação de borda com redes de petri estocásticas. In: *Anais do XIX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. Porto Alegre, RS, Brasil: SBC, 2020. p. 175–180. ISSN 2595-6167. Disponível em: <https://sol.sbc.org.br/index.php/wperformance/article/view/11117>. 66

BRITO, C. et al. Offloading data through unmanned aerial vehicles: a dependability evaluation. *Electronics*, MDPI, v. 10, n. 16, p. 1916, 2021. 9

CAMPOLONGO, F.; TARANTOLA, S.; SALTELLI, A. Tackling quantitatively large dimensionality problems. *Computer Physics Communication*, Institute Jbr Systems, Informatics and Safety. Joint Research Centre of the European Commission, v. 117, n. 1, p. 75–85, 1999. 10

CAMPOLONGO, F. et al. *Sensitivity analysis in practice: a guide to assessing scientific models*. [S.l.]: John Wiley and Sons, 2004. 10

CARDELLINI, V. et al. *Performance and dependability in service computing: concepts, techniques and research directions*. [S.l.]: Information Science Reference-Imprint of: IGI Publishing, 2011. 13

CHEN, L.; HA, W. Reliability prediction and qos selection for web service composition. *International Journal of Computational Science and Engineering*, Inderscience Publishers (IEL), v. 16, n. 2, p. 202–211, 2018. 9

CHEONG, K. et al. Practical automated video analytics for crowd monitoring and counting. *IEEE Access*, IEEE, 2019. 3

CHEONG, K. H. et al. Practical automated video analytics for crowd monitoring and counting. *IEEE Access*, IEEE, v. 7, p. 183252–183261, 2019. 4

COLLINS, C. et al. Are you ready for 5g? McKinsey & Company, 2018. 25

COSTA, I. et al. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, Wiley Online Library, v. 32, n. 7, p. 2191–2205, 2016. 10

DING, S.-z.; CHEN, X.-m.; YU, L. Markov chain-based platoon recognition model in mixed traffic with human-driven and connected and autonomous vehicles. *Journal of Central South University*, Springer, v. 29, n. 5, p. 1521–1536, 2022. 17, 21

ELHARROUSS, O.; ALMAADEED, N.; AL-MAADEED, S. A review of video surveillance systems. *Journal of Visual Communication and Image Representation*, Elsevier, v. 77, p. 103116, 2021. 3

FEITOSA, L. et al. Performance evaluation of message routing strategies in the internet of robotic things using the d/m/c/k/fcfs queuing network. *Electronics*, MDPI, v. 10, n. 21, p. 2626, 2021. 10

GERMAN, R. *Performance analysis of communication systems - modelling with non-Markovian stochastic Petri nets*. [S.l.]: Wiley, 2000. (Wiley-Interscience series in systems and optimization). ISBN 978-0-471-49258-0. 9, 10

GONÇALVES, I. et al. Surveillance system in smart cities: a dependability evaluation based on stochastic models. *Electronics*, MDPI, v. 10, n. 8, p. 876, 2021. 3, 4, 18, 21, 40

HOFFMAN, F.; GARDNER, R. *Evaluation of Uncertainties in Environmental Radiological Assessment Models*. [S.l.]: Radiological Assessments, 1983. 12

HUANG, Y. et al. Software rejuvenation: Analysis, module and applications. In: IEEE. *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*. [S.l.], 1995. p. 381–390. 4

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley & Sons, 1990. 53

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling, NY: Wiley*. [S.l.: s.n.], 1991. ISBN 0471503361. 65

JIANG, M. et al. Federated dynamic graph neural networks with secure aggregation for video-based distributed surveillance. *ACM Transactions on Intelligent Systems and Technology (TIST)*, ACM New York, NY, v. 13, n. 4, p. 1–23, 2022. 4

JOUBARI, O. E.; OTHMAN, J. B.; VÈQUE, V. A stochastic mobility model for traffic forecasting in urban environments. *Journal of Parallel and Distributed Computing*, Elsevier, v. 165, p. 142–155, 2022. 17, 21

JúNIOR, R. de S. M. *Identification of Availability and Performance Bottlenecks in Cloud Computing Systems: An Approach Based On Hierarchical Models and Sensitivity Analysis*. Tese (Doutorado) — Federal University of Pernambuco, Center for Informatics, Graduate in Computer Science, Recife, 2016. 12

KIM, T. K. T test as a parametric statistic. *Korean journal of anesthesiology*, The Korean Society of Anesthesiologists, v. 68, n. 6, p. 540–546, 2015. 75

KIM, Y.; JEONG, J. A simulation-based approach to evaluate the performance of automated surveillance camera systems for smart cities. *Applied Sciences*, MDPI, v. 13, n. 19, p. 10682, 2023. 17, 21

