Republic Democratic Algeria of People
Ministry of Higher Education and Scientific Research
University of Saad Dahleb – BLIDA 1
Faculty of Sciences
Department of Informatics

Report submitted for the fulfillment of the Master's degree
Domain: MI
Affiliation: Informatics
Specialization: Computer systems and networks

Design and implementation of a Simulator for performance evaluation of WSN using vacation policies

KOUADRI Maria and HENNI MANSOUR Yousra Yasmine

Supervisors:
Mme BOUTOUMI Bachira and Prof. Ould-Khaoua Mohamed

The jury:
Mr. KAMECHE Abdallah and Mme AROUSSI Sanaa

The academic year: 2022/2023

Acknowledgments

We are overwhelmed in all humbleness and gratefulness to acknowledge our depth to all those who have helped us to put these ideas, well above the level of simplicity and into something concrete.

Firstly, we would like to express our deepest gratitude and appreciation to Madame Boutoumi Bachira for her invaluable assistance and support. Her extensive knowledge in the field, as well as her willingness to provide guidance and clarification, have been instrumental in overcoming various challenges encountered during the research process. Her patience, encouragement, and commitment to excellence have been a source of inspiration. We would also like to extend our heartfelt thanks to Professor Ould-Khaoua Mohamed, whose unwavering support and guidance have been invaluable throughout our journey in completing this master's project. His profound knowledge, expertise, and teaching dedication have tremendously impacted our academic growth and personal development. Professor Mohamed's commitment to excellence, his willingness to share his wisdom, and try to challenge and inspire his students have made him an exceptional teacher. His continuous encouragement, insightful feedback, and patience have played a crucial role in shaping the direction of this research. We are truly grateful for his mentorship and the time he invested in us.

We would like to express our sincere appreciation to the jury members for their interest in this report and for taking the time to review and evaluate our work. We are truly grateful for their expertise and for recognizing the importance of our research.

We would like to express our deepest appreciation to our families for their unwavering love, support, and understanding throughout this academic endeavor. To our dear parents, who have been our pillars of strength, thank you for instilling the values of hard work, determination, and perseverance in us. Your constant encouragement and belief in our abilities have been a driving force behind our success. To our sisters, thank you for your continuous support and for being our sounding boards. Your words of encouragement and the unwavering faith you have in us have been invaluable.

To our friends, thank you for always being there to listen, offer advice, and provide a much-needed break from the demands of our research. Your unwavering belief in our abilities and your words of encouragement have been invaluable in keeping us motivated and focused. Your presence in our lives has made this academic journey more enjoyable and memorable. Thank you all.

Abstract

Sensor networks are widely used for various applications but are constrained by limited energy resources, which necessitates the development of energy-efficient network designs. The use of vacation policies is a promising approach for reducing energy consumption while maintaining network performance. This project presents the design and implementation of a simulator for the performance evaluation of sensor nodes using vacation policies for energy saving.

The proposed simulator is designed using discrete-event simulation techniques. The simulation model allows us to investigate the impact of various parameters such as arrival rates, service times, and threshold values used in vacation policies. The simulator also enables the comparison of different vacation policies and provides insights into the trade-off between energy conservation and system performance.

The report describes the underlying system dynamics, and the methods used for data collection and analysis, and provides examples of simulation studies to illustrate the usefulness of the simulator in evaluating the performance of a sensor node and guiding the design of energy-efficient sensor network systems.

Based on our analysis we demonstrate the interplay between arrival rate, buffer size, and service policies in determining system performance. By considering these factors and their trade-offs, it is possible to optimize queuing systems to achieve desired outcomes and improve overall efficiency.

Keywords

Performance evaluation, Discrete-time simulation, Sensor network, Vacation disciplines, Service differentiation, Energy conservation, Energy saving, Latency

Résumé

Les réseaux de capteurs sont largement utilisés pour diverses applications, mais sont limités par des ressources énergétiques limitées, ce qui nécessite le développement de conceptions de réseau économes en énergie. L'utilisation de politiques de vacances est une approche prometteuse pour réduire la consommation d'énergie tout en maintenant les performances du réseau. Ce projet présente la conception et la mise en œuvre d'un simulateur pour l'évaluation des performances des nœuds capteurs utilisant des politiques de vacances pour économiser l'énergie.

Le simulateur proposé est conçu à l'aide de techniques de simulation à événements discrets. Le modèle de simulation nous permet d'étudier l'impact de divers paramètres tels que les taux d'arrivée, les temps de service et les valeurs de seuil utilisées dans les politiques de vacances. Le simulateur permet également la comparaison de différentes politiques de vacances et fournit des informations sur le compromis entre la conservation de l'énergie et les performances du système.

Le rapport décrit la dynamique sous-jacente du système, les méthodes utilisées pour la collecte et l'analyse des données, et fournit des exemples d'études de simulation pour illustrer l'utilité du simulateur dans l'évaluation des performances d'un nœud capteur et guider la conception de systèmes de réseaux de capteurs économes en énergie.

Sur la base de notre analyse, nous démontrons l'interaction entre le taux d'arrivée, la taille du tampon et les politiques de service dans la détermination des performances du système. En tenant compte de ces facteurs et de leurs compromis, il est possible d'optimiser les systèmes de file d'attente pour atteindre les résultats souhaités et améliorer l'efficacité globale.

Mots clés

Évaluation des performances, Simulation en temps discret, Réseau de capteurs, Disciplines de vacances, Différenciation des services, Économie d'énergie, Économie d'énergie, Latence.

ملخص

تُستخدم شبكات الاستشعار على نطاق واسع في العديد من التطبيقات ولكنها مقيدة بمصادر الطاقة المحدودة، مما يستلزم تطوير تصميمات شبكات موفرة للطاقة. يعد استخدام سياسات الإجازات نهجًا واعدًا لتقليل استهلاك الطاقة مع الحفاظ على أداء الشبكة. يقدم هذا المشروع تصميم وتنفيذ جهاز محاكاة لتقييم أداء عقد الاستشعار باستخدام سياسات الإجازة لتوفير الطاقة.

تم تصميم جهاز المحاكاة المقترح باستخدام تقنيات محاكاة الأحداث المنفصلة. يسمح لنا نموذج المحاكاة بالتحقيق في تأثير المعلمات المختلفة مثل معدلات الوصول وأوقات الخدمة وقيم العتبة المستخدمة في سياسات الإجازة. يتيح المحاكي أيضًا مقارنة سياسات العطلات المختلفة ويوفر رؤى حول المفاضلة بين الحفاظ على الطاقة وأداء النظام.

يصف التقرير ديناميكيات النظام الأساسية، والطرق المستخدمة لجمع البيانات وتحليلها، ويقدم أمثلة لدراسات المحاكاة لتوضيح فائدة المحاكي في تقييم أداء عقدة المستشعر وتوجيه تصميم أنظمة شبكات الاستشعار الموفرة للطاقة.

بناءً على تحليلنا، نوضح التفاعل بين معدل الوصول وحجم المخزن المؤقت وسياسات الخدمة في تحديد أداء النظام. من خلال النظر في هذه العوامل والمفضالات الخاصة بها، من الممكن تحسين أنظمة قائمة الانتظار لتحقيق النتائج المرجوة وتحسين الكفاءة العامة.

الكلمات الدالة

تقييم الأداء، محاكاة الوقت المنفصل، شبكة المستشعرات، تخصصات الإجازات، تمايز الخدمة، الحفاظ على الطاقة، توفير الطاقة، زمن الوصول

Contents

List of abbreviations

ADCs	Analog to Digital Converters
BMAP	Batch Markovian Arrival Process
CSMA/CA	Carrier-sense multiple access with collision avoidance
D	Deterministic
DES	Discrete Event Simulation
ECI	Energy Consumption during Idle State
ECb	Energy Consumption during Busy State
ECd	Energy Consumption during Delayed Idle State
ECs	Energy Wasted During State Transitions
ECTx	Energy Consumption per Packet in Queue
ESNs	Environmental Sensor Networks
FIFO	First-In, First-Out
GI	General Independent
IEEE	Institute of Electrical and electronics engineers
IDE	Integrated Development Environment
LLC	Logical Link Control
LR-WPANs	Low-Rate Wireless Personal Area Networks
MAC	Medium access control
MAP	Markovian Arrival Process
M	Markov or Memoryless
PHY	Physical Layer
PDF	Probability Density Function
PORT	Portable Operating System Interface (possible expansion)
PR	Priority Service
PSFQ	Packet-Scale Feedback Queueing
QoS	Quality of Service
RNG	Random Number Generation
SIRO	Service in Random Order
SPT	Shortest Processing Time First
SSCS	Service Specific Convergence
STCP	Sensor Transport Control Protocol (possible expansion)
TCP	Transmission Control Protocol
TDMA	Time division multiple access
Tnow	Current Time in the Simulation
WSN	Wireless Sensor Network
ADCs	Analog to Digital Converters

List of Figures

Figure 33. Performance Results from D Policy Simulation Model Measuring Energy Consumption vs. Threshold D.
Figure 34. Performance Results from Different Policy Simulation Model Measuring Mean Waiting Time vs. service rate μ.
Figure 35. Performance Results from Different Policy Simulation Models Measuring Average Energy Consumption vs. service rate μ.

List of tables

List of Routines

General Introduction

Similar to any living being, the smart environment relies first and foremost on sensory data from the real world, a wireless sensor network (WSN) consists of a large number of sensor nodes that are deployed over an area to perform local computations based on information gathered from the surroundings [1]. Sensors are expected to run autonomously for long periods. However, they are occupied with batteries and it's almost difficult to change or recharge batteries. Therefore, [2] the fundamental query is "How to extend the lifespan of a battery?"

In order to address this matter, researchers have been prompted to explore ways to conserve battery life and maximize usage in sensor networks. Energy-efficient routing protocols can be designed to reduce the energy consumed during data transmission [1], taking into account the limited resources of sensor nodes. Additionally, energy harvesting techniques, such as solar or thermal energy, can be used to recharge the sensor nodes' batteries. However, in this project, we are simulating the performance of such a system Using vacation policies.

A single node can be modeled as a queue that receives data packets and serves them. This phenomenon is observed in everyday life as waiting queues, and it is an important area of research. Classic queues can be modeled according to various parameters; queuing with vacancies is a type of classic queue that is more representative of reality. Server idleness is restricted by vacation policies, which are a set of rules that determine when the server goes idle and when it resumes operation. In this project, we will simulate the performance of such a system using discrete event simulation to evaluate its performance under different conditions. By analyzing the simulation results, we can gain insights into the system's behavior and identify areas for improvement.

To achieve the following objectives

- **Energy Efficiency Evaluation:** The assessment of energy efficiency in WSNs with the aim of extending the battery life of sensor nodes while simultaneously enhancing their performance.
- **System Performance Analysis:** The utilization of discrete-event simulation models to analyze the performance of queueing systems under various conditions.
- **Sensor Node Resource Optimization:** An investigation into methods that involve exploring various vacation policies within queueing systems, with a primary emphasis on achieving a balance between energy conservation and system efficiency.
- **Policy Implication Examination:** An analysis and comparison of different vacation policies, particularly threshold-based policies, in order to comprehend their impact on system behavior.
- **Exploration of Future Directions:** Providing suggestions for potential future research directions that can contribute to the advancement of the wireless sensor networks field.

The rest of the report is organized as follows:

- Chapter 1: Provides background on wireless sensor networks, covering applications, architecture, components, communication protocols, and energy efficiency in WSNs.
- Chapter 2: Focuses on queues with vacations, explaining their structure, characteristics, notation, Little's Law, steady-state system, threshold-based policies, and comparing queues with vacations to classic queues.
- Chapter 3: Gives an overview of discrete-event simulation, covering its purpose, steps, data collection methods, measurement and analysis of simulation performance, model verification, and validation, and the application of queues with vacation for energy conservation in sensor networks.
- Chapter 4: Discusses the implementation of the simulation model, including the simulation environment, basic model, different simulation models (M/M/1, M/M/1/K, etc.), implementation of the delayed idle state, performance metrics, and methods for collecting simulation results.
- Chapter 5: Focuses on the experimental study, presenting numerical results, discussing steady state and system parameter values, analyzing variations in arrival rate, threshold, and service rate, and drawing overall conclusions from the experimental study.
- The general conclusion summarizes the findings of the project and restates the objectives. It highlights the contributions and insights gained from the simulation study. The conclusion also discusses the implications of the research and provides recommendations for future work in the field.

Chapter 1: Fundamentals of Wireless Sensor Networks and Queueing Systems.

Wireless sensor networks are crucial for real-time monitoring, enabling data collection from remote areas. They enhance decision-making, resource management, and productivity across industries.

In this chapter, we will be exploring the fascinating world of wireless sensor networks (WSNs). We explore their fundamental aspects, including their components, architecture, and communication protocols.

1. Wireless sensor networks

The wireless sensor network is a kind of network composed of nodes associated with sensors, and these nodes have characteristics such as small size, low computation power, limited power, and wireless access [3] . A node that generates data is called a source node, while a node that requests data is called a sink or sink node [4]. So, all these sensor nodes are responsible for collecting and delivering data over the wireless network, and these data should be kept confidential along the wireless transmission path from one node to another [3]. The below *Figure 1* illustrates the sensor network model, which includes a single sink node, also known as the base station, and numerous sensor nodes deployed across a vast geographic area, referred to as the sensing field.

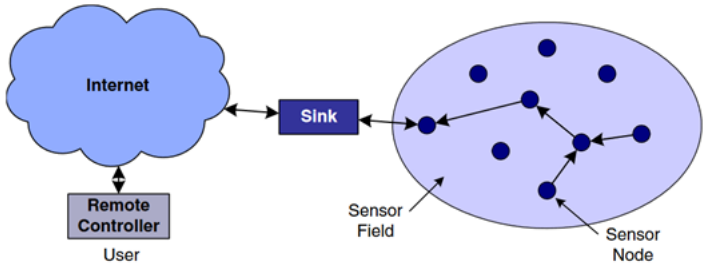


Figure 1. *Sensor network architecture* [5].

2. WSN Applications

the various conceivable applications of WSNs to every sector globally are essentially boundless, which is why they gained great admiration because of their flexibility in resolving issues in different application fields such as in:

-military communication, frontline surveillance, investigation and targeting systems, and other more… [6]

-agriculture it is also beneficial to farmers in tasks such as wiring in challenging environments, optimizing irrigation practices for efficient water use, and minimizing waste [6].

-Environmental sensor networks (ESNs), are used in a lot of environmental and earth science research. They also aid in agricultural and environmental sustainability. Key applications include air pollution, forest fires, greenhouse management, and landslide detection [6].

Even though sensor networks are used in different domains but they face a lot of challenges because they do not fit into any regular topology, due to [7]:

-their scattering during deployment

-their resources such as memory, computation, and power are really limited

-Maintenance is challenging due to fewer infrastructures.

-Unreliable communication and data transfer.

-Sensor nodes rely solely on batteries, which cannot be recharged or replaced easily.

-Dealing with node failures, changes in network structure, and adding or removing nodes can be difficult.

3. Sensor networks components and architecture

3.1 sensor node component

A sensor node comprises several essential components, as depicted in *Figure 3*. These components include a sensing unit, a processing unit, a transceiver unit, and a power unit. Additionally, depending on the specific application, there may be additional components such as location finding system, a power generator, and a mobilizer. The sensing unit typically consists of sensors and analog-to-digital converters (ADCs). The sensors capture analog signals related to the observed phenomenon, which are then converted to digital signals by the ADCs. These digital signals are then processed by the processing unit, which is equipped with a small storage unit. The processing unit manages the necessary procedures for the sensor node to collaborate with other nodes performing assigned sensing tasks. The transceiver unit facilitates the node's connection to the network. One crucial component of a sensor node is the power unit, which may be supplemented by a power scavenging unit like solar cells. Additionally, there may be other application-dependent subunits included [8].

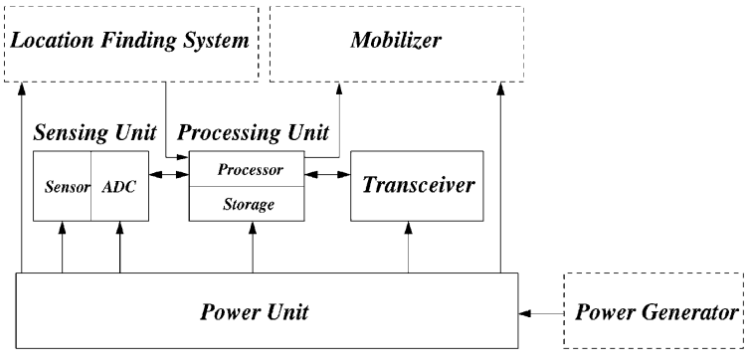


***Figure 2**. The elements of a sensor node* [8]

3.2 Layered architecture network

This network utilizes numerous sensor nodes and a base station. The nodes can be organized into concentric layers, including five main layers and three cross layers as depicted in *Figure 2*

The five layers in the architecture are

i. Application Layer
ii. Transport Layer
iii. Network Layer
iv. Data Link Layer
v. Physical Layer

The next three cross-layers are employed to control the network and synchronize the sensors, resulting in improved network efficiency, as compared to the above-mentioned five layers [9]

i. Power Management Plane
ii. Mobility Management Plane
iii. Task Management Plane

The following figure provides a visual representation that highlights the layers:

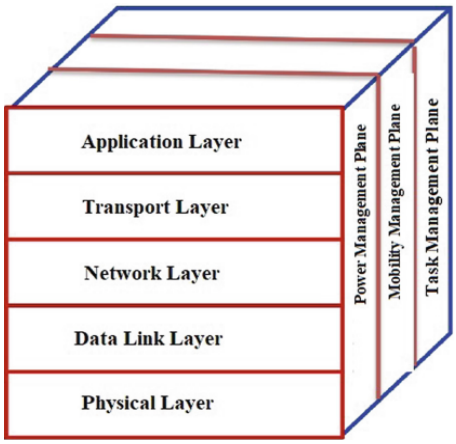


Figure 3. *Wireless Sensor Network Architecture* [10]

4. Communication protocol in WSN

There are three layers and several protocols used in Wireless Sensor Networks (WSNs), as illustrated in *Figure 4*. But the data link layer, specifically the MAC (Media Access Control) layer, is the primary focus when studying energy conservation in a sensor node using vacation policies and simulation methods. This layer controls access to the wireless medium and manages communication, making it a critical point for optimizing energy consumption. By implementing vacation policies at the MAC layer, the sensor node can intelligently schedule its active and sleep periods, leading to efficient energy usage. Simulation-based studies at the MAC layer allow for evaluating the effectiveness of different policies and scheduling mechanisms, enabling researchers to assess energy conservation and performance metrics. Therefore, focusing on the data link layer provides a targeted approach to enhance energy efficiency in sensor networks.

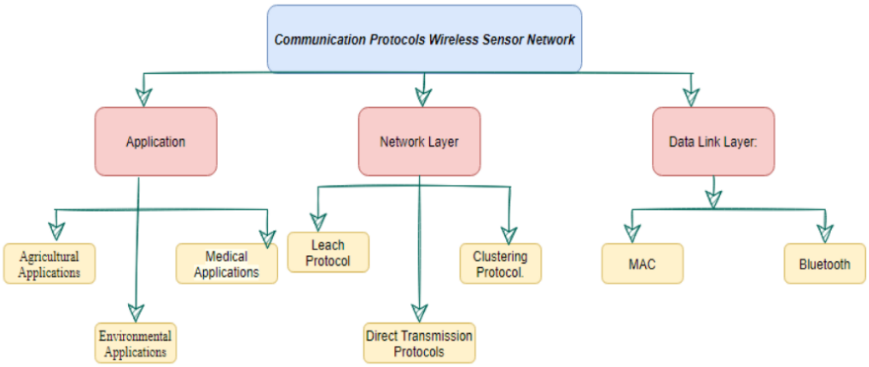


Figure 4. *Communication protocols wireless sensor network* [11]

4.1 Medium access control

The Medium Access Control (MAC) layer is a sublayer of the data link layer in the OSI model, its main function is frame delimiting and recognition, addressing, data transfer between upper layers, error protection using frame check sequences, and managing access to a shared channel among all nodes.

Overall, it plays a crucial role in managing access to the shared channel and ensuring efficient and reliable communication within a WSN using the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) mechanisms. [11]

- **CSMA/CA** The IEEE 802.11 MAC layer operates based on Carrier-sense multiple access with collision avoidance (CSMA/CA) protocols. In CSMA/CA, a node that wants to transmit a packet first check if there is an ongoing transmission. If there is, the node waits until the current transmission is complete. Afterward, it waits for a specific period called the short interframe space. If the medium is still not busy during this time, the node proceeds with its transmission. However, if there is traffic on the medium, the node must wait once again for the medium to become clear [12]

Energy efficiency and adaptability are the key considerations in designing MAC protocols for WSNs to prolong the network's lifespan. [11]

5. IEEE 802.15.4 standard

The IEEE 802.15.4 standard, designed in 2003, specifically targets Low-Rate Wireless Personal Area Networks (LR-WPANs) with low data throughput and limited power and computation resources. It addresses issues associated with existing standards like Wi-Fi and Bluetooth. The standard defines the PHY and MAC layers for LR-WPANs, providing specifications for physical layer communication and medium access control. It defines network topologies such as star and peer-to-peer.

The architecture of the IEEE 802.15.4 standard consists of a PHY layer (radio transceiver and low-level control) and a MAC layer (data transfer definitions). The service-specific convergence (SSCS) and IEEE 802.2TM Type 1 logical link control (LLC) provide a standard mechanism for upper layers to access the PHY and MAC layers. The simplicity of the IEEE 802.15.4 architecture allows developers to design low-level application software that interacts directly with data transfer. This simplicity is desirable for wireless sensor network

applications due to their limited resources, while more traditional standards based on the OSI model are too complex for WSN development. [12]

6. Energy efficiency in WSN

The primary focus of this exploration is on energy-efficient management techniques, as they play a pivotal role in extending the operational lifespan of sensor nodes, especially in mission-critical applications. Energy consumption in sensor nodes can be categorized into two main types: "useful" energy consumption and "wasteful" energy consumption. Useful energy consumption encompasses activities such as data transmission, query processing, and data forwarding to neighboring nodes. On the other hand, wasteful energy consumption results from various inefficiencies, including idle listening, collisions, overhearing, control-packet overhead, and over-emitting. It is imperative to address these wasteful energy consumption factors through well-designed protocols to prevent unnecessary energy depletion [1] ***Table.1*** summarizes the technologies used for improving energy efficiency in WSNs.

Table 1. *Classification of Existing Technologies.*

Technology	**Objective**
Power-Conscious Protocols	Reduce energy usage in sensor nodes.
Energy Harvesting	Replenish energy supplies from the environment.
Data Aggregation	Compress data to conserve energy during transmission.
Routing Protocols	Find energy-efficient pathways while ensuring reliable data transfer.
Dynamic Power Management	Adjust power usage based on network and application conditions.
Sleep Modes in Sensor Nodes	Reduce energy consumption during idle periods in our project.

➔ Types of Sleep Modes in Sensor Nodes

Duty cycling

takes place when nodes alternate between active and sleep states regularly. This implies that the node is active for a certain length of time, known as the active period, during which it detects or communicates data, and then goes into sleep mode for a set amount of time, known as the sleep period, during which it conserves power. The duty cycle ratio is the ratio of the active period to the entire cycle period (active and sleep periods combined). The duty cycle ratio can be changed depending on the application and network conditions [1].

Sleep scheduling

the fundamental approach to sleep scheduling involves choosing a subset of nodes to be active in a given time interval while the remaining nodes enter a sleep state, thereby minimizing power consumption and reducing overall energy usage. Existing research on sleep scheduling in Wireless Sensor Networks (WSNs) primarily concentrates on two objectives point coverage and node coverage. Point coverage aims to ensure that the awake nodes in each time slot cover every point in the deployed field, with different algorithms focusing on minimizing

energy consumption or average event detection latency. On the other hand, node coverage focuses on constructing a globally connected network where each sleeping node is near at least one active node. However, these studies typically concentrate on the medium access layer of static WSNs with stationary nodes [13].

In our project, we will be focusing on the implementation and optimization of sleep modes in sensor nodes, specifically duty cycling. This technique is critical to reducing energy consumption during idle periods, which aligns with our project's goal of improving energy efficiency in WSNs. By effectively managing the duty cycles of a sensor node, we aim to extend their operational lifespan and enhance the overall performance and reliability of the network.

7. Queueing Systems

Queueing theory, initially introduced by Agner Erlang in 1909, is a widely studied theory that finds applications in various fields such as telecommunication and computer science. Queueing systems, which are at the core of this theory, have garnered significant attention from both academic institutions and industry [14].

A basic queueing system is a service system where “customers” arrive at a bank of “servers” and require some service from one of them. It’s important to understand that a “customer” is whatever entity is waiting for service and does not have to be a person. For example, in a “back-office” situation such as the reading of radiologic images, the “customers” might be the images waiting to be read. Similarly, a “server” is the person or thing that provides the service [15].

8.1 Queueing System Structure

Within a queueing system, entities arrive and wait for service, proceed to single or multiple stations to receive the service, and subsequently have the option to either exit the system or continue within it [16].

Arenales (2007) identifies that a queuing system can be classified into 4 types [16]

a) single queue and a server
b) single queue and multiple servers in parallel
c) multiple queues and multiple servers in parallel
d) single queue and multiple servers in series

7.2 Characteristics of Queueing System

A queuing system is composed of entities (packets –or anything that arrives in a system and needs a service) and servers (any resource that provides a service)

To evaluate a queueing system and measure its inputs and outputs, it becomes essential to define specific characteristics that aid in modeling the system effectively**.**

These characteristics are [16].

1- **The arrival pattern of customers** represents the rate at which customers arrive, indicating the distribution of time intervals between the arrival of one entity and the next.

2- **The service pattern of servers** is the measurement of service time. It is crucial to understand the sequence of services and the total number of services provided by the system.
3- **The number of servers** is a characteristic that impacts the service time. It determines whether the system will incur higher costs or have more delays. Additionally, it is important to define the type of queue, whether it is a single queue or multiple queues within the system
4- **System capacity** is a measure or rate, that defines the space for the entrance of entities. In certain situations, this capacity must be limited, while in many cases, it remains unknown, resulting in an unlimited rate.
5- **Queue discipline** This refers to queue behavior based on customer actions when entering the server and waiting for service. The most commonly observed queue discipline is First-In, First-Out (FIFO), although others such as Last-In, First-Out (LIFO), Service in Random Order (SIRO), Shortest Processing Time First (SPT), and Priority Service (PR) exist.
6- **The number of service stages** is a characteristic that can either involve a single stage, or multiple stages, where there are queueing networks and the servers need to communicate between them.
7- **The service discipline** indicates the rules that determine how the system serves customers in the Classic queue.
    - **Exhaustive service** the server will continue to work with a customer until their service is completed under this form of service discipline. This implies that the consumer will not leave the system until they have gotten complete service, no matter how long it takes. This form of service discipline is frequent in instances when the Packet has a specific demand that the server must address [17].
    - **Non-exhaustive service** even if the service is incomplete, the server will only work with a Packet for a set length of time before moving on to the next customer. This implies that if the customer's service was not finished during the initial conversation, they may need to return to the line. This service discipline is prevalent in instances when the server must handle a large number of Packets fast and cannot devote too much time to each consumer [17].

7.3 Queueing Notation

In the fundamental queueing model, individuals referred to as "customers" arrive to request service, wait if necessary for an available "server" to provide the required service, and eventually depart. Therefore, this model comprises three key elements (i) the stochastic arrival process, (ii) the stochastic service requirements, and (iii) the physical arrangement of servers and their operational guidelines. The primary goal of this theory is to comprehend the interplay among these components and the system's behavior, as measured by its performance metrics [18].

In a now-classic 1953 paper, D.G. Kendall [18] proposed the queueing notation to describe how a queue system is classified [16], The presented table illustrates the key and widely used notations that are crucial for classifying and categorizing this study.

In this context, the notations used are as follows A represents the distribution of interarrival times, B represents the distribution of service times, c represents the number of servers, Y represents the system capacity, and Z represents the queue discipline [16].

It is important to introduce the shorthand notation explicitly when using Kendall-like notation to describe a specific model, rather than assuming the reader's understanding. Caution must be exercised when interpreting such notation to ensure a comprehensive grasp of the underlying model. Keeping this caution in mind, we will now proceed with an explanation of Kendall's notation [18].
Here are the conventions [18].

- G general (no particular assumption)
- GI general independent (the random variables in question are mutually independent and identically distributed)
- M Markov or memoryless (exponential random variables)
- D deterministic (the same constant value for each realization of the random variable)
- Ek k-phase Erlangian (sum of k independent, identical, exponentially distributed ‘‘phases’’)
- PH phase-type (sum of a (possibly random) number of independent, exponentially distributed phases)
- MAP Markovian arrival process
- BMAP batch Markovian arrival process

Some examples may help to better understand

The queuing system M/M/1/∞/FIFO represents a queueing system with a single server and unlimited capacity. In this system, both the interarrival times and service times follow exponential distributions, and the queue discipline is first-in, first-out (FIFO) [16].

M/G/1 The interarrival time distribution (A) follows an exponential distribution (M) with a constant rate. The service time distribution (B) follows a general distribution (G), which can be any arbitrary distribution.

Table 2. *Queueing Notation.* [16]

Characteristic	Symbol	Explanation
Interarrival-time distribution (A) Service-time distribution (B)	M	Markov Exponential
	D	Deterministic
	E_K	Erlang type k (k= 1, 2, …)
	H_K	Mixture of k exponentials
	PH	Phase type

	MAP	Markovian arrival process
	BMAP	Batch Markovian arrival process
	G	General
	GI	General Independent
Parallel servers (c)	1,2,…..,∞	
System capacity (Y)	1,2,…..,∞	
Queue discipline (Z)	FIFO	First in, First out
	LCFS	Last come, first served
	RSS	Random selection for service
	PR	Priority
	GD	General discipline

In the queuing system M/M/1/K, the system capacity or the maximum number of customers that the queue can accommodate simultaneously is K. When a customer arrives at the queue and finds all K positions occupied, it means that the buffer is full, and the customer is unable to enter the system. In this case, the customer will leave the system without joining the queue. This occurs when the system capacity is already reached. The probability of rejection can be calculated based on the arrival rate and the system capacity, as shown as rejected arrivals in **figure 6**

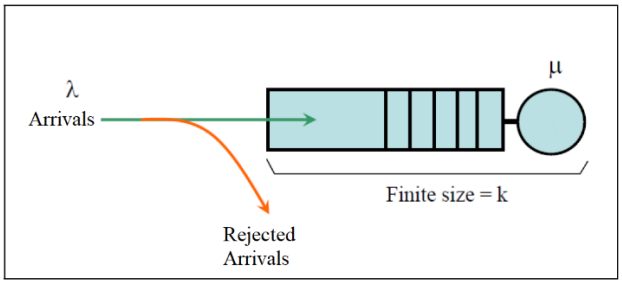


Figure 5. *A Single Node, One Server, and a limited buffer* [19].

In the context of the M/M/1 queuing model, the "M" represents Markovian, which is related to the use of exponential distributions. Specifically, it signifies that the arrival and service times in the model follow exponential distributions. This type of distribution is often employed for models that involve distinct and unrelated events. For example, the arrival time between a large number of customers happens independently of one another. The exponential

distribution is a continuous probability distribution that has a probability density function (PDF) [16].

8. Queuing performance measures

Little's Law One of the most important theorems in queuing theory that we used in our implementation is Little's law. Little's Law is an equation that states $L = \lambda W$, where L represents the expected number of customers in the "system," W represents the expected time spent by a customer in the system, and λ is the rate at which the customers enter the system. Initially, L represented the length of the queue, W represented the wait time in the queue, and λ represented the frequency of customer arrivals [18].

9. Steady state

In the context of queuing systems, a steady state refers to a situation where the system has reached a stable condition in terms of its performance measures. A steady state system implies that the system's behavior remains consistent over time, and its statistical properties do not change, one way to ensuring system stability is calculating traffic Intensity.

- **Traffic Intensity (Server Utilization)** Assuming λ, is the average rate of customers entering the system and μ, is the average rate of serving customers and c is the number of servers in the system, then the quantity $\rho = \lambda / c\ \mu$ is called the traffic intensity (also called the utilization factor or server utilization), ρ gives the fraction of time that the server is busy.

For the steady-state conditions to exist it is required that $\lambda < \mu$ ($\rho < 1$). This is the stability condition for the M/M/c systems.

- When the average number of arrivals into the system is more than the maximum number of customers the system can serve, i.e., $\rho > 1$ this means that the queue size never settles down, and there is no steady state.
- When the arrival rate equals the maximum average service rate of the system, i.e., $\rho = 1$, the randomness will prevent the queue from ever emptying and allowing the server to catch up, and this causes the unbounded growth of the queue. In this case, the steady state does not exist unless arrivals and services are deterministic and perfectly scheduled. [20]

10. Queues with Vacation

A vacation queueing model is an expanded version of the classic queueing system, where the server may occasionally become unavailable for a certain duration due to reasons such as maintenance checks, server damage, or personal breaks. This absence of the server is referred to as a "server vacation." Queueing models incorporating server vacations are important in understanding the queueing dynamics, as the server can utilize the idle period for various purposes, such as economizing energy in our case [21].

10.1 Threshold vacations

Also known as threshold-based vacation policies, are a concept used in queueing systems to manage the server's idle periods based on specific thresholds or conditions. In threshold vacations, the server remains idle until a certain condition is met, triggering a

vacation period. These conditions can be based on various factors such as queue length, accumulated work, or arrival rate [22].