KLEIJNEN, J. P. Sensitivity analysis and optimization in simulation: design of experiments and case studies. In: IEEE. *Winter Simulation Conference Proceedings, 1995.* [S.l.], 1995. p. 133–140. 10

LAPRIE, J.-C.; AVIZIENIS, A.; KOPETZ, H. *Dependability: Basic Concepts and Terminology*. Vienna: Springer-Verlag, 1992. v. 5. (Dependable Computing and Fault-Tolerant Systems, v. 5). ISBN 978-3-7091-9172-9. Disponível em: <https://link.springer.com/book/10.1007/978-3-7091-9170-5>. 12

LAVENBERG, S. S. *Computer Performance Modeling Handbook*. USA: Academic Press, Inc., 1983. ISBN 0124387209. 65

LIN, T.; PHAM, H. Modeling security surveillance systems with state dependent inspection-maintenance strategy. *IEEE Transactions on Computational Social Systems*, IEEE, v. 10, n. 5, p. 2467–2478, 2022. 18, 20

LINS, L. et al. Stochastic modeling for assessing the reliability and availability of drone-based surveillance systems. In: IEEE. *2024 IEEE International Systems Conference (SysCon)*. [S.l.], 2024. p. 1–8. 18, 21

LÓPEZ, J. et al. Watchbot: A building maintenance and surveillance system based on autonomous robots. *Robotics and Autonomous Systems*, Elsevier, v. 61, n. 12, p. 1559–1571, 2013. 18, 20

MA, S. et al. A retrieval optimized surveillance video storage system for campus application scenarios. *Journal of Electrical and Computer Engineering*, Wiley Online Library, v. 2018, n. 1, p. 3839104, 2018. 17, 20

MACIEL, P. R. M. *Performance, reliability, and availability evaluation of computational systems, Volume 2: Reliability, availability modeling, measuring, and data analysis*. [S.l.]: CRC Press, 2023. 9

MACIEL, P. R. M. *Performance, reliability, and availability evaluation of computational systems, volume I: performance and background*. [S.l.]: CRC Press, 2023. 9

MARI; EILA. The impact of maintainability on component-based software systems. In: IEEE. *2003 Proceedings 29th Euromicro Conference*. [S.l.], 2003. p. 25–32. 15

MARSAN, M. A. et al. *Modelling with Generalized Stochastic Petri Nets*. USA: John Wiley & Sons, Inc., 1994. ISBN 0471930598. 9

MARSAN, M. A. et al. Modelling with generalized stochastic petri nets. *ACM SIGMETRICS performance evaluation review*, ACM New York, NY, USA, v. 26, n. 2, p. 2, 1998. 10

MELO, C. et al. Availability models for hyper-converged cloud computing infrastructures. In: IEEE. *2018 Annual IEEE International Systems Conference (SysCon)*. [S.l.], 2018. p. 1–7. 40

MILLS, D. L. Internet time synchronization: the network time protocol. *IEEE Transactions on communications*, Ieee, v. 39, n. 10, p. 1482–1493, 1991. 69

MOLLOY, M. K. Performance analysis using stochastic petri nets. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 31, p. 913–917, September 1982. ISSN 0018-9340. 9

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989. 9

NGUYEN, T. A. et al. Blockchain empowered federated learning with edge computing for digital twin systems in urban air mobility. In: LEE, S. et al. (Ed.). *The Proceedings of the 2021 Asia-Pacific International Symposium on Aerospace Technology (APISAT 2021), Volume 2*. Singapore: Springer Nature Singapore, 2023. p. 935–950. ISBN 978-981-19-2635-8. 3

NGUYEN, T. A. et al. Dependability and security quantification of an internet of medical things infrastructure based on cloud-fog-edge continuum for healthcare monitoring using hierarchical models. *IEEE Internet of Things Journal*, v. 8, n. 21, p. 15704–15748, 2021. 3

O'CONNOR, P.; KLEYNER, A. *Practical reliability engineering*. [S.l.]: john wiley & sons, 2011. 13

OLAYODE, I. O.; TARTIBU, L. K.; OKWU, M. O. Traffic flow prediction at signalized road intersections: a case of markov chain and artificial neural network model. In: IEEE. *2021 IEEE 12th International Conference on Mechanical and Intelligent Manufacturing Technologies (ICMIMT)*. [S.l.], 2021. p. 287–292. 18, 21

PAIKARAY, D.; GANDHI, J. A secure and reliable device access control scheme for iot based monitoring systems. In: IEEE. *2022 5th International Conference on Contemporary Computing and Informatics (IC3I)*. [S.l.], 2022. p. 665–669. 4

PEREIRA, P. et al. Availability model for edge-fog-cloud continuum: an evaluation of an end-to-end infrastructure of intelligent traffic management service. *The Journal of Supercomputing*, v. 78, 02 2022. 66