During a threshold vacation, the server does not process any incoming requests and remains idle. The purpose of threshold vacations is to optimize system performance, resource utilization, and energy efficiency. By allowing the server to enter a vacation state during periods of low demand or when specific criteria are not met, unnecessary energy consumption can be minimized, reducing operational costs. Threshold vacations can be implemented with different policies, depending on the specific objectives and requirements of the system. These policies determine the specific conditions and thresholds for entering or ending a vacation period, ensuring that the server remains idle or resumes service at appropriate times.

10.2 Delayed Idle State

In the context of queueing systems, the idle delayed state is an extension of the traditional M/M/1 queue model. It introduces an additional state called the "delayed idle" state, which occurs when the server transitions to an idle state but remains in that state for a certain delay period before accepting new arrivals. This delay is known as the "idle delayed time" or "delayed vacation time."

The idle delayed state aims to address the impact of frequent transitions between the idle and busy states in traditional queueing systems. These transitions can lead to energy inefficiency and increased overhead due to the setup and teardown processes associated with each transition.

In the idle delayed state, when the server becomes idle, it enters the delayed idle state instead of immediately accepting new arrivals. During this delayed idle period, the server remains idle for a fixed duration known as Θ. If no arrivals occur during this delay period, the server transitions back to the idle state. However, if an arrival occurs before the delay expires, the server transitions to the busy state and serves the arriving packet.

11. Why use queues with vacation and not classic queues

Queueing systems with vacations can be advantageous over classic queues when it comes to energy-saving considerations. By incorporating scheduled breaks for servers, vacations allow for planned periods of non-service, which can be leveraged to conserve energy during low-demand periods. The server can be powered down or put into a low-power mode during these vacations, resulting in reduced energy consumption. This approach optimizes energy usage by aligning server availability with the actual demand, offering energy-saving benefits that are not possible in classic queues, where the server remains continuously active.

12. State of art on the application of queues with Vacation for energy conservation in sensor networks

Much of the prior research has been primarily focused on the development of mathematical node models, often neglecting the significant impact of real-world constraints on the performance of WSNs. Moreover, the available simulators in the field tend to either be

proprietary or designed with a broader focus on the entire WSN, rather than addressing the intricacies of practical limitations. The following are some related work descriptions

Biplab Sikdar and Mounir Hamdi [23] introduced an adaptive N-policy queueing system design to address the energy-delay tradeoff in wireless sensor networks. The authors propose an analytical model to characterize the tradeoffs between energy saving and latency in wireless devices. They argue that increasing the N-policy does not necessarily result in more energy savings. Based on analytical and simulation results, they present a scheme for the optimal selection of N considering arrival and service rates. The proposed system design is shown to save energy while meeting delay requirements. The paper highlights the importance of power management schemes in sensor systems and discusses the dynamic selection of N to adapt to real-time traffic, aiming to save energy without violating delay constraints. The work contributes by providing a practical approach to optimizing energy usage and system performance in wireless networks.

Goswami Veena and G. B. Mund [24] proposed a queue-based technique to enhance the lifespan of wireless sensor networks (WSNs). They investigate the effect of queueing schemes on node power consumption and suggest a modified N threshold queueing system to minimize energy usage. The model considers three node states (sleep, idle, and busy) and employs an M/M/1 queueing system. Once the packet threshold N is reached, packets are served in a single batch, reducing waiting time. The paper analyzes power consumption and its dependence on system parameters, presenting performance indices such as average queue length, expected periods, probabilities, and energy consumption rate. The authors discuss numerical results and the influence of the threshold value N on system behavior. They conclude by offering insights into optimal decisions based on the model, improving understanding of the proposed technique for prolonging WSN lifetimes.

Messous Ali and Mameche Mohamed [25] in their report provided a study utilizes a queuing system with a vacation policy N and multiple priority classes, which is considered an effective approach for modeling the problem at hand. Discrete event simulation was employed to gain a deeper understanding and analysis of the targeted system, allowing them to focus on important events. The developed simulation tool enables the testing and evaluation of various performance measures for a WSN, examining the impact of vacation policies, priority classes, and service preemption on system behavior and performance metrics. The tool facilitates the comparison of results between systems with/without priority classes and preemption.

The objective of their work was to design and implement a simulation tool that can analyze the behavior of an WSN based on different customer treatment scenarios, considering factors such as priority order, service preemption, and queue size limitations. Discrete event simulation is identified as the most suitable method due to its ability to prioritize critical system events.

Boutoumi Bachira and Nawel Gharbi [26] proposed an energy-saving and latency delay efficiency scheme for wireless sensor networks (WSNs) based on Generalized Stochastic Petri Nets (GSPNs). their goal was to optimize energy consumption by extending the duration of sleeping states of sensor nodes. The authors first model networked nodes' sleep/wakeup

pattern with different vacation policies using GSPNs. They introduce the N-policy as a queued wakeup scheme and propose a new vacation policy called the Hybrid-policy to minimize latency. They provide formulas for performance measures and analyze the impact of the two vacation policies on network performance. The paper concludes with numerical calculations using GSPN models and presents the potential benefits of the proposed schemes.

A concise summary of the research works discussed in the preceding sections represented in the following table

***Table 3**. Summary of the existing work*

Authors	Key Contributions	Methodology	Findings
Biplab Sikdar and Mounir Hamdi [23]	- Introduced an adaptive N-policy queueing system design for wireless sensor networks -Analyzed the energy-saving and latency tradeoffs -Proposed optimal N selection scheme based on analytical and simulation results	Analytical modeling and simulation	Increasing the N-policy does not necessarily result in more energy savings.
Goswami Veena and G. B. Mund [24]	- Proposed a modified N threshold queueing system to enhance the lifespan of wireless sensor networks - Analyzed the effect of queueing schemes on power consumption - Presented performance indices and energy consumption rate	Queue-based technique with an M/M/1 queueing system	The modified N threshold queueing system effectively reduced waiting time and minimized energy usage in wireless
Messous Ali and Mameche Mohamed [25]	- Provided an overview of WSNs and their challenges - Developed a simulation tool to analyze the behavior of WSN - Analyzed the impact of vacation policies and priority classes	Discrete event simulation with vacation policy N and multiple priority classes	Utilizing a queuing system with a vacation policy N and multiple priority classes was an effective approach for modeling energy conservation in WSNs

Boutoumi Bachira and Nawel Gharbi [26]	- Presented an analytical model based on priority vacation queueing theory for WSNs -Addressed energy consumption and latency reduction -Demonstrated the efficiency of the proposed Hybrid policy	Generalized Stochastic Petri Nets (GSPNs). modeling and analysis	The Hybrid-policy surpasses the N-policy by achieving significant reductions in both energy consumption and latency delay.

conclusion

In this chapter, we've explored the essential aspects of WSNs, ranging from their applications and components to communication protocols and energy efficiency. we have also explored the theory of classical queues and found that standard models do not accurately capture the behavior of packets and servers. However, the presence of vacancies complicates queue analysis, and analytical solutions are only feasible for a few models with specific assumptions. As a result, researchers often rely on numerical approaches, approximations, or simulations to overcome these limitations. The state-of-the-art on energy conservation in sensor networks were also explored. These insights set the stage for further analysis, which is why we aimed to develop our own simulation tool. In the next chapter, we will delve into simulation concepts and techniques as valuable tools for studying waiting lines and surpassing the constraints of analytical methods.

Chapter 2: Simulation modeling

In this Chapter, we will be exploring the intriguing realm of discrete-time simulation, a vital tool for modeling and evaluating dynamic systems. Discrete-time simulation is the process of portraying systems as sequences of events or states that change across discrete time intervals. This chapter gives an in-depth look at this powerful simulation approach, delving into its underlying concepts, and many components.

1. Justification of the method of Study

Performance evaluation is vital in computer system design and decision-making. It allows for comparing design options and finding the best solution. System administrators depend on performance evaluation to assess and select systems for specific applications. Even without alternatives, evaluating the existing system provides insights into efficiency and areas for improvement [27].

And there are 3 technics for performance evaluation analytical modeling, simulation, and measurement, we choose which one to use based on several considerations, listed in Table ***3*** that show different criteria for each technic.

Table 4. *A comparison between the 3 technics for performance evaluation.* [27]

Criterion	Analytical Modeling	Simulation	Measurement
1. Stage	Any	Any	Post prototype
2. Time required	Small	Medium	Varies
2. Tool	Analysts	Computer languages	Instrumentation
3. Accuracy	Low	Moderate	Varies
5.Trade-off evaluation	Easy	Moderate	Difficult
6.Cost	Small	Medium	High
7. Salability	Low	Medium	High

In our case, we used simulation over analytic modeling even though both can be used in designing an improved version of a product because Simulations are better at reflecting real-world situations because they include more details and make fewer assumptions compared to analytical modeling. Analytical modeling simplifies and assumes a lot, so when the results turn out to be accurate, even the analysts are surprised. On the other hand, simulations provide a more comprehensive and realistic picture of the system, making them valuable for understanding complex phenomena and obtaining accurate insights [27].

2. Steps of Simulation Study

The steps involved in developing a "simulation model", designing a simulation experiment, and performing simulation analysis are

Step 1. Identify the Problem List the issues present in the system and outline the desired features and requirements for the proposed system [28].

Step 2. Formulate the Problem specify the system boundaries (the problem to be studied), define the objective of this simulation, define performance metrics, and decide the time of this study and the end-user of the simulation model [28].
Step 3. Collect and Process Real System Data, gather data on the specification of the system, input variables, as well as the performance of the existing system [28].
Step 4. Formulate and Develop a Model, create system schematics and network diagrams, convert them to simulation software format, and validate the model through techniques like trace analysis, parameter testing, output evaluation, constant substitution, manual verification, and animation [28].
Step 5. Validate the Model Evaluate model's performance against real system performance in known conditions. Seek expert evaluation, asses user confidence, and address problems if any [28].
Step 6. Document Model for Future Use Document objective, assumptions and input variables, and the experimental design [28].
Step 7. Select Appropriate Experimental Design Define the measure of performance and input variables along with their levels. Decide whether to use terminating or nonterminating simulation runs. Select the appropriate run length and the appropriate starting conditions, including any required warm-up period. Determine the number of independent runs based on desired confidence levels. Consider using common random numbers for comparing different configurations. Identify correlated output data [28].
Step 8. Establish Experimental Conditions for Runs Address the question for obtaining precise information from each run, and determine if the system exhibits stationary behavior (performance measure does not change over time) or non-stationary (performance measure changes over time) [28].
Step 9. Perform Simulation Runs Perform runs according to steps 7-8 above.
Step 10. Interpret and Present Results calculate numerical estimates of the desired performance measure for each configuration of interest. Test hypotheses about system performance, and create visual representations, such as pie charts or histograms, to illustrate the output data. Document results and conclusions [28].
Step 11. Recommend Further Courses of Action like experiments to increase the precision and reduce the bias of estimators, to perform sensitivity analyses, etc. [28]
- While this sequence of steps provides a logical framework for a simulation study, multiple iterations, and adjustments may be necessary to achieve the study's objectives. Not all steps may be feasible or necessary, while additional steps may be required.

3. Discrete Event Simulation

Discrete event simulation (DES) is a powerful tool for studying real-life processes and systems. It models events occurring over time, such as packet arrivals and departures, enabling a realistic representation of individual components. DES allows us to evaluate the performance of vacation policies in sensor networks, considering metrics like packet delay, throughput, energy consumption, and service differentiation. By adjusting parameters and simulating different scenarios, we can assess the impact of policies on network performance. DES also provides methods for collecting observation data and analyzing performance

metrics, ensuring accurate and reliable results. Simulation allows decision-makers to test and explore alternative policies and make informed decisions [29].

4. Event Scheduling in discrete-event simulation

In Event Scheduling the basic building block is the event, the model program's code contains event routines that are waiting to be executed. Event routines associated with each event type in a simulation system perform the specific operations or tasks required for that event type when it occurs during the simulation. Simulation moves from one event to another executing their corresponding event routine [30].

In a "discrete-event model" the state variables don't change till a defined point called event times [27].

4.1 System state variables

- **Entities & Attributes** entity is an object its whose can be static or dynamic, while Attribute are local values used by the entity in our case they are
  Entities packet, Event.
  Attributes Time, Type, ID, etc...
- **Resources** is an entity that provides service to one or multiple dynamic entities at a time, this last one can request one or more resources, if it's approved, she can use this resource and when she finishes release it, or if rejected, the entity can join a queue. In our case, it's the s Server.
- **Lists** are the queue
- **Delay** is an indefinite duration that is caused by some combination of system conditions.

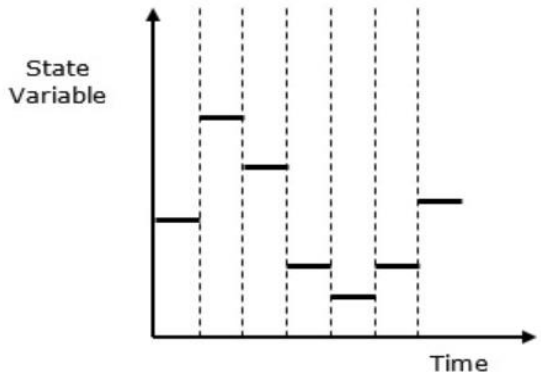


Figure 6. *Representation of state variable based on time in discrete event simulation* [27]

5.Method used to collect steady-state observation data during a simulation

The Mean Batch Method involves running a single, long simulation that is divided into different parts an initial transient period, and several batches. Each batch is treated as a separate simulation run, while no observations are made during the transient period, which is like a warm-up phase. Figure 7.10 provides an example of this process. The goal is to estimate confidence in the results. One advantage is that only the transient period needs to be accounted for and removed when recording observations. However, a drawback is that the

batches may not be fully independent, as high values tend to follow high values and low values tend to follow low values. This can affect the accuracy of confidence estimation if the batch sizes are not large enough [31].

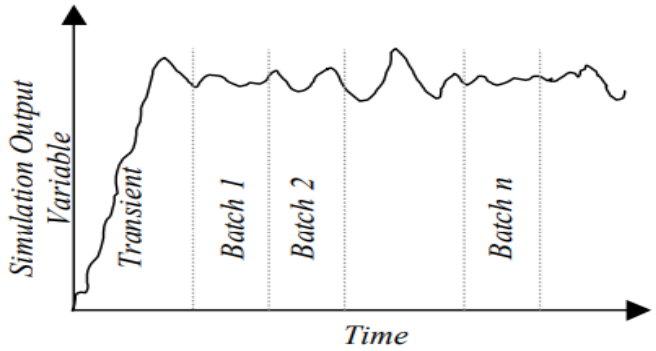


***Figure 7**. The Subinterval Method or The Method of Batch Means.* [31]

6. Model verification

Model verification techniques ensure the correctness and reliability of a simulation model. We employ the following techniques:

- Top-down, modular design: Breaking the model into smaller components for easier verification. Assertions can be used to check specific conditions or equations during the simulation.
- Tracing techniques: Using print statements or a debugger to track execution flow and inspect variable values, aiding in error identification.
- Consistency tests: Comparing outputs of similar inputs to ensure consistency in results.
- Seed independence: Verifying that the initial state of the random number generator doesn't significantly impact simulation conclusions, avoiding bias or dependency on a specific starting point.

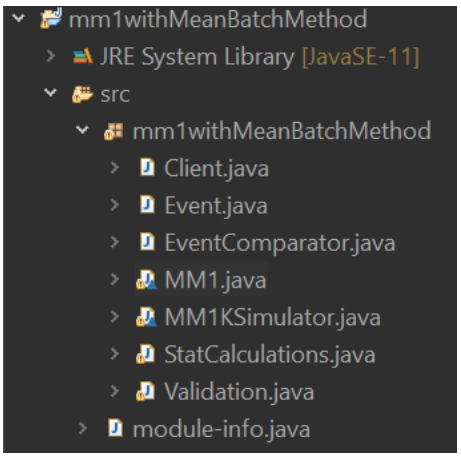


***Figure 8**: Simulator Project Structure*

7. Model validation

Model validation is crucial for ensuring that a simulation accurately represents the real system. Key aspects to consider include assumptions, input parameters, output values, and conclusions. Theoretical results can be useful for comparing simplified systems with simulation outcomes, but they should be complemented by measurements and expert intuition. Fully validating a model may be impractical, so validation is typically demonstrated in select cases to build confidence. One technique we use is comparing simulated results with analytical solutions, which helps assess the accuracy and reliability of the model. While slight differences may occur due to system stochasticity and the finite number of simulations runs, the general trend and magnitude of response time values should match between the two approaches.

Conclusion

In this chapter, we've laid the groundwork for our exploration of simulation modeling. We began by justifying the use of simulation as a crucial method for our study, followed by an overview of the steps involved in conducting a simulation study. We delved into discrete event simulation, emphasizing the importance of event scheduling and system state variables. Additionally, we discussed methods for collecting steady-state observation data during simulations, which is essential for drawing meaningful conclusions. Model verification and validation processes were also highlighted as critical steps in ensuring the accuracy and reliability of our simulation models. This chapter has equipped us with the necessary concepts and principles to embark on our simulation journey, providing a solid foundation for the practical applications and analyses we'll undertake in the subsequent chapters.

Chapter 3: Implementation of the simulation model

This chapter provides a comprehensive overview of our simulation model for performance evaluation using vacation policies. The model consists of various components that simulate the behavior of the node. We present the pseudo-code for the main program and describe the key events and their interactions within the model. We also explore different simulation models, extensions, and policies for scenarios with finite and infinite buffer capacities, discussing their advantages, limitations, and trade-offs. Additionally, we discuss the methods used to collect and analyze simulation results, ensuring accuracy and reliability.

1. Simulation Environment

The simulation model is created and executed using Java programming language in the Eclipse Integrated Development Environment. The development environment is configured on a Windows 10 computer with 16 GB of RAM. The Javac compiler is used for the project. Additional details about the version of Eclipse, Java, and the compiler used are listed below:

Eclipse IDE for Enterprise Java and Web Developers (includes Incubating components)
Version: 2022-06 (4.24.0)
Build id: 20220609-1112
OS: Windows 10, v.10.0, x86_64 / win32
Java vendor: Oracle Corporation
Java runtime version : 13.0.1+9
Java version: 13.0.1

- **Justification of the programming language choice**

We chose Java and Eclipse for several reasons, including its object-oriented programming nature that aligns well with simulation modeling concepts, the availability of rich Java libraries and frameworks for simulation development, and the robust debugging and testing capabilities offered by Eclipse, providing a powerful and flexible environment for building robust and scalable simulation models.

2. Basic Simulation Model (M/M/1 Queue)

To lay the groundwork for the subsequent models, we begin by establishing a basic model. This model serves as the building block upon which more complex and advanced models will be developed. It forms the fundamental framework for the rest of the simulation models and sets the stage for further exploration and analysis.

The node model represents a network node that follows the characteristics of the M/M/1 queue. It receives packets from an external source. The packets are then processed by the node's single server with an exponentially distributed service time. The node disposes of a buffer to hold incoming packets when the server is busy, and the queue length and waiting times of the packets can be analyzed to evaluate the performance of the node and the overall system.

2.1 Data Structures

- **LinkedList** (Packet queue): Used to store and manage packets waiting to be processed by the server. The queue follows specific policies based on the simulated model, the

following code illustrates the implementation of the queue structure in the simulation model taking into account the case where the queue size has to be finite.

```
Queue<Client> clients_queue;
clients_queue = new LinkedList<Client>() {
        @Override
         public boolean add(Client client) {
            if (size() < K)
               return super.add(client);
            return false;
        }
};
```

- **Priority queues** (Event queue): Arrange elements based on their priority values, which in our case is time. Events are inserted in ascending order of priority, where lower priority values have higher priority. If two events have the same priority, they are served according to their order in the queue. When dequeuing an event, it is removed from the queue and returned to the simulation. The event comparator class ensures the structure above since it's responsible for event ordering within the queue.

```
PriorityQueue<Event> events_queue ;
events_queue= new PriorityQueue<Event>(100, new EventComparator() );
```

2.2 Assumptions for the basic model

We assume that:

- The sensor node can only switch between the idle and the busy state.
- Single Server The system assumes the presence of a single server responsible for serving packets. The server can handle one packet at a time and follows a first-come-first-serve discipline.
- Service rate and arrival rate follows an exponential distribution
- Single Packet Queue The system utilizes a single packet queue to hold incoming packets or packets awaiting service. Packets enter the queue upon arrival and are served in the order of their arrival.
- Deterministic Energy Consumption The energy consumption values for different system states and events are assumed to be deterministic and fixed. This simplifies the energy calculation process by assuming known and constant energy consumption values.
- Fixed Simulation Time The simulation is designed to run for a fixed time duration of 10,000 units. This assumption limits the simulation to a specific time interval and does not consider dynamic simulation termination based on certain conditions or convergence criteria.

2.3. Simulation parameters

λ (Lambda): The arrival rate parameter, ensures that the arrival of packets into the system per unit of ime, follows a passion process.

μ (Mu): Service rate parameters, controlling how quickly the server can process packets per unit of time, following an exponential distribution.

ECI (Energy Consumption during Idle State): Energy consumed by the server when idle and not serving any packets or events.

ECb (Energy Consumption during Busy State)**:** Energy consumed by the server when actively serving a packet or event.

ECTx (Energy Consumption per Packet in Queue): Energy consumed for holding each packet in the queue, accounting for energy wastage due to packet buffering.

ECs (Energy Wasted during State Transitions): Energy wasted as the server transitions between idle and busy states, capturing energy inefficiencies during state changes.

K (Buffer Size): Maximum capacity of the queue or buffer, determining the maximum number of packets that can be accommodated before new arrivals are rejected.

N (Threshold): Number of packets required in the queue to trigger the server to start processing, transitioning from idle to busy state.

Number of Batches: Number of iterations or separate simulation runs with their own packets and events.

Simulation Time: Duration for which the simulation runs.

2.4 Simulation Clock and Time-Advancing Mechanism

The simulation clock represents the current time in the simulation (Tnow). It is advanced by processing events and determining the next event based on its time. The time-advancing mechanism ensures that events are processed in chronological order, and the simulation progresses according to the event timeline.

2.5 The process of generating arrival and service rates

Both arrival rate and service rate are generated using the exponential distribution, for the traffic rate, the passion process can be obtained by setting the outcomes of the exponential distribution process for the inter-arrivals. In the other hand, for the service rate of the server, the values are assigned directly. Here's a detailed explanation of exponential distribution

Its formula is given by $P(T \leq t) = 1-e^{-\lambda t}$

We will be using this formula to generate random periods of time

The first step is to extract the value of time t from the formula:

$R = 1- e^{-\lambda t}$

$e^{-\lambda t} = 1-R$

$-\lambda t = \log(1-R)$

$t = \log(1-R) / (-\lambda)$

Note: that the same process is done with the parameter **μ** (mu) instead of **λ** (lambda) to generate the service rate.

the obtained values “t” are assigned to the inter-arrivals and the service time of the packets at each iteration.

The following routine illustrate the implementation of process above in the simulation model:

Routine 3-1 generating arrival and service rates

```
double exponential_distribution (double parameter) {
    Random rand = new Random();
    double R = rand.nextDouble();
    return  Math.log(1-R)/(-parameter);
  }
```

- The said parameter, represent both parameters (λ) for the arrival time or (μ) for service rate.
- This function generates a random number **R** using the **next Double ()** method of the **Random** class. This method returns a random double value between 0 (inclusive) and 1 (exclusive).
- The expression **Math.log (1 - R)** calculates the natural logarithm of **1 - R**.
- The function returns the calculated value **Math.log (1 - R) / (-parameter)**, which is a random number drawn from the exponential distribution with the specified rate parameter.

By using this function, we were able to generate random numbers that follow an exponential distribution with a given average rate.

Now, let's discuss the concept of the seed in random number generation

The seed is a starting point or initial value used by a random number generator algorithm to generate a sequence of random numbers. By setting the seed, we can reproduce the same sequence of random numbers each time the program runs.

However, in our implementation, the seed is not explicitly set for the Random class. As a result, the default behavior of the Random class is used, which sets the seed based on the current system time. This means that each time the program runs, a different sequence of random numbers will be generated.

2.6 Simulation events

In our simulator, we encounter three different types of events: arrival, start service, and departure. Each of these events corresponds to a specific procedure that outlines how the event impacts the state variables, progresses time, and generates additional events. Apart from these procedures, we also have initialization and the gathering of statistics.

- **Process arrival**

The process arrival event represents the arrival of a new packet to the system. It creates a new packet with a service time following an exponential distribution (μ) and adds it to the packet queue. If the server is idle, it schedules the start of service at the current time (Tnow). It also schedules the next arrival event at Tnow + exponential distribution(λ).

Routine 3-2 arrival

```
Process arrival () {
        create new Packet given service time as exponential distribution(μ);
        add the packet to the queue;
        if (server is idle) {
        schedule start service at Tnow;
        }
        schedule arrival at Tnow + exponential distribution(λ));
}
```

- **process start service**

This event occurs when the server begins serving a packet. It changes the server state to busy, retrieves the next packet from the queue using the First-In-First-Out (FIFO) policy, records the time at which service starts (Tnow), and schedules the departure of the packet at Tnow + Packet.service_time. This event represents the initiation of processing for a packet by the server.

Routine 3-3 start service

```
Packet process start service () {
        Server state = busy;
        Get the packet from the queue; (FIFO)
        Packet time of starting service = Tnow;
        schedule departure at Tnow + Packet.service_time;
        return packet;
}
```

- **Process Departure**

The departure signifies the completion of service for a packet. It updates the server state to idle, increments the count of packets served, removes the processed packet from the system, and checks if there are any remaining packets in the queue. If packets are waiting, this event schedules the start of service for the next packet at the current time (Tnow). It represents the moment when a packet finish being serviced and leaves the system.

Routine 3-4 departure

```
Process departure (packet) {
        Server state = idle;
        packet served ++;
       delete packet;
       if (packets queue size > 0)
        schedule start service at Tnow;
}
```

- ## Main program

The main program controls the simulation and consists of the following steps

- Outer Loop Iterate over variations (λ or N) to analyze different scenarios.
- Inner Loop Simulate multiple batches of the system.

- Initialization Set up the initial state of the system.
- Batch Simulation Loop Simulate a certain number of packets/customers are served.
- Get Event and Time Retrieve the next event and its time.
- Event Processing Call the appropriate event function based on the event type.
- Delete Event Remove the processed event from the queue.
- Report Statistics Summarize and display the collected data.
- Statistical Calculations and Results Display Analyze and present batch means and results.

Routine 3-5 main program

```
main () {
        for (variation of λ or N) {
                for (each batch) {
                        initialization ();
                        while (number of served in each batch < 10000) {
                                event = get event from the event queue;
                                Tnow = event time;
                                switch (event type) {
                                case arrival process arrival ();
                                case start service Packet = process start service ();
                                case departure process departure(Packet);
                          }
                          Delete event;
                }
                Report statistics ();
        }
        Stat Calculations of batch means & results display (batch Means);
}
```

- **Initialization**

the initial setup phase of a simulation where the system's state is configured. The server is set to idle, and the packet and events queues are initialized. The current time is set to 0, and statistic variables are initialized. A new packet ID is assigned, and the first arrival event is scheduled to start the simulation. Initialization prepares the system's initial conditions and data structures for the simulation.

Routine 3-6 initialization

```
Initialization () {
        server state = idle;
        Initialize packet queue;
        Initialize events queue;
        Tnow = 0;
        Initialize Statistic Variable;
        New packet id = 0;
```

```
        schedule arrival at Tnow;
        }
}
```

- **Report Statistics**

Reporting statistics involves collecting and presenting performance metrics (measurements) that offer insights into the system's behavior. It includes displaying the calculated results in a readable format (numerical values).

3. Performance metrics

Here are some commonly used measurements [27]:
Note: that in the following, we will be explaining how we calculated performance metrics for both analytic and simulation model.
And we will be using Traffic Intensity (load factor) ρ which is calculated by $\rho = \lambda/\mu$.

4.1 Overall Packets served

It represents the total number of packets served in the entire simulation, which is the sum of Packets served in each batch. This metric provides an overall count of the Packets that have been successfully served.

```
        Overall Packets served += Packets served in each batch;
```

3.2 Mean waiting time

It is the average time a Packet spends waiting in the queue before it starts receiving service. The mean waiting time is calculated by dividing the cumulative waiting time of all served Packets by the number of Packets served in each batch.

```
Mean waiting time += packet.t start service - packet.t arrival;
Batch Means of mean waiting time[i]=mean waiting for time/packets served in each batch;
```

The batch Means of mean waiting time array stores the mean waiting time for each batch, assuming "i" represents the current batch index.
The mean waiting time is also calculated analytically using the formula $\rho/(\mu*(1-\rho))$.

3.3 Mean response time

It is the average time a Packet spends in the system from the moment it arrives until it departs, including both waiting time and service time. The mean response time can be calculated in two ways
Analytical calculation: The mean response time can be analytically calculated using the formula $1/(\mu-\lambda) + 1/\mu$. This formula takes into account the arrival rate (λ) and the service rate (μ) of the system.
Simulation calculation: The mean response time can also be calculated by the simulation model. This involves summing up the response time for each served packet, which is calculated as the difference between the current time (Tnow) and the arrival time of the packet

(packet arrival time). The total response time is then divided by the number of packets served in each batch.

```
Mean response time += Tnow – packet arrival time;
Batch Means of mean response time [i]= mean response time/packets served in each batch;
```

3.4 Throughput

It represents the average number of Packets served per unit of time. It is calculated by dividing the number of Packets served in each batch by the total simulation time (Tnow). analytically is equal to the arrival rate lambda and by simulation:

```
throughput = Packets served in each batch / Tnow;
```

3.5 Total Energy Consumption

It represents the total energy consumed by the system during the simulation. The formula given in calculates the total energy consumption by considering different components such as idle energy consumption (ECI), busy energy consumption (ECb), energy wasted during state transitions (ECs), and energy consumption for holding each packet in the buffer (ECTx). The formula accounts for the probability of being in the idle state (vacation prob) and the average number of cycles (Nc)

```
EC = vacation prob * ECI + ((1 – vacation prob) * ECb) + (ECs/Nc) + Q * ECTx;
```

3.6 Blockage

represents the proportion of time the server is busy (in a saturated state) compared to the total simulation time. It indicates the level of congestion or utilization of the server. The blockage can be calculated using the formula

Simulation calculation The simulation-based blockage can be calculated by dividing the saturation time (the time when the server is busy) by the total simulation time.

```
Within the arrival event
        if (packets queue size() == K ) {
        start of saturation time = Tnow;
        saturated = true;}
within the departure event
        if (packets_queue.size() < K && saturated == true) {
                saturation_time += Tnow – start_of_saturation_time;
                saturated = false;}
at the end of the simulation
        batchMeans_of_blockage[i] = saturation_time/Tnow ;
```

Analytical calculation the analytical blockage can be calculated using the formula
((1-ρ) * ρ^K) / (1 – ρ^(K+1)), where ρ is the traffic intensity and K is the buffer size.

3.7 Loss Rate

The loss rate represents the rate of rejected Packets in each batch, indicating the proportion of Packets that were not served due to queue limitations or other constraints. The loss rate can be calculated in two ways

Analytical calculation The analytical loss rate can be calculated using the formula λ * blocking_prob. The blocking probability (blocking_prob) represents the probability that a Packet is blocked or rejected due to queue limitations.

Simulation calculation The simulation-based loss rate can be calculated by dividing the number of rejected Packets by the total number of Packets that arrived in each batch.

```
Within the arrival event
        Number of arrived packets ++;
          if (packet size () > K) then don't add it to the packets queue
          // The packet was rejected
          Number of rejected packets ++;
At the end of the simulation
    Loss rate = number rejected Packet / Packets arrived in each batch;
```

3.8 Probability of being in the idle state (PI)

The probability of being in the idle state calculates the likelihood of the server being idle. It is determined by measuring the total time the server is idle (idle Time) and dividing it by the total simulation time.

```
within the Initialization
    idle Time=0; idle prob=0; idle start = Tnow;
      if (server was idle)
      add (Tnow – idle start) to idle Time;
      Then set the server busy;
      if (server was busy)
      idle start = Tnow;
      then set the server idle;
At the end of the simulation idle prob = idle Time / Tnow;
```

Analytically**:** for an infinite buffer, the probability of being in the idle state can be calculated as (1-ρ), where ρ is the traffic intensity.

For a finite buffer of size K, the probability of being in the idle state can be calculated as (1-ρ) / (1-ρ^K), where ρ is the traffic intensity and K is the buffer size.

3.9 Average number of packets in the queue (Q)

It represents the average number of packets or Packets present in the queue. It is calculated in the simulation by dividing the cumulative number of packets in the queue by the total number of events processed. Analytically calculated by the formula (ρ / (1-ρ)) – (ρ * (1 + K* ρ^ K) / (1- ρ^ K+1), and in our simulation model:

```
Within each event we extract and accumulate the number of packets in the
queue:
Number of packets in buffer += packets_queue.size();
Number of events++;
Mean no packets in buffer = number of packets in buffer /number of events;
```

3.10 number of cycles

which represents the count of cycles that occur during the simulation. A cycle in this context refers to the time interval between two consecutive arrivals that find an empty system. It consists of a busy period (BP), during which the server is actively serving customers, followed by an idle period (IP), during which the system remains empty.

To calculate the average number of cycles (Nc), the code divides the total number of cycles by the total simulation time (Tnow).

```
Within the departure event
    If (packets queue is Empty ()) {
        server state = idle;
          number of cycles ++;
    Nc = number of cycles / Tnow;
```

By measuring the number of cycles and normalizing it by the simulation time, we can get an understanding of how frequently the system transitions between busy and idle states. This metric can provide insights into the utilization and efficiency of the server and help analyze the behavior of the queuing system.

5. Methods to collect the simulation results

We have been using the mean batch method as a method for gathering simulation results by averaging several simulation runs.

- **Compute means for each batch**

The mean batch method is used to collect simulation results by averaging multiple simulation runs, increasing the accuracy and reliability of the findings. This approach involves conducting simulations with randomized inputs for each batch, producing a set of outcomes. Performance metrics of interest are gathered during each batch run, and the batch mean is computed by averaging the results. This process is repeated for multiple batches to capture system variability.