PIANOSI, F. et al. Sensitivity analysis of environmental models: A systematic review with practical workflow. *Environmental Modelling and Software*, v. 79, p. 214–232, 2016. 10

PINHEIRO, T. et al. The mercury environment: a modeling tool for performance and dependability evaluation. In: *Intelligent Environments 2021*. [S.l.]: IOS Press, 2021. p. 16–25. 40

POGADADANDA, V. et al. Abnormal activity recognition on surveillance: A review. In: IEEE. *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*. [S.l.], 2023. p. 1072–1077. 3

RODRIGUES, L. et al. Modelo estocástico para avaliaçao de disponibilidade de hospitais inteligentes. In: SBC. *Anais do XIX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. [S.l.], 2020. p. 145–156. 9

ROMERO, M. M. P. *Performance, Reliability, and Availability Evaluation of Computational Systems*. CRC Press, 2023. Volume 1. Performance and Background. ISBN 9781032295374,9781032306391,9781003306016. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=F34DD70C1001B47E7901CD9DCAE4349B>. 65, 67

SANTOS, L. et al. Data processing on edge and cloud: a performability evaluation and sensitivity analysis. *Journal of Network and Systems Management*, Springer, v. 29, n. 3, p. 27, 2021. 10

SAS, D. et al. Exploring the relation between co-changes and architectural smells. *SN Computer Science*, Springer, v. 2, p. 1–15, 2021. 14

SCHROEDER, B.; GIBSON, G. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In: . [S.l.: s.n.], 2007. p. 1–16. 65

SILVA, B. et al. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Proceedings of industrial track at 45th dependable systems and networks conference, DSN*. [S.l.: s.n.], 2015. p. 1–4. 57

SILVA, F. A. et al. Model-driven impact quantification of energy resource redundancy and server rejuvenation on the dependability of medical sensor networks in smart hospitals. *Sensors*, MDPI, v. 22, n. 4, p. 1595, 2022. 4, 36

SILVA, F. A. et al. Mobile cloud performance evaluation using stochastic models. *IEEE Transactions on Mobile Computing*, v. 17, n. 5, p. 1134–1147, 2018. 10

SINQADU, M.; SHIBESHI, Z. S. Performance evaluation of a traffic surveillance application using ifogsim. In: SPRINGER. *3rd International Conference on Wireless, Intelligent and Distributed Environment for Communication: WIDECOM 2020*. [S.l.], 2020. p. 51–64. 18, 20

STATISTA. *Video surveillance camera market size worldwide from 2019 to 2026*. 2023. Acesso em: 13 dez. 2023. Disponível em: <https://www.statista.com/statistics/477917/video-surveillance-equipment-market-worldwide/>. 3

SUDHEER, V. et al. Multipurpose surveillance system. In: IEEE. *2022 International Conference on Innovations in Science and Technology for Sustainable Development (ICISTSD)*. [S.l.], 2022. p. 244–247. 3

SUN, P.; ALJERI, N.; BOUKERCHE, A. A queueing model-assisted traffic conditions estimation scheme for supporting vehicular edge computing. In: IEEE. *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. [S.l.], 2019. p. 1–6. 17, 21

TIAN, T. et al. Availability evaluation and maintenance optimization of balanced systems considering state-dependent inspection intervals. *IEEE Transactions on Reliability*, IEEE, 2024. 18, 20

TORRES, E. et al. Storage services in private clouds: Analysis, performance and availability modeling. In: IEEE. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2016. p. 003288–003293. 13

UGLI, D. B. R. et al. Cognitive video surveillance management in hierarchical edge computing system with long short-term memory model. *Sensors*, MDPI, v. 23, n. 5, p. 2869, 2023. 17, 20

VINÍCIUS, L. et al. Docker platform aging: a systematic performance evaluation and prediction of resource consumption. *The Journal of Supercomputing*, Springer, v. 78, n. 10, p. 12898–12928, 2022. 36

WANG, L. et al. Design and implementation of fault injector for hybrid heterogeneous storage system. In: *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. [S.l.: s.n.], 2019. p. 10–15. 65

YOON, C.-S. et al. A cloud-based utopia smart video surveillance system for smart cities. *Applied Sciences*, MDPI, v. 10, n. 18, p. 6572, 2020. 17, 21

ZHANG, M. et al. Blockchain-based collaborative edge intelligence for trustworthy and real-time video surveillance. *IEEE Transactions on Industrial Informatics*, IEEE, v. 19, n. 2, p. 1623–1633, 2022. 17, 20

ZHIRNOV, N.; LYAKHOV, A.; KHOROV, E. Mathematical model of a network slicing approach for video and web traffic. *Journal of Communications Technology and Electronics*, Springer, v. 64, p. 890–899, 2019. 17, 20