- **Compute overall mean**

The overall mean is then calculated by summing the batch means, providing a more accurate depiction of the simulation findings.

- **Calculate the standard deviation**

The standard deviation is also calculated to quantify the variability and uncertainty in the data, offering insights into data distribution and reliability. Various statistical techniques, such as confidence intervals.

- **Calculate the confidence interval**

The confidence interval is a statistical measure that provides an estimate of the range within which the true value of a population parameter is likely to fall.

```
confidence Level = 0.95; // Desired confidence level
Critical Value = 1.96; // For a 95% confidence level
if (confidenceLevel > 0.95) {
        Critical Value = 2.576; // For a 99% confidence level
}
    standard Error = √ (variance/number of batches);
    critical Value = get Critical Value (confidence Level, number of batches - 1);
margin Of Error = critical Value * standard Error;
lower Bound = mean Of Means- margin Of Error;
upper Bound = mean Of Means + margin Of Error;
```

4. Extension of the Simulation Model

This elaboration focuses on exploring different simulation models with vacation policies in the context of M/M/1 queue models

For infinite buffer ∞ and finite buffer (size K)

- For infinite buffer ∞ where there is no buffer size restriction, all arriving packets are accepted regardless of the queue size.
- and a finite buffer (size K) Contrarily, a finite buffer with a specific size, denoted as "K," implies that the maximum number of packets that can be accommodated is limited to K. In this case, packets are accepted as long as the buffer is not full.

4.1 M/M/1 Queue Model with N-Policy

One of the most intuitive methods is to postpone service until a fixed number (N > 1) of customers has accumulated in the queue. Once the server becomes active, it stays active until the system becomes empty again. [32] as it shows in ***Figure 14***

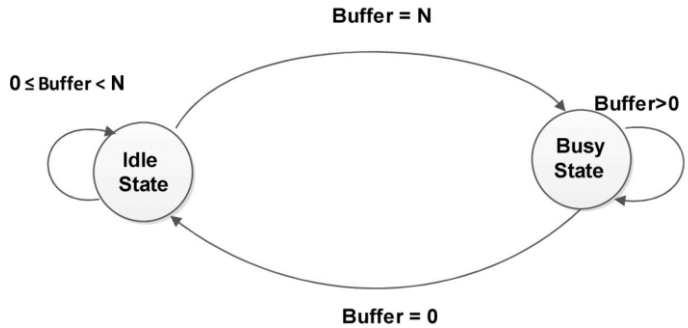


Figure 9: *Transition diagram of a sensor node with N-Policy.* [33]

Implementation

The N-Policy threshold condition is checked within the "process arrival " method

```
if the packets queue. Size () >= N
```

This condition checks if the number of packets in the queue is greater than or equal to the threshold N. If this condition is satisfied, a new event for starting the service is created.

4.2 M/M/1 queue Model with T-Policy (Timer policy)

if no packets arrive during a vacation period, the server extends its vacation and remains idle. This policy introduces the concept of vacations, which can help save energy or reduce costs during periods of low demand.

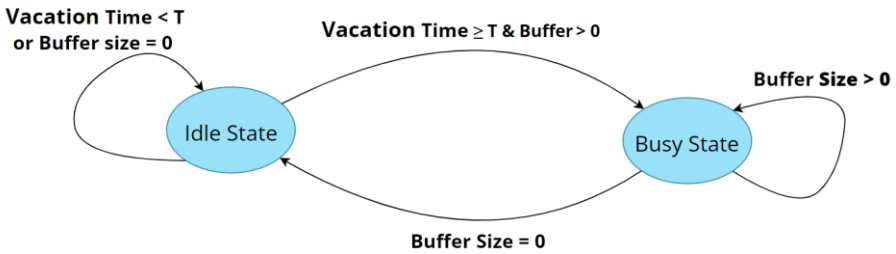


Figure 10: *Transition diagram of a sensor node with T-Policy*

After all the customers are served in the queue exhaustively, the server deactivates and takes at most J vacations of constant time length T repeatedly until at least one customer is found waiting in the queue upon returning from a vacation. If at least one customer presents in the system when the server returns from a vacation, then the server reactivates and requires a startup time before providing the service. On the other hand, if no customers arrive by the end of the Jth vacation, the server remains dormant in the system until at least one customer arrives [34].

Implementation

```
Setting up a variable T (max Time of a vacation)
Initialize the vacation timer at Tnow
Within the arrival process
        If (the vacation timer is over)
                start service;
Within the departure
        if (queue becomes empty)
                start a new vacation;
```

4.3 M/M/1 queue Model with min (N, T) Policy

The addition of timer T avoids endless waiting that may happen in some applications with a sparse arrival scenario. For example, in habitat monitoring, there's no traffic on the network for long periods, which are followed by short periods of network traffic, referred to as bursts. If the total number of arriving packets in a specific burst cannot reach N, then T -policy would save those stale queued packets and trigger the radio server to transmit them in due course. [35]

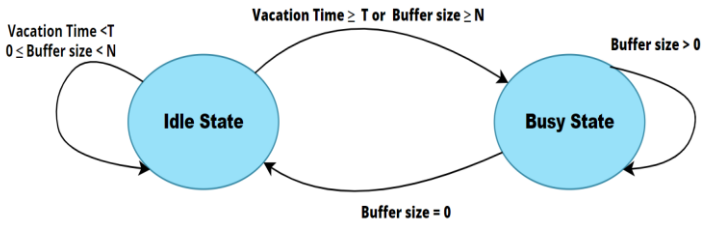


Figure 11: *Transition diagram of a sensor node with min (N, T) Policy*

Implementation

By incorporating the 'OR' operation in the condition check, we ensure that the server is activated when either of the policies is satisfied

```
if (server state = idle & (vacation time >= T || packet queue size () >= N))
        then        start service;
```

4.4 M/M/1 Queue Model with D-Policy

The server is turned off at the end of a busy period and turned on when the cumulative amount of work first exceeds some fixed value D [36]

This policy helps manage the workload of the server and ensures timely service delivery based on the accumulated work.

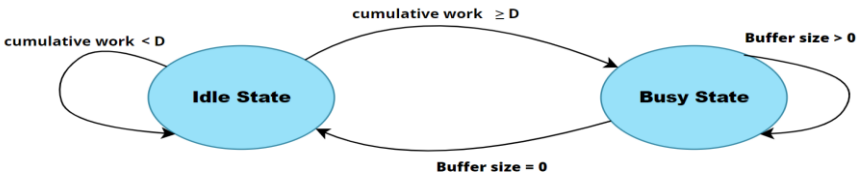


Figure 12: *Transition diagram of a sensor node with D-Policy*

4.5 M/M/1 queue Model with Hybrid Policy

the sensor node can switch from idle to busy state either

_ At the instant of the Nth packet arrival.

_ Or at the end of a vacation period which is an exponentially distributed random duration with the parameter ζ, even if the number of packets in the buffer is less than N. [33]

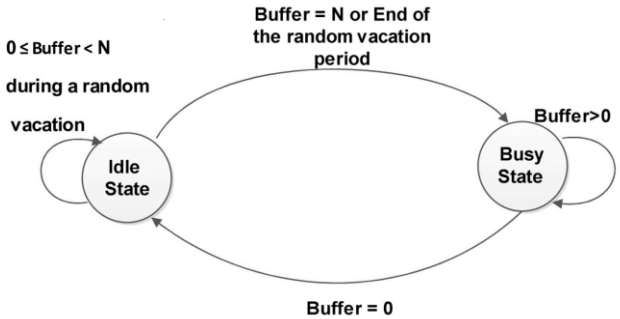


Figure 13: *State transition diagram of a sensor node with Hybrid Policy.* [33]

Implementation

After introducing the parameter zeta, we apply it to generate random vacation times as follows

Vacation time ≥ exponential distribution(ζ)

4.6 Implementation of delayed idle state

To simulate the delayed idle state in a queueing system, an extended version of the M/M/1 queue model is utilized. The implementation involves maintaining an additional state variable to track the server's state (idle, busy, or delayed idle), and the Θ parameter.

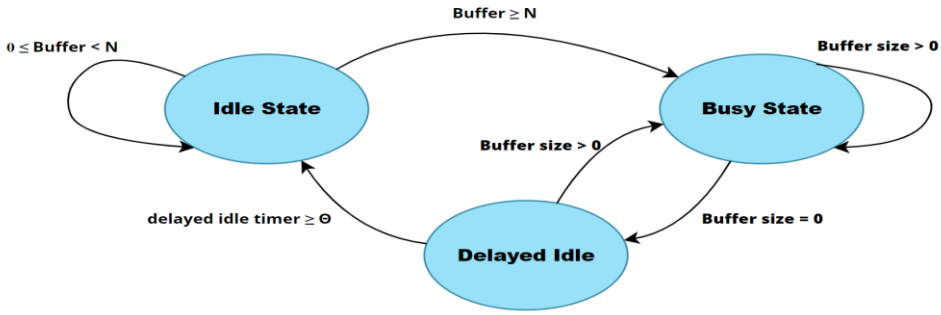


***Figure 14**: Transition diagram of a sensor node with Delayed Idle server state and N-Policy.*

```
if (packet queue is Empty () then   Server state= is delayed idle;
If (Server state = delayed idle) {
        if (Tnow – delayed idle start >= Θ)
        then     server = Server state idle;
```

By simulating the idle delayed state and analyzing the resulting performance metrics, it becomes possible to evaluate the effectiveness of this extension in improving energy efficiency, reducing overhead, and optimizing system behavior in queueing systems.

Conclusion

In conclusion, this chapter provided a comprehensive overview of implementing a simulation model for evaluating WSN node performance using vacation policies. It covered key aspects such as pseudo-code, simulation events, environment details, data structures, M/M/1 queueing system, service processes, state variables, exponential distribution, main program structure, statistics reporting, and assumptions. This chapter laid the foundation for subsequent analysis and discussions on WSN performance evaluation.

Chapter 4: Experimental Study

Chapter 4 of this report presents a comprehensive examination of the system's behavior and performance through the execution of controlled experiments and the utilization of a simulation model. The primary objective of this chapter is to provide an in-depth analysis of the experimental results and discuss the findings in detail. To evaluate the effectiveness of our simulation model, we compare its numerical outcomes with the corresponding analytical results. Furthermore, we employ visually appealing graphs to effectively present the collected data, leveraging their ability to simplify complex information. Through this rigorous examination, we aim to gain valuable insights into the system's functionality and performance, paving the way for further improvements and optimizations.

1. Experimental Study

the process of conducting controlled experiments using the simulation model to investigate and analyze the behavior and performance of a system. It involves designing and executing a series of experiments by manipulating various input parameters or configurations of the simulation model and observing the corresponding output results.

2. System parameters values

The system parameters values used in the simulation are as follows:

Table 5: *system parameters values used in the simulation*

The capacity of Buffer (K)	10
Queue Threshold ranges from	1 to 9
The mean data arrival rate (λ) ranges from	0,25 to 2
Mean service rate (μ)	2
The mean Vacation rate (ζ) ranges from	0,1 to 0,5
ECi	50
ECd	100
ECb	500
ECTx	5
ECs	300
Number of Batches	1000
Simulation Time	10000

3. Arrival rate Lambda (λ) variation

In this scenario, we will explore the impact of varying lambda values on the metric, assuming fixed values for the other parameters. We will consider a lambda range from 0.1 to 1.9 and μ=2. By analyzing this range, we can observe how the metric changes with different lambda values.

3.1 M/M/1 with infinite buffer provided by our model beside analytical results

Table 6: *Performance Results from M/M/1 Simulation model vs Analytic Equations for different performance metrics.*

λ	mean waiting time	analytic mean waiting time	throughput	analytic throughput	mean response time	analytic mean response time	Idle Proba	analytic Idle Proba
0,1	0,026183901	0,026316	0,099949708	0,1	0,526118162	0,526316	0,947064508	0,95
0,2	0,055460479	0,055556	0,199962301	0,2	0,554692798	0,555556	0,898031479	0,9
0,3	0,087779081	0,088235	0,299574796	0,3	0,588826692	0,588235	0,848515655	0,85
0,4	0,125075249	0,125	0,399550105	0,4	0,624596272	0,625	0,798885354	0,8
0,5	0,167011515	0,166667	0,499364364	0,5	0,665858784	0,666667	0,748817988	0,75
0,6	0,214318668	0,214286	0,598998501	0,6	0,712926926	0,714286	0,699280463	0,7
0,7	0,268635548	0,269231	0,699109571	0,7	0,768523944	0,769231	0,648865185	0,65
0,8	0,333176187	0,333333	0,799485588	0,8	0,83218321	0,833333	0,599163322	0,6
0,9	0,40851953	0,409091	0,898594158	0,9	0,907728994	0,909091	0,549706701	0,55
1	0,499237864	0,5	0,999050752	1	0,999271012	1	0,49966442	0,5
1,1	0,609912824	0,611111	1,098953869	1,1	1,109618533	1,111111	0,449794137	0,45
1,2	0,749080249	0,75	1,198477761	1,2	1,249275589	1,25	0,399439056	0,4
1,3	0,925790993	0,928571	1,298344407	1,3	1,427121985	1,428571	0,349934891	0,35
1,4	1,161668434	1,166667	1,398554108	1,4	1,66441727	1,666667	0,299821631	0,3
1,5	1,498716896	1,5	1,498033502	1,5	1,997433174	2	0,25020406	0,25
1,6	1,998229601	2	1,597449116	1,6	2,494545981	2,5	0,200557652	0,2
1,7	2,809334656	2,833333	1,697973628	1,7	3,339614187	3,333333	0,150409395	0,15
1,8	4,455374757	4,5	1,798016338	1,8	4,94296522	5	0,100452929	0,1
1,9	9,46430069	9,5	1,895983523	1,9	9,787403418	10	0,050575197	0,05

The results obtained from our simulator are very similar to the analytical ones based on the previously mentioned equations. When the difference between the simulated results and the analytical ones is small, often within a few decimal numbers, it suggests that our simulator is providing mostly correct results.

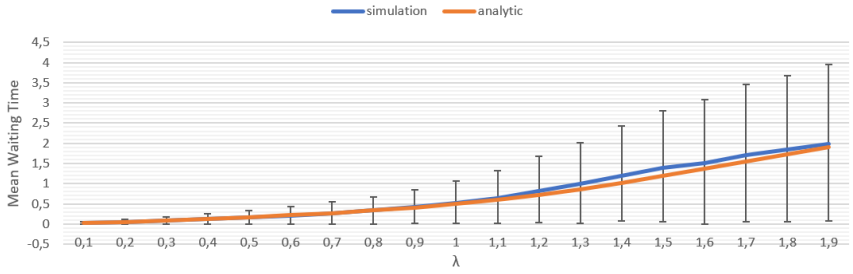
our simulator can accurately reproduce analytical results which is an important validation step in ensuring the reliability of the simulation. It indicates that our simulator is correctly implementing the underlying equations and models, and can be trusted to provide accurate predictions or outputs for similar scenarios.

From now on we will be using graphs instead of tables for results presentation because graphs are visually appealing and capture attention more effectively, they simplify complex data making it easier to understand and interpret trends and patterns, they present information concisely and avoid information overload.

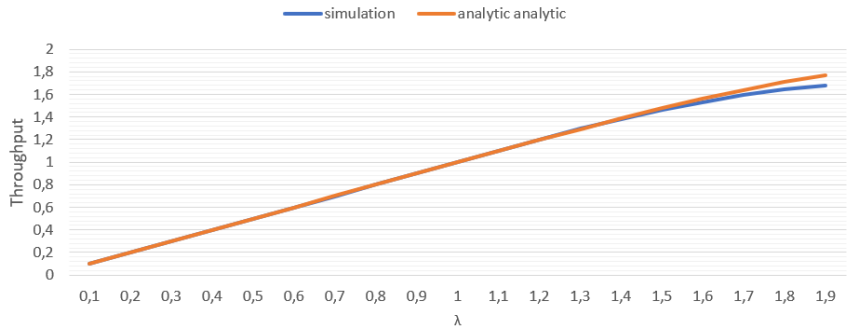
Next, we will be presenting the difference between our simulation results and analytical results for some metrics in the case where the buffer size is limited.

3.2 M/M/1/K queue (finite buffer) K= 10;

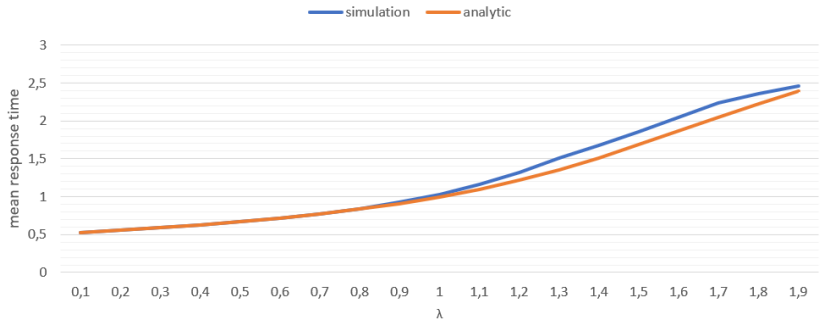
(i)


simulation
analytic
Mean Waiting Time
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
-0,5
0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9
λ

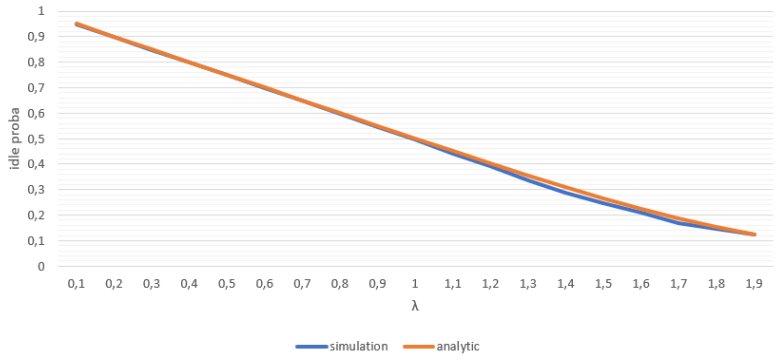
(ii)


simulation
analytic analytic
Throughput
2
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9
λ

(iii)


simulation
analytic
mean response time
3
2,5
2
1,5
1
0,5
0
0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9
λ

(iv)


idle proba
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
0,1 0,2 0,3 0,4 0,5 0,6 0,7 0,8 0,9 1 1,1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9
λ
simulation
analytic

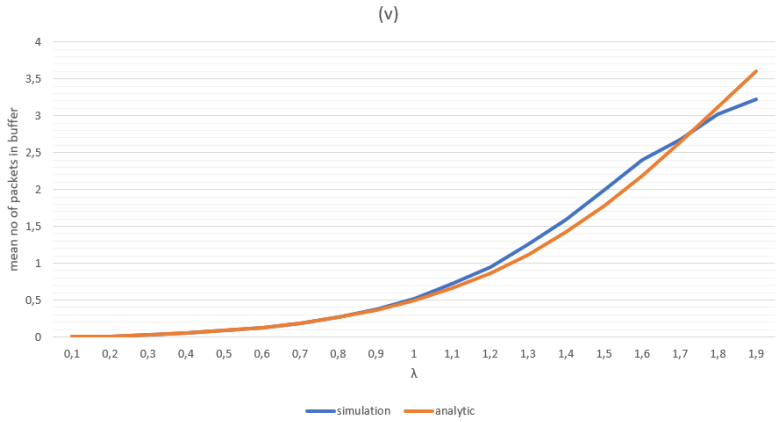


***Figure 15**: Performance Results from M/M/1/K Simulation model and from Analytic Equations (i) Mean Waiting Time (ii) Throughput (iii) Mean Response Time (iv) Idel Probability (v) Mean Number of Customers in the Buffer.*

Again, the values we obtained are very close, with only minor differences in the decimal places. This suggests that our simulation model is effectively capturing the behavior of the M/M/1/K queueing system.
Although there might be slight differences due to the inherent stochasticity in the system and the finite number of simulations runs, the general trend and magnitude of the metrics values match between the two approaches.

It's important to note that small variations can be expected due to the random nature of the system and the finite number of simulation runs. However, as long as the overall trend and magnitude of the results match the analytical values, it indicates that our simulation model is providing a reasonable approximation of the system's behavior.

3.3 Performance evaluation of different policies

Now for each metric, we will be discussing the difference between each policy model and why the results may differentiate from one case to another

- **Mean waiting time**
  - Infinite buffer N =4; D= 1; T= 1;

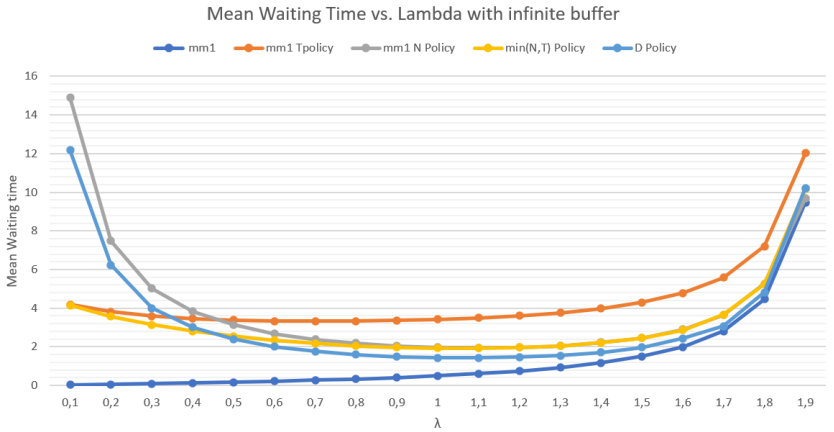


***Figure 16:** Performance Results from M/M/1 Simulation model with different Policies Measuring Mean Waiting Time vs arrival rate lambda.*

- Finite buffer K= 10; $\zeta = 0.2$; //Mean Vacation rate

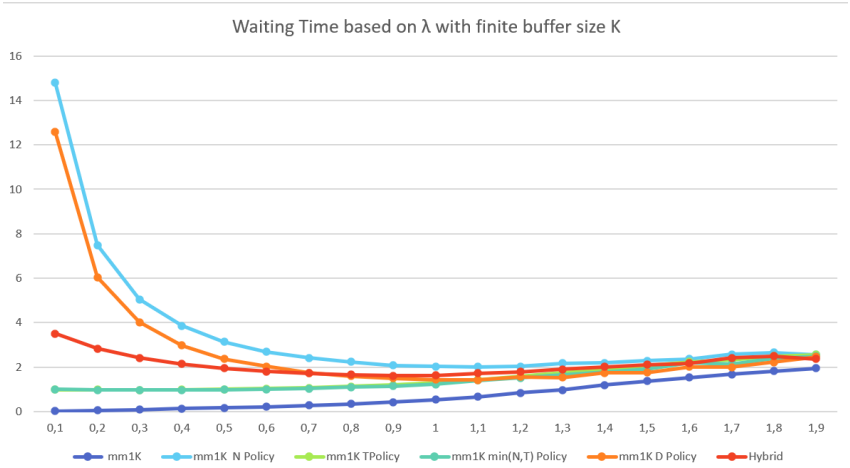


Figure 17: *Performance Results from M/M/1/K Simulation model with different Policies Measuring Mean Waiting Time.*

The waiting time values appear to decrease gradually at first and then start to stabilize or increase slightly. This behavior suggests that the waiting time reaches a minimum point and then starts to level off or slightly increase as the arrival rate continues to rise. This can be attributed to the fact that at higher arrival rates, the server may experience more congestion and longer queues, leading to slightly increased waiting times.

the T Policy, min (N, T) Policy, and Hybrid Policy tend to provide better performance in terms of waiting time compared to the N and D Policies. Among them, the min (N, T) Policy takes into account both capacity limitations and time constraints, making it a favorable choice in many scenarios. However, the specific choice of policy may depend on the specific requirements and priorities of the system under consideration.

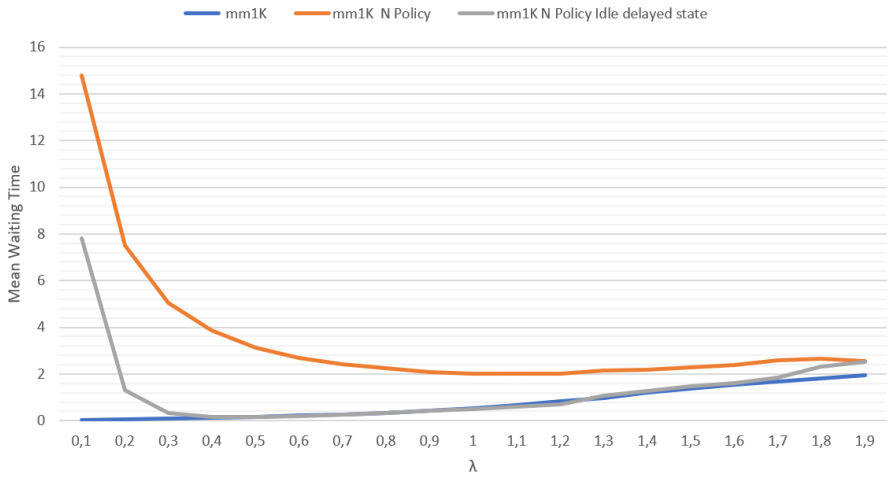


Figure 18*: Performance Results from M/M/1/K Simulation model, M/M/1/K Simulation model with N Policy, and M/M/1/K Simulation model with N Policy and Idle Delayed server state Measuring Mean Waiting Time.*

In the N Policy scenario, when new arrivals are not served immediately, and they must wait in the queue, resulting in longer overall waiting times. But once the arrival rate increases the number of customers in the buffer reaches N faster and the waiting time decreases.
Introducing the idle delayed state in the M/M/1/K system helps reduce the waiting time compared to the N Policy scenario. The idle delayed state means that when the buffer is empty, the server switch to a delayed idle state that is faster to switch from back to the busy state again so one arriving customer triggers the server back to busy and the waiting time becomes less. This reduces the long queuing time of customers and improves the overall waiting time, although it is still higher than the standard M/M/1/K system.

- **Throughput**
  - Finite buffer

Figure 19: *Performance Results from M/M/1/K with different Policies and Delayed Idle state Simulation Models Measuring Throughput vs. arrival rate variation.*

The D policy consistently shows lower throughput values compared to other policies. The Hybrid Policy shows competitive throughput values, often performing well compared to other policies. This indicates that combining different policies can lead to improved system performance and better throughput. The introduction of the idle delayed state reduces the overall throughput as the server spends more time in the delayed idle state, resulting in slower customer service.

- **Mean response time**

-infinite buffer case

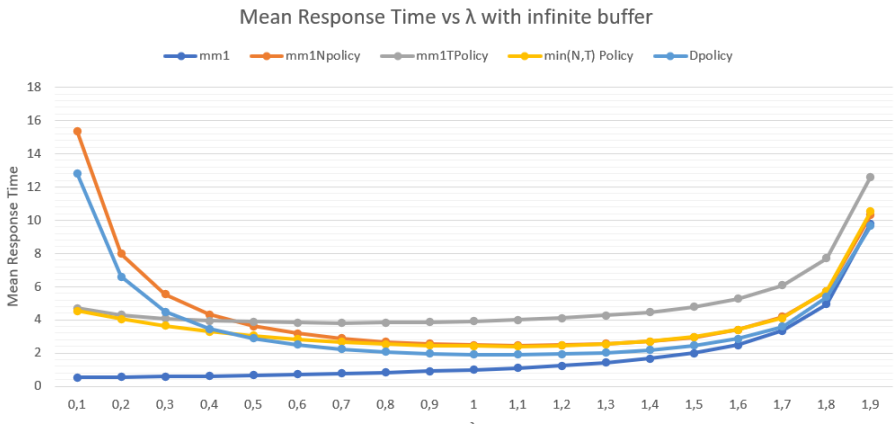


***Figure 20**: Performance Results from M/M/1 with different policies Simulation Models Measuring Mean Response Time vs arrival rate variation.*

In the N Policy the mean response time is significantly higher compared to the M/M/1 system. This is because customers have to wait longer before their service begins, leading to increased overall response times.

The response time first decreases as the arrival rate (λ) increases. This is expected because a higher arrival rate means more customers entering the system, resulting in longer queues and faster threshold reaching

But higher and higher arrival rates lead to more customers entering the system, resulting in longer queues and increased response times.

Mm1 system without policy consistently has the lowest response time. This is because it represents a basic M/M/1 queueing system without any optimization or control mechanisms.

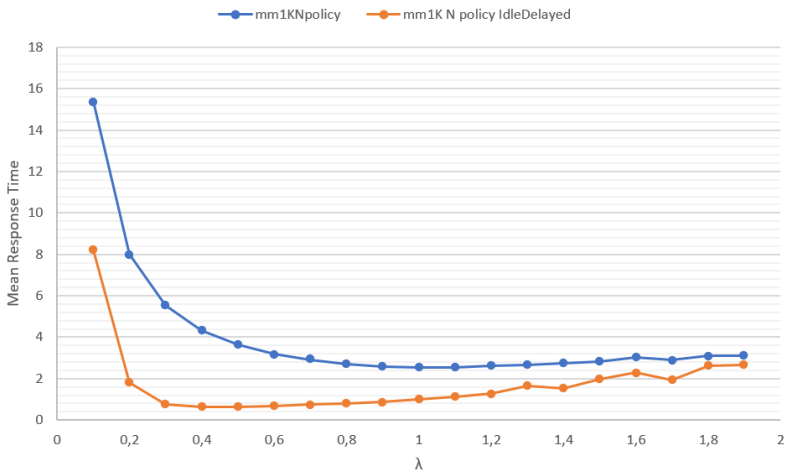


***Figure 21:** Performance Results from M/M/1/K with N Policy and Idle Delayed State Simulation Models Measuring Mean Response Time based on arrival rate variation.*

In the M/M/1/K system with N Policy and Delayed Idle, the mean response time decreases compared to the M/M/1/K with N Policy system.

By incorporating the idle delayed state, the waiting time for customers is minimized, leading to enhanced response times.

The probability that the server is idle

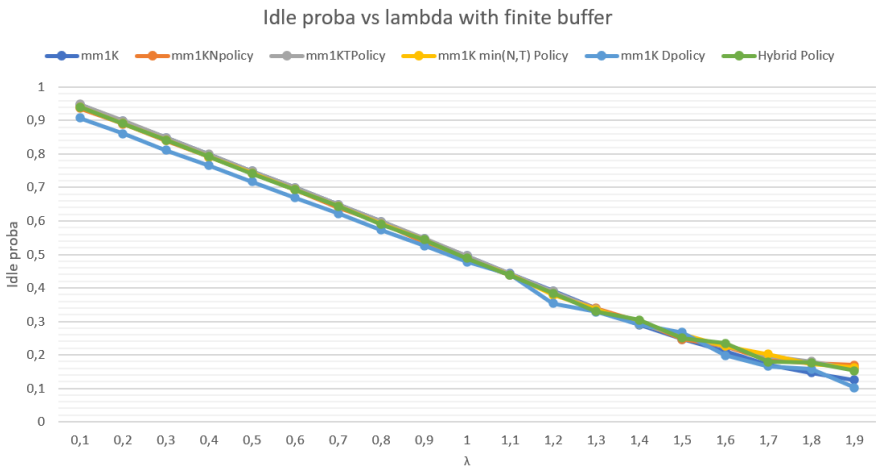


***Figure 22**: Performance Results from M/M/1 with different policies Measuring Idle Probability with finite buffer vs. arrival rate variation.*

The idle probability represents the likelihood of the server being idle, or not serving any customers, at a given time. A lower idle probability indicates that the server is more likely to be busy and actively serving customers, while a higher idle probability suggests that the server is more likely to be idle and waiting for customers.

with Threshold Policies the idle probability is slightly higher compared to the basic system. This is because customers have to wait in the queue before their service begins, which can lead to brief periods of server idleness.

In the M/M/1/K with N Policy, the idle probability increases further compared to the M/M/1/K system. Both the buffer size limitation and the delayed service policy contribute to potential periods of server idleness.

- **Mean number of customers in the buffer**

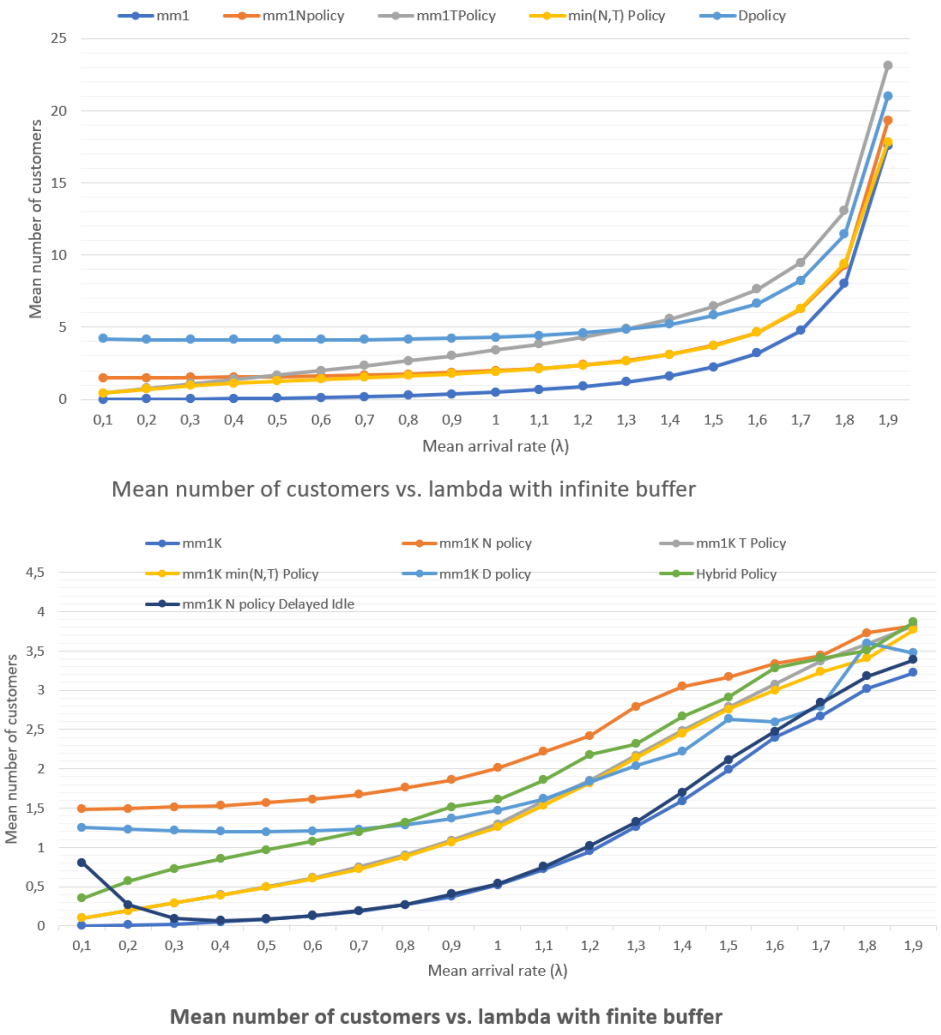


***Figure 23:** Performance Results from M/M/1 with different policies Measuring the Mean number of customers in the buffer for finite and infinite simulation models vs. arrival rate variation.*

with the N Policy system, the mean number of customers in the buffer is higher compared to the M/M/1 system. This is because customers have to wait in the queue before their service

begins, resulting in a larger number of customers in the buffer. Both the buffer size limitation and the delayed service policy contribute to a larger number of customers in the buffer.
In the M/M/1/K system with N Policy and Idle Delayed server state, the mean number of customers in the buffer decreases compared to the M/M/1/K with N Policy system. Incorporating the idle delayed state helps reduce the waiting time for customers in the queue, resulting in a smaller number of customers in the buffer.
The mean number of customers in the buffer values provides insights into the average queue length and the amount of work the server handles in different queuing scenarios, considering factors such as arrival rate, buffer size, and service policies.

Mean number of cycles

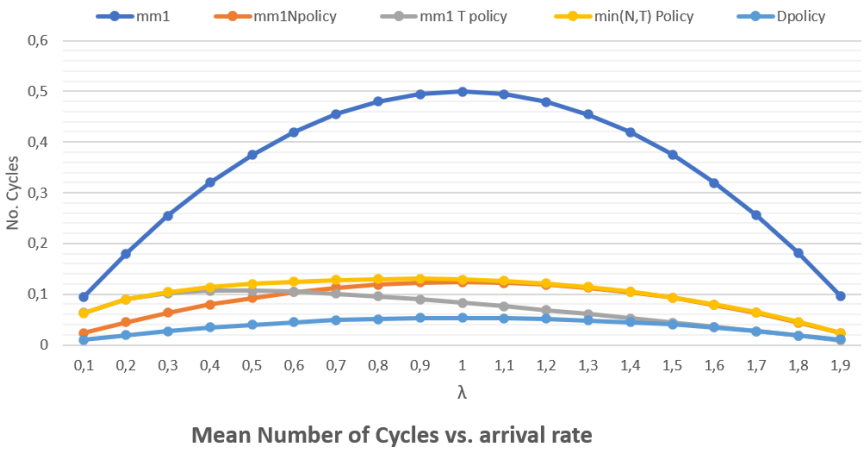


Figure 24: *Performance Results from M/M/1, with Threshold Policies Simulation Models Measuring Mean Cycles number vs. arrival rate variation.*

The Threshold Policies delay the start of service until the number of customers in the queue reaches the threshold N, which reduces the occurrence of cycles.
The mean number of cycles provides insights into the frequency of system utilization and idle periods in different queuing scenarios. It helps analyze the efficiency and utilization of the server and the impact of buffer size and service policies on the system's behavior.

Energy consumption

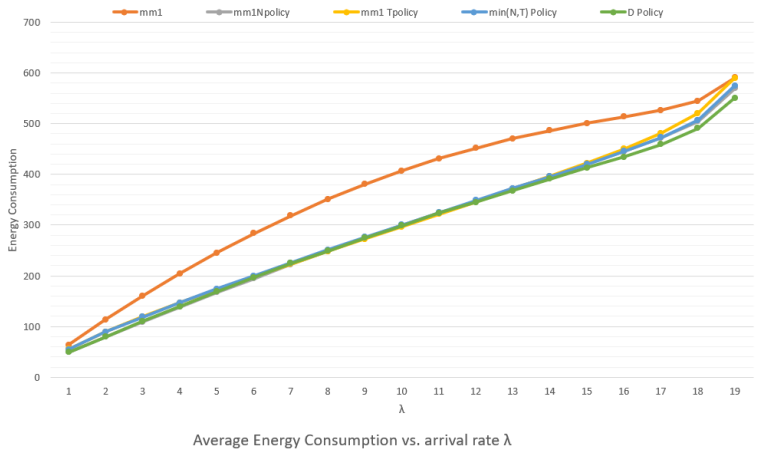


Figure 25*: Performance Results from M/M/1, M/M/1 with N Policy, M/M/1/K, and M/M/1/K with N Policy Simulation models Measuring Total Energy Consumption based on arrival rate variation.*

as the arrival rate increases, energy consumption also increases. This is because a higher arrival rate leads to more frequent arrivals, which require the server to be active for longer periods, resulting in higher energy consumption.
With threshold policies, energy consumption decreases compared to the basic system. Their delays at the start of the service reduce the overall energy consumption.

Overall Packets served

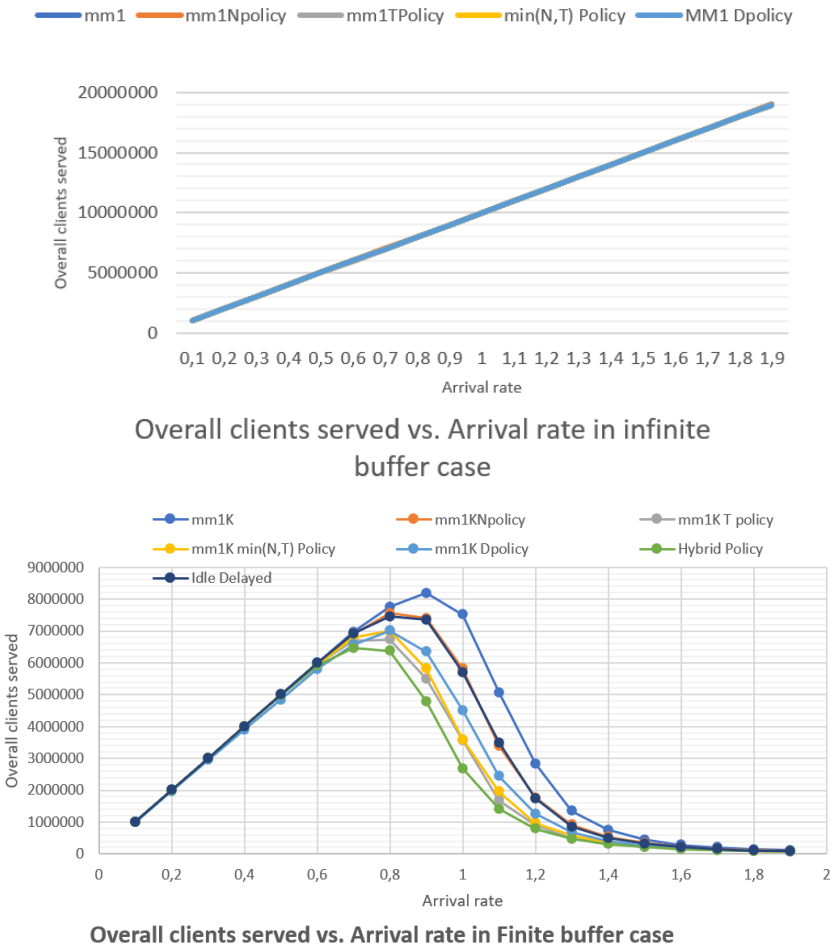


***Figure 26**: Performance Results from M/M/1 and M/M/1/K with different Policies Simulation models giving the number of overall Packets served vs. arrival rate variation.*

As the arrival rate (lambda) increases in the queuing system, the number of Packets served tends to increase. This relationship is intuitive because a higher arrival rate means more customers entering the system, resulting in a greater number of Packets being served.
However, it's important to consider the system capacity, which in this case is limited. The buffer can hold a maximum of 10 packets in the waiting queue at any given time. When the buffer is full, any additional arriving packets will be rejected or blocked from entering the system.
As lambda continues to increase, the buffer will fill up faster, leading to more Packets being rejected or blocked. Consequently, the total number of Packets served by the system will start to decline as the rejection or blocking of Packets becomes more frequent.

Conclusions for lambda variation

the numerical results and discussions demonstrate the following key findings:

- Arrival rate has a significant impact on system performance, with higher arrival rates leading to longer queues, increased waiting times, and decreased Idle probability.
- The buffer size limitation plays a crucial role in controlling system behavior. It helps maintain stability in mean waiting time, response time, and idle state probability.
- The idle delayed state helps reduce waiting times and improves response times compared to the N Policy scenario.
- As the arrival rate increases, the number of Packets served initially increases, but it eventually declines due to the rejection or blocking of Packets when the buffer is full.

These findings highlight the trade-offs between system performance metrics such as waiting times, throughput, energy consumption, and the impact of buffer size limitations and service policies.

4.Threshold N variation

Mean Waiting Time

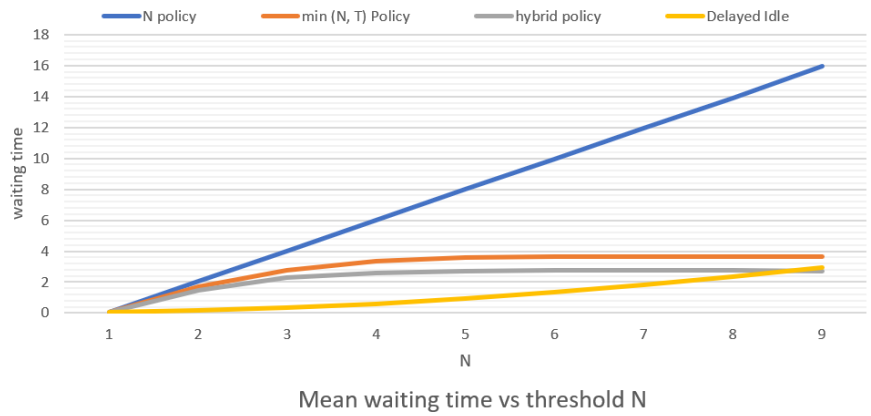


Figure 27: *Performance Results from M/M/1 with different Policies Simulation models giving the Mean Waiting Time vs Thresholds N variation in finite and infinite buffer cases.*

As the threshold (N) increases, the waiting time generally increases across all policies. This is expected, as a higher threshold allows more customers to accumulate in the system before service begins, resulting in longer waiting times.

The Hybrid Policy shows relatively lower waiting times compared to other policies for smaller threshold values (e.g., N = 2 or N = 3) but the N Policy with Delayed Idle server state provides better performance for all threshold values.

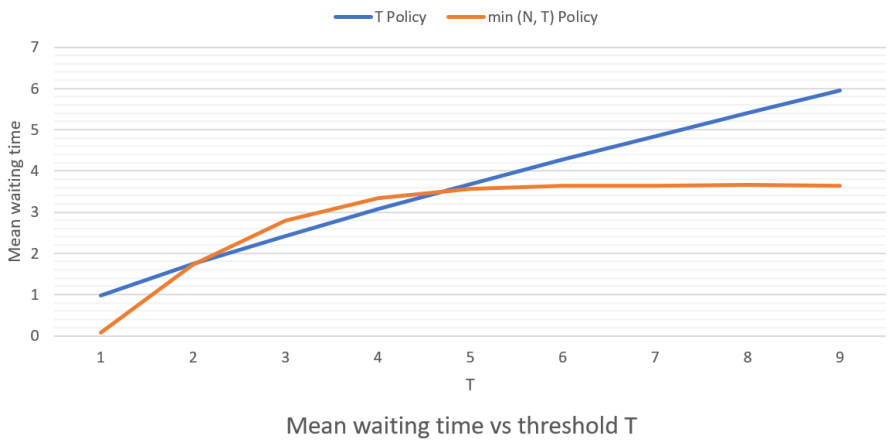


Figure 28: *Performance Results from M/M/1 with T and min (N, T) Policies Simulation models giving the Mean Waiting Time vs Threshold T variation in finite and infinite buffer cases.*

These results show that the mean waiting time varies significantly between the two policies as the threshold value changes. The min (N, T) Policy generally yields lower mean waiting times compared to the T-Policy for most threshold values. This suggests that the min (N, T) Policy is more effective in reducing waiting times in the queue.

Energy consumption

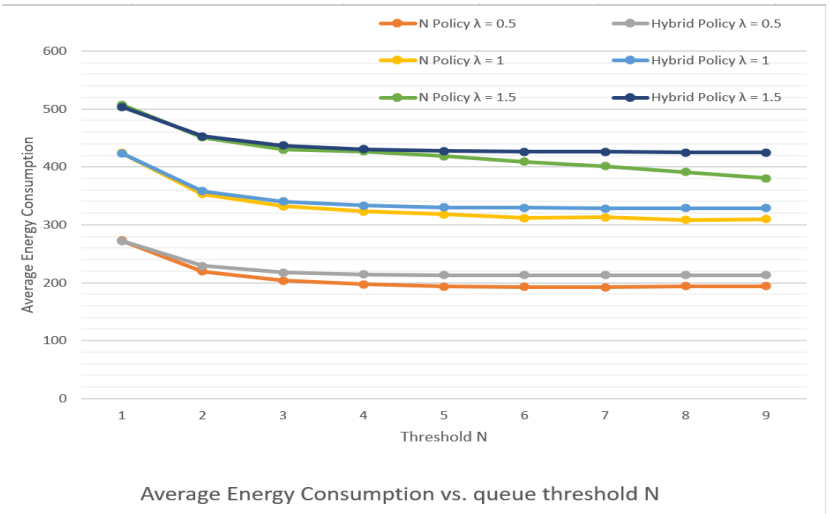


Figure 29: *Performance Results from N Policy and Hybrid Policy Simulation Models Measuring Energy Consumption based on Threshold N.*

We can observe the following:

The Hybrid Policy generally yields lower energy consumption compared to the N Policy for most threshold values and arrival rates. This suggests that the Hybrid Policy is more effective in reducing energy consumption in the system.

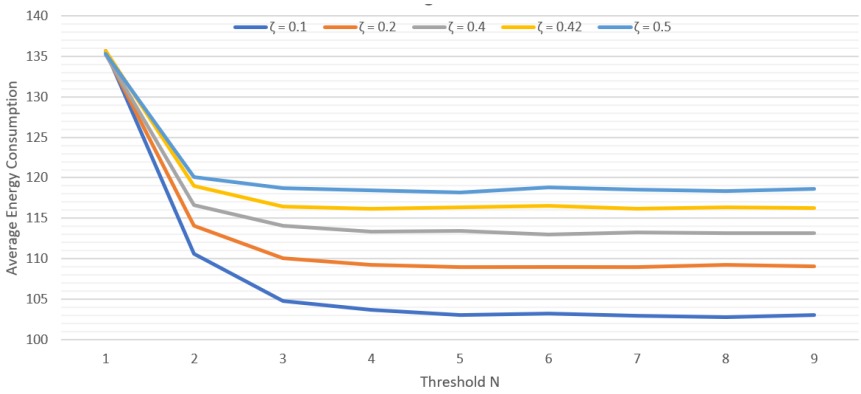


Figure 30: *Performance Results from Hybrid Policy Simulation Model Measuring Energy Consumption vs. Threshold N and mean vacation rate ζ Variation.*

Higher values of the mean vacation rate ζ indicate more frequent vacations taken by the server, the server undergoes more transitions between the idle and busy states. Each transition requires activation energy, resulting in higher overall energy consumption. This switching energy contributes to the increase in average energy consumption with a higher mean vacation rate.

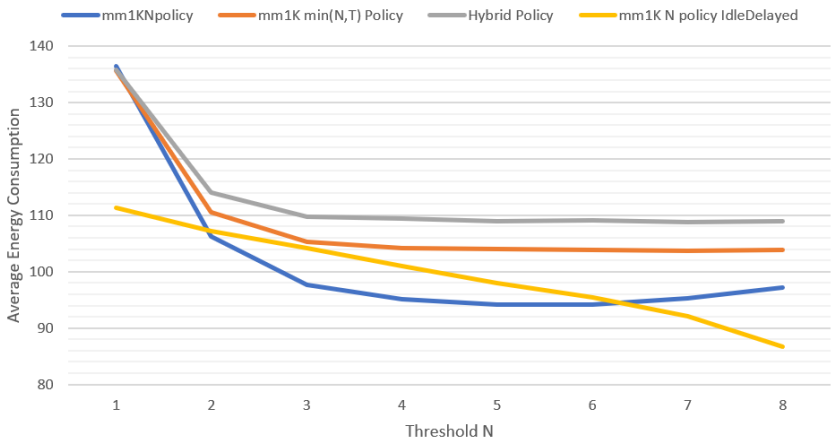


Figure 31: *Performance Results from M/M/1 with different Policies Simulation models giving the Mean Energy Consumption vs Thresholds N variation in finite and infinite buffer cases.*

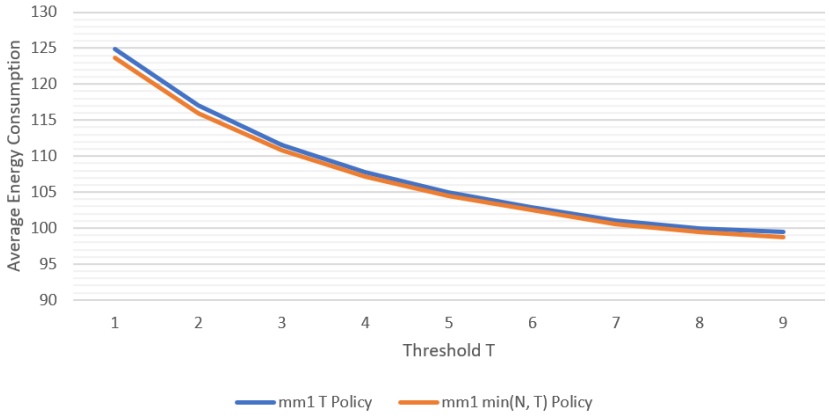


Figure 32: *Performance Results from T Policy Simulation Model Measuring Energy Consumption vs. Threshold T.*

Increasing the threshold value tends to reduce the average energy consumption in the considered policies. It indicates that allowing more work to accumulate in the queue before the server becomes active can lead to energy savings. However, the specific impact of threshold values on energy consumption may vary depending on the policy used.

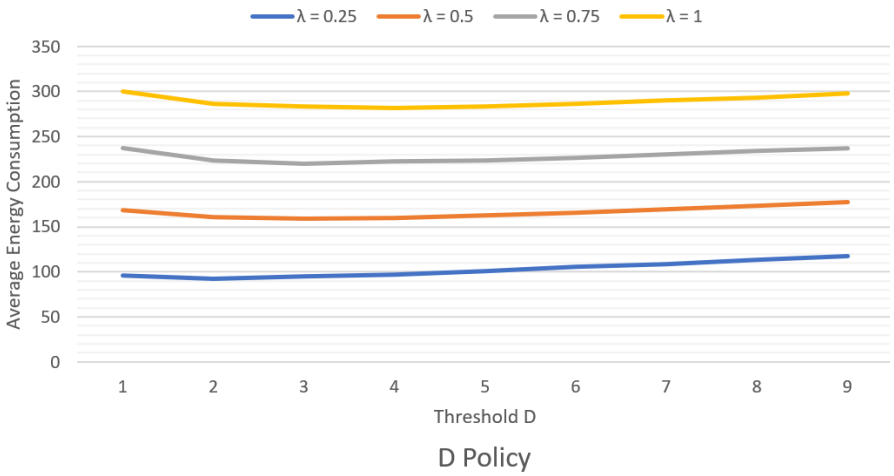


Figure 33: *Performance Results from D Policy Simulation Model Measuring Energy Consumption vs. Threshold D.*

As D increases, it takes longer for the workload to accumulate to that level, resulting in longer idle periods for the server. During these idle periods, the server consumes energy at the idle state energy consumption rate (ECI). Each transition from the idle state to the busy state or vice versa incurs an energy cost (ECs). Since increasing the threshold value D leads to longer idle periods and more transitions between states, it results in higher energy consumption.

5.Service rate mu (μ) variation

Mean Waiting Time

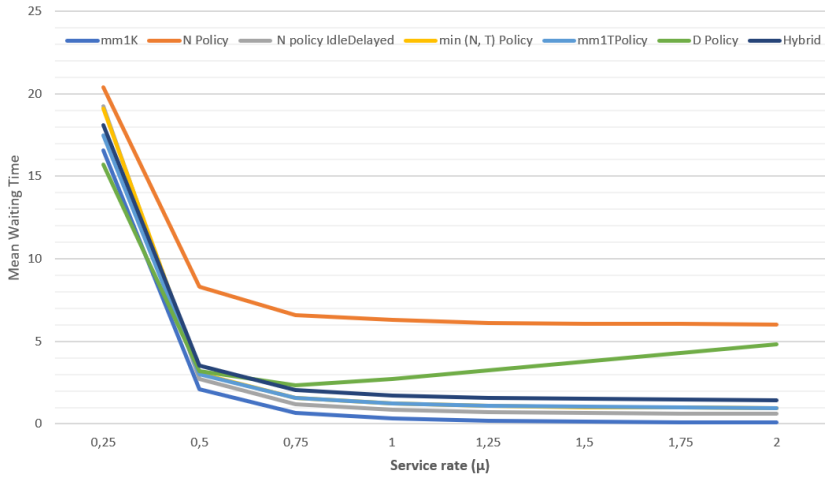


Figure 34: *Performance Results from Different Policy Simulation Models Measuring Mean Waiting Time vs. service rate μ.*

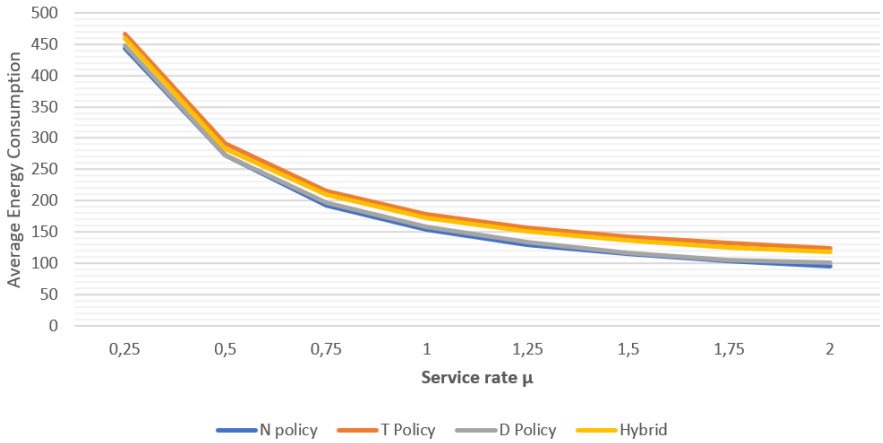


Figure 35: *Performance Results from Different Policy Simulation Models Measuring Average Energy Consumption vs. service rate μ.*

As the service rate (μ) increases, the mean waiting time and average energy consumption decrease for all systems. This is expected since a higher service rate allows customers to be served more quickly, reducing their waiting time and overall energy consumption.

6.Overall Discussion

This analysis reveals the interplay of various factors in the performance of queuing systems. The arrival rate (lambda) emerges as a critical determinant affecting system metrics such as waiting times, response times, throughput, and the number of packets served. These metrics, in turn, are influenced by the buffer size limitation (K), which plays a crucial role in maintaining system stability and preventing congestion.

The impact of the arrival rate becomes evident as it increases. Initially, more packets are served due to the higher influx, but the number eventually declines due to packet rejection or blocking when the buffer reaches its capacity. This highlights the trade-offs involved in system performance metrics, such as the potential for lower waiting times leading to higher average waiting times overall.

One notable policy, the N Policy, introduces a delay in starting service until the queue reaches a predefined threshold. While this policy leads to higher waiting times and response times, it can effectively reduce energy consumption. However, incorporating the idle delayed state into the system helps alleviate waiting times and enhances response times.

The min (N, T) Policy presents a favorable choice by considering both capacity limitations and time constraints, aiming to strike a balance between immediate access and queueing while accounting for system capacity. Additionally, the Hybrid Policy showcases competitive performance compared to other policies and can be considered for overall system optimization.

To sum up, determining the best policy for a queuing system necessitates evaluating specific requirements and priorities. By carefully considering different factors, queuing systems can be tailored to achieve efficient and effective performance.

Conclusion

In conclusion, this chapter presented a comprehensive experimental study to examine the behavior and performance of the system. In this chapter, we used controlled experiments and our simulation model to analyze the system's performance. The findings demonstrated the impact of various parameters, such as arrival rate and buffer size, on system performance metrics. The study highlighted the trade-offs between different performance metrics and the influence of service policies. Eventually, the experimental study provided valuable insights into the system's functionality and performance, laying the foundation for further improvements and optimizations.

Conclusions and future directions

In conclusion, this project has provided a comprehensive study of wireless sensor networks (WSNs), focusing on energy efficiency and the use of vacation policies in queueing systems. Throughout the report, we have explored the background of WSNs. We have also delved into the concept of queues with vacation, which offers a more realistic representation of real-world queueing systems. By implementing discrete-event simulation models, we were able to evaluate the performance of such systems under various conditions. This allowed us to analyze and compare different policies, such as threshold-based policies, and assess their impact on system behavior and energy consumption.

Through our simulation studies, we aimed to achieve several objectives. First and foremost, energy efficiency was a key focus. By considering limited sensor node resources, we sought to extend its battery life. We also aimed to improve the sensor node performance while reducing energy consumption, striking a balance between energy conservation and waiting delay.

The numerical results presented a deeper understanding of the system's behavior. We analyzed the system parameter variations. By measuring performance metrics, we gained insights into the system's performance and efficiency under different conditions. We were able to fine-tune the system design and gain valuable insights and guidelines for designing energy-efficient sensor networks.

The findings from our research have significant implications for the whole wireless sensor network not only a single node. They provide valuable knowledge and guidelines for designing energy-efficient systems, optimizing parameters, and striking the right balance between energy conservation and performance. The exploration of queues with vacation has provided a deeper understanding of system dynamics and behavior. By incorporating one or two vacation policies, we can achieve improved results, and the Hybrid policy stands out as the best example of this.

Moving forward, several future directions can build upon this research. One important aspect to consider is the implementation of quality of service (QoS) mechanisms in wireless sensor networks. This would enhance the overall efficiency and reliability of the network, particularly in scenarios where real-time or mission-critical data is being transmitted.

As technology advances, new communication protocols and networking architectures may emerge in the field of wireless sensor networks. Future research could investigate the integration of emerging technologies. These advancements have the potential to further enhance the performance, scalability, and adaptability of WSNs while optimizing energy consumption and system behavior.

Finally, this research has laid a solid foundation for understanding energy-efficient WSNs and the use of vacation policies in queueing systems. By exploring future directions such as implementing quality of-service mechanisms and considering different packet types, researchers and practitioners can continue to advance the field and contribute to the development of highly efficient and reliable wireless sensor networks.

References

[1] Z. Rezaei, "Energy Saving in Wireless Sensor Networks," *International Journal of Computer Science & Engineering Survey,* vol. 3, pp. 23-37, 2012.

[2] T. &. Rault, A. &. Bouabdallah and Y. Challal, "Energy Efficiency in Wireless Sensor Networks a top-down survey," *Computer Networks,* no. 67, pp. 104-122, 2014.

[3] "Buratti, C., Dardari, D., Verdone, R., and Conti, A," *An Overview on Wireless Sensor Networks Technology and Evolution,* vol. 9, pp. 6869-6896, 2009.

[4] M. &. Mcgrath and C. Ni Scanaill, Sensor Network Topologies and Design Considerations., 2013.

[5] M. C. M. D. F. A. P. Giuseppe Anastasi, "Energy conservation in wireless sensor networks A survey," *Ad Hoc Networks,* vol. 7, no. 3, pp. 537-568, 2009.

[6] K. E. O. I. T. S. K. R. G. O. A. S. &. Ukhurebor and A. Bobadoye, "Wireless Sensor Networks Applications and Challenges," in *IntechOpen eBooks*, 2020.

[7] V. J. R. ,. S. S. S. Aiswariya, "Challenges, Technologies and Components of Wireless Sensor Networks," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCICCT,* vol. 6, no. 3, 2018.

[8] I. &. Akyildiz, S. &. WY, Y. &. Sankarasubramaniam and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks,* no. 38, pp. 393-422, 2002.

[9] T. Agarwal, "Wireless Sensor Network Architecture Types, Working & Its Applications," *ElProCus - Electronic Projects for Engineering Students,* 2021, January 25.

[10] M. &. Matin and M. Islam, "Overview of Wireless Sensor Network," *2012.*

[11] M. J. Abdirahman, "Communication Protocols for Wireless Sensor Networks," 2022.

[12] S.-H. Yang, "Hardware Design for WSNs," *Signals and Communication Technology,* p. 49–72, Oct. 24, 2013.

[13] L. T. Y. L. S. L. W. a. T. H. C. Zhu, "Sleep scheduling towards geographic routing in duty-cycled sensor networks with a mobile sink," pp. 158-160, 2011.

[14] L. d. Smet, "Queue Mining Combining Process Mining and Queueing Analysis to Understand Bottlenecks, to Predict Delays, and to Suggest Process Improvements," *Eindhoven University of Technology,* 28 September 2014.

[15] L. Green, *QUEUEING THEORY AND MODELING,* New York: Graduate School of Business, Columbia University.

[16] Janaina Cristina Ferreira, "Queueing System Analysis A case study," *Escola Superior de Tecnologia e Gestão of Instituto Politécnico de Bragança,* may 2020.

[17] T. &. Naishuo and Z. G. Zhang, "Vacation QueueingModelsTheory and Applications," 2006.

[18] R. Cooper, Queueing Notation, 2011.

[19] D. Bouallouche, Congestion Control in the Context of Machine Type Communication in Long Term Evolution Networks: a Dynamic Load Balancing Approach, 2012.

[20] R. KHALAF, "On some queueing systems with server vacations, extended vacations, breakdowns, delayed repairs and stand-bys," *Thèse de doctorat. Brunel University,*

School of Information Systems, Computing, and Mathematics, 2012.

[21] S. &. Yuvarani and M. C. Saravanarajan, "Analysis of preemptive priority retrial queue with two types of customers, balking, optional re-service, single vacation, and service interruption," *IOP Conference Series.Materials Science and Engineering,* no. 4, p. 263.

[22] S. J. F. Khodadadi, "A fuzzy-based threshold policy for a single server retrial queue with vacations," *Cent Eur J Oper Res,* no. 20, p. 281–297, 2012.

[23] J. &. Chen, B. &. Sikdar and M. Hamdi, "An Adaptive N-Policy Queueing System Design for Energy Efficient and Delay Sensitive Sensor Network," *2018.*

[24] V. &. M. G. Goswami, "Prolonging Lifetime of Wireless Sensor Networks Using Modified N Policy Queueing Model," 2022.

[25] M. A. a. M. Mohamed, "Performance evaluation of an energy-saving mechanism in a wireless sensor network Presented," 2021.

[26] B. &. Boutoumi and N. Gharbi, "N-policy Priority Queueing Model for Energy and Delay Minimization in Wireless Sensor Networks Using Markov Chains," pp. 1-6, 2023.

[27] R.jain, The Art of Computer Systems Performance Analysis Technique for Experimental, New York : NY Wiley Computer Publishing, John Wiley & Sons, Inc, April 1991.

[28] "Simulation Modeling Steps," acqnote, 21 July 2021. [Online]. Available: https://acqnotes.com/acqnote/tasks/simulation-modeling-steps.

[29] S. A. G. A. e. a. Allen M, "What is discrete event simulation, and why use it?," in *Right cot, right place, right time: improving the design and organisation of neonatal care networks – a computer simulation study*, Southampton (UK), NIHR Journals Library, May 2015.

[30] J. M. Garrido, "chapter 3," in *Practical Process Simulation Using Object-oriented Techniques and C++*, the University of Virginia, Artech House computer science library, 1999, p. 219 .

[31] I. i. d. t. d. Kanpur, *Data Collection in Steady State Conditions CH1.*

[32] B. &. D. V. S. &. B. H. &. W. S. Feyaerts, "The impact of the NT policy on the behavior of a discrete-time queue with general service times.," *Journal of Industrial and Management Optimization. ,* no. 10, 2014.

[33] B. B. a. N. Gharbi, "An energy saving and latency delay efficiency scheme for wireless sensor networks based on GSPNs," in *4th International Conference on Control, Decision and Information Technologies (CoDIT)*, Barcelona, Spain, 2017.

[34] J.-C. Ke, "Modified T vacation policy for an M/G/1 queueing system with an unreliable server and startup," *Mathematical and Computer Modelling,* vol. 41, no. 11–12, pp. 1267-1277, 2005.

[35] S. G. a. S. Unnikrishnan, "Min (N, T) Policy M/G/1 Queue based Reduction of Power Consumption in Wireless Sensor Networks," *International Conference on Advances in Computing, Communication and Control (ICAC3),* pp. 1-6, 2019.

[36] J. Artalejo, " On the M/G/1 queue with D-policy," *Applied Mathematical Modelling,* vol. 25, no. 12, pp. 1055-1069, 2001.